# Link prediction in knowledge graphs using latent feature modelling and neural tensor factorisation

**Luyolo Magangane**


Thesis presented in partial fulfilment of the requirements for the degree of Master of Science (Applied Mathematics) in the Faculty of Science at Stellenbosch University

Supervisor: Prof. Willie Brink

December 2020

# Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

December 2020

I would like to dedicate this thesis to my loving parents, Duduzile Magangane and Ludumo Magangane, without whom this would not have been possible. They have supported me every step of the way in my education over the past 20 years (or so). They continue to inspire me to reach higher in my quest for knowledge, and I hope this contribution makes them proud.

I would also like to dedicate this thesis to my loving siblings, Yoliswa Musvosvi, Phathiswa Magangane and Phumla Zondo, whose safety and peace are the reason for everything I do. Their never ending energy has fuelled me at many points of giving up, and this contribution is as much theirs as it is mine. I'd also like to dedicate this thesis to my loving second mothers, Busisiwe Zondo and Patricia Zondo. Their warm support always inspires me to strive for the best, helping me achieve more than I ever would of on my own.

Finally, I'd like to dedicate this thesis to my late grandmother, Nomaziya Zondo. I've never known a more gentle and wise person. I hope to be more like her every day.

# Abstract

Reasoning over knowledge expressed in natural language is a problem at the forefront of artificial intelligence. Question answering is one of the core tasks of this problem, and is concerned with giving machines the capability of generating an answer given a question, by mimicking the reasoning behaviour of humans. Relational learning, in combination with information retrieval, has been explored as a framework for solving this problem. Knowledge graphs (KGs) are used to represent facts about multiple domains as entities (nodes) and relations (edges), and the resource description framework formalism, subject-predicate-object, is used to encode these facts. Link prediction then powers knowledge discovery by scoring possible relationships between entities.

This thesis explores latent feature modelling using tensor factorisation as an approach to link prediction. Tensor decompositions are an attractive approach as relational domains are usually high-dimensional and sparse, a setting where factorisation methods have shown very good results. Previous approaches have focused on shallow models that can scale to large datasets, and recently deep models have been applied, specifically neural tensor factorisation models, as these models are more expressive and automatically learn the most useful latent features for entities and relations. In this work we introduce training algorithm optimisations to the neural tensor network (NTN) and HypER neural tensor factorisation models.

We make use of the TensorFlow reimplementation of NTNs and apply early stopping, adaptive moment estimation and hyperparameter optimisation using random search. We see improvements in both cost and accuracy over the baseline NTN reimplementation, using standard link prediction benchmark datasets WordNet and Freebase. We then apply optimisations to the HypER model training algorithm. We begin with compensating for covariate shift caused by hypernetworks, using batch normalisation, and propose HypER+. We see similar performance to the HypER baseline on the WN18 dataset, and see significant improvement using the FB15k dataset. We extend our optimisation by initialising entity and relation embeddings using pre-trained word vectors from the GloVe language model. We see marginal improvements over the baseline using the WN18RR and FB15k-237 datasets. Our results establish HypER+ as a state-of-the-art model in latent feature modelling based link prediction.

# Opsomming

Redenering oor kennis wat in natuurlike taal uitgedruk word, is ’n probleem aan die voorpunt van kunsmatige intelligensie. Die beantwoording van vrae is een van die kerntake van hierdie probleem, en poog om masjiene die vermoë te gee om ’n antwoord te skep vir ’n gegewe vraag, deur die redenasiegedrag van mense na te boots. Verhoudingsleer, in kombinasie met die inwin van inligting, is al ondersoek as ’n raamwerk vir die oplossing van hierdie probleem. Kennisgrafieke (KG’s) word gebruik om feite oor veelvuldige domeine as entiteite (punte) en verhoudings (lyne) voor te stel, en die bronbeskrywingsraamwerk-formalisme, nl. onderwerp-predikaat-voorwerp, word gebruik om sulke feite te enkodeer. Skakelvoorspelling dryf dan kennisontdekking deur moontlike verhoudings tussen entiteite te bepunt.

Hierdie tesis ondersoek latente kenmerkmodellering met behulp van tensorfaktorisering, as ’n benadering tot skakelvoorspelling. Tensor-ontbindings is ’n aantreklike benadering, aangesien verhoudingsdomeine gewoonlik hoogdimensioneel en yl is; omstandighede waar faktoriseringsmetodes reeds baie goeie resultate getoon het. Vorige benaderings het op vlak modelle gefokus, wat kan skalleer met groot datastelle. Meer onlangs is diep modelle toegepas, spesifiek neurale tensorfaktoriseringsmodelle, aangesien hierdie modelle meer ekspressief is en outomaties die nuttigste latente kenmerke vir entiteite en verhoudings kan aanleer. In hierdie werk stel ons optimering van afrigalgoritmes voor vir die neurale tensornetwerk (NTN) en HypER neurale tensorfaktoriseringsmodelle.

Ons maak gebruik van die TensorFlow-herimplementering van NTN’s, en pas vroeë-stop, aanpasbare momentskatting, sowel as hiperparameteroptimering met ewekansige soeke, toe. Ons sien verbeterings in koste sowel as akkuraatheid oor die basiese NTN-herimplementering, in die standaard skakelvoorspellingsdatastelle WordNet en Freebase. Ons pas dan optimerings toe op die HypER-model se afrigtingsalgoritme. Ons begin met die kompensering van kovariantskuif wat deur hipernetwerke veroorsaak word, met behulp van bondelnormalisering, en stel HypER+ voor. Ons sien prestasies soortgelyk aan die HypER-basismodel op die WN18-datastel, en beduidende verbetering op die FB15k-datastel. Ons brei ons optimering uit deur entiteit- en verhoudingsinbeddings te inisialiseer met vooraf-afgerigte woordvektore van die GloVe-taalmodel. Ons sien marginale verbeterings oor die basismodel op die WN18RR en FB15k-237 datastelle. Ons resultate vestig HypER+ as ’n mededingende model in latente kenmerkmodelleringsgebaseerde skakelvoorspelling.

# Table of contents

# Chapter 1

# Introduction

## 1.1 Overview

*"... as they say, your model is only as good as the data you have.*
*So it all starts with the data ..."*
- Nyalleng Moorosi, *Deep Learning Indaba* (2017)

Artificial intelligence (AI) is the imitation of human-level intelligence by machines. Human beings are capable of extracting information from a passage of text, and transforming that information into a mental model of objects and relationships between those objects. As stated by the quote above, the model is only as good as the data available for training, and this is true for both human mental models as well as machine constructed models.

Human models can be confined to a single domain [83], for example the entertainment industry. The entertainment industry contains objects such as films, actors, actresses, characters and awards. These objects have relationships between them, such as "starred in", "worked along side", "a character in", and "was nominated for". Humans can develop a mental model to easily reason about new relationships between the objects that were not mentioned in a given passage of text. For example the text may say that Chadwick Boseman starred in a film called Black Panther and played the character Black Panther. Using that information we can reason that Chadwick Boseman is an actor.

We use such models to read and understand other passages of text, in conversations with other people, and to help us answer questions. The set of objects and the relationships between them, within a domain, can be thought of as knowledge. Figure 1.1 shows what (limited) knowledge about the entertainment industry might look like.

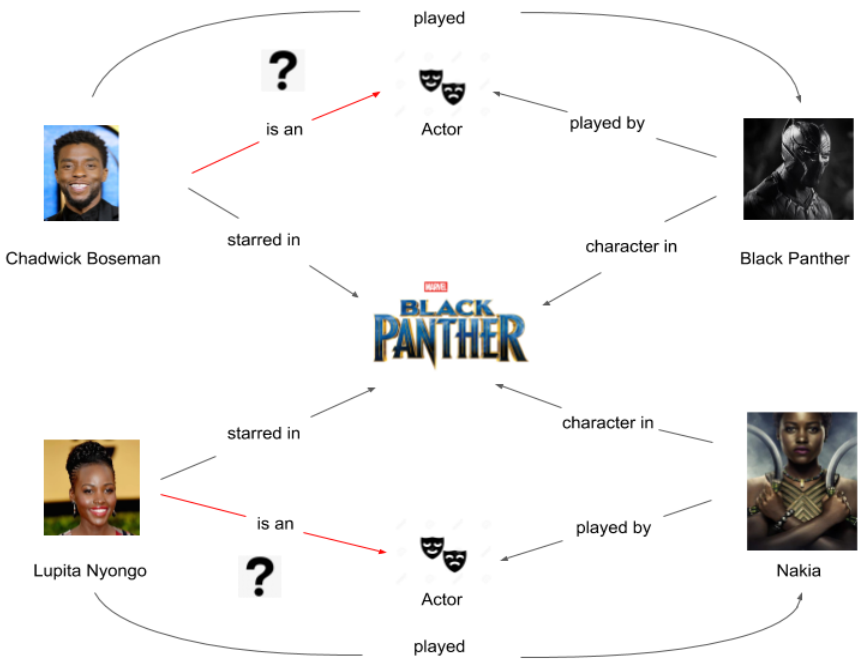



Figure 1.1: Objects and relationships between them. We see Chadwick Boseman starred in the Black Panther film and played the Black Panther character. We also see that the character was played by an actor, and can reason that Chadwick Boseman may thus be an actor.

### 1.1.1 Challenges

Reasoning about knowledge expressed in natural language [56] is simple for humans, but trying to imitate this behaviour on a computer reveals its underlying complexity.

The first task is providing a mechanism for perceiving syntax. Humans recognise text as sequences of words, and words as a sequence of characters. Characters can belong to different writing systems which can be dense (have a few characters used in a large number of combinations), or sparse (have a large number of characters used in fewer combinations) [77]. A computer has to first be able to perceive these characters.

The second task is even more challenging: modelling semantics. Semantics is the meaning of words in language [18]. It allows humans to understand each other in conversation, through written text, and through visual imagery. Semantics allows us to understand that an actor is a thing, and that things can have relationships with other things. Giving computers the capability of semantic understanding is challenging because of the unstructuredness of language. If a person is shown a word in the singular and a word in the plural, for example "film" and "films", they will most likely understand that there is not much difference in meaning between the

two words, while a computer may interpret them as having completely different meanings. Different words can also have similar meanings, for example "film" and "movie". Capturing this similarity is once again challenging for machines. And finally the order in which words are seen can provide context, for example the word "star" has completely different meanings in "Chadwick Boseman is a movie star" and "Black Panther looked up into the night sky and saw a star".

The third challenging task is organising information in such a way as to be able to reason about the knowledge it conveys. Humans build mental models using default reasoning [73], believing that most objects A have some relationship X with objects B, with a small number of exceptions. For example, we could believe that films (object A) have not won (relationship X) an Oscar (object B). We will believe this is true for any given movie, unless we are familiar enough with movie history to identify the exceptions. Formally, the default beliefs we hold are facts, and this process of reasoning allows us to infer new facts in the form of plausible relationships between objects.

In order to allow computers to use a similar method of reasoning, facts can be represented as graph-structured data. Such a representation is called a knowledge graph (KG) and models facts as entities (nodes) and the relations (edges) between them [62]. The resource description framework (RDF) formalism is used to encode facts as triples in the form, subject-predicate-object, where the subject and object are entities, and the predicate is a relation between two entities [9]. For example, a fact in a KG could be Black Panther (subject) is a (predicate) super hero (object).

Question answering (QA) relies on knowledge discovery, the step-by-step deductive process used to infer possible facts given known facts. This is the process used above to infer that Chadwick Boseman is an actor, given other facts known about Chadwick Boseman and the Black Panther film. Statistical relational learning (SRL) solves the knowledge discovery problem by constructing models with measures of uncertainty in plausible facts not contained in KGs [46].

### 1.1.2 Milestones

The task of QA can be extended to an open-domain setting [16], where the task is extended from answering questions from a single domain to multiple domains. In this setting, a KG contains data from multiple domains and more complex knowledge discovery chains can be constructed. There has been encouraging progress in SRL to model this problem, and link prediction, inferring a plausible edge (relationship) between two KG nodes (entities), is now

often used as a paradigm for knowledge discovery [47, 25, 61]. Latent feature modelling using tensor factorisation [32, 45] is an approach to link prediction that has seen some promising results [14, 40, 63].

The first major milestone was demonstrating the bilinear tensor product as a promising model for link prediction [64]. In this model, the multiplication is taken between a subject vector and a predicate matrix, producing an entity-relational vector. The dot product is then taken between the entity-relational vector and an object vector, which generates a relational score between the subject and object entities.

The second milestone was the integration of the bilinear tensor product and neural networks as the Neural Tensor Network (NTN) [79]. In this approach a recursive network [70] computes a compositional score between the subject and object, which is then added to a bilinear tensor product score. The result is then passed through a non-linear function which generates the final relational score. This approach effectively extended linear tensor factorisation techniques to nonlinear techniques, and simultaneously introduced the use of pre-trained word embeddings to initialise entity and relation vectors instead of random initialisation. In this one approach both decision making expressiveness as well as entity and relation semantic representations were improved.

The third time a milestone was realised was with the use of complex valued embeddings for link prediction [88]. This approach extended the bilinear tensor product by making use of the Hermitian dot product, which is the complex counterpart of the standard dot product between real vectors. It was proposed that complex vectors can effectively capture antisymmetric relations, e.g. "is not", while retaining the efficiency benefits of the dot product. This was achieved by using representations with complex embeddings, allowing the model to capture semantic meaning by computing a relational score dependent on the order of the entities in the triple.

The above three milestones relied on shallow models that could scale to large datasets. Up until that point research in link prediction had focused on minimising the parameterisation of models. Convolutional networks are parameter efficient models, and progress was again realised with the use of deep convolutional models for link prediction [19]. This approach makes use of $2D$ entity and relation representations that undergo a convolution operation. This enables the modelling of a large number of interactions between them, and convolutional models have achieved state-of-the-art (SOTA) link prediction performance as a result.

### 1.1.3 Remaining challenges

Open-domain knowledge graph question answering (KGQA) is getting closer to realising a form of generalisation for AI reasoning. One significant challenge still faced by this framework is automatic construction of KGs [22]. These large data sources are required to adequately train link prediction models. Combining such data sources for sufficient knowledge density remains an open area of research [20]. Relying on systems with a graph structure alone limits the set of available resources which can be used to acquire knowledge, which necessitates the use of other source types including news document sets amongst others. Presenting this information poses a significant data transformation challenge, as well as potentially requiring continual online learning of the model as new data becomes available [3].

It is also currently unclear which link prediction paradigm is the most useful in knowledge discovery. In addition to latent feature modelling, graph modelling [66, 76, 69] and inductive probabilistic logic programming (IPLP) [81] are two approaches also used for link prediction. Graph modelling is a generalisation of latent feature modelling which aggregates local and global entity neighbourhood context to generate relational scores between entities. IPLP introduces stochastic primitives to logic programming languages. Recently graph modelling has been receiving increased attention as an approach to link prediction, and may reveal efficient knowledge discovery patterns.

Finally, there seems to be large headroom for improvement in latent feature modelling based link prediction. SOTA performance on standard benchmark datasets, where the task of discovering the top 10 most plausible facts given an entity-relational composition, is currently achieved by a graph modelling technique [75]. There is currently no human-level benchmark with which to contrast this performance, and it is also unclear whether improvement in link prediction performance can be realised by further optimisations to tensor factorisation approaches, or if a paradigm shift to graph modelling or IPLP will be required.

### 1.1.4 Outline of contributions

We begin by introducing training algorithm optimisations to a TensorFlow [2] reimplementation of the NTN model [23] in an attempt to improve its link prediction performance. We apply early stopping [71], a regularisation technique that prevents overfitting of training data. We aslo apply adaptive moment estimation (Adam) [43], a parameter optimisation technique that makes use of an adaptive learning rate, where the learning rate is computed using the first- and

second-order moments of the gradient. We then make use of hyperparameter random search [7] to optimise the model.

HypER [5] is a neural tensor factorisation model [92] that introduces hypernetworks [30] to latent feature modelling based link prediction. Hypernetworks are a class of meta-networks used to configure a main network. They generate the weights of a main network from an embedding vector that contains a compressed encoding of the weights. The generated weights are subsequently used in the forward pass of the main network. HypER uses a multilayer perceptron (MLP) to transform relational vectors into relation-specific convolutional filters used by the main network. Hypernetworks suffer from covariate shift [39], the change in the distribution of inputs to the current layer, caused by the simultaneous update of current layer weights and previous layer weights during backpropagation. Covariate shift slows down training and degrades model performance during inference. We optimise the HypER training algorithm by applying batch normalisation as a hypernetwork model layer, which compensates for covariate shift, and implement a new model called HypER+ with this consideration.

Language models [89] are neural networks trained using very large text corpora. These models attempt to build a structural understanding of the syntax and semantics of a language, and are used to obtain pre-trained word vectors [54] for downstream tasks. Leveraging semantic information from pre-trained word vectors can provide richer latent representations for entities and relations, which may improve inference during knowledge discovery. We thus integrate pre-trained word vectors into the HypER+ training algorithm and use them to initialise entity and relational vectors in place of Xavier initialisation [26].

### 1.1.5 Long-term motivations

Imitation of human-level intelligence persists as an inspirational challenge to science. This challenge gave rise to the first attempts at AI research beginning with the perceptron [74], and continues to inspire the fervour present today. Intelligent virtual assistants like Ironman's J.A.R.V.I.S. (Just A Rather Very Intelligent System) [1], serve as a North Star for the desired capabilities of natural language based AI systems. These systems can be thought of as artificial general intelligence (AGI), indistinguishable from human capabilities in reasoning, dialogue and humor. Possible applications of such a technology are numerous, where perhaps one of the most useful may be an AI tutoring interface for education. South Africa experiences many educational challenges, and such a system could have a profound impact in helping students. Open-domain question answering becomes particularly important in this setting, where practical implementations may be grounded in continuous online training of knowledge

discovery models. This thesis explores ideas that aim to contribute incremental progress toward AGI.

### 1.1.6 Short-term motivations

One of the challenges facing KGQA is that there exists no standard benchmark of human-level performance in knowledge discovery. Such a benchmark could be used to more qualitatively assess link prediction performance of existing approaches. The current SOTA link prediction model's performance may be consistent with the minimum level of performance required to sufficiently satisfy user questions, however until baseline metrics exist with which to measure this, it may remain difficult to quantitatively validate gains in performance.

The most appropriate paradigm for knowledge discovery is also still an open question. Graph modelling has the advantage of longer sequences with which to build dependencies between entities. Such an advantage allows the modelling of relational correlations between entities that is not possible in latent feature modelling. Latent feature modelling has the advantage of computational efficiency. No matter how large the KG, latent feature modelling approaches remain adept at handling this density. Both of these paradigms are stochastic, however, and in certain scenerios more determinstic approaches may be required, a strength of IPLP. Explainability of logic rules with stochastic primitives [94] may be useful in safety-critical applications, or under decision-making understanding contexts.

Knowledge discovery is currently fulfilled through the task of prediction. Two machine learning paradigms have been used to structure approaches to this task: probabilistic modelling and mathematical estimation of generative distributions. Both of these paradigms rely on in-depth a priori structuring of a model used as an approach to prediction, for example Bayesian inference, shallow models and deep models [59]. Each paradigm has given rise to a number of models and corresponding training algorithms with inherent strengths and weaknesses. The trend in prediction research however is simple models of ever increasing size, trained on very very large datasets. It is unclear whether careful curation of model definitions and training algorithms will be able to continue to compete with this more brute force alternative. Exploring the dissymmetric opposition of these two philosophies should guide the development of the most efficient solutions.

### 1.1.7 Challenges of latent feature modelling

Tensor factorisation is the most common approach for latent feature modelling based link prediction. This model has a number of attractive properties including simplicity, effective handling of sparse and high dimensional data, as well as compute efficiency. Extensions to linear tensor factorisation approaches have involved integration with deep models. This approach has been effective in improving performance, however recent results suggest there are higher levels to be achieved. Deep models typically contain a large number of parameters, and as a result are prone to poor link prediction generalisation. Training algorithm optimisations, specifically regarding regularisation, have proved useful in improving model generalisation. Increasing model complexity while preserving test set generalisation continues to be a challenge.

Until recently, entity-relational interaction modelling has posed a significant challenge. Expressing a variety of dyadic interactions (e.g. antisymmetry) between entities, as well as modelling KG nodes with high in-degree have proved problematic. The use of nonlinear factorisation models has improved interaction representation. Determining model operators which extend expressiveness in interaction modelling may be useful in further improving performance.

Generating entity and relation representations is another challenge. Previous work represented entities and relations using randomly initialised, static embeddings. This approach does not allow the sharing of statistical strength between the embeddings describing each entity or relation. Chen et al. [79] showed that using pre-trained word vectors instead of randomly initialised word embeddings improves link prediction performance. This performance gain can be explained by the co-occurrence statistics used to train word vectors. Language models [10, 90] are thus useful in obtaining meaningful semantic representations for entity and relation vectors. They may also be useful in compensating for sparsity in interaction observations.

Compute resources required to perform inference also present a challenge. For any given entity-relational input pair, current methods compute a relation prediction score for every entity within the KG. This is not a challenge for KGs with a small number of entities (thousands), however poses a scalability problem for KGs with a large number entities (millions). Compute requirements thus continue to place a parameterisation constraint on latent feature models.

## 1.2 Related work

This thesis aims to extend research that aspires to give machines the capability of human-like reasoning in open-domain question answering [31]. Early work in this area focused on using

relational data to learn deterministic logical rules based on symbolic representations, which were used for formal reasoning [38]. This approach gave rise to the inductive logic programming paradigm, however due to finite domain utility, scalability issues and poor performance, research in reasoning shifted to statistical methods focused on attribute representation, which were increasingly successful in prediction tasks, particularly in computer vision and natural language processing. SRL is the hybridisation of these two approaches, learning statistical correlations between concept dependencies by leveraging structural information in relational data.

SRL is comprised of three paradigms: latent feature modelling, graph modelling and IPLP. The rest of this thesis is concerned with the analysis of latent feature modelling approaches. Tensor factorisation has become one of the most popular approaches to latent feature modelling, where a decomposition of relational data represented as a tensor of relational scores is used to generate a set of constituent vectors, as demonstrated in Figure 1.2 and Figure 1.3.

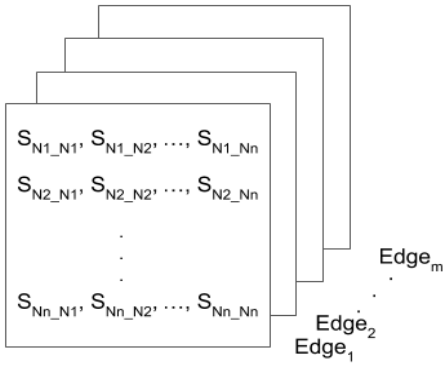



Figure 1.2: KG structured data represented as a relational score tensor of size $n \times n \times m$ with relational scores $S$ between nodes, indexed by $Edge$. $S_{N1\_N2}$ is the relational score between node $N1$ and node $N2$ for $Edge_1$, in the range $\in (0, 1]$, e.g. (Chadwick Boseman, starred in, Black Panther). If an edge is present between these two nodes in the data, the score is 1.0.

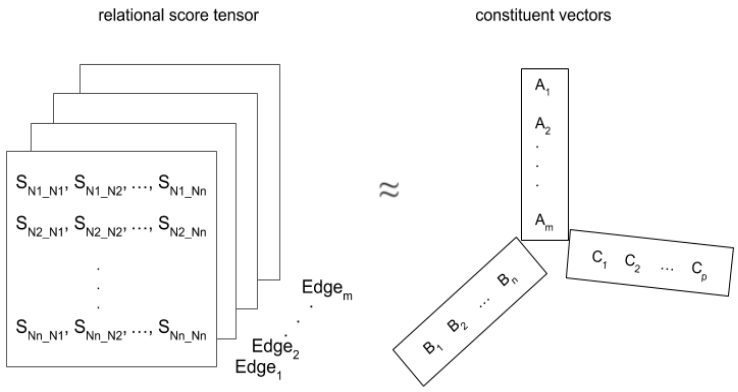



Figure 1.3: Relational score tensor decomposition into 3 constituent vectors $A$, $B$ and $C$, where $A \in \mathbb{R}^m$, $B \in \mathbb{R}^n$ and $C \in \mathbb{R}^p$. The inner product of these vectors generates the relational tensor.

RESCAL, by Nickel et al. [64], decomposes a relational score tensor $S_k$ into an entity matrix $E \in \mathbb{R}^{n \times r}$, a relational tensor $R_k \in \mathbb{R}^{r \times r \times m}$ and the transpose of the entity matrix $E^T \in \mathbb{R}^{r \times n}$, as illustrated in Figure 1.4.

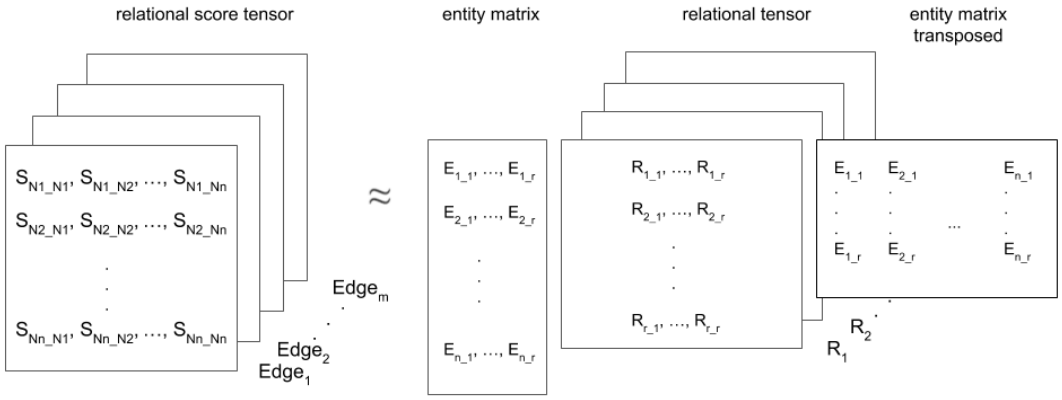



Figure 1.4: Relational score tensor decomposition into entity matrix $E \in \mathbb{R}^{n \times r}$, relational tensor $R_k \in \mathbb{R}^{r \times r \times m}$ and the transposed entity matrix $E^T \in \mathbb{R}^{r \times n}$. The product between $E$, $R_k$ and $E^T$ generates the entity-relational tensor.

