Thesis Title

Improving Training of Deep Neural Network Sequence Models

Farhana Ferdousi Liza

A thesis presented for the degree of
Doctor of Philosophy

School of Computing
University of Kent
United Kingdom
August 2019

Abstract

Sequence models, in particular, language models are fundamental building blocks of downstream applications including speech recognition, speech synthesis, information retrieval, machine translation, and question answering systems. Neural network language models are effective in generalising (i.e. perform efficiently with the data sparsity problem) compared to traditional N-grams models. However, neural network language models have several fundamental problems — the training of neural network language models is computationally inefficient and analysing the trained models is difficult. In this thesis, improvement techniques to reduce the computational complexity and an extensive analysis of the learned models are presented.

To reduce the computational complexity we have focused on the main computational bottleneck of neural training which is the softmax operation. Among different softmax approximation techniques, Noise Contrastive Estimation (NCE) is seen as a method that often does not work well with deep neural models for language modelling. A thorough investigation was done to find out the appropriate and novel integration mechanism of NCE with deep neural networks. We have also explained why the proposed specific hyperparameter settings could have an impact on the integration.

Existing analysis techniques are not sufficient to explain the training and learned models. Established wisdom on learning theory cannot explain the generalisation of over-parametrised deep neural networks. Therefore, we have proposed methods and analysis techniques to understand the generalisation and explain the regularisation. Furthermore, we have explained the impact of the stacked layers in deep neural networks.

The presented techniques have made the neural language models more accurate and computationally efficient. The empirical analysis techniques have helped us understand the model learning and improved our understanding of the generalisa-

tion and regularisation. The conducted experiments were based on publicly available benchmark datasets and standard evaluation frameworks.

Acknowledgements

I would like to thank my supervisors, Dr. Marek Grzes and Professor Alex Alves Freitas, for the patient guidance, encouragement and advice they have provided during my PhD study. I have worked more closely with Marek and I have been extremely lucky to have a supervisor who cared so much about my work, my well-being and who responded to my questions and queries so promptly. His knowledge, diligence, commitment, dedication and attention for details have always impressed and inspired me. I want to thank the thesis examiners Professor Howard Bowman and Dr. Dimitar Kazakov. Their constructive feedbacks have made this thesis a better piece of work.

I must express my gratitude to Md Shoaib Ahmed, my husband, for his continued support and encouragement. I was continually amazed by his willingness to proofread countless pages, and by his patience when I experienced all of the ups and downs of my research. I also would like to thank my father, mother and siblings who have always encouraged me to pursue my PhD and told me to give the PhD the up-most priority.

I thank Professor Sally Fincher for asking all those general but difficult questions that chase me whole PhD study and helped to structure my thinking process. I thank Professor Peter Rodger for approving my fund application for travels and summer school, and I must admit, his prompt response has always helped me to be less stressed and more productive. I thank the selection committee of the School of computing who have sponsored my PhD study through the 'School of Computing JILP Endowment Scholarship', without which this research would be impossible.

During my PhD study, I have received great support from the information services, school of computing's administrative staff members and IT technical support. I want to thank Angela Doe (ret.), Amanda Ollier, Sonnary Dearden, Julie Teulings and Angie Allen for their great administrative support. I want to thank the Graduate School for providing the skill trainings, those were really useful. I

would like to thank other PhD students, especially Fabio Fabris and Caroline Rizzi Raymundo for their friendly interaction, which made the PhD study less isolated. I would also like to thank Lee Harris for his proofread of my papers.

The PhD thesis has improved in quality by the critical reviews from the anonymous reviewers and I would like to thank them all, I also thank SPiCe competition organiser for setting the challenging datasets and organising such a research-oriented competition. I must also thank the members of my supervisory panel: Professor Sally Fincher and Dr. Colin Johnson, who have also contributed to my research with insightful comments.

Contents

List of Figures

List of Tables

1

INTRODUCTION

The aim of this thesis is to advance the understanding and improve the performance of neural network-based sequence models. As a way of illustrating some of the key sequence models that are used for various natural language processing tasks, a specific challenging sequence modelling task (i.e. language modelling) is used in this thesis extensively.

