Automating the Annotation of Data through Machine Learning and Semantic Technologies

Lorcán Anthony Karel Pigott-Dix

A thesis presented for the degree of Doctor of Philosophy

University of East Anglia
School of Biological Sciences
&
Earlham Institute

September 2023

Abstract

The ever-increasing scale and complexity of scientific research is surpassing our means to assimilate newly produced knowledge. Computer tools are necessary for the organisation, retrieval, and interpretation of new scientific knowledge and data. The efficacy of such tools requires that research outputs are described by rich machine-readable metadata. Ontologies provide the framework to unambiguously describe the meaning of knowledge and data, so that it may be re-used or combined to synthesise new knowledge. However, manually annotating research with ontology terms, a process called semantic annotation, is also infeasible due to the aforementioned scale.

This thesis describes research to develop deep learning-based tools for semantic annotation. The approaches described explore different methods for exploiting the domain knowledge encoded into ontologies to avoid the need to manually curate training corpora. They also take advantage of the inherent integrative capabilities of ontologies, to leverage combinations of heterogeneous knowledge to improve annotation performance and model interpretability. Several models exceeded previous benchmarks for semantic annotation in the bio-medical domain. This thesis concludes with a discussion of the strengths and limitations of the methods, and the implications for multi-domain ontology semantic annotation and for explainable artificial intelligence.

Contents

Dedication

This work is dedicated to my mum, Catherine.

Acknowledgements

Foremost, I would like to express my profound gratitude to the Biotechnology and Biological Sciences Research Council and the UKRI-BBSRC Norwich Research Park Biosciences Doctoral Training Partnership. Their financial backing has been instrumental to my research.

For their advice, support, and encouragement, my appreciation goes to my supervisors: Robert Davey, Felix Shaw, Tony Bagnall, and Wilfried Haerty. Special mention must also be made of Lowell O'Mard from Research Computing. His expertise and assistance have been crucial.

Furthermore, I would like to extend my sincere thanks to lab members past and present: Jazz Urog, Aaliyah Providence, Martin Ayling, Daniel Olvera, Nicola Soranzo, Simon Tyrell, Anil Thanki, Alice Minotto, Evanthia Samota, Catherine Knox, Emily Delva, Xingdong Bian, and Toni Etuk. I would also like to thank the other students at the Earlham Institute. In particular, Becky Shaw, Dasha Golubova, Sofia Kudasheva, Jess Peers, and Kamil Hepak.

Lastly, I would like to thank my mum for listening to me whinge down the phone throughout the pandemic, and I would like to thank Mia for listening to me whinge in person, and neither of them getting sick of me and my whinging.

List of Acronyms and Abbreviations

ADAGRAD	ADAptive GRADient algorithm
ADALINE	ADAptive LInear NEuron
ADAM	Adaptive Moment Estimation
AGI	Artifical General Intelligence
AI	Artificial Intelligence
BERT	Bidirectional Encoder Representations from Transformers
BGD	Batch Gradient Descent
Bi-LSTM	Bidirectional Long Short-Term Memory
Bi-RNNs	Bidirectional Recurrent Neural Network
BPE	Byte Pair Encoding
BPTT	Back Propagation Through Time
ChEBI	Chemical Entities of Biological Interest
CL	Cell Ontology
CNN	Convolutional Neural Net
CRF	Conditional Random Field
cTAKES	Clinical Text Analysis and Knowledge Extraction System
CTC	Connectionist Temporal Classification
DAML	DARPA Agent Markup Language
DDL	Dimensionality-Driven Learning
DisGeNET	Disease-Gene Network

DO	Disease Ontology
ELMo	Embeddings from Language Models
GCN	Graph Convolutional Network
GELU	Gaussian Error Linear Unit
Gene-Disease Association	
GloVe	Global Vectors for Word Representation
GNN	Graph Neural Net
GO	Gene Ontology
GO-BP	Gene Ontology Biological Process
GPT	Generative Pre-trained Transformer
GPU	Graphics Processing Unit
GRE	Graduate Record Examination
GTN	Graph Transformer Network
HGT	Heterogeneous Graph Transformer
HMM	Hidden Markov Model
HPO	Human Phenotype Ontology
ID3	Iterative Dichotomiser 3
LER	Label Error Rate
LID	Latent Intrinsic Dimensionality
LReLU	Leaky Rectified Linear Unit
LRP	Layerwise Relevance Propagation
LSA	Latent Semantic Analysis
LSTM	Long Short-Term Memory
MeSH	Medical Subject Headings
MHSA	Multi-Headed Self-Attention
MPO	Mammal Phenotype Ontology
MPT	Meta-Path Transformer

MSMSA	Multi-Scale Multi-headed Self-Attention
MSSA	Multi-Scale Self-Attention
NCBI	National Center for Biotechnology Information
NCBO	National Center for Biomedical Ontology
NCE	Noise Contrastive Estimation
NCR	Neural Concept Recogniser
NER	Named Entity Recognition
NLM	National Library of Medicine
NLP	Natural Language Processing
OBO	Open Biological and Biomedical Ontologies
OHPI	Ontology of Host-Pathogen Interactions
OIL	Ontology Infrastructure Language
PATO	Phenotype Attribute and Trait Ontology
PCA	Principal Component Analysis
PReLU	Parametric Rectified Linear Unit
RAM	Recurrent Attention Model
RDF	Resource Description Framework
ResNet	Residual Networks
RLHF	Reinforcement Learning from Human Feedback
RMSProp	Root Mean Square Propagation
RNN	Recurrent Neural Network
SAE	Squeeze-and-Excite
SASA	Scale-Aware Self-Attention
SAT	Scholastic Assessment Test
SGD	Stochastic Gradient Descent
SHAP	SHapley Additive exPlanations
SNP	Single Nucleotide Polymorphism

SPARQL	SPARQL Protocol and RDF Query Language
SRNN	Simple Recurrent Neural Network
TDNN	Time Delay Neural Network
UMLS	Unified Medical Language System
URI	Unique Resource Identifier
VAE	Variational Autoencoder
XML	Extensible Markup Language

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem statement

1.1.1 The scale and complexity of modern science

The amount of scientific literature published has been increasing exponentially - with volumes doubling every 15 years [65]. This surge in productivity has been accompanied by an ever-increasing deluge of data that is progressively both heterogeneous and complex. For example, genomic data includes genetic sequence data [46], protein structure and function information [47], and phenotype descriptions [187]. Manually sifting through this heterogeneous data is an insurmountable task. Computer-based tools that can rapidly find and interpret data promise a remedy. However, their usefulness depends on the richness of the metadata describing these data [3].

Metadata is most broadly described as "data about data" [184], it provides context to a data point, and can be used to combine relevant data to derive new insights that would be impossible from each data-source in isolation. For example, Jiang *et al.* [101] combined heterogeneous biomedical data from different domains using their metadata to train a machine learning model for predicting molecular function. Zitnik *et al.* [256] integrated protein-protein and drug-protein interaction data to train a model that predicts adverse outcomes from drug combinations. The integration required domain-specific metadata that ensures that the data sources were joined using compatible attributes. If the data sources integrated in those works were not richly described they may not have been combined at all, or may have been misinterpreted - depriving researchers of the opportunity to derive new insights or train more powerful models.

1.1.2 Ontologies and annotation

In the last two decades, domain experts in the life sciences have codified their expertise into ontologies [9, 45, 58, 48, 105, 211, 205]. These ontologies are a type

of semantic technology, and they describe a particular domain of knowledge using a formal, unambiguous, machine-readable "structured vocabulary" [61]. They describe concepts, their properties, and their relationships with other concepts in the ontology. These ontologies are intended to be used as metadata to describe research artefacts. Ontology-based metadata clarifies what the data represents, within a common framework, reducing the friction involved with the reuse and integration of data. For example, the Human Phenotype Ontology (HPO) describes hereditary diseases and phenotypic abnormalities, organising them via super-class and subclass relations [187]. These relations can be used to disambiguate terms - say two abnormalities can be shown to be conceptually related. Ontologies provide a rich vocabulary to annotate research artefacts and knowledge. These annotations describe and contextualise research outputs for future end-users.

Annotating artefacts with ontology terms is a process known as semantic annotation [104]. The ontology term annotations make implicit context explicit [231, 156]. Many proteins share names with the genes that encode them. An annotation of a name mention can specify whether it is referring to the gene rather than the protein or vice versa, as complementary genes and proteins often share the same name [104]. This explicit contextualisation allows computer-agents to behave more like expert human-agents and "interpret" the semantics of a dataset. To this end, research funding bodies have begun to require data stewardship plans in grant applications [26]. This has coincided with efforts to introduce data management best practices such as the FAIR principles [231]. Introduced by Wilkinson *et al.* [231] in 2016, the FAIR principles provide a set of guidelines that knowledge producers should aim to adhere to when publishing research artefacts to make them findable, accessible, interoperable, and reusable. These guidelines are heavily centred around describing artefacts with accurate domain-relevant metadata.

Unfortunately, efforts to annotate datasets have not been as fruitful as those producing the data itself. Mons *et al.* [155] identified a lack of incentive as the cause of the data annotation "bottleneck". Measures, such as the guidelines like those described by Wilkinson *et al.* [231], do not address this "bottleneck". Not only is a high level of domain expertise necessary to annotate datasets to the required standard, but the scale of the data annotation task far surpasses the capacity of domain experts. Especially so when considering that ontologies are works-in-progress and are continually updated and amended. Even if the task was better incentivised, computer tools must be developed to automate, or at least expedite, the semantic annotation process.

1.2 Thesis overview and contributions

This section provides a brief overview of the research undertaken in this thesis, and outlines its contributions. The aim of the work described in this thesis was to develop

deep learning models that could identify ontology terms in literature and tabular data. Specifically, models that could be easily re-trained to incorporate ontology updates, avoiding as much manual training data labelling as possible. In chapter 2, I describe the development of deep learning and semantic technologies, followed by an overview of previous semantic annotation tools. Chapter 3, details the exploration of using heuristic labelling to create a noisy labelled training corpus. This corpus was used to train deep learning models which were adapted to exploit the pattern of noise inherent to the heuristic labels. Chapter 4, describes the development of deep learning models that exploit the contents of multiple ontologies to train a classifier without the need for the additional training corpora. This work also explored how different ontology combinations and architectural modifications influenced the efficacy of a model. Chapter 5, explores how heterogeneous data sources can be integrated to improve semantic annotator performance. This chapter also describes the development of a semantic annotation architecture that can learn to selectively exploit heterogeneous data, with implications for explainable artificial intelligence. Finally, chapter 6 provides a discussion of all of the research together, covering its limitations and advantages. Future work is proposed, with consideration for the rapid changes that have occurred in the realms of artificial intelligence and text processing over the course of the last four years.

This work produced novel, relatively low-cost models for ontology-based semantic annotation which exceeded previous benchmarks. Insights were derived into how the combination of various domain ontologies can impact model performance, with implications for multi-ontology semantic annotation. The final empirical chapter demonstrated how ontology-annotated integrated data can be leveraged to build *a priori* interpretability into a deep neural architecture - which greatly enhances model explainability.

Chapter 2

Background

2.1 Probabilistic Artificial Intelligence and Deep Learning

2.1.1 Teaching machines to learn

In their 1975 Turing Award Lecture, Newell and Herbert [160] described symbols as being to intelligence as cells are to biology. They argued that qualitative structure is ubiquitous in science, citing examples from fields as diverse as plate tectonics in geology, to the germ theory of disease in epidemiology. They termed their position "the Physical Symbol System Hypothesis", which posits that symbolic structure is both the "necessary and sufficient" basis of intelligent behaviour. Further to this, they characterised intelligence as a search over the space of symbolic representation.

An agent's ability to learn is considered central to its intelligence [189]. Like human learning, machine learning is restricted by assumptions, both explicit and implicit, known as inductive biases [195]. These inductive biases influence, and are influenced by, how knowledge is represented and can result in some generalisations being more forthcoming in some contexts and less so in others [195].

It has been argued that Human learning conforms to the symbolic artificial intelligence (AI) paradigm [115], and that symbolic structures are conducive to representing and reasoning over knowledge [159]. However, others have appealed to the biological plausibility of probabilistic methods inspired by the networks of neurons that make up the nervous system [188, 66, 115, 196]. This divide, between these two schools of thought, is no more apparent than in the field of machine learning [204]. As a result, symbolic methods were more common in early machine learning research, before greater computational power allowed for the development of more effective statistical methods.

Symbolic learning Quinlan [178] noted that research into machine learning had either explored adaptive systems that respond to changes in performance by adjust-

ing their parameters, or had characterised learning as a process of building structured symbolic knowledge. Systems that build structured symbolic knowledge have the advantage of being much more comprehensible for humans [147]. In the 1980s symbolic knowledge-based systems called "expert-systems" had been devised and had commercial success [147, 178]. These systems operated using structured knowledge representations, however Quinlan [178] notes that such representations are time-consuming and expensive to produce, and refers to the discrepancy between their production and their demand as a "knowledge acquisition bottleneck".

To address this, Quinlan introduced the Iterative Dichotomiser 3 (ID3) algorithm for learning decision trees from data [178]. The algorithm builds features from-the-top-down, splitting samples using the attribute that provided the largest "Information Gain Criterion". This measure, taken from information theory, indicates the level of mixture between the classes of the samples divided into two groups by a given feature. The algorithm recursively selects features that provide the least class mixture to split the samples of each branch, until all terminal branches contain a single class or all attributes have been exhausted. This approach creates a classification model that makes predictions in a structured and interpretable way. However, the authors note that the more complicated a model, the more "unintelligible to human experts" it is [178].

Another model, RIPPERk, developed by Cohen [43], leverages both categorical and continuous scalar features for rule induction. Training data is split 2:1 into "growing" and "pruning" sets respectively. First, the model progressively adds new rules, Boolean (true or false) values based upon categorical features or on thresholds in the scalar features. Like in Quinlan's ID3 algorithm, a new rule is added depending upon which provides the greatest information gain criterion, and new rules are added until every sample in the growing set has been correctly classified, or that the rule set exceeds a predetermined size which is related to the smallest possible rule set (the minimum description length). Additionally, the model is prevented from learning any rule where a relevant attribute is missing from a subset that is being split, such that it only creates rules where the specified attribute is present for each sample in the subset in question.

Following the growing phase, the algorithm explores if removing any rule will improve classification accuracy on the pruning set. Specifically it deletes any rule from the end of a branch of rules that maximises a function of the proportion of accurately classified examples in the set. This is repeated, starting from the most recently added rule working backwards, until pruning does not improve classification accuracy on the pruning set. This pruning process improves the model's ability to generalise to unseen samples.

After the initial growing and pruning phases, the algorithm then optimises the rule set further. Iterating through the rules, in the order that they were learned, two new sets of rules are grown and then pruned from each rule, with the objective of

minimising error over the entire rule set. The rule set with the shortest description length is chosen from the original and two additional rule sets. This can be repeated multiple times over the rule set. The authors reported that this approach has better generalisability and is much more efficient at training than previous rule induction models [43].

Biologically inspired learning Arguing that the probabilistic view of cognition seemed more plausible than the symbolic, Rosenblatt [188] introduced a simple probabilistic model of a neuron, the Perceptron. The Perceptron models neurons as binary linear classifiers, where the importance of input stimuli are adaptively weighted through negative/positive feedback stimuli during an iterative learning process. Rosenblatt [188] suggested that multiple Perceptrons could be combined together to solve complex tasks. Further to this, he posits that representations could be distributed across the entire set of neurons, such that the system is afforded a level of redundancy. The neurons would not depend on specific neuronal pathways, so that when neurons are lost the entire systems performance is degraded, rather than that of a specific task.

Similarly, in their 1960 paper (declassified in the late 1970s), Widrow and Hoff [230] outlined an early approach for machine learning called ADAptive LInear NEuron (ADALINE). The algorithm iteratively adjusts the model parameter weight vector to minimise the mean squared error between the model predictions and the ground truth. It adjusts the weights based upon the correlation between each element of the input and the error signal, which although not explicitly described in this way in the paper, is effectively a form of gradient descent. The correlation between the error and the inputs is calculated by computing the partial derivatives of each weight with respect to the mean squared error. These derivatives, or gradients, are used to update the vector weights. However, like the Perceptron, as ADALINE was linear and only had a single layer, it could not map complex non-linear functions.

Inspired by the self-organised development of biological neurons, in particular retinal neurons, Fukushima developed the Neocognitron [66] (expanded in [67]) which was devised to classify handwritten numerals. The Neocognitron is a multi-layer, hierarchical image classifier, that can learn to extract features from input images, combining their signal in higher layers, ending with a classification layer. The architecture is comprised of three types of "cells": Simple cells (S-cells), complex cells (C-cells), and variable inhibitory cells (V-cells). S-cells extract the features, C-cells aggregate the feature signals of their connected S-cells while also providing a degree of spatial invariance, while V-cells, added in the 1988 paper [67], ensure feature extraction specificity. The architecture is explicitly designed to inhibit redundancy in the network and encourage specificity in the features, which stands in contrast with the generalisability Rosenblatt [188] intended for the Perceptron.

Fukushima described two training methods for the Neoconitron, the first being described as "without a teacher" [66], and the later expansion included a second "with a teacher" method [67]. Both use a process called "reinforcement of maximum-output cells". Here the S-cells perform a local weighted sum of patches of the input. The connections between the S- and C-cells that produce the largest output are boosted while the other connections are left unchanged. The author posits that, in a process similar to Hebb's rule of neuronal organisation [86], as a connection becomes stronger, the feature tends to become more specific [67]. According to Hebb's rule, often described by the maxim "cells that fire together, wire together", frequent simultaneous activation of connected neurons tends to increase the degree of their connections [86]. Although in the case of the Neocognitron, this specificity comes at the expense of other feature signals [67], which may not be the case in biological neurons.

In both training methods, the V-cells are fed an aggregate of the mean feature signals from all the S-cells connected to a particular C-cell. If the mean features reach a relative value compared with the maximum feature signal, the C-cell is inhibited from increasing any connections to the S-cells, thereby promoting the extraction of specific features. The "with a teacher" method introduces manual control of the V-cells so that the feature specificity can be adjusted [67].

S-cells and C-cells are conceptually similar to what would become known as convolutional neural nets [196] - in particular the degree of positional invariance, the local feature extraction, and global aggregation. Although the network was deep, the learning algorithm used to train the model is not backpropagation.

In 1986 Rumelhart, Hinton, and Williams [189] published their seminal paper "Learning representations by back-propagating errors". This paper, like Widrow and Hoff, and Rosenblatt before, describes a network of neurons with a learning procedure for iteratively updating parameter weights of a model to minimise the difference between the vector of values representing the ground truth and the output of the model. This iterative learning process consists of two passes. The first, or forward pass, is the calculation of the output values of the model. The difference between the model output and the ground truth is computed. For the second, backward pass, the partial derivative of the error with respect to each weight is calculated. Each partial derivative is a gradient that represents the sensitivity of the error between the output and ground truth to a change in the parameter weight. The chain-rule is then used to recursively calculate the gradient for the weights in each layer all the way back to the first. While the authors acknowledge that this method does not conform to "a biologically plausible model for learning in the brain" [189] (this implausibility has been disputed [115]), it does have some interesting properties. Chiefly, that it allows a model to represent novel features, that represent task-specific regularities, within its hidden layers [189]. The type of architecture used in this paper came to be called a feed-forward network. Feed-

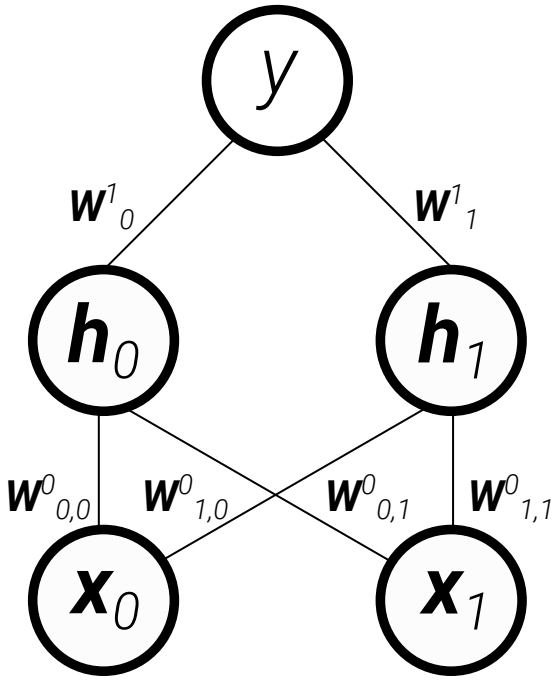

Figure 2.1: Overview of the simple neural net.

forward networks are architectures that model fixed-sized inputs to outputs through an acyclical series of parameter weights [122]. The single-layer Perceptron [188] is a simple feed-forward network, but typically modern feed-forward networks have multiple layers of parameter weights [122].

Shavlik *et al.* [204] compared both the Perceptron [188] and ID3 decision tree induction [178] against backpropagation [189], across five varied classification tasks. They found that the backpropagation model tended to outperform both ID3 decision tree induction and the Perceptron. Backpropagation was more robust to noise, and its performance was less effected by reduced training data [204].

The modern deep learning paradigm has rested upon backpropagation [196, 122], which in fact had been discovered independently multiple times throughout the 1970s and 1980s [122]. This methodology is central to this thesis so I will now describe the process in more detail using an example.

Consider a simple toy neural network, consisting of one hidden layer, with inputs of two dimensions. For simplicity this is a completely linear neural net without any activation functions. This can be described as follows:

$$\mathbf{h} = \mathbf{W}^0 \cdot \mathbf{x} \tag{2.1}$$

$$y = \mathbf{W}^1 \cdot \mathbf{h} \tag{2.2}$$

Where $\mathbf{x} \in \mathbb{R}^2$ is the input vector, $\mathbf{W}^0 \in \mathbb{R}^{2\times2}$ and $\mathbf{W}^1 \in \mathbb{R}^{1\times2}$ are parameter weights, $\mathbf{h} \in \mathbb{R}^2$ is the vector of hidden representations, and $y \in \mathbb{R}$ is the output . Please see figure 2.1 for a visualisation of the network architecture. To reflect the

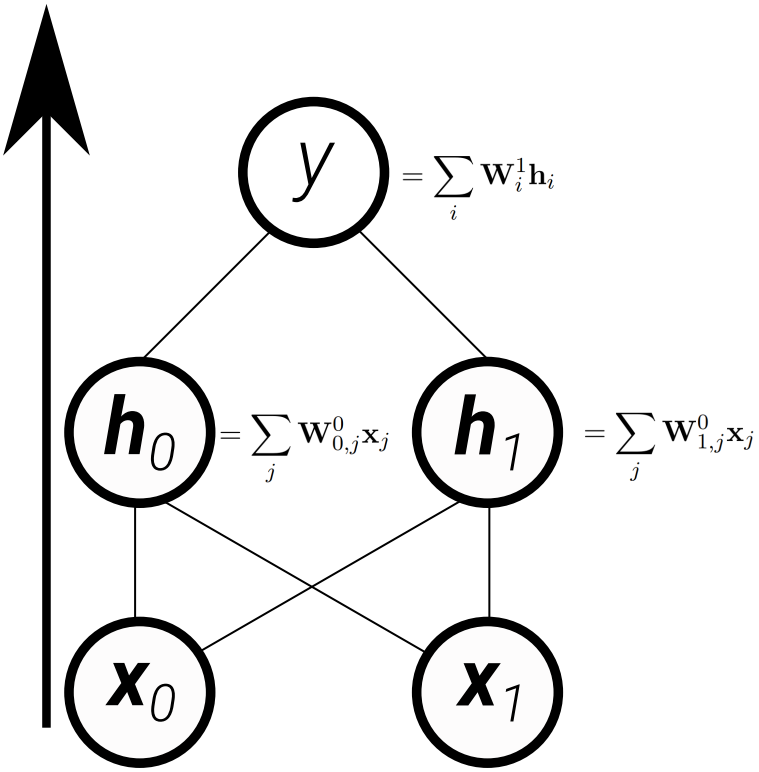

Figure 2.2: An illustration of the forward pass to compute the model outputs. For $\mathbf{W}^1$, i represents the element index. For $\mathbf{W}^0$, j represents the column index.

network architecture of figure 2.1 we can reformulate equations 2.1 and 2.2 like so:

$$\begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \end{bmatrix} = \begin{bmatrix} \mathbf{W}_{0,0}^0 & \mathbf{W}_{0,1}^0 \\ \mathbf{W}_{1,0}^0 & \mathbf{W}_{1,1}^0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \end{bmatrix} \tag{2.3}$$

$$y = \begin{bmatrix} \mathbf{W}_0^1 & \mathbf{W}_1^1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \end{bmatrix} \tag{2.4}$$

Here each element of the parameter matrices represents a specific connection between two nodes of the network, while each element of the input and hidden layer vectors represents a specific node. On the forward pass the model output y is computed during the forward pass, like so:

$$\mathbf{h}_0 = \sum_j \mathbf{W}_{0,j}^0 \cdot \mathbf{x}_j = \mathbf{W}_{0,0}^0 \cdot \mathbf{x}_0 + \mathbf{W}_{0,1}^0 \cdot \mathbf{x}_1 \tag{2.5}$$

$$\mathbf{h}_1 = \sum_j \mathbf{W}_{1,j}^0 \cdot \mathbf{x}_j = \mathbf{W}_{1,0}^0 \cdot \mathbf{x}_0 + \mathbf{W}_{1,1}^0 \cdot \mathbf{x}_1 \tag{2.6}$$

$$y = \sum_i \mathbf{W}_i^1 \cdot \mathbf{h}_i = \mathbf{W}_0^1 \cdot \mathbf{h}_0 + \mathbf{W}_1^1 \cdot \mathbf{h}_1 \tag{2.7}$$

This process is visualised on the network in figure 2.2. The error between the ground truth and the model output is then calculated. For this example the mean

squared error has been chosen as the error function. This is due to its simplicity to differentiate. It is defined like so:

$$E = \frac{1}{2}(y - y_{\text{true}})^2 \tag{2.8}$$

Once the model outputs are calculated, we need to measure the sensitivity of the error to changes in the output of the neural network. To do this we calculate the gradient of the error function with respect to all the parameter weights in the network. Firstly, the partial derivative of the error with respect to the model outputs is computed. This partial derivative, or gradient, tells us how changing the model output y influences the error. If the gradient is positive, then increasing y will lead to an increase in the error, decreasing y will decrease the error. If the gradient is negative, then increasing y will lead to a decrease in the error, while decreasing y will increase the error. The chain rule is then used to backpropagate the gradient, layer-by-layer, back through the network to the inputs.

First we compute the partial derivative of the error with respect to y, which is given by:

$$\frac{\partial E}{\partial y} = y - y_{\text{true}} \tag{2.9}$$

Then to measure how a change in the values of $\mathbf{W}^1$ influence y, we need to compute the partial derivative of the error with respect to each weight in $\mathbf{W}^1$. To do this we use the chain rule, which states that given a composite function, such as $y = f(x)$ and $z = g(y)$, then the derivative of x with respect to x can be expressed as the product of the derivatives of x with respect to y and of y with respect to z [72]. So we calculate the product of the partial derivatives of the error E with respect to y and of y with respect to the parameter weights $\mathbf{W}^1$, like so:

$$\frac{\partial E}{\partial \mathbf{W}_i^1} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial \mathbf{W}_i^1} = (y - y_{\text{true}})\mathbf{h}_i \tag{2.10}$$

Then to understand how a change to the parameter weights $\mathbf{W}^0$ will influence the error, we need to calculate the partial derivative of each weight in $\mathbf{W}^0$ with respect to the gradient of the weights in $\mathbf{W}^1$. Again using the chain rule:

$$\frac{\partial E}{\partial \mathbf{W}_{i,j}^0} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial \mathbf{h}i}\frac{\partial \mathbf{h}i}{\partial \mathbf{W}_{i,j}^0} = (y - y_{\text{true}})\mathbf{W}_i^1\mathbf{x}_j \tag{2.11}$$

Here i and j correspond to the row and column indices respectively. This is illustrated alongside the network in figure 2.3 To minimise the error function, which is the difference between the output y and the ground truth y_{truth}, the model then updates the value of each weights. The weights are iteratively adjusted during training, reducing the value if the weight's gradient is positive, increasing if negative. The extent to which the weights are updated by is typically controlled by a scalar learning

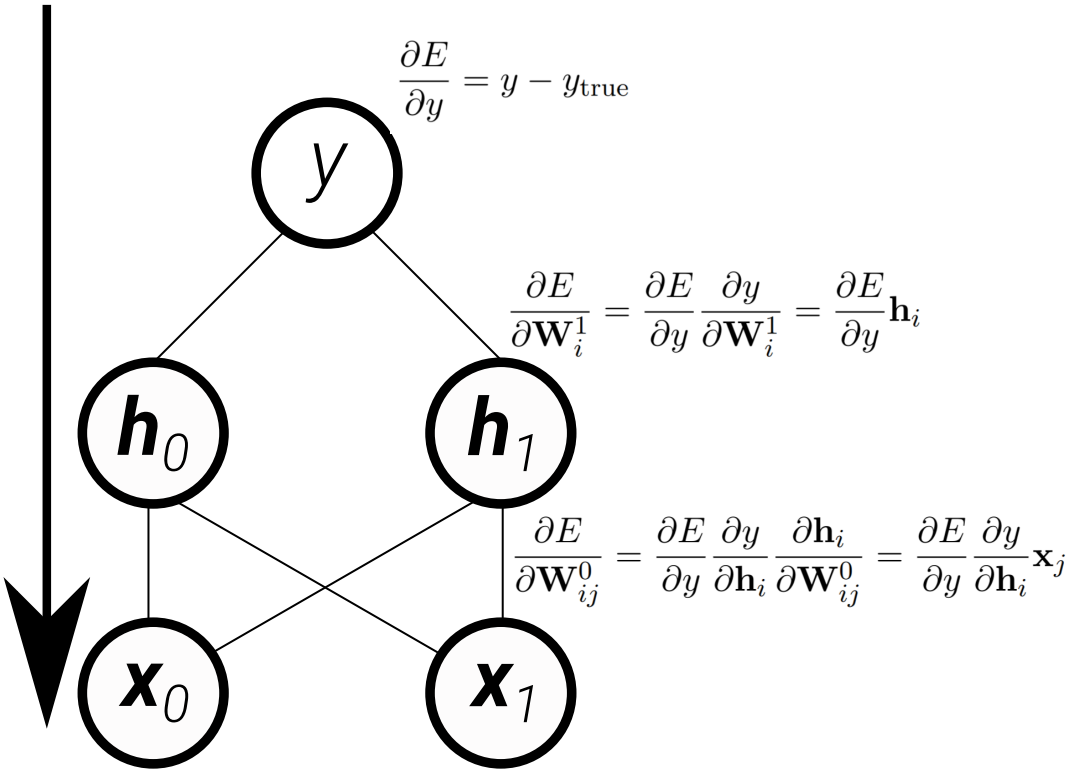

Figure 2.3: An illustration of the backward pass to compute the gradients for each weight. For $\mathbf{W}^1$, i represents the element index. For $\mathbf{W}^0$, i and j represent the row and column indices respectively.

rate parameter [245].

Optimisers The process to apply back propagation to train a neural network used by Rumelhart *et al.* [189] is called Batch Gradient Descent (BGD). Here the gradients are computed using the entire training dataset. There are a few limitations to this approach. If the error surface, or landscape, is highly curved and has many saddle points, or local minima, a model may not find a globally optimal solution [123]. Additionally, it may lead to problems when datasets begin to exceed the memory capacities of the machines training them, which became the case in the 2000s when deep learning methods were increasing in popularity [30].

Stochastic Gradient Descent (SGD), first introduced by Robbins and Monro [186], was shown to effectively limit memory capacity issues during training [123, 30]. This approach calculates the gradients and updates the parameter weights iteratively for every example in the training dataset, and is essentially the same as BGD but for single examples. This approach is noisy due to the updates being the result of single randomly selected examples, hence the name. However, as it makes more frequent updates per the number of seen training examples, it can converge more quickly than BGD, and is much more memory efficient. It has also been observed to allow the learning process to escape local minima to find deeper minima

[123].

A medium between these two approaches is called Mini-Batch Gradient Descent [123]. Here the training set is split into batches of randomly selected samples, and the algorithm updates its weights after seeing each batch. The stochasticity of the batch samples provides the learning process with some capacity to escape sub-optimal local minima, while the grouping of samples provides more stability.

Other methods have sought to optimise the learning process through use of momentum [176] or use previous gradient values to scale weight updates [60, 245]. Momentum was first introduced to the optimisation of iterative methods by Polyak in 1964 [174]. In the context of neural net learning optimisation, momentum adapts the weight updates so that previous updates have a continued but steadily diminishing influence on the latest updates. Rather than scaling the gradient by the learning weights, momentum updates incorporate the current gradients scaled by the learning rate and a fraction of the previous iteration's weight update [176]. The parameter weight updates with momentum can then be described as follows:

$$v_i = \beta v_{i-1} + \alpha \nabla \mathcal{L}(w_{i-1}) \tag{2.12}$$

$$w_i = w_{i-1} - v_i \tag{2.13}$$

Where v_i is the momentum for the current iteration i, scaled by the momentum coefficient β, α is the learning rate, and $\nabla \mathcal{L}(w_{i-1})$ represents the gradient of the loss function with respect to the parameter weights w for the previous iteration $i-1$. Using momentum has been shown to reduce the noisy learning exhibited by SGD and to lead to faster convergence than SGD.

The ADAptive GRADient algorithm (ADAGRAD) introduced by Duchi *et al.* [60], individually updates each parameter weight during training, by adjusting the learning rate for each particular weight based upon its historic gradient. To do this the learning rate is divided by the square root of the sum of all the previous gradients squared. This can be represented as follows:

$$G_t = \sqrt{\sum_{i=0}^{t-1} \left(\nabla \mathcal{L}(w_i)\right)^2} \tag{2.14}$$

$$w_{t+1} = w_t - \frac{\alpha}{G_t + \varepsilon} \nabla \mathcal{L}(w_t) \tag{2.15}$$

Where G_t is the square root of the cumulative sum of squared gradients up to iteration $t-1$ for parameter w, and ε is a small constant to prevent division by zero. $\nabla \mathcal{L}(w_t)$ is the gradient of weight w at iteration t. This G_t term adaptively scales the learning rate so that weights that received large gradients, which indicates stronger beneficial signals, have their updates scaled so they have smaller updates and change less. Meanwhile parameters with smaller gradients, and thus weaker signals, receive relatively larger updates, which may encourage them to move from

sub-optimal local minima. One limitation is that the G_t term also results in the learning rate becoming smaller over time, which may lead to a decay in the learning weights which prevents learning [245]. Another, is that the algorithm must store a memory of all previous gradients for each parameter, which may become prohibitive for large models.

Hinton *et al.* introduced the optimiser Root Mean Square Propagation (RMSProp) [90]. Unlike ADAGRAD before, it uses the exponential moving average of each weight's squared gradient, rather than the square root of the sum of the squared gradients of the parameter from all the previous elements to scale the learning rate. The moving average of the squared gradients for parameter weight w at iteration t can be formalised as follows:

$$E[\nabla\mathcal{L}(w_t)^2] = \rho E[\nabla\mathcal{L}(w_{t-1})^2] + (1-\rho)\nabla\mathcal{L}(w_t)^2 \tag{2.16}$$

Where $E[\nabla\mathcal{L}(w_{t-1})^2]$ is the exponential moving average of the previous squared gradients up to the previous iteration, and $\nabla\mathcal{L}(w_t)^2$ is the squared gradient for the parameter weight at the current iteration. The parameter ρ determines the importance the model gives to the previous gradients, as it determines the proportion of the current exponential moving average that comes from either the previous moving average or the current gradients. This is used to scale the weight updates like so:

$$w_{t-1} = w_t - \frac{\alpha}{\sqrt{E[\nabla\mathcal{L}(w_t)^2] + \varepsilon}}\nabla\mathcal{L}(w_t) \tag{2.17}$$

Where α is the learning rate, and ε is a small constant to prevent division by zero. This is intended to prevent continuously large updates for parameters with consistently large gradients, while scaling up the updates for parameters with consistently small gradients, without requiring to store all previous gradients for each individual weight in memory. The scaling mitigates the decay problem of ADAGRAD. Additionally, the decay parameter ρ, which is similar to the β parameter used in momentum, makes the memory act like a moving window of previous gradients and weight changes. At each successive iteration older values become less influential, and as each weight only stores two memory values for each parameter, the memory footprint is smaller than that of ADAGRAD.

ADADELTA (not an acronym) [245] was developed as an extension of ADAGRAD which, like RMSProp, mitigates the weight decay effect by restricting the scaling of the gradients to a fixed window of previous gradients. This time however, the scaling is a function of information on previous weight updates in addition to previous gradients, doing away with the requirement for a learning rate parameter entirely. This means that the model should learn the optimal learning rate for each parameter without requiring tuning. The moving average of the squared gradients is as described in equation 2.16.

In a similar fashion, ADADELTA also captures the exponential moving average of the previous squared weight updates for each weight, like so:

$$E[\Delta w_{t-1}^2] = \rho E[\Delta w_{t-2}^2] + (1-\rho)\Delta w_{t-1}^2 \tag{2.18}$$

Where Δw_{t-1}^2 is the squared change to the parameter weight at the previous iteration. Instead of scaling a learning rate, both of these exponential moving averages of the past squared gradients and weights are used to scale the update of the weights, as follows:

$$\Delta w_t = -\frac{\sqrt{E[\Delta w_{t-1}^2] + \varepsilon}}{\sqrt{E[\nabla \mathcal{L}(w_t)^2] + \varepsilon}} \nabla \mathcal{L}(w_t) \tag{2.19}$$

$$w_{t+1} = w_t + \Delta w_t \tag{2.20}$$

Where Δw_t is the change in the parameter weight, ε is small constant to prevent division by zero, and $\nabla \mathcal{L}(w_t)$ is the derivative, or gradient, of the loss with respect to the parameter weight w_t at iteration t. If the recent gradients were large but the weight changes have been small, the algorithm scales the weight update so that it is smaller. Otherwise if the weight updates have been large but the gradients small, the weight updates are relatively larger. This should help the model to find better minima, by allowing it to make large updates to the weights if necessary, but also preventing it oscillating around minima due to overshoot.

Kingma and Ba introduced Adaptive Moment Estimation (ADAM) [108], which incorporates the exponential moving average of the past gradients with the exponential moving average of past square gradients. The former, works like momentum capturing the direction of the gradients, while the latter captures the variability of the gradient which can indicate if the parameter is oscillating. The algorithm estimates the exponential moving average of the gradients and squared gradients as follows:

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot E[\nabla \mathcal{L}(w_t)] \tag{2.21}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{2.22}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot E[\nabla \mathcal{L}(w_t)^2] \tag{2.23}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{2.24}$$

The parameters β_1 and β_2 are the exponential decay rates for exponential moving average estimates of the gradients and squared gradients respectively. $\hat{m}_t$ and $\hat{v}_t$ are the bias corrected estimates. The bias is corrected to mitigate the effect of the initial value of the exponential moving averages being zero. Over the course of training the influence of the bias correction reduces and the influence of the initial zero values

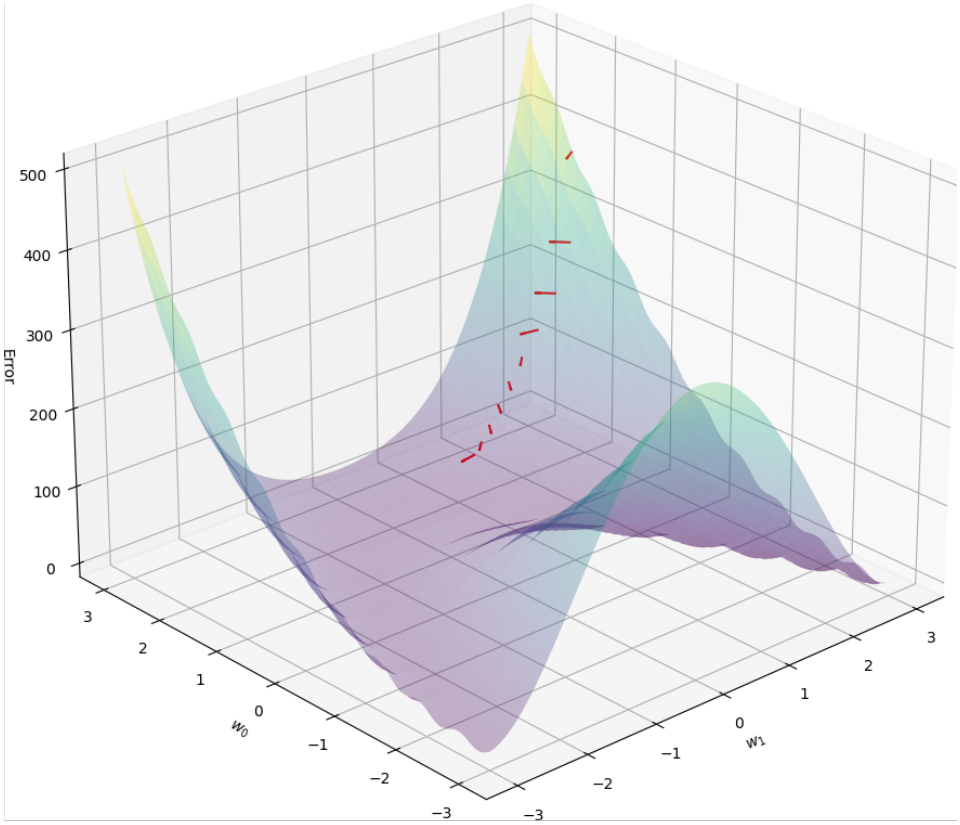

Figure 2.4: An illustration of gradient descent using the ADAM optimiser for a toy neural net which has a two-parameter hidden layer. The yellow-to-blue surface represents the error surface over the parameter values $\mathbf{w}_0$ and $\mathbf{w}_1$. The red arrows represent the direction of the parameter updates between each iteration.

diminishes. These bias corrected terms are then used to update parameter weight scaled by the learning rate α.

$$w_{t+1} = w_t - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} \tag{2.25}$$

Where ε is a small constant to prevent division by zero. The authors compared the effect of using either ADAGRAD, RMSProp, SGD, and ADAM on deep learning architectures for various tasks, finding that ADAM consistently lead to faster convergence across the tasks. The authors attribute the superior performance of ADAM to its abilities to scale-up the gradients for parameters with infrequent updates (similar to ADAGRAD) and to adapt to changes in the error landscape (akin to RMSProp). Please see figure 2.4 for a visualisation of gradient descent over an error landscape using ADAM with a toy neural net that has a two-parameter hidden layer.

2.1.2 Convolutional Neural Nets: Demonstrating the Utility of Deep Learning

In the decade following the Neocognitron, LeCun *et al.* [120] devised an architecture that was also designed for the classification of handwritten digits. Regarded as being one of the first practical, commercially viable, applications of deep learning [196]. This three layer model contained many of the attributes we would come to associate with the Convolutional Neural Net (CNN). The first two layers comprised of applying sets of shared weights across the plane of their inputs, spatially sub-sampling the input using a local receptive field. These proto-CNN layers were designed to be able to learn specific geometric patterns associated with characters, while also being invariant to their absolute position within an input image. These proto-CNNs produce what the authors called a feature map, which is a representation of the interaction between the shared invariant weights and the inputs. These two proto-CNN layers were followed by a fully connected layer and an output layer. Each layer had a scaled hyperbolic tangent function applied to their outputs, as it was believed to lead to faster model convergence. Once trained the model achieved a 95% classification accuracy.

In 1998, the paper "Gradient-Based Learning Applied to Document Recognition" by LeCun *et al* [121] heralded a step-change in the adoption of Deep Neural Nets over classical ML techniques. The authors argued that automatic feature learning can outperform engineered features in pattern recognition tasks, and when coupled with weight regularisation can be prevented from over-fitting to the training data. They proposed *LeNet-5*, a seven layer CNN-based architecture for image classification and document recognition. The intention was that the CNNs would extract useful local features automatically, with later layers combining these local features into global features for classification [121]. This architecture outperformed contemporaneous optical character recognition methods significantly.

It has been argued that parameter sharing and specific architectural design of CNNs and their pooling layers make their architecture less prone to gradient vanishing [21]. The convolutional filters are applied across all the possible positions of an input. When the gradient is calculated for a convolutional filter (or kernel) it is the sum of all gradients for each position of the input. Additionally, the max pooling layer ensures that the only the strongest signals from each feature map inform the calculation of the gradient, mitigating the risk of vanishing gradients. Figure 2.5 provides an illustration of the operation of a CNN, with sliding filters being passed over an image, with dimensionality reduction using a maxpooling function, leading to a classification layer.

AlexNet, introduced by Krizhevsky *et al.* [114], a CNN-based model that is regarded as the first large deep learning model. Comprised of eight layers: five convolutional layers, some with max pooling, followed by two fully connected layers

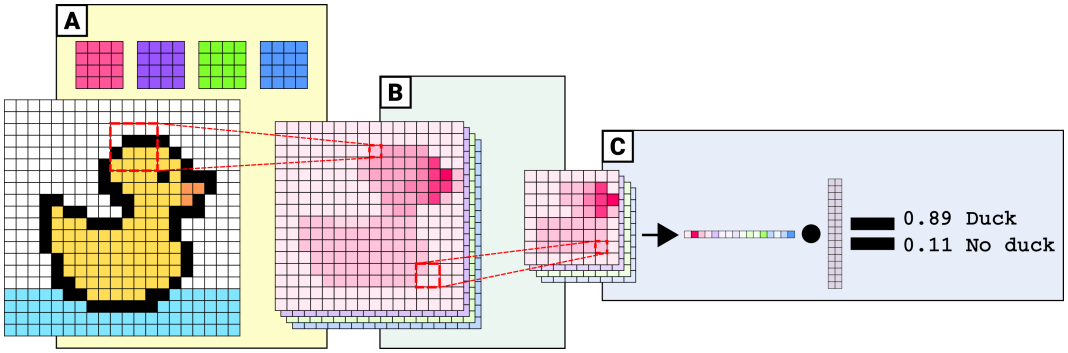

Figure 2.5: An illustration of how a CNN-based neural classifier works. **A** A set of 4×4 filter kernels, indicated by the coloured matrices, pass iteratively pixel-by-pixel across the image. They produce a series of feature maps which indicate the strength of interaction between each filter and specific the positions on the image. **B** These feature maps are downsampled. Here only the maximum value from each non-overlapping region of 2×2 pixels is retained. **C** The downsampled feature maps are flattened and combined into a single vector, which is multiplied by a set of parameter weights. Then typically a sigmoid or softmax activation function is applied to the resultant values to convert them into classification probabilities. The activation functions have been omitted from this figure for simplicity.

and finally a classification layer. This paper exceeded all previous attempts at the ImageNet [55] classification task, and in doing so popularised many techniques now ubiquitous across the field of deep learning, namely: the use of rectified linear units (ReLU) as an activation function; dropout during training; and the use of Graphical Processing Units (GPUs) [122].

Innovation in initialisation, activation, and regularisation The rapid improvements in the efficacy of neural nets during this time [196, 122], CNNs at image recognition in particular, led to a growing interest into deep learning research. In this period, Glorot and Bengio [70] investigated why deep feed-forward architectures do not have the efficacy of other deep neural architectures. Additionally they wanted to investigate why models with an unsupervised pre-training phase had been observed to perform better than those that do not, such as those in the work by Jaret *et al.* [99]. Their experiments revealed that the characteristics of activation functions coupled with the weight parameter initialisation can have a big impact on the ability of a feed-forward network to learn useful task-related features.

They note that the sigmoid activation function can saturate with low or high inputs with low and high inputs tending towards asymptotic values [70]. When this is combined with the common weight initialisation procedure of setting random values close to zero, this can lead to the vanishing gradient problem. As the error is backpropagated through the parameter weights the gradient gets diminished with each consecutive multiplication of the gradients. This results in lower layers being unable to learn useful features. The authors find that changing the activation func-

tion to one that does not saturate at zero, such as a hyperbolic tangent function, can somewhat mitigate this phenomena [70]. Please see figure 2.6 for a plot of the sigmoid activation function and its derivative.

Glorot and Bengio [70] also postulate that the observed benefit of pre-training largely derives from understanding the process as a form of weight parameterisation that avoids the vanishing gradient problem. Pre-training locates the parameters near "good local minima" that are non-random and have a greater inductive bias so the model can better learn lower-level features. The authors propose a new weight initialisation that scales the initial parameter weights proportionally to the number of input and outputs to a neuron in the layer. This initialisation (later called "Glorot initialisation") mitigates the relative advantage given by an expensive pre-training regimen.

The introduction of rectified non-linear activation functions have been key to the development of deep learning, in particular CNNs for image classification tasks [99, 71, 140, 114, 84, 122]. Jarret *et al.* [99] wanted to understand how different non-linear functions influence the accuracy of different object recognition models, that either had hand-crafted, random, or learned feature extraction kernels. They discovered that the use of activation functions inspired by biological neurons, rectified non-linearities, were the biggest factor in model performance. Additionally, they found that these type of activations are necessary to adequately learn suitable feature extraction kernels. The authors argue that this is because the "polarity of features are often irrelevant to recognise objects" [99].

Rectified non-linearities are defined by their property of rectifying all values below zero to zero. The simplest rectified non-linearity activation is the Rectified Linear Unit (ReLU) activation, given by $f(x) = \max(0, x)$. Its derivative is either one when $x > 0$, or zero. This means that the error is either backpropagated as-is or not at all, as the derivative of 0 and x with respect to x are 0 and 1 respectively. This mitigates the vanishing gradient effect. In contrast, the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ has the derivative $\sigma'(x) = \sigma(x)(1 - \sigma(x))$. The maximum derivative of the sigmoid function is 0.25 when $x = 0.5$. This results in an error signal backpropagated through a sigmoid activation being reduced by at least a quarter - significantly contributing to gradient vanishing. Nair and Hinton [158] and Maas *et al.* [140] found that using ReLU as an activation function led to better performance on several tasks, when compared with those that used a sigmoid activation. Both studies also found that as ReLU mitigated the vanishing gradient problem somewhat, and caused their models to reach convergence much more quickly.

Maas *et al.* [140] however, noted that there are limitations to using ReLU activation functions. As mentioned previously, ReLU does not allow the backpropagation of errors through neurons with negative activations. This means that if a neuron is always negative during training it will not learn any useful representations and will not contribute at all to a model [140]. Maas *et al.* [140] developed a version of

ReLU, that came to be called Leaky ReLU (LReLU). LReLU alters ReLU such that $f(x) = \alpha x$ when $x < 0$, where α is some small fixed parameter (e.g. 0.01). This means that negative neuron activations are kept close to zero, while also allowing for the backpropagation of the error through these neurons. Please see figure 2.6 for a visualisation of the Sigmoid, Tanh, ReLU and Leaky ReLU activation functions and their derivatives. Note that the Leaky ReLU appears very similar to the ReLU activation function except that the derivative for $x < 0$ is not zero.