The bilinear tensor product is the tensor product between entity matrix $E$ and relational tensor $R_k$, which generates an entity-relational tensor. The tensor product is then taken between it and the transposed entity matrix $E^T$,

$$S_k \approx E R_k E^T \ . \tag{1.1}$$

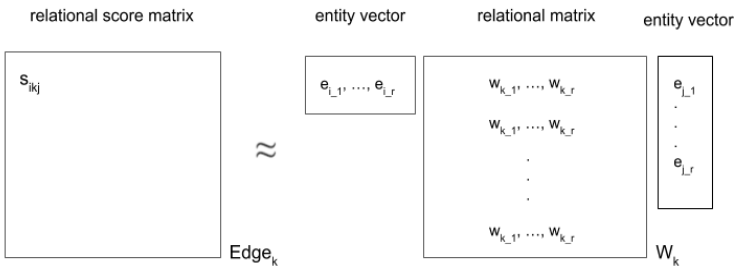



Figure 1.5: Computing the bilinear tensor product for a single entry in a relational score matrix, where $s_{i,k,j}$ is the relational score between entities $e_i$ and $e_j$ for relationship $W_k$.

Entity matrix $E$ is composed of entity vectors $e_i \in \mathbb{R}^{1\times r}$, and relational tensor $R_k$ is composed of full rank matrix slices $W_k \in \mathbb{R}^{r\times r}$ [65], where $k \in \{1, \ldots, m\}$ is the set of KG relations. The vector-matrix product between entity $e_i$ and relational matrix $W_k$ generates an entity-relational vector $h_{i,k} \in \mathbb{R}^{1\times r}$. The inner product between $h_{i,k}$ and entity vector $e_j \in \mathbb{R}^{r\times 1}$ generates a relational score $s_{i,k,j}$, as shown in Figure 1.5.

In the RDF formalism, $e_i$ is the subject, $W_k$ is the predicate, and $e_j$ is the object, and the set $\{e_i, W_k, e_j\}$ comprises an RDF triple used to encode a KG fact. $\{e_i, W_k, e_j\}$ have unknown latent representations (computed using a parameterised function), and the task is to learn the most optimal parameters that can be used for relation prediction. The RESCAL training algorithm minimises a squared error loss, and in doing so achieves competitive performance on standard link prediction benchmark datasets.

Translating the embeddings of entities (TransE) is another model for relation prediction [13]. The motivation for this approach is that a lot of KG facts are presented in hierarchies, therefore a translation of the subject by the predicate should produce an embedding close to the object. The model translation is thus a natural transformation of the entity and can be used to model hierarchies, as well as modelling embedding equivalence with reverse transformations.

Yang et al. [93] proposed DistMult, a bilinear diagonal model which first transforms the subject and object entities into low-dimensional vector representations, and then applies a bilinear tensor product operation using a predicate matrix that is diagonal. This approach effectively models a subset of the entity-relational interactions of RESCAL, and relies on the entity transformations to generate sufficient semantic information. The model's bilinear formulation is extremely parameter efficient but lacks expressiveness.

ComplEx, by Trouillon et al. [88], is the extension of DistMult into the complex domain. Entities are represented using complex vectors, and relations are represented using a diagonal

matrix with complex entries. Complex valued embeddings extend the modelling of entity-relational interactions from symmetric, such as when using the predicate "spouse of", to include antisymmetric interactions, such as when using the predicate "has part". In the former the subject and object are interchangeable, and in the latter they are not.

All these approaches rely on linear tensor factorisation for latent feature modelling, applying linear operators for entity-relational interaction modelling. These representations scale well with large datasets but are limited in their expressiveness, i.e. their capability to represent a variety of complex relationships between entities. Next we discuss nonlinear models.

Neural tensor networks (NTN), by Chen et al. [79], extend the bilinear tensor product in two ways. Firstly, they make use of a recursive network (RCN) [80] to construct a composition between two entities and output a compositional score. This compositional score is then added to the output of the bilinear tensor product, to produce a relational score. Secondly, the computed relational score is passed through a squashing nonlinearity, before being multiplied by an output parameter which generates a measure of confidence in a potential relationship between the two entities. The NTN model achieves a higher level of expressiveness, significantly increasing relation prediction performance. Chen et al. [79] also assessed the use of pre-trained word vectors on tensor factorisation, and find they outperform random initialisation.

Hohenecker and Lukasiewicz [38] extended NTNs by pre-computing an object representation as an aggregation of all facts (triples) in which the object plays a part. They introduce the new model as a relational NTN (RTN). Dong et al. [22] introduced ER-MLP, a two-layer MLP which takes a triple as an input set of vectors, and passes it to a fully-connected layer. The layer generates a hidden output vector which is passed through a nonlinearity, and finally through a second fully-connected layer. ER-MLP makes use of low-dimensional vectors for embedding representations.

HolE, by Nickel et al. [63], uses a circular correlation as an operator during relation prediction. The multiplication between subject features $e_i$ and object features $e_j$ is taken by sliding the object over the subject, as demonstrated in the following:

$$c_0 = e_{i0}e_{j0} + e_{i1}e_{j1} + e_{i2}e_{j2} \,, \tag{1.2a}$$

$$c_1 = e_{i0}e_{j2} + e_{i1}e_{j0} + e_{i2}e_{j1} \,, \tag{1.2b}$$

$$c_2 = e_{i0}e_{j1} + e_{i1}e_{j2} + e_{i2}e_{j0} \,, \tag{1.2c}$$

where $c_0$, $c_1$, $c_2$ are elements of a correlation vector. An inner product is then taken with a predicate vector to produce a relational score. HolE is particularly adept at modelling antisymmetric interactions.

ConvE, by Dettmers et al. [19], introduces the use of the convolutional operator in entity-relational modelling. The model reshapes the subject vector $e_i \in \mathbb{R}^r$ and the predicate vector $w_k \in \mathbb{R}^r$, creating two matrices $\overline{e_i} \in \mathbb{R}^{r_w \times r_h}$ and $\overline{w_k} \in \mathbb{R}^{r_w \times r_h}$, where $r = r_w r_h$. $\overline{e_i}$ and $\overline{w_k}$ are then lengthwise concatenated to create an entity-relational matrix $m \in \mathbb{R}^{r_w \times 2r_h}$, as demonstrated in the following:

$$\left(\begin{bmatrix} e_i & e_i & e_i \\ e_i & e_i & e_i \end{bmatrix}\right); \left(\begin{bmatrix} w_k & w_k & w_k \\ w_k & w_k & w_k \end{bmatrix}\right) = \begin{bmatrix} e_i & e_i & e_i \\ e_i & e_i & e_i \\ w_k & w_k & w_k \\ w_k & w_k & w_k \end{bmatrix}. \tag{1.3}$$

A convolution is then performed on $m$ using a set of convolutional filters. The same set of filters is used for convolutions across all subject-predicate combinations, allowing for parameter sharing across entity-relation feature interactions. The convolution generates a feature map that is reshaped into a vector and passed to a fully-connected layer, which generates a hidden vector that is passed through a nonlinearity. Finally the inner product of the vector is taken with all object vectors, to produce a relational score for each object. The convolutional operator increases expressiveness by modelling entity-relation feature interactions around the entire concatenation line, enabling high levels of relation prediction performance.

The above approaches comprise nonlinear tensor factorisation for latent feature modelling, where ER-MLP and ConvE introduce neural tensor factorisation. Two other paradigms, graph modelling and IPLP, apply alternative approaches to link prediction. We refer the reader to [62] for a recent review.

## 1.3 Contributions and outline

In this thesis we explore neural tensor factorisation models for latent feature modelling in link prediction. We assess how nonlinear extensions of the bilinear tensor product, and the use of convolutional operators for entity-relational modelling, result in higher levels of expressiveness. In particular, we attempt to further improve model performance by optimising the training algorithm, with particular attention to regularisation. To this end, we update the NTN training algorithm with modern regularisation, optimisation and hyperparameter tuning techniques. We then compensate for covariate shift introduced by hypernetworks, and use pre-trained word vectors for embedding in initialisation.

In **Chapter 2** we discuss training algorithm approaches based on the supervised learning paradigm. We then discuss deep learning models, beginning with convolutional and recurrent

networks. We provide mathematical definitions for the models, and discuss the primary problems they aim to address, as well as examples of applications. We also give a brief overview of recursive and hypernetworks. We present regularisation techniques used to address overfitting, namely early stopping, dropout [82] and batch normalisation. Finally we discuss language models and their utility in semantic representations.

In **Chapter 3** we begin by exploring the NTN model introduced by Chen et al. [79] and reimplemented in TensorFlow by Doss et al. [23]. We assess the expressive strength of this model due to the addition of entity-compositional modelling using an RCN, and initialisation of entity embeddings using pre-trained word vectors from a language model. We then explore reimplementing the model ourselves, and optimise the training algorithm by applying early stopping, Adam optimisation and hyperparameter random search.

We continue with exploring the HypER model introduced by Balažević et al. [5]. We assess the expressive strength of this model due to making use of the hypernetwork architecture to generate relation-specific convolutional filters. The convolution between these filters and entities are used to construct representations which are ultimately used for relation prediction. We explore compensating for covariate shift caused by hypernetworks by applying batch normalisation, and propose HypER+. We then initialise HypER+ entity and relation embeddings using pre-trained word vectors from the GloVe [67] language model.

In **Chapter 4** we conduct a number of experiments to test the hypotheses concerning the training algorithms of the respective models, derived in the previous chapter. We begin by discussing the WordNet [55] and Freebase [11] link prediction benchmark datasets. We use them to establish baseline cost and accuracy performance metrics for the reimplemented NTN TensorFlow model. We train the reimplemented model using an updated training algorithm, and compare performance against the baseline.

We then discuss the WN18 [12] and FB15k [13] link prediction benchmark datasets, as well as benchmark metrics used for those datasets. We train the HypER+ model with the updated HypER training algorithm, and compare performance against the baseline. Finally we discuss the WN18RR [19] and FB15k-237 [87] link prediction benchmark datasets. We extend HypER+ to take advantage of GloVe pre-trained vectors, and compare the performance of HypER and HypER+ without pre-trained word vectors.

In **Chapter 5** we summarise the thesis and identify opportunities for neural tensor factorisation. We comment on the suitability of latent feature modelling for link prediction, in contrast with alternative paradigms, and assess the current utility of KGQA. We conclude with proposals for future research.

# Chapter 2

# Deep learning

There are many tasks which are hard to write algorithms for. For example, writing an algorithm that identifies a type of fruit in a picture is a challenging undertaking. One could try to do this by asking a number of yes or no questions to help whittle down the possibilities. Sensible questions might be: "is the fruit round?", "is the fruit green?", and "is the fruit's surface rough?". The problem with this approach is that it is very brittle. More questions could be introduced to disambiguate more cases, but the large variations in the appearance of fruit can cause the algorithm to become confused. Writing rules in this way is also not scalable. There are many types of fruits, and writing questions to determine each one quickly becomes intractable.

Machine learning (ML) can be used to solve such tasks, as well as others like weather prediction, stock price forecasting and risk modelling. ML is the process of using data to build prediction models, where these models typically output discrete (classification) or continuous (regression) values. There are various learning paradigms, including supervised learning, unsupervised learning and reinforcement learning, used to train models from data in order to perform a specific task. The models can be divided into three classes, namely shallow, deep and probabilistic [33].

This chapter presents a brief introduction to the supervised learning paradigm, a description of deep convolutional [49] and recurrent [91] network models, as well as an overview of recursive [70] and hypernetworks [30]. It continues with a discussion on regularisation techniques for deep models, including early stopping [71], dropout [82] and batch normalisation [39], and concludes with a discussion of language models.

## 2.1 Supervised learning

An ML prediction model takes as input a set of features. These features are attributes of a data sample. For example in the fruit prediction task mentioned above, input features might be attributes such as shape, colour, texture and size. These inputs are provided to a model that maps them to outputs, where outputs might be types of fruit represented as values corresponding to orange, apple, strawberry, pineapple, etc.; see Figure 2.1. The model is then trained to identify the correct fruit (output) based on the features (input) it receives from a collection of input-output examples. This type of training is called supervised learning [8].

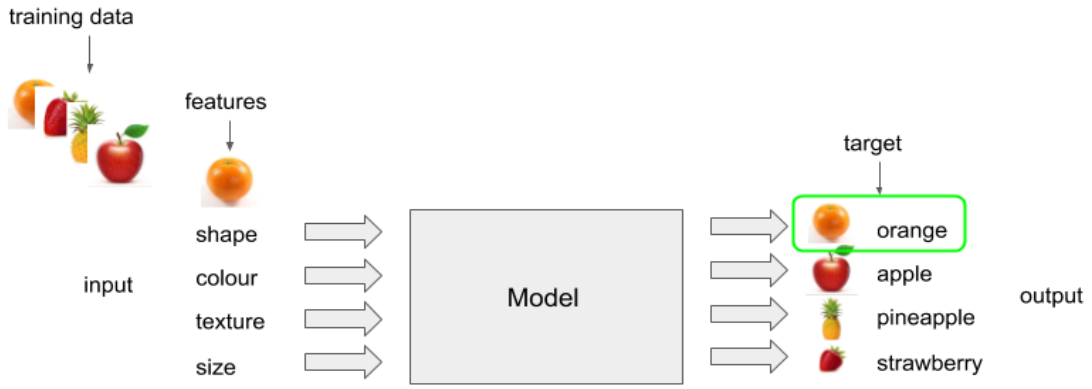



Figure 2.1: A machine learning model trained using supervised learning. The task of this model is to map input features to output categories, in this case predicting the type of fruit given an image.

Features can be continuous valued or discrete, where discrete values are referred to as categorical variables. Outputs can also be continuous or discrete, where discrete outputs are known as classes. Example input-output pairs comprise a training dataset which is represented as $D = \{(x_i, y_i)\}_{i=1}^{N}$, where $x_i$ is the input feature and $y_i$ is its expected output label. The model has to learn a general mapping from input features to output labels.

In the fruit example, the model will attempt to build decision boundaries between fruit classes in the feature space. We can visualise this process in a 2-dimensional ($2D$) setting using two fruit features, say shape and colour. Figure 2.2 shows an example of synthetically generated samples in this $2D$ feature space and a potential decision boundary.

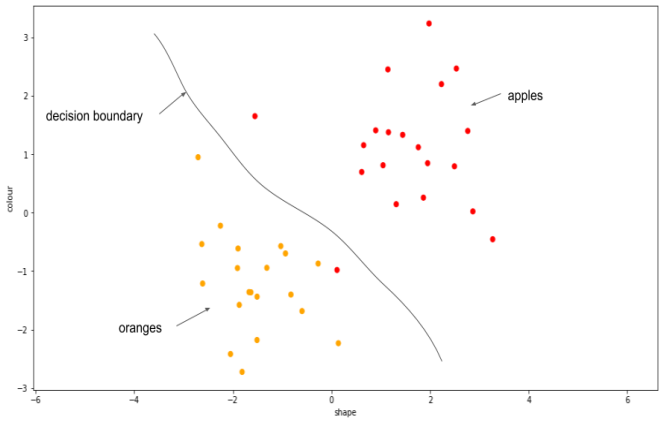



Figure 2.2: A decision boundary between two classes, apples and oranges. The model constructs this curve by learning discriminative feature characteristics corresponding to each class.

Data is assumed to be generated by the same underlying process, and would be either independent and identically distributed (IID), or not. In the context of IID data, samples are independent and therefore order-invariant; while in the context of dependent data samples, order might matter. The class of data can determine the best choice of model for prediction.

Available labelled data is typically subdivided into three sets: training, validation and test data. The training set provides in-sample data – input-output pairs which are exposed to the model during training and used to update its parameters. The validation set approximates out-of-sample data used to adjust model architecture, as well as hyperparameters (model parameters which configure its architecture and operation). The test set is used to assess the model's prediction performance after training and hyperparameter tuning are complete, and establish an unbiased measure of the model's utility.

An objective function is used to measure the prediction performance of a model during training, and can compute an error or reward signal. When the objective computes an error it is called a loss function, and measures the difference between the model's predicted output and the target output (the labels in the training set). As the model parameters are updated during training, the loss value changes and produces a multidimensional loss surface. The training algorithm tries to minimise this loss using a method of optimisation, such as Bayesian, evolutionary or first-order. When the objective computes a reward it is called a reward function, and is often used when a sequence of steps is required to complete a task [59]. The training algorithm tries to maximise the reward, which is typically applied in reinforcement learning settings. The reader is referred to [84] for a complete introduction.

A popular method of optimisation is gradient descent, which tries to descend to the lowest region of the loss surface: the global minimum. Model parameter updates are computed based on partial derivatives of the loss, with respect to the individual parameters. The size of the update is dependent on the magnitude of the gradient of the loss, i.e. the steeper the gradient, the larger the update. A training algorithm uses gradient descent to guide the loss toward its global minimum, but in practice a local minimum is usually the best that can be achieved. In doing so the training algorithm attempts to optimise the model, such that it can estimate the underlying distribution of the data and perform inference on samples not seen during training.

During training, mini-batch optimisation is often used along with gradient descent to update model parameters. In mini-batch optimisation the training set is divided into batches, where model parameters are updated after a mini-batch has been processed. This is in contrast to stochastic gradient descent where parameters are updated per sample, and batch gradient descent where parameters are updated after a run through the entire training set. Mini-batch optimisation is both robust in terms of convergence (not as susceptible to local optima), and compute efficient.

The supervised learning training algorithm is expressed as in Algorithm 1.

**Algorithm 1:** Supervised learning with mini-batch gradient descent

**Input** Training set $D = \{(x_i, y_i)\}_{i=1}^{N}$ of input features and output labels
Initialise parameters $w \in \mathbb{R}^d$, for model $M(w)$ // e.g. random normal initialisation
**for** *Epoch* $k = 1 \ldots K$ **do**
  **for** *Mini-batch* $b = 1 \ldots B$ **do**
    $S \leftarrow sample(D, b)$ // sample mini-batch $b$
    **for** $(x, y) \in S$ **do**
      $y' \leftarrow predict(x, w)$ // predict label for sample $x$ using parameters $w$
      $l \leftarrow loss(y - y')$ // compute loss e.g. cross-entropy or squared difference
      $e \leftarrow e + l$ // add loss to zero-initialised mini-batch loss $e$
    **end**
    $w \leftarrow w - \lambda \cdot J_e(w)$ // gradient descent update of model where $\lambda$ is the learning rate, and $J_e(w)$ are the partial derivatives of the loss with respect to the parameters $w$
  **end**
**end**

## 2.2 Deep learning models

Deep learning models have become popular for classification and regression tasks, and machine learning tasks in general. The reason for this is that deep models solve a major problem with shallow models, which is to select the best features with which to represent raw input samples [27]. The problem of feature selection has resulted in a methodology used during shallow model development, called feature engineering. It includes techniques such as bucketing, crossing, hashing and embedding. These techniques are designed to build feature representations that will result in high classification and regression accuracy. Deep models attempt to discover optimal representations automatically during training, hence their superior performance on a number of machine learning tasks.

Multilayer perceptrons (MLPs) are deep models implemented as feed-forward neural networks, consisting of $N$ layers applied to an input vector $x$. These models compute unnormalised scores, known as logits, for each of the possible outputs. They construct a composition of nonlinear functions which describe an analytical definition of the generative process of the data used to train them. The utility of composition is what gives them the capability needed to model complex datasets. MLPs achieve nonlinear expressiveness with the use of nonlinear activation functions, such as the ReLU, sigmoid and tanh functions.

MLPs generate an output by executing a number of steps in what is called the forward pass, where each step is represented as a layer. A basic MLP may consist of three layers (as shown in Figure 2.2): an input, hidden and output layer. A hidden layer is comprised of neurons (nodes) that generate an activation (output) per node. Each node computes a linear combination of inputs from the previous layer, and then transforms the result using a nonlinear activation function. The set of node outputs is a hidden vector representation, which is passed to the following layer as input. The output layer generates a logit, using a linear combination of inputs from the hidden layer. The more hidden layers the model has, the deeper it is said to be.

The forward pass can be expressed as follows:

$$f_0 = x\,, \tag{2.1a}$$

$$f_i = \sigma_i(W_i f_{i-1} + b_i) \quad i \in [1, N]\,, \tag{2.1b}$$

where $f_0$ is the input layer, containing the features of input vector $x \in \mathbb{R}^{m_0}$. $f_i$ is a hidden layer output vector of size $m_i$, $f_{i-1}$ is the input vector from the previous layer of size $m_{i-1}$. $W_i \in \mathbb{R}^{m_i \times m_{i-1}}$ is a matrix of layer parameters, $b_i \in \mathbb{R}^{m_i}$ is a vector of bias parameters and $\sigma_i$ is the activation function at layer $i$.

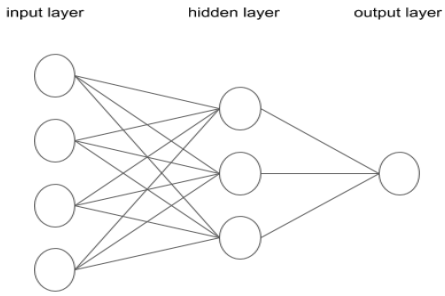



Figure 2.3: A basic MLP consisting of an input, hidden and output layer. The model can comprise an arbitrary number of hidden layers, containing an arbitrary number of nodes.

Loss functions are used to train deep models under a supervised learning paradigm. The loss function is minimised with gradient descent, which in the case of MLPs makes use of a process called backpropagation. This process involves the computation of first-order partial derivatives, and the adjustment of the parameters in a direction and magnitude relating to those derivatives. In a network with $N$ layers, the gradient for the respective parameter in layer $i$ is given by:

$$\frac{\partial E}{\partial W_i} = \frac{\partial E}{\partial z_N}\frac{\partial z_N}{\partial z_{N-1}}\cdots\frac{\partial z_{i+2}}{\partial z_{i+1}}\frac{\partial z_{i+1}}{\partial W_i}\,, \tag{2.2a}$$

$$\frac{\partial E}{\partial b_i} = \frac{\partial E}{\partial z_N}\frac{\partial z_N}{\partial z_{N-1}}\cdots\frac{\partial z_{i+2}}{\partial z_{i+1}}\frac{\partial z_{i+1}}{\partial b_i}\,, \tag{2.2b}$$

where $E$ is the loss function, $W_i$ is parameter matrix for layer $i$, $z$ is the layer output, and $b_i$ is the bias vector at layer $i$. The matrix $\frac{\partial z_{i+1}}{\partial W_i}$ is called a Jacobian matrix. It summarises all the ways in which changing the matrix $W_i$ by a small amount will influence the output $z_{i+1}$. The matrix is expressed as follows:

$$J = \begin{bmatrix} \frac{\partial z^1}{\partial W^1} & \frac{\partial z^2}{\partial W^1} & \cdots & \frac{\partial z^m}{\partial W^1} \\ \frac{\partial z^1}{\partial W^2} & \frac{\partial z^2}{\partial W^2} & \cdots & \frac{\partial z^m}{\partial W^2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial z^1}{\partial W^n} & \frac{\partial z^2}{\partial W^n} & \cdots & \frac{\partial z^m}{\partial W^n} \end{bmatrix}. \tag{2.3}$$

Different schemes are used to initialise layer parameters $W_i$ and $b_i$. The appropriateness of these schemes depends on the choice of activation function, $\sigma_i$, used. Sigmoid and tanh functions can lead to saturation if parameters are initialised with values that are too large, or to vanishment if initialised with values which are too small, which in turn can lead to exploding and vanishing parameter gradients. In order to overcome this problem Xavier initialisation [26] is used, which scales the initial parameter values by the number of layer inputs and outputs. This limits the variance of layer outputs and helps stabilise gradient updates. When the ReLU activation is used He initialisation [34], a modification of Xavier initialisation, is commonly applied.

Together the forward pass algorithm and backpropagation implicitly allow the automatic discovery of the most meaningful representations used to generate model output. We refer the reader to [27] for a review on loss functions, as well as a more detailed discussion on the forward pass and backpropagation.

### 2.2.1 Convolutional networks

Deep MLPs may suffer from an explosion in the number of model parameters [48]. In an MLP layer, all features of an input vector are connected to all other features when computing the layer output. This interconnectedness causes the number of model parameters to grow rapidly in the number of neurons per layer. For example, a feed-forward network with two hidden layers consisting of 512 and 256 neurons respectively, an output size of 10, and an input vector of shape $200 \times 1$, ends up with a total of $236,810$ parameters. This can cause a problem with high-dimensional input vectors such as images, where inputs may consist of a $3D$ input volume. MLPs may also need to be composed of a number of layers in order to produce satisfactory prediction performance when presented with high-dimensional inputs. In such a scenario, parameters can number in the millions, resulting in a model that can be impractically slow to train.