Language Modelling (LM) is the attempt to characterise, capture and exploit regularities in natural language. In Machine Learning (ML), the task of language modelling is central to both Natural Language Processing (NLP) and Language Understanding (Rosenfeld 2000). Concretely, language models assign probabilities to sequences of words or assign a probability to a word after seeing a given sequence of words. As a language model places a probability distribution over a sequence of words, language models capture the syntactic and semantic regularities of the language and extract a fair amount of information about the knowledge in the corpus (Mikolov, Yih & Zweig 2013, Mikolov, Sutskever, Chen, Corrado & Dean 2013, Józefowicz et al. 2016).

Moreover, language models are the fundamental building blocks for a large range of ML and NLP models that process natural language with incomplete knowledge. Overall, a model designed for a particular NLP task that requires language understanding uses language models. Therefore, a language model is a crucial part that determines the performance of other ML and NLP tasks, such as speech recognition (Mikolov et al. 2010, Prabhavalkar et al. 2017), machine translation (Luong et al. 2015, Gulcehre et al. 2017), text summarisation (Filippova et al. 2015, Gambhir & Gupta 2017), question answering (Wang et al. 2017), semantic error detection (Rei & Yannakoudakis 2017, Spithourakis, Augenstein & Riedel 2016), and fact-checking (Rashkin et al. 2017). Language models have also been used to transfer learning from one domain to another. Language models facilitate transfer learning, which is an efficient way to improve numerous ML systems (Erhan et al. 2010, Dai & Le 2015, Radford et al. 2018), as language models provide a systematic way to use unlabelled data through the learned distributed representation for words (i.e. word embeddings). The importance of a language model can be observed by removing the language model

from a downstream application. For example, Rosenfeld (2000) observed a drastic effect by removing the language model from a speech recognition system. Language models have also been used in clinical decision-making systems. For example, Fischer & Bauckhage (2018) showed how to use language models to analyse radiological reports, De Vine et al. (2014) showed how to extract clinical information using language modelling, and Spithourakis, Petersen & Riedel (2016) showed how to use language modelling for clinical text prediction. As language models improve many ML and NLP systems that have both economic and societal impact, in this thesis we investigate the research gaps and attempt to provide solutions that improve language modelling.

The complexity of language modelling or NLP tasks in general, might not be obvious. For example, in 1954, the Georgetown experiment involved fully automatic translation of more than sixty Russian sentences into English (Edwards 2016). The researchers in that experiment estimated that within three or five years, machine translation would be a solved problem (Hutchins 1995). However, real progress was much slower. In fact, in 1966 the Automatic Language Processing Advisory Committee (ALPAC) reported that ten-year-long research had failed to fulfil the expectations (Pierce & Carroll 1966). Consequently, funding for machine translation was dramatically reduced. Little further research in machine translation was conducted until the development of the first Statistical Machine Translation (SMT) system in the late 1980s. The basic idea of an SMT is to search for the most probable translation $\tilde{T}$ for a given source sentence S using the Bayes theorem and decompose the SMT into two sub-problems consisting of translation modelling (TM) and language modelling (LM) as follows:

An anecdote.

$$\tilde{T} = \operatorname{argmax}_T P(T|S)$$

$$\tilde{T} = \operatorname{argmax}_T \frac{P(S|T)P(T)}{P(S)}$$

$$\tilde{T} = \operatorname{argmax}_T \underbrace{P(S|T)}_{\text{TM}} \underbrace{P(T)}_{\text{LM}}$$

Although machine translation has improved since, it is still an active research area and language models help to improve the performance of machine translation systems (Brants et al. 2007, Luong et al. 2015).

In addition to the benefit that language modelling brings to a practical NLP task, language models play a role in facilitating natural language understanding by

a machine (i.e. model) (Dong et al. 2019). Natural language understanding is important for developing intelligent machine learning models. In 1950, Alan Turing (Turing 1950) proposed the Turing test as a criterion of intelligence. His works also show that any computable problem can be computed by a Universal Turing Machine, which means that if human intelligence can be defined and represented (e.g. learning representation) by some algorithm, a Turing Machine is powerful enough to compute it. In fact, the success of machine learning algorithms generally depends on a model's capacity on learning representation (Bengio et al. 2013) and language modelling helps in learning data representation. Computers today are Turing-complete, i.e., can compute any computable algorithm. Thus, the main problem is to find the configuration of a machine so that it would produce the desired behaviours that humans consider intelligent. These configurations are discovered by the hyperparameter tuning of models.