He *et al.* [84] further developed the LReLU so that each neuron had its own α parameter. This Parametric ReLU (PReLU) adaptively learns the α parameters for each neuron for the rectifier, optimising them for model performance. Their experiments showed that lower layers tended to learn a value of α that made the activations more linear, while deeper layers had smaller values, tending to be more non-linear. The authors postulate that for feature detection, such as edge detection or texture detection, negative features can be useful, however later layers need to discriminate between non-linear combinations of extracted features. The authors noted that the Glorot initialisation [70] assumes that the weight activations are linear which is invalid in the case of rectified non-linear activation functions [84]. They proposed a new weight initialisation, that came to be known as He initialisation, where weights were sampled from a zero-mean Guassian distribution with a standard deviation based upon the number of input connections to the layer. Together these innovations enabled their model to exceed human performance on the ImageNet [55] dataset [84].

Another important innovation popularised by AlexNet is dropout. Hinton *et al.* [91] made the observation that deep neural nets tend to overfit to the training data, which inhibits their ability to generalise leading to poor performance on predictions on inputs from outside of the training data. Their proposed solution is a technique inspired by sexual reproduction [212]. Sexual reproduction is a process whereby two organisms combine random subsets of their genes. As a result each gene must have some level of redundancy so that it provides some fitness benefit given various gene set contexts, reducing the chance that organisms develop complicated fragile gene processes. This process is emulated by dropout. At each step during training, the output of a neuron is randomly set to zero with a certain probability. These neurons are removed or "dropped out" for that training iteration. The remaining neurons' outputs are scaled proportionally to the inverse of the dropout probability, to maintain the absolute magnitude of the input to the next layer. The model is forced to learn many configurations of neurons, distributed across its architecture, to make a certain prediction, as opposed to relying on specific neurons for the prediction. Conceptually, the model now behaves like an ensemble of many smaller models, with the final prediction being an average of all the models' contributions. Hinton and his colleagues report that dropout improved the performance of multiple architectures across different tasks. The models were more robust to noise and could generalise

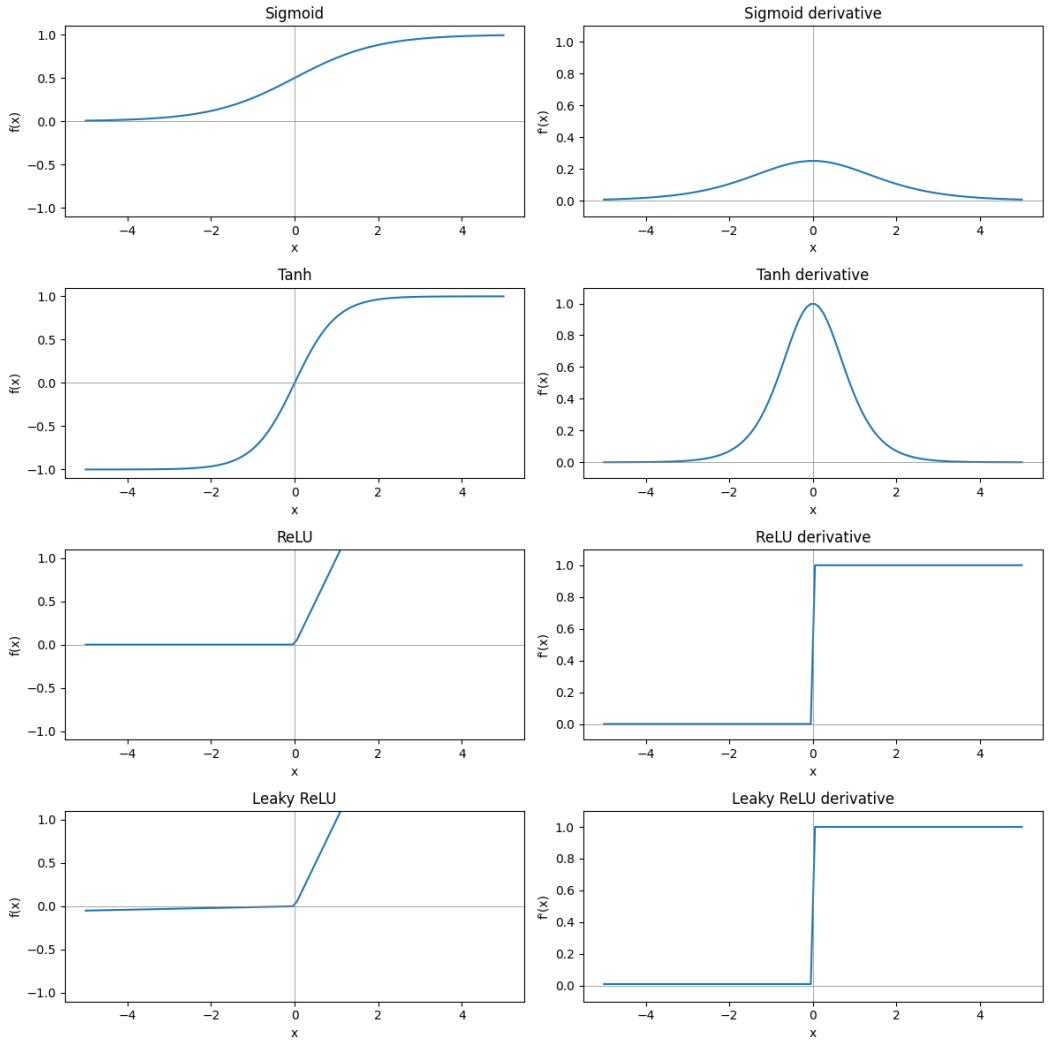

Figure 2.6: Plots of the Sigmoid, Tanh, ReLU, Leaky ReLU, and their respective derivatives for values of x between -5 and 5. Here the Leaky ReLU has an α parameter set to 0.01.

better to data unseen during training [91].

Wan *et al.* [225] introduced a version of dropout that instead of dropping entire neurons, drops the parameter weights for fully connected layers, which link the neurons of one layer to those of another. This means that certain neurons only receive information from a fraction of the neurons in the previous layer. This has the effect of helping the model to generalise, while ensuring that each neuron is likely to be involved with the prediction of each pass. The difference in the character of the network thinning between dropout applied to the parameter weights versus dropout applied to neuron activations is illustrated in figure 2.7.

Putting the "Deep" into Deep Learning: Normalisation and residuals
The deeper a network becomes, the more small changes to early parameters can be

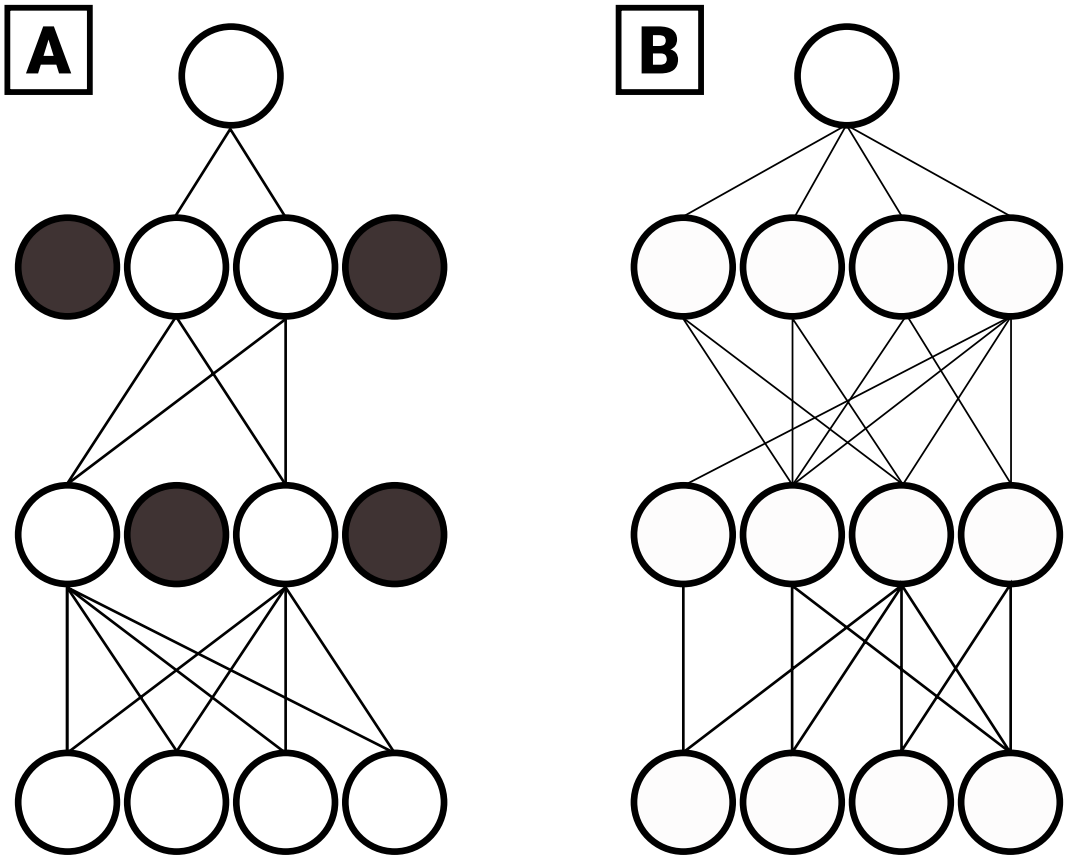

Figure 2.7: An illustration of dropout applied to **A** neuron activations (as described in Hinton *et al.* [91]) and **B** parameter weights (Wan *et al.* [225]). Black circles depict dropped neuron activations; white circles, active neurons. Lines between neurons indicate active connections between neurons.

amplified by later layers, as they significantly influence the inputs of the layers that follow [98, 12]. This phenomenon, where each proceeding layer has to continually adjust to changes to the distribution of their inputs, is known as covariate shift [98]. Ioffe and Szegedy [98] state that each layer in a neural network benefits from their inputs having a consistent distribution, so that the layers learn useful transformations of the input rather than also compensating for any shifts in its distribution.

As a way of addressing internal covariate shift, Ioffe and Szegedy [98] proposed batch normalisation. It works by making the outputs of a layer have a mean of zero and a unit variance (the variance is one), by doing the following: Given a batch, the mean and variance are calculated for each feature. Each feature corresponds to, for an image, the same element in a matrix of pixels across all channels and all samples in the batch. For a batch containing sequences of vectors, then each corresponding element across all vectors. These batch statistics are an estimate of the global mean and variance. Each feature then has the mean subtracted, and is then divided by the

standard deviation (the square root of the variance), which standardises the input.

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}} \tag{2.26}$$

Where $\hat{x}^{(k)}$ is the standardised input for layer k, $x^{(k)}$ is the original input for layer k, and $\mathrm{Var}[x^{(k)}]$ is its variance and $\mathrm{E}[x^{(k)}]$ its mean. The model then learns parameters to scale (γ) and shift (β) the standardised inputs as follows:

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)} \tag{2.27}$$

With $y^{(k)}$ being the final inputs for layer k. The authors added this learned scaling and shifting to prevent the inputs of a sigmoid activation being held at the linear section of the function and thus enforcing linearity. These parameters also allow the layer to unlearn the normalisation if it is beneficial. During training the model records average feature means and variances over the entire training set. At inference time these global feature mean and variance values are substituted into the standardisation process.

Ioffe and Szegedy [98] applied batch normalisation to an image classification model, and exceeded the state-of-the-art performance for a non-ensemble model on the ImageNet [190] dataset. Batch normalisation reduces the influence of the scale of parameters on the gradients, and reduces the dependence between parameters and their initial values and makes training more stable. It was found to mitigate the insensitivity caused by saturating non-linearites such as sigmoid and hyperbolic tangent. Finally, as parameter changes are less likely to be amplified into sub-optimal changes to gradients, the learning rate can be increased speeding up training.

Ba *et al.* [12] noted that batch normalisation has some limitations. Firstly, that batch normalisation is dependent on the batch size. When the batch size is small the mean and variance of the features in a batch can vary dramatically when compared with larger batch sizes which tend to have more stable means and variances. This impacts use cases where the batch size is small due to memory constraints. Secondly, having inputs that represent sequences of varying lengths may cause inconsistency in the application of batch normalisation.

To prevent covariate shift while mitigating the limitations of batch normalisation in those contexts, Ba *et al.* [12] proposed layer normalisation. For a given hidden layer, and the vector of summed inputs of each neuron in the layer a, layer

normalisation can be described as followed:

$$\mu = \frac{1}{H}\sum_{i=1}^{H} a^l \tag{2.28}$$

$$\sigma = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a-\mu)^2} \tag{2.29}$$

Where H is the number of hidden units in the layer, and μ and σ are the mean and standard deviation of the hidden units of the layer, respectively. While the batch norm, equation 2.26, computes statistics across corresponding features or hidden units in the batch, layer normalisation normalises each feature independently. With layer normalisation each vector in a sequence of vectors, or that each feature map in an image dataset, is normalised to have a mean of zero and a unit variance. This makes it invariant to batch size, and running global mean and variance values do not need to be stored during training.

Ba *et al.* [12] tested layer normalisation on numerous benching tasks finding that, for tasks that involve sequence processing, layer normalisation results in faster training and better performance. However, when compared to batch normalisation in CNNs for image processing, layer normalisation is sub-optimal and does not perform as well as batch normalisation.

The depth of neural nets had always been constrained by the observed phenomena of "degragation" whereby deeper networks do not produce as optimal solutions as shallower networks [85]. He and colleagues [85] argued that if we have two networks, one shallow and another deep, the solution found by the shallower network could be constructed into a deep model by adding layers that behave as an identity mapping, whereby their input is identical to their output. The implication is that, hypothetically, a deep model should be able to learn the solution of a shallower model.

The authors proposed a solution they christened "Residual Connections" [85]. Given the desired underlying mapping of $\mathcal{H}(x)$, they describe the stacked non-linear layers as $\mathcal{F}(x) := \mathcal{H}(x) - x$, and so reformulate the layer to $\mathcal{H}(x) = \mathcal{F}(x) + x$. In practice this is achieved by adding the input values of a layer to its outputs. With this explicit formulation the non-linear layer then learns the difference, or residual mapping, between its input and outputs. The authors posit that in an exaggerated case, if the identity mapping did not need to be changed to produce the best output, then it would be much more convenient for the model to learn to make the outputs of the layer closer to zero than it would be to learn to reproduce an identity mapping through a non-linear layer [85].

These residual connections were incorporated into image classification CNNs of varying depth, called Residual Networks (ResNets), and compared against the same architectures minus the residuals [85]. A ResNet with 110 non-linear layers achieved

the best ever score for the CIFAR-10 image classification task [113]. While a ResNet with 152 layers achieved the best score in the ImageNet image classification challenge [190] 2015, and a ResNet model achieved 1st place in the Microsoft Common Objects in Contexts (COCO) object detection challenge [126] 2015 - successfully demonstrating that residual connections enable deeper models to effectively learn better solutions than shallower models. ResNets saw wide adoption following their inception [221, 171].

2.1.3 Processing sequences

Capturing the order in which events happen has long been an interest to probabilistic AI researchers [189, 62]. While Rumelhart and colleagues' [189] backpropagation algorithm made a great contribution to training feed-forward networks, they had limitations when applied to sequence data [232]. The Recurrent Neural Network (RNN), and the Back Propagation Through Time (BPTT) algorithm to train it, were developed to address these limitations [232, 228]. The RNN, as described by Williams and Zipser [232], is similar to the sequence processing approach suggested by Rumelhart *et al.* [189] but without the requirement for a growing memory to capture information on previous network states. Here the state of the model at each timestep t is a function of the previous timestep's input and the outputs of the timesteps preceding that. The network state, or model output, at timestep $t+1$ ($\mathbf{y}_{t+1}$) can be computed like so:

$$\mathbf{y}_{t+1} = \sigma\left(\mathbf{W} \cdot \text{concatenate}([\mathbf{y}_t, \mathbf{x}_t])\right) \tag{2.30}$$

Where $\mathbf{x}_t$ is the model inputs at timestep t, $\mathbf{W}$ is the matrix of trainable parameter weights, and σ represents the logistic sigmoid activation function. This arrangement prevents the input at time t from influencing the state of the network until time $t+1$.

Williams and Zipser [232] trained their model starting with the first timestep of a sequence and progressing iteratively until the last. The partial derivatives of the weights with respect to the error of each individual timestep were calculated. These derivatives are used to update the weights. This means that the error of the more immediate timesteps will tend to dominate the weight updates [20]. In a BPTT-like setting this can lead to the "vanishing gradients" problem, where the error signal from previous timesteps diminishes as the sequence progresses - hindering the learning of long range dependencies [20].

Elman [62] proposed the Simple Recurrent Neural Net (SRNN). The SRNN consist of input and output layers, and an internal hidden layer that is connected to a hidden context layer which stores the hidden layer from the previous timestep. The input layer transforms the input of a timestep into the representational space of the hidden layer. The hidden layer passes information to the context layer, while the context layer passes task specific information, from the previous timestep, back

to the hidden layer. Finally the output layer converts the hidden representation into the output space. The sequential processing of inputs combined with the context layer, which behaves as a memory, allows the model to learn to capture beneficial contextual information and transmit it between timesteps. The architecture can be defined as follows:

$$\mathbf{h}_t = f\left(\mathbf{W}_h \cdot \mathbf{h}_{t-1} + \mathbf{W}_x \cdot \mathbf{x}_t\right) \tag{2.31}$$

$$\mathbf{y}_t = g\left(\mathbf{W}_y \cdot \mathbf{h}_t\right) \tag{2.32}$$

Where $\mathbf{h}_t$ is the hidden state vector, $\mathbf{h}_{t-1}$ the context vector, $\mathbf{x}_t$ the vector of inputs, and $\mathbf{y}_t$ is the vector of outputs all at time t. Both $f(.)$ and $g(.)$ are non-linear activation functions. $\mathbf{W}_h$, $\mathbf{W}_x$, and $\mathbf{W}_y$ are the trainable parameter weights for passing information from the context vector to the hidden layer, the current input vector to the hidden vector, and the hidden vector to the output respectively.

Elman's [62] experiments showed that the model could be used to predict the next word in a sequence, and was sensitive to dependencies between timesteps as demonstrated by the model appearing to learn grammatical structure. These findings brought attention to the value of capturing temporal context in neural networks designed for sequential data. Although the SRNN can capture dependencies between different elements in a sequence, they still struggle with longer range dependencies as the more numerous short range dependencies tend to dominate the gradients [20].

In 1997, expanding on the RNN, Hochreiter and Schmidhuber [92] introduced the Long Short-Term Memory (LSTM) architecture. Prior to this, using back propagation to train RNNs tended to result in the gradients "blowing up" (becoming so large as to swamp any meaningful weight updates) or "vanishing". The reasons for this become clear if you conceptualise an RNN as a very deep network, that uses the same layer repeatedly, with the first timestep's input passed to the second layer, the second output passed to the third layer with the previous layer's information through the context layer and so on. As a result, prior sequence processing models would struggle to maintain representations of useful information for later steps in a sequence.

The LSTM mitigates this problem by using a system of memory cells and gates that control the flow of information into, and out of, the cells [92]. These gates consist of: an input gate which determines how much of the current step's input representation should be permitted into the memory cell; and an output gate which determines how much of the current memory cell's internal state should be passed to the next layer of the LSTM. I will now formally define the architecture. Firstly, the input gate, $\mathbf{i}_t$, at time t:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \cdot \text{concatenate}([\mathbf{h}_{t-1}, \mathbf{x}_t]) + \mathbf{b}_i) \tag{2.33}$$

The new memory cell, $\hat{\mathbf{c}}_t$:

$$\hat{\mathbf{c}}_t = \tanh(\mathbf{W}_c \cdot \text{concatenate}([\mathbf{h}_{t-1}, \mathbf{x}_t]) + \mathbf{b}_c) \tag{2.34}$$

To update the cell state, $\mathbf{c}_t$:

$$\mathbf{c}_t = \mathbf{i}_t \odot \hat{\mathbf{c}}_t \tag{2.35}$$

The output gate, $\mathbf{o}_t$:

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \cdot \text{concatenate}([\mathbf{h}_{t-1}, \mathbf{x}_t]) + \mathbf{b}_o) \tag{2.36}$$

Finally to update the hidden state, $\mathbf{h}_t$:

$$\mathbf{h}_t = \text{o}_t \odot \tanh(\mathbf{c}_t) \tag{2.37}$$

Where $\odot$ signifies element-wise multiplication, the $\mathbf{W}$ matrices are the trainable parameter matrices and $\mathbf{b}$ the biases for each specific function. $\mathbf{x}_t$ is the vector of inputs at time t. tanh(.) and $\sigma(.)$ represent the hyperbolic tangent and sigmoid activation functions respectively. Depending on the task either the hidden representation for the final timestep is, or some function of all timesteps are, passed to a further classification layer.

The results of Hochreiter and Schmidhuber's experiments [92] showed that the LSTM outperformed traditional RNNs on a number of sequence processing tasks. This is likely due to the LSTM's ability to more effectively remember previous timesteps. Although the LSTM is good at ordering timesteps relative to each other, it struggles with capturing information on the specific hops between timesteps. The authors posit that the memory cells ensure that, although the error is not back-propagated across timesteps, error signals can be held indefinitely [92]. The series of gates allows the model to limit the influence of both irrelevant inputs on the hidden state and of irrelevant memories on the output, while also mitigating the vanishing and exploding gradient problems associated with previous RNN models [69].

Gers *et al.* [69] identified another shortcoming of the LSTM: the memory cell vector has been observed to grow linearly from the start of the first sequence it is exposed to during training and onward [69]. This means that the activation functions of various gates can become saturated as the values grow, even if a new sequence is passed to the LSTM. Eventually, the saturation of the memory cell effectively prevents it from influencing the processing of the sequence. Gers *et al.* [69] proposed the "forget gate" so that the LSTM can learn to flush the contents of the memory cell when their contents becomes irrelevant. Their forget gate, f(t) at time t, can be described thusly:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \cdot \text{concatenate}([\mathbf{h}_{t-1}, \mathbf{x}_t]) + \mathbf{b}_f) \tag{2.38}$$

They also amend the memory cell update:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathrm{i}_t \odot \hat{\mathbf{c}}_t \tag{2.39}$$

Gers and colleagues's experiments [69] showed that the LSTM could now effectively reset its memory cell at the beginning of a new sequence without being informed by some hand crafted cue or external intervention. The LSTM with a forget gate could solve problems that an LSTM without a forget gate could not, in particular those involving continual prediction such as predicting the next token in a sequence [69].

Schuster and Paliwal [199] introduced Bidirectional RNNs (Bi-RNNs), which comprise a combination of a forward and a backward RNN. Given an input sequence, both the forward (timestep 0 to T) and backward (T to 0) RNNs compute outputs for each timestep in the sequence. The hidden states of the positive RNN are not seen by the negative and vice versa. However, their output representations are concatenated together. This means that at inference time the output encodes information from across the entire sequence. Although the RNNs do not see each other's hidden states, the backpropagation along the separate RNNs is informed by a loss that takes account of the entire sequence. When tested on a synthetic task, plus phoneme classification and speech recognition tasks, the Bidirectional RNNs out performed single-directional RNNs. This demonstrated that past and future information can be leverage to improve model performance.

A study by Graves and Schmidhuber [74] explored different architectures for the task of phoneme classification, including one incorporating a Bidirectional LSTM (Bi-LSTM), a typical LSTM, feed-forward networks, a type of network called a Time Delay Neural Network (TDNN), RNNs, and Bi-RNNs. The Bi-LSTM is similar to the approach outlined for RNNs by Schuster and Paliwal [199], however the authors here also stacked Bi-LSTMs on top of one another so that there were multiple hidden layers, with the first LSTM processing the inputs, and subsequent LSTMs processing the outputs of the previous LSTM [74]. They also did the same with the Bi-RNNs. Audio data was split into frames and the architectures classified the sequences of frames into different phoneme types. The Bi-LSTM outperformed the other architectures significantly, and converged to a solution an order of magnitude more quickly than the next best performing architecture the Bi-RRN. The authors also found that training a Bi-LSTM twice, once with a weighted error function and then again with typical cross-entropy error, improved the performance of the model. The authors believe this helps the model to climb out of local minima. Additionally, the wider the context window that the model had access to, the better the models performed.

A limitation of sequencing labelling algorithms, including RNNs and LSTMs, is that they often require the sequence to be segmented appropriately prior to labelling, which in turn requires some understanding of which input features correspond to

output features, a process called alignment [75]. Acquiring the necessary information to align the segments may not be trivial, limiting the applicability of these models [75]. Graves and colleagues [75] identified that this limitation arises due to the loss being computed independently for each step in the series, rather than simultaneously with respect to each others labels. This means that dependencies between labels that may be inherent in the sequence data may not be learned. The authors proposed a remedy to this limitation, Connectionist Temporal Classification (CTC) [75], which treats the model outputs as a probability distribution over all possible sequences of labels given an input sequence. This allows the model to learn implicit associations between the labels of a sequence without the need for the sequence data to be segmented prior to learning.

The authors [75] formulate the problem like so: Given a set of training examples S comprising the vectors of input features $\mathcal{X}$ and the set of labels $\mathcal{Z}$. The samples in S are pairs of input sequences and label sequences $(\mathbf{x}, \mathbf{z})$. The input sequence is at least as long as the label sequence but usually longer, so the sequences cannot be aligned practically *a priori*. Given a test set S', containing examples not included in the set S, the authors define an error function, the Label Error Rate (LER). This function can be described as a length normalised edit distance between predicted and ground truth labels:

$$\mathrm{LER}(h, S') = \frac{1}{|S'|} \sum_{(\mathbf{x},\mathbf{z}) \in S'} \frac{\mathrm{ED}(h(\mathbf{x}), \mathbf{z})}{|\mathbf{z}|} \tag{2.40}$$

Where $h(.)$ is the sequence classification model, and $|S'|$ and $|\mathbf{x}|$ are the number of samples in the test set and the number of steps in a sequence respectively. $\mathrm{ED}(h(\mathbf{x}), \mathbf{z})$ represents the edit distance between the model outputs and the ground truth labels, which is a count of the minimum number of individual changes to transform the output of the model into the ground truth.

During training the raw outputs of each step in the model are converted into probabilities over all possible segment labels using the softmax function. The probability of a label k at a given step t is $P(k|t)$, which is computed as the softmax over the network outputs at step t for the index of label k. Additionally, the algorithm learns the total probability of observing the label sequence preceding and including step t and its label k, and the analogous probability for the remainder of the label sequences from s and k onward. How CTC incorporates the probability of the preceding sequence of labels using recursive functions can be simplified as follows:

$$\alpha_t(s) = \alpha_{t-1}(s) + \alpha_{t-1}(s-1) \tag{2.41}$$

$$C_t = \sum_s \alpha_t(s) \tag{2.42}$$

$$\hat{\alpha_t(s)} = \frac{\alpha_t(s)}{C_t} \tag{2.43}$$

Where s represents the index of the output label sequence. Here $\alpha_{t-1}(s)$ represents the probability of the previous step in the input $t-1$ and the current input step t both aligning with the same output index s, while $\alpha_{t-1}(s-1)$ is the probability of the current and previous steps aligning to different labels in the output sequence. Similarly, the probability of the sequence of labels for the remainder of the sequence can also be simplified as:

$$\beta_t(s) = \beta_{t+1}(s) + \beta_{t+1}(s-1) \tag{2.44}$$

$$D_t = \sum_s \beta_t(s) \tag{2.45}$$

$$\hat{\beta_t}(s) = \frac{\beta_t(s)}{D_t} \tag{2.46}$$

Iteratively dividing by the sum of probabilities for each possible output label at step t is meant to ensure that the probabilities do not become vanishingly small when they are multiplied together recursively - to avoid underflow. Finally the total probability of a set of output labels given a model output is defined as followed:

$$P(\mathbf{z}|\mathbf{x}) = \sum_t \hat{\alpha}_t(s) \cdot \hat{\beta}_t(s) \cdot P(k|t) \tag{2.47}$$

Which is the combined probabilities of the label, given the preceding and proceeding labels, and the model outputs, at step t.

In experiments they compared a CTC RNN and a CTC LSTM, which were trained on unsegmented data, against both a Hidden Markov Model and one combined with a Bi-LSTM, trained on segmented data [75]. These models were evaluated on a phonetic labelling task, which is a type of speech recognition task. The best CTC models performed as well as the best performing Hidden Markov models, however they did not require the input data to be manually segmented. This work demonstrated that neural models can classify raw sequence data directly without the need for manual feature engineering, learning to implicitly segment the data during training. This is particularly valuable in task such as handwriting recognition or speech recognition, where specific boundaries between the features represented across multiple steps can be ambiguous.

Neural Language Modelling Sequence-based neural networks developed in tandem with neural representations of language [22, 151, 152, 214]. Earlier models of language typically centred around modelling the conditional probabilities between n-grams, n-length sequences of co-occurring words, and the next word in a sequence [22]. Bengio *et al.* [22] noted that these models soon face problems due to the "curse of dimensionality". This was illustrated by an example they gave: If using these approaches to model the joint distribution of 10 words in a sequence, with a vocabulary of 100 000 words, then there may be as many as $10^{50}-1$ model paramet-

ers [22]. Additionally, these models cannot compute any probabilities for a sequence containing words from outside the fixed vocabulary.

As a solution, the Bengio and colleagues [22] proposed a neural network-based language model. For a given vocabulary V, and a training set comprised of word sequences $\{w_1, ..., w_T\}$ where $w_t \in V$. The model can be defined as:

$$f(w_t, \ ... \ , \ w_{t-n}) = \hat{P}(w_t|w_1^{t-1}) \tag{2.48}$$

Where $f(.)$ is the learned model, and $\hat{P}(w_t|w_1^{t-1})$ is log probability of the word w_t given the preceding sequence w_1^{t-1}.

The authors divide this function $f(.)$ into two parts: Firstly, a function for mapping each word in the vocabulary to a distributed feature vector. This vector, or embedding, represents the semantics of the word as a point in continuous vector space. Secondly, a function for computing the probabilities of each word in the vocabulary being the next word in the sequence. In this case, the function is a feed-forward network.

In essence, the model takes an input of word embeddings and outputs a vector. This vector is compared with all the embeddings of the words in the vocabulary by computing the dot-product between this vector and the embeddings. The log-softmax of the resultant vector of dot-product logits converts it into a vector of log probabilities, where the ith element of the probability vector corresponds to the probability of the ith word in the vocabulary. During training the model aims to maximise the log probability of the correct word by learning to produce a vector that is in close vicinity to the embedding of the correct word. Out of vocabulary words are represented by averages of the known words in the sequence. The authors' experiments [22] showed that this neural language model could predict the next word in a sequence with more accuracy than a contemporaneous n-gram-based method.

Mikolov *et al.* [151] applied the Elman RNN [62] to the task of language modelling, replacing the feed-forward networks used in previous works like Bengio *et al.* [22]. The RNN has some noted advantages over the feed-forward network. Whereas for the feed-forward one needs to determine the size of the layer that transforms words into dense embeddings, the size of the hidden representations, and the size of the context window. For an RNN, only the hidden representation size needs to be considered. The authors also note that it took six hours to train the RNN implementation on a dataset for which it took Bengio's feed-forward network 113 days [151].

In their experiments, Mikolov *et al.* [151] found that RNN-based language models significantly outperformed the contemporary state of the art - even when the other models were trained with a hundred-times greater volumes of data. This, the authors argue, shows that language models can be improved through architectural changes rather than just collecting more training data. Which, they add, aligns language

modelling more closely with the machine learning field.

Mikolov and colleagues [152] later proposed various improvements to the RNN language model. One of the improvements centred around categorising words in the vocabulary into classes. Here, each word is assigned to one class, and the model is adapted to produce a vector for representing the class the word belongs to, in addition to the vector representing the predicted word. Each class is represented with an embedding, and is associated with particular indices of the matrix of word embeddings, representing the words that belong to this class. This means that the model can estimate the probability over classes given the context, and then estimate the probability over the words in that class, given the context. This is known as a hierarchical softmax, where the probability of a word given its context becomes:

$$P(w_i|\text{context}) = P(c_j|\text{context})P(w_i|c_j, \text{context}) \tag{2.49}$$

Where c_j is the jth class, and w_i is the ith word in the vocabulary of the jth class. This means that at prediction time the model only has to compute the most similar class from a reduced number of class embeddings, and then look up the most similar word embedding from that class' subset of the vocabulary. This could potentially greatly reduce computational complexity, with Mikolov and colleagues' [152] experiments showing that there were trade offs between model performance and prediction latency. A greater number of word classes lead to better model performance, while after a certain number of classes it increased the time taken to compute a prediction.

Another improvement involved the creation of an ensemble of RNN models. Here the outputs of multiple RNN models were linearly interpolated. This means that the predictions of each model were combined using a weighted sum, where the coefficients of each models prediction were learned parameters. Linear interpolation resulted in significant improvement in model performance [152]. This work demonstrated again the importance of architectural design on model performance.

Sundermeyer *et al.* [214] also used a type of recurrent network for language modelling. However noting the limitations of the RNN, the authors opted to use an LSTM which incorporated a forget-gate as in Gers *et al.* [69]. The LSTM was incorporated into an architecture that included a set of trainable parameter matrices: an input projection matrix, a word class classification matrix, and a word classification matrix. The separate word class and word classification matrices are to compute a hierarchical softmax as in Mikolov *et al.* [152]. Their model also used one-hot encoding vectors to represent each word in the vocabulary. These one-hot encodings are vectors which have a dimensionality equal to the vocabulary lengths. They are all zeros except for a one at the index that corresponds to the index of the specific word in the vocabulary.

The input matrix linearly projects the context words' one-hot encodings into a

dense vector space. In practice, this means that the rows of the input projection matrix, at the indices that correspond to the index of each context word in the vocabulary, are extracted. Each row of the parameter matrix represents a word's embedding. The dense vectors of the context words are iteratively fed into the LSTM to compute the hidden layer representation of the step corresponding to the word to be predicted. This hidden representation is then multiplied by the word class classification matrix followed by a softmax function. The resultant vector represents a probability distribution over the classes. The same hidden representation is then multiplied by the word classification indices corresponding to the correct class, and again the outputs are passed to a softmax function so that they represent a probability function over the word in that class.

During training, the model learns to maximise the probability of the correct word, given the correct class and the preceding words. The author's experiments showed that the LSTM language models, as for other tasks, see improvement over the performance of typical RNNs at language modelling [214].

Mikolov *et al.* [150] noted that training a neural language model also leads to the learned representations of words in continuous vector space, or word embeddings. These word representations can be used to great effect as inputs for other unrelated tasks. Mikolov and colleagues investigated how these word embeddings captured the syntactic and semantic patterns that made them so useful for downstream tasks. To understand the syntactic patterns, they tested the word embeddings from an RNN language model using a series of analogy questions. These analogy questions comprised the form "a is to b what c is to _", comparing simple adjectives with their comparative and superlative forms, singular nouns with their plurals, possessive nouns with their non-possessive versions, and simple present tense verbs with their third-person and past tense forms. To test the semantic information encoded into the word embeddings, the authors use another analogy task from the second task of SemEval-2012 [185]. This time the semantic similarity of word pairs is used to rank them in the order that the word-pair relationship analogy holds.

For both of these tasks, the authors assume that a simple vector offset suffices to capture the relationship between two words. Whereby all embeddings are normalised to a unit absolute sum, and they compute $y = x_b - x_a + x_c$, where y is the representation of the word we expect to best answer the analogy question, and x_b represents the word embedding for word b and so on. As y does not precisely represent the answer, the cosine similarity is used to find the most similar word embedding out of the entire matrix of word embeddings x_w like so:

$$x_d = \text{argmax}_w \left(\frac{x_w y}{||x_w||\ ||y||} \right) \tag{2.50}$$

The cosine similarity is a measure of angular distance. The argmax is a function which returns the index of the element with the highest vector, indicating the vocab-

ulary index of the most similarly represented word. The embeddings produced by the RNN model performed as well as the next best method for the syntactic analogy task, while they far outperformed all other methods for the semantic analogy task, despite not being trained for this task [150]. This paper demonstrated the utility of unsupervised training for creating rich dense vector embeddings of words, that reliably capture the syntactic and semantic relationships required for downstream tasks.

After previously establishing the usefulness of language models and their word embeddings, Mikolov *et al.* [149] noted that RNN-based language models tended to be computationally expensive to train. So they explored if architectural design could be altered to improve computational efficiency of training, without compromising on embedding quality, so that models could feasibly be trained on much larger corpora. Both models are a simplification of the Neural Language Model by Bengio *et al.* [22].

Here they introduced the Continuous Bag-of-Words model and the Continuous Skip-gram model [149]. The Continuous Bag-of-Words model is trained to predict a word from its context. It does this by first representing every word in the vocabulary as a hot-hot encoding. As in Sundermeyer *et al.* [214], these are used to retrieve the context words' embeddings. These embeddings are then averaged into a single vector. The dot product of this vector and the classification parameter matrix is computed, transforming the vector so it has a dimensionality equal to the vocabulary size. This vector is passed to a softmax classifier to compute the probability of each word in the vocabulary. This outputted probability of the correct class is then used to calculate the loss.

The Continuous Skip-Gram model runs this process in reverse [149]. To train this model a word's context is predicted from that word. As before, the one-hot encoding is used to extract the word's embedding from the input parameter matrix. Then, for each word in the context, this vector is transformed into a vector that has the dimensionality equal to the vocabulary size using a separate parameter matrix. These are then transformed into probability distributions using the softmax function, and the probability of each word in the context is separately used to calculate the loss.

These training processes produce word embeddings whose semantic and syntactic properties exceeded that of those learned by RNN language models. These embeddings were used for tasks as described before in Mikolov *et al.* [150], with both models significantly outperforming previous approaches. These approaches would come to be know as word2vec [168].

Mikolov *et al.* [148] further improved the word2vec algorithms. As an alternative to the hierarchical softmax they proposed instead to use a negative sub-sampling algorithm based upon the Noise Contrastive Estimation (NCE) algorithm by Gutman and Hyvärinen [79].

NCE converts a multiclass classification problem into a binary classification task, simplifying the learning process [79]. During training with the NCE algorithm the log-softmax of the sub-sample is calculated and then weighted by the sample probability of each class in the sub-sample. This weighted log-softmax is used to calculate the loss. In this subsample, one class is correct while the others are incorrect. The model thus learns to distinguish between the positive class and the "noise" (the negative samples). Gutman and Hyvärinen [79] showed that the sample log-softmax approximates the full softmax scores of a sub-sample of the total classes, with more samples in a sub-sample leading to a better approximation.

Mikolov and colleagues [148] realised that they did not need to approximate the full softmax across all classes, as ultimately they were not training a classification model. Instead they just needed the model to learn to distinguish the true positive classes from the negatives. Their Negative Sampling algorithm simplified the NCE algorithm so that the model just learned a binary classification task without approximating the full softmax. In the continuous skip-gram case, rather than comparing the predicted context words against all the word classes, the predicted words were compared against a subset that included the positive class and sampled negatives. These negatives were sampled with proportions inversely related to their frequency in the training corpus, with rarer words more likely to be sampled.

Word2vec was extended by Le and Mikolov to create doc2vec [119]. Doc2vec adapts both the skip-gram and continuous bag of words methods so that they also generate vector representations of documents in the same space as the words, regardless of the document length. A vector is learned for each document in the training corpus. During training the document vector is included in the context of each prediction, either using the context words' vectors plus the document vector to predict the missing word, known as distributed memory (DM), or by a method which uses a word vector and the document vector to predict the context words, called distributed bag of words (DBOW). After the conclusion of training, a previously unseen document can have its embedding inferred by fixing the word embeddings but adjusting a new document vector so that the model makes the correct word predictions. The document vectors can be used for document similarity, clustering, or classification tasks.

Pennington *et al.* [168] introduced the Global Vectors for Word Representation (GloVe) model, aimed at combining context window-based language models, like the word2vec algorithm [149], with those that rely on global word co-occurrence statistics, such as Latent Semantic Analysis (LSA) [53]. Firstly, a global word co-occurrence matrix $\mathbf{C}$ is computed. Here $\mathbf{C}_{ij}$ is a count of the number of times word j appears in the context of word i in the corpus. The GloVe algorithm works by carrying out a kind of matrix decomposition. It tries to learn word embeddings that satisfy this expression:

$$\mathbf{w}_i^\top \cdot \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j = \log(\mathbf{C}_{ij}) \tag{2.51}$$

Where $\mathbf{w}_i$ is the ith word's embedding from embedding matrix $\mathbf{W}$, and $\tilde{\mathbf{w}}_j$ is the jth word's embedding from the separate $\tilde{\mathbf{W}}$ embedding matrix. Both b are bias terms for their corresponding words and embedding matrices. To do this, during training the loss function is given by:

$$L = \sum_{i,j=1}^{V} f(\mathbf{C}_{ij}) \cdot (\mathbf{w}_i^\top \cdot \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log(\mathbf{C}_{ij}))^2 \tag{2.52}$$

Where V is the number of words in the vocabulary, and $f(.)$ is a weighting function which scales the influence of rare word co-occurrences on the loss value L. So rather than decomposing the co-occurrence matrix into two matrices directly, the model learns word embeddings that capture global co-occurrence statistics, while also learning the statistical properties of words and their local contexts. When applied to the same word analogy and similarity tasks as the word2vec skip-gram and bag-of-words models, GloVe embeddings performed significantly better [168].

Bojanowski *et al.* [27] noted that as neural language models tend to treat words as atomic objects, ignoring sub-word morphology, they are not well predisposed to representing languages that have lots of noun cases or where verbs have multiple forms. Additionally, these models have difficulty representing out of vocabulary words. The authors outlined a sub-word-aware word embedding model, based on the skip-gram word2vec algorithm of Mikolov *et al.* [149]. Here, each word is represented by the character n-grams it contains. Each word also has word boundary markers $<$ and $>$ appended to the start and end of each word respectively. For example the word *apple* would be represented by the character n-grams, when $n = 3$, by <ap, app, ppl, ple, le>, and finally by a special <apple> token. The authors specify that they split a each word in to character n-grams for $n \in \{3, \ldots, 6\}$. Each of these tokens is represented by a vector embedding, and the full word is the represented by the sum of the embeddings of the sub-word tokens it contains [27]. Otherwise, the training procedure is the same as described in Mikolov *et al.* [148] and uses both the skip-gram and bag-of-words algorithms from Mikolov *et al.* [149]. Again, the word embeddings produced by the model are evaluated with word similarity and analogy tasks, and compared against the original word2vec implementations. The model achieved the state of the art performance on both tasks[27]. The model demonstrated particular improvement on representing rare words, and in languages with greater sub-word morphology.

Word embedding models typically only have one representation per word [22, 151, 149, 168, 27]. However, the meaning of a word can change depending on the context. A word with multiple meanings is *polysemous*. Peters *et al.* [171] proposed a language model, with contextualised word embeddings, called Embeddings from Language Models (ELMo), which also captures sub-word information. The model consists of a character-level CNN extracts sub-word features from the words, these

features are passed to a two-layer Bi-LSTM with residual connections. During training the objective function is to predict a word given the rest of the sequence, with one LSTM looking backwards, the other forwards.

Once the model is trained, the Bi-LSTM and character-level CNN weights are frozen. The model can then be used to produce contextualised embeddings for downstream tasks. These contextualised embeddings consist of the character-level features extracted by the CNN concatenated with the weighted sum of the hidden representations from both layers of both LSTMs for each token. The weights for the weighted sum are trainable parameters that can be tuned for downstream tasks.

When applied to various downstream tasks, ELMo improved error rates from between 6 and 20%. These tasks included question answering, textual entailment, semantic role labelling, co-reference resolution, named entity extraction, and sentiment analysis. The author's analysis suggested that the lower hidden representation captured more syntactic information, while the upper representation captured greater semantic information [171].

Machine Translation LSTMs were found to be particularly suited for language translation [215, 41]. Sutskever *et al.* [215] used a model comprising of four stacked LSTMs to the task of language translation. The input sequence would be followed by the target sequence separated by a special end of sentence token. The model learns to iteratively generate the correct next word from the inputs plus the previously generated words. At inference time the model employed a beam search. This means that at each step the model generates the k most likely outputs, selecting the most likely sequence once the maximum length or the end of the sentence token has been generated for all the sequences.

Cho *et al.* [41] incorporated an explicit encoder and decoder structure, where by one RNN encodes the input text sequence into a single latent vector representation and another decodes this into the output sequence. Specifically the final hidden layer of the encoder RNN, corresponding to the end of sequence token, is taken as the latent representation of the entire sequence, known as a "context vector". The decoder RNN's outputs are then conditioned on the encoded latent representation, the current hidden representation of the decoder, and the last output of the decoder.

Bahdanau *et al.* [14] argued that the compression of a sequence into a single vector representation is a bottleneck that hampers translation ability, especially for longer sequences. To address this, Bahdanau and colleagues [14] adapted an bi-directional RNN-based translation model so that the decoder employs an attention mechanism. The bidirectional RNNs encode the entire input sequence to produce a sequence of hidden representations, and then a mechanism to selectively attend to all hidden representations at once is used to iteratively decode the output sequence from these hidden representations. When the model is decoding the hidden states into the output sequence, it dynamically computes a context vector for each output

step using a weighted aggregate of the hidden states. The mechanism calculates the relevance of each hidden representation to the current step, given the state of the decoder. These relevance scores are used to create the context vector for this step, aggregating a weighted sum of all the hidden representations. The context vector is then used, along with the previous output and the current state of the decoder, to generate the current output word.

Given the decoder state vector $\mathbf{s}_t$ and the matrix of concatenated pairs of hidden states from the Bi-directional RNN $\mathbf{H} \in \{\mathbf{h}_1, \ \dots \ , \ \mathbf{h}_T\}$, the attention mechanism aggregates hidden states into a context vector $\mathbf{c}_t$ like so:

$$\mathbf{e}_t = \mathbf{s}_t^\top \cdot \mathbf{W}_a \cdot \mathbf{H} \tag{2.53}$$

$$\mathbf{a}_{t,i} = \frac{\exp(\mathbf{e}_{t,i})}{\sum_{i=1}^{T} \exp(\mathbf{e}_{t,i})} \tag{2.54}$$

$$\mathbf{c}_t = \mathbf{a}_t \cdot \mathbf{H} \tag{2.55}$$

Where the matrix $\mathbf{W}_a$ is a trainable parameter weight which learns to capture signals from the hidden representations. The dot product of these captured signals and the transpose of the decoder state vector produces raw logits which signify relevance between specific steps of the hidden state and the current decoder state. These are turned into attention scores, which sum to one, using the softmax function. Finally the attention scores are used to aggregate a weighted sum of all of the hidden state vectors determined by their relevance to the decoder state. As this attention mechanism attends to across the entire input and combines input features using weights that sum to one, it is regarded as a "soft" attention mechanism. The authors found this approach to significantly outperform previous methods, particularly for the translation of longer sequences [14].

Gehring *et al.* [68] adapted this soft-attention method so that the encoding LSTM was replaced with a CNN. The CNN filters capture signals from across the input sequence, which are then decoded using soft-attention and an LSTM. The CNN variant was faster to train and achieved competitive performance with LSTM variants.

Language translation necessitates computing the probabilities of words over at least two large vocabularies. This can be very expensive and led to the development of techniques to represent words in more efficient ways, such as Byte Pair Encoding (BPE) [201]. Byte pair encoding is a data compression algorithm, it works by splitting large vocabularies into a fixed set of sub-words - as opposed to learning a representation for every overlapping sub-word as in FastText [27]. To generate the required sub-word set size, the algorithm iterates through a vocabulary, merging the most frequently occurring sequences of characters from within words, until the sub-word vocabulary size is met. This algorithm reduces vocabulary size while allowing words from outside the vocabulary to be represented by the sub-words that

it contains. Experiments incorporating BPE into Bahdanau *et al.*'s achitecture [14] showed that it improves translation performance, particularly in rare word scenarios [201].

Attention Around the time of the work by Bahdanau *et al.* [14], other researchers had started to experiment with models that selectively attend to parts of a sequence or image rather than being limited to remembering information between steps [73, 154] in what became called attention models.

Attention mechanisms were designed to emulate human attention. If a human were classifying a passage of text they would pay more attention to parts of the text that provide relatively more relevant information. For example, to decide whether the phrase "the husky loved going for long walks" referred to dogs or not, the words "the" and "loved" provide less relevant information than "husky" and "walks". Additionally, the meaning of a word can change depending upon its context. The word "husky" is polysemous, being both a breed of dog, an adjective to describe a voice, and to describe a person's build. In neural networks, attention mechanisms are specific trainable weights, that learn to augment the model by increasing the signal of task-relevant features, while diminishing those of less relevant features [221].

Graves [73] developed a model that selectively attends to different areas of a sequence to generate handwriting by modelling sequences of pen positions on a plane of two-dimensions. However, in this case it can be described as selectively attending to all possible areas of output space rather than across inputs. To do this it captures previous pen coordinates from a sequence using an RNN and then samples the output probabilities over a mixture of multiple two-dimensional Gaussians. The parameters for each Gaussian are created dynamically using the outputs of the RNN. Specifically, the hidden layers of the RNN are passed through a series of linear transformations to produce a set of parameters for each mixture of Gaussians: a coefficient to weight it relative importance, its mean and standard deviations in both x and y dimensions, and a correlation coefficient between these planes. These Gaussians are sampled to produce the next likely pen position. This process is repeated iteratively to generate sequences of handwritten characters. The model is trained by comparing the next true point with the predicted point, minimising the negative log-likelihood of the data given the Guassian mixture models. As a result the model learns to produce the correct Gaussian distributions to sample in order to generate sequences of human-like handwriting with subtle variations in form that one would expert from human writing.

Mnih *et al.* [154] introduced the Recurrent Attention Model (RAM) which uses an RNN to iterative take a fixed number of "glimpses" of an image. Following the first glimpse the model selectively attends to specific regions of the image, given the previous glimpse, in order to make a classification following the final glimpse. So rather than sliding a series of convolutional filters over an image, the model learns to

focus on the most salient regions of an image in order to make a classification. This is regarded as a "hard" attention mechanism as the model does not attend to to the whole of the feature space, but instead selects a subset. As it is not deterministic it cannot be trained directly using differentiation and must use a reinforcement learning procedure. This method outperformed CNNs at digit classification tasks containing cluttered images and could perform tasks in dynamic environments, such as object tracking in videos [154].

Chorowski *et al.* [42] applied a soft-attention mechanism to a speech recognition architecture. The authors recognised that speech recognition is a sequence-to-sequence generation task much like translation. However, they also noted that speech recognition is more challenging due to the larger scale of the data and its inherent noise. The model they proposed consisted of a CNN to encode the audio data into feature vectors, an LSTM with attention mechanism to aggregate relevant information between the vectors, and a final layer to transform the aggregated vectors into word pieces or phonemes.

Insights derived from their experiments included that the model required a period of pre-training, without the attention mechanism, for some time prior to introducing it to ensure improved performance [42]. Additionally the model had a tendency of skipping regions of the input data, to mitigate this the authors encoded location-based features into the data. This was found to help the attention model properly attend to all features across the input. The authors also manipulated the attention scores with a temperature parameter, with a higher temperature softening the scores by scaling them so that the relative difference between the highest and lowest scores are reduced. Overall, the model outperformed previous speech-to-text algorithms, without requiring the development of hand-crafted features.