MLPs are also not able to take direct advantage of structure in data samples. When images are represented as a matrix, common shapes (features) may appear in a regular composition. For example, different cars may appear regularly as a compositions of rectangles and circles. The inner product operator applied between the input (which is typically flattened into a vector) and layer parameters of an MLP, disregards the spatial information of the data, making the task of learning class-discriminative features harder. As a result, in order to possess sufficient learning capacity that adequately models variance in structural representations, MLPs need to be impractically large.

Convolutional networks (CNs) have been developed to overcome both the above-mentioned problems experienced by MLPs. CNs make use of the convolutional operator instead of the inner product operator, with a small filter consisting of trainable parameters, as demonstrated in Figure 2.4. This process generates a latent feature map which is used in subsequent layers in the network. Latent feature maps may capture structural patterns in data, building simple intermediate representations. CNs compose these representations into more sophisticated representations in subsequent layers.

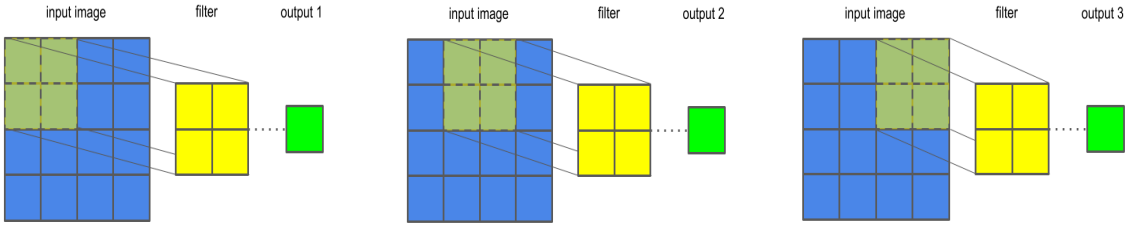



Figure 2.4: A convolutional operation taken between an input image and a filter. The filter slides across the image, and at each step a summary of that location is computed as a single value.

The convolutional operator allows CNs to capture translation invariance in inputs, due to filter parameter sharing [78]. Translation invariance is achieved when a CN layer generates an intermediate representation that is consistent under translational transformations. For example, if a different image of car is used as input, and the car is located in a different position, that would lead to a consistent feature map representation being generated. This property allows CNs to model the distribution of features using far fewer parameters and smaller datasets.

The trade-off one makes when using CNs is reduced flexibility in modelling complex decision boundaries. To mitigate this trade-off, CNs are often implemented with a final set of fully-connected layers, which enable the expressiveness required for accurate prediction, as illustrated in Figure 2.5.

In the following paragraphs we discuss the CN forward pass.

**Convolutional layers.** Convolutional layers rely on filters that are small spatially (along width and height), and extend through the full depth of an input volume. During the forward pass, each filter is convolved across the width and height of the input volume, where the elementwise dot product is computed between the entries of the filter and the input at a particular position. A $2D$ feature map is generated, that gives the responses of that filter at every spatial position of the input. Every convolutional layer has a set of filters, and each of them produces separate

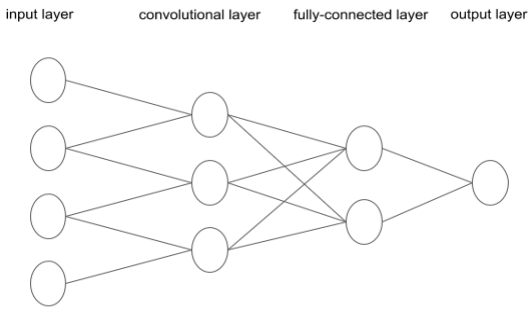



Figure 2.5: A basic CN consisting of an input, convolutional layer, fully-connected layer and output layer.

2*D* feature maps, which are then stacked along the depth dimension to produce an output volume [28].

The size of the output volume is controlled by the hyperparameters of the convolutional layer, namely the filter size, number of filters (channels), stride, padding, and dilation. The filter size defines the width and height of the filters, which are often square. The number of channels controls the output volume depth, as illustrated in Figure 2.6. The stride defines the number of entries by which the filter will move when convolving it along the input volume, as demonstrated in Figure 2.7 and Figure 2.8. Padding refers to the number of 0 entries added to the input volume along the width and height dimensions. It can be used to ensure that the output volume has the same width and height as the input volume. Dilation introduces spaces between the filter parameters, and is used to increase its receptive field, as shown in Figure 2.9.

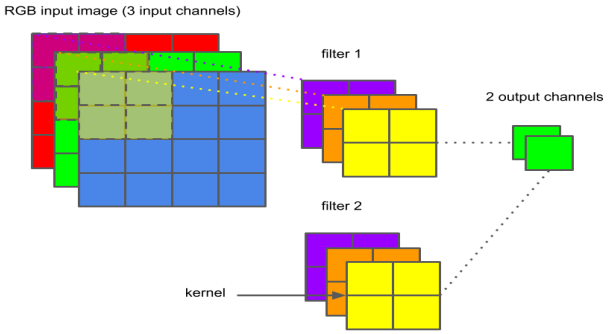



Figure 2.6: A 3*D* input image with three input channels. A kernel size of 2 is used, where the filter depth is the same as the number of input channels. Using two filters produces two output channels.

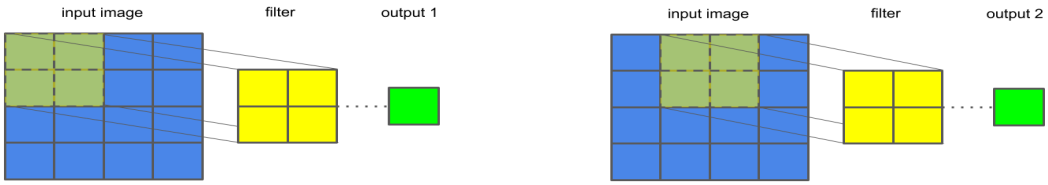



Figure 2.7: Consecutive steps in a convolutional operation with a stride of 1.


Figure 2.8: Consecutive steps in a convolutional operation with a stride of 2.

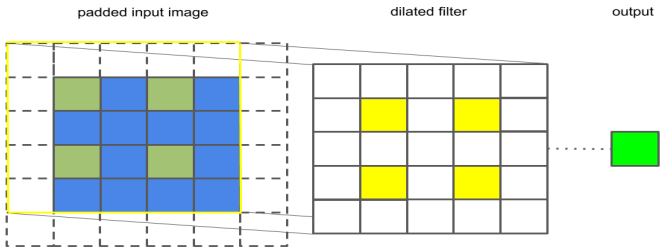



Figure 2.9: An input image with a padding of 1, convolved with a filter with a dilation of 2.

Convolution of an input volume $X$ with a single filter $W$ is given by:

$$O_{ij}^{d} = \sum_{a=0}^{f_w-1} \sum_{b=0}^{f_h-1} \sum_{c=0}^{d_i-1} W_{a,b,c,d_o} X_{i+a,j+b,c}^{pad} + b_{d_o} \,, \tag{2.4}$$

where $O_{ij}^{d}$ is the value of the output volume at position $(i,\ j)$ of feature map $d$. $W_{a,b,c,d} \in \mathbb{R}^{f_w f_h d_i d_o}$ is a set of filters with size $f_w \times f_h$, number of input channels $d_i$, and number of output channels $d_o$. $X^{pad}$ is the padded input volume, and $a$, $b$, $c$ are indices into the volume dimensions. $b \in \mathbb{R}^{d}$ is a bias vector.

**Pooling layers.** A pooling layer is used to reduce the spatial size of the representation, in order to reduce the number of parameters in the network. Pooling layers provide latent features

deeper in the network with a larger receptive field. In particular, pooling gives deeper latent features much larger receptive fields so that they can effectively combine smaller features together. Pooling layers apply some $2D$ aggregation operation (usually a maximum, but others like average may also be used) to local regions of the input volume. A pooling layer has no trainable parameters itself.

**Fully-connected layers.** In order to compute the output of a CN, one or more fully-connected layers are required to flatten the output volume of the last convolutional layer. These layers perform the same functions as an MLP to compute an output vector.

**Convolutional backpropagation.** Computing parameter updates for convolutional filters follows the same first-order differential procedure used in backpropagation for MLPs. Assuming a loss function $E$, and having computed the partial derivatives of this loss up to the output of the convolutional layer, $\frac{\partial E}{\partial O}$, in order to update the parameters the convolutional layer, we require the derivative of $E$ with respect to the weights and biases of the convolutional layer. We also require the derivative with respect to the input to the current layer $\frac{\partial E}{\partial X}$, and use this when computing the error to pass back to the previous layer.

The required derivatives are:

$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial O}\frac{\partial O}{\partial b}\,, \tag{2.5a}$$

$$\frac{\partial E}{\partial W_{a,b,c,d}} = \sum_{i=0}^{O_w-1}\sum_{j=0}^{O_h-1}\frac{\partial E}{\partial O^d_{ij}}\frac{\partial O^d_{ij}}{\partial W_{a,b,c,d}}\,, \tag{2.5b}$$

$$\frac{\partial E}{\partial X^{pad}_{m,n,c}} = \sum_{i=0}^{O_w-1}\sum_{j=0}^{O_h-1}\sum_{d=0}^{D-1} W_{m-i,n-j,c,d}\frac{\partial E}{\partial O^d_{ij}}\,. \tag{2.5c}$$

### 2.2.2 Recurrent networks

An IID dataset is order invariant, because the probability of seeing one data point in no way influences the event of seeing another. Practically, this means shuffling the data when training a model has no influence on prediction performance. However some data may contain dependencies, and when training a model the order of the data points matters. Such data is presented as a sequence which is broken into steps, where each step is influenced by the steps before it.

An example of dependent data is the daily temperature of Cape Town over a year. This sequence can be broken into steps, where each step is the daily high in degrees Celsius. The temperature on day $t$ is influenced by the temperature on the preceding days, $t-1$, $t-2$, $\ldots$ $t-n$.

Text expressed in natural language is another example of sequential data. This data is arranged in sentences which are comprised of sequences of words. Each word represents a step in the sequence, and if we try to predict the word at step $t$, it may be useful to have context of the words at $t-1$ and $t-2$. For example, in the sentence "I think, therefore I ...", we would want to predict "am" as the next word.

In the task of one step ahead prediction, we build models that attempt to estimate a probability distribution over a sequence. This distribution estimates the likelihood of seeing a data point given a preceding sequence.

Recurrent networks (RNs) model the conditional distribution of sequential input data by computing a summary of previous data points. This summary is defined as an internal state of the model, and contains outputs from previous sequence steps combined with a recurrent parameter matrix using a matrix-vector product operator. RNs thus extend feed-forward networks like MLPs and CNs by directly incorporating sequential dependencies between data in the model, as illustrated in Figure 2.10.

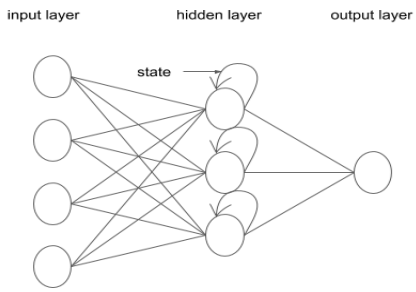



Figure 2.10: A basic RN consisting of an input, hidden and output layer. An internal state is added to hidden layer nodes which tracks previous output. This state is summed with the node output.

Additional context applied to RN prediction is made at the expense of a higher number of model parameters. The increase in parameters is linear in the number of sequence steps considered in the context summary. Practically this means a truncated sequence is used to provide context during training, to keep computation tractable.

In the following paragraph we discuss the RN forward pass.

**Recurrent layers.** RNs build context by generating a representation of a window of data samples. This representation is defined as a state vector, which is added to a hidden vector and bias of a layer within the network. The layer output is then passed through a nonlinear

activation before being passed to the next layer. RN layers are comprised of cells, which contain an internal state, hidden output and a bias [29]. An RN cell can thus be defined as follows:

$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b) \,, \tag{2.6}$$

where $h_t$ is the cell output, and added to the state vector as context for the following sequence step. $\sigma$ is a nonlinear activation function, $W_{hh}$ contains the recurrent parameters, $W_{xh}$ contains the non-recurrent parameters, $x_t$ is the input and $b$ is the bias. $h_t$ is then used as input to a fully-connected layer for prediction at every step, which is given by:

$$y_t = \sigma(W_{hy}h_t + b) \,. \tag{2.7}$$

**Backpropagation through time.** RNs are trained using a method called backpropagation through time (BPTT), a gradient-based process that updates recurrent parameters by unrolling cell outputs over a window of sequence steps. Parameter gradients are computed at each step with respect to the error generated at that prediction step. The error signal generated is backpropagated through the RN layer to adjust the cell state, hidden and bias parameters, by computing the appropriate first-order partial derivatives. In practice, a fixed sequence length is decided upfront for updating cell parameters, leading to a process called truncated BPTT, which has compute resource advantages. Standard backpropagation is used for non-recurrent parameter updates.

The gradient of the loss function $E_t$ at time $t$ with respect to $W_{hh}$ is a function of the current hidden state and model predictions $\hat{y}_t$ at time $t$:

$$\frac{\partial E_t}{\partial W_{hh}} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}} \,. \tag{2.8}$$

One problem experienced by RNs is the multiplicative effect of state vector derivates throughout sequence-time. This problem can be seen in the following expression:

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \,. \tag{2.9}$$

This multiplicative operation causes small gradients to become progressively smaller as the sequence window becomes larger, or large gradients to become much larger. It is called the vanishing or exploding gradient problem and has lead to the development of RN variant models such as long short-term memory [36] and gated recurrent units [17].

### 2.2.3 Recursive networks and hypernetworks

A number of non-mainstream architectures will also be used in this study. Recursive networks (RCNs) construct a composition tree using sequential input to produce composition scores. Figure 2.11 illustrates. This approach differs from RNs, which construct a composition chain using sequential input. Composition trees may be able to model more complex structure in input sequences, however require pre-training to first prune them, and may not be as computationally efficient during inference.

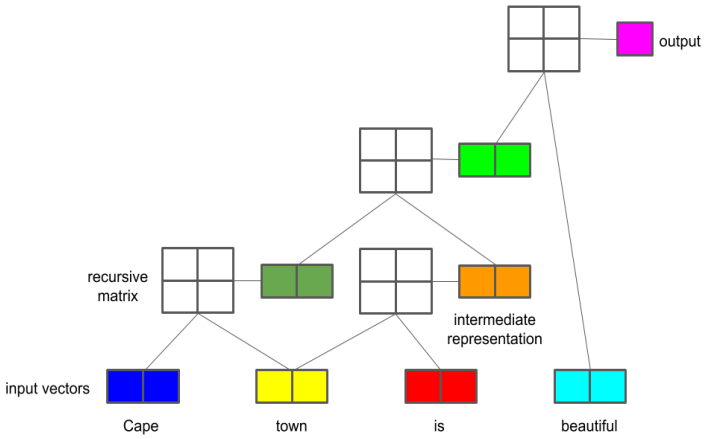



Figure 2.11: A recursive network which constructs a composition tree between inputs. It begins as a full tree and is pruned during training to reach an optimal structure.

Hypernetworks are meta-networks that generate parameters for a main network. An embedding is supplied as input to a hypernetwork, which encodes information about the parameters of a main network's layers. For example, an embedding may be used as input to an MLP hypernetwork which generates the layer parameters for a CN main network, consisting of a convolutional layer and a fully-connected output layer. Figure 2.12 illustrates. Hypernetworks enable the design of more parameter efficient main networks, by compressing main network parameter information into the embedding input of the hypernetwork. They are also useful for a form of relaxed parameter sharing across layers, without suffering from vanishing or exploding gradients experienced by RNs.

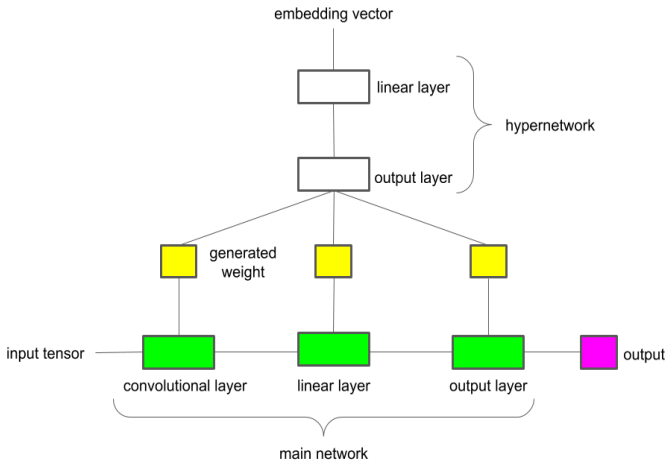



Figure 2.12: A hypernetwork generating convolutional filters for a main network. The hypernetwork takes a static embedding vector as input.

## 2.3 Regularisation

Model over-parameterisation can lead to overfitting on in-sample data. This is because the degrees of freedom available to estimate a function allow very complex nonlinear functions to be discovered, where these nonlinear functions are not representative of the generative process of the data. In fact, the model may end up fitting to the noise in the dataset. Overfitting results in poor generalisation to new data. This problem forms part of the broader bias/variance trade-off, specifically high model variance due to over parameterisation. Deep learning models are particularly sensitive to overfitting given the high number of parameters present within them. Regularisation techniques are used to improve the generalisation capability of deep models.

**L1 and L2 regularisation.** Lasso (L1) [85] and ridge (L2) [37] regularisation are two common approaches to prevent overfitting, by adding a term to the loss that penalises the model if it becomes too complex. They are defined as follows:

$$loss_{L1} = loss + \lambda \sum_i |w_i| \ , \tag{2.10a}$$

$$loss_{L2} = loss + \lambda \sum_i w_i^2 \ . \tag{2.10b}$$

L1 regularisation has the effect of forcing some parameters to shrink to 0, effectively removing them from the model. L2 regularisation has the effect of preventing any of the parameters from becoming too large and overpowering the others.

**Early stopping.** As deep models are trained using a supervised learning paradigm, the model training accuracy gradually increases. As training continues, the model validation accuracy gradually increases, and then begins decreasing. At this point the model has begun overfitting the training data. If the model continues training, the validation accuracy may continue decreasing, and should the model then be tested, the resulting test accuracy may be significantly lower than the training accuracy. Early stopping is a technique used to try to prevent overfitting. Once the training and validation accuracies begin to diverge, training continues for a few more epochs, and if the divergence persists, training is terminated.

**Dropout.** During training, nodes in a deep model can generate redundant latent features that overfit to the training data. Dropout is a technique that is used during training to zero out a randomly selected proportion of neurons in the layer of a deep model. This has the effect of removing partial dependence between nodes deeper within the network, preventing complex co-adaptation of hidden layers, and resulting in more independent latent features [35]. These features produce a more robust representation of the generative process of the data, allowing greater model generalisation to out-of-sample data.

**Batch normalisation.** Deep models are composed of parameterised layers. When the parameters of these layers are adjusted during training using mini-batch optimisation, each data point within the batch contributes to the model output individually, however the error is aggregated across the entire batch. This may result in a shift in the distribution of node output for each layer; a problem known as internal covariate shift. Covariate shift makes it difficult for the model to estimate the underlying generative distribution of the data, and batch normalisation has been proposed [39] as a technique to compensate for this problem.

Batch normalisation solves two other problems experienced during deep model training, namely nonlinearity saturation and weight initialisation sensitivity. It does this by computing layer-specific moving average estimators of the batch mean and standard deviation. Each layer's node output is then modified by these statistics before being passed onto the following layer. Batch statistics at training time are then aggregated into global population statistics during test time. The application of batch normalisation often shortens the required model training time, and can result in better prediction performance.

## 2.4 Language models

Language models are a set of vector representations for the words in a language. They capture statistical correlations between the words, and in so doing build an understanding of semantic

meaning. They encode this meaning in a word vector space, where for example the vectors for the words "film" and "character" may be collocated, given they share semantic meaning. Language models are often trained using RNs on large sequential text datasets. The following presents a brief overfiew on their development.

Word2Vec, introduced by Mikolov et al. [54], is an algorithm for training distributed representations of words. The skip-gram model [53] is one of the models used by Word2Vec to generate word vectors from very large text corpora (more than a billion words). The model essentially takes a centre word, and then tries to predict the probability of seeing a context word to the right or left of the centre word. This context window can also be longer than one word. The skip-gram model uses the hierarchical softmax loss function [58] for classification (a computationally efficient approximation of the full softmax) and employs the subsampling of frequent stop words such as "in", "the", and "a". The continuous bag of words (CBOW) model is an alternative approach to the skip-gram model. Here the model tries to predict a target word, given a sequence of context words. CBOW has the advantage of being faster to train and producing better representations for more frequent words.

GloVe, by Pennington et al. [67], is another unsupervised learning algorithm for generating word vectors. This model generates word vectors using the co-occurrence statistics of words in a text corpus. The corpus is viewed as a global statistics space, and the model begins with a pass over the entire corpus to generate a co-occurrence matrix of words against all other words in the corpus. GloVe thus captures both global and local statistics of a corpus, compared to Word2Vec which only captures local corpus statistics. GloVe then attempts to model the co-occurrence statistics between words using a neural network that takes two words as input and scores the co-occurrence statistic as output. The training procedure minimises a logarithmic squared error loss that incorporates both global and local corpus statistics. The distributional representation of word embeddings that GloVe employs, allows it to outperform Word2Vec on word analogy, word similarity, and named entity recognition tasks. It is therefore the chosen word embedding method used in this thesis.

A number of alternative approaches to building language models have become popular, including FastText [10], BERT [90] and ELMo [68]. FastText takes advantage of modelling every word as a combination of character n-grams, providing representations for rare and unseen words. BERT and ELMo take advantage of contextual representations of words, generating on-the-fly representations for words given a sequence of words instead of returning a precomputed word representation by index lookup. These representations are superior at capturing semantic information than static representations. The need for sequences of words to generate entity representations makes BERT and ELMo unsuitable for use in our context.

## 2.5 Summary

Supervised learning is a paradigm used to train ML models where target output is provided with corresponding input. This paradigm allows models to build an input-output map that approximates the underlying data distribution. The effectiveness of this mapping is heavily dependent on the input representation of samples, known as features.

We can take advantage of automatic feature representation using deep learning models, which unlike shallow models, compute these directly from the data. This has been shown to be more effective than manually constructing feature representations using techniques such as bucketing, crossing, hashing and embedding. CNs solve the over parameterisation problem in MLPs, and simultaneously take advantage of spatial structure present in data samples. RNs provide context of previous samples using state vectors when modelling sequential data. RCNs construct composition trees to model complex structure in sequential data, and hypernetworks enable efficient parameter sharing between network layers.

Deep models can overfit to training data and as a result generalise poorly to test data. A number of approaches have been used to address this problem, including lasso and ridge regression, early stopping and dropout. Batch normalisation has proved particularly effective at regularisation as it compensates for the covariate shift experienced by deep models during training. In addition, batch normalisation also speeds up training time, and makes deep models less sensitive to weight initialisation.

Language models build an understanding of the meaning of words in a language. They are constructed using the distributed representation of words or distributional representation of words hypotheses, and trained using RNs on large sequential text datasets. Language models generate word vectors, embedded in a word vector space, which can be used in downstream ML tasks.

# Chapter 3

# Neural tensor factorisation

Knowledge graphs (KGs) are a framework used to represent logical structure in data [46]. The resource description framework (RDF) formalism, subject-predicate-object, is especially useful in encoding the relational data representation in KGs: dyadic relationships between entities expressed as nodes and edges. Recent approaches to include statistical learning focused on feature representation of data, with relational data representation having been attempted in a KG link prediction setting [42, 15, 41] for statistical relational learning (SRL). Tensor factorisation has been explored as an approach to link prediction [64, 13, 88]. This has shown promise as a machine learning approach in domains that are high-dimensional and sparse. It is also compute efficient, and lends itself to GPU computation as is common in modern machine learning workloads.

KG tensor factorisation decomposes relational score tensors into an entity matrix, a relational tensor and a transposed entity matrix. The entity matrix is composed of entity vectors, which are latent entity representations consisting of trainable parameters that need to be learned. The relational tensor is composed of relational matrix slices, which are latent relational representations also consisting of trainable parameters that need to be learned, corresponding to the set of relation types in the KG. The product between the entity matrix, the relational tensor and the transposed entity matrix is the bilinear tensor product. In practice this is performed by expressing KG facts (entity-relation-entity) as RDF triples (subject-predicate-object). A vector-matrix product is taken between the subject and predicate, generating a subject-predicate vector, and an inner product of this vector and the object vector is taken. Figure 3.1 illustrates. This operation computes a relational score tensor with unnormalised scores of potential relationships between entities, indexed by relation. Each score is a measure of confidence in the

relationship between two entities, and can be passed through a sigmoid function to produce a probability of relational plausibility.

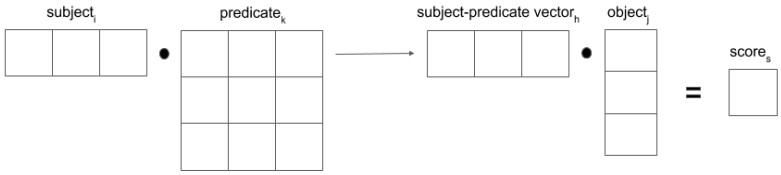



Figure 3.1: Relation prediction between two entities. Subject $i$ and object $j$ are two distinct entities, and predicate $k$ is a relation from the set of relations in the KG. A product is taken between subject-predicate vector $h$ and an object $j$ to compute a relational score $s$.