An immediate attempt to make a machine (e.g. NLP model) intelligent is very difficult, however, we can think of several ways that would lead us towards intelligent machines. For NLP models, a reasonable way would be to mimic the learning processes of humans. Natural language is an immensely complicated phenomenon. It is a great mystery about how children learn their first language without learning any specific grammatical rules. In general, a language is learned by observing the real world, recognising its regularities, and mapping acoustic and visual signals to higher-level representations in the brain.

To make a model intelligent, we might need to mimic the whole human learning process which is difficult to define and represent at the moment. The main difficulty in defining and representing the human learning process is that we do not know how real-life learning happens. The learning problem is complex and many open questions still need to be answered. For example, we need to have a better understanding of human learning and then get the best formal representation of a language. Whereas formal language is a set of strings of symbols together with a set of rules that are specific to it, such as regular grammar or context-free grammar or programming languages, natural language does not have such well-defined rules. Natural languages have emerged and therefore no specific rules can describe or specify them fully. This difficulty makes learning complex as we cannot use a formal language to define a natural language. Another question, to mimic the real-world situation, how to learn language jointly with other modalities, such as acoustic, image, and video. Moreover, when taking a data-driven approach to training a model, we do not have a precise understanding of the data requirement that must be processed during the learning of a model, and this also raises several questions about

the model size.

A current approach to language modelling is to use as much data as possible to beat the state-of-the-art results. However, when we have a limited dataset, this approach fails. Furthermore, the size of a dataset can be small, medium or large depending on the model type. The same dataset can be medium-sized for one model type but can be large-sized for another model type.

Model training time and model interpretability raise challenging research questions. While some models are fast to train, other models can take much longer time to train. Improving the training time is thus essential for slower models to be practically usable. While some models are interpretable, others are essentially black-box models. To make a model intelligent, we often want all good qualities from a single model, which is itself a challenging research direction.

It might be too ambitious to attempt to solve all these research problems together and to expect too much from models or techniques that even do not allow the existence of a solution (an example might be the well-known limitations of finite state machines to represent efficiently large patterns or neural network models to be interpretable). While it is likely that attempts to build language models that can understand text in the same way as humans do just by reading huge quantities of text data is unrealistically hard (as humans would probably fail in such task themselves), language models estimated from huge amounts of data are very interesting due to their practical usage in a wide variety of successful applications which have economic and societal impact.

1.1 The Focus of This Research

In this PhD study, we are concentrating on the fundamental problems of language modelling. A Statistical Language Model (SLM) is the state-of-the-art language model for downstream applications. The main reason for Statistical Language Models (SLMs) being the state-of-the-art language models for downstream applications is that these models are easy to train (e.g. only one hyperparameter), interpretable and fast in training. However, SLMs work in discrete word indices space (Schwenk & Gauvain 2002) and have a sparsity problem (Allison et al. 2006) that makes them inefficient in learning representation and generalising as there is no obvious relation between the word indices. Moreover, SLMs are based on the maximum likelihood estimation (MLE) and the assumption is that anything unseen in a training corpus cannot

happen in the corresponding test corpus. This assumption is often wrong with a limited training corpus (i.e. incomplete knowledge) and gives rise to a sparsity problem. A Neural Language Model (NLM) uses a distributed continuous representation and is less prone to the data sparsity problem compared to a SLM (Kim et al. 2016). Words are represented as vectors (i.e. word embeddings) and are fed as inputs to a NLM to enhance the learning representation. However, a NLM has **training and interpretability problems**. The well-known vanishing gradient problem makes the training of neural models inefficient. In this thesis, our goal is thus to use models that have a superior capacity for learning representation and restricting the vanishing gradient problem. Contrary to the statistical models, neural models are complex (e.g. a large hyperparameter set), not interpretable and their training is slower compared to SLMs. In this thesis, our goal is thus to improve the training and understanding of neural language models. To improve the training, our objective is to reduce the training time and improve the generalisation performance on test data. To improve the understanding of neural models, we will use the spectral weighted automata as a probing tool. To understand why a component makes neural models perform superiorly, we will focus on the understanding of the regularisation of deep neural networks.

The research is motivated by the fact that neural network models demonstrate superior representation capacity, but are inefficient in terms of slow training and difficulty in explaining the learning (i.e. explaining why better or worse performance resulted from a specific learning algorithm) as trained models are essentially uninterpretable. The focus of this research is thus on improving the training speed and the generalisation ability of neural models, as well as improving the understanding of the learning of neural network language modelling.