Building on the work by Bahdanau *et al.* [14] using soft attention for machine translation, Luong *et al.* [137] introduced global and local attention mechanisms. With global attention the mechanism attends to all features across a sequence, aggregating the signals from the whole sequence. Luong and colleagues argue that this may result in the attention mechanism capturing a lot of noise. Particularly so if the sequence is long, as words tend to become less relevant to each other the further apart from one another they are in a sequence.

To mitigate this, Luong *et al.* [137] introduce local attention. Reminiscent of the hard attention of the RAM [154], a local attention mechanism only attends to a subset of the source sequence located around an alignment point. This alignment point $p_t \in [0, S]$, within an S-length source sequence for target step t, is dynamically predicted by the model as follows:

$$p_t = S \cdot \sigma(\mathbf{v}_p^\top \tanh(\mathbf{W}_p \cdot \mathbf{h}_t)) \tag{2.56}$$

Where $\mathbf{h}_t$ is the hidden state of the LSTM for the target at step t, and σ repres-

ents the sigmoid activation. $\mathbf{W}_p$ and $\mathbf{v}_p$ are trainable parameters which learn to predict the optimal alignment point. This point is used as the mean of a Gaussian distribution over the source sequence. While the global attention scores are calculated across the sequence, they are then scaled by the Gaussian centred on p_t with either a pre-defined or learned variance. This means that source representations in close proximity to the alignment point are relatively more influential when the model aggregates the hidden source representations into a context vector $\mathbf{c}_t$, with distant source representations having a negligible effect. This effectively creates a context window around the point p_t, which reduces the noise from words not in close proximity. The reduction in noise and scale improves the discriminatory effect of the softmax which makes the model more effective at handling long sequences [137].

The authors also explored different ways to compute attention scores between the target hidden representation and the source hidden representations, either through the direct computation of the dot product between hidden representations or with intermediate transformations by trainable parameters. They found that attention incorporating intermediate transformations outperformed those using direct dot-products of the hidden representations [137].

Hu *et al.* [94] developed the squeeze-and-excite attention mechanism. Here an attention mechanism was used to augment a CNN architecture, emphasising or diminishing the maximum feature signals by modelling dependencies between the average signals of the convolutional filters. This allows the model to capture a greater variability from its feature maps, improving performance, while only increasing the computational overhead by a negligible amount.

Transformers In 2017, Vaswani *et al.* [221] introduced the Transformer. The Transformer is a sequence-to-sequence model, initially designed for machine translation. Unlike the previously mentioned attention architectures, this model is entirely based on attention mechanisms and does not use attention to augment another architecture. The transformer model consists of multiple layers of encoder and decoder blocks. The key component of both encoder and decoder blocks is a soft attention mechanism, called a multi-headed self-attention (MHSA) layer.

MHSA explicitly models the semantic dependencies between steps in a sequence, in order to compute updated representations of each step. For instance, when applied to a sequence of word embeddings, it learns to capture the composite semantics of words that interact within a sequence, and then uses these composite semantics to update the representations of the words. This is particularly useful for modelling contextual polysemy in language. For example, the phrase “kick the bucket” represents very different semantics to the words “kick”, “the” and “bucket” in isolation. A transformer modifies the embeddings of “kick” and “bucket” to more readily represent the composite semantics. Multi-headed attention is comprised of multiple “heads”. Each head has a fraction of weights proportional to the number of heads.

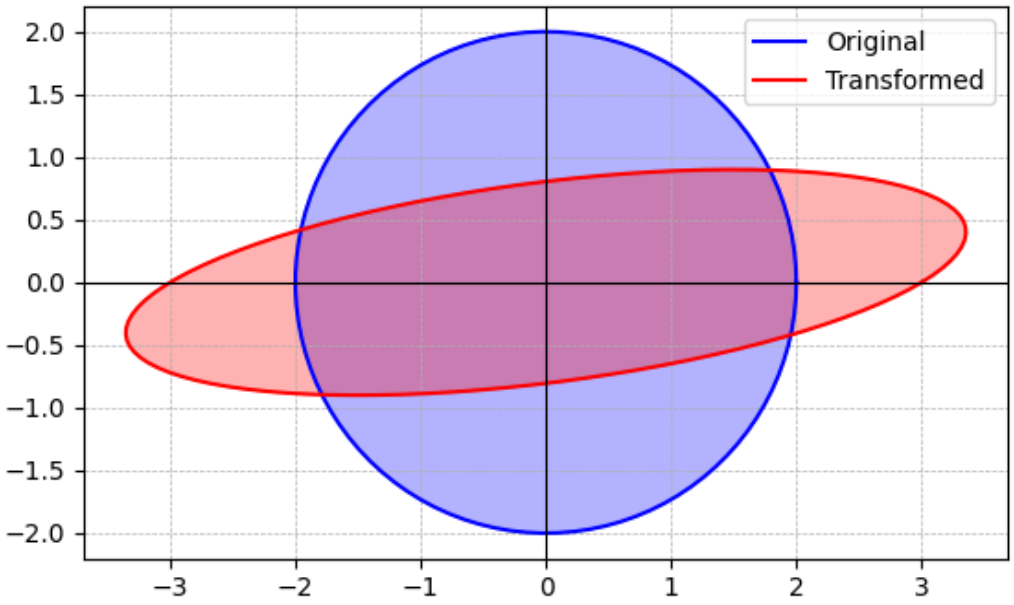

Figure 2.8: An illustrative example of an affine transformation in two-dimensional space of the original shape (blue) into the transformed shape (red). Differences between areas along the x-axis are accentuated, while those along the y-axis are compressed. The transformation has also induced skewness in the shape leading to a change in orientation. This change in orientation can be understood as a transfer of some of the variance between dimensions.

Different sets of attention weights attend to different representational subspaces and sequence positions [221]. The intuition here is that the heads learn to pay attention to different task relevant signals to one another. This mechanism is an integral part of state-of-the-art transformer models [221]. However, Schlag *et al.* [194] disputed the stated intuition that different attention heads attend to different areas of representational space. Instead they argued that each head performed an affine transformation on the entire representational space of the input, preserving all the input's information as opposed to just a subset. For an illustration of an affine transformation please see figure 2.8.

To understand how the transformer works in practice the model must be described. In the encoder blocks the attention mechanism aggregates information between each vector in a sequence. Given a matrix representing a sequence of vectors, $\mathbf{X} \in \mathbb{R}^{n \times d}$, each attention head within the MHSA layer can be defined as

follows [221]:

$$\mathbf{Q} = \mathbf{X} \cdot \mathbf{W}_Q \tag{2.57}$$

$$\mathbf{K} = \mathbf{X} \cdot \mathbf{W}_K \tag{2.58}$$

$$\mathbf{V} = \mathbf{X} \cdot \mathbf{W}_V \tag{2.59}$$

$$\mathbf{Z} = \text{softmax}\left(\frac{\mathbf{Q} \cdot \mathbf{K}^\top}{\sqrt{\frac{d}{h}}}\right) \cdot \mathbf{V} \tag{2.60}$$

Where d is the dimensionality of the input vectors, n the length of the input sequence. The matrices $\mathbf{W}_{Q|K|V} \in \mathbb{R}^{d \times \frac{d}{h}}$, where h denotes the number of attention heads, are trainable parameters. They transform the input sequence, $\mathbf{X}$, into the Query ($\mathbf{Q}$), Key ($\mathbf{K}$), and Value ($\mathbf{V}$) representations, respectively.

The dot product of $\mathbf{Q}$ and the transpose of $\mathbf{K}$ captures the relative task-specific importance of every position with respect to each position in the sequence $\mathbf{X}$. This dot product is scaled by $\frac{d}{h}$ to stabilise the gradients, which is important if the dot product produces large values. These scaled values are turned into attention scores by passing them through a softmax function. These attention scores are used to aggregate a weighted sum of the $\mathbf{V}$ values. The aggregated output $\mathbf{Z}$ captures task-specific information deriving from interactions between vectors from the original sequence $\mathbf{X}$.

Each head has its own parameter weights and produces a different aggregate of $\mathbf{X}$. Each head's aggregate is concatenated together and then linearly transformed by a parameter matrix $\mathbf{W}_0$. The linear transformation serves to rotate or scale the combined aggregate signals so that they update to conform to the representation space of the inputs. This linear transformation is followed by a residual connection, which explicitly encourages the attention heads to learn useful aggregate information without also having to capture the identity function. Residual connections are integral to training deeper networks, as found by He *et al.* [85] and discussed previously.

$$\hat{\mathbf{X}} = \text{concat}\left(\mathbf{Z}_0, \ \ldots\ , \ \mathbf{Z}_h\right) \cdot \mathbf{W}_0 + \mathbf{X} \tag{2.61}$$

The residual connection is followed in turn by a feed-forward network, consisting of a non-linear transformation followed by a linear transformation. After the feed-forward network there is another residual connection.

$$\hat{\mathbf{Z}} = \mathbf{W}_2 \cdot \text{ReLU}(\mathbf{W}_1 \cdot \hat{\mathbf{X}} + \mathbf{b}_1) + \mathbf{b}_2 + \hat{\mathbf{X}} \tag{2.62}$$

Where each $\mathbf{W}$ and $\mathbf{b}$ are sets of trainable parameters. While the attention mechanism captures fine-grained dependencies between positions in a sequence, it is predominantly a linear process. The ReLU activation in the feed-forward network

introduces non-linear transformations to the information aggregated through these dependencies. This non-linearity allows the model to capture complex functions. The feed-forward transforms each row of the matrix identically, so therefore learns transformations that are beneficial across all positions in a sequence. Again, the residual function ensures that the feed-forward network explicitly learns useful transformations and not the identity function.

The decoder blocks are similar to the encoder blocks but contain an additional layer. The first layer is a self attention layer, as in the encoder, but this is followed by a cross-attention layer. Here the **K** and **V** are transformations of the previous encoder's output, while the **Q** value is a transformation of the outputs of the decoders self attention layer. The cross-attention layer is intended to capture dependencies between the aggregated signal of the input and output sequences. The output of the cross-attention layer is passed to a feed-forward network. Multiple pairs of encoder decoder block pairs can be stacked on top of each other. Finally the last decoder outputs pass to a classification layer.

To train a transformer for language translation, the inputs to the encoder would be a sequence of embeddings from one language, and the decoder inputs would be the semantically equivalent sequences of text from the target language. Although the transformer was intended for sequence-to-sequence tasks it has no inherent ability to capture the order of sequences. Instead the inputs have to be encoded with features which indicate the position of each part of the sequence, prior to being passed to first the encoder-decoder layer. This encoding can be either relative or absolute. During training the decoder would be iteratively masked so that each position in the target sequence can only attend to positions up to and including its own position. The outputs of each position of the final decoder layer are passed to the classification layer, usually a linear transformation followed by a softmax, to predict the true next word in the sequence. At deployment the model takes text to be translated as an input, and the output inputs are initially a special start token. The model then iteratively generates the next word in the sequence given the input plus the previously generated output word.

Since their introduction in 2017, models based upon the original transformer have become the state-of-the-art for many natural language processing (NLP) and computer vision tasks, from machine translation [221] and text classification [5] to object detection [38]. Transformer-based applications have found success in many domains outside of computer science, including drug-target classification [218] and mapping gene networks in single cell expression data [138]. Thafar *et al.* [218] exploited a pre-trained transformer-based amino acid sequence model combined with gene expression data to predict novel cancer drug targets for seven cancer types. Embeddings representing protein structure were created along with gene expression statistics for the corresponding gene per cancer type. These embeddings were used as inputs to train a deep neural classifier which out performed previous methods

for predicting the efficacy of drugs for five out of seven cancer types. The use of the amino acid sequence embeddings explicitly represent structure, which in turn implicitly represents protein function, allowing researchers to sidestep the scarcity of protein interaction profile data. Ma *et al.* [138] used a transformer to learn aggregated representations of cells and their genes from gene expression data. These representations were found to improve the clustering of single cell data, and aid in the construction of biological networks capturing gene-gene and cell-cell interactions.

Despite their success, transformer models have noted limitations: they require large volumes of training data to be effective [78]. Guo *et al.* [78] argues that this is because self-attention models have a poor inductive bias, and instead rely heavily on these large volumes in order to generalise well. This has implications for models that are trained on limited datasets, such as the text provided by an ontology, rather than a large corpus containing millions of examples.

2.1.4 Large Language Models

Following the success of the transformer, many models incorporating its architecture were developed [179, 56, 131]. The Generative Pre-trained Transformer (GPT-1), developed by OpenAI, is one such model [179]. GPT-1 consists of 12 layers of transformer decoder blocks, with a total of 117 million parameters. It used BPE [201] to tokenise sequences, which all start with a special start-of-sequence token, end with a special end-of-sequence token, and with multiple sequences in an input separated with a special delimiter token [179]. Each token in an input sequence is represented by a semantic embedding summed with a positional embedding which indicates the absolute position of the token in the sequence.

Like the transformer, GPT-1 is trained to predict the next token in a sequence given the inputs and the previous model outputs. However, this generative next token prediction task is used as a general pre-training stage prior to task specific fine-tuning. For all downstream tasks the hidden representation of the special end-of-sequence token in the final layer is passed to a classification layer. GPT-1 achieved state-of-the-art or competitive results across many natural language processing tasks.

Two further transformer-based models, Bidirectional Encoder Representations from Transformers (BERT) [56] were introduced, doing away with the decoder block. One, called $\text{BERT}_{\text{BASE}}$, consisted of 12 layers of stacked transformer encoder blocks with 12 attention heads each, and 110 million trainable parameters in total. The other, called $\text{BERT}_{\text{LARGE}}$, had 24 encoder layers with 16 attention heads, and 340 million parameters in total. The models were pre-trained using two distinct training objectives, followed by task-specific fine-tuning. All input sequences begin with a special "[CLS]" token, and with each element in the sequence being represented in the model by the element-wise sum of its token embedding, its segment embedding,

which indicates which sentence the token belongs to, and its position embedding, which represents the absolute position index of a token within the sequence.

The primary training objective was to predict masked tokens. This differs from the masked next token prediction task used to train the generative transformer for sequence-to-sequence translation in Vaswani *et al.* [221]. The reasoning here is that directional masking would not allow the model to learn deep bidirectional contextual representations, either limiting attention to one direction or to relying on "shallow concatenation" [56]. Accordingly, during training 15% of sequence tokens were randomly masked by replacing them with a special "[MASK]" token. As this special token would not be seen outside of training, 10% of these mask tokens were replaced with random tokens from the word-piece vocabulary, and a further 10% would be left unchanged. The model would then be trained to predict the correct tokens.

The secondary training objective was next sentence prediction. Here, the model would classify whether one input sentence is followed by another input sentence, separated by a special "[SEP]" token. To do this, the special "[CLS]" token embedding would be extracted from the model outputs and passed to a binary classification layer. The intention here is that the model would learn long range dependencies useful for question-answering tasks [56].

After pre-training the model can then be fine-tuned for various downstream tasks, with token-level representations being used for tasks such as sequence tagging, and the special "[CLS]" embedding can represent the entire sequence for classification tasks. BERT was evaluated on a number of downstream tasks including question answering (where the model has to extract an answer from a passage given a question), sentiment classification, and paraphrase identification. BERT achieved state-of-the-art results across the tasks it was benchmarked against [56].

BERT adopted a Gaussian Error Linear Unit (GELU) activation function to replace the activation functions used in the original transformer [56]. Introduced by Hendryks and Gimpel [88], GELU is inspired by the ReLU activation but intended to weight activation outputs by their magnitude rather than just their sign. It can also mitigate the risk of having dead neurons, which can occur with the ReLU activation [140, 15]. GELU was found to improve transformer performance [56, 15]. Figure 2.9 is a visualisation of both the activation function and its derivative, for values of between -6 and 6.

Large transformer models like BERT can be difficult to train effectively [131]. Liu *et al.* [131] explored how design choices and parameter selection influence the performance of BERT models. Their experiments demonstrated that model training stability, and thus performance, is improved by increasing the scale of training. Such measures as increasing the batch size during training, increasing the size of the training sequences, and extending the duration of training, all contributed to improved performance. Additionally, the authors found that removing the next-

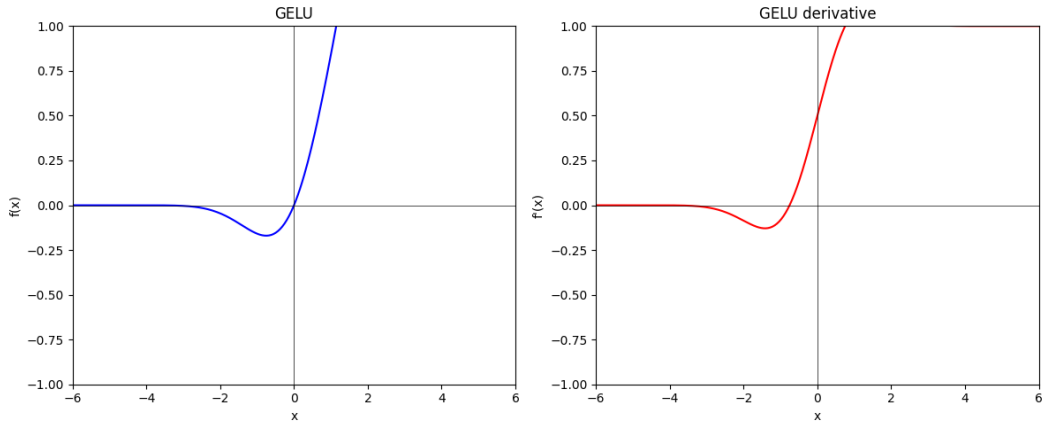

Figure 2.9: The GELU activation function and its derivative.

sentence prediction task from training also improved performance.

GPT-1 and BERT ushered in a new paradigm of large language models [29], involving large parameter models trained using large-scale general pre-training regimes followed by more accessible task-specific fine-tuning. Since then, transformer-based large language model architectures have been applied to domains outside of Natural Language Processing. For example, BERT has been fine-tuned for predicting drug-target interactions [106]. BERT has also been pre-trained on predicting the next protein in a protein sequence and predicting Gene Ontology (GO) function annotations to create a large contextualised model of proteins and protein sequences for downstream tasks [34]. Lee *et al.* [124], took a pre-trained BERT architecture and continued to pre-train it using a corpus of biomedical texts to create BioBERT. This model exceeded the vanilla BERT model at identifying disease, chemical, gene, protein, and species names in text.

Recently, Meta [219] released a suite of open source large language models called Llama 2. These models use the transformer architecture from Vaswani *et al.* [221] but have between 7 billion to 70 billion parameters [219]. The Llama 2 models are pre-trained using a self-supervised masked-token objective like the original transformer, but on a massive text corpus of 2 trillion tokens, using 2000 80Gb NVIDIA A100 GPUs. The authors evaluated the model at the end of pre-training on coding tasks (where the model would be asked to generate code with a specific functionality), commonsense reasoning, world knowledge, reading comprehension, and mathematics tasks, among other Artificial General Intelligence (AGI) benchmarks. The 70 billion parameter Llama 2 achieved near-human performance at these tasks and achieved the best open-source large language model performance to date. Additionally, the authors reveal that after the pre-training, having seen all the 2 trillion tokens, the losses of each Llama 2 model had yet to saturate, which suggests that they may be capable of even better performance given more training. A system of reinforcement learning from human feedback (RLHF) was then used to incorporate

Llama 2 into a chat-bot assistant program.

In 2017, Lake *et al.* [115] noted that previously language modelling has not been considered of high importance by the AGI community. While today large language models are central to contemporary discussion about AI [253, 36]. The latest closed source models, trained using volumes of data and with a number of parameters without precedent, such as OpenAI's GPT-4 [165], have achieved expert-human level performance on professional and academic exams, such as: code generation and code understanding [36, 165], the bar exam [107, 165], university entrance exams (SAT and GREs [165], and Brazilian examinations [164]), and university exams ranging from chemistry to statistics [165]. Additionally, OpenAI's technical report for GPT-4 [165] made the observation that large language model performance scales predictably with compute.

Closed-source models have been applied in Bio-Medical contexts. Liu *et al.* [130] adapted the chat-bot version of the closed-source 175 billion parameter GPT-3.5 model developed by OpenAI, ChatGTP, to the task of clinical decision support. They experimented by asking a panel of medical experts to score a selection of clinical alert suggestions, generated by ChatGPT and clinical experts. The AI generated alerts achieved scores comparable to those made by the human medical experts.

There have also been efforts to create open-source domain specific large language models for biomedical applications [206, 237]. Sin *et al.* [206] created BioMegatron, a domain-specific model for bio-medical natural language processing tasks, developed in partnership with NVIIDIA. It achieved state-of-the art performance at question answering, relation extraction, and named entity recognition tasks involving disease and chemical term identification. Yang *et al.* [237] introduced GatorTron, a large language model for bio-medical applications, also developed in partnership with NVIDIA. Although the large version of this model only contained 8.9 billion parameters, it excelled at clinical concept and medical relation extraction, with the authors also finding that the larger the model, the more effective it is. Large language models, whether closed- or open-source require vast computational resources to train and run, and are prone to hallucination [165, 219]. Hallucination describes when a large language model generates factually incorrect outputs that have the form of an authoritative, correct output.

2.1.5 Graph Neural Nets

The first graph neural net Another important area of deep learning research is that of graph neural nets (GNNs). First introduced by Scarselli *et al.* [193], who observed that lots of tasks could be modelled using graphs, and that neural networks could be used to model graphs. The components of the GNN are vectors to represent each vertex state, a function for transferring task-specific information between nodes, and an output function. The model iteratively passes information from a

vertex's neighbouring vertices to the vertex, updating its state vector. The iterative information transfer function allows message passing between nodes in undirected or directed graphs, and even cyclical graphs. Once the aggregated vertex representations converge, the final representations can be passed to an output function for inference. The GNN was applied successfully to node classification, edge prediction, graph classification, web-page ranking, and even image classification. The authors note that, although other architectures may outperform the GNN at these tasks, the GNN is a more general architecture that can be applied to a wide variety of tasks.

Graph convolutional nets Kipf and Welling [110] adapted the CNN to be applied to graphs, introducing the Graph Convolutional Network (GCN). Their motivation was to create a more scalable neural net for semi-supervised learning on graph structured data. A convolutional layer creates an aggregate representation of a vertex using signals from the vertex and its immediately neighbouring vertices, using an adjacency matrix with self-connections $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, where $\mathbf{A}$ is the adjacency matrix and $\mathbf{I}$ is the identity matrix which connects a vertex to itself. The aggregate vertex feature vectors $\mathbf{Z}$ can computed from the original vertex features $\mathbf{X}$ as follows:

$$\mathbf{Z} = \sigma\left(\mathbf{D}^{-\frac{1}{2}}\hat{\mathbf{A}}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}\mathbf{W}\right) \tag{2.63}$$

Where $\mathbf{W}$ is a matrix of parameter weights learned during training, σ is some activation function, and $\mathbf{D}^{-\frac{1}{2}}$ is the inverse square-root of the degree matrix. The degree matrix represents the number of edges a vertex is connected to. Multiplying the adjacency matrix row- and column-wise by the inverse square-root of the degree matrix normalises message passing between vertices, preventing highly connected vertices from dominating the aggregation process. The aggregated representations can be passed through more graph convolutional layers or to a classification layer. The GCN can be trained on a partially labelled graph in a semi-supervised setting. The authors' experiments showed improvements over the state-of-the-art for node classification, which suggests that the model learns both useful vertex representations while also encoding graph structure [110].

Inductive graph neural nets Hamilton *et al.* [82] developed an inductive graph neural network, GraphSAGE, which can compute representations of a previously unseen vertex based upon aggregate features of the a fixed-size sample of the vertex's neighbourhood. The way GraphSAGE aggregates signals between vertices differs from that of the GCN or the GNN. The aggregation methods used are mean aggregation, LSTM aggregation, and a pooling aggregator. The mean aggregator takes the element-wise mean of every vertex vector from a sample of the neighbourhood of a vertex, and concatenates this mean representation with that of the vertex.

The concatenated vectors are passed to a non-linear transformation to compute the layers output representation. The LSTM aggregator processes a vertex's neighbourhood sample sequentially using an LSTM. The sequence of neighbouring vertices is created in a random order. This sequence's embeddings are passed to the LSTM, and the final hidden representation is taken to represent the neighbours. This final hidden representation is concatenated with the original vertex embedding, before being non-linearly transformed into the layer output for this vertex. The pooling aggregator applies a non-linear transformation to each of the vertex embeddings in a vertex's neighbourhood sample, and then takes the mean or the maximum value of each element across the transformed embeddings.

The different aggregators were evaluated for vertex classification on social media data and scientific paper citation data, where the initial representations of abstracts or posts were averages of their GLoVE [168] embeddings. The models were also evaluated on a protein role classification task using a combination of molecular features and GO labels. These experiments showed that the LSTM and pooling aggregators were the best performers. However, the LSTM is a more complex architecture so it took twice as long to train.

Another inductive graph neural net was proposed by Veličković *et al.* [222], the Graph Attention Network. As the name suggests, this method adapts a multi-headed self-attention architecture to a graph context. Specifically, attention scores are calculated between connected vertices to weight the task-specific importance of edges, and use these weighted edges to update vertex representations. Due to the multiple attention heads the model can learn multiple edge weight parameters for the same relationship, allowing for more nuanced feature aggregation. This model was evaluated against GraphSAGE for a number of tasks, achieving a superior performance. The authors note that implicitly learning edge importance with a graph lends itself to greater interpretability than other types of graph neural nets.

2.1.6 Autoencoders

Autoencoders typically transform input features into some latent representation space and then transform them back into the input space, and are trained to reconstruct the inputs so that the latent representations capture essential features [89, 223, 109]. The soft-attention encoder-decoder machine translation models by Bahdanau *et al.* [14] and by Gehring *et al.* [68], discussed previously, are similar to autoencoders. Here, I will describe the development and function of autoencoders in more detail.

Hinton and Salakhutdinov [89] showed how to effectively train autoencoders using backpropagation, and demonstrated that they are powerful tools for compressing high-dimensional data into lower-dimensional spaces. Autoencoders use an encoder to compress high-dimensional data into a lower-dimensional space, before recon-

structing the high-dimensional data from the compressed data using a decoder. The encoders and decoders can be composed of multiple layers of transformations, iteratively reducing, or increasing, the dimensions of the representational space. To train the autoencoders, the authors used a series of unsupervised individual-layer pre-training followed by a compression-reconstruction learning objective. Without the pre-training stage the autoencoders did not perform well.

Hinton and Salakhutdinov's experiments [89] demonstrated that autoencoders could more effectively reconstruct images of hand-written digits, and produced more informative visualisations, compared with Principal Component Analysis (PCA) [6]. Additionally, the experiments showed that text document retrieval using the low-dimensional representations learned by the autoencoder outperformed those using the representation created by LSA [53].

Vincent *et al.* [223] were the first to explore denoising as a training strategy for autoencoders. To learn to denoise the input data, autoencoders were trained to reconstruct an an image using a corrupted version of it as the input to the encoder. Here each layer of the encoder and decoder corresponding to the alternate dimension transformation are trained together, with the next encoder and decoder using the compressed layers as inputs. The corruption of the inputs enforces a form of sparsity on the data, which encourages the autoencoder to generalise to the most informative features of the training data, rather than memorising specific reconstruction patterns. The authors show that the autoencoders trained with the denoising strategy could learn local edge detection features and grating filters, which capture periodic patterns, capturing sophisticated patterns without the need for hand-engineered features. The authors argue that this suggests that the low-dimensional projections of data learn to capture the characteristic features of the original representations. In their experiments, autoencoders outperform other non-neural image reconstruction methods.

Kingma *et al.* [109] introduced the variational autoencoder (VAE), as the generative component of a wider generative-discriminative framework. Here an autoencoder is used to meaningfully capture the features of the training data in a low-dimensional latent space, and reconstruct, or generate, the inputs from this latent space. The VAE is also used to train a discriminative model, devised for semi-supervised image classification tasks. These models are trained simultaneously. The discriminative model learns to classify an image based upon its latent representation. The objective function is a combination of the reconstruction error and the loss of the classifier. The unsupervised VAE training can benefit from training on unlabelled data, while the supervised classifier training benefits from more refined latent representations that result from the unsupervised training. The authors showed that this approach leads to improved performance over other methods, with particular improvement for cases where labelled data is limited.

2.2 Semantic Technology and Ontologies

2.2.1 Realising the Semantic Web

Originating in philosophy, Ontologies started to become popular with computer scientists in the early 90s [77]. They sought to create ontologies as actionable symbolic representations for computer systems. The idea is that these would provide the conceptual framework for interoperability between machine agents, in a format interpretable by machines. This occurred during the period of rapid development and expansion of the World Wide Web which started in the 1990s.

By 2001, Tim Berners-Lee, James Hendler, and Ora Lassila laid out their vision for "The Semantic Web" in their homonymous paper [23]. In which, they described a world wide web that was described in such a manner as to make it navigable by machine agents, not just humans. The authors noted that while Extensible Markup Language (XML) had allowed people to create annotations to describe web pages, these descriptions have arbitrary structure and do not necessarily represent meaning. The key to machine navigability is providing a structure that can be reliably interpreted by machines, as a proxy for understanding meaning. The core of the semantic web is knowledge representation. The transformation of the information on the web into a machine actionable format has since been described as a "one of key challenges of computer science" [11].

To create a semantic web of machine interpretable and actionable meaning, the authors called for the use of ontologies in concert with the Resource Description Framework (RDF) [23]. In the context of knowledge representation, ontologies are structured machine interpretable taxonomies of a domain of knowledge which have formally defined relationships between the entities represented. These ontologies can be used as a framework for machine reasoning, enabling sophisticated querying. Introduced by Brickley *et al.* [35] and Lassila and Swick [118], RDF is a model for meaningful data exchange on the web [146]. Specific entities receive their own unique identifiers known as Unique Resource Identifiers (URIs) and with expressions comprised of subject-predicate-object triples it allows for the definition of machine interpretable statements about entities.

Berners-Lee *et al.* [23] argued that the main benefit of the semantic web will be the development of autonomous computer agents that can process machine readable content. The semantic web has the potential to enable what came to be called the internet of things [143]. They also envision that the semantic web will enable highly precise data integration and powerful knowledge management systems. Ontologies will allow the precise semantic descriptions required to combine heterogeneous data into new forms of knowledge. RDF is intended to allow these structures of knowledge to be queried at scale.

In order to create this semantic web, ontologies would need to be constructed to describe various knowledge domains, and to do this would require standards to

specify how ontologies should be constructed. In 2000, Hendler and McGuinness [87] built a markup language, on top of XML, RDF and RDF Schema, called the DARPA Agent Markup Language (DAML) which included an ontology language DAML-ONT. Influenced by knowledge representation frameworks, it extended the RDF schema to allow for the greater expression required to build an ontology, with the express aim of using these ontologies to enable autonomous machine agents on the web. It was also influenced by formal description logics. Classes with multiple inheritance and structured hierarchies can be described using DAML-ONT.

Separately, Fensel *et al.* [64] proposed OIL an Ontology Infrastructure Language, a description logic. Again, this was inspired by previous knowledge representation frameworks and adapted from RDF schema into a framework for describing classes, sub-classes, properties and sub-properties.

OIL was combined with DAML into DAML+OIL by McGuiness *et al.* [145]. The intention was to capture the best features from both: the description logics from OIL into an Ontology markup language. Eventually these developed into the Web Ontology Language (OWL) [146]. Comprised of three sub-languages with OWL-lite providing the framework for classification hierarchies, OWL-DL is a description logic version of OWL where all reasoning is guaranteed to be computable, and OWL-full which provides no guarantees but is a maximally expressive extension of RDF. OWL provides further vocabulary to describe properties and classes beyond those contained in RDF, to specify relationships between classes, describe the cardinality of relationships, and to characterise properties.

With these frameworks in place, domain ontologies and resources to curate domain knowledge for re-use could be developed [162]. One such repository is DBpedia, a large community project to create structured machine interpretable representations of the Wikipedia encyclopedia [11]. The wealth of structured knowledge captured by DBpedia has been leveraged in knowledge graphs for question answering applications [57] and for information retrieval.

The story of ImageNet [55] showcases the value of richly described data annotation. Released in 2009, ImageNet is a hierarchically structured database of labelled images, described by its authors as an "image ontology". The foundation of ImageNet is WordNet [153], a structured database of synonym sets connected to one another through various semantic relationship types, such as hyponyms, holonyms, antonyms, and entailment. WordNet was converted into a linked data resource by van Assem *et al.* [10]. To create ImageNet, for each of the 80 000 synonym sets in WordNet, between 500 to 1000 images were collated [55]. These images were retrieved using search-engines and then cleaned manually by curators, who would ensure that the image contains the subject of the synonym set. This dataset formed the basis of the "ImageNet Large Scale Visual Recognition Challenge", and is widely regarded as a major catalyst to the rapid advances in computer vision from 2010 onward [190, 84], in particular AlexNet [114] and ResNet [85].

2.2.2 Linked Data and the Life Sciences

The Gene Ontology Since the inception of the semantic web there have been initiatives to represent biological knowledge with structured, machine-interpretable linked data [9, 54, 187, 16, 205]. One of the first of these was the GO [9], which was developed in response to the perception that sequencing was outstripping the scale with which biologists were describing shared biological entities with concepts, and the lack of interoperability between genomic databases that this caused.

The GO is actually comprised of three separate ontologies: Biological Process, which describes the networks of gene products in different biological functions; Molecular function, which describes cellular activities at the scale of single molecules; and cellular component, which describes the structures within a cell which the molecules and biological processes take place in or in proximity to. These sub-ontologies allow for within- and cross-species comparisons of genes and gene products, and interoperability with other ontologies and linked data. Links between between concepts in the GO and gene products are called annotations [97]. The GO was initially use to describe the gene products for model organism databases, such as the *Saccharomyces* Genome Database, but by 2004 its use had expanded to all major genomic repositories [45].

In 2004, Boyle *et al.* [33] introduced GO::TermFinder, a method for annotating genes with functional information. Using a simulation, it computes the probability of a GO term being associated with a set of genes. Bonferroni correction is used to adjust the probabilities to control for false discovery. These probabilities are then used to determine the most likely annotations that indicate the biological and molecular functions, and cellular components of the gene set.

Hong *et al.* [93] expanded the *Saccharomyces* Genome Database to also allow the annotation of gene products with evidence from outside traditional individual gene function experimental studies. This reflected not only the expanding scale of genomic analyses and the rise of high throughput comparative sequence and genomic studies, but also the scalability and flexibility of the GO to accommodate such a change. By 2014, 99% of annotations made by the Uniprot GO Annotation Project were created automatically, this automation expands knowledge on the function of gene products from model species to less well studied species [97].

The GO is a dynamic and evolving ontology. The scientific knowledge represented by the ontology can be updated in light of new data and discoveries [97]. The hierarchical nature of the GO, and the constraint that all ancestor terms must apply to child terms, means that changes to the ontology can greatly influence the inferred annotations that are made [97].

Yon *et al.* [240] argued that the GO is often used incorrectly, with scientists treating different types of evidence equally when experimentally validated knowledge should take precedence. In a review of computational gene function prediction,

Padlidis and Gillis [167] noted that there was increasingly more integration between GO annotation and protein interaction networks. The authors were worried that, rather than generating new knowledge, scientists end up retrieving existing data. They argued that scientists should ensure that they are aware of the biases and drawbacks of the data they use.

Open Biological and Biomedical Ontologies (OBO) Foundry Along with the GO, other linked data resources were created in the life sciences [25]. These include: PubChem [28], a repository of experimentally confirmed small molecule functional information; Reactome [103], a curated linked data graph of biological pathways; and Chemical Entities of Biological Interest (ChEBI) [54], a repository of molecular entities and their attributes. Other data sources have been adapted to incorporate linked data. The Universal Protein Resource (UniProt) [47], a database of protein sequences and functions became a linked data resource. The PubMed search engine from the United States National Library of Medicine (NLM) has also adopted linked data [25].

By 2007, the use of ontologies in the life sciences had become so commonplace that the large number of different ontologies was actually posing problems for data integration, as opposed to alleviating them [211]. Smith *et al.* [211] cite the Unified Medical Language System (UMLS) as an example. The UMLS is collection of machine readable biomedical and clinical vocabularies. Smith *et al.* [211] argued that while the UMLS has been shown to be useful for information retrieval, the vocabularies it contains do not share a common architecture so different concepts cannot easily conform to a single system.

As a response to these observed issues, the Open Biological and Biomedical Ontologies (OBO) foundry was formed to coordinate ontology efforts [211]. The imitative aims to create and maintain a set of non-overlapping but inter-operable biomedical ontologies. These ontologies are to be written in a common shared format, and made freely available. The OBO foundry has since been responsible for the development and maintenance of many OBO ontologies, including: the development and curation of the Human Phenotype Ontology (HPO) [187], an ontology to describe all phenotypic abnormalities stemming from genetic mutations; the curation of the Cell Ontology [17], which hierarchically categorises cell types across prokaryote, fungi, animal, and plant kingdoms; and the development and curation of the Disease Ontology (DO) [198].

Other ontologies have been created using the concepts contained within others. For example, the Epidemiology Ontology [170] was constructed along OBO guidlines, reusing a combination of elements from the DO, Symptom Ontology, Vaccine Ontology, and the Pathogen Transmission Ontology, combined with novel concepts. The Ontology for Biomedical Investigations [16], reuses parts of the GO, ChEBI, and Phenotype Attribute and Trait Ontology (PATO).

Platforms, such as BioPortal [163], use the OBO ontologies as the backbone of the data retrieval and integration services. More recently, the Monarch Initiative [205] has sought to reconcile all species phenotype ontologies. By combining ontologies and annotated databases into an integrative platform for genotype-to-phenotype knowledge across species, they authors hope to bridge the gap between clinical and basic research, creating a "Monarch Knowledge Graph" - A structured heterogeneous-domain genotype-to-phenotype knowledge representation. Knowledge graphs are increasingly recognised as the most prevalent form of knowledge representation [132].

Applications of Knowledge Graphs and Ontologies in the Life Sciences Organising data and domain expertise into knowledge graphs can be leveraged to derive new insights [255, 157, 63, 207, 132]. Zhu *et al.* [255] explored using semantic technologies to identify candidates for breast cancer drug repositioning. They constructed linked data profiles of breast cancer drugs using information from the PharmaGKB (pharmacogenomics knowledge base). This knowledge base (or graph) contains structured clinical, genomic, and phenotype information curated from pharmacogenomics research. From this drug-drug, drug-gene, drug-SNP, gene-disease, disease-SNP, and gene-gene associations were extracted. Additionally, chemical structure similarity was computed between drugs - with those over a threshold linked with an "is structurally similar to" relationship. A meta-ontology was defined to describe the types of entity classes and their relationships. Then sets of axioms were used to identify candidate chemical compounds that are structurally similar to recognised breast cancer drugs. The authors stressed that while this work was preliminary, semantic technologies had shown promise for the purposes of drug repositioning. They predicted that greater volumes of semantically annotated data would lead to greater volumes of candidates.

Myneni *et al.* [157] devised an ontology-based framework for building interactive support apps for young cancer survivors. An application ontology was made to model knowledge regarding after care plans, capturing information including patient marital status, age, gender, treatment information, and the details of their care providers. This ontology was used as the basis for personalising content, with SPARQL queries used to extract answers for patients questions regarding their care. However the authors manually created the ontology, and note that there is not a particular taxonomy for consumer engagement let alone heath-specific engagement, but hope that advances in NLP could lead to scalable ontology generation.

Esteban-Gil *et al.* [63] also built an application on top of linked data. The authors argue that off-the-shelf cancer registry databases are not amenable to integrating with other data sources. In this work a simulated cancer registry representing 207 190 patients was generated. It was then transformed into RDF using OBO best practices, reusing the Semanticscience Integrated Ontology and the Ontology

for Biomedical Investigations. An application with a graphical user interface was constructed over the RDF registry. Again the authors note that one of the limitations to work like this is that it requires a more comprehensive ontology to capture greater granularity, which is labour intensive. This is made more difficult by the ever evolving scientific and legal frameworks around cancer research and treatment.

A review by Silva *et al.* [207] explored a plethora of knowledge graph applications in the realm of cancer biology. They note that ontologies provide terminology and structure that can form the skeleton of a knowledge graph, or provide the vocabulary for NLP data mining. The semantics encoded into ontologies allows for machine reasoning, which can be used for interfaces or for error detection and data validation.

2.3 Combining symbolic and neural architectures

Lake *et al.*'s highly influential paper [115] discussed advances in statistical (connectionist) AI and how they relate to symbolic AI in the context of building a human-like intelligence. The authors note that human learning and intelligence exhibits facets aligned with both points of view. They state that humans learn in a way that seems to align with the symbolic AI paradigm, but concede that while humans learning from analogy appears to be facilitated by symbolic representation, the statistical methods seem better suited to capture more abstracted and complex relationships from unstructured noisy data. However, the transfer of the rules captured by statistical machine learning cannot easily be transferred to the more easily intelligible format of symbolic representation [195].

Although not specifically defined by Lake and colleagues [115] as neuro-symbolic AI, the authors suggest that a deep integration, not a mere combination, of both statistical and symbolic AI, may provide a framework for developing machines that learn and reason like humans. A survey of recent neuro-symbolic and statistical relational learning AI, Deraedt *et al.* [180] posited that neuro-symbolic architectures tend to be more data-efficient, and can be used to learn numeric vector representations for symbolic representations. They also note that they can learn the structure of the symbolic systems and use this to their advantage.

Onto2Vec by Smaili *et al.* [209] and OPA2Vec by Smaili *et al.* [210] both model symbolic knowledge, in the form of ontologies, by learning dense vector representations. Onto2Vec trains a skip-gram word2vec model [148, 149] on a corpus of sentences constructed from the information encoded within an ontology (e.g. *Class rdfs:label Label*). OPA2Vec is an extension of Onto2Vec that also incorporates a word2vec model. However in this case it is pre-trained on a corpus of PubMed abstracts prior to further training on a corpus of sentences constructed from the natural language annotations, the inferred annotations, and the structural information contained in an ontology. Both of the embeddings produced by these methods were evaluated by using them as the basis for predicting protein-protein interactions,

with favourable results, particularly for the richer OPA2Vec embeddings.

Althubaiti *et al.* [4] applied OPA2Vec to learn embeddings for the entities in an integrated knowledge graph for use in predicting potential cancer drivers. The knowledge graph comprised experimental data from cell growth assays, functional data from model organisms, and several ontologies such as the Cellular Microscopy Phenotype Ontology and the Mammalian Phenotype Ontology. They built various iterations of this knowledge graph, with various combinations of ontologies and data. Model performance was significantly influenced by the makeup of the knowledge graph that was embedded, with the combined use of all of the data sources leading to the best performance. Consistent with previous research, their predicted cancer driver genes had higher somatic mutation rates, and were functionally related to known cancer drivers. The authors go on to argue that although incomplete, biomedical ontologies form a "comprehensive web" of domain knowledge, and are a rich resource to be exploited by machine learning algorithms.

There have also been other attempts to embed the vertices and edges of knowledge graphs using transformer-based language models. Zhang *et al.* [250] used a pre-trained BERT model to generate embeddings for vertices using their natural language descriptions. These embeddings were transformed into a smaller representational subspace and then used to train a typical knowledge embedding algorithm (subject + predicate − object ≈ 0) which imbues transformed embeddings with structural information from the knowledge graph. The authors demonstrated that this method leads to improved performance in low-resource settings and posit that this is due to the language model imbuing the knowledge graph embeddings with information from its "world model".

A similar approach was explored by Yao *et al.* [238], who instead represented knowledge graph subject-predicate-object triples as sequences of their natural language descriptions separated by special "[SEP]" tokens, and fine-tuned a BERT model with a binary classification task predicting triple validity. Again, the authors found improvements in low-resource scenarios.

Wang *et al.* [227] explored approaches for combining language modelling and knowledge graph embedding with a single transformer model. Masked language modelling was used to learn language while a series of knowledge embedding approaches were explored. The knowledge embedding methods all used the subject + predicate − object ≈ 0 framework, but either used vertex natural language descriptions as embeddings, vertex and edge descriptions as embeddings, or vertex embeddings conditioned on edges. This joint training objective allowed the model to learn to embed new entities into the graphs representational space even if they did not feature in the training set.

Explainable AI Deep learning models are commonly referred to as "black boxes" [208, 112]. A review by Novakovsky *et al.* [161] discusses the explainable AI and

its particular relevance for genomics. Here the authors argue that the distillation of mechanistic insight into the biological processes that neural models capture, will only become more necessary as the scale of biological data and the complexity of the relations they represent increases.

As of yet, most explainable AI approaches are *post hoc* [161], this means that an explanation is generated by examining the influence of features on model predictions after training. One of the most influential explainable AI methods is SHapley Additive exPlanations (SHAP) by Lundberg and Lee [135]. SHAP calculates feature importance, or SHAP values, in such a way that the importance it assigns to each feature can be characterised as the average marginal contribution of a feature over all possible combinations of features. For each prediction, the sum of the SHAP values assigned to all the features equals the difference between the model prediction and the average prediction across all instances. Each feature is assigned a SHAP score that indicates its relative contribution to a given prediction. An average of these SHAP values provides a global insight into feature importance, however the real utility of an explanation tool lies within its prediction-level explanations, which indicate the contribution of features for a single output [183]. Prediction-level importance measures help to derive insight into complex interactions between features, which may be missed on the global level.

SHAP is not without its criticisms however, as it assumes that features are independent of each other [191]. This means that the importance of features that vary co-linearly may not be correctly accounted for by the model, or that these features can cause instability in model explanations. Slack *et al.* [208] exploited these aspects to show that SHAP was susceptible to being fooled by synthetic features. These synthetic features, while not directly co-linear with or proxies for sensitive characteristics, were constructed to induce the effects of sensitive features on model predictions. When SHAP was applied to explain models trained using these synthetic features it attributed model predictions to these features rather than the true discriminatory ones. In general, *post hoc* model explanation approaches may be inconsistent and unreliable - and different explanation tools may even disagree substantially [112].

Although the previously mentioned neuro-symbolic methods leverage structured knowledge to improve model performance, they are not explainable nor interpretable. Conard *et al.* [44] define "interpretable" to mean that the parameters of a model correspond to functional concepts, and "explainable" to mean having parameters that can be used to account for the model predictions. Neuro-symbolic models do have the potential to form the basis of explainable/interpretable models and are slowly starting to find application in bioinformatics [32, 44]. This is largely due to their utility in creating explainable and interpretable systems [44].

DeepGONet, created by Bourgeais *et al.* [32], is a type of feed-forward network where each neuron represents a GO Biological Process (GO-BP) concept. The first

layer extracts gene activation levels from gene expression data and assigns them to specific biological functions. The concepts represented by the subsequent layers become more and more abstracted. Each of the neurons in one layer are fully connected with the following layer but the connections are constrained by a custom regularisation function. The regularisation function effectively penalises the optimiser for passing signals between concept-neurons that are not connected in the GO-BP. The strength of this regularisation is weighted by a scalar parameter. A ReLU activation function is applied following every hidden layer, this is to represent the presence or absence of a particular phenotype, and to model its significance.

This model was trained and validated using two different datasets, one for a binary cancer/not-cancer classification task, the other for a multi-class cancer-type classification task. The model achieved similar performance to other state-of-the-art un-explainable methods, while penalising connections not in the GO-BP and conserving signals along valid GO-BP connections.

The authors explored how the concept-neurons were contributing to predictions using Layerwise Relevance Propagation (LRP) and hierarchical clustering of the hidden layers. LRP showed that only a small number of layer neurons contribute to each layer output. It also revealed that, despite the custom regularisation function, some relationships between phenotypes not represented by the GO-BP contribute to classification. This reflects the incompleteness of the ontology, suggesting the presence of some yet to be described biological process. The clustering revealed that earlier neurons capture tissue specific signatures, with deep layers representing more general cancer signals. This suggests that the model had captured both tissue-specific and universal cancer features. The authors believe that tools such as DeepGONet can be utilised for personalised medicine.

Bourgeais and colleagues [31] also created GraphGONet. This time the model represents every concept in the GO with a neuron, however links between neurons are strictly constrained by the relationships present in the GO. Each neuron receives an input from its child neurons. If a neuron has no child neurons, then it receives an input of relevant gene expression data. The activations of each neuron in the graph are computed, progressing from the most specific to the most general. Once calculated, a proportion of the neurons with the highest activations are then used as inputs for the classification layer. A vector representing all the neurons is computed, with the elements corresponding to outside of the top activations being masked to zero. A softmax is then applied to the vector and then passed to the classification layer. As it trains the model learns to associate particular neuron activations with certain classes. This model achieves comparable performance to state of the art "black-box" models when compared on the same tasks as DeepGONet, while the subset of activated neurons can be used to explain the model predictions. The authors found that these interpretations are stable - other iterations of the model with different initial parameterisations learned similar explanations for the same

classes.

In a variation on this theme, Lotfollahi *et al.* [134] developed expiMap, an interpretable gene program mapper. It uses an unconstrained encoder to non-linearly transform the cell expression data into a latent representation of the cell. However, the decoder is linear and constrained by domain knowledge. A binary matrix representing the genes associated with specific gene programs is used to softly constrain the decoder. Whereas, with DeepGONet [32], a custom regularisation function penalises connections between genes and gene programs not represented by the symbolic domain knowledge, the binary matrix. This soft constraint results in expiMap learning a mapping of one gene program to each element of the latent variable, while also allowing it to capture associations outside of the domain knowledge. A Group-LASSO regression is used to select only the most informative gene programs for a cell atlas. Group-LASSO learns to select all of the variables in a group or none of them.

Lotfollahi and colleagues' [134] expiMap is trained on a reference dataset. To map further datasets to this reference a form of transfer learning is employed. Weights corresponding to the known gene products are frozen, but a number of new trainable latent variables are added to the model. These new latent variables learn new important features that are not present in the reference set, while also representing the new data in the same space as the reference set. The authors tested their approach by removing certain gene programs from the training data set and then seeing if the transfer learning could identify them in a new set of data. The model learned specialised gene programs corresponding to those removed, while also identifying gene-program-to-gene associations not defined in the binary matrix. A statistical measure of variable independence demonstrated the exclusivity of these new latent representations. The model only learns new pathways as necessary due to the regularisation constraint silencing surplus connections.

Pan *et al.* [166] explored the many ways that large language models could be combined with knowledge graphs. The idea being that the generalisability of generative large language models would be enhanced by the accuracy and explainability supplied by structured knowledge graphs, with the knowledge graph providing a guardrail.

2.4 Related work

2.4.1 Semantic table interpretation

To realise the semantic web, heterogeneous data needs to be incorporated into a linked data format [251, 216, 224, 49]. The task of transforming tabular data into linked data is called Semantic Table Annotation [251]. It involves assigning relevant semantic classes, from either a knowledge graph or an ontology, to the elements, rows, or columns of tabular data. This is typically done by matching columns to

known semantic model, which are subsets of the knowledge graph known to represent tabular data [251, 216]. These annotations allow the data to be integrated into a wider knowledge graph, and provide unambiguous machine-readable context which allows machine agents to reason on the contents of the table.