Previous tensor factorisation approaches focused on limiting the total number of model parameters in order to scale to large KGs. This approach was often at the expense of complex entity-relational interaction modelling. The neural tensor network (NTN) [79], a nonlinear extension of tensor factorisation, was the first successful attempt to take advantage of the expressiveness of neural compositional models. It relies on using a recursive network (RCN) [70] to output a compositional score between two entities, adding that score to the bilinear tensor product, and passing the result through a hyperbolic tangent function. The product between the result and an output layer of relation parameters is then taken, which generates a measure of confidence in potential relations between the two entities.

Convolutional entity-relational modelling (ConvE) [19] was similarly effective at extending tensor factorisation. ConvE lengthwise concatenates reshaped subject and predicate vectors into an subject-predicate matrix, then applies a 2-dimensional ($2D$) convolution operation on this matrix, which leads to convolutional tensor factorisation. This generates feature maps which are reshaped and flattened into a hidden vector and then passed through a nonlinearity. A product is then taken between the vector and an object vector, to compute a relational score. Here a sigmoid function is applied to generate a probability of a potential relationship between the two entities.

HypER [5] extends convolutional tensor factorisation. It models entity-relational interactions by using a hypernetwork [30] to generate $2D$ relation-specific convolutional filters as predicate matrices. A $2D$ convolution is then taken between a subject vector and predicate matrix, which generates a feature map that is reshaped and passed through a fully-connected layer with a nonlinearity. An inner product is taken between the generated hidden vector and an object vector, which computes a relational score. Once again a sigmoid function is applied to generate a probability of a potential relationship.

In the rest of this chapter we assess the NTN model and attempt a simple improvement by applying early stopping [71], adaptive moment estimation (Adam) optimisation [43], and hyperparameter random search [7] to the training algorithm. We then propose HypER+, an extension of the HypER model which compensates for covariate shift introduced by the hypernetwork when generating relation-specific convolutional filters. Finally we extend HypER+ by initialising entity and relation model embeddings using pre-trained word vectors from the GloVe language model [67]. In doing so we take advantage of semantic information contained in these vectors.

## 3.1 Neural tensor networks

### 3.1.1 Nonlinear latent feature modelling

Nickel et al. [64] introduced the bilinear tensor product for relational scoring, in a model called RESCAL. This model makes use of entity vectors and a relational tensor, where each relation type is modelled as a matrix slice of the tensor. RESCAL is defined as follows:

$$f_{i,k,j} = e_i^T W_k e_j = \sum_{a=1}^{H_e} \sum_{b=1}^{H_e} W_{a,b,k} e_{ia} e_{jb} \,, \tag{3.1}$$

where $f_{i,k,j}$ is the relational score, $e_i$ is the subject vector and $e_j$ is the object vector. The latter two are entity representations implemented as model embeddings. $W_k$ is the predicate matrix for relation $k$, which is a relation representation implemented as a model embedding. Entity-relational interactions are modelled using the vector-matrix product between the subject vector and predicate matrix, generating a subject-predicate vector. A relational score is then computed by taking the inner product between the subject-predicate vector and the object vector. RESCAL is thus a linear tensor factorisation composition.

A natural extension to this model is to include a nonlinearity for relational scoring. This extension was introduced by Jenatton et al. [40]. Their model uses a sigmoid nonlinearity to generate a probability of a possible relationship from the relational score computed using the bilinear tensor product. The model is defined as follows:

$$n_{i,j,k} = e_i^T W_k e_j \,, \tag{3.2a}$$

$$\mathbb{P}\left[R_j(S_i, O_k) = 1\right] = \sigma(n_{i,j,k}) \,, \tag{3.2b}$$

where $n_{i,j,k}$ is the relational score, $\sigma(t) = 1/(1+e^{-t})$ is the sigmoid function which maps to the range $(0,1)$, and $\mathbb{P}$ is the probability of a potential relationship between subject $S_i$ and object $O_k$, indexed by relation $R_j$. This model introduces nonlinear relational scoring to latent feature modelling based link prediction.

### 3.1.2 Recursive neural tensor factorisation

Chen et al. [79] introduced nonlinear tensor factorisation by adding recursive entity representations in the composition of the relational score. Recursive networks (RCNs) try to capture the rules for word combinations by constructing a composition tree between words, in a sequence of words [80]. The NTN model tries to take advantage of these compositional rules by adding them to the bilinear tensor product, illustrated in Figure 3.2, and is defined as follows:

$$g_{i,k,j} = u_k^T f(e_i^T W_k^{[1:m]} e_j + V_k \begin{bmatrix} e_i \\ e_j \end{bmatrix} + b_k) \,, \tag{3.3}$$

where $g_{i,k,j}$ is the relational score tensor, $u_k^T \in \mathbb{R}^{1\times m}$ is an output layer of trainable parameters, and $f$ is the tanh activation function. $e_i^T W_k^{[1:m]} e_j$ is the bilinear tensor product between subject vector $e_i^T \in \mathbb{R}^{1\times n}$ and object vector $e_j \in \mathbb{R}^{n\times 1}$, computed for relations $W_k \in \mathbb{R}^{n\times n}$, where $k \in \{1 \dots m\}$, and $b_k$ is the bias. $V_k \begin{bmatrix} e_i \\ e_j \end{bmatrix}$ is the recursive entity composition, where $V_k \in \mathbb{R}^{k\times 2n}$ is a matrix of trainable parameters, and the matrix-vector product is taken between $V_k$ and the concatenation of the subject and object vectors. A row of matrix $V_k$ corresponds to relation $k$. The output of a subject $i$ and object $j$ input, $\{e_i, e_j\}$, is a vector of relational scores $\{s_{i,1,j}, \dots, s_{i,m,j}\}$, indicating the plausibility of relationships that may exist between $i$ and $j$.

The contrastive max-margin loss is minimised during training. This loss computes a confidence magnitude on a correct sample (a fact present in the KG), and a confidence magnitude on a corrupt sample (a randomly generated fact not present in the KG). The correct and corrupt samples are used in the loss function as follows:

$$J(\Omega) = \sum_{i=1}^{N} \sum_{c=1}^{C} \max(0, 1 - g(T^{(i)}) + g(T_c^{(i)})) + \lambda \left\|\Omega\right\|_2^2 \,, \tag{3.4}$$

where $J$ is the loss value, $\Omega$ contains the trainable parameters $u, W, V, b$, and $E$. $N$ is the number of training sample triplets, and $C$ is the number of randomly corrupted facts. $g(T^{(i)})$ is the

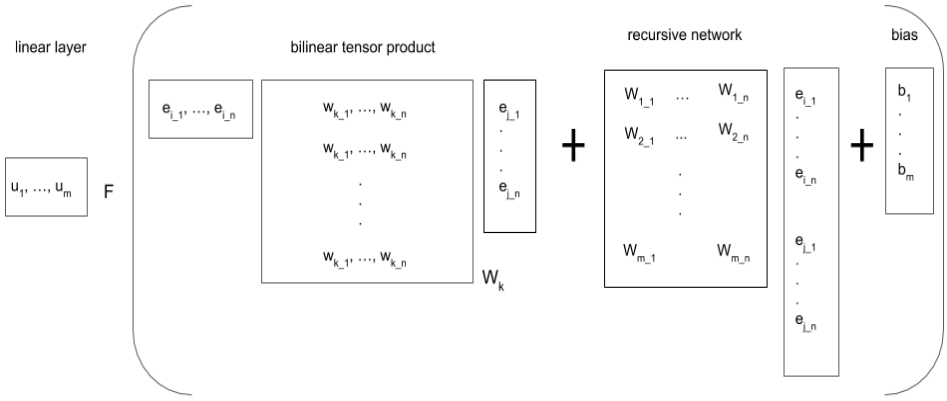



Figure 3.2: Recursive neural tensor network. This model computes the bilinear tensor product, and adds it to a recursive composition of entities and a bias. The result is squashed using a hyperbolic tangent function, and then multiplied by an output vector of relation parameters.

confidence in the true fact computed by the model, and $g(T_c^{(i)})$ is the confidence in the corrupt fact computed by the model. Finally $\lambda \|\Omega\|_2^2$ is the ridge ($L_2$) regulariser.

The NTN model training algorithm also makes use of pre-trained word vectors, from a language model by Turian et al. [89], to initialise entity embeddings. These are 100-dimensional vectors, which are aggregated from the set of words that represent an entity. For example for the entity *homo sapiens*, the resulting entity embedding is $V_{homo\ sapiens} = 0.5(V_{homo} + V_{sapiens})$. These entity embeddings are trainable, and updated using backpropagation to generate a distributed representation of words that aligns with the KG of the particular application.

### 3.1.3 Optimisation

Overfitting can be a concern for models with large numbers of parameters. This problem results in poor generalisation to testing data by the model. It can occur when training has progressed beyond the optimal point, and the model parameters begin to bias toward the training data. High model bias is revealed by good model performance on training data, and poor performance on validation data. Early stopping [71] is often used to prevent this and encourage generalisation.

Neural model training can suffer from slow convergence when using stochastic gradient descent, mini-batch optimisation, and regularisation techniques such as dropout. Neural models can also suffer from insufficient signal to update parameters, which is a problem known as sparse gradients. Adaptive moment estimation (Adam) is a first-order gradient-based stochastic optimisation algorithm that compensates for these problems [43]. It computes the

adaptive learning rates for parameters using the first- and second-order moments of the gradient, combining the advantages of AdaGrad [24] and RMSprop [86]. Adam efficiently regulates the size of parameter updates, accelerating toward local optima, and remaining small in sparse regions where the gradient signal is small.

Setting model hyperparameters remains a challenge in neural model training. A simple method proposed to address this problem is hyperparameter random search [7], which is the process of defining intervals for model hyperparameters, and randomly sampling from those intervals for a finite number of training runs (experiments). This approach improves on grid search, which is a brute force process that iterates through combinations of hyperparameter intervals, as models often match or exceed performance in a fraction of the training time. It takes advantage of the fact that for most datasets, only a subset of the hyperparameters contribute meaningful variance in model performance, thus eliminating the need for an exhaustive search over a large combinatorial space.

### 3.1.4 Training algorithm

Doss et al. [23] reimplemented the NTN model in TensorFlow [2]. This model underperforms compared to the original model, relying on AdaGrad optimisation and the same hyperparameters. We apply early stopping, Adam optimisation as well as hyperparameter random search, in an attempt to improve performance. The new training algorithm is given in Algorithm 2.

## 3.2 Hypernetwork tensor factorisation

### 3.2.1 Convolutional factorisation

ConvE introduced the convolutional operator to neural tensor factorisation [19]. Specifically, this operator increases expressiveness in entity-relation interaction modelling by using $2D$ convolutions, which have been found to be particularly effective at modelling the interactions of entities involved in a large number of relations. This may be due to filter parameter sharing between entity-relational combination features, but perhaps more pertinently, the convolutional operator captures a larger variety of entity-relational feature interactions when summarising regions between the respective representations, whereas the bilinear tensor product performs interaction modelling using a simple inner product.

**Algorithm 2:** Optimised NTN training algorithm

**Input** Training set $D = \{(x_i, y_i)\}_{i=1}^{N}$, of samples and labels
**Repeat** for $Z$ experiments using random uniform hyperparameter configuration
**for** *Epoch* $k = 1 \ldots K$ **do**
  **Input for** *Mini-batch* $b = 1 \ldots B$ **do**
    $S \leftarrow sample(D, b)$ // sample mini-batch $b$
    **for** $(x, y) \in S$ **do**
      $y^{'}_{correct} \leftarrow predict(x)$ // for $\{e_i, e_{jt}\}$ predict relational score $g(T^{(i)})$
      $y^{'}_{corrupt} \leftarrow predict(x)$ // for $\{e_i, e_{jc}\}$ predict relational score $g(T_c^{(i)})$
      $l \leftarrow contrast(y^{'}_{correct}, y^{'}_{corrupt})$ // compute error
      $e \leftarrow e + l$ // add loss to zero-initialised mini-batch loss $e$
    **end**
    Update model w.r.t. $e$ // using Adam optimiser
  **end**
  $V \leftarrow sample(D, v)$ // sample validation data of size $v$
  **for** $(x, y) \in V$ **do**
    $y^{'} \leftarrow predict(x)$ // predict relational score
    $Y^{'} \leftarrow append(Y^{'}, y^{'})$ // append prediction $y^{'}$ to predictions vector $Y^{'}$
  **end**
  $vScore_t \leftarrow accuracy(Y^{'}, Y)$ // compute current epoch validation score
  if $vScore_t < vScore_{t-1}$ // check validation score against previous epoch score
    Stop training // early stopping
  else
    $vScore_{t-1} \leftarrow vScore_t$ // assign current validation score to previous score

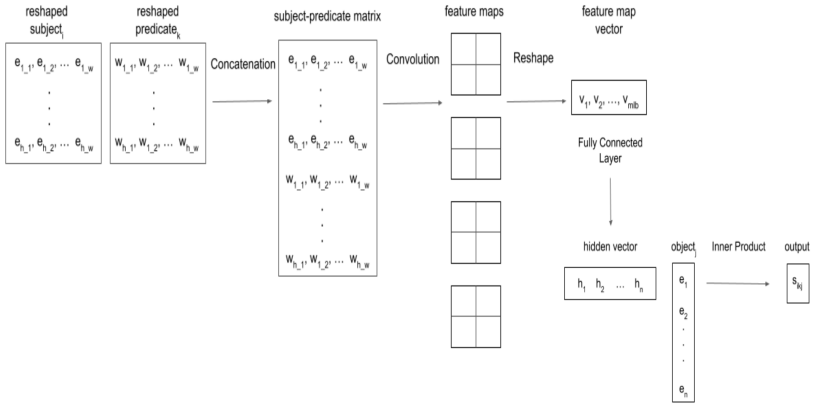



Figure 3.3: ConvE forward pass which computes a relational score $s_{i,k,j}$ between subject $i$, predicate $k$ and object $j$.

ConvE concatenates subject and predicate matrices along the row axis, creating a subject-predicate matrix. Convolutions are then taken between the matrix and filters, producing feature maps which are reshaped into a vector, flattened by a fully-connected layer and passed through a nonlinearity, to generate a hidden vector. The inner product of the hidden vector is then taken with the object vector to compute an unnormalised relational score between the subject and object. A sigmoid function is applied to generate a probability of relational plausibility.

The ConvE model is defined as follows:

$$\psi_k(e_i,\, e_j) = g(vec(f([\overline{e_i};\, \overline{w_k}]) * c)W)e_j\,, \tag{3.5}$$

where $\psi_k$ is the relational score between subject $e_i \in \mathbb{R}^{n\times 1}$ and object $e_j \in \mathbb{R}^{n\times 1}$ for predicate $w_k \in \mathbb{R}^{n\times 1}$. $\overline{e_i} \in \mathbb{R}^{n_w \times n_h}$ is a $2D$ reshaping of $e_i$, and similarly $\overline{w_k} \in \mathbb{R}^{n_w \times n_h}$ is a $2D$ reshaping of $w_k$, where $n = n_w \times n_h$, and $f$ is the lengthwise concatenation operation between $\overline{e_i}$ and $\overline{w_k}$. $c$ contains $2D$ convolutional filters operated on by $*$, the convolutional operator. This generates the set of feature maps $C \in \mathbb{R}^{l\times b\times m}$, where $l$ and $b$ are the $2D$ feature map dimensions, and $m$ is the number of maps. $vec$ reshapes $C$ into a vector $v \in \mathbb{R}^{1\times lbm}$ which is passed through a fully-connected layer parameterised by $W \in \mathbb{R}^{lbm\times n}$, and generates a vector $h \in \mathbb{R}^{1\times n}$ which is passed through a ReLU nonlinearity $g$. Finally the inner product is taken with this vector and $e_j$ to compute a relational score. ConvE is thus a convolutional tensor factorisation. Figure 3.3 illustrates the ConvE forward pass.

The generated relational probability is defined as follows:

$$\mathbb{P} = \sigma(\psi_k(e_i,\, e_j)) \,. \tag{3.6}$$

The binary cross-entropy loss is minimised during training. This loss function is particularly appropriate as we expect only a single object to belong to the fact (subject-predicate-object) and generate a probability close to 1, and every other object to not belong to the fact, generating a probability close to 0.

The loss function is defined as follows:

$$L(p,\, t) = -\frac{1}{N}\sum_i (t_i \cdot log(p_i) + (1 - t_i) \cdot log(1 - p_i)) \,, \tag{3.7}$$

where $L$ is the loss value, $p_i$ is the relational probability computed by the model, and $t_i$ is the target. $N$ is the batch size, and $i$ is the sample index.

### 3.2.2 Hypernetwork factorisation

#### Hypernetworks

A hypernetwork is a meta-network that generates parameters for a main network [30]. It compresses layer weights of a main network into an input embedding vector, which is analogous to encoding a main network configuration. Hypernetworks are used to discover an efficient subset of main network parameters, reducing the total number of parameters without sacrificing performance. They are also used to enable parameter sharing across layers of a main network, similar to recurrent networks, however without incurring the problem of vanishing and exploding gradients.

A standard tensor serves as input to a main network, for example an image or sequence of words, while an embedding vector serves as input to a hypernetwork. The embedding vector size and hypernetwork layer weights are used to determine the respective layer weight sizes of the main network. Static or dynamic embedding vectors can be used as hypernetwork input. For the dynamic case, different embedding vectors are used at each time step, and can be useful for adapting the main network to its input sequence. Embedding vector parameters are learned during end-to-end training of the main network, using backpropagation.

For a feed-foward convolutional network with $D$ layers, the hypernetwork can be used to generate main network weight tensors $K^j \in \mathbb{R}^{f_w f_h d_i d_o}$ for each layer $j = 1, \ldots D$, where $K^j$ is a

set of filters with filter size $f_w \times f_h$, $d_i$ is the number of input channels, and $d_o$ is the number of output channels. The hypernetwork model can then be defined as follows:

$$K^j = g(z^j), \quad \forall j = 1, \ldots, D\,, \tag{3.8}$$

where $K_j$ contains the parameters of layer $j$, $g$ is the hypernetwork composition, and $z_j \in \mathbb{R}^n$ is the embedding vector input. If the hypernetwork is a two-layer multilayer perceptron (MLP), $g(z^j)$ can be defined as follows:

$$a_i^j = W_i z^j + B_i, \quad \forall i = 1, \ldots, d_o, \forall j = 1, \ldots, D\,, \tag{3.9a}$$

$$K_i^j = W_{out} a_i^j + B_{out}, \quad \forall i = 1, \ldots, d_o, \forall j = 1, \ldots, D\,, \tag{3.9b}$$

$$K^j = (K_1^j, K_2^j, \ldots, K_{d_o}^j), \quad \forall j = 1, \ldots, D\,, \tag{3.9c}$$

where $a_i^j \in \mathbb{R}^m$ is the output of the first layer of the hypernetwork, and $i$ corresponds to feature map $i$. $W_i \in \mathbb{R}^{m \times n}$ contains the first layer parameters and $B_i \in \mathbb{R}^m$ is the bias. $K_i^j$ is filter $i$ for layer $j$ in the main network. $W_{out} \in \mathbb{R}^{f_w f_h d_i \times m}$ contains the second layer parameters of the hypernetwork, and $B_{out} \in \mathbb{R}^{f_w f_h d_i}$ is the bias. Finally $K^j$ is the set of filters for layer $j$ of the main network.

### HypER

HypER is inspired by ConvE, and implements a convolutional operator that models entity-relation feature interactions [5]. It makes use of a hypernetwork to generate relation-specific convolutional filters, where the subject and predicate filter are used in a convolution operation to generate a subject-predicate feature map. This is reshaped into a vector and passed through a fully-connected layer which outputs a hidden vector that is passed through a nonlinearity. An inner product is then taken with the object vector to produce a relational score, as illustrated in Figure 3.4. Finally a sigmoid function is applied to the score, which generates a probability for a potential relationship between the two entities.

The HypER model is defined as follows:

$$\phi_k(e_i, e_j) = f(vec(e_i * (vec^{-1}(w_k H)))W)e_j \tag{3.10}$$

where $\phi_k$ is the relational score between subject $e_i \in \mathbb{R}^{n \times 1}$ and object $e_j \in \mathbb{R}^{n \times 1}$ for predicate $w_k \in \mathbb{R}^{1 \times n}$. $H \in \mathbb{R}^{n \times h}$ are the fully-connected layer parameters of the hypernetwork. $vec^{-1}$ is

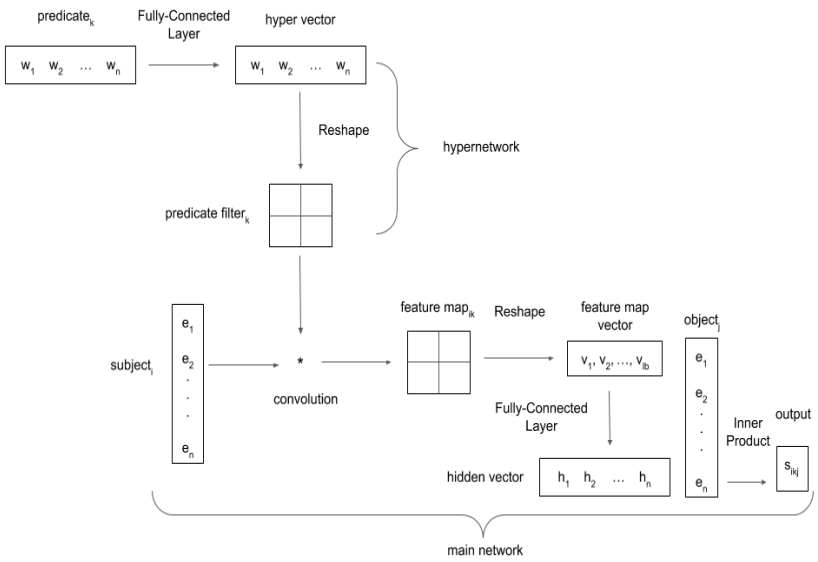



Figure 3.4: Hyper-convolutional entity representations. The hypernetwork takes the predicate embedding $k$ as input, which generates predicate-specific convolutional filter. A convolutional is taken between this filter and a subject embedding $i$, which generates a feature map which is reshaped into vector $v$. A fully-connected layer transforms the feature map vector to a hidden vector $h$, and the inner product is taken between this vector and the object embedding $j$ to produce relational score $s_{ikj}$.

the transformation that reshapes the output of the hypernetwork into a set of relation-specific convolutional filters $c_{ik}$. $*$ is the convolutional operator that takes the convolution between $e_i$ and $c_{ik}$, generating a set of feature maps $C_{ik} \in \mathbb{R}^{l \times b}$, where $l$ and $b$ are the feature map dimensions. *vec* reshapes the feature maps into a vector $v \in \mathbb{R}^{1 \times lb}$ which is passed through a fully-connected layer parameterised by $W \in \mathbb{R}^{lb \times n}$, generating a hidden vector $h \in \mathbb{R}^{1 \times n}$ which is passed through $f$, a ReLU nonlinearity. HypER is thus a hyper-convolutional tensor factorisation.

The relational probability is computed as follows:

$$\mathbb{P} = \sigma(\phi_k(e_i,\ e_j))\ . \tag{3.11}$$

The HypER training algorithm minimises a binary cross-entropy loss. Entity and relation embeddings are initialised using Xavier initialisation [26], from a uniform distribution with zero mean and variance of $\frac{2}{n}$, where $n$ is the embedding length.

### 3.2.3 Covariate shift in hypernetworks

We make the observation that hypernetworks may also suffer from covariate shift. The hypernetwork generates relational filters for the main network from relational embedding inputs. During training, the distribution of the latent parameters of the relational embeddings change, altering the inputs used to generate the filters. We consider whether this change in distribution makes it hard for the hypernetwork's fully-connected layer to learn the most useful parameters for creating relational filters. We further consider that given the relational filters are used early (upstream) in the main network, the effect of their suboptimal representations might be amplified.

To address this issue, we introduce relational input batch normalisation. This operation performs normalisation on relational input mini-batches during training, such that each batch has zero mean and unit variance, to regulate hypernetwork input. The components of batch normalisation are as follows:

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \,, \tag{3.12a}$$

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i-\mu_B)^2 \,, \tag{3.12b}$$

$$\hat{x_i} \leftarrow \frac{x_i-\mu_B}{\sqrt{\sigma_B^2+\varepsilon}} \,, \tag{3.12c}$$

$$y_i \leftarrow \gamma\hat{x_i}+\beta \equiv BN_{\gamma,\beta}(x_i) \,, \tag{3.12d}$$

where $\mu_B$ is the mini-batch mean, $\sigma_B^2$ is the variance, $\hat{x_i}$ is the normalised input, and $y_i$ is the result of scaled and shifted normalisation. Parameters $\gamma$ and $\beta$ are learned during training. Population mean and variance, and the parameters $\gamma$ and $\beta$, are used to regulate layer inputs during inference. The components are as follows:

$$E[x] \leftarrow E_B[\mu_B] \,, \tag{3.13a}$$

$$Var[x] \leftarrow \frac{m}{m-1}E_B[\sigma_B^2] \,, \tag{3.13b}$$

$$y = \frac{\gamma}{\sqrt{Var[x]+\varepsilon}}\cdot x + \left(\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\varepsilon}}\right) \equiv BN_{\gamma,\beta}(x) \,, \tag{3.13c}$$

where $E[x]$ is the population mean, $Var[x]$ is the population variance, and $y$ relies on the $\gamma$ and $\beta$ parameters transformed using population statistics. We adjust the HypER model accordingly, and call the adjusted model HypER+. Algorithm 3 presents the updated training algorithm.