1.2 Original Contribution

Our research has been carried out to fulfil the focus and objectives described above, and we have made the following contributions:

1. To identify a machine learning algorithm that is best suited for language modelling, comparative analysis of different algorithms is essential. Comparative analysis of the machine learning algorithms of interest (i.e. statistical language models, spectral weighted finite-state automata and neural network models) on diverse datasets was not available in the literature. We unravel the characteris-

tics and enhance the understanding of different models and datasets by applying three types of machine learning algorithms on fifteen benchmark datasets. Moreover, we carefully choose the models so that we can use the understanding of an interpretable model to enhance the understanding of a black-box model.

2. Among different softmax approximation techniques, Noise Contrastive Estimation (NCE) is seen as a method that does not work well with deep neural models for language modelling. The deep neural models are essentially black boxes in terms of understanding, and it is difficult to explain why a certain hyperparameter setting works and why other settings do not work. A thorough investigation was done to find out the appropriate and novel integration mechanism of NCE with deep neural networks. We have also attempted to explain why the specific hyperparameter setting could have an impact on the integration. We were able to push the state-of-the-art of a language modelling task to new limits in a given class of method.

3. The established wisdom on learning theory could not explain the generalisation of deep neural networks. We have proposed methods and analysis techniques to understand the generalisation and explain the regularisation of deep neural language models.

1.3 Structure of the Thesis

The thesis contains a total of six chapters. Two chapters (i.e. chapters 3 and 4) are based on peer-reviewed published works. A journal paper is under preparation based on chapter 5. The remaining chapters consist of the introduction, background, literature review, and conclusions along with the future research directions. Concretely, the chapter organisation is as follows:

Chapter 1 contains a general introduction of this thesis outlining the main objectives, key contributions and research output.

Chapter 2 has a non-exhaustive literature review on language modelling methods. Concretely, we have highlighted the strengths and weaknesses of different language modelling methods, their learning procedure and learning complexity. We have also reviewed the generalisation properties of machine learning models.

Chapter 3 has a comparison of the performance of sequence prediction methods including the methods for language modelling reviewed in chapter 2. This chapter uses the benchmark datasets from the SPiCe sequence prediction challenge. The datasets are interesting as they are from different domains including the language modelling benchmark dataset PennTree Bank. We have also related neural models with weighted finite-state automata trained using a spectral algorithm to improve the understanding of neural models. This chapter is based on three of our published works (Liza & Grześ 2016, 2017, Liza & Grzes 2019).

Chapter 4 Based on the comparison in chapter 3, we have seen that neural network models have enhanced capacity in language modelling. Neural language models take longer to train and do not scale well when the vocabulary is large. NCE is an approximation approach developed to reduce the computational complexity of partition functions. However, it was shown in the literature that it does not work well with deep neural language models. We introduced the 'search-then-converge' learning rate schedule and other important hyperparameters for NCE and designed a heuristic that specifies how to appropriately use the hyperparameters of neural network to make NCE successfully integrated with deep neural language models. This chapter is based on our publication (Liza & Grześ 2018).

Chapter 5 Although the main goal of the NCE investigation was to incorporate NCE into a deep neural network, we have found that NCE improves the generalisation performance. In this chapter, we show that the improved generalisation is due to the fact that NCE can be seen as a regulariser because it leads to output layers that have an approximately lower rank than softmax. Our analysis is also supported by spectral analysis.

Chapter 6 concludes the thesis outlining the detailed contributions and limitations. This chapter has research questions for future work that can deepen our findings.

1.4 The Publications Derived From This Research

A Liza, F. and Grzes, M. (2016). Estimating the Accuracy of Spectral Learning for HMMs. in: Proceedings of the 17th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA). Springer, pp. 46-56.

B Liza, F. and Grzes, M. (2016). A Spectral Method that Worked Well in the SPiCe'16 Competition. in: Verwer, S., van Zaanen, M. and Smetsers, R. eds. Proceedings of the 13th International Conference on Grammatical Inference. Journal of Machine Learning Research, pp. 143-148.