Taheriyan *et al.* [216] approached the task of semantic data annotation by mapping ontology terms to dataset attributes, and then using the properties of known semantic models to link the terms together. This mapping was represented as a "semantic network" or graph, where the ontology classes were the nodes, and the model relations the edges between them. Dataset attributes were mapped to candidate ontology terms using either text embeddings with vector similarity measures for textual attributes, or distribution analysis for numerical attributes. The known semantic networks were then overlaid upon the candidate nodes. The weights of the edges in common between semantic models were increased. Candidate semantic networks were then generated to represent the dataset by computing the minimal known networks that encompass the attributes. These candidate networks were then scored by a function of graph size and edge weightings, with more common patterns being preferred.

This approach was adapted by Vu *et al.* [224] to incorporate a probabilistic graphical model. Again, a weighted directed graph of known semantic models was used to represent the possible links between nodes. Then a "transition function" was used to generate all of the possible connecting paths between each data attribute and the leaves and root of the tree. All of the candidate paths were scored using a Conditional Random Field (CRF) which has been trained on the known semantic models.

Zhang [251] developed TableMiner$^+$, where subject column detection is followed by an iterative process which alternates between a learning stage and an update stage. In the learning stage, TableMiner$^+$ compares contents of table cells against the concepts in the knowledge graph, scoring the relevance of the concepts, and retaining those with the highest scores. Then during the update stage, a domain representation bag-of-words is created using the highest scoring concepts for each cell. Each of the concepts assigned to represent the columns are compared against the domain representations. To do this the definitions of each concept are turned into a bag-of-words which are used to compute the similarity of the definition with the domain representation. These similarity scores are used to determine whether a column concept is updated. If the column annotation is updated then the annotations for the individual cells are updated. This process is repeated until the concepts stabilise and no further updates are made.

Cremaschi *et al.* [49] created an automatic annotation tool, MantisTable, where a knowledge graph plays a role similar to the known semantic models in both Taheriyan *et al.* [216] and Vu *et al.* [224]. A knowledge graph is an abstraction used for integrating information from multiple sources or domains. Much like an ontology it

represents knowledge as a graph of nodes and edges, with the nodes again representing concepts and the edges relationships. In this case the knowledge graph provides the context required to infer the semantics of the dataset instead of a collection of known semantic models.

All of these models depend on how accurately dataset attributes can be mapped with the correct ontology terms. Which, due to polysemy, is a non-trivial problem [104]. As these methods use exact string matching approaches [216, 224, 251, 49], this may pose an acute challenge.

2.4.2 Named Entity Recognition and Biomedical Concept Recognition

Although a dataset may not contain enough information to disambiguate its semantics, the paper that accompanies it should. By reading a publication, a human-agent can identify concepts and infer semantics. Machine agents could be used to identify specific concepts and relationships from text, and construct a machine readable graph. This graph can function as a proxy of the text's semantics.

The first step in transforming unstructured text into a meaningful graph is to recognise the relevant concepts it contains. Extracting concepts from unstructured text is a natural language processing task called Named Entity Recognition (NER) [24]. NER is a classification task where words or phrases corresponding to entities or concepts of interest are identified within text. State-of-the-art models can achieve F1 scores in excess of 90% on standard NER datasets. However, this level of performance depends on the availability of high quality annotated data which can be prohibitively expensive to produce [24] - particularly if expert annotation is required [252].

Bikel *et al.* [24] introduced one of the first Named Entity Recognition models, IdentiFinder. The model started by extracting handcrafted word features. These features included letter capitalisation, whether the word was the first in a sentence and if the word was a specific numerical pattern such as a date. Another feature extracted represented whether two words occurred frequently together, such as "New" and "York". A Hidden Markov Model (HMM) was trained to maximise the joint probabilities of the correct sequence of labels given an input sequence of text and their extracted features.

Bender *et al.* [19] also devised an approach to NER that incorporated dependencies between words in sentences, this time using a Maximum Entropy model. Again, the presence or absence of certain word features were used as the inputs to the model. The model learned to predict labels for a word given its features and those within a fixed context window of the word. Both this work and IdentiFinder demonstrated the importance of word structure and context to NER.

McCallum and Li [144] applied a Conditional Random Field (CRF) model to

NER, as an alternate approach to probabilistic models such as HMMs and Maximum Entropy. The CRF layer learns the conditional probabilities of all the sequence of labels given the entire sequence of input words. It takes into account both the individual labels for words, and the transition relationships between labels of consecutive words (For example the label "surname" following "firstname"). They also devised a method for the automatic induction of features by combining multiple handcrafted features. This is an iterative process when candidate feature combinations are used to train a model on a subset of the training data. If the new features increase the log probability of the correct labels the new feature is retained. These feature combinations were found to lead to better model performance than including the total set of around one million possible features.

Lample *et al.* [116] combined a Bi-LSTM with Conditional Random Fields (Bi-LSTM-CRF) for NER. The model considers both word morphology and semantics. It does this by applying a character-level CNN to the input words and concatenating its output with the word's embedding from a pre-trained word2vec model. The concatenated vector is passed to the Bi-LSTM which captures contextual information between words in the sequence. The Bi-LSTM outputs are passed through a linear transformation, reducing the dimensionality to that of the number of classes. Finally these are passed to the CRF layer, which behaves as in McCallum and Li [144], assigning join probabilities given the individual class probabilities. Due to the character-level CNN, the Bi-LSTM-CRF model does not require hand-crafted features. However, it still requires a corpus, manually annotated with labels of interest.

Yan and Wong [236] created a text classifier to identify cancer hallmark mentions in biomedicine literature for the purposes of annotation. Firstly text was preprocessed into lemmatised bags of words, noun bi-grams, named entities, grammatical relations, and verb classes. It used ABNER [202] to detect the presence of gene, protein, and cell type entities in text. Then MetaMap [8] was used to detect Unified Medical Language System (UMLS) entity terms. Synonymous entities were combined using the related concept relationship terms from the Medical Subject Headings (MeSH) structured vocabulary. Each feature was then represented with a binary presence-absence matrix. Around 200 000 features were crafted, while the average abstract contained between only 250 to 450. To reduce the dimensionality of the data various feature scoring mythologies were used, including a score based on decision tree feature importance measures. These scores were used to sort the features, with the top h features for each class being selected, followed by selecting the top k of this set.

A random forest classifier was then trained on the selected binary features, alongside other classical machine learning methods. The random forest performed best of all, with feature selection greatly influencing performance. The decision tree influence scoring outperformed all other feature selection methods, by excluding the

most frequent and infrequent terms, leaving the most informative. This method relied on string matching and on heavy feature selection.

Other methods have been designed specifically to extract ontology terms, typically in biomedical contexts, these are typically used to annotate electronic medical records [104]. One of the earliest biomedical semantic annotators was MetaMap [8], a tool for identifying UMLS terms in unstructured text. Firstly, MetaMap tokenises and then normalises the input text by removing stop words and lemmatising and stemming words. Variants of tokens are generated by looking up synonyms of tokens. Sets of tokens are used to look up UMLS terms using fuzzy string matching. Each match is then scored based upon each input phrase's similarity with the matched UMLS term. A disambiguation process resolves any exactly overlapping terms. The user can define the rules and parameters of the term disambiguation.

Clinical Text Analysis and Knowledge Extraction System (cTAKES) [192] is a clinical information extraction system that leverages both machine learning and rule-based models. The model works by first splitting text into sentences, and then tokenising each sentence based upon rules based upon token types. These tokens are then normalised (lemmatised or stemmed) prior to a machine learning-based parser determining phrase chunks. Then a string-matching UMLS term dictionary being applied to these phrases to look-up specific concept matches. Heuristic rules are used to detect negations in the vicinity of identified terms and to deal with overlapping matches. cTAKES returns the identified UMLS terms and can process both plain text and XML format files. Both MetaMap [8] and cTAKES [192] are designed for identifying UMLS terms.

Whetzel *et al.* [229] introduced the Annotator Web Service as part of the National Center for Biomedical Ontology (NCBO) web services. The annotator works by tokenising text and then using exact string matching to match words to the names of ontology concepts and their synonyms. Once a positive match is made the section of text is annotated with the matching concept. This process may be followed by further steps to disambiguate terms - in case of multiple matches for the same span of text. Another web service tool, Whatizit [181], which provides separate modules for the semantic annotation of terms from SwissProt (the manually reviewed part of UniProt), ChEBI, MedlinePlus (disease names), UMLS, DrugBank, NCBI taxonomy (species names), and GO. The GO terms are annotated using an exact-matching process that also matches word stem variations of words. It also has functionality for the identification of relations between terms.

NOBLE Coder [220] is a string-matching-based semantic annotator. It can match concepts from any given vocabulary, which means that it can be supplied with user defined terms. It uses a greedy strategy to capture all matches in a supplied text, followed by heuristics to disambiguate.

Neji [37] is a biomedical annotator that uses a system of modules: those for reading in text, modules for tokenising and normalising the inputs, and those for

performing concept recognition. The concept recognition modules may be a dictionary string-matching approach or a machine learning using a CRF system. Following concept recognition, heuristics are used to resolve overlap. Here longer spans are favoured, while overlapping matches from different dictionaries are allowed. There is also some functionality for abbreviation resolution.

2.5 Evaluation metrics

Throughout this thesis the models developed are assessed against benchmark datasets using measures that indicate the types of errors in the model predictions. A model's predictions can be classified as being either true positives, false positives, true negatives, or false negatives. The true positives represent the correct positive predictions, the false positives represent the incorrect positive predictions, while the true negatives and the false negatives correspond to the correct negative and the incorrect negative predictions respectively.

Accuracy Accuracy is a metric that captures both the negative and positive classification rates [175]. It is given as follows:

$$\text{Accuracy} = \frac{\text{true positives} + \text{true negatives}}{\text{true positives} \ + \ \text{false positives} \ + \ \text{true negatives} \ + \ \text{false negatives}} \tag{2.64}$$

Precision Precision represents the proportion of true positive predictions in the total number of predictions [175]. It can be computed as follows:

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} \ + \ \text{false positives}} \tag{2.65}$$

Recall Recall, sometimes referred to as the sensitivity or true positive rate, represents the proportion of the true positives predicted in the total number of actual positives [175]. It is calculated like so:

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} \ + \ \text{false negatives}} \tag{2.66}$$

F_1-score The F_1-score, or f-score, is the harmonic mean of both precision and recall [175]. It can be calculated as follows:

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.67}$$

$$F_1 = \frac{2 \times \text{true positives}}{2 \times \text{true positives} \ + \ \text{false positives} \ + \ \text{false negatives}} \tag{2.68}$$

It is a composite of both metrics but the harmonic mean tends towards the lower of the two, so that it more strongly reflects any trade-off between them than an arithmetic mean does.

Precision, recall and f-score do not capture any information about the true negative rate [175]. However, as in the case of semantic annotation there are potentially thousands of true negative classes, we are much more concerned with the rates of positive predictions rather than the true negatives.

Chapter 3

Heuristic Training Data Creation for Distantly Supervising Semantic Annotators

3.1 Beyond string-matching

As described in the previous chapter, the majority of tools developed for ontology-based concept recognition have been rule-based [102, 220]. Rule-based methods typically identify potential concepts within text using string matching, to identify candidate concepts, coupled with heuristics, to refine these candidates. These heuristics usually comprise mechanistic rules for concept subsumption, word order, and word overlap. However, these methods cannot identify synonyms of words if they are not written as explicitly defined in the ontology, or have the stems of the words as they are found in the ontology. As a consequence, these methods tend to have high precision but low recall scores [136].

This synonym-gap can be partially addressed by transforming words into numerical vectors that represent their semantics, known as word embeddings. Embeddings of words that have similar meanings are closer to one another in semantic space [149, 119, 27]. These embeddings can be used as inputs for neural networks for a variety of tasks such as classification, language translation, and named entity recognition (NER).

Recently, Ontology-based concept recognition methodologies have begun to incorporate neural nets, typically employing RNNs [18, 59], as they can learn dependencies between words in sequences. However, these methods rely on substantive manual annotation or noisy heuristic data generation. For example, Batbaatar and Ryu [18] used an ontology to heuristically label a training corpus, while Dong *et*

al. [59] relied on manual annotation carried out by medical specialists. Manual annotation of a corpus by a small group of experts may result in inconsistent model performance, as common concepts are likely to be more readily identified than those under-represented in the training corpus.

3.1.1 Reducing the cost of training data annotation

Researchers have explored ways of mitigating the cost of manual expert annotation, usually through distant supervision [252, 1]. Zhi *et al.* [252] merged a number of partially annotated datasets. The tags from each dataset were propagated to the others, in order to extend the coverage of each set of gold-standard annotations. To generate a noisy training dataset for low-resource language NER, Adelani *et al.* [1] used heuristic rules to tag entities that matched, preceded, or followed specific patterns. For example, lists of names were used to label matches from text. However, the dataset this produces will contain a high proportion of false negatives - where positive entity mentions are left untagged. To mitigate this they estimated a "noise channel" using a small set of gold-standard data. This noise channel was then leveraged to clean the labels of the larger dataset. While these methods produce NER models that perform nearly as well as those trained on large gold-standard datasets, they still require some expertly-annotated data.

Outside the realm of NER, unsupervised techniques have been developed to train image classifiers using noisy-label training data [2]. Xia *et al.* [234] constructed an image dataset by inputting keywords into an image search engine, labelling the returned images with the keywords. The dataset is noisily labelled, as although most of the results returned by the search engine are likely in the correct semantic class, an unknown proportion of the images are not. A variational autoencoder is trained to reconstruct the images. The reconstruction error of the autoencoder is then used to discriminate between true positive and false positive labels in the noisily labelled dataset. The intuition here is that the image vectors of the true positives represent the same semantic concept - be it a cat or a yacht - therefore their vectors will be situated in close proximity to one another in semantic space or contain similar features. Conversely, the false positives will be outliers scattered around the semantic space, or may contain different features to those of the true positives. When an autoencoder reconstructs a vector, the reduction of dimensionality between the hidden and input layers causes a bottleneck. To minimise the reconstruction error of an image dataset, the autoencoder has to find the signals that best represent the statistical regularities of the dataset. As the positive samples are clustered close to one another in semantic space, the autoencoder will learn to better reconstruct the features of these samples. Thus the reconstructions of the true positives have much less reconstruction error than the false positives. A deep neural classifier can then be trained on the augmented dataset. The authors found that expanding a training

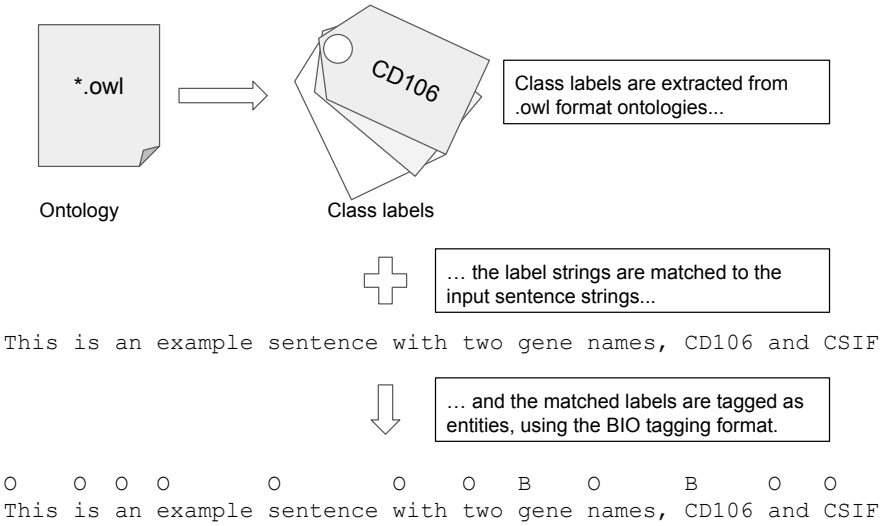

Figure 3.1: An overview of the heuristic data generation process. An owl format ontology and a text document of sentences divided by newline characters are input into the pipeline. The pipeline extracts the ontology class labels and then matches them to occurrences in the text sentences. Class labels from the ontology that are present in the sentences are then tagged as entities. Note that the data generated is noisy - only one of the gene names present is correctly tagged.

dataset with this method improved the performance of the classifier when evaluated against a gold standard dataset.

Ma *et al.* [139] recognised two distinct phases of deep neural classifier training: Dimensionality compression, and dimensionality expansion. During dimensionality compression the low-dimensional subspaces of the representation space are closely modelled on the elementary distribution of the data. This means that the model tries to learn the smallest number of variables needed for a minimal representation of the data - it learns to generalise. This is followed by dimensionality expansion, where the model learns to overfit to the less general features of the dataset. When the training data has noisy labels, the classifier overfits to the noise. The Latent Intrinsic Dimensionality (LID) of the deep representational subspaces can be estimated, so that these distinct phases can be differentiated. The authors devise a training strategy, called "dimensionality-driven learning" (DDL), to recognise the transition between stages and prevent overfitting to the noisy labels. Not only do these methods demonstrate marked improvements to image classifiers trained with noisy data but they also require limited supervision, eliminating the need for further expert annotation [234, 139].

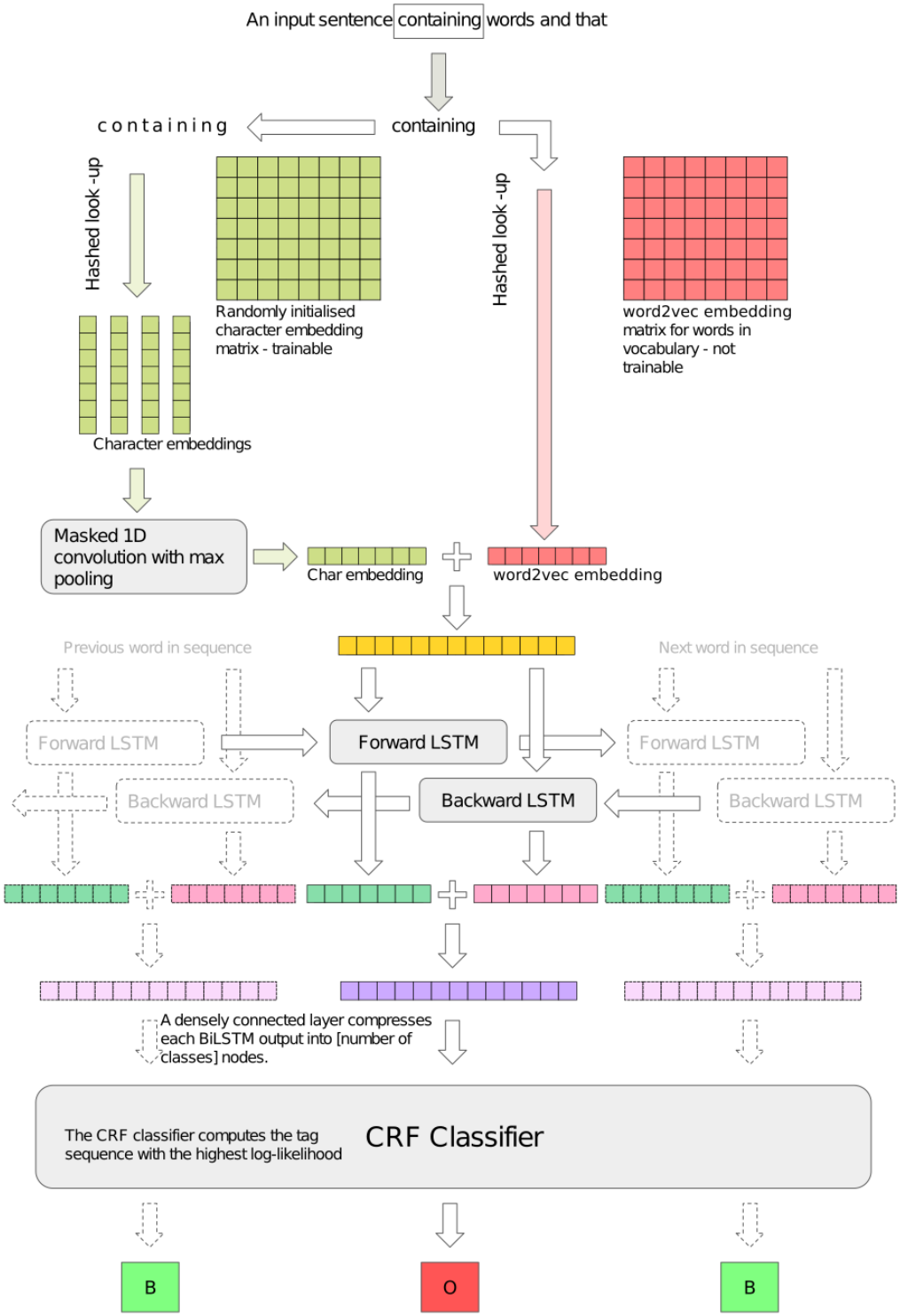

Figure 3.2: The architecture of the CNN-Bi-LSTM-CRF Named Entity Recognition model prior to any modifications.

3.1.2 Contribution

This work attempted to translate methodologies developed for distantly-supervised image classification to NER. First, a noisy training dataset was created. Class labels were extracted from the Cell Ontology (CL) and used to heuristically label the relevant ontology concepts within a large corpus of PLOS papers. The PLOS corpus was used as it is covered by a permissive Creative Commons Attribution (CC-BY) license and may be reused for any purpose [200]. This noisy-labelled dataset was used to train three NER architectures: an unmodified CNN-Bi-LSTM-CRF NER model, a CNN-Bi-LSTM-CRF incorporating discriminative autoencoders, and one integrating the DDL strategy. The models were evaluated against the expertly-annotated gold-standard CRAFT dataset [13] to ascertain which model best recreated the expert annotations.

Of the three models, qualitative assessment found the model that responded to changes in dimensionality to be defective. The remaining two models (the discriminative autoencoder and the unaltered NER model) were found to produce inadequate results (F1 scores of ∼6.5%). Following the results, the implications and future work are discussed. The scripts for the work undertaken in this chapter can be found here: https://github.com/lorcanpd/TagOntoText.

3.2 Methodology

3.2.1 Generating a noisy training dataset with heuristic labelling

The heuristic data generation process begins with the extraction of label strings from the domain ontology. Sentence strings from the raw training text are searched for ontology label string matches. When a match is found the words that match are tagged appropriately using the BIO format. Once all matches in a given sequence are found, all the words that were not matched are tagged as belonging to the negative class. Figure 3.1 contains an illustration of the process.

To build a training corpus, label strings were extracted from the CL [58] and matched to sentences from a collection of 263 548 PLOS papers. Of the 19 267 249 sentences in the PLOS dataset, 9 315 552 sentences contained at least one CL term.

3.2.2 The vanilla Named Entity Recognition Model

The unaltered NER model is a Character-CNN-Bi-LSTM-CRF like that described by Lample *et al.* [116] (see figure 3.2 for an overview of the architecture). The model incorporates embeddings to represent both the semantic meaning and the character morphology of the sequence tokens. To create the word embeddings, a word2vec [149] model was trained on the PLOS papers. The word2vec was parameterised with a context window of five tokens, and had a minimum word frequency of seven. The NER model learns word-morphology-specific character embeddings as it trains,

using a character-based CNN. Each token's pairs of character and word embeddings are concatenated and fed into a Bi-LSTM layer. The Bi-LSTM consists of a forward and a backward layer, which are used to learn order dependence between the word representations. The outputs of both the forward and backward LSTM are concatenated for each token. Each of these vectors are passed to a dense neural layer that compresses the LSTM representation into the same number of nodes that there are classes. The output of this layer is fed into a Conditional Random Field (CRF) classifier. The CRF classifier predicts the most probable sequence of labels for a sentence given the current configuration of model weights.

The CRF prediction is scored using a combination of the raw outputs from the dense layer and the transition values from the CRF transition matrix. For each word there are raw values corresponding to each label class. The higher the raw value for a label class the more likely the model will predict that label for a word. The predicted labels are compared to the training labels by summing together the raw values of the correctly labelled tokens. So if incorrect labels are predicted, the resultant sequence scores will be lower.

The transitions between each label in the sequence are scored using a transition matrix. The transition matrix provides a value that reflects the likelihood of a particular label succeeding another in a sequence. The more likely the transition is, the closer its matrix value is to 1. The less likely the transition, the closer the value is to zero. For example, it learns from the training labels that an "I" label cannot follow an "O" label. If an "O" label follows an "I" label in a predicted sequence this transition is awarded a zero. The transition values and values of the correct sequence labels are summed together and normalised to create the sequence score. This score is then converted into a negative log-likelihood loss. The model learns to use a word's morphology, semantic representation, context, and the most probable labels of neighbouring words, to predict which words in a sequence are named entities.

3.2.3 Incorporating a Discriminative Autoencoder

The discriminative autoencoder methodology, as originally described, was used to clean the labels of a training set prior to training an image classifier [234]. The cleaning was an iterative process with two steps: Discriminative labelling, and reconstruction learning. During discriminative labelling, true positives are estimated from the noisy dataset based upon their current reconstruction errors. During reconstruction learning, the autoencoder is trained on the samples labelled as true positives in the discriminative labelling step. Initially the network parameters are randomly initialised, so the reconstruction error is not discriminative. However, after a few iterations the error distributions start to become more separable, and the labelling process produces fewer false positives. In turn, this makes the autoencoder

better at reconstructing true positives. As the process continues, the reconstruction errors become ever more separable, and the labels gradually become stable between iterations. Once the labels are fixed the training is stopped.

There are reasons why, in this NER case, using an autoencoder as a pre-processing step is not ideal. In the NER model, each word in a sequence is represented by both a semantic vector and a word morphology vector. The word morphology vectors are randomly initialised but change as the model fits. If the autoencoder were used to correct the labels prior to the NER training, it would not be provided with morphological information - which can be important for classifying words that are not covered by the word embedding model. Instead the autoencoder can be incorporated into the training process of the NER.

At each training iteration, the concatenated semantic and morphology vectors for each word in the batch are passed through the autoencoder. The reconstruction error for all of the vectors is clustered into two groups using the Jenk's natural breaks optimisation algorithm [100]. This algorithm minimises the variance within each group while maximising the variance between them, and returns a value that best demarcates the two clusters. Vectors that have a reconstruction error below this value are considered to be true positives, and those above to be true negatives. Here, the CRF scoring function is amended so that if a word is labelled as negative in the training set, yet has a reconstruction error in the lower cluster, then the CRF score for this word is multiplied by -1. This effectively penalises the model for contradicting the discriminative autoencoder proportionally to the magnitude of disagreement. At the end of the training iteration, the vectors of all of the words predicted to be positive by the NER model are used to train the autoencoder.

The intuition is that the heuristic dataset likely contains a far greater proportion of false negative labels than false positive [1]. False positives are likely to have broadly similar semantics and/or word morphology to true positives from the same domain (For example, words beginning with the prefix "phyto-"). Preventing the NER model from learning negative labels for possible false negatives, while learning the positive labels for semantically/morphologically similar true positives should result in the model relying on a word's in-sentence context to predict its label. If a word's context suggests that it is a positive sample, and it has similar semantic meaning or similar morphology to a positive sample, it should be more likely to be labelled as a positive sample - even if it is negative in the training data.

It is important to note that this adaptation of the methodology does not re-label these potential false negatives as positives. Rather, it is intended to discourage learning negative labels for these false positives. This is because NER in the BIO form is always a multi-class classification problem, even if there really is only *one* positive class. Deciding which positive class the false negative should belong to is complex in an unsupervised setting. Even in the "single" positive class case, the algorithm must decide between two positive labels "B" (beginning of entity) and

"I" (inside an entity).

3.2.4 Employing a dimensionality-driven learning strategy

Dimensionality-driven learning measures the latent intrinsic dimensionality (LID) of a model's representational subspaces to identify when it begins to overfit to the training data. In the original paper, Ma *et al.* [139] evaluated the strategy in a CNN architecture for image classification, with the penultimate neural layer being the representational subspace measured.

Computing the LID for an entire dataset with respect to a sample is prohibitively expensive [139]. Instead the LID of a training sample is estimated from its k-nearest neighbours, within its training batch, randomly selected from the entire dataset. The LID estimate, $\hat{\text{LID}}$, for a sample x is given by:

$$\hat{\text{LID}}(x,\ X_B) = -(\frac{1}{k}\Sigma_{i=1}^{k} \log \frac{r_i(g(x),\ g(X_B))}{r_{\max}(g(x), g(X_B))})^{-1} \tag{3.1}$$

Where X_B is the sample batch. The function $g(x)$ is the output of the second-to-last layer of the network for sample x. The expression $r_i(g(x), g(X_B))$ is the distance between the outputs of the second-to-last layer for the sample point x and its k nearest neighbours from the batch X_B. While $r_{\max}(g(x), g(X_B))$ represents the radius from sample x, or the greatest distance found between x and the samples in X_B. The LID score that would have been calculated using the full dataset can be estimated reliably, as long as the batch size is sufficient to ensure that the k-nearest neighbours of x at the second-to-last output layer remain in the vicinity of x in representational space [139].

The LID score for each epoch is approximated by computing the $\hat{\text{LID}}$ for each sample within a batch, for each batch, and then averaging them. The $\hat{\text{LID}}$ score is used to calculate a factor α that updates at the end of every training epoch:

$$\alpha = \exp(-\lambda \frac{\hat{\text{LID}}_i}{\min_{j=1}^{i-1} \hat{\text{LID}}_j}) \tag{3.2}$$

Where $\lambda = i/T$, i is the epoch, and T is the total number of epochs. $\min_{j=1}^{i-1} \hat{\text{LID}}_j$ is the lowest LID score of the previous epochs. This factor serves as a weighting that represents decreasing confidence in the raw labels when the training is in the dimensionality expansion stage. This factor is used to compute adaptive LID-based labels, which are used to "reduce the effect of noisy labels on learning the true data distribution":

$$y^* = \alpha_i y + (1 - \alpha_i)\hat{y} \tag{3.3}$$

Where y is the raw label, $\hat{y}$ is the predicted label, and y^* is the corrected label. Once the LID score of the present epoch is greater than two standard deviations of the mean LID score of the w preceding epochs, the model is rolled back to the state of

the previous epoch, and the loss is transformed from a cross-entropy loss to this:

$$\mathcal{L} = -\frac{1}{N}\sum_{n=1}^{N}\sum_{y_n^*} y_n^* log P(y_n^*|x_n) \tag{3.4}$$

Where N is the number of training samples, and $P(y_n^*|x_n)$ is the predicted class probability of y_n^* given the sample x_n.

To adapt the dimensionality-driven learning strategy to NER, two changes were required. The first change was to which layers are extracted to have their LID estimated. The outputs of the Bi-LSTM layers were chosen. This representational layer encodes information about the word semantics, morphology, and the dependencies between words. Only the LIDs of Bi-LSTM elements corresponding to words were estimated. This (as opposed to all 100 elements of the Bi-LSTM) ensures that the semantic-morphology-dependence space does not represent any out-of-sequence tokens. Out-of-sequence tokens would skew any estimate of the LID in a manner highly dependent on the length of sequences in a batch. For example, when there are shorter sentences there are more out-of-sequence tokens. As these tokens occupy the same semantic space this would result in a lower LID estimation.

An additional change was made to the CRF scoring function. The scoring function had to be altered to incorporate the policy factor. The scoring of a sequence was transformed into a two step process: Firstly the normal CRF label scoring is calculated for the sequence. Secondly, the score is calculated again, but as if the labels predicted were correct, that is the highest scoring raw values for each element were used to score the algorithm. These two scores are input into equation 3.3, where the scores are apportioned, depending on the policy factor, into an aggregate score. This resultant aggregate score is transformed into the loss used for training. The policy factor does not affect the training of the CRF transition parameters, so while increasing confidence is given to the predictions of the model, the correct transition rules are still learned. See figure 3.3 for an illustration of both of the proposed changes to the algorithm.

3.2.5 Training

The 9 315 552 heuristically tagged sentences were divided into 60-20-20 training-test-validation split. This resulted in 5 589 332 sentences for training, and 1 863 110 for testing and validation. The models were trained over two epochs, using a batch size of 300 sequences. The control and DDL models all achieved >98.7% precision and recall on the heuristically generated test data at the end of the second training epoch.

Table 3.1: Gold-standard validation metrics for the control and discriminative autoencoder models.

Model	Precision %	Recall %	F1 %
Standard NER (control)	**4.0**	**19.5**	**6.6**
Discriminative Autoencoder	3.9	19.2	6.5

3.2.6 Evaluation

The trained models were evaluated using the gold-standard annotated CRAFT corpus [13]. The CRAFT corpus consists of 97 documents that have been annotated with occurrences of concepts from nine biomedical ontologies (the CL, the Chemical Entities of Biological Interest ontology, the NCBI Taxonomy, the Protein Ontology, the Sequence Ontology, entries of the Entrez Gene database, and the three sub-ontologies of the GO. For this study only the CL annotations were used. The annotations in the craft corpus were transformed into the BIO format and the predicted labels compared with those from the corpus.

3.3 Results

Table 3.1 displays the validation metrics for the control and discriminative autoencoder models. Qualitative assessment of the trained models found the DDL model to be defective. It produced series of alternating positive and negative tags that had no apparent relation to the sequence being classified. As a result it has not been included in the results table.

3.4 Discussion

As shown in table 3.1, both functioning models performed poorly. As indicated by the recall, fewer than a fifth of concepts were correctly tagged as positive by the models. The precision shows that the positive predictions made by both models were incorrect in excess of 95% of the time. The models appear to replicate the heuristic tagging process.

The inclusion of the discriminative autoencoder in the model training did not improve the performance of the model. The assumption that the autoencoder from the image-dataset cleaning task could be straightforwardly translated to an LSTM model was false. While, the autoencoder used to clean image datasets only had to identify one class, the ontology terms comprise many semantic classes in various combinations. The relatively simple autoencoder architecture used here, likely lacked the parameter space to encompass the required signals to reliably reconstruct the multiple semantic classes. This was also likely compounded by the limitations discussed earlier in subsection 3.2.3.

A key limitation of LSTM-CRF-based NER models, using BIO labelling, in an ontology-based concept recognition scenario is that they do not map text to specific concepts [203] but to spans of text. Further work should explore other methodologies that map text to specific ontology terms, and avoid the need for noisy datasets or distant supervision. For example, doc2vec [119] learns vector representations for words and documents in a corpus. Firstly, the vocabulary is represented by a matrix, where each row corresponds to a unique word. During training the model tries to predict a word from its context, using the concatenation of the context words' vectors as input features. To learn representations of documents, the model alters the word representation process. It can do this using two different algorithms: Distributed Memory (DM), or Distributed Bag of Words (DBOW). For DM, each document in the training corpus is represented by a vector. This vector is added to the feature inputs for word prediction for words in that document. This vector behaves as a memory of words that had previously appeared in the document. This extends the context beyond the immediate context window of the current word. DBOW, like DM, represents documents with a vector, but ignores the context words. Instead words are randomly sampled from the document and the document vector is used as the feature input to predict the words. Once trained, either iteration of doc2vec model can be used to infer a vector for a novel document.

I trained a doc2vec model using the corpus of PLOS papers. The trained model was used to infer a document vector for each ontology concept using the text string contained within its associated "<owl:AnnotationProperty>". Vectors were also inferred for strings of text that described or contained references to ontology concepts. I then calculated the cosine similarity between the vectors of the sentences and the ontology terms to find the ten most similar ontology terms for each sentence. The sentences and the best matching terms were qualitatively assessed. On inspection, it was found that the top ten ontology terms returned when using the DBOW algorithm were more accurate. This suggests that word order is not a decisive factor in this scenario, and that the combination of semantic signals contained in text are more important.

If the semantic content alone is the best feature to use for this task then a CNN could be used to identify concepts in a sequence. As a CNN uses filters to search for particular numerical features, and it would not be affected by additional words in the way that the inferred word vector would be. The CNN would learn to quantify the strength of the task-appropriate semantic features.

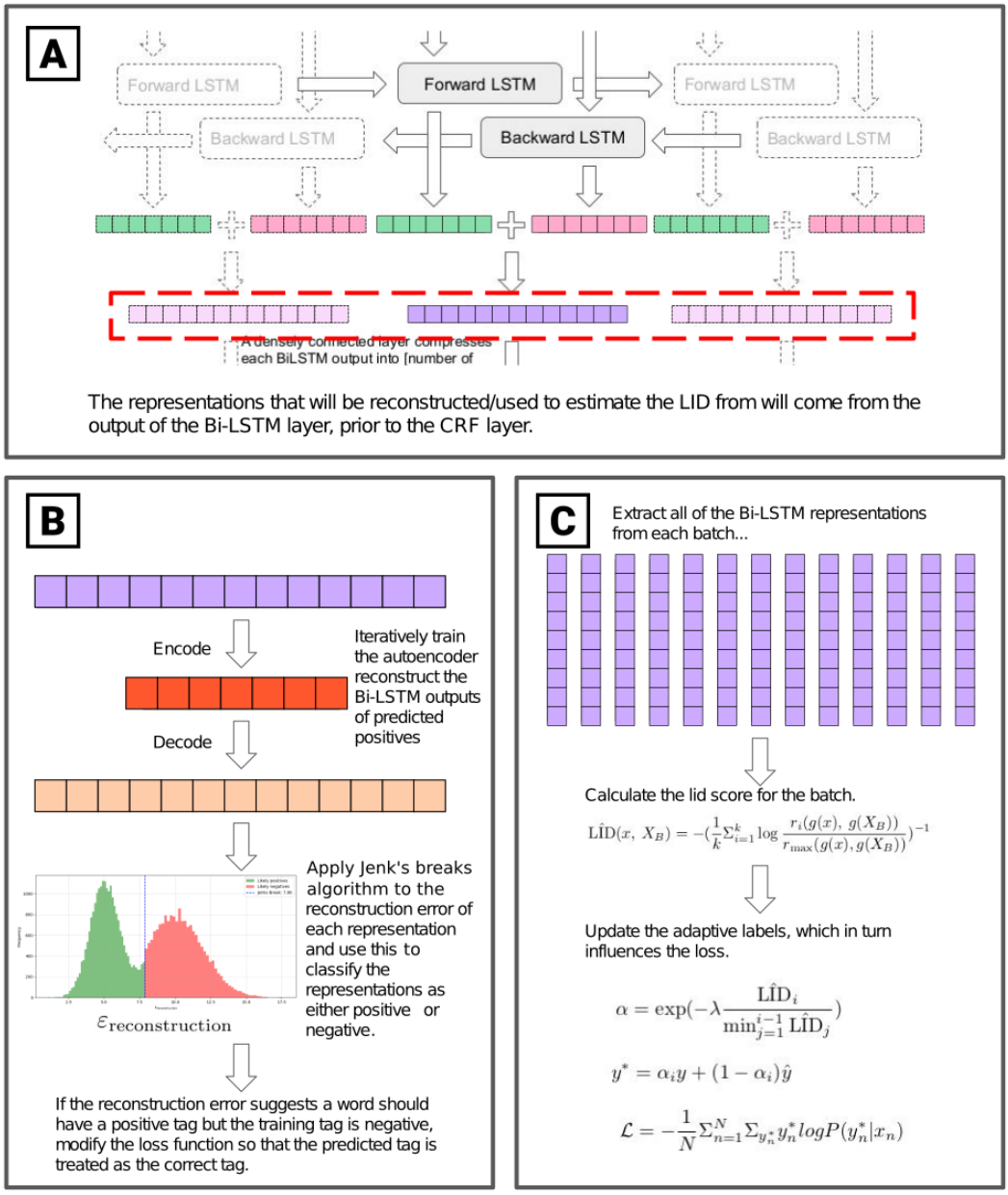

Figure 3.3: An overview of the proposed changes to the model: **A** Indicating which representations will be used as the inputs to the modifications; **B** A visualisation of the error reconstruction method; and **C** a visualisation of the Latent Intrinsic Dimensionality method.

Chapter 4

Augmenting Neural Dictionaries with Attention for Multiple-Ontology Semantic Annotation

Portions of the work detailed in this chapter were published as a peer-reviewed conference paper [172] and have been reproduced with permission.

4.1 Introduction

4.1.1 Neural Dictionaries

Simultaneously addressing the synonym gap while avoiding the training corpus issues, Arbabi *et al.* [7] created a method that exploits semantic word embeddings and which only requires an ontology for training: the Neural Concept Recogniser (NCR). The NCR is a "neural dictionary": a type of deep neural net that learns to behave like a lookup hash or dictionary - given a key an appropriate value is returned. The NCR uses a simple convolutional neural net (CNN) to learn associations between sequences of word embeddings and concept embeddings. Instead of searching based on exact or partial text matches, the neural dictionary converts signals from the input text into the semantic representation space of the concepts and finds the most similar concept embeddings. The NCR shares semantic features between related concepts using a "concept ancestry" matrix, where the semantic features of parent concepts are "inherited" by its children. Arbabi *et al.* [7] found that pooling semantic information between related concepts using the ancestry matrix improved model performance.

One advantage of the NCR is that, unlike the RNN- or LSTM-based models, it requires no manual or heuristic annotation for training. Additionally, the model can

be applied to any domain for which there is an OBO format ontology. No further domain expertise beyond that already contained in the ontology is required. While the simple CNN architecture allows the model to exploit semantic embeddings, word order dependency information is lost. When the NCR model was evaluated against a Gold-Standard Corpus [133] of text annotated with concepts from the Human Phenotype Ontology (HPO), the Neural Concept Recognition model achieved micro and macro F1 scores of 70.2% and 73.9% respectively.

Luo *et al.* [136] combined a string matching approach with a neural classifier for the task of Human Phenotype Ontology concept recognition, "PhenoTagger". A string matching dictionary of concepts names, synonyms, and their lemmatised forms was created from an ontology. This dictionary was then used to create a distantly supervised training dataset, where an ontology term's labels were used to create positive samples associated with the concept ID, and negative samples were sampled from a biomedical text corpus and associated with a "none" ID. The neural classifier employed was a pre-trained BioBERT language model [124], a transformer-based language model, which uses attention to modify the word embeddings of tokens based on their context. The pre-trained BioBERT model was trained further to associate the samples with their respective tags. In deployment, PhenoTagger uses the string-matching dictionary to identify likely terms from text and BioBERT to classify the text with the most likely ID. PhenoTagger achieves a "document-level" f-score of 75.7% - the current state-of-the-art for neural dictionary methods.

4.1.2 Attention

Arbabi *et al.* [7] mention that attention mechanisms were tried as an alternative to the simple convolutional layer, but that they were found to not be as effective. This may be explained by either the conjunction of poor inductive bias and the limited training data, or the relative efficacy of CNNs at local feature extraction.

Hu *et al.* [94] developed an attention-based technique for augmenting a CNN, called Squeeze-and-Excitation. This technique moderates the maximum feature signals using the average signals of the feature maps. To do this, a vector of all the average signals of each feature map is passed to a neural network consisting of two transformations: a non-linear transformation to reduce the dimensionality of the data, followed by another non-linear transformation back to the size of the average signal vector. The final vector is then used to scale the maximum signal vector element-wise. The bottleneck caused by the dimensionality reduction encourages the model to learn dependencies between average features allowing it to selectively attend to different maximum signals. The authors found this technique to significantly improve the performance of CNN-based classifiers at image recognition tasks.

Guo *et al.* [78] describe an alternative configuration of multi-headed self-attention, meant to address the transformer's poor inductive bias, called Scale-Aware

Self-Attention (SASA). With SASA, each attention head attends to a variable scale. The variable scale restricts attention to within a certain neighbourhood of each sequence position. The intuition here is that words that are in close proximity within a sentence are more likely to contain contextual information relevant to each other. In addition, this forces the attention heads to attend to a smaller set of features, so the relative differences between the remaining features are more pronounced. Attention is focused on words more likely to provide relevant context and the signals from these words are also relatively stronger, providing an improved inductive bias. This technique was found to exceed the state-of-the-art, or was at least competitive for a number of NLP tasks, while simultaneously requiring far fewer training examples than more conventional attention-based models.

As transformer models have many parameters, they have a propensity to overfit to training data, preventing the model from generalising. To remedy this, different drop-out techniques have been proposed. Zhou *et al.* [254] applied a structural dropout mechanism, DropHead, to prevent overfitting. During training, DropHead randomly drops entire attention heads. The model may no longer be able to rely on a particular attention head to attend to a certain region of feature space, so all attention heads must also attend to overlapping areas of feature space. This prevents a minority of attention heads from dominating the model while the other heads contribute little.

Wu *et al.* [233] describe UniDrop, which combines multiple dropouts to further improve the transformer's ability to generalise: "feature dropout", "structure dropout", and "data dropout". Feature dropout consists of applying dropout at various points within the attention mechanism: to the attention weights, activation layer, to the query, key, and value matrices, and to the output features prior to the linear transformation. In this case structure dropout alters the architecture so that an entire attention layer can be dropped during training. Data dropout refers to randomly removing a proportion of tokens from the input sequence. UniDrop was found to improve the performance of transformer models without additional training data or computational power.

Transformers, as originally described [221], also require a learning rate warm-up in order to train properly. During the warm-up, the learning rate is gradually increased from a small value to a maximum, over a set number of training iterations. Xiong *et al.* [235] argue that as the architecture applies layer normalisation after the residual blocks, the output parameters' expected gradients are always large at initialisation. This impacts the stability of training when a large learning rate is used. The warm-up period results in "well-behaved" gradients, but also in a longer training process and the addition of sensitive hyper-parameters that require tuning. Xiong *et al.* [235] show that moving the layer normalisation inside the residual blocks removes the need for a warm-up period, as the gradients are smaller at initialisation.

4.2 Contribution

The work described in this chapter achieves a new state-of-the-art for ontology-based concept recognition. The performance of the CNN neural dictionary architecture is improved by using higher-quality word embeddings and incorporating an attention mechanism that models dependencies between convolutional filters. Unlike previous approaches, this neural dictionary can incorporate multiple domain ontologies at once, with the model leveraging multiple ontologies comprised of more diverse domains performing best of all. This work also adds further credence to modifications enabling transformer-based architectures to perform competitively with CNNs, when training data is limited. The code for this chapter can be found here: https://github.com/lorcanpd/adorNER.

4.3 Methodology

The models described here build upon the work of Arbabi *et al.* [7] and adapt the NCR architecture.

4.3.1 Neural Concept Recogniser adapted to use ELMo Word Embeddings

The NCR model broadly consists of two parts: the concept embeddings, and the CNN encoder. All of the concepts in the ontology are represented by a matrix of embeddings, $\mathbf{H}$, where each row on the matrix corresponds to a concept and each column represents a dimension of feature-space. The NCR does not learn $\mathbf{H}$ directly. Instead, it learns the matrix $\tilde{\mathbf{H}}$, where $\tilde{\mathbf{H}}_c$ are the semantic features of concept c that are "novel" compared with the concept's parents. This can be thought of as a local embedding, positioning a concept relative to its parents in feature space. The global representations, $\mathbf{H}$, are derived by calculating the product of $\tilde{\mathbf{H}}$ and the ancestry matrix $\mathbf{A}$ ($\mathbf{H} = \mathbf{A}\tilde{\mathbf{H}}$). Each element $\mathbf{A}_{i,j}$ is non-zero only if the concept j is an ancestor of i (this includes $i = j$). Each row of the ancestry matrix is calculated as follows:

$$\mathbf{A}_i = \text{OneHot}(i) + \frac{1}{|\text{parents}(i)|} \sum_{j \in \text{parents}(i)} \mathbf{A}_j \tag{4.1}$$

This results in a concept's ultimate embedding being the average of its parents' embeddings summed to its own unique "raw" embedding ($\mathbf{H}_c = \mathbf{H}_{p_c} + \tilde{\mathbf{H}}_c$). This incorporation of taxonomic structure from the ontology means that when a concept's raw embedding is updated, the global embeddings of its ancestor concepts are also updated.

The CNN encoder projects text into a numerical representation that positions the input phrase in the semantic space of the concept embeddings. To do this, first the text is encoded into a sequence of fixed-length semantic embeddings. Unlike

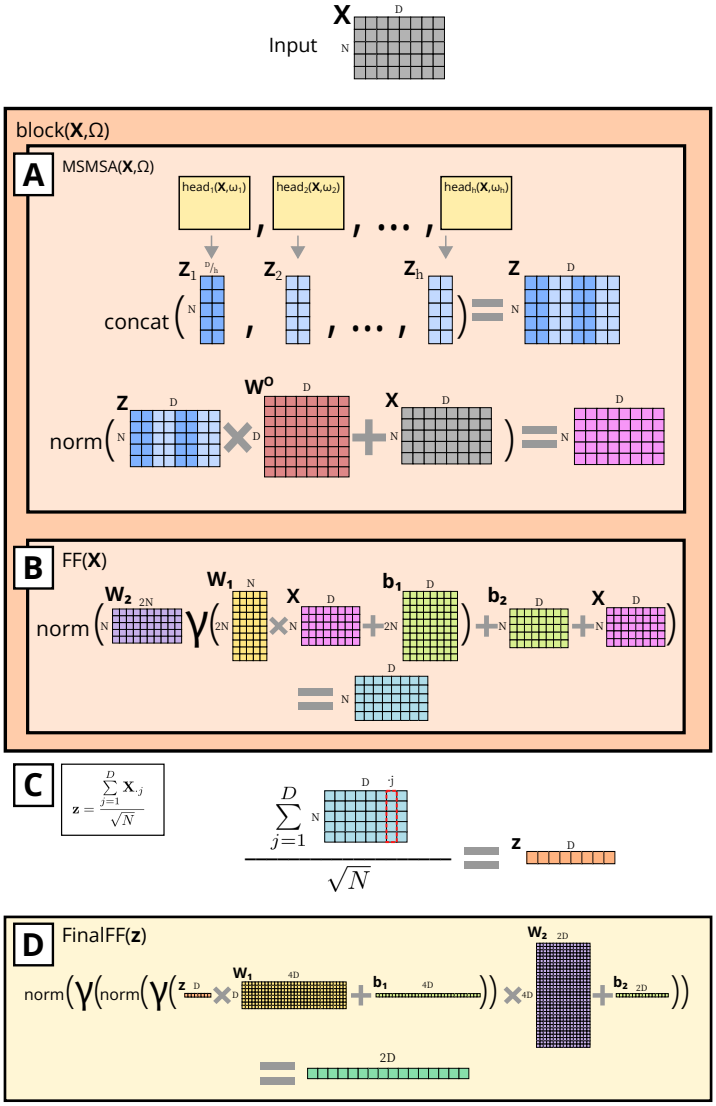

Figure 4.1: Overview of the scaled-attention encoder. Each attention block comprises two layers, a Multi-Scale Multi-headed Self Attention (MSMSA) layer and a feed-forward (FF) layer. **A** Inside the MSMSA layer, the scaled attention heads extract composite semantic signals from interactions between each token and its context in the input sequence (see figure 4.2 for an illustration of the operation of the attention heads). Each attention head attends to different regions of semantic space. These composite embeddings are linearly transformed and then added to the original input to enrich the original embeddings with contextual information. **B** The MSMSA outputs are then fed into the FF layer. The FF layer allows the outputs of the MSMSA to be projected non-linearly, and adds these non-linear projections to the MSMSA output to further enrich the embeddings. These attention blocks can be stacked multiple times. The intuition here being that more blocks allows for greater abstraction, as further composite signals can be extracted from the interactions between embeddings enriched with composite signals. **C** After the final block, the enriched embeddings are amalgamated into a single vector by summing element-wise across each dimension, and then scaling the resulting vector by the square root of the length of the token sequence (excluding padding tokens). **D** A final feed-forward (FinalFF) network non-linearly transforms the semantics of this vector into the semantic space of the ontology concept embeddings.