**Algorithm 3:** Training HypER+ with batch normalisation

**Input** Training set $D = \{(x_i, y_i)\}_{i=1}^{N}$, of input features and output labels
Initialise parameters $w$, for model $M(w)$ // e.g. random normal initialisation
**for** *Epoch* $k = 1 \ldots K$ **do**
  **for** *Mini-batch* $b = 1 \ldots B$ **do**
    $S \leftarrow sample(D, b)$ // sample mini-batch $b$
    $j \leftarrow j \in \{$*hypernetwork input layer, main network input layer main network convolution layer*$\}$
    **for** $(x, y) \in S$ **do**
      $y'_j = BN_{\gamma_j,\beta_j,\mu_{S_j},s_{S_j}}(x_j)$ // normalisation of outputs from layers $j$
      $y' \leftarrow predict(x, w)$ // predict label for sample $x$ using parameters $w$
      $l \leftarrow loss(y - y')$ // compute binary cross-entropy loss
      $e \leftarrow e + l$ // add loss to zero-initialised mini-batch loss $e$
    **end**
    $w \leftarrow w - \lambda \cdot J_e(w)$ // gradient descent update of model where $\lambda$ is the learning rate, and $J_e(w)$ are the partial derivatives of the loss with respect to the parameters $w$
    $\gamma_j \leftarrow \gamma_j + \Delta\gamma_j$, $\beta_j \leftarrow \beta_j + \Delta\beta_j$ // optimise batch normalisation parameters
  **end**
**end**
$E[x] \leftarrow E[\mu_B]$, $Var[x] \leftarrow E[\sigma_B^2]$ // average $\mu_B$ and $\sigma_B^2$ over mini-batches $B$
$\gamma \leftarrow \frac{\gamma}{\sqrt{Var[x]+\varepsilon}}$, $\beta \leftarrow \left(\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\varepsilon}}\right)$ // update $\gamma$ and $\beta$ with population statistics

### 3.2.4 Pre-trained word vectors

We also incorporate pre-trained word vectors from the GloVe [67] language model into the HypER+ training algorithm. KGs are comprised of respective vocabularies of entities and relations. These vocabularies contain a KG word to ID map, which is used during model training to identify the respective entity or relation. The language model itself is a map of words to vectors, and is used to look up the corresponding vectors for entities or relations in the KG. Where a pre-trained vector does not exist for the word, a randomly initialised vector is generated in its place. This makes the size of the language model important, as it determines the total pre-trained vector coverage of KG words.

We use GloVe word vectors to initialise 200$D$ entity and relation embeddings. The respective embeddings are generated by aggregating the set of vectors corresponding to the sequence of words that describe them; the same method of aggregation used to construct embeddings for NTN model training. This process generates two KG pre-trained vector to ID maps, one for entities and one for relations, and these maps are used to initialise the model entity and relation embeddings respectively. The embeddings are trainable, and updated using backpropagation.

## 3.3 Summary

Neural tensor networks constitute an early successful attempt at using the expressive power of neural networks for link prediction. These models make use of RCNs, which try to model semantic compositionality, and also make use of pre-trained word vectors to improve relation prediction performance. We apply training algorithm optimisations to the NTN model aimed at improving its performance, namely early stopping, the Adam optimiser and hyperparameter random search.

Hypernetworks can be used to compress parameters for a main network. This approach is analogous to applying a configuration to a main network, and is used to reduce the number of parameters, as well as enable parameter sharing between layers. The HypER model uses this architecture to generate relation-specific filters for a convolutional main network. These filters are then used to generate relational probabilities between entities. We optimise the HypER training algorithm to compensate for potential covariate shift caused by hypernetworks, and propose HypER+. We then extend HypER+ to make use of pre-trained word vectors from the GloVe language model, and use them to initialise entity and relation embeddings.

# Chapter 4

# Results and analysis

This chapter examines the three hypotheses of this study, including the application of training algorithm optimisations to recursive neural tensor networks (NTNs), compensating for covariate shift introduced by hypernetworks during convolutional tensor factorisation, and finally the initialisation of entity and relation embeddings using pre-trained word vectors.

## 4.1 Recursive neural tensor networks

### 4.1.1 Datasets

For the first set of experiments we use the WordNet [55] and Freebase [11] link prediction benchmark datasets. WordNet is a lexical database for English, and a taxonomy with hypernym relationships ("is a") and synonym sets. Freebase is a large collaborative knowledge base consisting of data about the world, composed mainly by community members. It is an online collection of structured data harvested from many sources, including user-submitted wiki contributions. The WordNet dataset contains a total of 136,611 triples, and Freebase a total of 375,499 triples. Visualisations of the respective knowledge graphs (KGs), as well as a sample of resource description framework (RDF) triple encoded facts, are presented in Figures 4.1 to 4.4. KG summary statistics are presented in Figures 4.5 to 4.7, and Table 4.1.

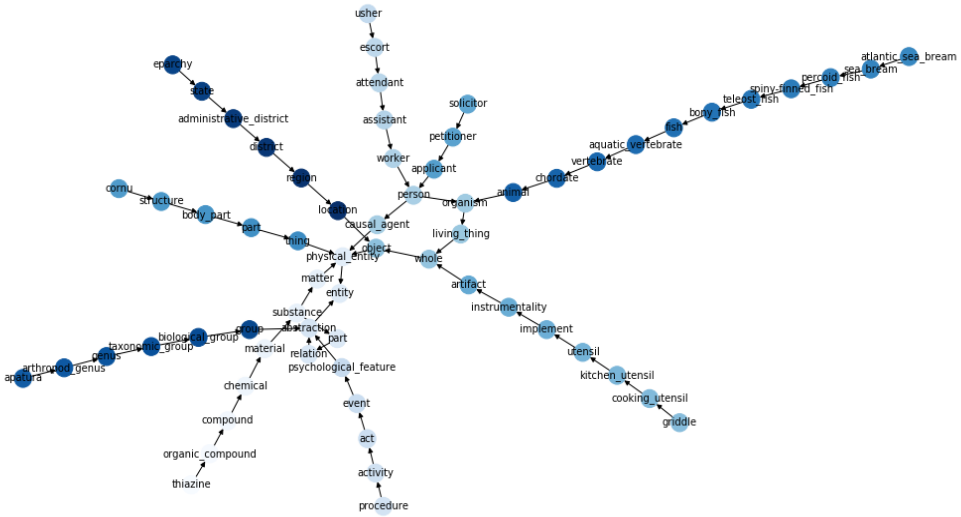



Figure 4.1: A subset of WordNet facts structured as a KG. Entities are nodes, and relations are edges, where facts are encoded as RDF triples.

| id | subject | s_name | predicate | object | o_name |
|---|---|---|---|---|---|
| 1 | 115525 | spiritual_bouquet | type of | 103855 | sympathy_card |
| 2 | 115525 | spiritual_bouquet | synset domain topic | 85313 | church_of_rome |
| 3 | 92849 | absorption | type of | 94868 | attention |
| 4 | 92849 | absorption | has instance | 104271 | centering |
| 5 | 112427 | avenge | type of | 88820 | penalise |
| 6 | 112427 | avenge | has instance | 105584 | get_back |
| 7 | 100673 | chamaecyparis_lawsoniana | member holonym | 89687 | chamaecyparis |
| 8 | 109091 | cutter | type of | 85704 | boat |
| 9 | 94530 | solar_array | type of | 89188 | artificial_satellite |
| 10 | 94530 | solar_array | has part | 106656 | photovoltaic_cell |

Figure 4.2: A sample of RDF triples from the WordNet KG.

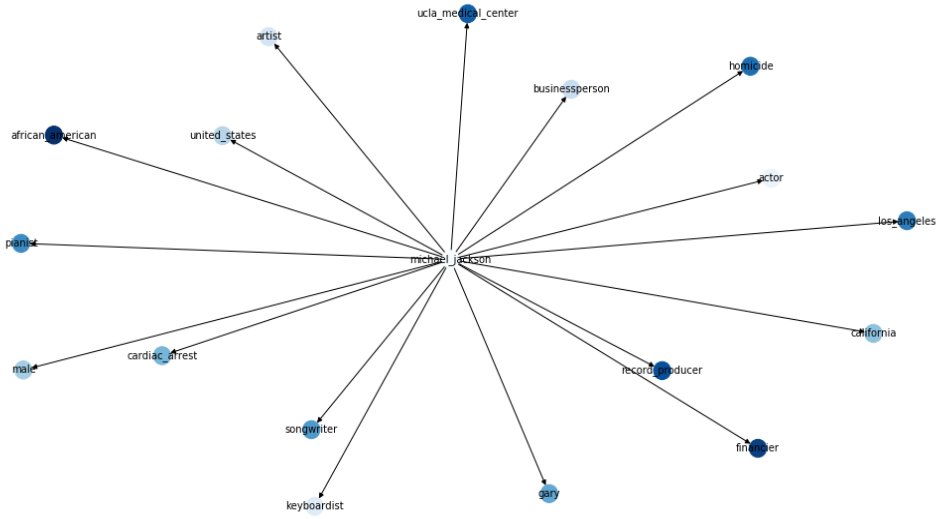
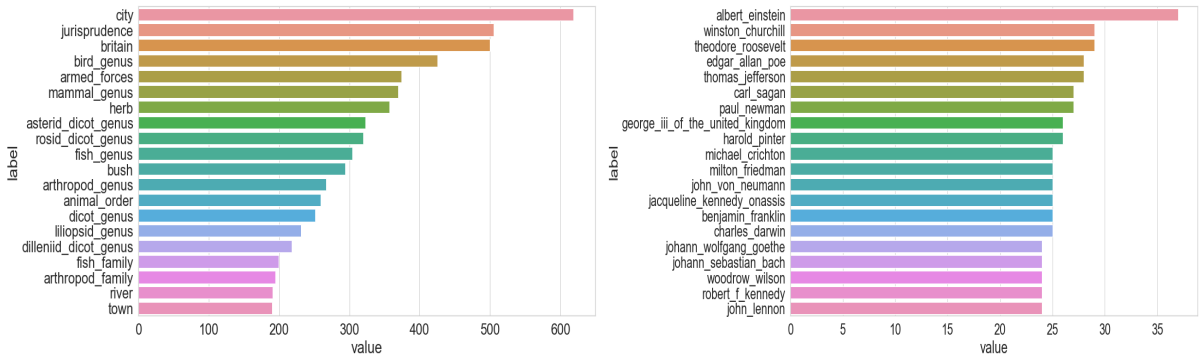



Figure 4.3: A subset of Freebase facts structured as a KG. Due to the size of Freebase, only a subset of facts related to the subject "Michael Jackson" is presented.

| id | subject | s_name | predicate | object | o_name |
|---|---|---|---|---|---|
| 1 | 13945 | antoine_brutus_menier | religion | 26 | roman_catholic_church |
| 2 | 52253 | denys_rayner | cause of death | 38 | cancer |
| 3 | 16827 | nietzchka_keene | place of death | 679 | madison |
| 4 | 4838 | friedrich_bessel | profession | 37 | mathematician |
| 5 | 10633 | thomas_harrison_1740 | profession | 30 | engineer |
| 6 | 14063 | richard_brautigan | profession | 18 | novelist |
| 7 | 4633 | anthony_asquith | location | 13 | london |
| 8 | 5306 | robert_noyce | profession | 51 | physicist |
| 9 | 18051 | ignaz_franz_castelli | profession | 1258 | dramatist |
| 10 | 70428 | russell_bufalino | gender | 1 | male |

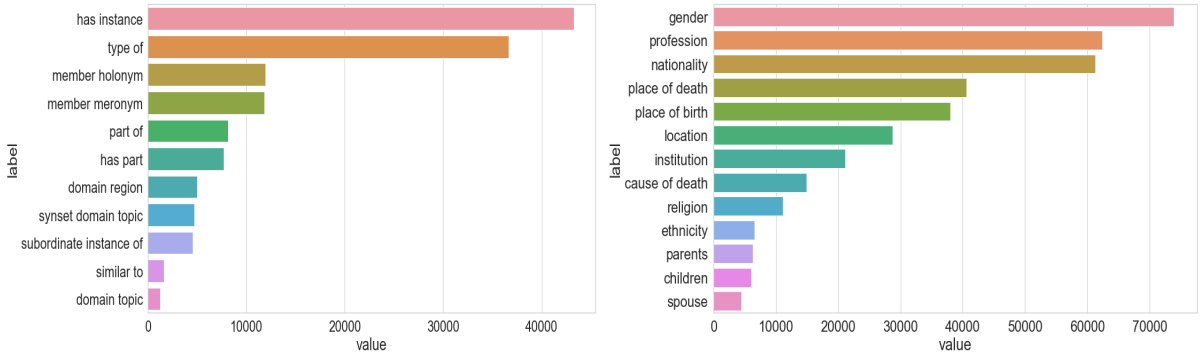
Figure 4.4: A sample of RDF triples from the Freebase KG.

WordNet is comprised of a set of relation types including similar, opposite, subordinate, part, and entailment. Freebase is comprised of an open-ended collection of relation types, including symmetric, asymmetric, transitive, composition, hierarchy, reflexive, irreflexive and inversion. Freebase also contains more entities, relations between entities, and facts, than WordNet.



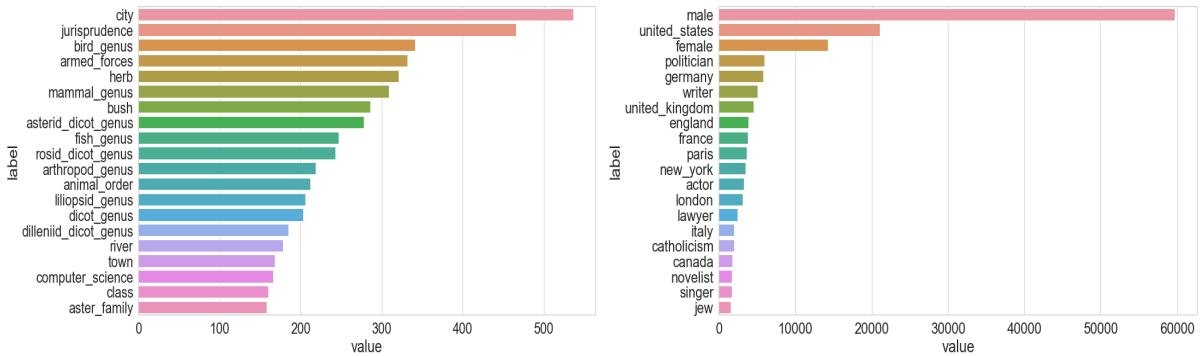

(a) WordNet (b) Freebase

Figure 4.5: Histogram showing the number of times the 20 most frequent subject labels occur in KG facts, in the WordNet and Freebase link prediction datasets.




(a) WordNet (b) Freebase

Figure 4.6: Histogram showing the number of times predicate labels occur in KG facts, in the WordNet and Freebase link prediction datasets.




(a) WordNet (b) Freebase

Figure 4.7: Histogram showing the number of times the 20 most frequent object labels occur in KG facts, in the WordNet and Freebase link prediction datasets.

| | **WordNet** | | | **Freebase** | | |
|---|---|---|---|---|---|---|
| | subject | predicate | object | subject | predicate | object |
| Count | 32,270 | 11 | 33,011 | 67,393 | 13 | 15,342 |
| Max | 619 | 43,312 | 537 | 37 | 73,897 | 59,663 |
| Min | 1 | 1,229 | 1 | 1 | 4,464 | 1 |
| Median | 2 | 7,705 | 3 | 5 | 21,149 | 3 |
| IQR | 2 | 7,258 | 2 | 4 | 34,033 | 6 |

Table 4.1: Statistics of the WordNet and Freebase link prediction datasets. We show counts of unique subject, predicate and object labels, as well as the maximum, minimum, median and interquartile range of label occurrences.

For WordNet, it can be seen that predicates are skewed toward "has instance" with $43,312$ occurrences, and "type of" with $36,659$ occurrences. Freebase predicates are somewhat more uniform, however four predicates have occurrences under $10,000$.

WordNet and Freebase subjects are somewhat uniform, although the median number of occurrences is 2 and 5 respectively, with an interquartile range (IQR) of 2 and 4 respectively. WordNet object occurrences are somewhat uniform while Freebase object occurrences are skewed, with a single object, "male" occurring $59,663$ times, representing $15,88\%$ of facts. This is in comparison to a median object occurrence of 3 and an IQR of 6.

The WordNet dataset is split into training, validation and test sets of $110,362$ $(80.8\%)$, $5,215$ $(3.8\%)$ and $21,034$ $(15.4\%)$ triples respectively. The Freebase dataset is split into training, validation and test sets of $316,232$ $(84.2\%)$, $11,815$ $(3.1\%)$ and $47,452$ $(12.6\%)$ triples respectively. These ratios are chosen to align with those used to train the baseline model.

### 4.1.2 Baseline training algorithm

**Model summary.** Our baseline model for this section is inspired by recursive networks (RCNs). The NTN is a bilinear tensor product between the subject, predicate and object, added to an RCN composition of the subject and object. It computes relational scores between pairs of entities.

**Contrastive max-margin loss.** The contrastive max-margin loss is used to train the NTN model. A relational score is computed for the target triple containing a subject, predicate and object. A relational score is then computed for the same subject and predicate, along with a non-related entity randomly selected and presented as a corrupt object. Relational scores in

the range $(-1,\ 1)$ are produced for the target and corrupt objects, respectively. The training task is to compute a large value for the target score, and small value for the corrupt score. The computed loss is backpropagated through the network to update model parameters.

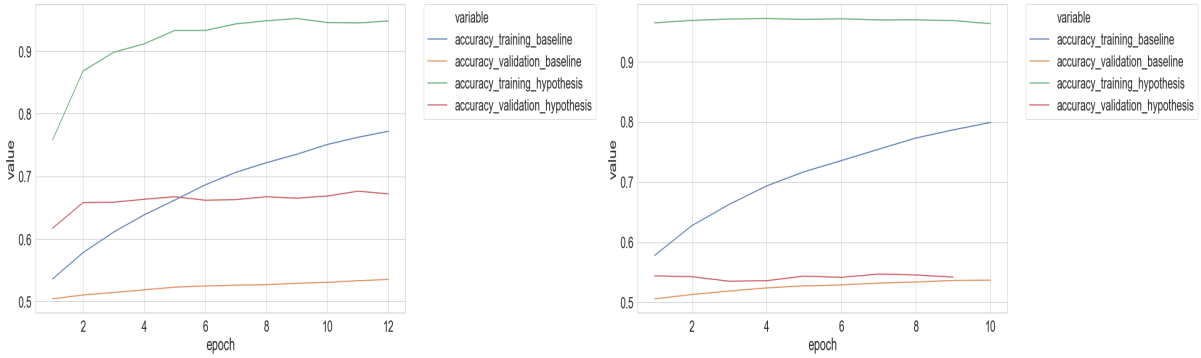
### 4.1.3 Optimised training algorithm

**Implementation.** We use the TensorFlow framework [2] to implement our NTN training algorithms. The NTN model introduced by Chen et al. [79] and reimplemented in TensorFlow by Doss et al. [23], serves as the baseline model. We update the baseline's training algorithm by including early stopping, adaptive moment estimation (Adam) optimisation and hyperparameter random search. We also make use of the same pre-trained word vectors [89] used to initialise entity and relational embeddings. The embedding parameters are adjusted during training to generate latent representations specific to a KG. The models in this section were trained on a MacBook Pro 2015 with 8 cores, 16GB RAM, and 512GB SSD, and no GPU acceleration. We evaluate the models by ranking the accuracy scores of the predicted triples for the respective test sets of WordNet and Freebase.

**Code to reproduce.** In the interest of reproducibility, all code needed to train and test the models in this section can be found at the following links.
Baseline NTN: https://github.com/xhosaBoy/recursive-neural-tensor-networks
Optimised NTN: https://github.com/xhosaBoy/optimised-neural-tensor-network

#### Results

The accuracy results of the NTN model trained using the original baseline training algorithm, compared to the optimised training algorithm are presented in Figure 4.8, as well as Table 4.2. The optimised NTN outperforms the baseline NTN accuracy across both KGs, and significantly outperforms the baseline on the WordNet KG. Hyperparameter random search seems to be responsible for this improvement, as the hypothesis begins outperforming the baseline from the first epoch. All experiments were only able to complete a maximum of 12 epochs before memory saturation and overloading CPU resources.




(a) WordNet (b) Freebase

Figure 4.8: Training and validation accuracy vs training epochs for the two datasets. The optimised algorithm performs significantly better, especially on WordNet.

| Model | WordNet | Freebase | Avg |
|---|---|---|---|
| Original NTN [79] | **.862** | **.900** | **.881** |
| Reimplemented NTN baseline [23] | .562 | .535 | .549 |
| Optimised NTN (ours) | .674 | .548 | .611 |

Table 4.2: Link prediction accuracy on WordNet and Freebase KG test sets. Our reimplemented NTN with the optimised training algorithm outperforms the baseline NTN, but the original NTN algorithm significantly outperforms all models. This may be due to differences in the choice of hyperparameters and optimiser, L-BFGS for the Original NTN, and Adam for the Optimised NTN. The difference between the number of iterations may also be having an impact, 500 and < 500 respectively. A slow but steady increase in the baseline and hypothesis validation accuracy can be seen before compute resources are exhausted.

## 4.2 HypER and HypER+

### 4.2.1 Datasets

For the second set of experiments we use the WN18 [13] and FB15k [13] link prediction benchmark datasets. WN18 is a subset of WordNet, containing 40,943 entities, 18 relations and 151,442 triples. FB15k is a subset of Freebase, containing 14,951 entities, 1,345 relations and 592,213 triples. Visualisations of the respective KGs, as well as a sample of RDF triple encoded facts, are presented in Figures 4.9 to 4.12. KG summary statistics are presented in Figures 4.13 to 4.16, and Table 4.3.

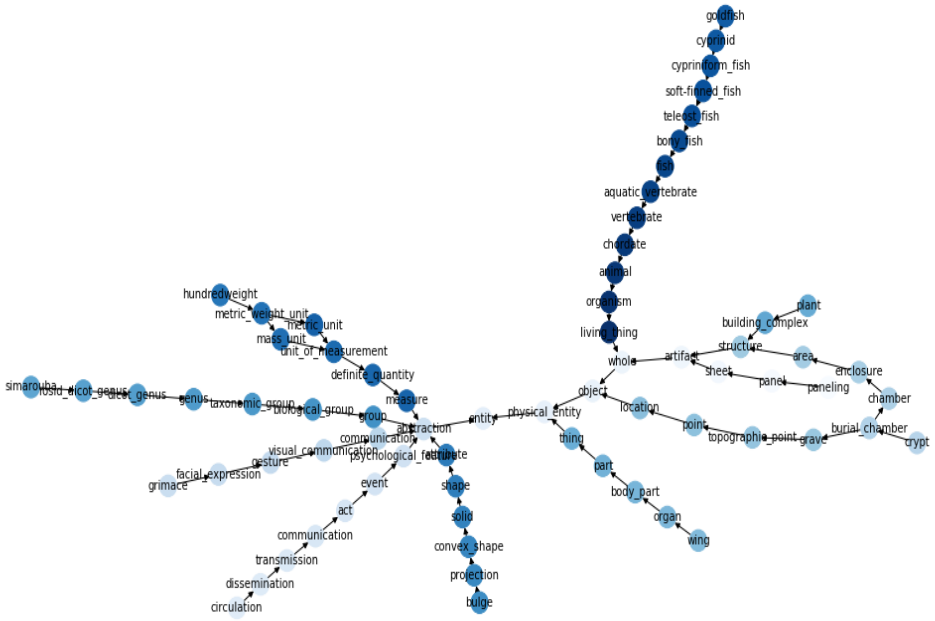



Figure 4.9: A subset of WN18 facts structured as a KG. Entities are nodes, and relations are edges, where facts are encoded as RDF triples.

| id | subject | s_name | predicate | object | o_name |
|---|---|---|---|---|---|
| 1 | 03964744 | toy | hyponym | 04371774 | swing |
| 2 | 00260881 | land reform | hypernym | 00260622 | reform |
| 3 | 02199712 | suborder nematocera | member holonym | 02188065 | order diptera |
| 4 | 01332730 | cover | derivationally related form | 03122748 | covering |
| 5 | 06066555 | phytology | derivationally related form | 00645415 | botanize |
| 6 | 09322930 | kamet | instance hypernym | 09360122 | mountain peak |
| 7 | 11575425 | dilleniid dicot genus | hyponym | 12255934 | genus pyrola |
| 8 | 07193596 | question | derivationally related form | 00784342 | inquire |
| 9 | 05726596 | system | hyponym | 06162979 | ontology |
| 10 | 01768969 | class arachnida | derivationally related form | 02636811 | spidery |

Figure 4.10: A sample of RDF triples from the WN18 KG.