D Liza, F. and Grzes, M. (2016). An Improved Crowdsourcing Based Evaluation Technique for Word Embedding Methods. in: Proceeding of the First Workshop on Evaluating Vector Space Representations for NLP (RepEval at ACL). USA: The Association for Computational Linguistics, pp. 55-61.

E Liza, F. and Grzes, M. (2018). Improving Language Modelling with Noise Contrastive Estimation. in: Proceeding of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18). Palo Alto, California, USA: AAAI Press, pp. 5277-5284.

C Liza, F. and Grzes, M. (2019). Relating RNN layers with the spectral WFA ranks in sequence modelling. in: Proceedings of the ACL Workshop on Deep Learning and Formal Languages: Building Bridges. Florence, Association for Computational Linguistics, pp. 24-33.

2

BACKGROUND

This chapter provides the background material and a literature review for language modelling. In this chapter we will describe three powerful modelling techniques, the traditional N-gram models, mathematical (formal method based) Weighted Finite-state Automata models and the recent advances of neural models. In the next chapter, we will compare these three methods with publicly available benchmark datasets.

In this thesis, we will concentrate mainly on the improvement, understanding and the analysis of the neural models. Neural networks are related to conventional statistical models (Warner & Misra 1996, Ciampi & Lechevallier 2007). These models are powerful sequence predictors and are able to learn an intricate computation, for example, their ability to sort N N-bit numbers using only 2 hidden layers of quadratic size (Razborov 1992).

This chapter is organised as follows: section 2.1 briefly reviews language modelling in general. Section 2.2 covers the intrinsic evaluation technique of the language models. Section 2.3 covers the classical techniques of language modelling. We can call them a Markov model as they are based on the Markov assumption on limited history or context. Section 2.4 defines neural network frameworks for language modelling. Contrary to the models in section 2.3, theoretically, a specific type (designed for temporal sequence) of neural models can capture unlimited history, thus they are not based on the Markov assumption. In section 2.4, we describe the different classes of neural models for language modelling and the factors that impact on the performance of neural language modelling. In section. 2.5, we will discuss the generalisation and regularisation techniques for different models.

2.1 Language modelling

Language modelling (LM) is the attempt to capture regularities of natural language for the purpose of improving the performance of various natural language applications. Generally, sequence prediction is a problem that involves using historical

sequence information to predict the next value or values in the sequence. For language modelling, the values in the sequence are linguistic units (character, subword, word, sentence, document). Language modelling is the task to model the probability that a given linguistic unit appears next after a given sequence of the same linguistic units. The historical sequence information is called the *context*. Specifically, when we study word language modelling with text data where given a sequence of observations $O = (o_0, o_1, o_2, \ldots, o_T)$ over the vocabulary V, the sequence probability is

$$p(O) = \prod_{i=1}^{T} p(o_i|o_0, \ldots, o_{i-1}) = \prod_{i=1}^{T} p(o_i|c_i). \tag{2.1}$$

Here, for a given word o_i, $c_i =< o_0, o_1, \ldots, o_{i-1} >$ represents its full, non-truncated context. For the case of just one observation in the sequence, there is no context. Language modelling can be formulated as a classification task: each word can be thought of as a class, and predicting the next word is classifying the context into a class for each next word.

2.1.1 Literature Review on Language Modelling

Word language modelling is challenging for various reasons. Linguistically, the task is ill-defined since the term 'word' has no unique meaning (Manning et al. 1999). This leads to many other challenges like data sparsity. As the same concept can be expressed in many different ways, many possible valid word sequences are not observed in a given corpus. Despite this difficulty, the researchers have developed practical models to deal with language. N-gram models are one of the earliest statistical techniques (Bahl et al. 1983, Church 1989, Jelinek 1997) for language modelling. There are smoothing techniques applied to the N-gram model to deal with the sparsity problem. We will describe N-gram modelling approaches in section 2.3.1. The basic idea is to consider the structure of a text, corpus, or language as the probability of different words occurring alone (unigram) or occurring in sequence (bigram, trigram etc). Some typical extensions to a traditional N-gram model are a Class-based N-gram model (Brown et al. 1992) and a Grammatical Trigrams (Lafferty et al. 1992). These approaches are proposed to solve the large vocabulary problem and incorporate grammatical features.

Other statistical language models were developed based on decision trees (Potamianos & Jelinek 1998, Heeman 1999) and maximum entropy-based techniques

(Rosenfeld 1994, Peters & Klakow 1999, Wang et al. 2005). These models allowed the incorporation of various manual (hand-engineered) features (e.g., part of speech (POS) tags, syntactic structure) into the language models, rather than having to rely on the words alone.