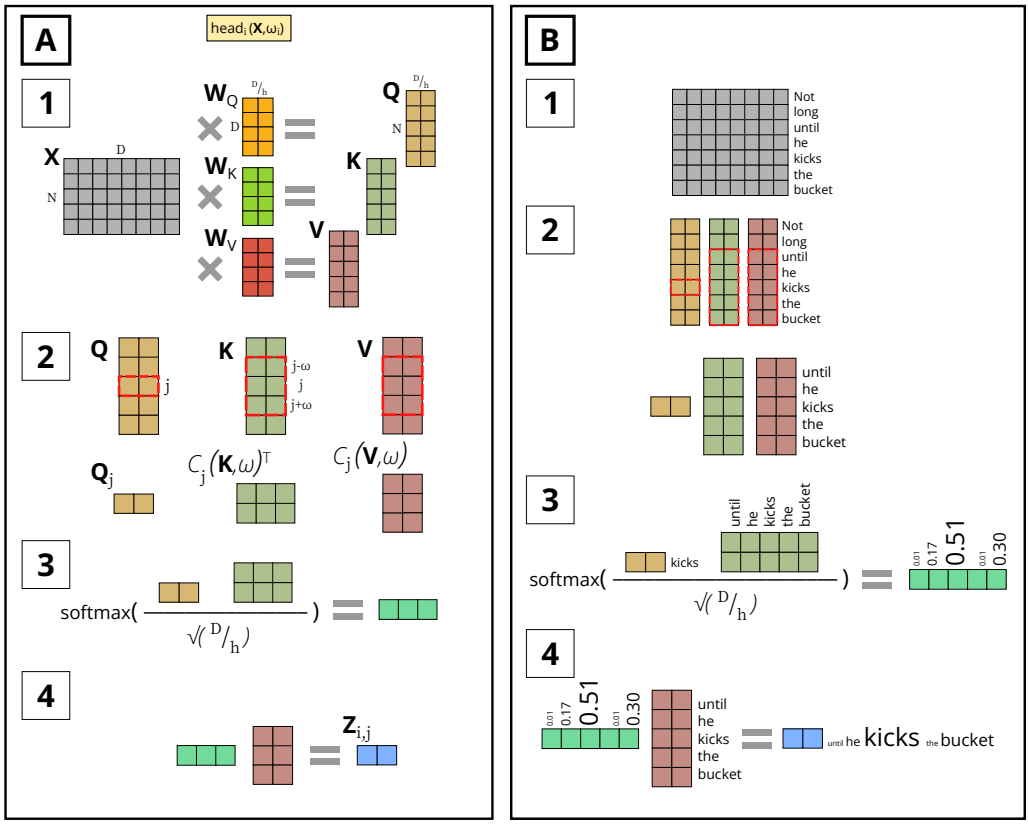

Figure 4.2: **A** A scaled attention head. **B** An illustration of the how an attention head weighs semantic signals. [**1**] The input matrix $\mathbf{X}$ represents the D-dimensional embeddings of an N-length sequence of tokens. The input matrix is multiplied separately by three different parameter matrices $\mathbf{W}_{\{Q,K,V\}}$ to produce query $\mathbf{Q}$, key $\mathbf{K}$, and value $\mathbf{V}$ matrices. These multiplications linearly transform the sequence of tokens into different representational subspaces. They also reduce the dimensionality of the representational space, causing bottlenecks. Bottlenecks reduce the number of signals so that only the more relevant signals are retained. $\mathbf{Q}$ and $\mathbf{K}$ retain signals that interact to provide indications of semantic relevance between tokens in the sequence. While $\mathbf{V}$ retains the semantic signals that are to be combined into the composite embeddings. [**2**] The context extraction function, described in equation 4.7, is iteratively applied to the $\mathbf{K}$ and $\mathbf{V}$ matrices to extract the within-ω-tokens context of the j-th row, while only the j-th row is extracted from $\mathbf{Q}$. [**3**] Self-attention scores are calculated for each token using the iteratively extracted segments of $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$. The outputs corresponding to the j-th token in the sequence are calculated using only the representations of tokens within ω tokens of j. This means that only the self-attention scores of the within-ω context tokens of the j-th token are calculated with respect to the j-th token. [**4**] These self-attention scores are then used to weight the signals from the context of the j-th token in $\mathbf{V}$, combining them into the composite-signal output vector $\mathbf{Z}_{i,j}$. For each attention head, each token's output is concatenated together to represent the entire sequence, $\mathbf{Z}_i$.

the original NCR, which used 100-dimensional word embeddings from a pre-trained FastText [27] model, this work instead used 512-dimensional ELMo [171] embeddings from a pre-trained model to encode the input text. Only the character-CNN embeddings were used from the ELMo model. This was necessitated by a compiler incompatibility between FastText and Tensorflow 2.2. The filters of the encoder's convolutional layer then project the word embeddings into feature space as feature maps. Max-pooling then concentrates the feature maps of a sequence into a single vector by selecting the strongest signal from each feature map. This vector is then passed to a fully-connected layer that projects the feature signals into the semantic representation space of the concept embeddings. The dot product of the phrase's feature representation and the global concept embeddings are input into a softmax classifier, represented by the following equation:

$$p(c|\mathbf{e}) \propto \exp(H_c\mathbf{e} + b_c) \tag{4.2}$$

Where b_c is the bias term, and $p(c|\mathbf{e})$ is the probability of the concept c being the correct concept, given the phrase embedding $\mathbf{e}$.

To train the model, the label and synonym text for each concept is paired with the concept's unique identifier. The objective of model training is to increase the softmax score between the correct concept's embedding and the representations of the labels, output by the CNN. As the model trains, the weights of the CNN encoder and the "raw" embedding values are updated through backpropagation. This results in the ontology label representations produced by the CNN gradually becoming more similar to those of the concepts they represent.

Once training is complete, the model can be used to perform concept recognition. Input sentences are split into all of the possible n-grams they contain ($n \in \{1, \dots, 7\}$). These n-grams are filtered to only include candidates that have a concept-matching softmax score that is greater than a predetermined threshold. A post-processing step filters the remaining n-grams so that if two n-grams correspond to the same concept, the smaller n-gram is retained. If the overlapping n-grams are matched to different concepts, the shorter n-gram is dropped. This is to ensure that more specific concepts are prioritised over more general ones.

4.3.2 Squeeze-and-Excitation

The NCR model is augmented to include a simple Squeeze-and-Excitation (SAE) mechanism. A SAE mechanism uses information from the feature maps produced by a convolutional layer to weight their importance. This work adapts the methodology described in Hu *et al.* [94] to a one-dimensional CNN.

To "squeeze" the feature information, average-pooling is applied to each of the NCR CNN's output feature maps. The average-pooling distills the feature maps into a single vector, $\mathbf{z} \in \mathbb{R}^F$ where F is the number of feature maps. Each element

of $\mathbf{z}$ is calculated thusly:

$$z_f = \frac{1}{N_f} \sum \mathbf{u}_f \tag{4.3}$$

Where z_f is the statistic for the f-th filter, N_f is the number of non-zero elements in the f-th filter, and $\mathbf{u}_f$ is the f-th feature map vector.

The vector of all of the feature map statistics is then passed through the "excitation" bottleneck, as follows:

$$\mathbf{s} = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z})) \tag{4.4}$$

Where $\mathbf{s}$ is the vector of filter weights, $\mathbf{W}_1 \in \mathbb{R}^{\frac{F}{r} \times F}$ the parameter weights of the compression transformation, $\mathbf{W}_2 \in \mathbb{R}^{F \times \frac{F}{r}}$ the parameter weights of the decompression transformation, σ and δ are the sigmoid and ReLU activation functions respectively, and r is the compression ratio. This "excitation" bottleneck function captures non-linear dependencies between features, with no limit to the number of features that can be enhanced. The feature maps are then multiplied filter-wise by $\mathbf{s}$ to weight them prior to the max-pooling layer. As the model is trained, the SAE mechanism learns to use global signals to help the model pay relatively more attention to relevant features, improving discriminability.

4.3.3 Multi-Scale Self Attention (MSSA)

Architecture Here, the NCR model CNN architecture is replaced entirely by an attention-based sentence encoding architecture, not unlike the Universal Sentence Encoder [39]. This attention architecture is altered to attend to multiple fixed-scale windows, as in Guo et al. [78].

Given an input of word embeddings $\mathbf{X} \in \mathbb{R}^{N \times D}$, where N represents the number of embeddings and D their dimensionality, each attention head can be described like so:

$$\text{head}(\mathbf{X}, \omega)_{i,j} = \text{softmax}(\frac{\mathbf{Q}_{ij} C_{ij}(\mathbf{K}, \omega)^\top}{\sqrt{\frac{D}{h}}}) C_{ij}(\mathbf{V}, \omega) \tag{4.5}$$

Where ω is the scale parameter, i corresponds to the i-th head, and j to the j-th element of the sequence. $\mathbf{K}$, $\mathbf{Q}$, $\mathbf{V}$ are the projections of $\mathbf{X}$ into $N \times \frac{D}{h}$ subspaces, with h being the number of heads. The input $\mathbf{X}$ is multiplied separately by each of the parameter matrices $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$ (all $\mathbf{W} \in \mathbb{R}^{D \times \frac{D}{h}}$), projecting the input into the Key Query and Value subspaces. The multiplication by parameter matrices reduces the dimensionality of the representational subspace. This causes a bottleneck meaning that only the most relevant signals are retained.

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \ \mathbf{K} = \mathbf{X}\mathbf{W}_K, \ \mathbf{V} = \mathbf{X}\mathbf{W}_V \tag{4.6}$$

$C_{ij}(\mathbf{X}, \omega)$ is the context-extraction function, where:

$$C_{ij}(\mathbf{X}, \omega) = [\mathbf{x}_{i,j-\omega},\ \ldots\ ,\ \mathbf{x}_{i,j+\omega}] \tag{4.7}$$

The context-extraction function dynamically pads the inputs with zeros if the iterative context window extends beyond the first and last rows of the input. Please see figure 4.2 for an illustration of how the self-attention mechanism is scaled using the context extraction function.

The h heads are incorporated into a Multi-Scale Multi-headed Self-Attention (MSMSA) block. The block consists of the attention layer and a feed-forward network. The attention layer is computed as follows:

$$\text{MSHSA}(\mathbf{X}, \Omega) = \text{norm}([\text{head}_1(\mathbf{X}, \omega_1),\ \ldots\ ,\ \text{head}_h(\mathbf{X}, \omega_h))]\mathbf{W}^O + \mathbf{X}) \tag{4.8}$$

Where $\Omega \in \{\omega_1, ..., \omega_h\}$ is the set of scale parameters, $\mathbf{W}^O$ is a parameter matrix and norm is the layer normalisation function. A residual connection is applied by adding the input to the transformed concatenated outputs of the attention heads. The attention heads extract contextual cues regarding whether the strength of a token's semantic signals should be increased or decreased, and the linear transformation and the residual connection serve to enrich the original embeddings with this information. These enriched embeddings are then passed to the feed-forward (FF) layer. The FF layer is computed thusly:

$$\text{FF}(\mathbf{X}) = \text{norm}(\gamma(\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2 + \mathbf{X}) \tag{4.9}$$

Where γ is the GELU activation function, $\mathbf{W}_n$ are the parameter matrices and $\mathbf{b}_n$ are their bias terms. The FF layer non-linearly transforms the signals from the MSMSA output and a residual connection adds these transformed signals onto the outputs of MSMSA to further enrich them. Therefore each MSSA block can be described as:

$$\text{block}(\mathbf{X}, \Omega) = \text{FF}(\text{MSMSA}(\mathbf{X}, \Omega)) \tag{4.10}$$

These blocks can be stacked multiple times, with varying sets of scale parameters.

The final output of the MSSA blocks is distilled into a single vector $\mathbf{z}$ by summing each output vector element-wise. The resultant vector is normalised by dividing it by the square-root of the sequence length.

$$\mathbf{z} = \frac{\sum_{j=1}^{D} \mathbf{X}_{\cdot j}}{\sqrt{N}} \tag{4.11}$$

Where j is the j-th column vector in the output sequence. The average sentence vector $\mathbf{z}$ is then transformed into the semantic space of the concept embeddings

using a final feed-forward layer. This layer comprises two consecutive non-linear transformations, each with GELU activations and l2 normalisation, followed by a final linear transformation with no activation layer. It can be computed as:

$$\text{FinalFF}(\mathbf{z}) = \text{norm}(\gamma(\text{norm}(\gamma(\mathbf{z}\mathbf{W}_1 + \mathbf{b}_1))\mathbf{W}_2 + \mathbf{b}_2)) \tag{4.12}$$

Where $\mathbf{W}$ are parameter matrices and $\mathbf{b}$ are the bias terms for the final FF network. As in the other FF network, γ is the GELU activation function. Please see figure 4.1 for an illustration of the architecture. As in the original NCR model, the output embedding of the final feed-forward network is then multiplied by the global ancestry matrix to obtain the dot product scores between the sentence representation and all of the concept embeddings. The dot-product scores are then passed to a softmax function, as described in equation 4.2, to approximate the probability of each concept being correct given the output embedding.

Dropout regime See figure 4.3 for an illustration of the dropout regime for the attention blocks and the final FF network. The first dropout is applied to the input sequence. Within each batch, there is a 50% chance for dropout being applied to the batch of input sequences. This input dropout removes entire embeddings from a sequence with a probability of 20%. To reduce the chances of the total degradation of useful signals for shorter sequences, sequences with fewer than three tokens were exempt from the input dropout. The second dropout is a structural dropout function. Here there is a 25% chance of an attention head being entirely dropped out for an iteration. This is to prevent only a minority of heads being relied on for predictions, while encouraging all the heads to learn useful representations and to contribute to the model. The third dropout is a feature dropout applied after the ReLU activation function within the attention block's FF network. This dropout function removes random elements from the matrix with a probability of 10%. The fourth dropout is the same as the third except that it is applied following the ReLU activation function within the Final FF layer. Both of these dropouts degrade the signal within the FF networks to promote generalisability.

There are further feature dropouts applied inside the scaled self-attention heads, which are illustrated in figure 4.4. These are applied to the iteratively extracted sections of the $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ matrices. A further dropout is then applied to the attention scores, prior to them being scaled for the application of the softmax function.

Scale regimes The scale regime for the Multi-Scale Self-Attention in Guo *et al.* [78] was designed for much larger sequences, while the NCR takes a maximum length input of ten tokens. So a number of alternate scale regimes and block configurations were tested. Please see figure 4.5 for an illustration of the different combinations of self-attention head scaling parameters tested.

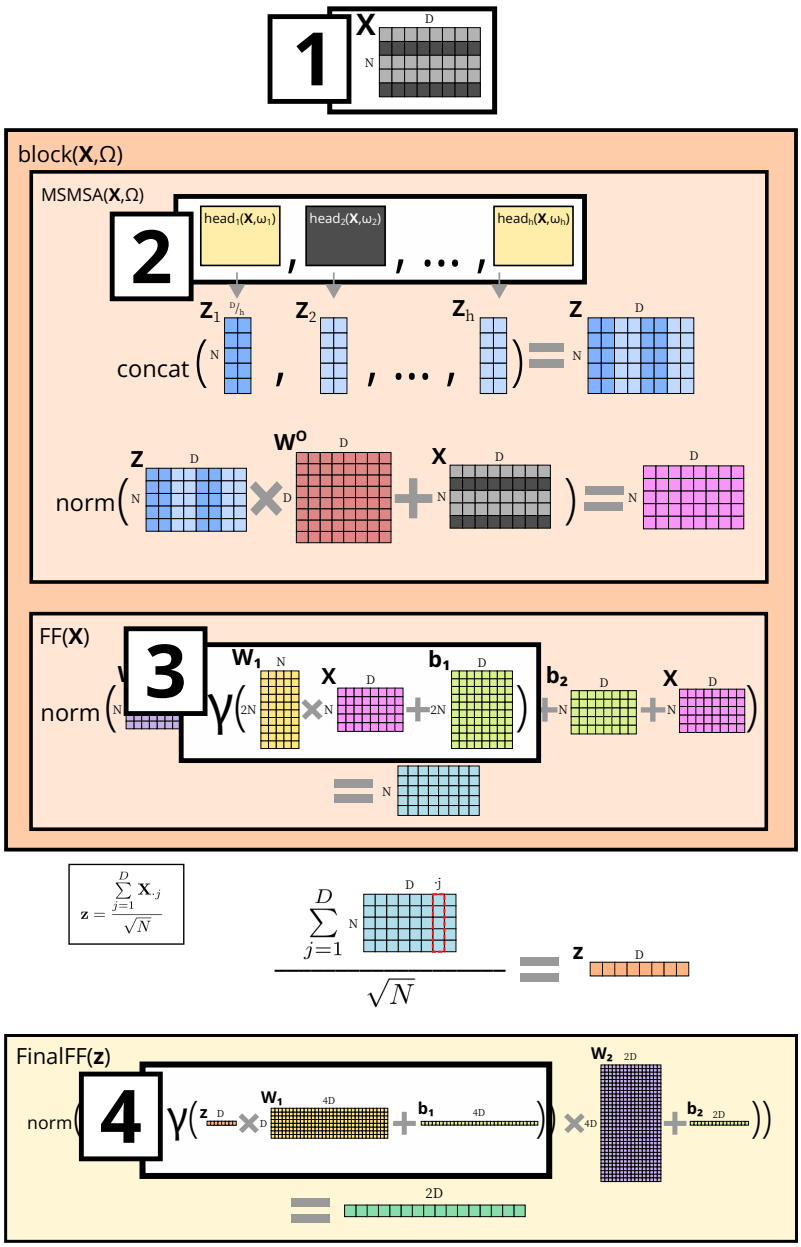

Figure 4.3: The dropout mechanisms applied to the scaled self-attention architecture during training. [**1**] Input dropout. Here there is a 50% chance of the sequences in an input batch, longer than two tokens, being subjected to a dropout with a probability of 20% (in effect a 10% dropout rate). This dropout replaces entire embeddings with zeros rather than random elements across all embeddings. [**2**] Attention head dropout. A structural dropout is applied to randomly replace entire attention head outputs with zeros, with a 25% probability. [**3**] Attention block FF dropout. A feature dropout, with a probability of 10%, is applied after the ReLU activation function. [**4**] Final FF dropout. Again, a feature dropout probability of 10% is applied following the ReLU activation function.

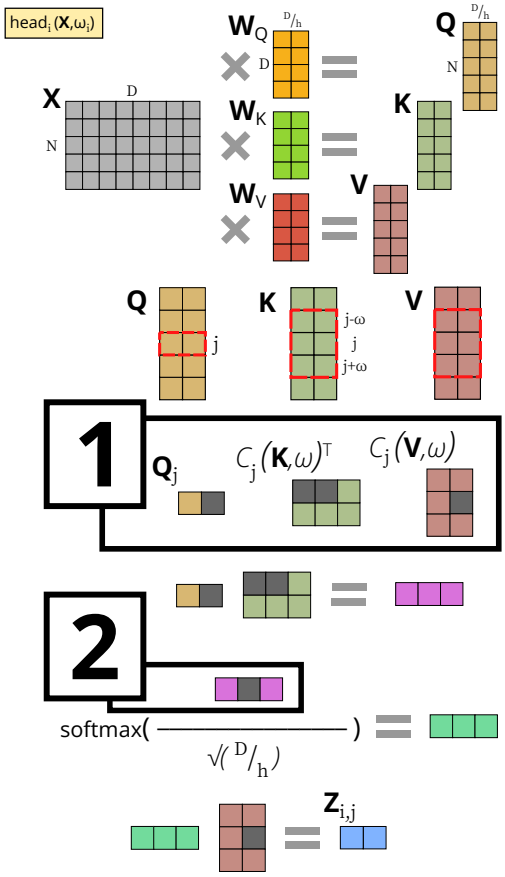

Figure 4.4: The feature dropouts applied inside the scaled self-attention heads during training. [**1**] A feature dropout with a probability of 10% is applied to each of the extracted context matrices. [**2**] Dropout, again with 10% probability, is applied to the iteratively extracted rows from the **Q**, **K**, and **V** matrices.

Normalisation and training stability A version of the MSSA was developed where, rather than applying a layer normalisation to the output of the addition of residuals (post multi-head attention and post-FF network), it instead applies layer normalisation to the inputs of the multi-head attention and FF network, as per Xiong *et al.* [235].

4.3.4 Ontologies

This work also explored extending the NCR model to incorporate multiple ontologies from various domains of knowledge. In particular, I wanted to see if the addition of terms from other ontologies, and thus providing more training data, would improve the performance of the model. I also wanted to understand if the semantic proximity of the domain ontologies used in combination with the Human Phenotype Ontology affected model performance. This was done by training the models using different combinations of the following OBO ontologies: The Cell Ontology (CL), the Human Phenotype Ontology (HPO), the Mammal Phenotype Ontology (MPO), and the

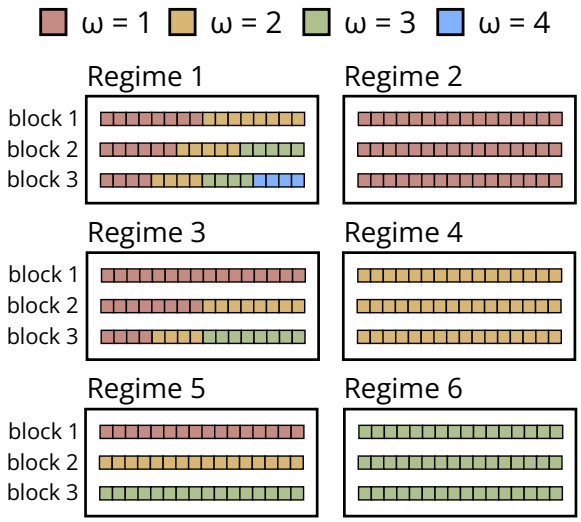

Figure 4.5: The scaling parameter combinations used by the blocks in different scaling regimes. Each regime was tested by starting with just the first blocks alone then progressively stacking further layers until finally all three blocks were together. This was to understand the influence that the synergy between the scaling parameters and the depth of the model had upon model performance. Figure originally published in Pigott-Dix & Davey [172] and reproduced here with permission.

Table 4.1: Ontology combinations used to train models.

Ontologies	Unique concepts	Training examples
Human Phenotype Ontology	16 059	35 969
Human Phenotype Ontology and Mammal Phenotype Ontology	29 370	75 298
Human Phenotype Ontology, Cell Ontology, and Ontology of Host-Pathogen Interactions	29 662	59 175

Ontology of Host-Pathogen Interactions (OHPI). Please see table 4.1 for an overview of the ontology combinations, the number of unique concepts represented, and their total number of training examples.

One set just contained the HPO, another both the HPO and the similar-domain MPO. The last set contains three semantically distinct ontologies: the HPO, the CL, and the OHPI. The NCR algorithm was adapted to incorporate multiple ontologies, and was trained to classify their terms. However, for the sake of evaluation on the HPO dataset, it could only return predictions from the HPO. If two ontologies shared any terms, they were merged into one, combining the concept's properties shared between the ontologies.

4.3.5 Training

Using Python 3.6.13 with TensorFlow 2.2.0 [142], 81 models were trained: 78 MSSA and three NCR. Due to a compiler issue, FastText could not be used with TensorFlow

2, as was used in Arbabi *et al.*'s work [7]. Instead, pre-trained ELMo [171] (v3) embeddings were incorporated via TensorFlow Hub.

Nine SAE models were trained using Python 3.9.12 and TensorFlow 2.9.1 due to a bug in the earlier TensorFlow version affecting the calculation of the means of only non-zero elements for each feature map.

Due to the memory constraints of the NVIDIA TITAN XP GPU, each training batch contained 256 samples. Training ceased and reverted to the best parameters for NCR and SAE models after five consecutive epochs where the loss had not improved. However, MSSA models reduced their learning rate by a fifth and resumed training under the same conditions. After the fifth learning rate change without improvement, training was stopped.

Initial learning rates were set at $\frac{1}{512}$ for both SAE and NCR models. MSSA models had a warm-up period, with the learning rate linearly increasing to $\frac{1}{512}$ over the first 20 000 iterations.

4.3.6 Evaluation

Once the models were trained, the optimal classification thresholds for each of the models were calibrated. To calibrate, 40 abstracts were randomly sampled from the 228 HPO Gold-Standard HPO-annotated PubMed abstract corpus, created by Lobo *et al.* [133]. They were then annotated by each model with each of the following confidence thresholds $n \in \{0.05, 0.10, \ldots, 0.95\}$. For each model, the threshold with the highest sum of both macro and micro F-scores was then set as the threshold parameter for the annotation of the remaining 188 abstracts.

To understand if there were any patterns or semantic themes shared between the false positives, true negatives, and false negatives, the character of the annotations predicted by the models, and the annotations that they failed to predict, were inspected.

4.4 Results

4.4.1 Identifying Human Phenotype Ontology Terms

Tables 4.2 and 4.3 display the full results for each NCR and SAE model, and each MSSA model respectively. In each table the best performing score for each metric is indicated with bold font.

When the NCR model with ELMo embeddings is trained with the HPO alone, the micro and macro F-scores are improved by no less than 4% when compared with the scores of the NCR model with FastText embeddings reported in Arbabi *et al.* [7]. However, the performance of the NCR model declines when additional ontologies are used in combination with the HPO. The SAE model trained with the more diverse combination of ontologies (HPO, CL and OHPI) accomplished a new

Table 4.2: The evaluation metrics for each NCR and SAE model benchmarked on the HPO annotation dataset. The bold font indicates the highest score for each metric. Table originally published in Pigott-Dix & Davey [172] and appears here with permission.

				Micro			Macro		
Ontology	Model	Filters	Threshold	Precision	Recall	F-score	Precision	Recall	F-score
HPO	NCR	1024	0.5	78.72	**72.75**	75.62	82.16	74.78	78.29
	SAE	1024	0.55	77.63	72.60	75.03	80.46	**75.04**	77.65
		1536	0.55	80.81	70.89	75.53	83.07	73.49	77.99
		2048	0.7	80.25	70.81	75.24	82.81	73.65	77.96
+ MPO	NCR	1024	0.5	74.12	66.12	69.89	77.38	70.75	73.91
	SAE	1024	0.85	82.28	52.20	63.87	84.28	57.09	68.07
		1536	0.5	76.17	65.23	70.28	78.68	68.90	73.46
		2048	0.5	74.62	61.73	67.56	76.78	66.19	71.10
+ CL	NCR	1024	0.75	**84.04**	64.71	73.12	86.01	66.61	75.08
+ OHPI	SAE	1024	0.6	83.05	70.07	**76.01**	85.13	72.82	**78.50**
		1536	0.5	80.55	69.69	74.73	82.18	72.07	76.79
		2048	0.65	82.94	65.15	72.98	**86.12**	66.96	75.34

state-of-the-art for neural dictionary methods. The best performing MSSA models were competitive with the other models, again with the use of more diverse domain ontologies as the training set leading to the better results.

Both tables 4.2 and 4.3 show that all models, when trained using the HPO and MPO combined, see a decrease in performance when compared to those trained using the either the HPO alone or jointly with the CL and OHPI.

4.4.2 Influence of the scale regime and the number of self-attention blocks on MSSA performance

For MSSA models trained using only the HPO, two-block models with scale regimes three and four performed better than the previous state-of-the-art. Of the MSSA models trained using the HPO, CL, and OHPI, the single-block models with scale regimes one, two, and three also performed better than the previous state of the art.

4.4.3 Exploring the concept embeddings

Figure 4.6 contains the density contour plots of the global concept embeddings for the best-performing SAE models trained using each of the three ontology combinations.

4.4.4 Impact of ontology concept properties

The plot in figure 4.7 shows the relationship between the number of natural language labels that a concept has, against its proportion of true positive, false positive, and false negative predictions. Figure 4.8 shows the counts of the prediction error classifications over the number of descriptions. There does not appear to be any particularly strong relationship between the number of labels and the true/false positives/negatives for concepts with fewer that nine descriptions. Concepts with more that nine descriptions are not well represented in the union of the predicted

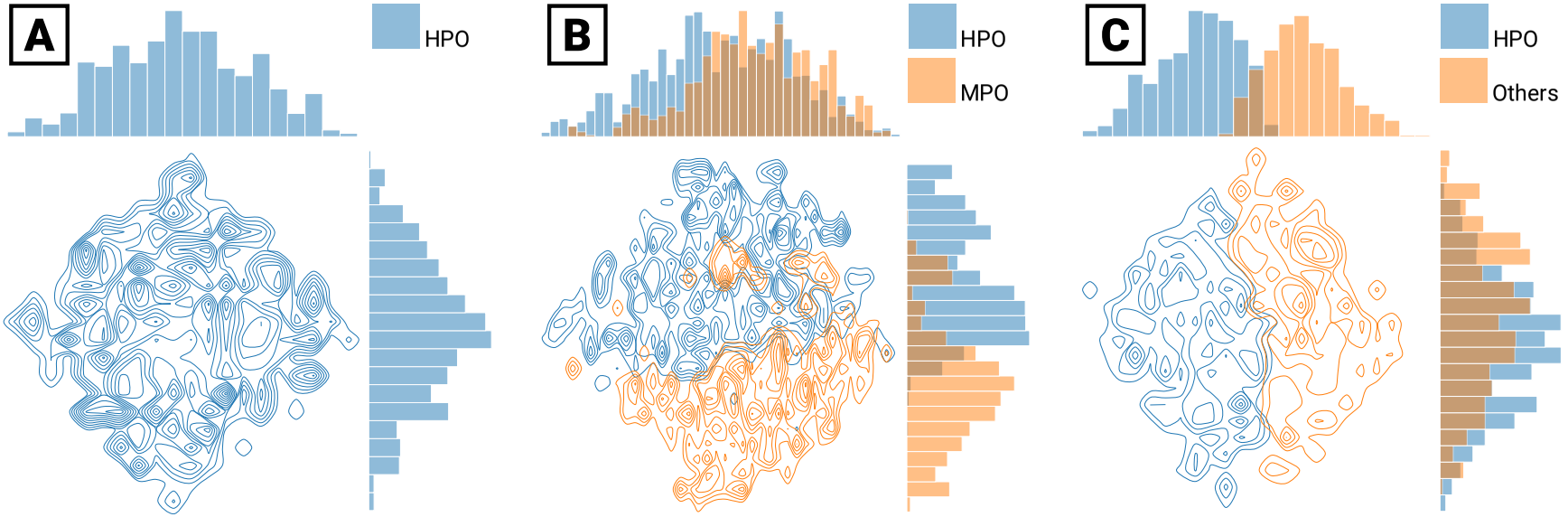

Figure 4.6: Density-contour plots of the concept embeddings from the best-performing SAE models trained using [**A**] the HPO (1536 convolutional filters), [**B**] the HPO and MPO (1536 filters), and [**C**] the HPO, CL, and OHPI (1024 filters). Principle component analysis created 50 composite dimensions from the original embedding dimensions, then a tSNE reduced these further to two dimensions. The explained variance for each set of embeddings' principal components were 76.02%, 56.05%, and 73.72% respectively. Figure originally published in Pigott-Dix & Davey [172] and appears here with permission.

Table 4.3: The evaluation metrics for each MSSA model benchmarked using the HPO annotation dataset. The best score for each metric are highlighted with bold font. Please note that the first blocks of the 3rd and 5th scale regimes were identical to the first block of the 2nd.

Ontology	Scale Reg.	Blocks	Threshold	Micro Precision	Micro Recall	Micro F-score	Macro Precision	Macro Recall	Macro F-score
HPO	1	1	0.55	77.96	69.02	73.22	79.76	71.74	75.54
		2	0.65	80.82	65.90	72.60	83.91	68.25	75.27
		3	0.5	77.16	64.41	70.21	79.33	66.39	72.29
	2	1	0.55	78.38	66.94	72.21	80.90	68.94	74.44
		2	0.55	78.77	69.62	73.91	81.06	70.89	75.63
		3	0.65	79.78	64.33	71.23	81.48	66.80	73.41
	3	1	-	-	-	-	-	-	-
		2	0.55	78.91	69.10	73.68	81.43	71.32	76.04
		3	0.4	71.96	64.41	67.98	75.33	67.77	71.35
	4	1	0.5	76.11	70.22	73.04	78.15	73.08	75.53
		2	0.45	75.04	**71.63**	73.30	78.16	**74.08**	76.06
		3	0.45	76.72	68.73	72.51	79.21	71.66	75.25
	5	1	-	-	-	-	-	-	-
		2	0.55	75.39	68.21	71.62	77.83	69.98	73.70
		3	0.2	55.84	55.55	55.69	59.20	58.02	58.60
	6	1	0.5	75.71	69.17	72.30	78.66	71.29	74.79
		2	0.65	80.30	64.04	71.25	82.08	65.97	73.15
		3	0.45	75.45	68.21	71.65	77.86	70.25	73.86
HPO + MPO	1	1	0.4	76.05	63.37	69.13	79.30	65.87	71.96
		2	0.25	68.68	63.51	66.00	70.94	66.94	68.88
		3	0.35	75.59	57.19	65.11	77.72	60.39	67.97
	2	1	0.45	78.63	61.36	68.93	79.86	65.02	71.68
		2	0.35	72.00	64.33	67.95	73.72	66.45	69.90
		3	0.2	66.79	54.06	59.75	69.52	57.65	63.03
	3	1	-	-	-	-	-	-	-
		2	0.4	75.00	65.00	69.64	76.89	68.05	72.20
		3	0.25	70.36	53.39	60.71	73.01	58.28	64.82
	4	1	0.25	70.53	57.04	63.07	73.95	61.73	67.29
		2	0.35	73.47	63.51	68.13	75.17	65.44	69.97
		3	0.25	68.27	63.29	65.69	69.49	65.88	67.64
	5	1	-	-	-	-	-	-	-
		2	0.45	78.69	62.40	69.60	79.68	64.88	71.52
		3	0.35	75.28	55.99	64.22	78.64	59.19	67.54
	6	1	0.3	74.06	64.41	68.90	76.48	67.70	71.82
		2	0.4	77.17	56.37	65.15	80.85	59.64	68.65
		3	0.05	53.00	50.71	51.83	57.29	54.20	55.70
HPO + CL + OHPI	1	1	0.45	79.13	69.47	73.99	81.25	71.95	76.31
		2	0.65	**84.44**	65.08	73.51	**86.19**	67.50	75.70
		3	0.3	72.53	61.73	66.69	76.70	63.93	69.74
	2	1	0.5	80.37	68.58	**74.01**	83.21	70.72	**76.46**
		2	0.45	80.75	62.47	70.45	82.79	64.29	72.38
		3	0.5	82.41	55.47	66.31	83.37	57.74	68.23
	3	1	-	-	-	-	-	-	-
		2	0.5	79.81	67.09	72.90	81.52	68.61	74.51
		3	0.2	57.31	52.27	54.67	60.94	52.83	56.60
	4	1	0.45	80.07	67.91	73.49	81.38	69.07	74.72
		2	0.65	83.53	62.70	71.63	85.06	64.95	73.66
		3	0.45	79.40	66.87	72.59	80.38	68.40	73.91
	5	1	-	-	-	-	-	-	-
		2	0.5	80.39	65.00	71.88	81.29	66.02	72.87
		3	0.05	0	0	0	0	0	0
	6	1	0.5	81.51	67.61	73.91	83.29	69.65	75.86
		2	0.45	82.49	64.56	72.43	82.43	66.41	73.56
		3	0.45	77.99	65.97	71.48	79.67	67.61	73.14

concepts and ground truth concepts.

A linear regression with log-transformation was used to fit the relationship between the proximity of a concept's nearest neighbour with its true positive rate. If a concept in close proximity interferes with classification, we should expect to see

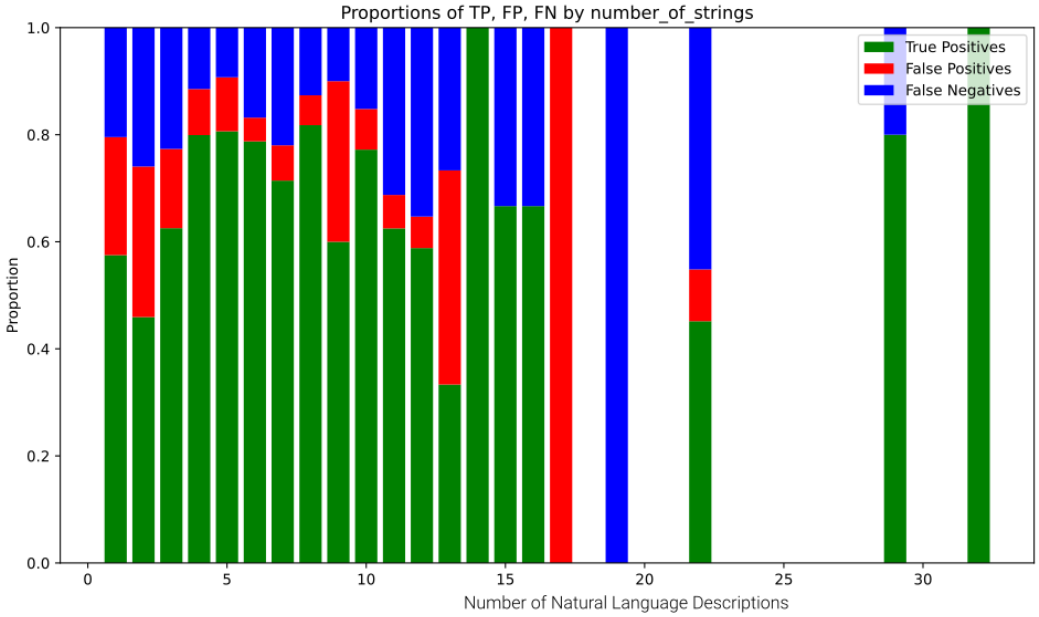

Figure 4.7: A plot of the cumulative proportion of true positives, false positives, and false negatives of all HPO concepts with each particular number of natural language descriptions.

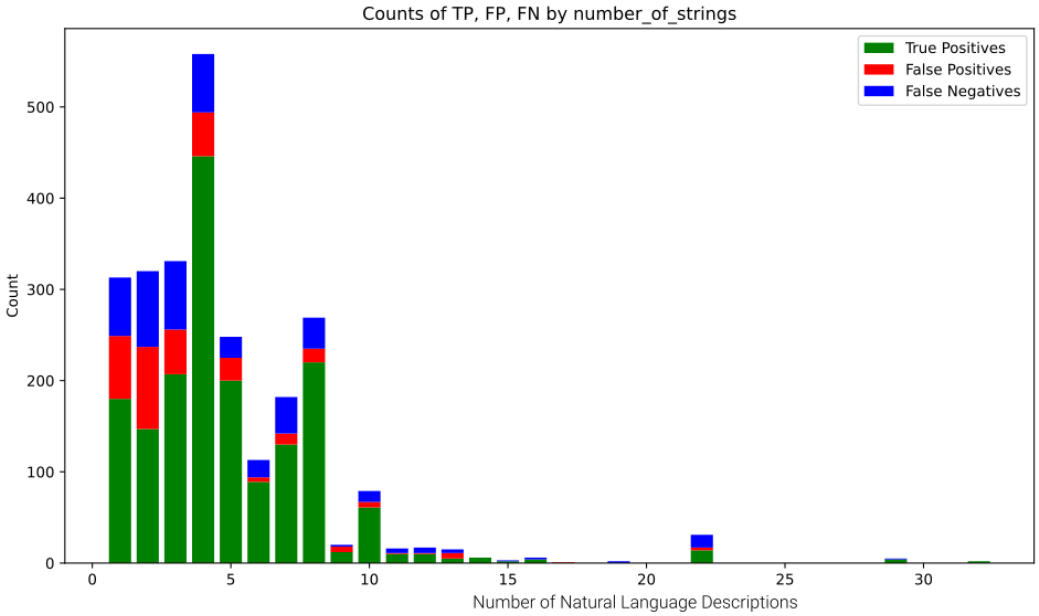

Figure 4.8: A plot of the cumulative counts of true positives, false positives, and false negatives of all HPO concepts with each particular number of natural language descriptions.

a positive relationship between the true positive rate and concept proximity. The results of the regression rounded to three significant figures were as follows: Slope: 0.0221; Intercept: -0.355; R-squared: 6.16×10^{-5}; p-value: 0.862. The p-value

and R-squared value suggests that the regression does not classify this relationship well. This suggests that there is not a strong link between concept proximity and classification error.

4.5 Discussion

4.5.1 Ontology-based concept extraction

Concept overlap heuristics This work has highlighted that adjustments to both the NCR annotation heuristics and architecture are required in order to accommodate multiple domain ontologies. The current model annotation heuristics prevent any annotations overlapping. While it was believed that it did not affect the current assessment, as the model was prevented from tagging terms outside of the HPO, Luo *et al.* [136] found that the gold-standard PubMed abstracts actually contained instances of overlapping HPO concepts. As a result, the heuristics may have been excluding valid terms identified by the model, to the detriment of model performance. In future, the heuristics should be adapted with a view to incorporating different domains of knowledge by extracting all terms exceeding the confidence threshold regardless of overlap.

Figure 4.7 shows that the proportion of true positives is higher for concepts with more than three natural language descriptions. However, figure 4.8 illustrates that concepts with more than eight descriptions are not well represented in the benchmarking dataset and model annotations, which may mean that any error rates for these concepts are unreliable. The regression carried out between true positive rates and the number of natural language descriptions that an ontology term has could not establish a relationship between the two.

Combining multiple ontologies Architectural issues arise as a consequence of incorporating multiple ontologies together. Training models using combinations of ontologies with overlapping domains appears to detrimentally impact model performance, as demonstrated by the evaluation metrics in tables 4.2 and 4.3. In the case of the HPO and MPO, neither ontology subsumes the other, and their terms do not become combined. This means that parallel ancestry trees are represented in the ancestry matrix, while a lot of the natural language descriptions are highly similar for concepts in each ontology. This likely leads to Gordian parameter weights, as the models try to reconcile concepts with near-identical natural language descriptions that, due to their alternate networks of semantic inheritance, are positioned far from each other in embedding space. In figure 4.6 panel **B**, it is apparent that the concept embedding space is congested with lots of overlap between the HPO and MPO concepts. For an illustration of the ancestry matrix influences the classification of similar concepts from two different ontologies see figure 4.9.

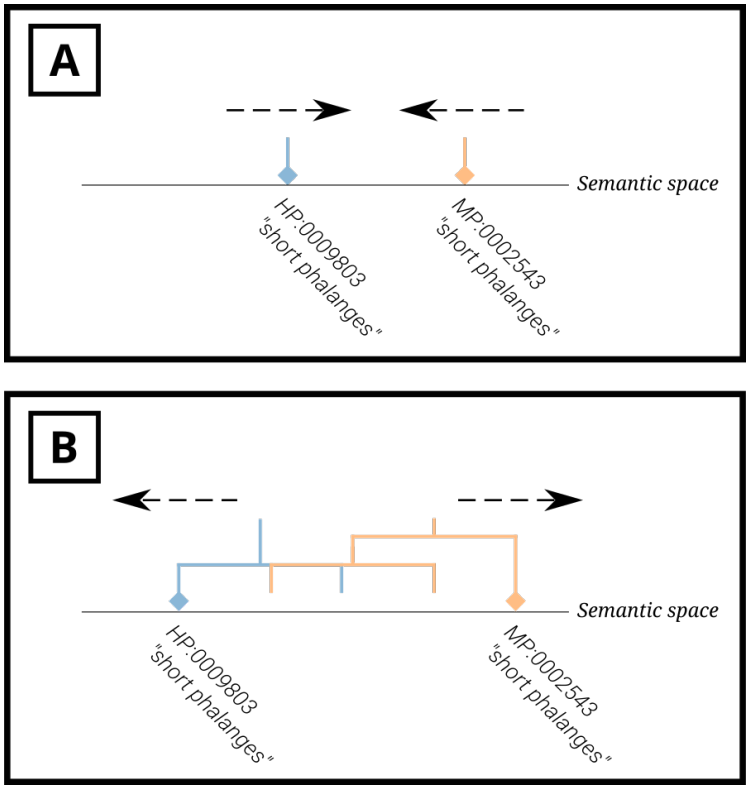

Figure 4.9: An illustration of how overlapping domains may be influencing classifier performance. **A** The text encoder learns to create similar representations for similar concepts from different ontologies. **B** However, the separate ancestry structure aggregates information differently between the two ontologies, resulting in the final representations being separated in representational space.

The explained variance percentages from the PCAs used to create this figure suggest that to discriminate between these concepts the model needs to attend to a greater number of features. This may be partially resolved by removing the ancestry matrix from the model when using such closely aligned domains. However, this would also remove the benefits to performance of sharing semantic information between related concepts, found by Arbabi *et al.* [7]. Additionally, Althubaiti *et al.* [4] found that when creating ontology embeddings for a downstream classification task, the combination of multiple ontologies led to superior performance.

The evaluation metrics in table 4.3 show that the MSSA model is improved by being trained on a combination of ontologies with more diverse domains. This is likely happening for two reasons. Firstly, transformer-based models benefit from larger volumes of training data; although this extra training data may not be semantically relevant to the HPO annotation task, the general syntactic patterns of the language may well be. Secondly, the aforementioned distinctness between domains limits the chance of semantic overlap between terms from different ontologies.

Considering its superior performance, relatively low number of parameters, relatively quicker training times, and the simplicity of adding filter kernels and modifying

the SAE mechanism, the SAE is likely the best architecture for continuing to develop a multi-domain ontology-based text classifier.

4.5.2 Adapting transformer-based architectures for training-data-poor scenarios

The combination of scaling and dropout regimes applied to the transformer-based MSSA led to competitive performance when compared with the CNN architectures. While typical transformers require millions of training examples, the work here corroborates that of Guo *et al.* [78]: that they can be adapted to perform well with orders of magnitude fewer training examples. This may have implications for transformers used for machine translation in low-resource scenarios, such as Nepali or Sinhala to English translation [80].

Chapter 5

Incorporating More Sophisticated Symbolic Information into Neural Dictionaries

5.1 Knowledge graphs as model architecture

As mentioned at the start of the chapter 2, the field of AI research has long been regarded as split between those that see knowledge representation as primarily symbolic and those who view knowledge as distributed across neural structures [204, 115]. However, recently Neuro-Symbolic AI which combines the two paradigms, has garnered interest, particularly in the bio-medical domain [32, 31, 134]. Bio-medical research is well placed to take advantage of Neuro-Symbolic AI due to the extensive work developing ontologies [9, 45, 211, 187, 17, 198, 170], linked data resources [28, 103], and the recent emphasis on combining linked data into knowledge graphs for task-specific heterogeneous domain knowledge representation [205, 132]. Lobentanzer *et al.* [132] observed two key characteristics of knowledge graphs. Firstly, that knowledge graphs explicitly model heterogeneous knowledge. Secondly, that they can be built for specific tasks. As a result Lobentanzer and colleagues [132] argued that they are particularly amenable to explainable AI applications. This development was foreshadowed by LeCun and colleagues [121] who stated in 1998 that "a good way to incorporate knowledge [into a deep neural network] is to tailor its architecture to the task".

Graph Neural Nets are statistical deep learning models that aim to model structured data or structured representations of knowledge. They have found broad application across the life sciences, in particular towards drug discovery [246], predicting drug-drug interaction [127], and bioinformatics [239]. These methods typically

encode graph sequence information using random walks [169, 182], graph convolutions [249, 241], or triple validity prediction [242], to aggregate structured symbolic information into vertex and edge representations. A triple refers to an edge and two vertices from a graph. Transformers can be regarded as graph neural nets as the attention mechanisms model a fully connected graph between all the elements of a sequence, and update their representations using a form of message passing between the elements [194]. Recently, sophisticated attention-based transformer-style graph neural nets have been increasing in prominence [226, 243, 95, 244].

Among them, Hu *et al.* [95] reformulated the transformer [221] to create the "Heterogeneous Graph Transformer" (HGT). The original transformer architecture was reformulated with sets of specific parameter weights for each source vertex-type, target vertex-type, and edge-type. To calculate the attention scores, the source vertex embeddings are transformed into a Key matrix and the target vertex embeddings into a Query matrix. The attention scores are obtained with the dot product of the Key and Query matrices, moderated by an edge-specific matrix. Rather than calculating the softmax within each head as is done in a typical transformer, the scores from each head are concatenated together. Then the softmax is calculated across all heads with respect to each target vertex. To pass messages between the source and target vertices, the source-vertex-specific parameter weights transform the source embedding into a value vector, that is further transformed by edge-specific weights. Finally, the softmax scores are used to aggregate the messages before a target-specific parameter weight transforms these aggregated messages into the target vertex representation space. To pass messages over multiple hops requires the stacking of HGT layers.

Yun *et al.* [243] (refined in [244]) proposed the "Graph Transformer Network" (GTN) to represent heterogeneous graphs. Rather than passing information between vertices along single-hop edges, or predefined combinations of edges (termed meta-paths), this model uses a soft-attention mechanism to learn useful meta-paths. This model was found to not only recreate expertly handcrafted meta-paths from previous research, but also novel combinations of edges. For example, when classifying the research areas of authors in a paper, conference, and author DBLP dataset, in addition to the model learning the [author → paper → conference → paper → author] meta-path, replicating typical hand-crafted meta-paths, it also learned a novel [conference → paper → conference → paper → author] meta-path.

As discussed in the previous chapter, Arbabi *et al.* [7] created the Neural Concept Recogniser (NCR), a neural dictionary model, for identifying ontology terms from unstructured scientific literature. The NCR uses a CNN to extract semantic signals from word embeddings representing unique ontology identifiers. The similarity of the vector of the text input's semantic signals was compared with the embeddings of the concepts. Due to the limited number of natural language representations contained within an ontology, the model used an hierarchical aggregation via an ancestry

matrix to pool representations between related concept embeddings. This pooling of information was found to improve the performance of the model at semantic annotation, when compared to the models without. The hierarchical aggregation of concept representations via an ancestry matrix can be regarded as a simple GNN which both statically models the relationships between concepts and passes messages between them.

5.1.1 Contribution

While there has been research into "injecting" knowledge graph embeddings with information from the rich semantic implicit knowledge contained within large language models [166, 250, 238, 227], the work in this chapter takes an alternate perspective, and instead tries to inject the structure of a knowledge graph into a deep neural text classifier. Here, the concept of augmenting neural architecture with symbolic knowledge representation, initially explored by Arbabi *et al.* [7], is expanded. In addition to the data contained within the Human Phentotype Ontology (HPO), I incorporated information from a complementary knowledge graph. I used a subset of the DisGeNET knowledge graph [177, 173], which represents gene-disease associations and their related genes, genetic variants, and proteins. This subset included all gene-disease associations linked to an HPO concept and their associated genes and proteins. See figure 5.1 for a visualisation of the schema. I trialled three approaches and compared them with a control.

- **Squeeze-and-Excite (SAE) CNN Text Classifier (Control)**. This control method is a text classifier trained to identify HPO terms that correspond to their natural language text labels. No symbolic information is incorporated into the model.