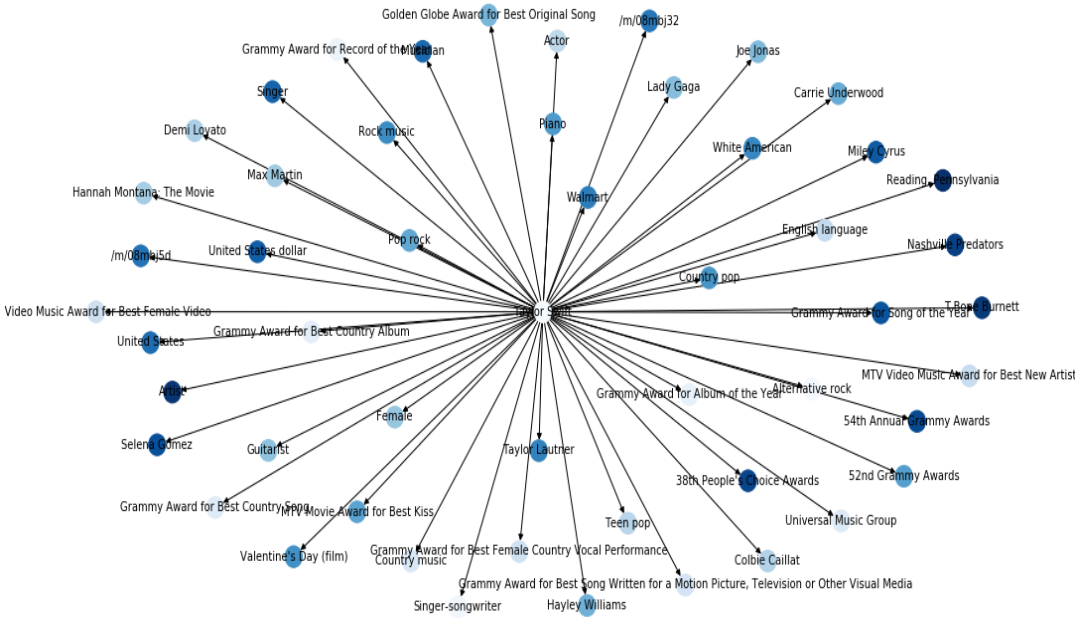
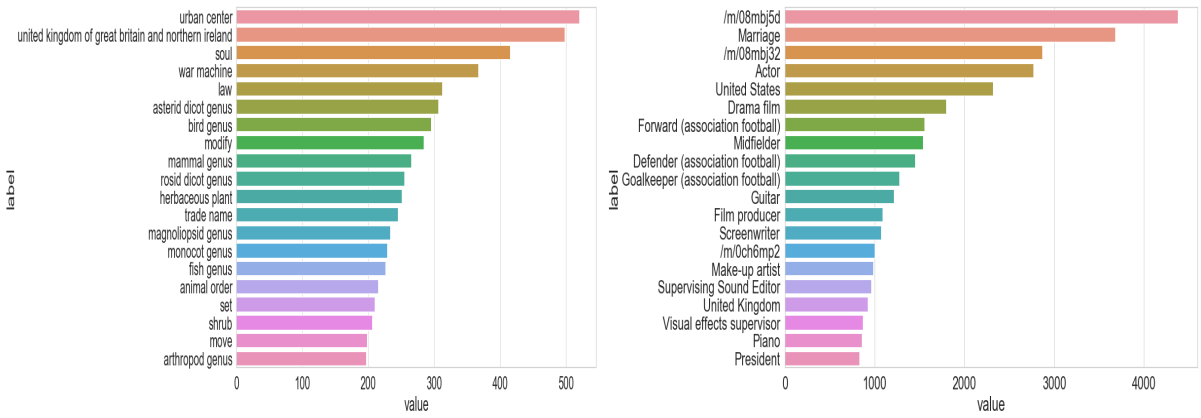



Figure 4.11: A subset of FB15k facts structured as a KG. Entities are nodes, and relations are edges, where facts are encoded as RDF triples.

| id | subject | s_name | predicate | object | o_name |
|---|---|---|---|---|---|
| 104329 | /m/010016 | Denton, Texas | /location/capital of administrative division/capital of./location/administrative division capital relationship/administrative division | /m/0mr_8 | Denton County, Texas |
| 240598 | /m/010016 | Denton, Texas | /location/hud county place/county | /m/0mr_8 | Denton County, Texas |
| 66964 | /m/010016 | Denton, Texas | /location/hud county place/place | /m/010016 | Denton, Texas |
| 155525 | /m/010016 | Denton, Texas | /location/hud foreclosure area/estimated number of mortgages./measurement unit/dated integer/source | /m/0jbk9 | United States Department of Housing and Urban Development |
| 65623 | /m/010016 | Denton, Texas | /location/hud foreclosure area/total 90 day vacant residential addresses./measurement unit/dated integer/source | /m/0jbk9 | United States Department of Housing and Urban Development |
| 113178 | /m/010016 | Denton, Texas | /location/hud foreclosure area/total residential addresses./measurement unit/dated integer/source | /m/0jbk9 | United States Department of Housing and Urban Development |
| 255837 | /m/010016 | Denton, Texas | /location/location/containedby | /m/07b_l | Texas |
| 36778 | /m/010016 | Denton, Texas | /location/location/containedby | /m/0mr_8 | Denton County, Texas |
| 304656 | /m/010016 | Denton, Texas | /location/location/contains | /m/05zjtn4 | University of North Texas College of Music |
| 258802 | /m/010016 | Denton, Texas | /location/location/events | /m/0b_6h7 | 1980 NCAA Men's Division I Basketball Tournament |

Figure 4.12: A sample of RDF triples from the FB15k KG.

WN18 is an intuitively usable dictionary and thesaurus, and supports automatic text analysis, while FB15k is a KG of general facts. FB15k contains more entities, relations between entities, and facts, than WN18.




(a) WN18 (b) FB15k

Figure 4.13: Histogram showing the number of times the 20 most frequent subject labels occur in KG facts, in the WN18 and FB15k link prediction datasets.

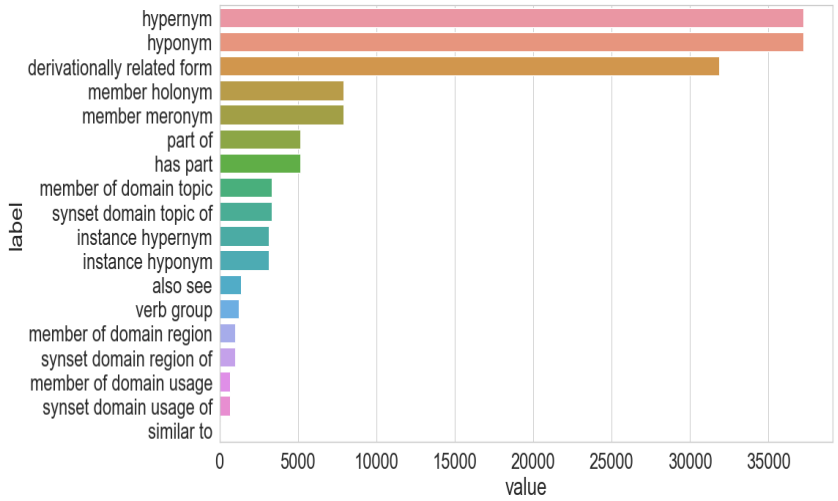



Figure 4.14: Histogram showing the number of times predicate labels occur in KG facts, in the WN18 link prediction dataset.

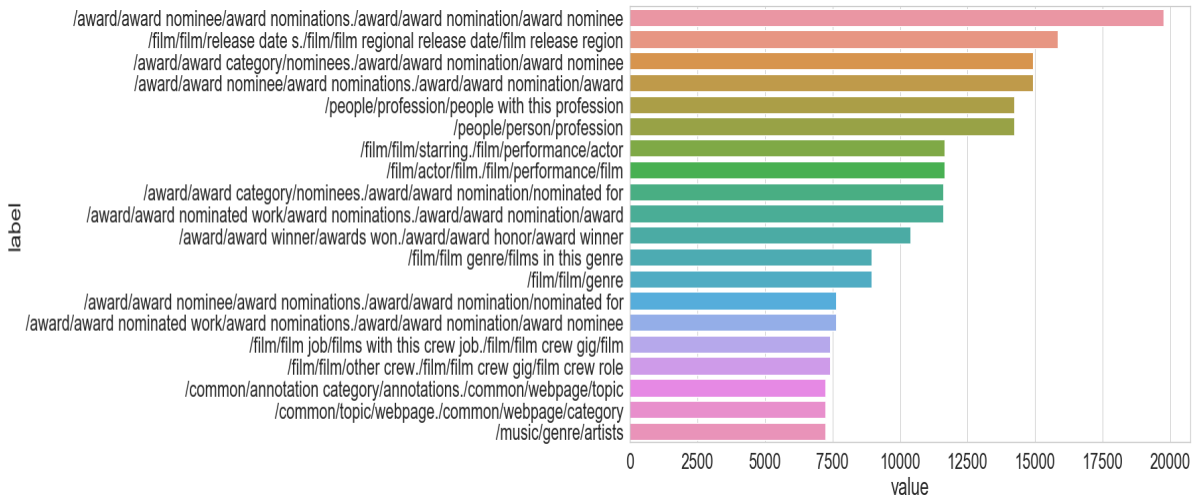
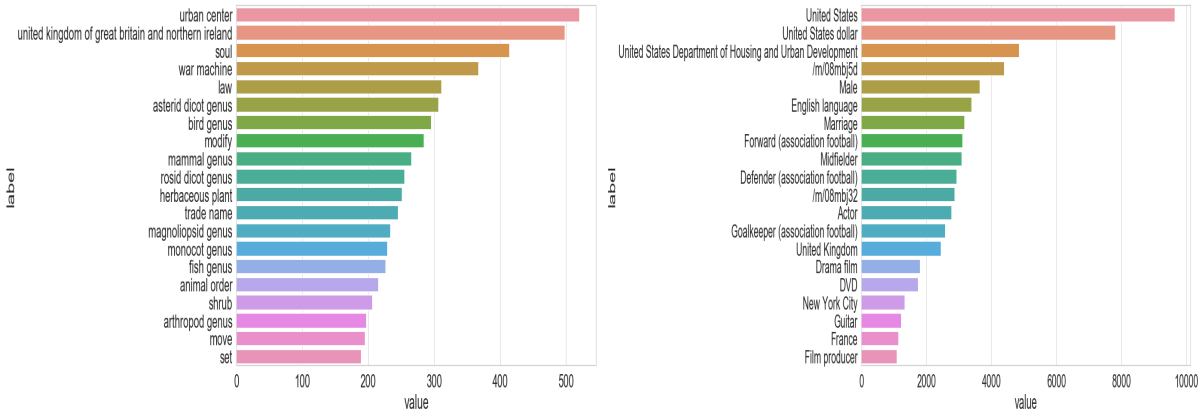



Figure 4.15: Histogram showing the number of times the 20 most frequent predicate labels occur in KG facts, in the FB15k link prediction dataset.




(a) WN18 (b) FB15k

Figure 4.16: Histogram showing the number of times the 20 most frequent object labels occur in KG facts, in the WN18 and FB15k link prediction datasets.

| | **WN18** | | | **FB15k** | | |
|---|---|---|---|---|---|---|
| | subject | predicate | object | subject | predicate | object |
| Count | 32,544 | 18 | 32,543 | 14,865 | 1,345 | 14,930 |
| Max | 520 | 37,221 | 520 | 4,381 | 19,764 | 9,645 |
| Min | 1 | 86 | 1 | 1 | 1 | 1 |
| Median | 3 | 3,243 | 3 | 27 | 26 | 23 |
| IQR | 2 | 6,191 | 2 | 32 | 166 | 30 |

Table 4.3: Statistics of the WN18 and FB15k link prediction datasets. We show counts of unique subject, predicate and object labels, as well as the maximum, minimum, median and interquartile range of label occurrences.

For WN18, it can be seen that predicates are skewed toward the relations "hyponym", "hypernym", and "derivationally related from", with a maximum of $37,221$ occurrences. FB15k predicates are somewhat more uniform.

WN18 and FB15k subjects are somewhat uniform aside from a small number of high occurring entities, with the median number of occurrences being 3 and 27 respectively, and with an IQR of 2 and 32 respectively. WN18 object occurrences are somewhat uniform. FB15k object occurrences are skewed, with the "United States" partaking in the highest number of triples. This is in comparison to a median object occurrence of 23 and an IQR of 30.

The WN18 dataset is split into training, validation and test sets of $141,442$ $(93.4\%)$, $5,000$ $(3.3\%)$ and $5,000$ $(3.3\%)$ triples respectively. The FB15k dataset is split into training, validation and test sets of $483,142$ $(81.6\%)$, $50,000$ $(8.4\%)$ and $59,071$ $(10.0\%)$ triples, respectively. These ratios are chosen to align with those used to train the baseline model.

### 4.2.2 Baseline algorithm

**Model summary.** HypER is a model that takes the convolution between a predicate-specific filter and subject vector, to generate a subject-predicate feature map. This map is flattened and passed through a nonlinearity, before an inner product is taken with an object vector. The result is a relational score which is passed through a logistic sigmoid to generate a probability of a potential relationship between pairs of entities.

**Binary cross-entropy loss.** The binary cross-entropy loss is used to train HypER. Like the NTN model, the input consists of a subject-predicate pair, and an object is presented as a target to complete the triple. A relational score is generated for each sample and passed through a

logistic sigmoid. Loss is generated by comparing the produced likelihood with the expected likelihood, 0 or 1. The sum of losses in a batch of samples is aggregated and backpropagated through the network for parameter update.

**Benchmark metrics.** The current suite of link prediction benchmark metrics includes Hit@10, Hit@3, Hit@1, Mean Rank, and Mean Reciprocal Rank. For these models, all objects in the KG are assigned a probability and then ranked in descending order. The Hit@X metrics comprise the relation prediction accuracy measures of the model, where X describes the lowest rank the predicted object can occupy. For example in Hit@10, only the top 10 objects are considered as accurate. The Mean Rank is the average predicted object rank and can be measured after any training epoch. The Mean Reciprocal Rank is the inverse of the Mean Rank.

### 4.2.3 HypER+

**Implementation.** Here we use the PyTorch framework to develop our model. The HypER model introduced by Balažević et al. [5] serves as a baseline. We apply batch normalisation to the hypernetwork input layer and thus introduce HypER+. Entity and relational embeddings are randomly initialised during model training. These embeddings are dynamically adjusted during the training process to generate latent representations specific to a KG. The models in this section were trained on Google Cloud Platform, on an N1 series instance with 8 CPU cores, 30GB RAM, 512GB SSD and an Nvidia Tesla P100 GPU. We train the respective models for 500 epoch, and evaluate them using the test sets of WN18 and FB15k.

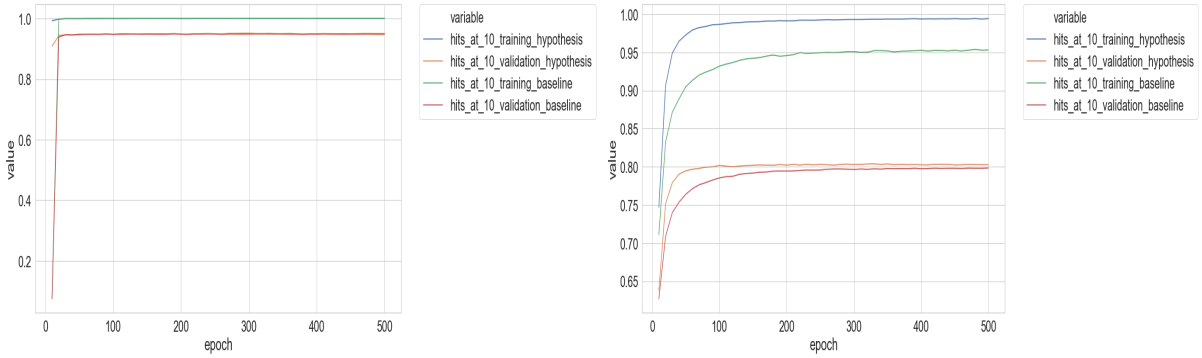
**Code to reproduce.** In the interest of reproducibility, all code needed to train and test the models in this section can be found at the following links.
Baseline HypER: https://github.com/xhosaBoy/HypER-baseline
HypER+: https://github.com/xhosaBoy/HypER-normalised-relations

#### Results

Results based on current standard link prediction metrics (Hit@10, Hit@3, Hit@1, Mean Rank and Mean Reciprocal Rank) for the Hyper+ model compared against the HypER model baseline, are presented in Figures 4.17 to 4.19, as well as Tables 4.4 and 4.5. T-SNE analysis [51] of HypER+ on the FB15k KG is presented in Figures 4.20 to 4.23.

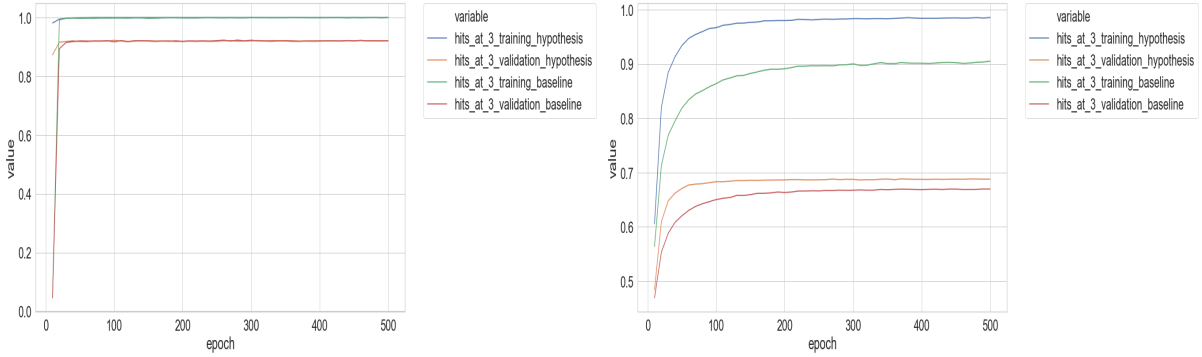
HypER+ achieves near-identical results with the HypER baseline on the WN18 KG across all metrics, and significantly outperforms the HypER baseline on the FB15k KG. The impact of

hypernetwork batch normalisation is pronounced, perhaps due to the upstream influence of predicate input covariate shift.



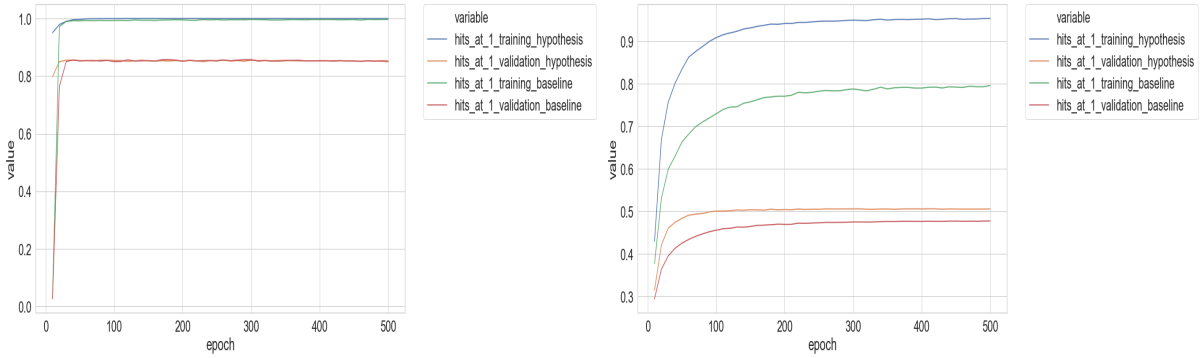

(a) WN18 (b) FB15k

Figure 4.17: Hit@10 vs training epoch. There is hardly any difference between the hypothesis and baseline models for the WN18 KG. The hypothesis model outperforms the baseline on the FB15k KG, although the validation difference is not as pronounced as the training difference.




(a) WN18 (b) FB15k

Figure 4.18: Hit@3 vs training epoch. The models have similar behaviour to the Hit@10 metric, and both have lower accuracy. There is however a larger performance difference between the hypothesis and baseline. This may be due to the more robust generalisation requirements with a smaller subset of acceptable answers, resulting in a more pronounced impact from covariate shift of latent predicate parameters.




(a) WN18 (b) FB15k

Figure 4.19: Hit@1 vs training epoch. The models have similar behaviour to the Hit@10 and Hit@3 metrics, and both have even lower accuracy. There is no discernible difference between the models on the WN18 KG, however there is once again a larger difference between them on the FB15k KG.

| **Model** | **H@10** | **H@3** | **H@1** | **MR** | **MRR** |
|---|---|---|---|---|---|
| TransE [13] | .892 | - | - | **251** | - |
| DistMult [93] | .936 | .914 | .728 | 902 | .822 |
| ComplEx [88] | .947 | .936 | .936 | - | .941 |
| Neural LP [94] | .945 | - | - | - | .940 |
| R-GCN [76] | **.964** | .929 | .697 | - | .819 |
| TorusE [25] | .954 | .950 | .943 | - | .947 |
| ConvE [19] | .956 | .946 | .935 | 374 | .943 |
| HypER [5] | .958 | **.955** | **.947** | 431 | **.951** |
| HypER+ (ours) | .957 | .954 | .946 | 565 | .950 |

Table 4.4: Relation prediction test results on WN18. The hypothesis is outperformed by the baseline HypER model across all metrics. It should however be noted that the difference is in the order of 0.1%, indicating almost identical performance. This is consistent with training and validation results shown earlier. Interestingly, the R-GCN model achieves the highest Hit@10 performance. This model uses the graph modelling SRL paradigm, as opposed to latent feature modelling, and does perform poorly across all other metrics. There may be opportunity in exploring this paradigm further.

| Model | H@10 | H@3 | H@1 | MR | MRR |
|---|---|---|---|---|---|
| TransE [13] | .471 | - | - | 125 | - |
| DistMult [93] | .824 | .733 | .546 | 97 | .654 |
| ComplEx [88] | .840 | .759 | .599 | - | .692 |
| Neural LP [94] | .837 | - | - | - | .760 |
| R-GCN [76] | .842 | .760 | .601 | - | .696 |
| TorusE [25] | .832 | .771 | .674 | - | .733 |
| ConvE [19] | .831 | .723 | .558 | 51 | .657 |
| HypER [5] | .885 | .829 | .734 | **44** | .790 |
| HypER+ (ours) | **.894** | **.856** | **.790** | 79 | **.829** |

Table 4.5: Relation prediction test results on FB15k. The hypothesis outperforms the baseline HypER model across all metrics, aside from Mean Rank. Unlike the performance difference on the WN18 KG, the difference here is by a number of percentage points, indicating a significant improvement over the baseline. The hypernetwork approach also significantly outperforms the R-GCN model.

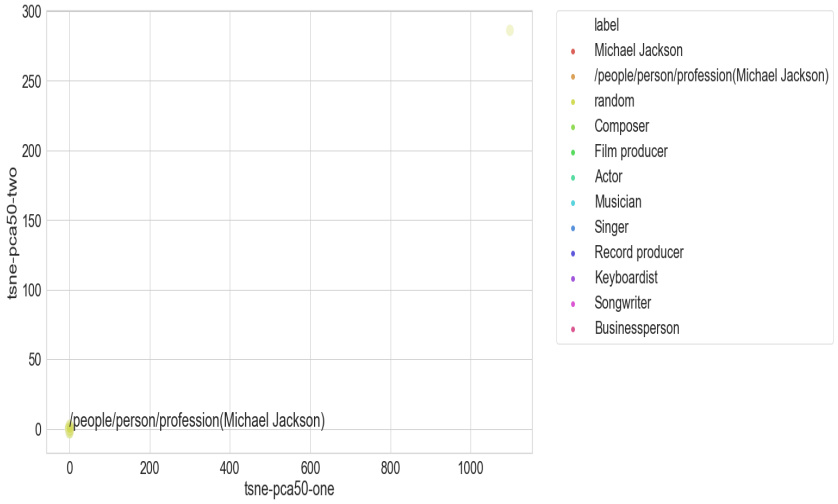



Figure 4.20: FB15k T-SNE: Triples about "Michael Jackson"'s (subject) "profession" (predicate), pre HypER+ training. The legend lists all professions (objects) known to have been performed by Michael Jackson. Most objects in the KG, regardless of type, begin clustered in the same embedding region on the bottom left. This includes the hidden subject-predicate vector, "/people/person/profession(Michael Jackson)", used to take an inner product with an object.

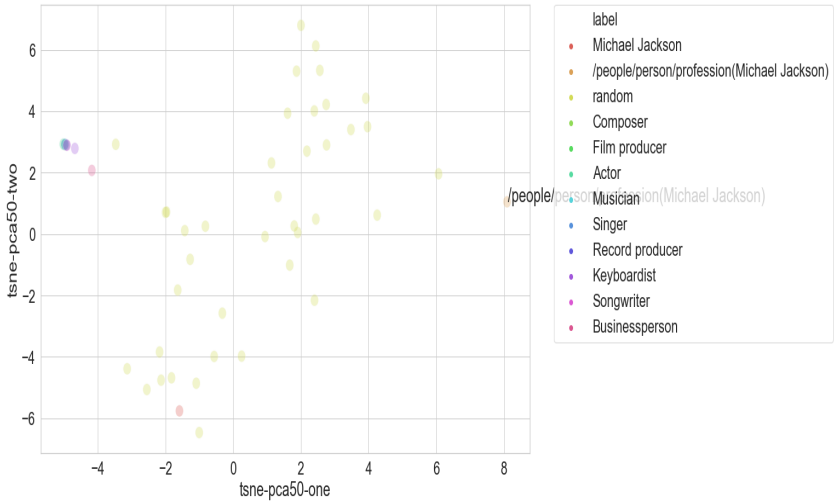



Figure 4.21: FB15k T-SNE: Triples about "Michael Jackson"'s (subject) profession (predicate), post HypER+ training. Professions Michael Jackson is known to have performed are now all clustered within the same embedding region. This indicates the model has built some conceptual understanding around the objects.