Weighted Finite-state Automata (WFAs) provide a general framework for the representation of functions for mapping strings to real numbers. They include as special instances Deterministic Finite-state Automata (DFAs), Hidden Markov Models (HMMs), and Predictive States Representations (PSRs). Learning Finite-state Automata is a fundamental task in grammatical inference. Grammatical inference is about learning a grammar given information about a language. The mathematical theory behind WFAs has been extensively studied in the past (Eilenberg 1974, Salomaa & Soittola 1978) and recently in a dedicated handbook (Droste & Kuich 2009). In NLP, WFA has been applied in parsing (Mohri & Pereira 1998), sequence modelling and prediction (Cortes et al. 2004), and language modelling (Lothaire 2005, Mohri 1997*a*,*b*).

A probabilistic WFA (PFA) is a WFA satisfying some constraints that computes a probability distribution over strings; PFA are expressively equivalent to Hidden Markov Models (HMM) (Dupont et al. 2005). The main problem with the WFA-based approach is the computational complexity (Mohri 2004). Gold (1978) and Angluin (1978) showed that the problem of finding a consistent Deterministic Finite-state Automaton (DFA) of minimum size is NP-hard. Pitt & Warmuth (1993) further strengthened these results by showing that even an approximation within a polynomial function of the size of the smallest consistent automaton is NP-hard. The survey paper by Balle & Mohri (2015) reports the general approaches for learning WFA.

In recent years, there has been a renewed interest in weighted automata in machine learning due to the development of efficient and provably correct spectral algorithms for learning weighted automata. Spectral learning has been proposed as an alternative to Expectation Maximisation (EM) based algorithms to learn HMMs (Hsu et al. 2012), WFAs (Balle & Mohri 2012, Balle et al. 2014), predictive state representations (Boots et al. 2013), and other related models. Compared to EM-based methods, the spectral method has the benefits of providing consistent estimators and reducing **computational complexity**. These models offer polynomial information-theoretic complexity in the PAC learning model (Angluin 1988, Guedj 2019). In section 2.3.2, we will describe the spectral approach for the WFA for language modelling.

The connectionist approach of language modelling gained popularity due to the highly cited artificial neural networks based language model by Bengio et al.

(2003). Bengio et al. (2003) applied a feedforward neural network (FFNN) to a training set consisting of a sequence of words, showing how the neural model could simultaneously learn the language modelling probability that a certain word appears next after a given sequence of words and at the same time learn a *real-valued vector* representation for every word in a predefined vocabulary. Later, Mikolov, Sutskever, Chen, Corrado & Dean (2013), Mikolov, Yih & Zweig (2013) showed that the word representation learned by the neural models captures linguistic regularities and word compositionality. Thus the neural model learned an appropriate set of linguistic features. This is in contrast to the decision tree and maximum entropy language models, which typically required the features to be manually engineered before any model could be trained (Heeman 1999, Peters & Klakow 1999). More specifically, neural language models learn their features jointly with other parameters, while maximum entropy models use fixed hand-engineered features designed by the domain experts and only learn the parameters for those features. Designing features is time-consuming and requires many years of experience.

Feature learning

The main limitation of the FFNNs of (Bengio et al. 2003) is that only a fixed number of previous words (limited fixed context) can be taken into account to predict the next word. The limitation is structural as the FFNN does not have so-called 'memory' represented by dedicated processing units for temporal sequence. The 'memory' is related to the ability to capture a non-fixed context. In the FFNN, the words that are presented via the fixed number of processing units can be used to predict the next word and all words that were presented during earlier iterations are disregarded, although these words can be essential to determine the context and thus to determine a suitable next word. The shortcoming of lack of 'memory' was attempted to address by Jordan (1986) (using Jordan networks) and Elman (1990) (using Elman networks) by adding extra processing units. These extra processing units are context processing units known as *context neurons* and hold the contents of the previous context. The context length was extended to indefinite, one could even say infinite, size by using a recurrent version of neural networks mostly inspired by the Elman network and conveniently called recurrent neural networks (RNN), which can handle arbitrary context lengths. The term 'recurrent' applies as they perform the same task over each instance of the context such that the output is dependent on the previous computations and results (Young et al. 2018).