- **SAE CNN Text Classifier sharing embeddings with Meta-Path Transformer (MPT)** . In this setup, the SAE CNN text classifier is trained as above, but the concept embeddings are shared by an MPT that is trained to perform walk-validity classification.

- **SAE CNN Classifier integrated with MPT**. This approach involves integrating the MPT directly into the architecture of the SAE CNN text classifier. The intention is that the MPT will allow the text classifier to learn useful structural information from the knowledge graph, providing additional context for the classifier.

- **SAE CNN Classifier with MPT and multi-task training**. In this approach, the SAE CNN is integrated with the MPT and trained for text classification, but additionally, the MPT component is also trained to predict walk validity.

Both of the multi-task training versions were trained with two different regimes: one, with the walk validity classification as a pre-training stage followed by training on the text classification task; the other, with simultaneous walk validity and text classification training objectives. These models were validated against Lobo *et al.*'s [133] corpus of PubMed abstracts, annotated with HPO terms, which was used in the previous chapter.

Pre-training the concept embeddings used by the SAE CNN text classifier, by training the MPT on the walk validity-task, led to a slightly improved performance over the SAE CNN classifier without pre-training, and achieved the best performance metrics overall. The text classifiers that incorporated the MPT did not perform as well.

Investigation of the multi-hop meta-paths learned by the MPT suggested that the DisGeNET knowledge graph was not an optimal structure for learning graph message passing for the task of concept identification in text. However, this investigation demonstrated the utility of designing architectures so that they have in-built *a priori* interpretability. The code for this chapter is available here: https://github.com/lorcanpd/MetAnn.

5.2 Related Work

Combining language models and structured knowledge representation. TransR by Lin *et al.* [128] was the first knowledge graph embedding method to explicitly model entities and relations in different representational subspaces. Here, head and tail entity embeddings were transformed from the entity space into a separate relation space. Then relation arithmetic is applied to the transformed representations before being transformed back into entity space. This was a divergence from the typical $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$ formulation of knowledge graph embedding arithmetic, where $\mathbf{h}$, $\mathbf{r}$, and $\mathbf{t}$ represent the head entity, relation and tail entity embeddings respectively. Instead it was formulated as $(\mathbf{hM} + \mathbf{r})\,\mathbf{M}^{-1} \approx \mathbf{t}$. In addition, a variation of the model, CTransR, clustered the valid head-tail entity pairs into groups and then learned specific transformation parameter weights for each cluster. Both methods (CTransR in particular), outperformed previous methods at triple validity classification and link prediction tasks.

Che *et al.* [40] took a similar approach. They created a knowledge graph completion model that learned to transform the embedding of a head vertex into the embedding of a tail vertex from a valid $\langle head, edge, tail \rangle$ triple. As in the previously mentioned work, the vertices were embedded in one space, however the neural network parameters were regarded to represent the edges between them instead of the relations having embeddings themselves. Conceptually, Hu *et al.* [95] and Yun *et al.* [243, 244] are essentially more complex iterations of this theme. Inspired by the wildly successful transformer, a version of the self-attention mechanism aggregates

edge-specific information to vertices' embeddings through edge-specific transformations.

Zhang *et al.* [250] used a transformer-based BERT model to imbue knowledge graph entity and relation embeddings with semantic information. This was done by transforming text descriptions of relations and entities into embeddings using BERT. These embeddings were then converted into the knowledge graph embedding space with a non-linear transformation, as a part of a pre-training process where the distance between the sum of the head entity and relation embeddings and the tail entity embedding was minimised, the typical embedding arithmetic training objective. The pre-training process is followed by using BERT and the trained non-linear transformation to create embeddings for all of the entities in the knowledge graph. Then the knowledge graph is trained using a typical knowledge graph embedding process, whereby the embeddings are used to train a classifier that determines whether a triple is valid or not. This is intended to capture structural information inside the knowledge graph embeddings. This pre-training methodology was incorporated with previous knowledge graph embedding methods and found to improve performance on benchmarking tasks, especially in low resource settings.

Yao *et al.* [238] also employed a BERT model for knowledge graph embedding. However, this model used sentences representing relation triples in the case of the triple validity task, and head-tail pairs for the relation classification task. This comprised a special $[CLS]$ token followed by tokenised knowledge graph entity text descriptions, separated by special separation tokens ($[SEP]$), and with the addition of a segment specific vector to encode the tokens with information on their particular segment, like the positional encoding used by the original transformer. Both triple validity and relation prediction tasks passed the $[CLS]$ embedding from the output layer to task specific classification layer. This model outperformed previous knowledge graph embedding models, again performing particularly well in low resource settings. The authors suggest that the volume of external data contained within the language model helps to mitigate the data sparsity of the knowledge graph.

KEPLER, by Wang *et al.* [227], is a transformer model trained with the joint training objectives of knowledge graph embedding and masked language modelling. Multiple approaches were explored for knowledge graph embedding. Zhang *et al.* [250] encoded entity descriptions as embeddings using the transformer and had an embedding for each relation type, or also encoding the relation descriptions using the transformer. Yao *et al.* [238] jointly encoded the text of head-tail pair descriptions and classified the relation type as the training objective. Additionally they trained a transformer to predict the masked tokens in a sequence.

Using ontologies to enrich entity representations. Regarding specifically enriching graph embeddings with information from ontologies, D'Amato *et al.* [52] extracted axioms from ontologies to "inject" further structured information into a

knowledge graph in order to better embed a bioinformatics knowledge graph.

Smaili, Gao, and Hoehndorf developed OPA2Vec [210], a method for creating numerical vector representations of ontologies. This works by transforming the information contained in an ontology into a corpus of text sentences, made up of ontology entity ids, relationship types, and natural language. Then a word2vec model is trained on the corpus, creating embeddings for each token. This results in each relationship, entity class and word in an ontology having a numerical representation. OPA2Vec was used by Althubaiti *et al.* [4] to create feature representations to use as inputs to train a classifier that determine whether genes were cancer drivers.

Exploiting graph structure for text classification tasks. A review of graph neural networks applied to text classification tasks by Malekzadeh *et al.* [141] does not discuss any methods that incorporate knowledge graph structure inside a classifier. In most cases text is converted into a graph and then the task is treated as a node classification problem. Huang *et al.* [96] and Zhang *et al.* [248] did this by representing co-occuring words with a graph. Liang *et al.* [125] created Sentic-GCN, a graph convolution model for sentiment analysis. It builds a dependency graph of a sentence, and uses word sentiment scores from SenticNet to weight the message passing between nodes. Zhang and Qian [247] built lexical and syntactic graphs, and then used them in combination with graph neural nets and pre-trained language models for text sentiment classification. Tang *et al.* [217] created a sentiment classification model that employed both a traditional transformer in concert with a modification of the transformer. The modification replaced the attention scores with an adjacency matrix based upon the word dependency graph. However, in their discussion Tang *et al.* [217] alluded to the potential of incorporating domain specific knowledge.

Explicitly incorporating knowledge graphs for text classification tasks. Lan *et al.* [117] combined knowledge graphs with word co-occurrence graphs to classify Chinese medical documents. Text in a document is matched to knowledge graph entities to extract a subset of a medical knowledge graph. This knowledge graph subset is combined with the word co-occurrence graph to create a joint graph, which is then passed to a graph neural net for classification.

5.3 Methods

Here the training data construction is described, followed by the different types of models and any applicable graph sampling algorithms. Finally, the training and evaluation processes are described.

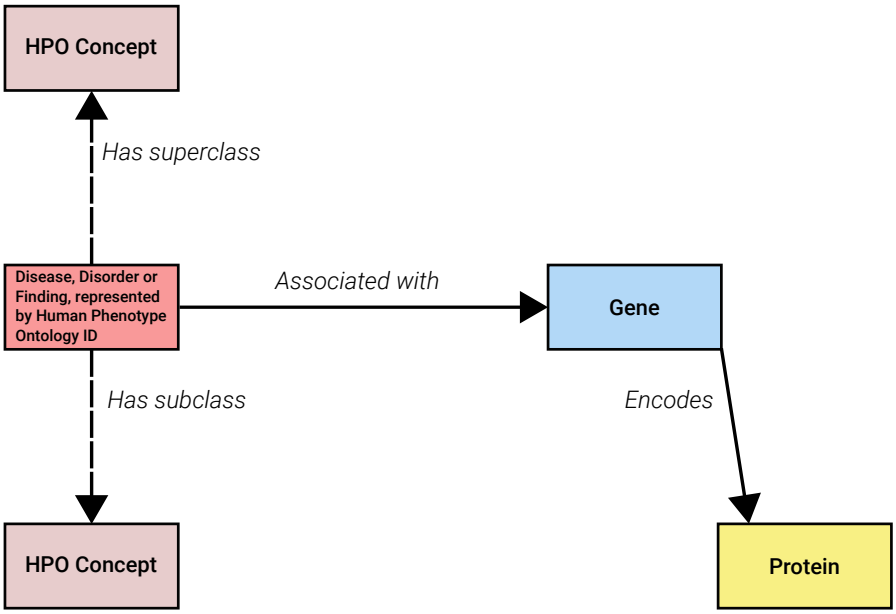

Figure 5.1: Schema of the merged Human Phenotype Ontology and Gene Disease Associations, including corresponding Genes and Proteins, from the DisGeNET knowledge graph. The reified edge *Gene Disease Association* has been simplified into an *Associated with* edge, rather than being represented by two edges and a vertex as in the knowledge graph.

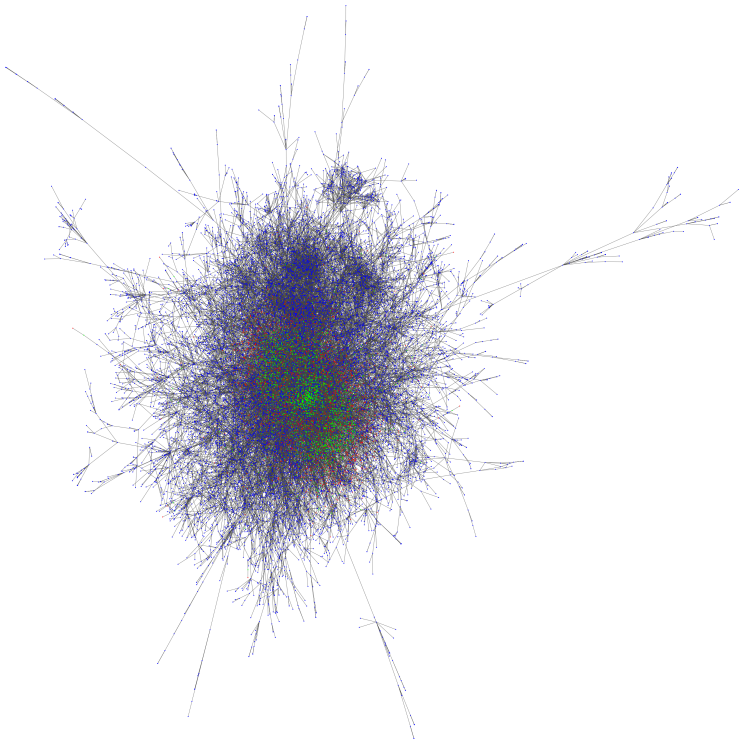

Figure 5.2: Graph of combined HPO terms (blue), their associated genes (green), and the proteins (red) that the genes encode for. Plotted using the Python package igraph (version 0.10.5) [50].

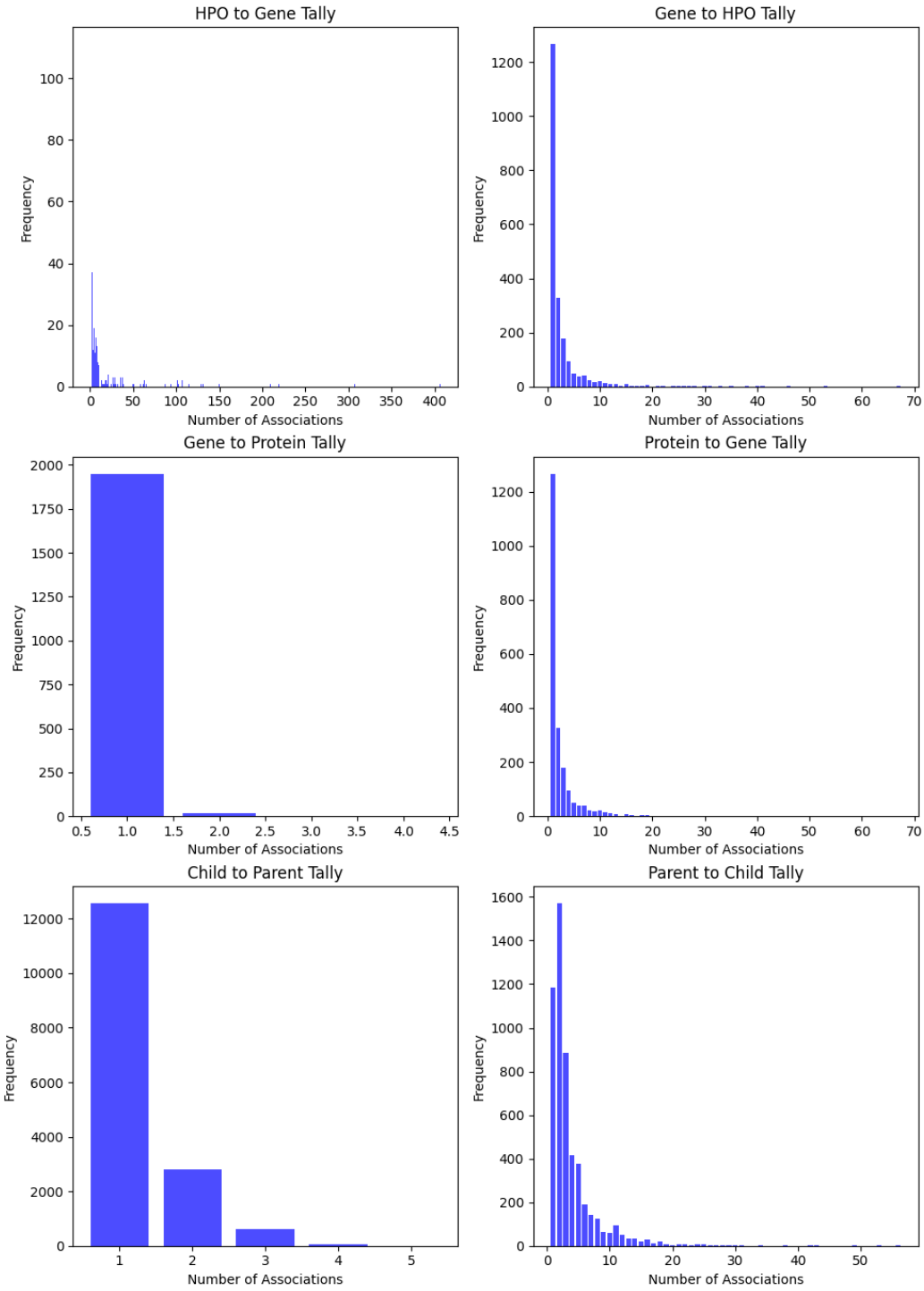

Figure 5.3: The frequency of specific edge-type associations between individual vertices in the graph. Reading from left to right, top to bottom: A tally of the number of genes associated with each HPO concept, the number of HPO terms associated with each gene, proteins associated with genes, genes with proteins, the number of parents an HPO term has, the number of children an HPO terms has.

5.3.1 Data

Combining the HPO and DisGeNET A sub-set of the RDF triples that represent Gene-Disease and Variant-Disease associations were extracted from the DisGeNET [177] knowledge graph using the Python package RDFLib (version 6.3.2) [76]. This sub-set comprised all "Disease, Disorder, or Finding" vertices with an associated Human Phenotype Ontology (HPO) ID, all "GeneDiseaseAssociation" (GDA)vertices related to the HPO vertices, all "Gene" vertices referred to by these GDA vertices, and all "Protein" vertices encoded by these "Gene" vertices. The graph data was further enriched by the addition of subclass and super-class relationships from the HPO. For a visualisation of the schema of this subset see figure 5.1. Despite simplifying the schema to three edge-types, the graph is still large and complex as can be observed in figure 5.2.

From this graph, parent-to-child dictionaries were created for each relation type. These relation types were: '*gene encodes protein*', '*HPO term is a subclass of HPO term*' and a single dictionary was used to represent the '*GDA refers to HPO term*' and '*GDA refers to gene*' relations as a single '*HPO is associated with gene*' relation. Dictionaries of the inverse relationships were also generated: '*protein encoded by gene*', '*HPO term is a super-class of HPO term*', and '*gene is associated with HPO*'. As a disease may have many associated genes, and a gene may encode for more than one protein, each parent-key may have multiple child-values. Figure 5.3 displays the tallies of the total number of associations of each type for each source vertex. This shows that almost every gene codes for a single protein in the graph, while conversely proteins are more likely to be encoded by multiple genes. Similarly, the tallies indicate that a HPO vertex is much more likely to have more children vertices than parents.

For the purposes of efficiently sampling the graph, an all-relationship dictionary was constructed from the edge-type-specific dictionaries, and then the Python package NetworkX (version 3.1) [81] was used to calculate the combined in- and out-degree centrality of every vertex.

Text data As in the previous chapter, a training dataset was constructed from the information contained within the HPO ontology. This dataset contained 39 766 pairs of HPO IDs with their name or one of their synonyms.

TensorFlow version 2.9 [142] no longer supports the use of the ELMo model from TensorFlow-hub. I managed to build a compatible version of the FastText package, and so this was used to produce the word embeddings used by all of the models in this chapter.

5.3.2 SAE CNN classifier

The text encoder replicates the Squeeze-and-Excite CNN from the previous chapter, minus the ancestry matrix that was used for pooling information between related HPO terms (Figure 5.4). Text sequences, represented by word embeddings, are fed into a one-dimensional CNN. A number of filter kernels create activation map vectors for each filter type. A maxpool layer distils each feature map into their strongest signals. The average signals of the non-zero elements of each feature map vector are also aggregated (Equation 4.3). These average signals are then compressed and decompressed by a fully-connected layer with a ReLU activation after the first layer (Equation 4.4). These re-expanded signals weight the strongest signals from the maxpool, which in turn are multiplied against the matrix of vertex embeddings. Finally, a softmax layer transforms logits into the classification probability for each vertex in the graph.

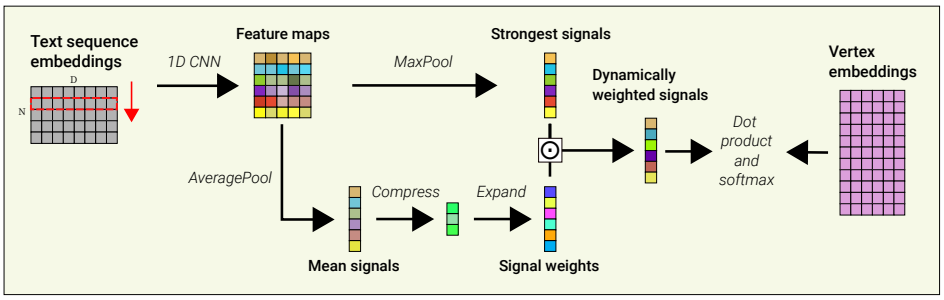

Figure 5.4: Overview of the Squeeze-and-Excite Convolutional Neural Net text classifier architecture.

5.3.3 Walk-validity classifier

The walk-validity model is comprised of two parts: an MPT block, and a classification layer. The MPT is comprised of a multi-headed Meta-Path Attention layer followed by a row-wise feed-forward network, then by a multi-headed self-attention layer, and finally another row-wise feed-forward network. See figure 5.5 for an illustration of the walk validity architecture.

Multi-headed self-attention The self-attention layer is described again here, as the meta-path attention layer is a direct modification of it. A self attention layer is comprised of H attention heads. Given an input matrix of vertex embeddings $\mathbf{X} \in \mathbb{R}^{l \times d}$, where l is the maximum number of vertices in a sequence and d is the dimensionality of the embeddings, an attention head projects the input into three representational subspaces through linear transformations with parameter weights,

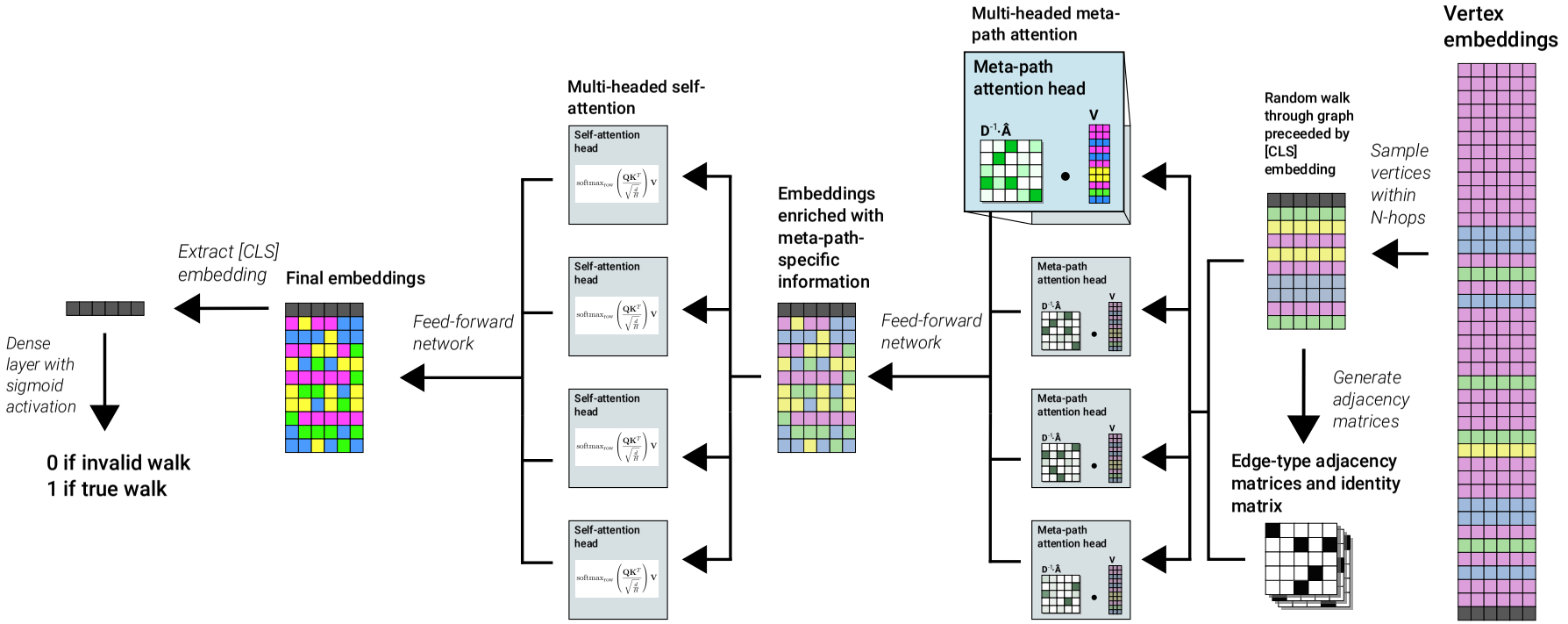

Figure 5.5: The Walk Validity architecture. Vertex embeddings are sampled using a random walk through the graph, preceded by a special $[CLS]$ embedding. Adjacency matrices for this subset of the graph are constructed for each edge-type. Both the vertex embeddings and their adjacency matrices are passed to a multi-headed meta-path attention layer, which updates the embeddings using message passing along multi-hop meta-paths (this is illustrated by figures 5.6 and 5.7). These are further transformed using a feed-forward network, and then passed to a conventional multi-headed self-attention layer, again followed by a feed-forward network. The embedding corresponding to the special $[CLS]$ embedding is extracted and passed to the final classification layer, returning a one if the walk is valid or a zero if it is invalid. Layer normalisation and the addition of residuals are omitted for simplicity

$\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$ ($\mathbf{W}_{q|k|v} \in \mathbb{R}^{l \times \frac{d}{H}}$):

$$\begin{aligned} \mathbf{Q} &= \mathbf{X}\mathbf{W}_Q, \\ \mathbf{K} &= \mathbf{X}\mathbf{W}_K, \\ \mathbf{V} &= \mathbf{X}\mathbf{W}_V. \end{aligned} \tag{5.1}$$

Self-attention scores, computed from $\mathbf{Q}$ and $\mathbf{K}$, moderate the message passing between the vertices of $\mathbf{V}$:

$$\begin{aligned} &\text{Input: } \mathbf{Q}, \mathbf{K}, \mathbf{V} \\ &\mathbf{Z} = \text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \\ &\text{where:} \end{aligned} \tag{5.2}$$

$$\mathbf{Z} = \text{softmax}_{\text{row}} \left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{\frac{d}{H}}} \right) \mathbf{V} \tag{5.3}$$

The matrix product of $\mathbf{Q}$ and $\mathbf{K}^T$ represents the pairwise interactions between vertices. Subsequent scaling with a row-wise softmax function gives the attention scores. The head output $\mathbf{Z}$, the product of the attention scores and $\mathbf{V}$, is a relevance-weighted aggregation of messages between vertices, with respect to the recipient vertices.

The output of each attention head is concatenated together and then linearly transformed by the parameter weight $\mathbf{W}_0$:

$$\text{SelfAttentionLayer}(\mathbf{X}, H) = \text{concat}\left([\text{head}_0(\mathbf{X}),\ \ldots\ ,\ \text{head}_{H-1}(\mathbf{X})]\right) \cdot \mathbf{W}_0 \tag{5.4}$$

Row-wise feed-forward network The row-wise feed-forward network (FFN) is used to non-linearly transform the outputs of the attention layers. It applies the same transformation identically to each column of a matrix but separately to each row. It can be described as follows:

$$\text{FeedForwardNetwork}(\mathbf{X}) = \mathbf{W}_2 \cdot \text{LayerNorm}\left(\gamma\left(\mathbf{W}_1 \cdot \mathbf{X} + \mathbf{b}_1\right)\right) + \mathbf{b}_2 \tag{5.5}$$

Both $\mathbf{W}$ are parameter matrices, both $\mathbf{b}$ are bias vectors, and γ represents the GELU activation function [88].

Meta-path attention layer Here the Graph Transformer Network from Yun *et al.* [243, 244] is reformulated in the style of the original transformer encoder from Vaswani *et al.* [221]. The meta-path attention layer modifies the self-attention layer to leverage multiple hops of a set of pre-defined edge matrices. An attention head in this layer softly selects N adjacency matrices from the set of adjacency matrices $\mathbf{A} \in \mathbb{R}^{(r+1) \times l \times l}$, including an identity matrix in addition to the r edge-type matrices.

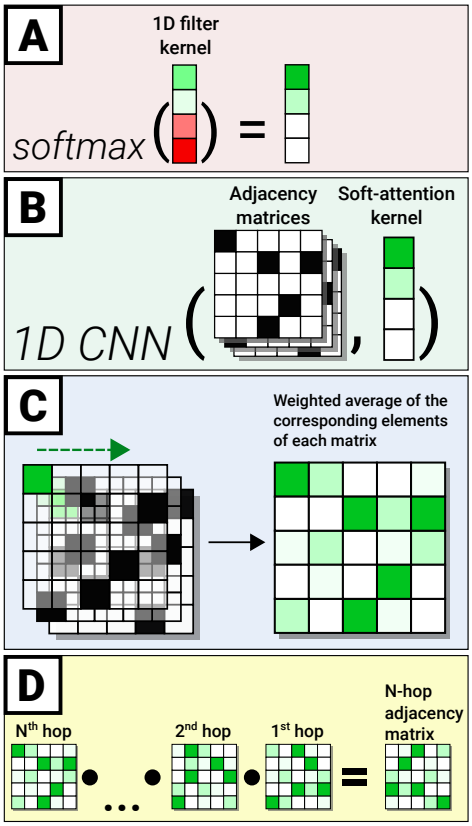

Figure 5.6: An illustration of the adjacency matrix selection using soft-selection. **A** The softmax scales the kernel elements so that they sum to one. **B** The 1D CNN uses this soft-attention kernel to get the weighted sum of the corresponding elements form each adjacency matrix. **C** An illustration of the filter kernel being applied sequentially across each corresponding element, producing an aggregated matrix. **D** Multiple aggregated matrices are combined together into meta-paths using matrix multiplication. Prior to this combination, each softly selected aggregated matrix is normalised by its row-wise inverse degree matrix. This normalisation is omitted from this figure for simplicity.

The identity matrix allows the meta-path attention layer to learn a meta-path that is shorter than the allowed maximum number of hops. This selection is performed by applying a softmax function to the N filter kernels $\mathbf{\Phi} \in \mathbb{R}^{(r+1)\times N}$ of a 1D convolution operation. See figure 5.6 for an illustration of this process.

The selected adjacency matrices are multiplied to produce the meta-path adjacency matrix $\hat{\mathbf{A}} \in \mathbb{R}^{l\times l}$, as follows:

$$\hat{\mathbf{A}} = \mathbf{D}_N^{-1} \cdot \mathbf{A}_N \cdot \ \ldots \ \cdot \mathbf{D}_1^{-1} \cdot \mathbf{A}_1 \tag{5.6}$$

Where $\mathbf{D}_n^{-1}$ is the inverse degree matrix for softly-selected adjacency matrix $\mathbf{A}_n$. Row-wise inverse degree normalisation of each softly selected matrix occurs prior to each matrix being multiplied together into the final meta-path adjacency matrix $\hat{\mathbf{A}}$. This provides numerical stability by scaling signals to vertices with many incoming

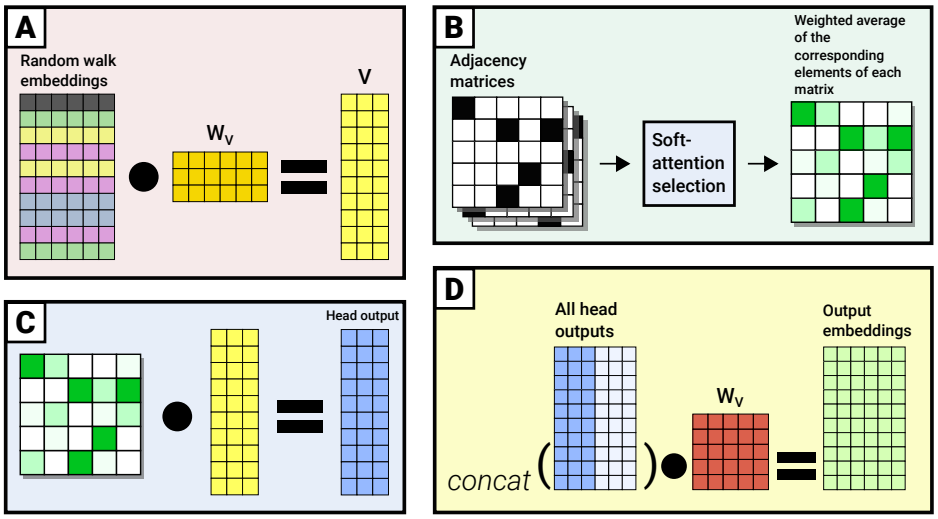

Figure 5.7: An illustration of how the multi-headed meta-path attention layer uses the softly-selected meta-path adjacency matrix to replace the functionality of the self-attention. **A** As in the self-attention heads a parameter matrix linearly transforms the input embeddings into a different representational subspace. **B** The adjacency matrices are combined into a multi-hop meta-path adjacency matrix via soft selection (as illustrated by figure 5.6). **C** The meta-path adjacency matrix is used to pass messages between the subspace embeddings, as opposed to the attention scores computed as the softmax of the dot-product of the Query and Key matrices in typical Self attention. **D** The resultant embeddings outputted by each head are then concatenated together before being linearly transformed by a parameter matrix.

connections, and also prevents the signal vanishing due to the multiplication of consecutive below-one valued matrices. This matrix replaces the attention scores in the self-attention equation. Therefore the final output of a meta-path attention head, $\mathbf{Z}$, is given by:

$$\mathbf{Z} = \hat{\mathbf{A}} \cdot \mathbf{V} \tag{5.7}$$

The outputs of all heads are concatenated and linearly transformed to give the final output of the layer:

$$\text{MetaPathAttentionLayer}(\mathbf{X}, \mathbf{A}, H) = \text{concat}\left([\text{head}_0, \ \ldots\ , \ \text{head}_{H-1}]\right) \cdot \mathbf{W}_0 \tag{5.8}$$

Putting the model together For the walk validity classifier the components are combined as follows:

$$\hat{\mathbf{X}} = \text{LayerNorm}(\mathbf{X}) \tag{5.9}$$

$$\mathbf{Z} = \text{MetaPathAttentionLayer}(\hat{\mathbf{X}}, \mathbf{A}, H) + \hat{\mathbf{X}} \tag{5.10}$$

$$\hat{\mathbf{Z}} = \text{LayerNorm}(\mathbf{Z}) \tag{5.11}$$

$$\mathbf{F} = \text{FeedForwardNetwork}(\hat{\mathbf{Z}}) + \hat{\mathbf{Z}} \tag{5.12}$$

$$\mathbf{S} = \text{LayerNorm}(\text{SelfAttentionLayer}\,(\mathbf{F}, H) + \mathbf{F}) \tag{5.13}$$

$$\mathbf{U} = \text{LayerNorm}(\text{FeedForwardNetwork}\,(\mathbf{S}) + \mathbf{S}) \tag{5.14}$$

The inputs of the meta-path attention and first feed-forward layer are layer-normalised [12], while the self-attention layer and the second feed-forward network are normalised after each addition of residuals. Shallow transformers that have encoders with pre-residual connection Layer Normalisation have been found to have greater training stability than post-residual variants [129]. The self-attention layer and second feed-forward network are essentially a decoder block. Liu *et al.* [129] found no difference between pre- and post-residual connection Layer Normalisation for decoder blocks.

Finally the first row of the output of the attention layers $\mathbf{U}$, which always represents the special $[CLS]$ token during the walk validity task, is extracted. This $[CLS]$ vector is passed to a modified version of the feed-forward network described in equation 5.5. This feed-forward network outputs a single value, to which a sigmoid activation function σ is applied:

$$\mathbf{Y} = \sigma\left(\text{FeedForward}(\mathbf{U}_{1,*})\right) \tag{5.15}$$

The sigmoid activation function bounds the output between zero and one. A one represents a positive classification, while zero represents a negative.

Walk validity graph sampling Walk validity data was generated using the all-relationship dictionary and the degree centrality information. Adjacency matrices for each relationship were constructed for each walk using the individual edge-type-specific relation dictionaries. Positive samples were created by sampling valid walks from the graph. Negative samples, in contrast, were derived by perturbing random walks, and accompanying them with the edge information from a separate walk sample. This process is described by algorithm 1.

The process of sampling an individual random walk can be described accordingly: Given a graph $G = (V, E)$ with vertex set V and all relation type edge set E. Let $C(v)$ denote the degree centrality of vertex $v \in V$, which represents the number of edges that this vertex has with other vertices in the graph, and $N(v)$ the neighbours of vertex v in G. Let $p(v)$ be a probability distribution function defined over its

neighbours $N(v)$ such that:

$$p(v) \propto \frac{1}{C(v)} \tag{5.16}$$

The use of the inverse degree centrality reduces the influence of highly connected vertices by making less connected vertices more likely to be sampled.

Define $[CLS]$ to be a special vertex to represent the start of a walk sequence. Let $W_{[CLS]}$ be a walk sequence initialised with $[CLS]$ followed by a randomly selected vertex $v_1 \in V$. The walk sequence W_{v_i} for a vertex v_i is then recursively defined as:

$$W_{v_i} = \begin{cases} W_{v_{i-1}} \cup v_i & \text{if } v_i \text{ has neighbours in } G \\ W_{v_{i-1}} & \text{otherwise} \end{cases} \tag{5.17}$$

Where, for $i > 1$:

$$v_i \sim p(v_{i-1}) \tag{5.18}$$

Here, $\cup$ denotes the concatenation operation, while the tilde $\sim$ denotes sampling from a probability distribution. The walk sequence W of maximum sample size L is given by $W = W_{v_{L-2}}$. This process is described further by algorithm 2.

Algorithm 1 Sampling valid and generating invalid batches of samples of the DisGeNET knowledge graph for the walk validity task. For details on the how the individual random walks were sampled please see algorithm 2.

```
Initialise labels, batch walks, adjacency matrices, degree matrices
Sample starting vertex
for each starting vertex do
    Perform and pad (if needed) a random walk
    Compute and store the degree matrix for this random walk
    if random value > negative sample rate then
        Store adjacency matrices of the walk
        Label as valid walk (1)
    else
        Replace some walk vertices with random vertices with probability of 10%
        Compute adjacency matrices for a different random walk
        Label as invalid walk (0)
    end if
    Store the walk in batch walks
end for
return labels, batch walks, adjacency matrices, degree matrices
```

5.3.4 SAE CNN classifier with Meta-Path Encoder

Architecture. Here parts of the walk validity architecture are incorporated into the training of the text classifier. During training a depth-first search is used to sample vertices within N-hops of the vertex corresponding to the text input. The sampled vertices are enriched by being passed through the meta-path attention layer,

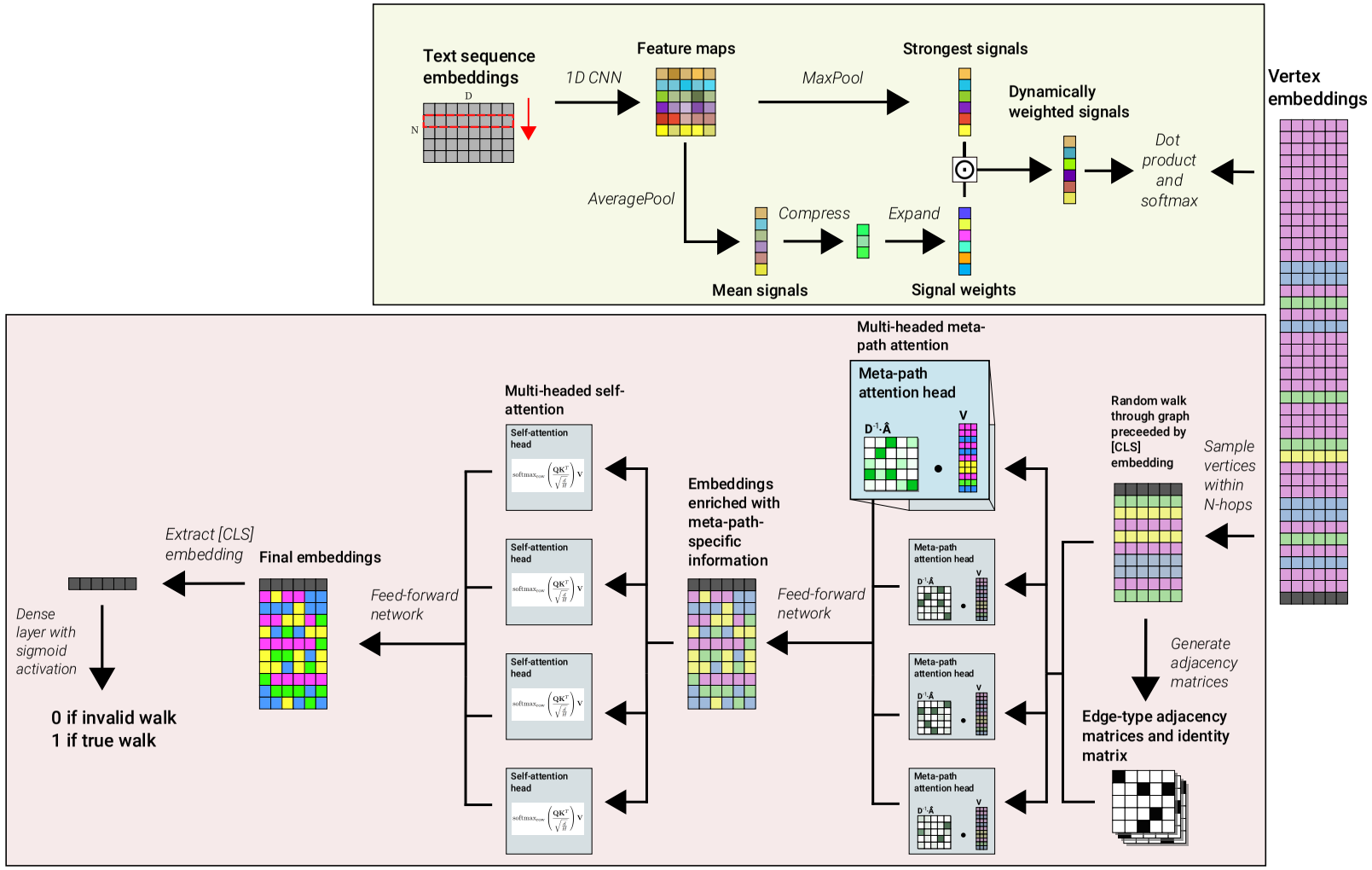

Figure 5.8: An illustration of the SAE CNN text classifier and a walk validity classifier sharing graph vertex embeddings. The SAE CNN text classifier has its dot-product and softmax computed against the entire set of vertex embeddings, while the walk validity classifier is trained using sub-samples of the graph, created using random walks.

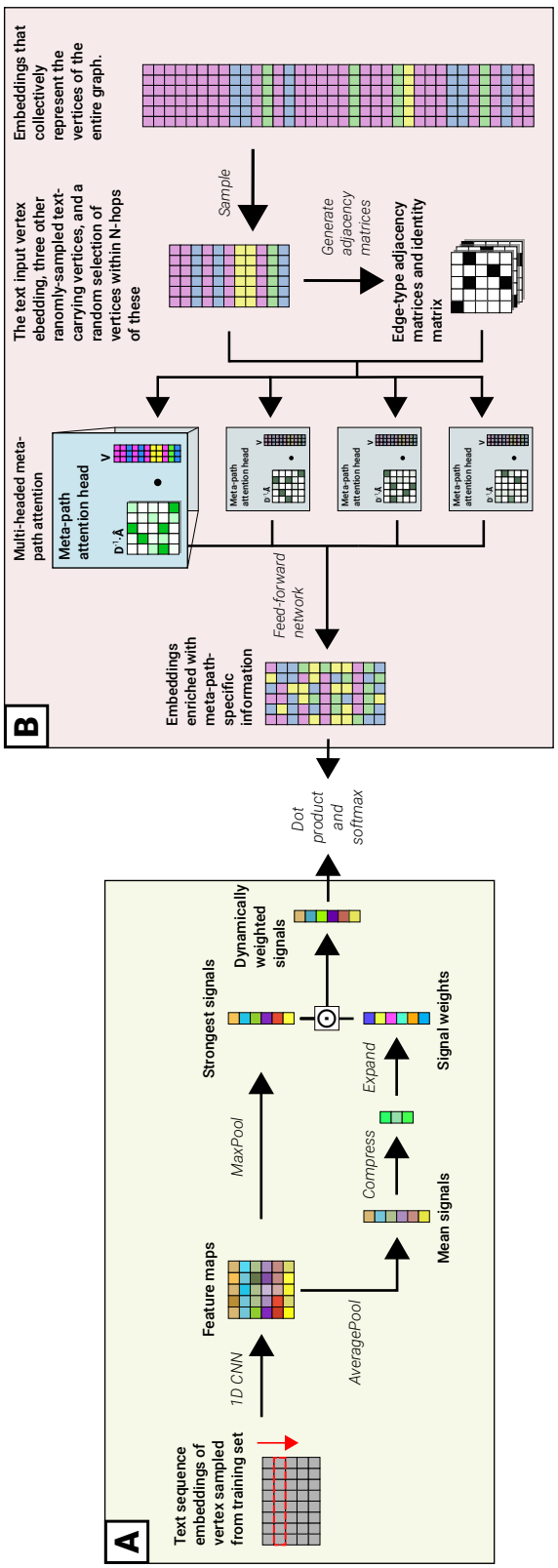

Figure 5.9: An illustration of the Squeeze-and-Excite Convolutional Neural Net (SAE CNN) text classifier, incorporating Meta-Path attention, at training time. **A** The SAE CNN architecture extracts signals from the sequence of word embeddings that represent the natural language description of a vertex. **B** The Meta-Path Attention architecture passes messages between vertices within N-hops of the vertex represented by the text embeddings. These messages are passed through specific multi-hop meta-paths, enriching the vertex embeddings with context encoded by the structure of the knowledge graph.

Algorithm 2 Random walk sampling algorithm for the walk validity task. The probabilities used for the sampling of the related vertices of the current vertex are inversely proportional to their degree centrality.

```
Initialise walk with special [CLS] token followed by input starting vertex
Set current vertex as input starting vertex
for each step up to sample size - 2 do
    if current vertex exists in the relations then
        Get related vertices from the relations of the current vertex
        Select next vertex from related vertices based on sampling probabilities, as in equation 5.16
        Append next vertex to the walk
        Set next vertex as current vertex
    else
        Break the loop
    end if
end for
return walk
```

followed by a feed-forward network. The enriched vertex embeddings corresponding to HPO terms are compared to the output vector of the SAE CNN. See figure 5.9 for an illustration.

At the end of training, a depth-first sample is taken for each HPO vertex, and passed through the meta-path attention layer to create a final global representation for each HPO vertex. The intuition here is that the meta-path layer will learn to pass messages between vertices that are useful for the text classification task, with the Gene and Protein vertices in common with other HPO terms behaving as a "memory" for relevant shared information between HPO vertices.

Graph sampling for text classifier with meta-path attention. The graph sampling for the text classifier with meta-path attention begins with the same preliminaries as the walk validity graph sampling. A graph $G = (V, E)$ is defined with vertex set V and all relation type edge set E. The degree centrality of vertex $v \in V$ is represented again by $C(v)$, and its neighbours in G are denoted by $N(v)$. The probability distribution function $p(v)$ over neighbours $N(v)$ is as defined by equation 5.16.

For this text classifier, however, the graph sampling procedure departs from that of the walk validity sampling method. Given a vertex v_0, which has a natural language description (a vertex that represents an HPO term or a Gene), a random walk begins from v_0 and proceeds until it reaches a pre-specified maximum number of hops away from v_0. Upon reaching this threshold, a new random walk is initiated from a randomly sampled vertex from the unsampled vertices within the closest possible proximity to v_0. This process is repeated until the maximum number of unique vertices have been sampled, or all possible vertices within the defined hop

maximum from v_0 have been visited. All sampling from v_0 is from the probability distribution defined in equation 5.16.

5.3.5 SAE CNN classifier and Walk Validity classifier sharing a Meta-Path Encoder

Here, the text and the walk validity classification architectures share parts of their models: the meta-path attention layer and the subsequent feed-forward network. The text classifier is as described in the previous section, and the walk validity classifier is also as described previously.

5.3.6 Training

Model parameters All models were trained using the ADAM optimiser [108]. Training halted either after 200 epochs or if there was no loss improvement over five consecutive epochs. Models using random walks from the graph sampled 128 vertices per sample per batch, and a standard batch size of 1024 was used. However, the dual-task meta-path model used a batch size of 512 due to GPU memory constraints.

For models with multi-headed attention layers, training began with a "burn-in" period. The learning rate started at 1×10^{-6} and linearly increased with each iteration to reach 5×10^{-4}. These attention layers used eight heads and had a whole-head dropout rate of 25%. The standalone SAE CNN classifier had a constant learning rate of 5×10^{-3}.

The meta-path hop size was fixed at three for relevant models to manage computational load and ensure a sample size of 128 captured a representative N-hop neighbourhood. As N grows, the proportion of vertices sampled from the total neighbourhood shrinks. See figures A.1 and A.2 in the appendices for more details on vertex tallies based on hop sizes between two to seven.

Loss functions The training of the SAE CNN classifier either in isolation, or while sharing embeddings with the walk validity classifier, used a softmax cross entropy as the loss function. However, as it is computationally infeasible to use the MPT to aggregate embeddings for every vertex in the graph, for each sample in a batch at every iteration, the loss for the MPT text classifier had to be adapted to operate on a sub-sample of the vertices. Gutman and Hyvärinen [79] developed an algorithm, Noise-Contrastive Estimation, for approximating the softmax across all possible classes when training using a subsample. During training the model learns to discriminate between the true class and a sample of noise classes. As the noise sample is drawn from the space of possible classes, the model indirectly captures the relationships between the classes. At inference time the model will behave as if it was trained on the full softmax as it has approximated the probability distribution of the full softmax.

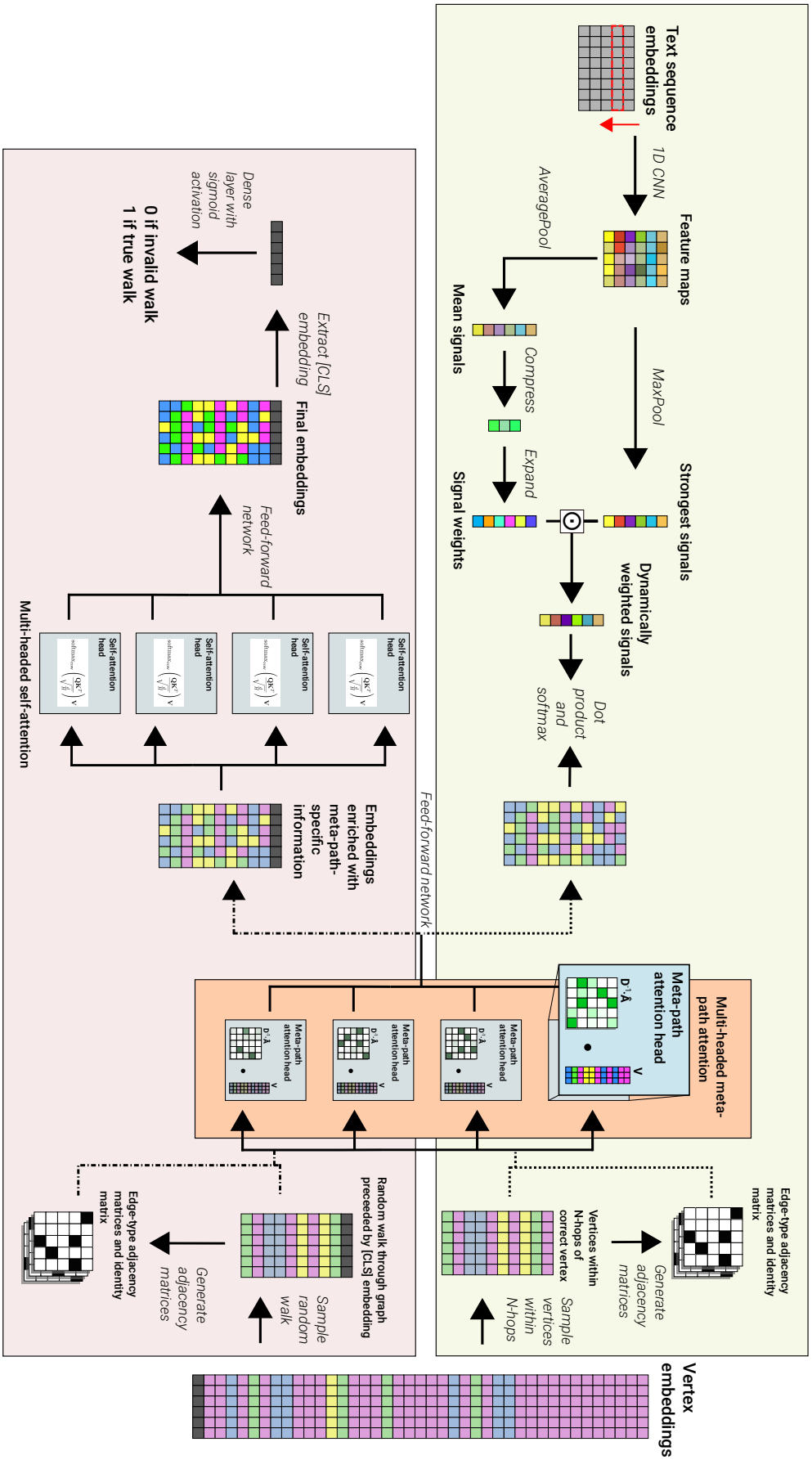

Figure 5.10: A walk validity and text classifiers sharing vertex embeddings, the meta-path attention layer, and its subsequent feed-forward network. Both models retain their separate graph sampling algorithms.