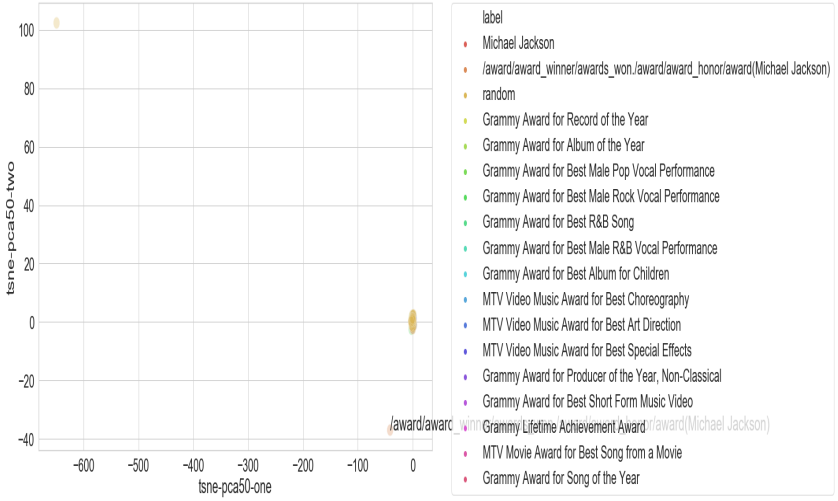



Figure 4.22: FB15k T-SNE: Triples about "Michael Jackson"'s (subject) "awards" (predicate), pre HypER+ training. The legend lists all awards (objects) known to have been won by Michael Jackson. Once again most objects in the KG, regardless of type, are clustered in the same embedding region.

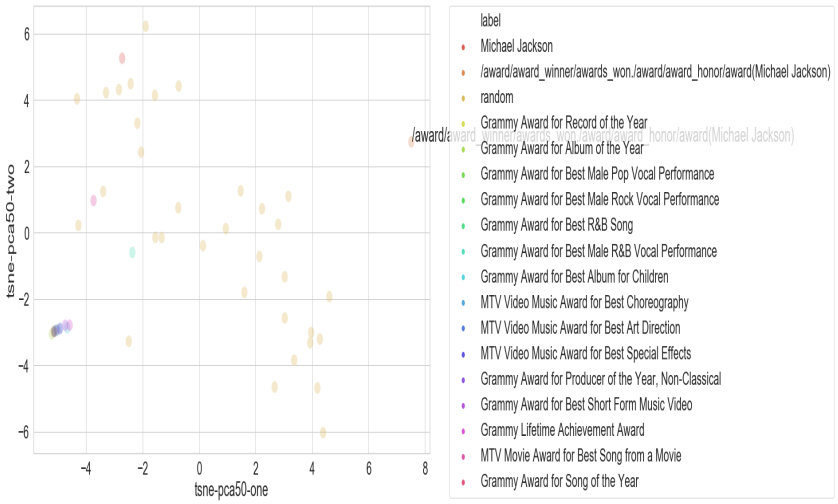



Figure 4.23: FB15k T-SNE: Triples about "Michael Jackson"'s (subject) "awards" (predicate), post HypER+ training. Awards Michael Jackson is known to have won are now somewhat clustered within the same embedding region. This indicates the model has built some conceptual understanding around the objects, and attempts to account for sub-domain differences, as well as conceptual differences between the awards. For example, the "MTV Movie Award for Best Song from a Movie" is located in a region relatively far from all the "Music"-related awards.

## 4.3 HypER+ with pre-trained word vectors

### 4.3.1 Datasets

For the third and final set of experiments we use the WN18RR and FB15k-237 benchmark datasets. WN18RR is a subset of WN18, created by Dettmers et al. [19] through removing the inverse relations from WN18. WN18RR contains $40,943$ entities, 11 relations and $40,943$ triples. FB15k-237 was created by Toutanova et al. [87], who noted that the validation and test sets of FB15k and WN18 contain the inverse of many relations present in the training set, making it easy for simple models to do well. FB15k-237 is a subset of FB15k with the inverse relations removed. It contains $14,541$ entities, 237 relations and $14,541$ triples. Visualisations of the respective KGs, as well as a sample of RDF triple encoded facts, are presented in Figures 4.26 to 4.29. KG summary statistics are presented in Figures 4.30 to 4.33, and Table 4.6.

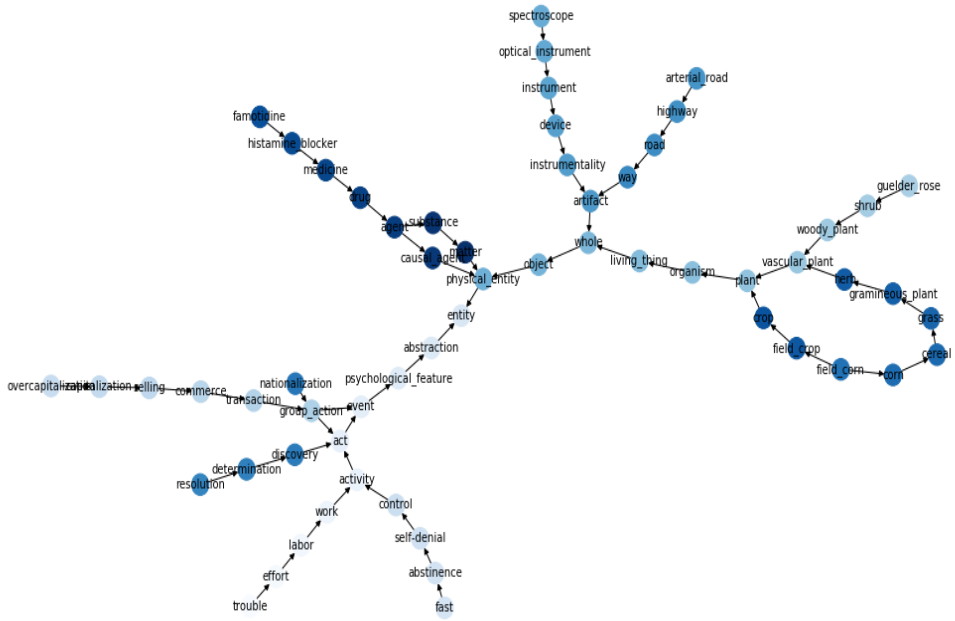



Figure 4.24: A subset of WN18RR facts structured as a KG. Entities are nodes, and relations are edges, where facts are encoded as RDF triples.

| id | subject | s_name | predicate | object | o_name |
|---|---|---|---|---|---|
| 1 | 00260881 | land reform | hypernym | 00260622 | reform |
| 2 | 01332730 | cover | derivationally related form | 03122748 | covering |
| 3 | 06066555 | phytology | derivationally related form | 00645415 | botanize |
| 4 | 09322930 | kamet | instance hypernym | 09360122 | mountain peak |
| 5 | 07193596 | question | derivationally related form | 00784342 | inquire |
| 6 | 01768969 | class arachnida | derivationally related form | 02636811 | spidery |
| 7 | 01455754 | run up | hypernym | 01974062 | raise |
| 8 | 07554856 | empathy | hypernym | 07553301 | sympathy |
| 9 | 00057306 | pullout | hypernym | 00056912 | retreat |
| 10 | 10341660 | variation | derivationally related form | 02661252 | vary |

Figure 4.25: A sample of RDF triples from the WN18RR KG.

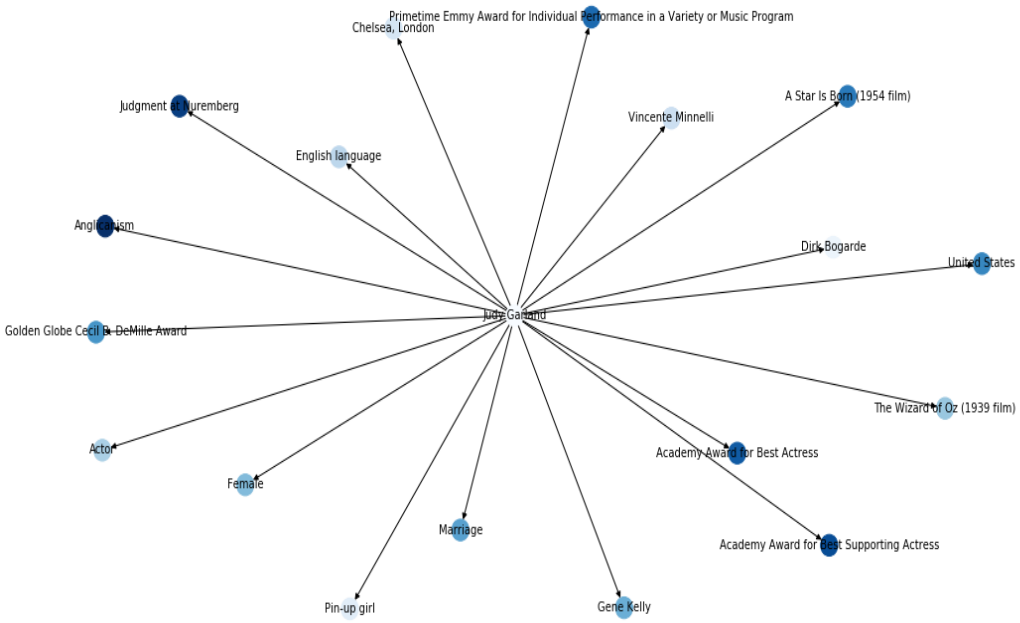
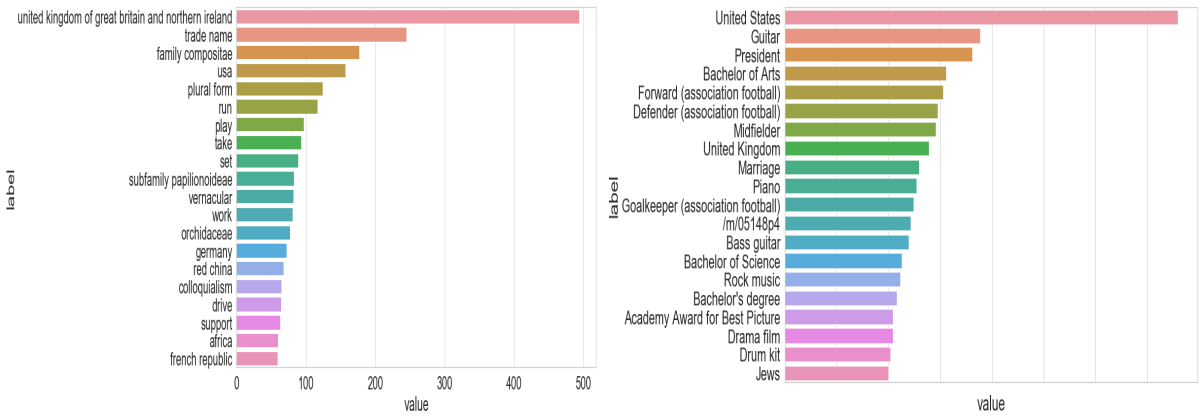



Figure 4.26: A subset of FB15k-237 facts structured as a KG. Entities are nodes, and relations are edges, where facts are encoded as RDF triples.

| id | subject | s_name | predicate | object | o_name |
|---|---|---|---|---|---|
| 43201 | /m/010cw1 | Hackensack, New Jersey | /common/topic/webpage./common/webpage/category | /m/08mbj5d | /m/08mbj5d |
| 225264 | /m/010cw1 | Hackensack, New Jersey | /location/capital of administrative division/capital of./location/administrative division capital relationship/administrative division | /m/0n5j_ | Bergen County, New Jersey |
| 178573 | /m/010cw1 | Hackensack, New Jersey | /location/hud county place/county | /m/0n5j_ | Bergen County, New Jersey |
| 245887 | /m/010cw1 | Hackensack, New Jersey | /location/location/time zones | /m/02hcv8 | Eastern Time Zone |
| 148957 | /m/010dft | Logan, Utah | /base/biblioness/bibs location/country | /m/09c7w0 | United States |
| 35710 | /m/010dft | Logan, Utah | /location/capital of administrative division/capital of./location/administrative division capital relationship/administrative division | /m/0jc6p | Cache County, Utah |
| 229655 | /m/010dft | Logan, Utah | /location/hud county place/county | /m/0jc6p | Cache County, Utah |
| 98108 | /m/010dft | Logan, Utah | /location/hud county place/place | /m/010dft | Logan, Utah |
| 946 | /m/010dft | Logan, Utah | /location/location/contains | /m/01nkcn | Utah State University |
| 250202 | /m/010dft | Logan, Utah | /location/location/time zones | /m/02hczc | Mountain Time Zone |

Figure 4.27: A sample of RDF triples from the FB15k-237 KG.

Once again FB15k-237 contains more entities, relations between entities, and facts, than WN18RR.




(a) WN18RR (b) FB15k-237

Figure 4.28: Histogram showing the number of times the 20 most frequent subject labels occur in KG facts, in the WN18RR and FB15k-237 link prediction datasets.

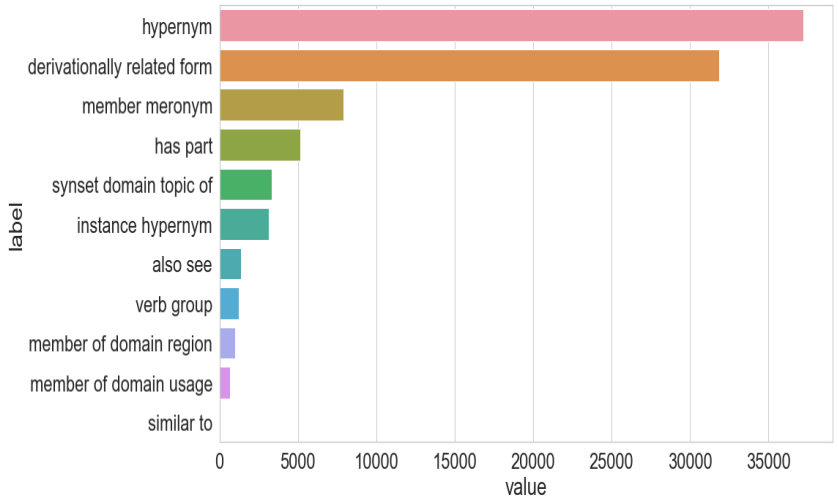



Figure 4.29: Histogram showing the number of times predicate labels occur in KG facts, in the WN18RR link prediction dataset.

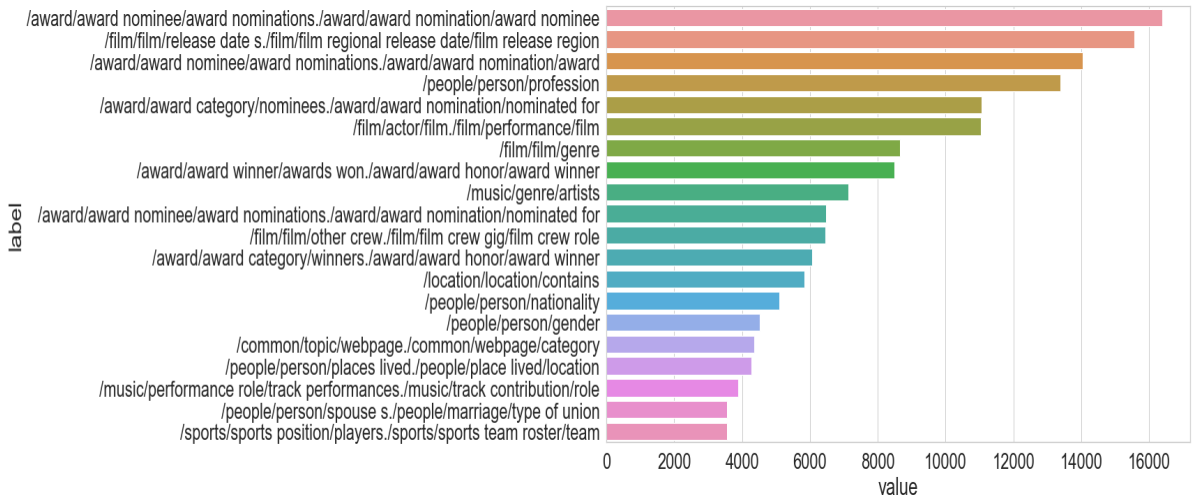
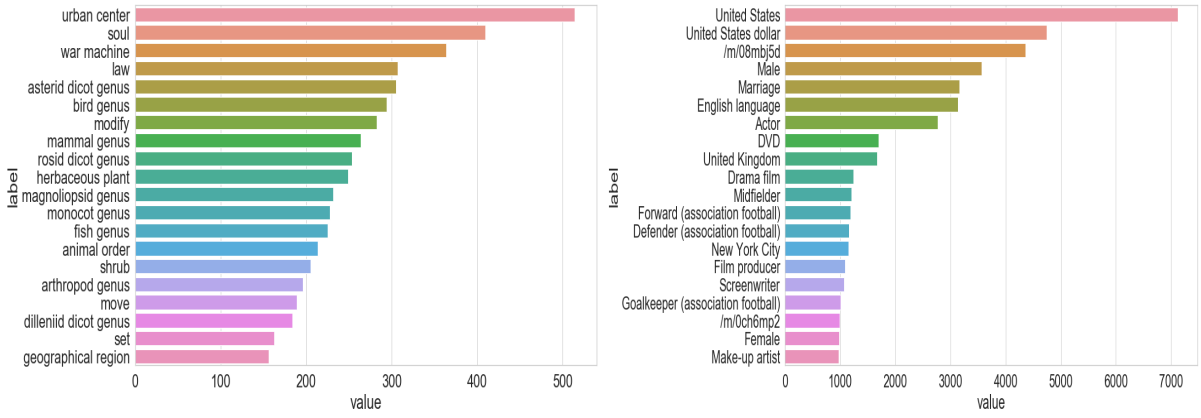



Figure 4.30: Histogram showing the number of times the 20 most frequent predicate labels occur in KG facts, in the FB15k-237 link prediction dataset.




(a) WN18RR (b) FB15k-237

Figure 4.31: Histogram showing the number of times the 20 most frequent object labels occur in KG facts, in the WN18RR and FB15k-237 link prediction datasets.

| | **WN18RR** | | | **FB15k-237** | | |
|---|---|---|---|---|---|---|
| | subject | predicate | object | subject | predicate | object |
| Count | 32,349 | 11 | 26,162 | 13,891 | 237 | 13,504 |
| Max | 494 | 37,221 | 514 | 1,518 | 16,391 | 7,124 |
| Min | 1 | 86 | 1 | 1 | 45 | 1 |
| Median | 2 | 3150 | 1 | 16 | 426 | 10 |
| IQR | 2 | 5434 | 2 | 20 | 819 | 16 |

Table 4.6: Statistics of the WN18RR and FB15k-237 link prediction datasets. We show counts of unique subject, predicate and object labels, as well as the maximum, minimum, median and interquartile range of label occurrences.

For WN18RR, it can be seen that predicates are skewed toward the relations "hypernym" and "derivationally related from", with a maximum of $37,221$ occurring, and an IQR of 5434 and 819 respectively. FB15k-237 predicates are skewed toward the film relation.

WN18RR and FB15k-237 subjects are somewhat uniform aside from a small number of high occurring entities, with the median number of occurrences being 2 and 16 respectively, and with an IQR of 2 and 20 respectively. WN18RR object occurrences are also somewhat uniform, while FB15k-237 object occurrences are skewed, with the "United States" partaking in the highest number of triples. This is in comparison to a median object occurrence of 1 and 10, and an IQR of 2 and 16 respectively.

The WN18RR dataset is split into training, validation and test sets of $86,835$ $(93.4\%)$, $3,034$ $(3.3\%)$ and $3,134$ $(3.4\%)$ triples respectively. The FB15k-237 dataset is split into training, validation and test sets of $272,115$ $(87.8\%)$, $17,535$ $(5.7\%)$ and $20,466$ $(6.6\%)$ triples respectively. These ratios are chosen to align with those used to train the baseline model.

### 4.3.2 Baseline algorithm

**Model summary.** HypER+, introduced in Section 4.2.3, is a model which compensates for covariate shift caused by hypernetworks, by applying batch normalisation to the hypernetwork input layer. The same algorithm as the HypER model is then used to produce a relational score, which is passed through a logistic sigmoid to generate a probability of a potential relationship between pairs of entities. HypER+ makes use of Xavier initialised entity and relation embeddings, and is trained using the binary cross-entropy loss.

### 4.3.3 GloVe word vector initialisation

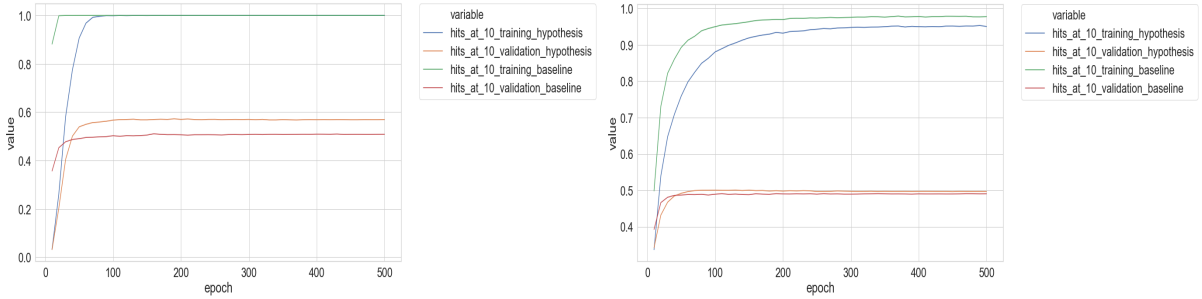
**Implementation.** We use the PyTorch framework to develop the HypER+ model with pre-trained embeddings, which is built on top of HypER+. Pre-trained GloVe word vectors replace Xavier initialised entity and relational embeddings for model training. These embeddings are dynamically adjusted during the training process to generate latent representations specific to the KG. The models in this section were trained on Google Cloud Platform, on an N1 series instance with 8 CPU cores, 30GB RAM, 512GB SSD and an Nvidia Tesla P100 GPU. We train the respective models for 500 epoch, and evaluate them using the test sets of WN18RR and FB15k-237.

**Code to reproduce.** In the interest of reproducibility, all code needed to train and test the models in this section can be found at the following links.
Baseline HypER+: https://github.com/xhosaBoy/HypER-normalised-relations
HypER+ with GloVe word vectors: https://github.com/xhosaBoy/HypER-pretrained-word-vectors

#### Results

Current standard link prediction metrics (Hit@10, Hit@3, Hit@1, Mean Rank and Mean Reciprocal Rank) are used to first compare the HypER+ model without pre-trained embeddings gainst the HypER+ model with pre-trained embeddings. The results are presented in Figures 4.32 and 4.33, as well as Tables 4.7 and 4.8. Qualitative results are presented in Figure 4.34 and Table 4.9.

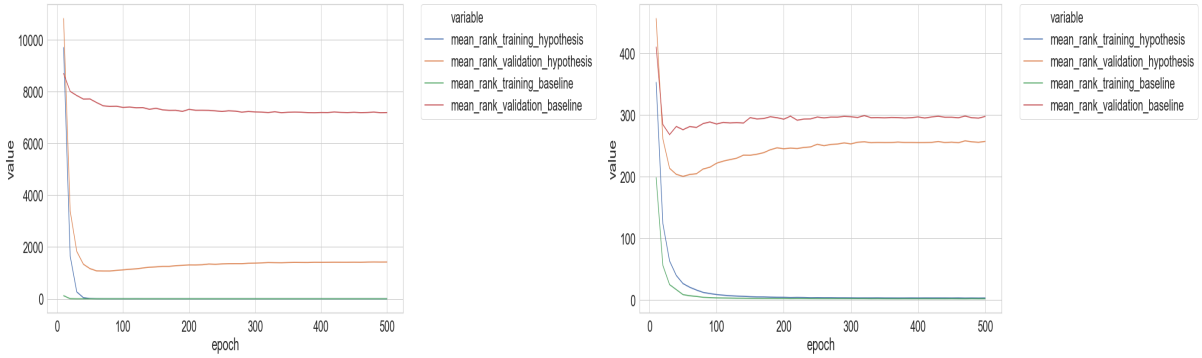
HypER+ with pre-trained embeddings gives state-of-the-art (SOTA) performance across both WN18RR and FB15k-237. Pre-trained word embeddings have a particularly pronounced impact on the WN18RR Hit@10 and Hit@3 metrics, perhaps due to the smaller number of samples in this KG. They however have limited impact on the Hit@1 metric, where HypER+ without pre-trained embeddings achieves SOTA performance.




(a) WN18RR: HypER+ vs HypER+ with pre-trained embeddings

(b) FB15k-237: HypER+ vs HypER+ with pre-trained embeddings

Figure 4.32: Hit@10 vs training epoch. The hypothesis significantly outperforms the baseline on the WN18RR KG. The hypothesis only marginally outperforms the baseline on the FB15k-237 KG. This suggests pre-trained embeddings have more of an impact on smaller KGs, where relations are likely to be more sparse, making it harder to build entity and relation conceptual understanding.




(a) WN18RR: HypER+ vs HypER+ with pre-trained embeddings

(b) FB15k-237: HypER+ vs HypER+ with pre-trained embeddings

Figure 4.33: Mean Rank vs training epoch. There is a stark difference in performance between the hypothesis and baseline on the WN18RR KG. Pre-trained embeddings are very effective at providing semantic information that can be used to produce good predictions for smaller, less complex KGs. The difference is less pronounced on the FB15k-237 KG, however is still significant.

| Model | H@10 | H@3 | H@1 | MR | MRR |
|---|---|---|---|---|---|
| DistMult [93] | .490 | .440 | .390 | 5110 | .430 |
| ComplEx [88] | .510 | .460 | .410 | 5261 | .440 |
| Neural LP [94] | - | - | - | - | - |
| ConvE [19] | .520 | .440 | .400 | 4187 | .430 |
| HypER [5] | .522 | .477 | .436 | 5798 | .465 |
| HypER+ (ours) | .519 | .479 | **.438** | 7061 | .466 |
| HypER+ with pre-trained embeddings (ours) | **.578** | **.493** | .435 | **1063** | **.480** |

Table 4.7: Relation prediction test results on WN18RR. HypER+ with pre-trained word embeddings significantly outperforms both the HypER as well as HypER+ models. HypER+ outperforms HypER+ with pre-trained embeddings on the Hit@1 metric, consistent with the expectation of diminished impact of pre-trained embeddings at high levels of accuracy constraints. It should be noted that there is a difference of 0.3% between all three models (HypER, HypER+ and HypER+ with pre-trained word embeddings), suggesting almost identical performance. The largest differences in performance occur at Hit@10 and Mean Rank, bearing the effectiveness of pre-trained embeddings at compensating for sparsity, however also highlighting how their significance diminishes at higher levels of accuracy.

| Model | H@10 | H@3 | H@1 | MR | MRR |
|---|---|---|---|---|---|
| DistMult [93] | .419 | .263 | .155 | 254 | .241 |
| ComplEx [88] | .428 | .275 | .158 | 339 | .247 |
| Neural LP [94] | .408 | - | - | - | .250 |
| ConvE [19] | .501 | .356 | .237 | 244 | .325 |
| HypER [5] | .520 | .376 | .252 | 250 | .341 |
| HypER+ (ours) | .516 | .368 | .245 | 268 | .335 |
| HypER+ with pre-trained embeddings (ours) | **.525** | **.379** | **.255** | **196** | **.345** |

Table 4.8: Relation prediction test results on FB15k-237. The HypER+ model with pre-trained embeddings achieves state-of-the-art performance across all metrics. Covariate shift seems to not be playing as meaningful a role, and pre-trained embeddings are providing a more meaningful contribution, as suggested by the HypER+ model's inferior performance to the HypER model. Strangely, this is inconsistent with training and validation set results, where HypER+ consistently outperforms HypER on FB15k-237. The fact that we use the quoted HypER test results, as opposed to reimplemented and verified test results, may explain this inconsistency. This inconsistency aside, pre-trained embeddings remain effective at improving relation prediction performance on the FB15k-237 KG.

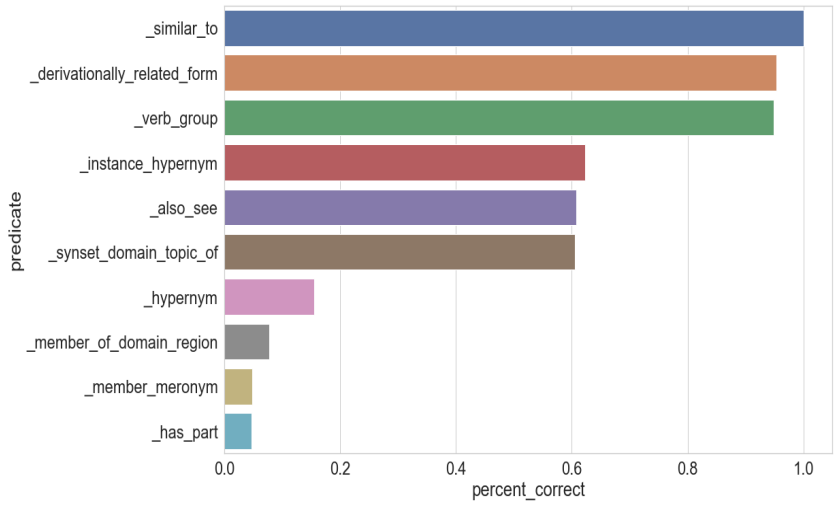



Figure 4.34: WN18RR Hit@1 predicate performance of the HypER+ model with pre-trained embeddings. The model performs well with synonym relation types, but performs poorly with compositional and hierarchical relations. This may be due to the inherent similarity and analogy in concepts, whereas compositions and hierarchies can be defined by strict rules. Perhaps, more simply, it could be due to the number of test set samples for each relation, where higher numbers increase the prediction error rate.

| **Subject** | **Predicate** | **Object Target** | **Object Prediction** |
|---|---|---|---|
| usa | has part | colorado | missouri river |
| spain | has part | cadiz | jerez de la frontera |
| kilobyte | has part | computer memory unit | word |
| electromagnetic spectrum | has part | actinic ray | radio spectrum |
| systema respiratorium | has part | respiratory tract | respiratory organ |
| respiratory organ | has part | nsa | defense advanced research projects agency |
| africa | has part | nigeria | senegal |
| antigen | has part | substance | epitope |
| amphitheatre | has part | theatre | tiered seat |
| indian ocean | has part | mauritius | antarctic ocean |

Table 4.9: Qualitative Hit@1 test results on WN18RR. The table presents a set of questions posed to the HypER+ with pre-trained embeddings model. "Object Target" is the expected answer, and "Object Prediction" is the answer given by the model. The model demonstrates basic conceptual understanding, never making mistakes that would be considered obvious by humans. We would expect reasonable knowledge discovery utility from the model, when used jointly with information retrieval for open-domain question answering [16].

## 4.4 Summary

**NTN with optimised training algorithm.** We attempted to improve the NTN model by applying training algorithm optimisations including early stopping, the Adam optimiser and hyperparameter random search. The results indicate that there are potential performance gains to be realised simply through using training algorithm techniques known to improve performance. In this instance we see an accuracy gain of 6.2% across the WordNet and Freebase datasets.

**HypER+.** We compensated for possible covariate shift introduced by hypernetworks. The latent representation distribution drift of relations is pronounced enough that we are able to improve the Hit@1 accuracy of the original HypER model on average by 2.7%, across the WN18 and FB15k KGs. We note graph modelling may be a worthwhile paradigm to explore for link prediction, given R-GCN's SOTA performance for Hit@10 accuracy on the WN18 KG. We also note the greater influence of relational normalisation on prediction performance for larger and more complex KGs.

**HypER+ with pre-trained word vectors.** Finally we extended HypER+ to make use of pre-trained GloVe word vectors. The semantic information inherent in these embeddings significantly improves relation prediction performance on sparse relational data. Their influence is somewhat reduced at higher levels of accuracy, where KG domain alignment becomes more important. Pre-trained word vectors also offer less utility for large and complex KGs, however still contribute an improvement to prediction. The combination of relation normalisation, and entity and relation embedding initialisation using pre-trained word vectors, both addresses the problems of data sparsity, as well as produces higher quality relational latent representations for large and complex datasets. As a result, HypER+ with pre-trained word vectors achieves SOTA performance on almost all standard link prediction benchmark metrics, across the challenging WN18RR and FB15k-237 KGs.

# Chapter 5

# Conclusions

Statistical relational learning (SRL) is a relatively young subfield in artificial intelligence (AI). It is concerned with reasoning, and makes use of the structure found in relational data, often represented as graphs, along with statistical attribute representations of data, common in many machine learning (ML) approaches [46]. Motivation in this line research is driven by the ambition of realising artificial general intelligence; machine systems with human-like reasoning capability. A potentially useful application is open-domain question answering (QA) [16], and another more speculative application is conversational AI [57, 6]. Recent progress has been realised with a focus on multi-hop reasoning [4, 50, 72]. This is a new SRL paradigm that attempts to directly imitate the step-by-step reasoning process of humans. A new dataset called HotpotQA [95] has also been proposed which aims to address some of the issues discussed in Chapter 1 with standard link prediction benchmark metrics. It is a more robust measure of assessing progress in knowledge graph question answering (KGQA), by enforcing explainable predictions. Latent feature modelling based link prediction may continue to play a role in KGQA, as a component, in a larger reasoning system.

Neural tensor factorisation shows great promise in further extending the performance of latent feature modelling based link prediction. The original insight to apply tensor decompositions to relational modelling, by Nickel et al. [64], was a watershed moment in SRL. It gave rise to a number of models [14, 40, 79, 88, 38, 19, 5] which ultimately lead to HypER+, seeing significant improvements in relation prediction performance in each subsequent model. There remains an open question as to the extent to which representation structure can continue to improve performance, for example HypER [5] extends the concept of multi-dimensional inputs proposed by ConvE [19], by generating relational matrices as opposed to relying on relational vectors. A natural extension to this approach is generating entity matrices, and then potentially

generating tensor representations for both. Such dense representations may better capture entity and relational concepts. The application of alternative entity-relation feature interaction operators is another natural extension. Others have demonstrated [88, 63, 19] that gains that can be realised using an alternative to the standard dot product when computing relation scores, relying on the Hermitian dot product, circular correlation and convolution respectively. An as yet unexplored avenue in latent feature modelling is the use of stochastic operators [44]. Such operators directly encode uncertainty, and may produce relational score probabilities with more appropriate confidence measures.

Deep learning methods have lead to the development of a number of successful approaches to neural tensor factorisation. Performance is now at 43.5% and 25.5% on the benchmark WN18RR and FB15k-237 KGs, respectively. In pursuit of improved link prediction performance, in this thesis we argued that (1) ML methods are a sensible approach to reasoning about knowledge expressed in natural language. Specifically, latent feature modelling in SRL enables the generation of a measure of confidence for relation prediction, expressing possible links in a KG using probabilities, and ranking those measures to derive the set of facts with the highest potential for being true. We also argued that (2) deep learning models can be used to improve latent feature modelling approaches to link prediction in SRL, and that (3) semantic information in pre-trained word vectors can be used to model richer entity-relation feature interactions.

To this end, in **Chapter 2** we conducted a survey of deep learning models, specifically the convolutional and recurrent models. We examined their respective properties, as well as representational modelling strengths. Convolutional networks (CNs) are particularly adept at efficiently encoding complex interactions of dense representations. This trait maps well to the dense entity and relation representations used in KGs, as well as the complex interactions they take part in. Recurrent networks (RNs) on the other hand have strengths in modelling sequential data, and are useful for generating word vectors. We then gave quick overviews of recursive networks (RCNs) and hypernetworks, along with their respective modelling strengths. We concluded the chapter with a discussion on ML model regularisation strategies, as well as a discussion on building language models.

In **Chapter 3** we reviewed an early successful attempt at link prediction using a neural compositional model: the recursive neural tensor network (NTN). The NTN was successful given the insight that compositionally of language gives rise to semantics. This insight inspired RCNs which are applied in the NTN model to enhance the entity-relational interaction representation. We reasoned that simple adjustments to the original NTN training algorithm, making use of ML techniques known to improve performance, should result in higher prediction accuracy.

We also analysed the HypER model, a convolutional model that uses a hypernetwork for relation parameter generation. This model was inspired by another convolutional model called ConvE, and both use a convolutional operator to increase the entity-relational interaction expressiveness of previous tensor factorisation models. We reasoned that hypernetworks may introduce covariate shift, and relation normalisation using batch normalisation may compensate for this. We also reasoned that pre-trained word vectors may enhance entity and relation representation modelling.

In **Chapter 4** we implemented an updated version of the NTN in TensorFlow. We applied early stopping, adaptive moment estimation (Adam) optimisation, and hyperparameter random search. This resulted in a performance improvement in link prediction accuracy, validating our reasoning that simply adjusting training algorithms with ML techniques known to improve performance, may result in increased relation prediction accuracy. We then adjusted the HypER model to compensate for covariate shift introduced by a hypernetwork, and thus introduced HypER+. Again we saw improvements in link prediction performance, validating our reasoning of distribution drift in relation latent representations experienced during training, resulting in suboptimal relation parameters. Finally we extended HypER+ by initialising entity and relation embeddings using pre-trained word vectors from the GloVe language model. We saw significant relation prediction improvement on the WN18RR KG, as well as relation prediction improvement on the FB15k-237 KG. Our model is also the state-of-the-art (SOTA) latent feature modelling approach to link prediction at the time of writing, validating our final reasoning of the utility of semantically rich word vectors in relation prediction.

Some progress in link prediction was recently achieved using the graph modelling [62] paradigm. A SOTA Hit@1 accuracy of 46% was achieved on FB15k-237 by Nathani et al. [60], compared to an accuracy of 25.5% achieved by HypER+ with pre-trained word vectors. This represents an increase of 20.5% [75], which is a staggering improvement. That level of performance was however not realised on WN18RR, with the model achieving a Hit@1 accuracy of 36.1%, compared to 43.5% achieved by HypER+ with pre-trained word vectors; a difference in performance of 7.4%. Pinter and Eisenstein [69], using another graph modelling approach, were able to achieve a SOTA performance on WN18RR of 45.37%; an increase in performance of 1.87% on HypER+ with pre-trained word vectors. Graph modelling approaches clearly demonstrate potential in pushing link prediction performance forward. The strength in graph modelling approaches could be exploiting a similar idea to the one used to construct the GloVe [67] language model, incorporating both local and global context. Inductive probabilistic logic programming (IPLP) also shows promise [21, 52], although research in this paradigm is a lot more sparse. It would seem graph modelling, as opposed to latent feature modelling, may provide the biggest contribution to SRL in the near-term.

# References

[1] (2020). J.A.R.V.I.S. https://ironman.fandom.com/wiki/J.A.R.V.I.S. [Ironman Wiki].

[2] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: A System for Large-Scale Machine Learning. In *USENIX Conference on Operating Systems Design and Implementation*, pages 265–283.

[3] Abujabal, A., Saha Roy, R., Yahya, M., and Weikum, G. (2018). Never-Ending Learning for Open-Domain Question Answering over Knowledge Bases. In *World Wide Web Conference*, pages 1053–1062.

[4] Asai, A., Hashimoto, K., Hajishirzi, H., Socher, R., and Xiong, C. (2020). Learning to Retrieve Reasoning Paths over Wikipedia Graph for Question Answering. In *International Conference on Learning Representations*.

[5] Balažević, I., Allen, C., and Hospedales, T. M. (2019). Hypernetwork Knowledge Graph Embeddings. In *International Conference on Artificial Neural Networks*, pages 553–565.

[6] Basu, K. (2019). Conversational AI: Open Domain Question Answering and Commonsense Reasoning. *Electronic Proceedings in Theoretical Computer Science*, 306:396–402.

[7] Bergstra, J. and Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

[8] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[9] Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., and Hellmann, S. (2009). DBpedia-A Crystallization Point for the Web of Data. *Journal of Web Semantics*, 7(3):154–165.

[10] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

[11] Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *ACM SIGMOD International Conference on Management of Data*, pages 1247–1250.

[12] Bordes, A., Glorot, X., Weston, J., and Bengio, Y. (2014). A Semantic Matching Energy Function for Learning with Multi-Relational Data. *Machine Learning*, 94(2):233–259.

[13] Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013). Translating Embeddings for Modeling Multi-Relational Data. In *Advances in Neural Information Processing Systems*, pages 2787–2795.

[14] Bordes, A., Weston, J., Collobert, R., and Bengio, Y. (2011). Learning Structured Embeddings of Knowledge Bases. In *AAAI Conference on Artificial Intelligence*, pages 301–306.

[15] Chang, K.-W., Yih, W.-t., Yang, B., and Meek, C. (2014). Typed Tensor Decomposition of Knowledge Bases for Relation Extraction. In *Conference on Empirical Methods in Natural Language Processing*, pages 1568–1579.

[16] Chen, D., Fisch, A., Weston, J., and Bordes, A. (2017). Reading Wikipedia to Answer Open-Domain Questions. In *Association for Computational Linguistics*, pages 1870–1879.

[17] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734.

[18] Chomsky, N. (1955). Logical Syntax and Semantics: Their Linguistic Relevance. *Language*, 31(1):36–45.

[19] Dettmers, T., Minervini, P., Stenetorp, P., and Riedel, S. (2018). Convolutional 2D Knowledge Graph Embeddings. In *AAAI Conference on Artificial Intelligence*, pages 1811–1818.

[20] Diefenbach, D., Singh, K., and Maret, P. (2018). WDAqua-Core1: A Question Answering Service for RDF Knowledge Bases. In *The Web Conference*, pages 1087–1091.

[21] Dong, H., Mao, J., Lin, T., Wang, C., Li, L., and Zhou, D. (2019). Neural Logic Machines. In *International Conference on Learning Representations*.

[22] Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., and Zhang, W. (2014). Knowledge Vault: A Web-Scale Approach to Probabilistic Knowledge Fusion. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 601–610.

[23] Doss, D., LeNail, A., and Liu, C. (2015). Reimplementing Neural Tensor Networks for Knowledge Base Completion in the TensorFlow Framework. https://github.com/dddoss/tensorflow-socher-ntn/blob/master/notes/paper.pdf.

[24] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

[25] Ebisu, T. and Ichise, R. (2018). Toruse: Knowledge Graph Embedding on a Lie Group. In *AAAI Conference on Artificial Intelligence*, pages 1819–1826.

[26] Glorot, X. and Bengio, Y. (2010). Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *International Semantic Web Conference*, pages 249–256.

[27] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[28] Gouws, S. (2017). Deep Learning Indaba Practicals. https://github.com/deep-learning-indaba/practicals2017.

[29] Gouws, S. (2018). Deep Learning Indaba Tutorials. https://github.com/deep-learning-indaba/indaba-2018.

[30] Ha, D., Dai, A. M., and Le, Q. V. (2017). Hypernetworks. In *International Conference on Learning Representations*.

[31] Hakimov, S., Jebbara, S., and Cimiano, P. (2019). Evaluating Architectural Choices for Deep Learning Approaches for Question Answering over Knowledge Bases. In *International Conference on Semantic Computing*, pages 110–113.

[32] Harshman, R. A. (1978). Models for Analysis of Asymmetrical Relationships Among N Objects or Stimuli. In *Psychometric Society and the Society of Mathematical Psychology*.

[33] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media.

[34] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification. In *IEEE International Conference on Computer Vision*, pages 1026–1034.

[35] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors. *arXiv preprint arXiv:1207.0580*.

[36] Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

[37] Hoerl, A. E. and Kennard, R. W. (1970). Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, 12(1):55–67.

[38] Hohenecker, P. and Lukasiewicz, T. (2020). Ontology Reasoning with Deep Neural Networks. *Journal of Artificial Intelligence Research*, 68:503–540.

[39] Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, pages 448–456.

[40] Jenatton, R., Roux, N. L., Bordes, A., and Obozinski, G. R. (2012). A Latent Factor Model for Highly Multi-Relational Data. In *Advances in Neural Information Processing Systems*, pages 3167–3175.

[41] Kang, U., Papalexakis, E., Harpale, A., and Faloutsos, C. (2012). Gigatensor: Scaling Tensor Analysis up by 100 Times-Algorithms and Discoveries. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 316–324.

[42] Kazemi, S. M. and Poole, D. (2018). Simple Embedding for Link Prediction in Knowledge Graphs. In *Advances in Neural Information Processing Systems*, pages 4284–4295.

[43] Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

[44] Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In Bengio, Y. and LeCun, Y., editors, *International Conference on Learning Representations*.

[45] Kolda, T. G. and Bader, B. W. (2009). Tensor Decompositions and Applications. *SIAM Review*, 51(3):455–500.

[46] Koller, D., Friedman, N., Džeroski, S., Sutton, C., McCallum, A., Pfeffer, A., Abbeel, P., Wong, M.-F., Heckerman, D., Meek, C., et al. (2007). *Introduction to Statistical Relational Learning*. MIT Press.

[47] Kristiadi, A., Khan, M. A., Lukovnikov, D., Lehmann, J., and Fischer, A. (2019). Incorporating Literals into Knowledge Graph Embeddings. In *International Semantic Web Conference*, pages 347–363.

[48] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105.

[49] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[50] Lin, X. V., Socher, R., and Xiong, C. (2018). Multi-Hop Knowledge Graph Reasoning with Reward Shaping. In *Conference on Empirical Methods in Natural Language Processing*.

[51] Maaten, L. v. d. and Hinton, G. (2008). Visualizing Data using T-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

[52] Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and De Raedt, L. (2018). DeepProbLog: Neural Probabilistic Logic Programming. In *Advances in Neural Information Processing Systems*, pages 3749–3759.

[53] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient Estimation of Word Representations in Vector Space. In *International Conference on Learning Representations*.

[54] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

[55] Miller, G. A. (1995). WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41.

[56] Minervini, P., Bošnjak, M., Rocktäschel, T., Riedel, S., and Grefenstette, E. (2019). Differentiable Reasoning on Large Knowledge Bases and Natural Language. *arXiv preprint arXiv:1912.10824*.

[57] Moon, S., Shah, P., Kumar, A., and Subba, R. (2019). OpenDialKG: Explainable Conversational Reasoning with Attention-Based Walks over Knowledge Graphs. In *Association for Computational Linguistics*, pages 845–854.

[58] Morin, F. and Bengio, Y. (2005). Hierarchical Probabilistic Neural Network Language Model. In *AISTATS*, pages 246–252.

[59] Murphy, K. P. (2012). *Machine Learning: a Probabilistic Perspective*. MIT Press.

[60] Nathani, D., Chauhan, J., Sharma, C., and Kaul, M. (2019). Learning Attention-based Embeddings for Relation Prediction in Knowledge Graphs. In *Association for Computational Linguistics*, pages 4710–4723.

[61] Nguyen, D. Q., Nguyen, T. D., Nguyen, D. Q., and Phung, D. Q. (2018). A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network. In *Conference of the North American chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 327–333.

[62] Nickel, M., Murphy, K., Tresp, V., and Gabrilovich, E. (2016a). A Review of Relational Machine Learning for Knowledge Graphs. *Proceedings of the IEEE*, 104(1):11–33.

[63] Nickel, M., Rosasco, L., and Poggio, T. (2016b). Holographic Embeddings of Knowledge Graphs. In *AAAI Conference on Artificial Intelligence*, pages 1955–1961.

[64] Nickel, M., Tresp, V., and Kriegel, H.-P. (2011). A Three-Way Model for Collective Learning on Multi-Relational Data. In *International Conference on Machine Learning*, pages 809–816.

[65] Nickel, M., Tresp, V., and Kriegel, H.-P. (2012). Factorizing Yago: Scalable Machine Learning for Linked Data. In *International Conference on World Wide Web*, pages 271–280.

[66] Niepert, M. (2016). Discriminative Gaifman Models. In *Advances in Neural Information Processing Systems*, pages 3405–3413.

[67] Pennington, J., Socher, R., and Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. In *Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543.

[68] Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep Contextualized Word Representations. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2227–2237.

[69] Pinter, Y. and Eisenstein, J. (2018). Predicting Semantic Relations using Global Graph Properties. In *Conference on Empirical Methods in Natural Language Processing*, pages 1741–1751.

[70] Pollack, J. B. (1990). Recursive Distributed Representations. *Artificial Intelligence*, 46(1-2):77–105.

[71] Prechelt, L. (1998). Early Stopping-But When? In *Neural Networks: Tricks of the Trade*, pages 55–69. Springer.

[72] Qi, P., Lin, X., Mehr, L., Wang, Z., and Manning, C. D. (2019). Answering Complex Open-domain Questions Through Iterative Query Generation. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

[73] Reiter, R. (1980). A Logic for Default Reasoning. *Artificial Intelligence*, 13(1-2):81–132.

[74] Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65(6):386.

[75] Ruder, S. (2020). Natural Language Processing Progress - Relation Prediction. http://nlpprogress.com/english/relation_prediction.html.

[76] Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M. (2018). Modeling Relational Data with Graph Convolutional Networks. In *European Semantic Web Conference*, pages 593–607.

[77] Shu, H. (2003). Chinese Writing System and Learning to Read. *International Journal of Psychology*, 38(5):274–285.

[78] Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*.

[79] Socher, R., Chen, D., Manning, C. D., and Ng, A. (2013). Reasoning with Neural Tensor Networks for Knowledge Base Completion. In *Advances in Neural Information Processing Systems*, pages 926–934.

[80] Socher, R., Huval, B., Manning, C. D., and Ng, A. Y. (2012). Semantic Compositionality through Recursive Matrix-Vector Spaces. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1201–1211.

[81] Speichert, S. and Belle, V. (2019). Learning Probabilistic Logic Programs over Continuous Data. In *Inductive Logic Programming*, pages 129–144.

[82] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[83] Staab, S. and Studer, R. (2010). *Handbook on Ontologies*. Springer Science & Business Media.

[84] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.

[85] Tibshirani, R. (1996). Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.

[86] Tieleman, T. and Hinton, G. (2012). Lecture 6.5-RMSProp: Divide the Gradient by a Running Average of its Recent Magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2):26–31.

[87] Toutanova, K. and Chen, D. (2015). Observed Versus Latent Features for Knowledge Base and Text Inference. In *Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66.

[88] Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., and Bouchard, G. (2016). Complex Embeddings for Simple Link Prediction. In *International Conference on Machine Learning*, pages 2071–2080.

[89] Turian, J., Ratinov, L., and Bengio, Y. (2010). Word Representations: A Simple and General Method for Semi-Supervised Learning. In *Association for Computational Linguistics*, pages 384–394.

[90] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. U., and Polosukhin, I. (2017). Attention is All you Need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

[91] Werbos, P. J. (1988). Generalization of Backpropagation with Application to a Recurrent Gas Market Model. *Neural Networks*, 1(4):339–356.

[92] Wu, X., Shi, B., Dong, Y., Huang, C., and Chawla, N. (2018). Neural Tensor Factorization. *arXiv preprint arXiv:1802.04416*.

[93] Yang, B., Yih, S. W.-t., He, X., Gao, J., and Deng, L. (2015). Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *International Conference on Learning Representations*.

[94] Yang, F., Yang, Z., and Cohen, W. W. (2017). Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In *Advances in Neural Information Processing Systems*, pages 2319–2328.

[95] Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. (2018). HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Conference on Empirical Methods in Natural Language Processing*.