To ensure that the MPT text classifier learns to differentiate between HPO terms, every sample in the batch is compared against every other in the batch to train the model. For each sample, the embedding in the first position always corresponds to the correct HPO class. So after the MPT has produced the aggregated vertex representations, the first aggregated embedding of each sample is concatenated with a copy of every other first position embedding in the batch. Any occurrences of a sample's correct embedding in its copy of all the other embeddings is masked with zeros. The dot-product is computed between each sample's vector representation of the input text in the batch and every vertex embedding in the batch. The loss is computed using Noise-Contrastive Estimation.

For the walk validity task, a sigmoid cross entropy loss function was employed to learn to distinguish between valid and invalid walks. The models trained with both tasks simultaneously weighted the importance of the loss between each task so that the text classification task was favoured by a ratio of 4:1.

5.3.7 Evaluation

As in the last chapter, all the models were evaluated using the 228 PubMed abstracts which had been expertly annotated by domain experts [133]. 40 abstracts were used to determine the optimum classification threshold for each model, with the remaining 188 used to score the models' performance using each model's best classification threshold.

5.4 Results

Table 5.1: Performance metrics for each model on the HPO concept recognition on benchmark corpus. The highest score for each metric is indicated by the bold font.

Model	Batch Size	Threshold	Macro Precision	Macro Recall	Macro F1	Micro Precision	Micro Recall	Micro F1
SAE CNN	1024	0.9	0.425	0.414	0.405	0.464	**0.724**	0.566
+ shared emb. pre-train	1024	0.9	**0.440**	**0.419**	**0.412**	**0.482**	0.698	**0.570**
+ shared emb. sim. train	1024	0.1	0	0	0	0	0	0
SAE CNN w. MPT	1024	0.9	0.203	0.211	0.194	0.201	0.611	0.303
+ pre-train	1024	0.9	0.191	0.197	0.182	0.197	0.629	0.301
+ sim. train	512	0.8	0.187	0.172	0.167	0.257	0.381	0.307

Table 5.1 displays the results of the evaluation. The SAE CNN sharing embeddings with an MPT walk validity classifier trained sequentially, achieves the best performance, with slightly higher f-scores than the SAE CNN alone. When the models sharing embeddings were trained together, the model failed to train and did not make any correct predictions. All of the SAE CNN models incorporating an MPT performed poorly compared with the best performing shared embedding model. The addition of the walk-validity task with the pre-training did not result in much difference from the MPT-based model trained only on the text classification

task. The MPT simultaneously trained on both tasks achieved a similar f-score to the other MPT models, except that its micro precision metric was higher while its recall was lower.

Figures 5.11 to 5.13 display snapshots of the softmax filter kernels for the meta-path attention heads used for soft-selection of the adjacency matrices. Specifically, figure 5.11 shows the kernels after the end of training a model on the text classification task alone. Figure 5.12 are the kernels of a single head from a model trained simultaneously on the walk validity classification and text classification tasks. Finally, figure 5.13 shows the kernels from the same attention head after being pre-trained on the walk validity classification task (sub-figure **A**), and after subsequent training on the text classification task (sub-figure **B**). See figures A.4 to A.6 to see the softmax filter kernels learned by each MPT in full, including from those which only shared embeddings with the SAE CNN text classifier.

5.5 Discussion

Impact of multi-task training In this work, the valid-walk pre-training objective only led to an improvement in performance when an MPT was not incorporated into the text classifier. Otherwise, pre-training with this task slightly inhibited model performance. This agrees somewhat with Liu *et al.* [131], who found they could improve BERT pre-training with the removal of the binary classification task. Instead they trained their model using only the masked word prediction task.

Conversely, for the SAE CNN text classifier only sharing embeddings with the walk validity classifier, pre-training did improve performance. Glorot *et al.* [70] suggested that such an outcome indicates that the method used to set the initial parameters of the vertex embeddings is sub-optimal. In this context the walk validity pre-training may serve as an expensive initial parameterisation of the vertex embeddings. However it is not immediately clear which kind of initial parameterisation would improve performance.

In the original GTN papers [243, 244] and the HGT paper [95], the models were employed for vertex type classification. It may be that the binary classification task did not optimally impart vertex-type and structure information into the embeddings. Perhaps replacing this with a vertex classification task would serve as a more optimal, yet still expensive initial parameterisation, and explicitly impart more useful vertex-type signals. However, in every instance where the walk-validity classifier was used, whether for pre-training or simultaneous training, once trained it consistently achieved accuracies in excess of 90%. See table A.1 in the appendices to see the epoch-by-epoch walk validity classification accuracy for the MPT after walk-validity pre-training. This suggests that there is something that does not work with the MPT in the context of text classification.

Additionally, the MPT employed regularisation techniques, dropout in partic-

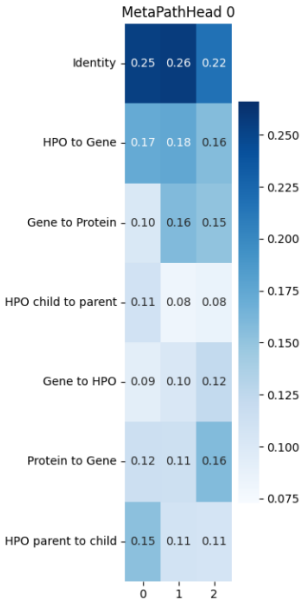

Figure 5.11: The softmax of the filter kernels from a single meta-path attention head after the completion of the text classifier training, with no walk-validity training.

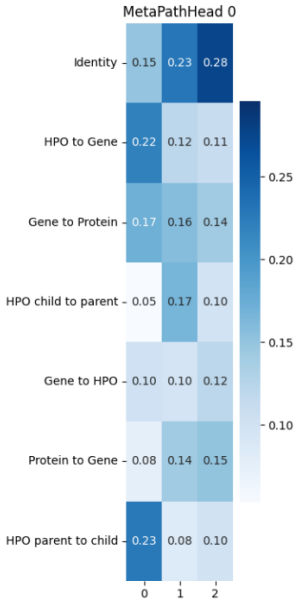

Figure 5.12: The softmax of the filter kernels from a single meta-path attention head after the conclusion of simultaneous walk validity and text classification training.

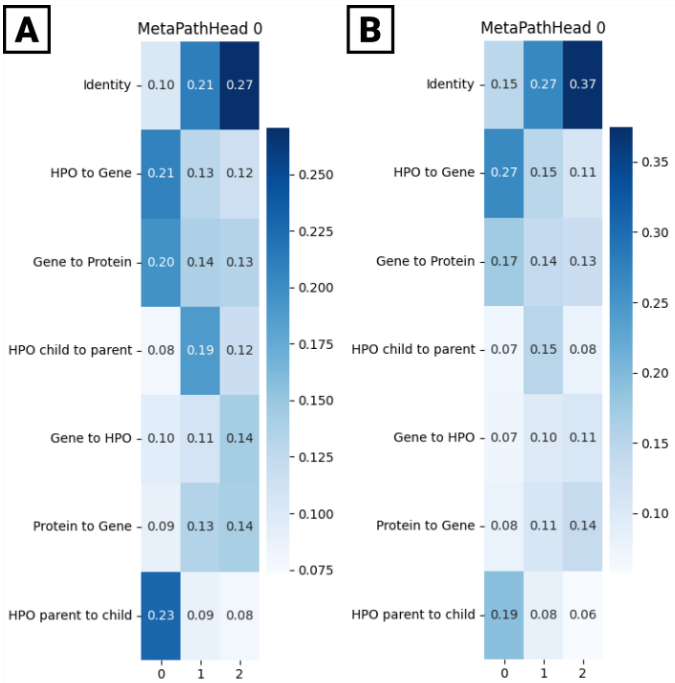

Figure 5.13: The softmax of the filter kernels from a single meta-path attention head after the conclusion of **A** pre-training on the walk validity task, and **B** the same kernels after the subsequent text classifier training.

ular, which were intended to encourage generalisability. Encouraging the model to generalise may have inhibited the MPT heads from learning specific meta-paths through the graph and specific signal transmission between vertices. The softmax function being applied to a meta-path head adjacency matrix would only further compound this issue. A MPT head which is trying to account for multiple meta-paths is only able to increase the signal through one meta-path at the expense of another.

Meta-path learning and explainability To gain insight into what is happening in the MPTs, we can explore the attention weights being used to construct the multi-hop adjacency matrices. Figure 5.11 and figure 5.12 show a single head's softmax filter kernels from the Meta-Path text classifier, trained on the text classification task alone, and simultaneously trained on walk validity and text classification, respectively.

Both heads are adapting to avoid learning meta-paths that terminate with a HPO concept vertex, and to favour an initial hop from a parent HPO concept to its children. Either the head is passing signals away from HPO vertices, or using self-connections through the identity matrix. This is mirrored in figure 5.13.**B**, which shows the final filter kernels following the conclusion of the walk validity pre-training followed by the text classification training. Here, again the model has learned to avoid terminating at an HPO term unless it is a single hop from parent to child and even then favours self-connections.

When we look at this same set of filter kernels prior to the text classification training, in figure 5.13.**A**, we can see that the head is simultaneously learning both [gene → protein] [HPO → gene] and [HPO parent → HPO child → HPO parent] meta-paths. This suggests that for the task of walk validity classification the MPT can learn intuitive, meaningful meta-paths through the graph, and significantly, that message passing between vertices in the graph is beneficial to model performance. This contrasts with those trained for text classification. He *et al.* [85] hypothesised that when a layer with residual connections could not improve a model it would drive its contribution to zero so that the layer performed an identity function and returned its inputs. We can observe that each model appears to be preventing signals from being passed to HPO terms or only passing self-connections at the conclusion of text classifier training.

As shown by figure 5.3, HPO concepts are much more likely to be linked to genes, while genes are much more likely to be associated with fewer HPO concepts. A gene's association with multiple phenotypic disorders may be completely orthogonal to the natural language descriptions of said disorders. A mutated gene may affect cells from different tissues in different ways. Additionally, the HPO ontology is organised so that, for example, disorders that affect the structure of one's hands are similarly named and categorised. It is not organised by the genes which influence these

disorders. This, and the models appearing to avoid message passing, suggests that message passing along the knowledge graph was not beneficial for text classification, and that the DisGeNET knowledge graph was an inappropriate design choice for a HPO term concept recognition system.

Although, the knowledge graph was not the best design choice, the *a priori* hard-coded interpretability of the MPT allowed us to investigate and diagnose the model. The graph with MPT approach has allowed us more insight toward what the model is learning, and how it is aggregating features to make predictions. While a *post hoc* explainability method like SHAP [135] would only indicate which input features appear to influence model outputs the most. On the other hand, SHAP provides single prediction explanations which the MPT-based classifier does not.

Chapter 6

Critical Assessment of Work

6.1 Overview and chapter summaries

The semantic annotation of scientific artefacts, such as literature and data, with terms from controlled vocabularies is a non-trivial problem. Ontologies are updated frequently [197, 111], which necessitates adaptable solutions. This means that any deep learning-based annotation methods must either be general enough to adapt without requiring expensive re-training, or be inexpensive to re-train. The approaches explored in the work described here focused on developing tools that can be retrained within a day, using limited computational resources. I later explored the incorporation of *a priori* interpretability into neural architectures, motivated primarily by concerns regarding the interpretability of deep learning systems.

This chapter provides a comprehensive overview of the research presented in this thesis, putting it into a higher-level context. The work of the chapters is summarised, and then followed by discussions of the results. Then a comparison of the approaches with other methods is made, followed by an exploration of the conceptual contributions of this work. Future avenues of research are suggested throughout.

6.1.1 Summary of chapters

Chapter 1 Here I provided the motivation for the project: the need for efficient tools for categorising, indexing and collating scientific knowledge, that can help us to keep pace with the ever increasing scale of scientific production. The chapter ends with a brief summary of this thesis's contributions.

Chapter 2 I described the evolution of artificial intelligence, with a particular focus on deep neural machine learning and semantic technologies. I described how work to emulate the function of biological neurons led to the development of artificial neural nets and how these have led to computer agents outperforming human agents in specific tasks. I also discussed the development of the semantic web and various subsequent initiatives to describe domains of knowledge, biology in particular, with

machine interpretable structures. I then gave an overview of efforts to combine structured symbolic knowledge representation with deep learning, and concluded by describing previous work towards the semantic annotation of literature and data.

Chapter 3 This chapter explored the use of heuristic annotation with neural error correction methods. Specifically, I employed a string-matching dictionary method to annotate a PLOS paper corpus with Cell Ontology terms. Then I devised various models to exploit noise inherent to the annotations. Two models, including a control, were evaluated against a benchmark. However, one approach failed to produce a viable model, while the others performed poorly.

Chapter 4 I refined a neural dictionary-based concept recognition method, extending it to allow the incorporation of multiple ontologies, and examined the impact of using different combinations of ontologies. The models were benchmarked against a corpus of text annotated with Human Phenotype Ontology terms, and diverse domain ontologies were found to improve performance, especially for models with attention mechanisms. Domain overlap was identified as having a detrimental effect on performance, and investigations into this phenomenon were carried out.

Chapter 5 I constructed a knowledge graph using subsets of UniProt, the Human Phenotype Ontology, and the Disease Gene Association Network. I then developed a neural dictionary concept recognition model that could incorporate this knowledge graph into its architecture. Although these models did not compete with the state-of-the-art, they provided insight into the role that structured domain knowledge can have in deep learning, particularly in respect to providing constraints necessary for model interpretability and explainability.

In summary The chapters proceed from an exploration of the foundational principles of AI, including knowledge representation, and symbolic and statistical learning, to practical applications of deep learning for the semantic annotation of scientific literature with ontology terms. Each chapter builds from the previous, culminating in work to build interpretability into models, with all methods being reasonably inexpensive to train.

6.2 Assessment of results

While the models of chapters 3 and 5 did not perform optimally, the work of chapter 4 produced models whose performance exceeded the state-of-the-art for neural dictionary methods. Additionally these models were extended to operate across multiple domains. However, the generalisability of these approaches is not certain. I had benchmarked my models using both the Cell Ontology segment of the Colorado

Richly Annotated Full-Text (CRAFT) corpus and the version of the Gold Standard Corpus (GSC) for Biomedical concept recognition, annotated with Human Phenotype Ontology Terms, released by Lobo *et al.* [133]. In particular, the work of chapters 4 and 5 were benchmarked against the HPO annotated corpus. The dataset of abstracts annotated with HPO terms came in a table format that is easy to use for evaluating a model output, however it only covers a single ontology. The Colorado Richly Annotated Full-Text (CRAFT) corpus, on the other hand, contains annotations in 97 full journal publications for the Gene Ontology, Cell Type Ontology, Sequence Ontology, Chemical Entities of Biological Interest Ontology, Protein Ontology, NCBI Taxonomy, and the Uberon Ontology. However, until recently I lacked the technical expertise to transform the linked data format of these annotations into a table format amenable to my evaluation scripts. In future work the models developed in chapter 4 should be trained using the ontologies covered by the CRAFT corpus. Then these models can be evaluated against the corpus to better gauge the generality of the improvements brought by combinations of diverse domain ontologies and attention mechanisms.

Although some models excelled at semantic annotation, others did not. This demonstrates the challenge of building broadly applicable deep learning-based tools for semantic annotation, and the requirement for robust multi-domain testing.

6.3 Conceptual contributions

6.3.1 Interpretability and the validity of assumptions

The approaches devised and trialled in this thesis could be described as a progression: from attempting to exploit assumptions with neural networks, to actively encoding assumptions to constrain deep neural networks. The research presented in this thesis demonstrates how complex and challenging it is to identify and validate assumptions in deep learning.

In chapter 3, it was found that the assumption that an autoencoder architecture could be straightforwardly transferred from the domain of image classification to ontology-based concept recognition did not hold. Several factors, such as the simplicity of the autoencoder or the nature of BIO sequence labelling, may have led to the poor performance of the discriminative autoencoder. The discriminative autoencoder approach for noisy image classification by Xia *et al.* [234] was applied to one single positive class which composed the majority of samples. In the heuristic annotation case, the positive samples are in a minority and actually represent a great diversity of semantic concepts. The tokens of a sequence labelled the beginning of a sequence may not have much in common with other tokens with the same classification. The same is true for the words labelled as negative cases. While I can make *post hoc* explanations, these explanations remain speculative.

Though in chapter 4 I was again left to make *post hoc* explanations, the ontology structures were statically incorporated into the model architectures. These structures formed a conceptual scaffold with which to guide investigations into model performance disparities, which allowed a more rational explanation of differences in model performance.

In chapter 5, the assumption that domain specific structured information can help improve model performance is explicitly encoded into the model, but the model had flexibility over how it leverages this structure. This explicit encoding meant that how the model was learning could be interpreted. It could be identified that for the walk validity task the model was learning intuitive multi hop meta-paths through the graph. In contrast, during the semantic annotation task it was actively forgoing message passing through the knowledge graph structure - indicating that the graph was detrimental to model performance.

To summarise, the explicit encoding of assumptions into a model provides a greater degree of explanatory power. This allows us to better intuit why a model is under-performing, and may also boost our confidence in their predictions when they perform well. Given developments toward simplifying the production of high quality task-specific knowledge graphs [132], it would be interesting to explore if there may be a more suitable graph structure which may improve model performance. This work could explore whether the architecture needs to take into account imbalances in data, beyond the inverse degree normalisation used in chapter 5. Regarding the vertices linked along a common meta-path, perhaps adaptive systems need to weight the relative importance of each vertex sharing a meta-path terminating with the same vertex.

6.3.2 Limitations of architectures

Incorporating graph structure into architecture Despite suggesting the graph was sub-optimal, there is also the possibility that the meta-path transformer architecture is not a suitable architecture for semantic annotation. Although the knowledge graph contained tens of thousands of vertices, this does not approach the volumes of data typically required by transformer architectures. The scaled-transformer architecture used in chapter 4 did mitigate the need for vast quantities of training data somewhat. However, the scaling is not straightforwardly compatible with a transformer in a graph context because graph data is unordered. While the scaled-transformer is intended to leverage the assumption that the order of words in a sentence and their proximity to one another provides relevant signals, the same assumption cannot be made in the case of an unordered set of vertices.

In chapter 4, *post hoc* analysis of the model embeddings and their proximity suggested that different ancestry matrices from overlapping ontology domains hindered model performance. Concepts with similar, or the same, natural language descrip-

tions had different aggregated representations due to signals passed through their respective ancestry matrices. In future these concepts may be reconciled or combined into the same concepts where necessary in the pre-processing steps. However, this would require domain expertise, so perhaps overlapping domains should be avoided. In this chapter it was also found that model performance may be heavily influenced by heuristics, particularly when two concepts overlap. Additionally, all the models devised in this thesis have no mechanism for dealing with negation. A passage could be specifying that the work describes a particular process and not another. Large language models have the capacity to make these distinctions and could be used to develop expert systems for semantic annotation.

Generative large language models An alternative approach to semantic annotation entails the development of large-scale generalist models, such as generative large language models capable of few-shot learning and question-answering formats, like GPT-4 [165] and Llama 2 [219]. This would necessitate vast computational resources, relative to those employed in this work.

For perspective, the GatorTron model developed by Yang *et al.* [237] required 992 state-of-the-art NVIDIA A100 80Gb GPUs and six days of computational time for training. In contrast, the research for this thesis was conducted using a single NVIDIA TITAN XP GPU for Chapters 3 and 4, and a single A100 80Gb GPU for Chapter 5, with all models training for under 24 hours. GatorTron was applied to the task of concept recognition, however it was used as the basis for a sequence labelling problem, generating B-I-O format labels for each word in a sequence, rather than returning specific concept IDs for spans of text, which semantic annotation requires.

Large language models are prone to hallucination, where the model outputs appear to correspond to bonafide facts but in reality only take the authoritative form one would expect from a fact [165, 219]. This would make such a solution unacceptable for the task of semantic annotation. Additionally, generative models have a temperature parameter which means that the sequences they generate are not deterministic [213]. Given an input of text, with higher temperature parameterisations the model may return various different outputs, and with a lower temperature outputs may become repetitive. Both of which may compromise the suitability of large language models for semantic annotation.

Recycling large language model components A different use of large language models instead of generative question-answering is by recycling its components rather than generating outputs. The Large Language Model BERT was incorporated into a neural dictionary architecture Phenotagger [136] in this way. This method side-steps the problem posed by non-deterministic outputs by using the hidden layers of the language model as part of a deterministic classifier. This method still requires the initial expense of training a large language model, but only the classifier layer

would need to be retrained when an ontology is updated. The methods devised in chapter 4 incorporated a much smaller ELMo pre-trained language model to reuse its hidden components and outperformed Phenotagger on the HPO benchmark set.

Complementing the strengths of different approaches The field of natural language processing is vast, and potential semantic annotation solutions vary from simpler neural dictionaries to the sophisticated large language models - each with their own advantages and limitations. Avenues of research combining various methodologies may prove beneficial.

There has been work to develop approaches to ensure greater consistency and trustworthiness of generative question-answering large language models. These typically involve the incorporation of vector databases or use knowledge graphs as guardrails [51, 83]. For example, ChatLaw, introduced by Cui *et al.* [51], which combined a Llama generative large language model with a BERT-based system for extracting keywords and relevant laws from a database, for Chinese legal question answering. There are two BERT systems which both use a vector similarity look-up to either identify the most similar keywords or relevant laws from a database, which are neural dictionary look-ups like the models explored in chapters 4 and 5. The text describing these laws and the keywords are then provided to the Llama model alongside the user query as context. The authors found that this significantly reduced the model's propensity to hallucinate.

If the resources were provided, it would be worthwhile to develop an open source generative large language model for researchers, and build an expert system that uses it for data annotation. This would likely be large collaborative project, with many researchers involved in its development, and require an appropriate investment of resources. Perhaps a system could be developed that uses a neural dictionary to look up ontology concepts from within a passage of text, and even then use structured knowledge to provide further context for these concepts. Then these concepts could be used to augment a generative language model query that returns an updated set of relevant ontology terms based upon the passage of text and the matched terms. This would strike a balance between the consistency of the neural dictionary models and the greater expressive power of the large language models.

Bibliography

[1] David Ifeoluwa Adelani et al. 'Distant Supervision and Noisy Label Learning for Low Resource Named Entity Recognition: A Study on Hausa and Yorùbà'. In: (2020). DOI: 10.48550/arXiv.2003.08370. arXiv: 2003.08370 [cs.CL].

[2] Görkem Algan and Ilkay Ulusoy. 'Image Classification with Deep Learning in the Presence of Noisy Labels: A Survey'. In: (2019). DOI: 10.48550/arXiv.1912.05170. arXiv: 1912.05170 [cs.LG].

[3] Basharat Ali and Peter Dahlhaus. 'The Role of FAIR Data towards Sustainable Agricultural Performance: A Systematic Literature Review'. In: *Agriculture* 12.2 (2022), p. 309. DOI: 10.3390/agriculture12020309.

[4] Sara Althubaiti et al. 'Ontology-based prediction of cancer driver genes'. In: *Scientific Reports* 9.1 (2019), p. 17405. DOI: 10.1038/s41598-019-53454-1.

[5] Artaches Ambartsoumian and Fred Popowich. 'Self-Attention: A Better Building Block for Sentiment Analysis Neural Network Classifiers'. In: (2018). DOI: 10.48550/arXiv.1812.07860. arXiv: 1812.07860 [cs.CL].

[6] 'Analysis of a complex of statistical variables into principal components'. In: *Journal of Educational Psychology* 24.6 (1933), pp. 417–441. ISSN: 1934-00645-001. DOI: https://doi.org/10.1037/h0071325.

[7] Aryan Arbabi et al. 'Identifying Clinical Terms in Medical Text Using Ontology-Guided Machine Learning'. In: *JMIR Medical Informatics* 7.2 (2019), e12596. DOI: 10.2196/12596.

[8] Alan R Aronson. 'Effective mapping of biomedical text to the UMLS Metathesaurus: the MetaMap program'. In: *Proceedings of the AMIA Symposium* (2001), pp. 17–21. URL: https://pubmed.ncbi.nlm.nih.gov/11825149.

[9] Michael Ashburner et al. 'Gene Ontology: tool for the unification of biology'. In: *Nature Genetics* 25.1 (2000), pp. 25–29. DOI: 10.1038/75556.

[10] Mark van Assem, Aldo Gangemi and Guus Schreiber. 'Conversion of WordNet to a standard RDF/OWL representation'. In: *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*. Genoa, Italy: European Language Resources Association (ELRA), May 2006.

URL: http://www.lrec-conf.org/proceedings/lrec2006/pdf/165_pdf.pdf.

[11] Sören Auer et al. 'DBpedia: A Nucleus for a Web of Open Data'. In: *The Semantic Web*. Ed. by Karl Aberer et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 722–735. ISBN: 978-3-540-76298-0. DOI: 10.1007/978-3-540-76298-0_52.

[12] Jimmy Lei Ba, Jamie Ryan Kiros and Geoffrey E. Hinton. *Layer Normalization*. 2016. DOI: 10.48550/arXiv.1607.06450. arXiv: 1607.06450 [stat.ML].

[13] Michael Bada et al. 'Concept annotation in the CRAFT corpus'. In: *BMC Bioinformatics* 13.1 (2012), pp. 1–20. DOI: 10.1186/1471-2105-13-161.

[14] Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. DOI: 10.48550/arXiv.1409.0473. arXiv: 1409.0473 [cs.CL].

[15] Yuhan Bai. 'RELU-Function and Derived Function Review'. In: *International Conference on Science and Technology Ethics and Human Future (STEHF 2022)*. Vol. 144. SHS Web of Conferences. 2022, p. 02006. DOI: 10.1051/shsconf/202214402006.

[16] Anita Bandrowski et al. 'The Ontology for Biomedical Investigations'. In: *PloS one* 11.4 (2016), e0154556. DOI: 10.1371/journal.pone.0154556.

[17] Jonathan Bard, Seung Y Rhee and Michael Ashburner. 'An ontology for cell types'. In: *Genome Biology* 6.2 (2005), pp. 1–5. DOI: 10.1186/gb-2005-6-2-r21.

[18] Erdenebileg Batbaatar and Keun Ho Ryu. 'Ontology-Based Healthcare Named Entity Recognition from Twitter Messages Using a Recurrent Neural Network Approach'. In: *International Journal of Environmental Research and Public Health* 16.19 (2019), p. 3628. DOI: 10.3390/ijerph16193628.

[19] Oliver Bender, Franz Josef Och and Hermann Ney. 'Maximum Entropy Models for Named Entity Recognition'. In: *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*. CONLL '03. Edmonton, Canada: Association for Computational Linguistics, 2003, pp. 148–151. DOI: 10.3115/1119176.1119196.

[20] Y. Bengio, P. Simard and P. Frasconi. 'Learning Long-Term Dependencies with Gradient Descent is Difficult'. In: *Transactions on Neural Networks* 5.2 (Mar. 1994), pp. 157–166. ISSN: 1045-9227. DOI: 10.1109/72.279181. URL: 10.1109/72.279181.

[21] Yoshua Bengio. 'Learning Deep Architectures for AI'. In: *Foundations and Trends in Machine Learning* 2.1 (2009), pp. 1–127. ISSN: 1935-8237. DOI: 10.1561/2200000006.

[22] Yoshua Bengio, Réjean Ducharme and Pascal Vincent. 'A Neural Probabilistic Language Model'. In: *Advances in Neural Information Processing Systems*. Ed. by T. Leen, T. Dietterich and V. Tresp. Vol. 13. MIT Press, 2000. URL: https://proceedings.neurips.cc/paper_files/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf.

[23] Tim Berners-Lee, James Hendler and Ora Lassila. 'The Semantic Web'. In: *Scientific American* 284.5 (2001), pp. 34–43. ISSN: 00368733, 19467087. URL: http://www.jstor.org/stable/26059207.

[24] Daniel M Bikel, Richard Schwartz and Ralph M Weischedel. 'An Algorithm that Learns What's in a Name'. In: *Machine learning* 34 (1999), pp. 211–231. DOI: 10.1023/A:1007558221122.

[25] Christian Bizer, Tom Heath and Tim Berners-Lee. 'Linked Data - The Story So Far'. In: *International Journal on Semantic Web and Information Systems (IJSWIS)* 5.3 (2009), pp. 1–22. DOI: 10.4018/978-1-60960-593-3.ch008. URL: https://EconPapers.repec.org/RePEc:igg:jswis0:v:5:y:2009:i:3:p:1-22.

[26] Margreet Bloemers and Annalisa Montesanti. 'The FAIR Funding Model: Providing a Framework for Research Funders to Drive the Transition toward FAIR Data Management and Stewardship Practices'. In: *Data Intelligence* 2.1-2 (Jan. 2020), pp. 171–180. ISSN: 2641-435X. DOI: 10.1162/dint_a_00039.

[27] Piotr Bojanowski et al. 'Enriching Word Vectors with Subword Information'. In: *Transactions of the Association for Computational Linguistics* 5 (June 2017), pp. 135–146. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00051.

[28] Evan E. Bolton et al. 'Chapter 12 - PubChem: Integrated Platform of Small Molecules and Biological Activities'. In: ed. by Ralph A. Wheeler and David C. Spellmeyer. Vol. 4. Annual Reports in Computational Chemistry. Elsevier, 2008, pp. 217–241. DOI: https://doi.org/10.1016/S1574-1400(08)00012-1.

[29] Rishi Bommasani et al. *On the Opportunities and Risks of Foundation Models*. 2022. DOI: 10.48550/arXiv.2108.07258. arXiv: 2108.07258 [cs.LG].

[30] Léon Bottou. 'Large-Scale Machine Learning with Stochastic Gradient Descent'. In: *Proceedings of COMPSTAT'2010*. Ed. by Yves Lechevallier and Gilbert Saporta. Heidelberg: Physica-Verlag HD, 2010, pp. 177–186. ISBN: 978-3-7908-2604-3. DOI: 10.1007/978-3-7908-2604-3_16.

[31] Victoria Bourgeais, Farida Zehraoui and Blaise Hanczar. 'GraphGONet: a self-explaining neural network encapsulating the Gene Ontology graph for phenotype prediction on gene expression'. In: *Bioinformatics* 38.9 (Mar. 2022), pp. 2504–2511. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btac147.

[32] Victoria Bourgeais et al. 'Deep GONet: self-explainable deep neural network based on Gene Ontology for phenotype prediction from gene expression data'. In: *BMC Bioinformatics* 22.10 (2021), pp. 1–25. DOI: 10.1186/s12859-021-04370-7.

[33] Elizabeth I. Boyle et al. 'GO::TermFinder—open source software for accessing Gene Ontology information and finding significantly enriched Gene Ontology terms associated with a list of genes'. In: *Bioinformatics* 20.18 (Aug. 2004), pp. 3710–3715. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bth456.

[34] Nadav Brandes et al. 'ProteinBERT: a universal deep-learning model of protein sequence and function'. In: *Bioinformatics* 38.8 (Feb. 2022), pp. 2102–2110. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btac020.

[35] Dan Brickley, Ramanathan V Guha and Andrew Layman. *Resource Description Framework (RDF) Schema Specification*. Tech. rep. Technical report, W3C, 1999. W3C Proposed Recommendation., 1998. URL: http://www.w3.org/TR/PR-rdf-schema.

[36] Sébastien Bubeck et al. *Sparks of Artificial General Intelligence: Early experiments with GPT-4*. 2023. DOI: 10.48550/arXiv.2303.12712. arXiv: 2303.12712 [cs.CL].

[37] David Campos, Sérgio Matos and José Luis Oliveira. 'A modular framework for biomedical concept recognition'. In: *BMC Bioinformatics* 14.1 (2013), pp. 1–21. DOI: 10.1186/1471-2105-14-281.

[38] Nicolas Carion et al. 'End-to-End Object Detection with Transformers'. In: *European Conference on Computer Vision*. Ed. by Andrea Vedaldi et al. Cham: Springer International Publishing, 2020, pp. 213–229. ISBN: 978-3-030-58452-8. DOI: 10.1007/978-3-030-58452-8_13.

[39] Daniel Cer et al. 'Universal Sentence Encoder'. In: (2018). DOI: 10.48550/arXiv.1803.11175. arXiv: 1803.11175 [cs.CL].

[40] Feihu Che et al. 'ParamE: Regarding Neural Network Parameters as Relation Embeddings for Knowledge Graph Completion'. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.03 (Apr. 2020), pp. 2774–2781. DOI: 10.1609/aaai.v34i03.5665. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5665.

[41] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. DOI: 10.48550/arXiv.1406.1078. arXiv: 1406.1078 [cs.CL].

[42] Jan K Chorowski et al. 'Attention-Based Models for Speech Recognition'. In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper_files/paper/2015/file/1068c6e4c8051cfd4e9ea8072e3189e2-Paper.pdf.

[43] William W. Cohen. 'Fast Effective Rule Induction'. In: *Machine Learning Proceedings 1995*. Ed. by Armand Prieditis and Stuart Russell. San Francisco (CA): Morgan Kaufmann, 1995, pp. 115–123. ISBN: 978-1-55860-377-6. DOI: 10.1016/B978-1-55860-377-6.50023-2.

[44] Ashley Mae Conard, Alan DenAdel and Lorin Crawford. 'A spectrum of explainable and interpretable machine learning approaches for genomic studies'. In: *Wiley Interdisciplinary Reviews: Computational Statistics* (2023), e1617. DOI: https://doi.org/10.1002/wics.1617.

[45] Gene Ontology Consortium. 'The Gene Ontology (GO) database and informatics resource'. In: *Nucleic Acids Research* 32.suppl_1 (Jan. 2004), pp. D258–D261. ISSN: 0305-1048. DOI: 10.1093/nar/gkh036.

[46] International Human Genome Sequencing Consortium. 'Initial sequencing and analysis of the human genome'. In: *Nature* 409 (2001), pp. 860–921. DOI: 10.1038/35057062.

[47] The UniProt Consortium. 'The Universal Protein Resource (UniProt)'. In: *Nucleic Acids Research* 36.suppl_1 (Nov. 2007), pp. D190–D195. ISSN: 0305-1048. DOI: 10.1093/nar/gkm895.

[48] Laurel Cooper et al. 'The Plant Ontology as a Tool for Comparative Plant Anatomy and Genomic Analyses'. In: *Plant and Cell Physiology* 54.2 (2013), e1–e1. ISSN: 0032-0781. DOI: 10.1093/pcp/pcs163.

[49] Marco Cremaschi, Roberto Avogadro and David Chieregato. 'MantisTable: an Automatic Approach for the Semantic Table Interpretation'. In: (2019).

[50] Gabor Csardi and Tamas Nepusz. 'The igraph software package for complex network research'. In: *InterJournal* Complex Systems (2006), p. 1695. URL: https://igraph.org.

[51] Jiaxi Cui et al. *ChatLaw: Open-Source Legal Large Language Model with Integrated External Knowledge Bases*. 2023. DOI: 10.48550/arXiv.2306.16092. arXiv: 2306.16092 [cs.CL].

[52] Claudia d'Amato, Nicola Flavio Quatraro and Nicola Fanizzi. 'Injecting Background Knowledge into Embedding Models for Predictive Tasks on Knowledge Graphs'. In: *The Semantic Web*. Ed. by Ruben Verborgh et al. Cham: Springer International Publishing, 2021, pp. 441–457. ISBN: 978-3-030-77385-4. DOI: 10.1007/978-3-030-77385-4_26.

[53] Scott Deerwester et al. 'Indexing by Latent Semantic Analysis'. In: *Journal of the American Society for Information Science* 41.6 (1990), pp. 391–407. DOI: https://doi.org/10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9.

[54] Kirill Degtyarenko et al. 'ChEBI: a database and ontology for chemical entities of biological interest'. In: *Nucleic Acids Research* 36.suppl_1 (Oct. 2007), pp. D344–D350. ISSN: 0305-1048. DOI: 10.1093/nar/gkm791.

[55] Jia Deng et al. 'ImageNet: A large-scale hierarchical image database'. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[56] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. DOI: 10.48550/arXiv.1810.04805. arXiv: 1810.04805 [cs.CL].

[57] Dennis Diefenbach, Kamal Singh and Pierre Maret. 'WDAqua-Core1: A Question Answering Service for RDF Knowledge Bases'. In: *Companion Proceedings of the The Web Conference 2018*. WWW '18. Lyon, France: International World Wide Web Conferences Steering Committee, 2018, pp. 1087–1091. ISBN: 9781450356404. DOI: 10.1145/3184558.3191541.

[58] Alexander D Diehl et al. 'The Cell Ontology 2016: enhanced content, modularization, and ontology interoperability'. In: *Journal of Biomedical Semantics* 7.44 (2016). DOI: 10.1186/s13326-016-0088-7.

[59] Xishuang Dong et al. 'Transfer bi-directional LSTM RNN for named entity recognition in Chinese electronic medical records'. In: *2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom)*. IEEE. 2017, pp. 1–4. DOI: 10.1109/HealthCom.2017.8210840.

[60] John Duchi, Elad Hazan and Yoram Singer. 'Adaptive Subgradient Methods for Online Learning and Stochastic Optimization'. In: *Journal of Machine Learning Research* 12.null (July 2011), pp. 2121–2159. ISSN: 1532-4435. URL: https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf.

[61] Bastian Eine, Matthias Jurisch and Werner Quint. 'Ontology-Based Big Data Management'. In: *Systems* 5.3 (2017). ISSN: 2079-8954. DOI: 10.3390/systems5030045.

[62] Jeffrey L. Elman. 'Finding Structure in Time'. In: *Cognitive Science* 14.2 (1990), pp. 179–211. DOI: 10.1207/s15516709cog1402_1.

[63] Angel Esteban-Gil, Jesualdo Tomás Fernández-Breis and Martin Boeker. 'Analysis and visualization of disease courses in a semantically-enabled cancer registry'. In: *Journal of biomedical semantics* 8 (2017), pp. 1–16. DOI: 10.1186/s13326-017-0154-9.

[64] D. Fensel et al. 'OIL: Ontology Infrastructure for the Semantic Web'. In: *IEEE Intelligent Systems* 16.2 (2001), pp. 38–45. DOI: 10.1109/5254.920598.

[65] Santo Fortunato et al. 'Science of science'. In: *Science* 359.6379 (2018), eaao0185. DOI: 10.1126/science.aao0185.

[66] Kunihiko Fukushima. 'Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position'. In: *Biological Cybernetics* 36.4 (1980), pp. 193–202. DOI: 10.1007/BF00344251.

[67] Kunihiko Fukushima. 'Neocognitron: A hierarchical neural network capable of visual pattern recognition'. In: *Neural Networks* 1.2 (1988), pp. 119–130. ISSN: 0893-6080. DOI: 10.1016/0893-6080(88)90014-7.

[68] Jonas Gehring et al. *A Convolutional Encoder Model for Neural Machine Translation*. 2017. DOI: 10.48550/arXiv.1611.02344. arXiv: 1611.02344 [cs.CL].

[69] Felix A. Gers, Jürgen Schmidhuber and Fred Cummins. 'Learning to Forget: Continual Prediction with LSTM'. In: *Neural Computation* 12.10 (2000), pp. 2451–2471. DOI: 10.1162/089976600300015015.

[70] Xavier Glorot and Yoshua Bengio. 'Understanding the difficulty of training deep feedforward neural networks'. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[71] Xavier Glorot, Antoine Bordes and Yoshua Bengio. 'Deep Sparse Rectifier Neural Networks'. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323. URL: https://proceedings.mlr.press/v15/glorot11a.html.

[72] I. Goodfellow, Y. Bengio and A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016. ISBN: 9780262337373. URL: https://books.google.co.uk/books?id=omivDQAAQBAJ.

[73] Alex Graves. *Generating Sequences With Recurrent Neural Networks*. 2014. DOI: 10.48550/arXiv.1308.0850. arXiv: 1308.0850 [cs.NE].

[74] Alex Graves and Jürgen Schmidhuber. 'Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures'. In: *Neural Networks* 18.5 (2005). IJCNN 2005, pp. 602–610. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2005.06.042.

[75] Alex Graves et al. 'Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks'. In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2006, pp. 369–376. ISBN: 1595933832. DOI: 10.1145/1143844.1143891.

[76] Gunnar Aastrand Grimnes et al. *RDFLib/rdflib: RDFlib 6.3.1*. Version 6.3.1. Mar. 2023. DOI: 10.5281/zenodo.7748890.

[77] Thomas R. Gruber. 'A translation approach to portable ontology specifications'. In: *Knowledge Acquisition* 5.2 (1993), pp. 199–220. ISSN: 1042-8143. DOI: 10.1006/knac.1993.1008.

[78] Qipeng Guo et al. 'Multi-scale Self-Attention for Text Classification'. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 05. 2020, pp. 7847–7854. DOI: 10.1609/aaai.v34i05.6290.

[79] Michael Gutmann and Aapo Hyvärinen. 'Noise-contrastive estimation: A new estimation principle for unnormalized statistical models'. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 297–304. URL: https://proceedings.mlr.press/v9/gutmann10a.html.

[80] Francisco Guzmán et al. 'The FLoRes Evaluation Datasets for Low-Resource Machine Translation: Nepali-English and Sinhala-English'. In: (2019). DOI: 10.48550/arXiv.1902.01382. arXiv: 1902.01382 [cs.CL].

[81] Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart. 'Exploring Network Structure, Dynamics, and Function using NetworkX'. In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught and Jarrod Millman. Pasadena, CA USA, 2008, pp. 11–15.

[82] Will Hamilton, Zhitao Ying and Jure Leskovec. 'Inductive Representation Learning on Large Graphs'. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf.

[83] Hangfeng He, Hongming Zhang and Dan Roth. *Rethinking with Retrieval: Faithful Large Language Model Inference*. 2022. DOI: 10.48550/arXiv.2301.00303. arXiv: 2301.00303 [cs.CL].

[84] Kaiming He et al. 'Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification'. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015. URL: https://openaccess.thecvf.com/content_iccv_2015/papers/He_Delving_Deep_into_ICCV_2015_paper.pdf.

[85] Kaiming He et al. 'Deep Residual Learning for Image Recognition'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016. URL: https://openaccess.thecvf.com/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf.

[86] D.O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Originally published in 1949. Taylor & Francis, 2005. ISBN: 9781135631918. URL: https://books.google.co.uk/books?id=uyV5AgAAQBAJ.

[87] James Hendler, Deborah L McGuinness et al. 'The DARPA Agent Markup Language'. In: *IEEE Intelligent systems* 15.6 (2000), pp. 67–73. URL: http://www-ksl.stanford.edu/pub/KSL_Reports/KSL-00-10.html.

[88] Dan Hendrycks and Kevin Gimpel. 'Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units'. In: *CoRR* abs/1606.08415 (2016). DOI: 10.48550/arXiv.1606.08415. arXiv: 1606.08415.

[89] G. E. Hinton and R. R. Salakhutdinov. 'Reducing the Dimensionality of Data with Neural Networks'. In: *Science* 313.5786 (2006), pp. 504–507. DOI: 10.1126/science.1127647.

[90] Geoffrey Hinton, Nitish Srivastava and Kevin Swersky. *Lecture 6.e-rmsprop: Divide the Gradient by a Running Average of Its Recent Magnitude*. Online course. 2012. URL: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[91] Geoffrey E. Hinton et al. *Improving neural networks by preventing co-adaptation of feature detectors*. 2012. DOI: 10.48550/arXiv.1207.0580. arXiv: 1207.0580 [cs.NE].

[92] Sepp Hochreiter and Jürgen Schmidhuber. 'Long Short-Term Memory'. In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[93] Eurie L. Hong et al. 'Gene Ontology annotations at SGD: new data sources and annotation methods'. In: *Nucleic Acids Research* 36.suppl_1 (Nov. 2007), pp. D577–D581. ISSN: 0305-1048. DOI: 10.1093/nar/gkm909.

[94] Jie Hu, Li Shen and Gang Sun. 'Squeeze-and-Excitation Networks'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018, pp. 7132–7141. URL: https://openaccess.thecvf.com/content_cvpr_2018/papers/Hu_Squeeze-and-Excitation_Networks_CVPR_2018_paper.pdf.

[95] Ziniu Hu et al. 'Heterogeneous Graph Transformer'. In: *Proceedings of The Web Conference 2020*. WWW '20. Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 2704–2710. ISBN: 9781450370233. DOI: 10.1145/3366423.3380027.

[96] Lianzhe Huang et al. *Text Level Graph Neural Network for Text Classification*. 2019. DOI: 10.48550/arXiv.1910.02356. arXiv: 1910.02356 [cs.CL].

[97] Rachael P Huntley et al. 'Understanding how and why the Gene Ontology and its annotations evolve: the GO within UniProt'. In: *GigaScience* 3.1 (Mar. 2014), pp. 2047-217X-3–4. ISSN: 2047-217X. DOI: 10.1186/2047-217X-3-4.

[98] Sergey Ioffe and Christian Szegedy. 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'. In: *CoRR* abs/1502.03167 (2015). DOI: 10.48550/arXiv.1502.03167. arXiv: 1502.03167.

[99] Kevin Jarrett et al. 'What is the Best Multi-Stage Architecture for Object Recognition?' In: *2009 IEEE 12th International Conference on Computer Vision*. 2009, pp. 2146–2153. DOI: 10.1109/ICCV.2009.5459469.

[100] George F. Jenks. 'The Data Model Concept in Statistical Mapping'. In: vol. 7. 1967, pp. 186–190.

[101] Pengcheng Jiang et al. *Bi-level Contrastive Learning for Knowledge-Enhanced Molecule Representations*. 2023. DOI: 10.48550/arXiv.2306.01631. arXiv: 2306.01631 [cs.LG].

[102] Clement Jonquet et al. 'NCBO Annotator: Semantic Annotation of Biomedical Data'. In: *International Semantic Web Conference, Poster and Demo session*. Vol. 110. Washington DC, USA. 2009.

[103] G. Joshi-Tope et al. 'Reactome: a knowledgebase of biological pathways'. In: *Nucleic Acids Research* 33.suppl_1 (Jan. 2005), pp. D428–D432. ISSN: 0305-1048. DOI: 10.1093/nar/gki072.

[104] Jelena Jovanović and Ebrahim Bagheri. 'Semantic annotation in biomedicine: the current landscape'. In: *Journal of Biomedical Semantics* 8.1 (2017). DOI: 10.1186/s13326-017-0153-x.

[105] Simon Jupp et al. 'A new Ontology Lookup Service at EMBL-EBI.' In: *SWAT4LS*. 2015, pp. 118–119. URL: https://ceur-ws.org/Vol-1546/paper_29.pdf.

[106] Hyeunseok Kang et al. 'Fine-tuning of BERT Model to Accurately Predict Drug–Target Interactions'. In: *Pharmaceutics* 14.8 (Aug. 2022), p. 1710. ISSN: 1999-4923. DOI: 10.3390/pharmaceutics14081710.

[107] Daniel Martin Katz et al. 'GPT-4 Passes the Bar Exam'. In: *Available at SSRN 4389233* (2023). DOI: 10.2139/ssrn.4389233.

[108] Diederik P. Kingma and Jimmy Ba. *ADAM: A Method for Stochastic Optimization*. 2017. DOI: 10.48550/arXiv.1412.6980. arXiv: 1412.6980 [cs.LG].

[109] Durk P Kingma et al. 'Semi-supervised Learning with Deep Generative Models'. In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/d523773c6b194f37b938d340d5d02232-Paper.pdf.

[110] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. DOI: 10.48550/arXiv.1609.02907. arXiv: 1609.02907 [cs.LG].

[111] Sebastian Köhler et al. 'Expansion of the Human Phenotype Ontology (HPO) knowledge base and resources'. In: *Nucleic Acids Research* 47.D1 (Nov. 2018), pp. D1018–D1027. ISSN: 0305-1048. DOI: 10.1093/nar/gky1105.

[112] Satyapriya Krishna et al. *The Disagreement Problem in Explainable Machine Learning: A Practitioner's Perspective*. 2022. DOI: 10.48550/arXiv.2202.01602. arXiv: 2202.01602 [cs.LG].

[113] Alex Krizhevsky, Geoffrey Hinton et al. 'Learning Multiple Layers of Features from Tiny Images'. In: (2009). URL: http://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf.

[114] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. 'ImageNet Classification with Deep Convolutional Neural Networks'. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[115] Brenden M. Lake et al. 'Building machines that learn and think like people'. In: *Behavioral and Brain Sciences* 40 (2017), e253. DOI: 10.1017/S0140525X16001837.

[116] Guillaume Lample et al. *Neural Architectures for Named Entity Recognition*. 2016. DOI: 10.48550/arXiv.1603.01360. arXiv: 1603.01360 [cs.CL].

[117] Ge Lan et al. 'Knowledge Graph Integrated Graph Neural Networks for Chinese Medical Text Classification'. In: *2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 2021, pp. 682–687. DOI: 10.1109/BIBM52615.2021.9669286.

[118] Ora Lassila and Ralph R. Swick. *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation. Feb. 1999. URL: https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

[119] Quoc Le and Tomas Mikolov. 'Distributed Representations of Sentences and Documents'. In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, June 2014, pp. 1188–1196. URL: https://proceedings.mlr.press/v32/le14.html.

[120] Y. LeCun et al. 'Backpropagation Applied to Handwritten Zip Code Recognition'. In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[121] Y. Lecun et al. 'Gradient-based learning applied to document recognition'. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[122] Yann LeCun, Yoshua Bengio and Geoffrey Hinton. 'Deep learning'. In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[123] Yann LeCun et al. 'Efficient BackProp'. In: *Neural Networks: Tricks of the Trade*. Ed. by Genevieve B. Orr and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 9–50. ISBN: 978-3-540-49430-0. DOI: 10.1007/3-540-49430-8_2.

[124] Jinhyuk Lee et al. 'BioBERT: a pre-trained biomedical language representation model for biomedical text mining'. In: *Bioinformatics* 36.4 (2020), pp. 1234–1240. DOI: 10.1093/bioinformatics/btz682.

[125] Bin Liang et al. 'Aspect-based sentiment analysis via affective knowledge enhanced graph convolutional networks'. In: *Knowledge-Based Systems* 235 (2022), p. 107643. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2021.107643.

[126] Tsung-Yi Lin et al. 'Microsoft COCO: Common Objects in Context'. In: *European Conference on Computer Vision 2014*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 740–755. ISBN: 978-3-319-10602-1.

[127] Xuan Lin et al. 'KGNN: Knowledge Graph Neural Network for Drug-Drug Interaction Prediction'. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. Ed. by Christian Bessiere. Main track. International Joint Conferences on Artificial Intelligence Organization, July 2020, pp. 2739–2745. DOI: 10.24963/ijcai.2020/380.

[128] Yankai Lin et al. 'Learning Entity and Relation Embeddings for Knowledge Graph Completion'. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 29.1 (Feb. 2015). DOI: 10.1609/aaai.v29i1.9491.

[129] Liyuan Liu et al. 'Understanding the Difficulty of Training Transformers'. In: *CoRR* abs/2004.08249 (2020). DOI: 10.48550/arXiv.2004.08249. arXiv: 2004.08249.

[130] Siru Liu et al. 'Using AI-generated suggestions from ChatGPT to optimize clinical decision support'. In: *Journal of the American Medical Informatics Association* 30.7 (Apr. 2023), pp. 1237–1245. ISSN: 1527-974X. DOI: 10.1093/jamia/ocad072.

[131] Yinhan Liu et al. 'RoBERTa: A Robustly Optimized BERT Pretraining Approach'. In: *Computing Research Respository (CoRR)* abs/1907.11692 (2019). DOI: 10.48550/arXiv.1907.11692.

[132] Sebastian Lobentanzer et al. 'Democratizing knowledge representation with BioCypher'. In: *Nature Biotechnology* (2023), pp. 1–4. DOI: 10.1038/s41587-023-01848-y.

[133] Manuel Lobo, Andre Lamurias and Francisco M Couto. 'Identifying Human Phenotype Terms by Combining Machine Learning and Validation Rules'. In: *BioMed Research International* 2017 (2017). DOI: 10.1155/2017/8565739.

[134] Mohammad Lotfollahi et al. 'Biologically informed deep learning to query gene programs in single-cell atlases'. In: *Nature Cell Biology* 25.2 (2023), pp. 337–350. DOI: 10.1038/s41556-022-01072-x.

[135] Scott Lundberg and Su-In Lee. *A Unified Approach to Interpreting Model Predictions*. 2017. DOI: 10.48550/arXiv.1705.07874. arXiv: 1705.07874 [cs.AI].

[136] Ling Luo et al. 'PhenoTagger: A Hybrid Method for Phenotype Concept Recognition using Human Phenotype Ontology'. In: *Bioinformatics* 37.13 (July 2021), pp. 1884–1890. DOI: 10.1093/bioinformatics/btab019.

[137] Minh-Thang Luong, Hieu Pham and Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*. 2015. DOI: 10.48550/arXiv.1508.04025. arXiv: 1508.04025 [cs.CL].

[138] Anjun Ma et al. 'Single-cell biological network inference using a heterogeneous graph transformer'. In: *Nature Communications* 14.1 (2023), p. 964. DOI: 10.1038/s41467-023-36559-0.

[139] Xingjun Ma et al. 'Dimensionality-driven learning with noisy labels'. In: (2018). DOI: 10.48550/arXiv.1806.02612. arXiv: 1806.02612 [cs.CV].

[140] Andrew L Maas, Awni Y Hannun, Andrew Y Ng et al. 'Rectifier Nonlinearities Improve Neural Network Acoustic Models'. In: *Proceedings of the International Conference on MAchine Learning*. Vol. 30. 1. Atlanta, GA. 2013, p. 3. URL: http://robotics.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.

[141] Masoud Malekzadeh et al. 'Review of Graph Neural Network in Text Classification'. In: *2021 IEEE 12th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. 2021, pp. 0084–0091. DOI: 10.1109/UEMCON53757.2021.9666633.

[142] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. Software available from tensorflow.org. 2016. DOI: 10.48550/arXiv.1603.04467. arXiv: 1603.04467 [cs.DC]. URL: https://www.tensorflow.org/.

[143] Friedemann Mattern and Christian Floerkemeier. 'From the Internet of Computers to the Internet of Things'. In: *From Active Data Management to Event-Based Systems and More: Papers in Honor of Alejandro Buchmann on the Occasion of His 60th Birthday*. Ed. by Kai Sachs, Ilia Petrov and Pablo Guerrero. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 242–259. ISBN: 978-3-642-17226-7. DOI: 10.1007/978-3-642-17226-7_15.

[144] Andrew McCallum and Wei Li. 'Early Results for Named Entity Recognition with Conditional Random Fields, Feature Induction and Web-Enhanced Lexicons'. In: *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*. CONLL '03. Edmonton, Canada: Association for Computational Linguistics, 2003, pp. 188–191. DOI: 10.3115/1119176.1119206.

[145] D.L. McGuinness et al. 'DAML+OIL: an ontology language for the Semantic Web'. In: *IEEE Intelligent Systems* 17.5 (2002), pp. 72–80. DOI: 10.1109/MIS.2002.1039835.

[146] Deborah L McGuinness, Frank Van Harmelen et al. 'OWL Web Ontology Language Overview'. In: *W3C recommendation* 10.10 (2004). URL: https://static.twoday.net/71desa1bif/files/W3C-OWL-Overview.pdf.

[147] Ryszard S. Michalski. '4 - A THEORY AND METHODOLOGY OF INDUCTIVE LEARNING'. In: *Machine Learning*. Ed. by Ryszard S. Michalski, Jaime G. Carbonell and Tom M. Mitchell. San Francisco (CA): Morgan Kaufmann, 1983, pp. 83–134. ISBN: 978-0-08-051054-5. DOI: 10.1016/B978-0-08-051054-5.50008-X.

[148] Tomas Mikolov et al. 'Distributed Representations of Words and Phrases and their Compositionality'. In: *Advances in Neural Information Processing Systems*. Ed. by C.J. Burges et al. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

[149] Tomas Mikolov et al. 'Efficient estimation of word representations in vector space'. In: (2013). DOI: 10.48550/arXiv.1301.3781. arXiv: 1301.3781 [cs.CL].

[150] Tomáš Mikolov, Wen-tau Yih and Geoffrey Zweig. 'Linguistic Regularities in Continuous Space Word Representations'. In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, Georgia: Association for Computational Linguistics, June 2013, pp. 746–751. URL: https://aclanthology.org/N13-1090.

[151] Tomáš Mikolov et al. 'Recurrent Neural Network Based Language Model'. In: *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*. Ed. by Takao Kobayashi, Keikichi Hirose and Satoshi Nakamura. ISCA, 2010, pp. 1045–1048. DOI: 10.21437/Interspeech.2010-343.

[152] Tomáš Mikolov et al. 'Extensions of Recurrent Neural Network Language Model'. In: *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2011, pp. 5528–5531. DOI: 10.1109/ICASSP.2011.5947611.

[153] George A. Miller. 'WordNet: A Lexical Database for English'. In: *Communications of the ACM* 38.11 (Nov. 1995), pp. 39–41. ISSN: 0001-0782. DOI: 10.1145/219717.219748.

[154] Volodymyr Mnih et al. 'Recurrent Models of Visual Attention'. In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/09c6c3783b4a70054da74f2538ed47c6-Paper.pdf.

[155] Barend Mons et al. 'The value of data'. In: *Nature genetics* 43.4 (2011), pp. 281–283. DOI: 10.1038/ng0411-281.

[156] Barend Mons et al. 'Cloudy, increasingly FAIR; revisiting the FAIR Data guiding principles for the European Open Science Cloud'. In: *Information Services & Use* 37.1 (Mar. 2017), pp. 49–56. DOI: 10.3233/ISU-170824.

[157] Sahiti Myneni et al. 'Towards an Ontology-driven Framework to Enable Development of Personalized mHealth Solutions for Cancer Survivors' Engagement in Healthy Living'. In: *Studies in Health Technology and Informatics* 216 (2015), p. 113. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4946640/.

[158] Vinod Nair and Geoffrey E. Hinton. 'Rectified Linear Units Improve Restricted Boltzmann Machines'. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 9781605589077.

[159] Allen Newell. 'Production Systems: Models of Control Structures'. In: *Visual Information Processing*. Ed. by William G. Chase. Academic Press, 1973, pp. 463–526. ISBN: 978-0-12-170150-5. DOI: 10.1016/B978-0-12-170150-5.50016-0.

[160] Allen Newell and Herbert A. Simon. 'Computer Science as Empirical Inquiry: Symbols and Search'. In: *Communications of the ACM* 19.3 (1976), pp. 113–126. DOI: 10.1145/1283920.1283930.

[161] Gherman Novakovsky et al. 'Obtaining genetics insights from deep learning via explainable artificial intelligence'. In: *Nature Reviews Genetics* 24.2 (2023), pp. 125–137. DOI: 10.1038/s41576-022-00532-2.

[162] Natalya F Noy, Deborah L McGuinness et al. *Ontology Development 101: A Guide to Creating Your First Ontology*. 2001. URL: http://protege.stanford.edu/publications/ontology_development/ontology101.pdf..

[163] Natalya F. Noy et al. 'BioPortal: ontologies and integrated data resources at the click of a mouse'. In: *Nucleic Acids Research* 37.suppl_2 (May 2009), W170–W173. ISSN: 0305-1048. DOI: 10.1093/nar/gkp440.

[164] Desnes Nunes et al. *Evaluating GPT-3.5 and GPT-4 Models on Brazilian University Admission Exams*. 2023. DOI: 10.48550/arXiv.2303.17003. arXiv: 2303.17003 [cs.CL].

[165] OpenAI. *GPT-4 Technical Report*. 2023. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774 [cs.CL].

[166] Shirui Pan et al. *Unifying Large Language Models and Knowledge Graphs: A Roadmap*. 2023. DOI: 10.48550/arXiv.2306.08302. arXiv: 2306.08302 [cs.CL].

[167] Paul Pavlidis and Jesse Gillis. 'Progress and challenges in the computational prediction of gene function using networks: 2012-2013 update'. In: *F1000Research* 2.230 (2013). DOI: 10.12688/f1000research.2-230.v1.

[168] Jeffrey Pennington, Richard Socher and Christopher Manning. 'GloVe: Global Vectors for Word Representation'. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162.

[169] Bryan Perozzi, Rami Al-Rfou and Steven Skiena. 'DeepWalk: Online Learning of Social Representations'. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA: Association for Computing Machinery, 2014, pp. 701–710. ISBN: 9781450329569. DOI: 10.1145/2623330.2623732.

[170] Catia Pesquita et al. 'The epidemiology ontology: an ontology for the semantic annotation of epidemiological resources'. In: *Journal of Biomedical Semantics* 5 (1 2014). DOI: 10.1186/2041-1480-5-4.

[171] Matthew E. Peters et al. 'Deep Contextualized Word Representations'. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202.

[172] Lorcán Pigott-Dix and Robert P. Davey. 'Attention for Multi-Ontology Concept Recognition'. In: *14th International Conference on Semantic Web Applications and Tools for Health Care and Life Sciences (SWAT4HCLS 2023)*. Ed. by Atsuko Yamaguchi et al. Vol. 3415. CEUR Workshop Proceedings. Basel, Switzerland, 2023, pp. 52–61. URL: https://ceur-ws.org/Vol-3415/paper-6.pdf.

[173] Janet Piñero et al. 'The DisGeNET knowledge platform for disease genomics: 2019 update'. In: *Nucleic Acids Research* 48.D1 (Nov. 2019), pp. D845–D855. ISSN: 0305-1048. DOI: 10.1093/nar/gkz1021.

[174] B.T. Polyak. 'Some methods of speeding up the convergence of iteration methods'. In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17. ISSN: 0041-5553. DOI: 10.1016/0041-5553(64)90137-5.

[175] David M. W. Powers. *Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation*. 2020. DOI: 10.48550/arXiv.2010.16061. arXiv: 2010.16061 [cs.LG].

[176] Ning Qian. 'On the momentum term in gradient descent learning algorithms'. In: *Neural Networks* 12.1 (1999), pp. 145–151. ISSN: 0893-6080. DOI: 10.1016/S0893-6080(98)00116-6.

[177] Núria Queralt-Rosinach et al. 'DisGeNET-RDF: harnessing the innovative power of the Semantic Web to explore the genetic basis of diseases'. In: *Bioinformatics* 32.14 (Mar. 2016), pp. 2236–2238. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btw214.

[178] J. Ross Quinlan. 'Induction of Decision Trees'. In: *Machine Learning* 1 (1986), pp. 81–106. DOI: 10.1007/BF00116251.

[179] Alec Radford et al. *Improving Language Understanding by Generative Pre-training*. Tech. rep. OpenAI, 2018. URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

[180] Luc De Raedt et al. *From Statistical Relational to Neuro-Symbolic Artificial Intelligence*. 2020. DOI: 10.48550/arXiv.2003.08316. arXiv: 2003.08316 [cs.AI].

[181] Dietrich Rebholz-Schuhmann et al. 'Text processing through Web services: calling Whatizit'. In: *Bioinformatics* 24.2 (Nov. 2007), pp. 296–298. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btm557.

[182] Leonardo F.R. Ribeiro, Pedro H.P. Saverese and Daniel R. Figueiredo. 'Struc2vec: Learning Node Representations from Structural Identity'. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '17. Halifax, NS, Canada: Association for Computing Machinery, 2017, pp. 385–394. ISBN: 9781450348874. DOI: 10.1145/3097983.3098061.

[183] Marco Tulio Ribeiro, Sameer Singh and Carlos Guestrin. *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. 2016. DOI: 10.48550/arXiv.1602.04938. arXiv: 1602.04938 [cs.LG].

[184] Jenn Riley. *Understanding Metadata: What is Metadata, and What is it For?* Bethesda, MD: National Information Standards Organization (NISO) Press, 2004. ISBN: 978-1-937522-72-8. URL: https://groups.niso.org/higherlogic/ws/public/download/17446/Understanding%20Metadata.pdf.

[185] Bryan Rink and Sanda Harabagiu. 'UTD: Determining Relational Similarity Using Lexical Patterns'. In: *Proceedings of the First Joint Conference on Lexical and Computational Semantics - Volume 1: Proceedings of the Main Conference and the Shared Task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*. SemEval '12. Montréal, Canada: Association for Computational Linguistics, 2012, pp. 413–418. URL: https://dl.acm.org/doi/10.5555/2387636.2387702.

[186] Herbert Robbins and Sutton Monro. 'A Stochastic Approximation Method'. In: *The Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407. ISSN: 00034851. URL: http://www.jstor.org/stable/2236626.

[187] Peter N. Robinson et al. 'The Human Phenotype Ontology: A Tool for Annotating and Analyzing Human Hereditary Disease'. In: *The American Journal of Human Genetics* 83.5 (2008), pp. 610–615. ISSN: 0002-9297. DOI: https://doi.org/10.1016/j.ajhg.2008.09.017. URL: https://www.sciencedirect.com/science/article/pii/S0002929708005351.

[188] F Rosenblatt. 'The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain'. In: *Psychological Review* 65.6 (1958), pp. 386–408. DOI: 10.1037/h0042519.

[189] David E Rumelhart, Geoffrey E Hinton and Ronald J Williams. 'Learning representations by back-propagating errors'. In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0.

[190] Olga Russakovsky et al. 'ImageNet Large Scale Visual Recognition Challenge'. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[191] Ahmed Salih et al. *Commentary on explainable artificial intelligence methods: SHAP and LIME*. 2023. DOI: 10.48550/arXiv.2305.02012. arXiv: 2305.02012 [stat.ML].

[192] Guergana K Savova et al. 'Mayo clinical Text Analysis and Knowledge Extraction System (cTAKES): architecture, component evaluation and applications'. In: *Journal of the American Medical Informatics Association* 17.5 (Sept. 2010), pp. 507–513. ISSN: 1067-5027. DOI: 10.1136/jamia.2009.001560.

[193] Franco Scarselli et al. 'The Graph Neural Network Model'. In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80. DOI: 10.1109/TNN.2008.2005605.

[194] Imanol Schlag et al. 'Enhancing the Transformer With Explicit Relational Encoding for Math Problem Solving'. In: (2019). DOI: 10.48550/arXiv.1910.06611. arXiv: 1910.06611 [cs.LG].

[195] Ute Schmid and Emanuel Kitzelmann. 'Inductive rule learning on the knowledge level'. In: *Cognitive Systems Research* 12.3 (2011). Special Issue on Complex Cognition, pp. 237–248. ISSN: 1389-0417. DOI: 10.1016/j.cogsys.2010.12.002.

[196] Jürgen Schmidhuber. 'Deep learning in neural networks: An overview'. In: *Neural Networks* 61 (2015), pp. 85–117. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2014.09.003.

[197] Lynn M Schriml et al. 'Human Disease Ontology 2018 update: classification, content and workflow expansion'. In: *Nucleic Acids Research* 47.D1 (Nov. 2018), pp. D955–D962. ISSN: 0305-1048. DOI: 10.1093/nar/gky1032.

[198] Lynn Marie Schriml et al. 'Disease Ontology: a backbone for disease semantic integration'. In: *Nucleic Acids Research* 40.D1 (Nov. 2011), pp. D940–D946. ISSN: 0305-1048. DOI: 10.1093/nar/gkr972.

[199] M. Schuster and K.K. Paliwal. 'Bidirectional Recurrent Neural Networks'. In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681. DOI: 10.1109/78.650093.

[200] Elizabeth Seiver, M Pacer and Sebastian Bassi. 'Text and data mining scientific articles with allofplos'. In: *Proceedings of the 17th Python in Science Conference*. Ed. by Fatih Akici et al. 2018, pp. 61–64. DOI: 10.25080/Majora-4af1f417-009.

[201] Rico Sennrich, Barry Haddow and Alexandra Birch. *Neural Machine Translation of Rare Words with Subword Units*. 2016. DOI: 10.48550/arXiv.1508.07909. arXiv: 1508.07909 [cs.CL].

[202] Burr Settles. 'ABNER: an open source tool for automatically tagging genes, proteins and other entity names in text'. In: *Bioinformatics* 21.14 (Apr. 2005), pp. 3191–3192. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bti475.

[203] George J. Shannon et al. 'Comparative study using inverse ontology cogency and alternatives for concept recognition in the annotated National Library of Medicine database'. In: *Neural Networks* 139 (2021), pp. 86–104. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2021.01.018.

[204] Jude W Shavlik, Raymond J Mooney and Geoffrey G Towell. 'Symbolic and Neural Learning Algorithms: An Experimental Comparison'. In: *Machine Learning* 6 (1991), pp. 111–143. DOI: 10.1007/BF00114160.

[205] Kent A Shefchek et al. 'The Monarch Initiative in 2019: an integrative data and analytic platform connecting phenotypes to genotypes across species'. In: *Nucleic Acids Research* 48.D1 (Nov. 2019), pp. D704–D715. ISSN: 0305-1048. DOI: 10.1093/nar/gkz997.

[206] Hoo-Chang Shin et al. *BioMegatron: Larger Biomedical Domain Language Model*. 2020. DOI: 10.48550/arXiv.2010.06060. arXiv: 2010.06060 [cs.CL].

[207] Marta Contreiras Silva et al. 'Ontologies and Knowledge Graphs in Oncology Research'. In: *Cancers* 14.8 (2022). ISSN: 2072-6694. DOI: 10.3390/cancers14081906.

[208] Dylan Slack et al. 'Fooling LIME and SHAP: Adversarial Attacks on Post Hoc Explanation Methods'. In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*. AIES '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 180–186. ISBN: 9781450371100. DOI: 10.1145/3375627.3375830.

[209] Fatima Zohra Smaili, Xin Gao and Robert Hoehndorf. 'Onto2Vec: joint vector-based representation of biological entities and their ontology-based annotations'. In: *Bioinformatics* 34.13 (June 2018), pp. i52–i60. DOI: 10.1093/bioinformatics/bty259.

[210] Fatima Zohra Smaili, Xin Gao and Robert Hoehndorf. *OPA2Vec: combining formal and informal content of biomedical ontologies to improve similarity-based prediction*. 2018. DOI: 10.48550/arXiv.1804.10922. arXiv: 1804.10922 [cs.CL].

[211] Barry Smith et al. 'The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration'. In: *Nature Biotechnology* 25.11 (2007), pp. 1251–1255. DOI: 10.1038/nbt1346.

[212] Nitish Srivastava et al. 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting'. In: *The Journal of Machine Learning Research* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435. URL: https://dl.acm.org/doi/10.5555/2627435.2670313.

[213] Douglas Summers-Stay, Claire Bonial and Clare Voss. 'What Can a Generative Language Model Answer About a Passage?' In: *Proceedings of the 3rd Workshop on Machine Reading for Question Answering*. Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 73–81. DOI: 10.18653/v1/2021.mrqa-1.7.

[214] Martin Sundermeyer, Ralf Schlüter and Hermann Ney. 'LSTM Neural Networks for Language Modeling'. In: *INTERSPEECH 2012, 13th Annual Conference of the International Speech Communication Association, Portland, Oregon, USA, September 9-13, 2012*. ISCA, 2012, pp. 194–197. DOI: 10.21437/Interspeech.2012-65.

[215] Ilya Sutskever, Oriol Vinyals and Quoc V Le. 'Sequence to Sequence Learning with Neural Networks'. In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf.

[216] Mohsen Taheriyan et al. 'Learning the semantics of structured data sources'. In: 37-38 (2016), pp. 152–169. ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2015.12.003.

[217] Hao Tang et al. 'Dependency Graph Enhanced Dual-transformer Structure for Aspect-based Sentiment Classification'. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 6578–6588. DOI: 10.18653/v1/2020.acl-main.588.

[218] Maha A. Thafar et al. 'OncoRTT: Predicting novel oncology-related therapeutic targets using BERT embeddings and omics features'. In: *Frontiers in Genetics* 14 (2023). ISSN: 1664-8021. DOI: 10.3389/fgene.2023.1139626.

[219] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. DOI: 10.48550/arXiv.2307.09288. arXiv: 2307.09288 [cs.CL].

[220] Eugene Tseytlin et al. 'NOBLE–Flexible concept recognition for large-scale biomedical natural language processing'. In: *BMC Bioinformatics* 17.32 (2016). DOI: 10.1186/s12859-015-0871-y.

[221] Ashish Vaswani et al. 'Attention is All you Need'. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[222] Petar Veličković et al. *Graph Attention Networks*. 2018. DOI: 10.48550/arXiv.1710.10903. arXiv: 1710.10903 [stat.ML].

[223] Pascal Vincent et al. 'Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion'. In: *Journal of Machine Learning Research* 11 (Dec. 2010), pp. 3371–3408. ISSN: 1532-4435. URL: https://dl.acm.org/doi/abs/10.5555/1756006.1953039.

[224] Binh Vu, Craig Knoblock and Jay Pujara. 'Learning Semantic Models of Data Sources Using Probabilistic Graphical Models'. In: *The World Wide Web Conference*. WWW '19. San Francisco, CA, USA: Association for Computing Machinery, 2019, pp. 1944–1953. ISBN: 9781450366748. DOI: 10.1145/3308558.3313711.

[225] Li Wan et al. 'Regularization of Neural Networks using DropConnect'. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1058–1066. URL: https://proceedings.mlr.press/v28/wan13.html.

[226] Xiao Wang et al. 'Heterogeneous Graph Attention Network'. In: *The World Wide Web Conference*. WWW '19. San Francisco, CA, USA: Association for Computing Machinery, 2019, pp. 2022–2032. ISBN: 9781450366748. DOI: 10.1145/3308558.3313562.

[227] Xiaozhi Wang et al. 'KEPLER: A Unified Model for Knowledge Embedding and Pre-trained Language Representation'. In: *Transactions of the Association for Computational Linguistics* 9 (Mar. 2021), pp. 176–194. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00360.

[228] P.J. Werbos. 'Backpropagation Through Time: What It Does and How to Do It'. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560. DOI: 10.1109/5.58337.

[229] Patricia L. Whetzel and NCBO Team. 'NCBO Technology: Powering semantically aware applications'. In: *Journal of biomedical semantics*. Vol. 4. suppl_1. BioMed Central. 2013, S8. DOI: 10.1186/2041-1480-4-S1-S8.

[230] B. Widrow and M.E. Hoff. *Adapative Switching Circuits*. Tech. rep. Stanford Electronics Laboratory, Stanford University, 1960. DOI: 10.21236/AD0241531.

[231] Mark D Wilkinson et al. 'The FAIR Guiding Principles for scientific data management and stewardship'. In: *Scientific Data* 3.160018 (2016). DOI: 10.1038/sdata.2016.18.

[232] Ronald J. Williams and David Zipser. 'A Learning Algorithm for Continually Running Fully Recurrent Neural Networks'. In: *Neural Computation* 1.2 (June 1989), pp. 270–280. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.2.270.

[233] Zhen Wu et al. 'UniDrop: A Simple yet Effective Technique to Improve Transformer without Extra Cost'. In: *arXiv preprint arXiv:2104.04946* (2021). DOI: 10.48550/arXiv.2104.04946.

[234] Yan Xia et al. 'Learning Discriminative Reconstructions for Unsupervised Outlier Removal'. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1511–1519. DOI: 10.1109/ICCV.2015.177.

[235] Ruibin Xiong et al. 'On Layer Normalization in the Transformer Architecture'. In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 10524–10533.

[236] Shankai Yan and Ka-Chun Wong. 'Elucidating high-dimensional cancer hallmark annotation via enriched ontology'. In: *Journal of Biomedical Informatics* 73 (2017), pp. 84–94. ISSN: 1532-0464. DOI: 10.1016/j.jbi.2017.07.011.

[237] Xi Yang et al. 'A large language model for electronic health records'. In: *npj Digital Medicine* 5.1 (2022), p. 194. DOI: 10.1038/s41746-022-00742-2.

[238] Liang Yao, Chengsheng Mao and Yuan Luo. *KG-BERT: BERT for Knowledge Graph Completion*. 2019. DOI: 10.48550/arXiv.1909.03193. arXiv: 1909.03193 [cs.CL].

[239] Hai-Cheng Yi et al. 'Graph representation learning in bioinformatics: trends, methods and applications'. In: *Briefings in Bioinformatics* 23.1 (2022). bbab340. ISSN: 1477-4054. DOI: 10.1093/bib/bbab340.

[240] Seung Yon Rhee et al. 'Use and misuse of the gene ontology annotations'. In: *Nature Reviews Genetics* 9.7 (2008), pp. 509–515. DOI: 10.1038/nrg2363.

[241] Donghan Yu et al. 'Knowledge Embedding Based Graph Convolutional Network'. In: *Proceedings of the Web Conference 2021*. WWW '21. Ljubljana, Slovenia: Association for Computing Machinery, 2021, pp. 1619–1628. ISBN: 9781450383127. DOI: 10.1145/3442381.3449925.

[242] Jinxing Yu et al. 'MQuadE: A Unified Model for Knowledge Fact Embedding'. In: *Proceedings of the Web Conference 2021*. WWW '21. Ljubljana, Slovenia: Association for Computing Machinery, 2021, pp. 3442–3452. ISBN: 9781450383127. DOI: 10.1145/3442381.3449879.

[243] Seongjun Yun et al. 'Graph Transformer Networks'. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/9d63484abb477c97640154d40595a3bb-Paper.pdf.

[244] Seongjun Yun et al. 'Graph Transformer Networks: Learning meta-path graphs to improve GNNs'. In: *Neural Networks* 153 (2022), pp. 104–119. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2022.05.026.

[245] Matthew D. Zeiler. *ADADELTA: An Adaptive Learning Rate Method*. 2012. DOI: 10.48550/arXiv.1212.5701. arXiv: 1212.5701 [cs.LG].

[246] Xiangxiang Zeng et al. 'Toward better drug discovery with knowledge graph'. In: *Current Opinion in Structural Biology* 72 (2022), pp. 114–126. ISSN: 0959-440X. DOI: https://doi.org/10.1016/j.sbi.2021.09.003.

[247] Mi Zhang and Tieyun Qian. 'Convolution over Hierarchical Syntactic and Lexical Graphs for Aspect Level Sentiment Analysis'. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 3540–3549. DOI: 10.18653/v1/2020.emnlp-main.286.

[248] Yufeng Zhang et al. *Every Document Owns Its Structure: Inductive Text Classification via Graph Neural Networks*. 2020. DOI: 10.48550/arXiv.2004.13826.

[249] Zhao Zhang et al. 'Relational Graph Neural Network with Hierarchical Attention for Knowledge Graph Completion'. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.05 (Apr. 2020), pp. 9612–9619. DOI: 10.1609/aaai.v34i05.6508.

[250] Zhiyuan Zhang et al. 'Pretrain-KGE: Learning Knowledge Representation from Pretrained Language Models'. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 259–266. DOI: 10.18653/v1/2020.findings-emnlp.25.

[251] Ziqi Zhang. 'Effective and Efficient Semantic Table Interpretation Using TableMiner+'. In: *Semant. Web* 8.6 (Jan. 2017), pp. 921–957. ISSN: 1570-0844. DOI: 10.3233/SW-160242.

[252] Shi Zhi et al. 'Partially-Typed NER Datasets Integration: Connecting Practice to Theory'. In: (2020). DOI: 10.48550/arXiv.2005.00502. arXiv: 2005.00502 [cs.LG].

[253] Ce Zhou et al. *A Comprehensive Survey on Pretrained Foundation Models: A History from BERT to ChatGPT*. 2023. DOI: 10.48550/arXiv.2302.09419. arXiv: 2302.09419 [cs.AI].

[254] Wangchunshu Zhou et al. 'Scheduled DropHead: A Regularization Method for Transformer Models'. In: *arXiv preprint arXiv:2004.13342* (2020). DOI: 10.48550/arXiv.2004.13342.

[255] Qian Zhu et al. 'Exploring the Pharmacogenomics Knoledge Base (PharmGKB) for Repositioning Breast Cancer Drugs by Leveraging Wed Ontology Language (OWL) AND Cheminformatics Approaches'. In: *Biocomputing 2014*, pp. 172–182. DOI: 10.1142/9789814583220_0017.

[256] Marinka Zitnik, Monica Agrawal and Jure Leskovec. 'Modeling polypharmacy side effects with graph convolutional networks'. In: *Bioinformatics* 34.13 (June 2018), pp. i457–i466. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty294.

Appendix A

Appendix

A.1 Chapter 5

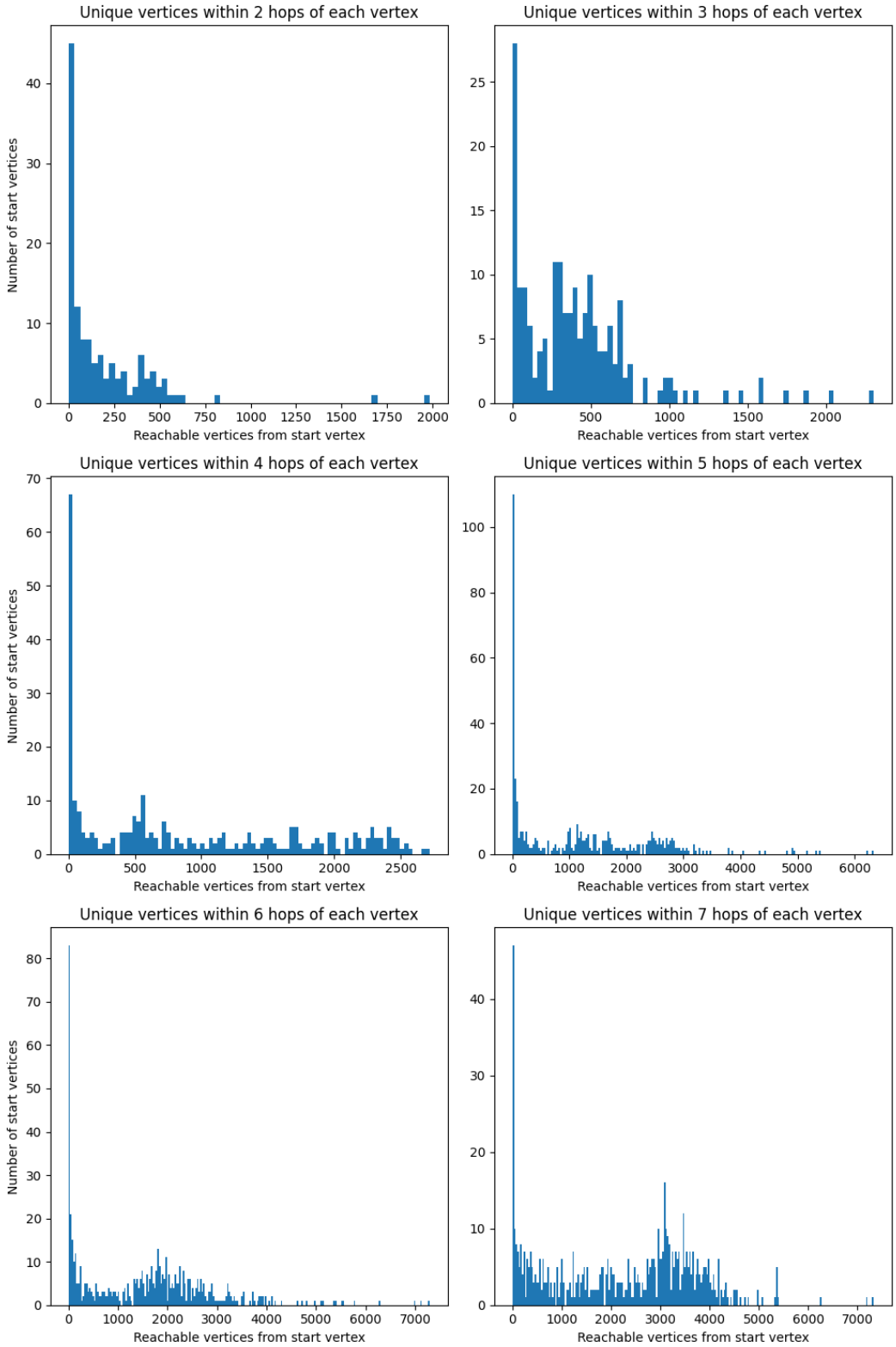

Figure A.1: Histograms tallying the number of unique vertices reachable within N-hops of every vertex in the graph. The bin size for every plot is 32.

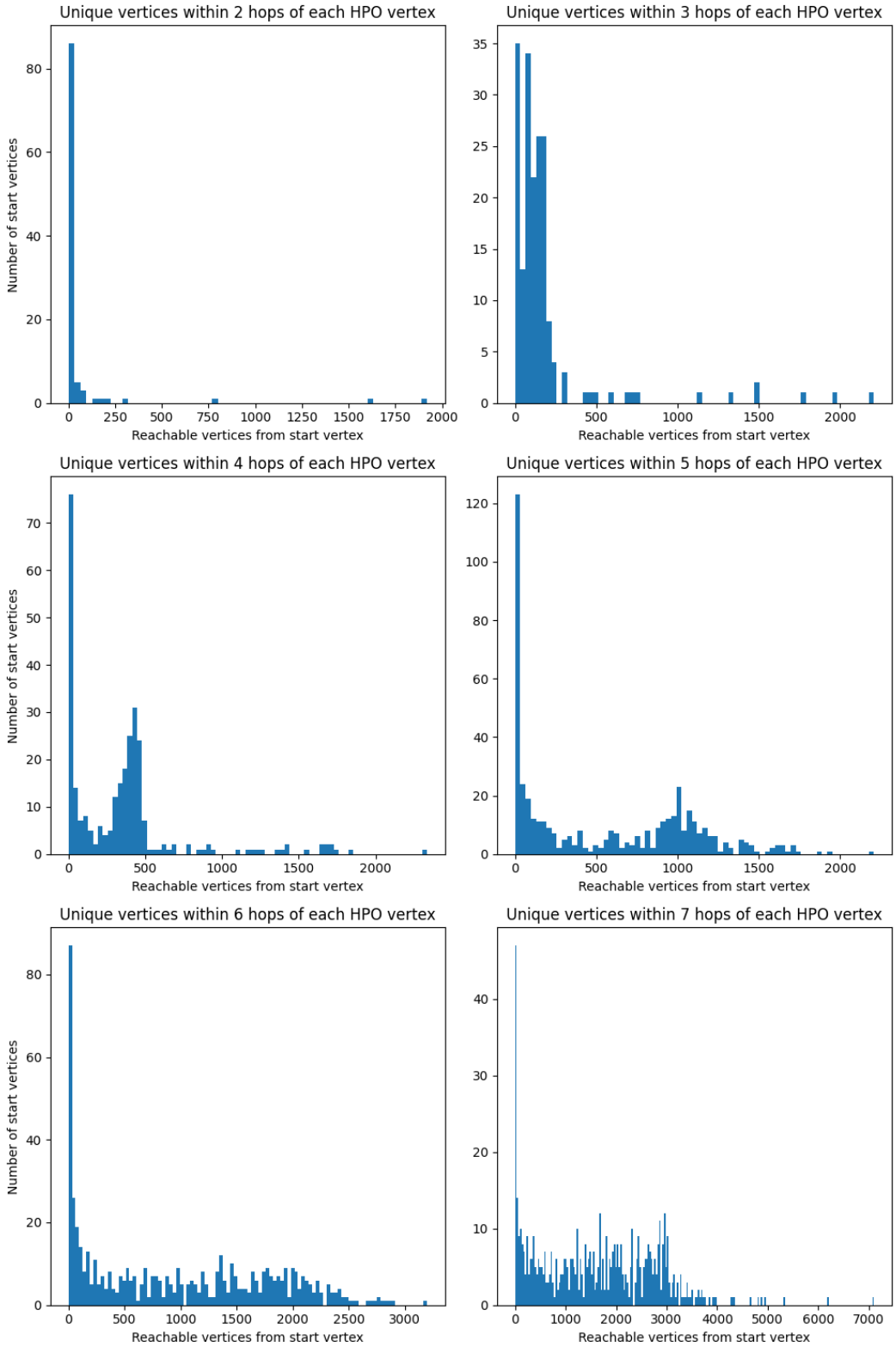

Figure A.2: Histograms tallying the number of unique vertices reachable within N-hops of every Human Phenotype Ontology concept vertex in the graph. The bin size for every plot is 32.

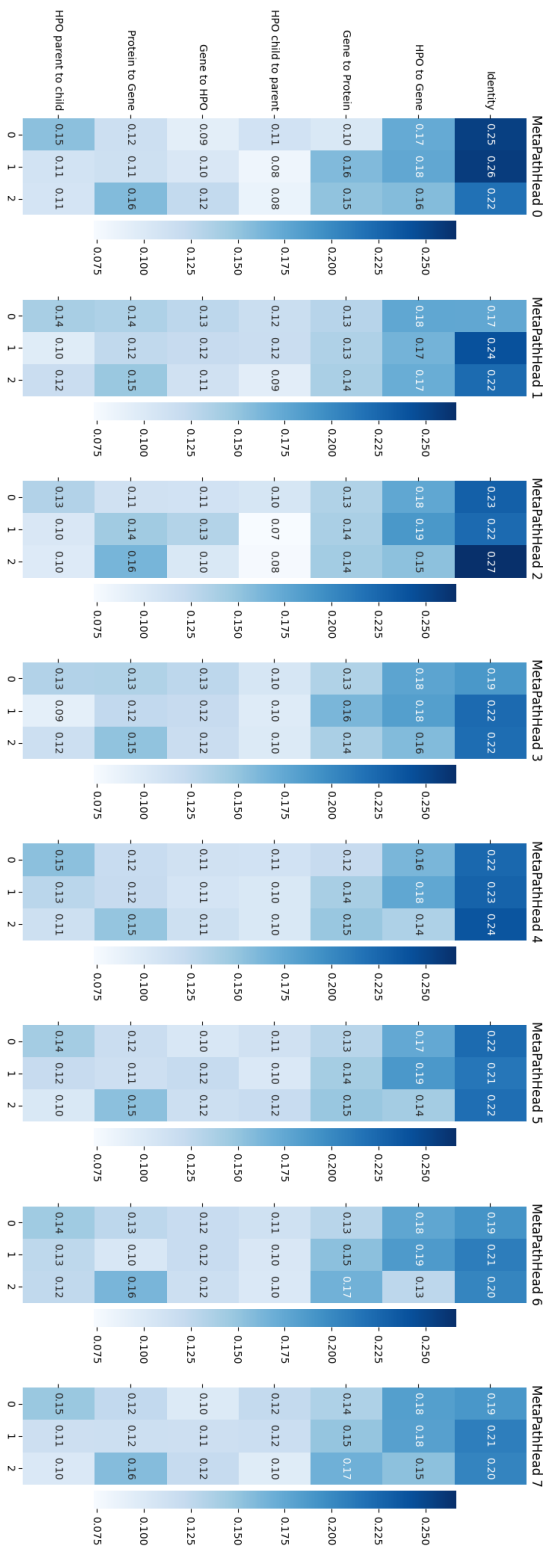

Figure A.3: The softmax of the filter kernels learned by the Meta-path transformer, following training on the text classification task.

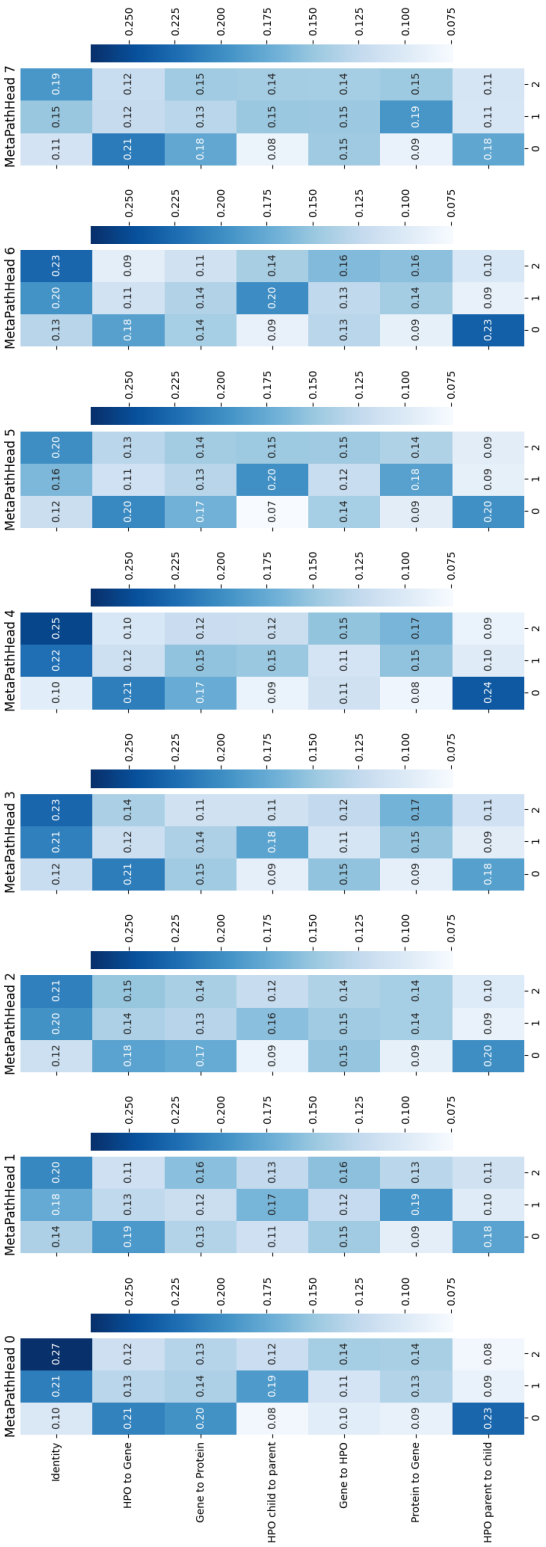

Figure A.4: The softmax of the filter kernels learned by the Meta-path transformer, following pretraining on the walk validity task.

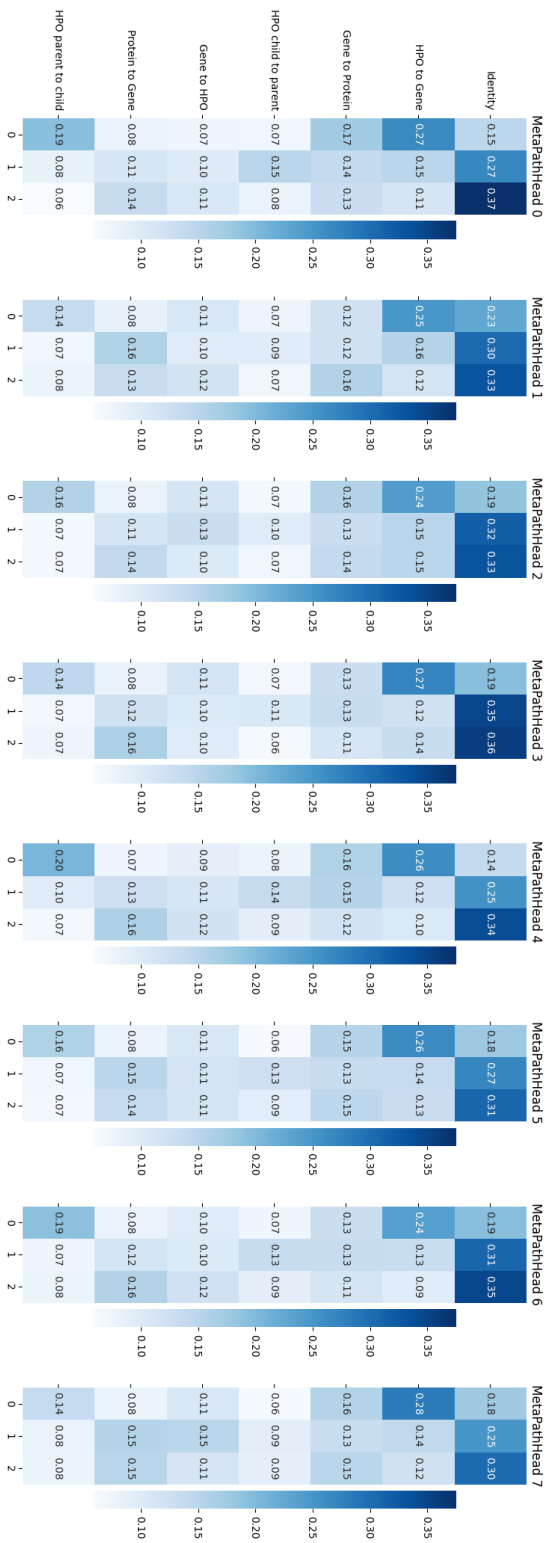

Figure A.5: The softmax of the filter kernels learned by the Meta-path transformer, after both pretraining on the walk validity task and then training on the text classification task.

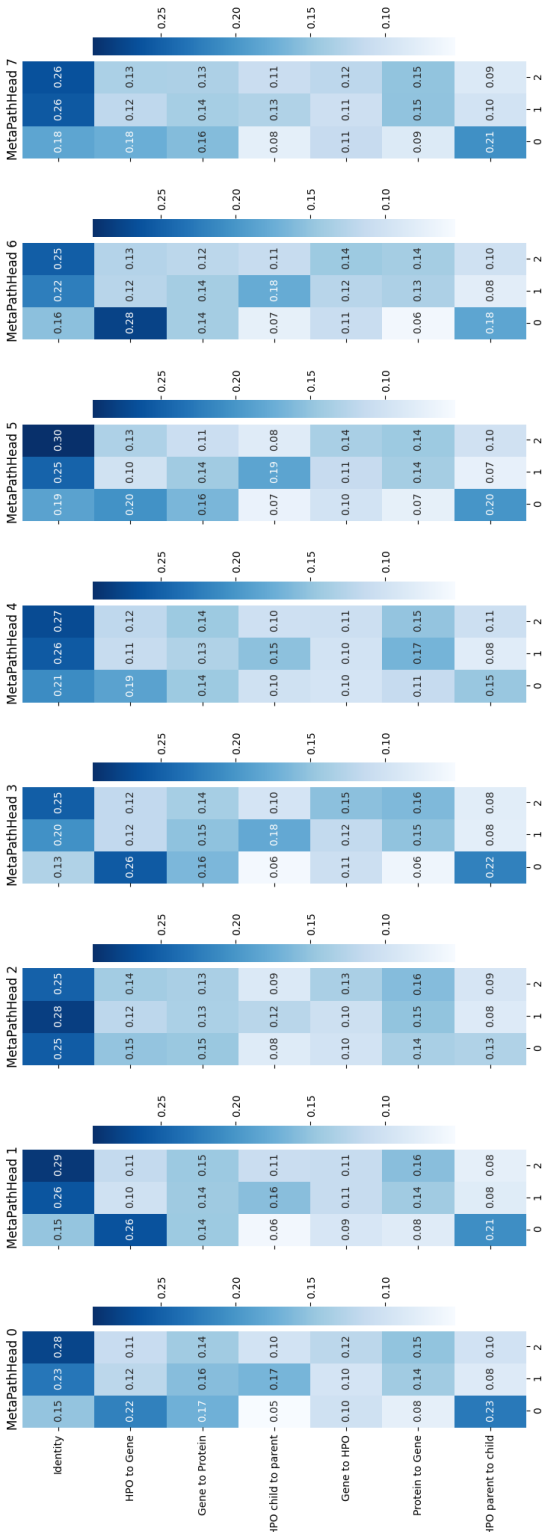

Figure A.6: The softmax of the filter kernels learned by the Meta-path transformer, after being trained simultaneously on the walk validity and text classification tasks.

Table A.1: Metrics for the walk-validity pre-training of the Meta-Path Transformer.

Epoch	Training Loss	Test Loss	Test Accuracy
1	31.465	0.786	0.525
2	28.023	0.690	0.560
3	26.453	0.648	0.581
4	25.060	0.590	0.604
5	23.247	0.508	0.632
6	20.659	0.390	0.664
7	17.659	0.301	0.695
8	15.769	0.253	0.719
9	14.345	0.194	0.742
10	12.470	0.182	0.762
11	10.461	0.231	0.775
12	10.741	0.104	0.790
13	9.618	0.113	0.803
14	8.161	0.071	0.816
15	10.516	0.266	0.820
16	8.060	0.095	0.829
17	8.842	0.091	0.837
18	5.225	0.105	0.844
19	6.762	0.071	0.851
20	5.850	0.121	0.856
21	4.283	0.065	0.862
22	7.137	0.090	0.867
23	4.893	0.071	0.871
24	5.082	0.065	0.876
25	3.857	0.036	0.880
26	3.417	0.062	0.884
27	3.929	0.043	0.888
28	3.013	0.069	0.891
29	3.098	0.050	0.894
30	3.460	0.052	0.897
31	2.780	0.052	0.900
32	2.878	0.049	0.902
33	2.510	0.042	0.905
34	2.851	0.070	0.907
35	2.696	0.039	0.909
36	4.554	0.042	0.912
37	2.959	0.018	0.914
38	2.382	0.046	0.916
39	2.186	0.041	0.918
40	1.910	0.021	0.920
41	2.427	0.080	0.921
42	2.677	0.045	0.922
43	2.031	0.055	0.924
44	1.542	0.018	0.925
45	1.533	0.020	0.927
46	1.502	0.047	0.928
47	1.300	0.022	0.929
48	1.593	0.064	0.931
49	1.702	0.014	0.932
50	1.257	0.043	0.933
51	1.441	0.038	0.934
52	2.680	0.066	0.935
53	2.784	0.024	0.936
54	1.774	0.026	0.937
55	1.395	0.022	0.938