The RNN based statistical language model (Mikolov et al. 2010) was shown to be very slow in training, although it performed superiorly to an N-gram model (with Modified Kneser-Ney Smoothing and $N = 5$). The initial enthusiasm about RNN as statistical language models was mainly driven by their abilities to learn vec-

tor representations for words (as in FFNN) and to handle arbitrarily long contexts. In addition, they are universal approximators (Schäfer & Zimmermann 2006). However, the enthusiasm was quickly counterbalanced by their extremely *slow learning* and by the empirical observation that the theoretical incorporation of arbitrarily long context lengths does not occur in practice. To reduce the training time and capture larger context (if not infinite), many approached have been proposed (Mikolov, Deoras, Povey, Burget & Černockỳ 2011, Mikolov & Zweig 2012, Chen et al. 2015, Wang & Cho 2016). In this thesis, we explore a similar direction to contribute to deep learning research.

2.2 Evaluation of Language modelling

2.2.1 Intrinsic Evaluation: Perplexity

Perplexity is a well established intrinsic evaluation metric for language modelling. Intrinsic evaluation metrics allow measuring the quality of a model independent of a particular application (Jurafsky 2000, Goodman 2001, Mikolov 2012). Perplexity measures how well a probability distribution predicts a sample. Higher probability means a model better predicts the future. There is a limit to how well you can predict a random future, and the limit depends on 'how random' the dataset is. For example, it is easier to predict the weather than the news headlines.

Thus perplexity is related to Shannon's Entropy (Shannon 1948) and more specifically cross-entropy between the model and the dataset when we do not know the true distribution of the model that is generating the sequence. The upper bound of the cross-entropy is the entropy of the true distribution. If the predicted model is the same as the true model then the cross-entropy is the same as the entropy of the model. Thus the more accurate the predicted model is, the closer cross-entropy will be to the true entropy. Therefore, between two models the better model is one that has lower cross-entropy. The relation between the cross-entropy and the perplexity is that the lower the cross-entropy (uncertainty), the lower the perplexity. Therefore, *between two models, the better model is the one that has lower perplexity*. The perplexity metric in NLP is a way to capture the degree of 'uncertainty' that a model has in predicting (assigning probabilities to) some text. Formally, the relation between cross-entropy and perplexity for language modelling is explained below.

The Perplexity (PPL) of a language model on a sample data $w = w_1, \ldots, w_L$

is the inverse probability of w, normalized by the number of words (Jurafsky 2000):

$$\begin{aligned} PPL(w) =& p(w_1, \ldots, w_L)^{-\frac{1}{L}} \\ =& \sqrt[L]{\frac{1}{p(w_1, \ldots, w_L)}} \\ =& \sqrt[L]{\frac{1}{\prod_{i=1}^{L} p(w_i | w_1 \ldots, w_{i-1})}} \end{aligned} \tag{2.2}$$

The cross entropy of the language model on generating the sample data $w = w_1, \ldots, w_L$ is given by (Jurafsky 2000):

$$\begin{aligned} H(w) =& -\frac{1}{L} \log_2 p(w_1, \ldots, w_L) \\ =& -\frac{1}{L} \log_2 \prod_{i=1}^{L} p(w_i | w_1, \ldots, w_{i-1}) \end{aligned} \tag{2.3}$$

From equation 2.2 and A.1, the relation between the perplexity and the cross entropy is as follows:

$$\begin{aligned} 2^{H(w)} =& 2^{-\frac{1}{L} \log_2 p(w_1, \ldots, w_L)} \\ =& 2^{\log_2 p(w_1, \ldots, w_L)^{-\frac{1}{L}}} \\ =& p(w_1, \ldots, w_L)^{-\frac{1}{L}} \\ =& \sqrt[L]{p(w_1, \ldots, w_L)^{-1}} \\ =& \sqrt[L]{\frac{1}{p(w_1, \ldots, w_L)}} \\ =& \sqrt[L]{\frac{1}{\prod_{i=1}^{L} p(w_i | w_1, \ldots, w_{i-1})}} \\ =& PPL(w) \end{aligned} \tag{2.4}$$

$$\boxed{PPL(w) = 2^{H(w)}}$$

A more detailed description of the relationship between cross-entropy and perplexity can be found in Appendix A.

2.3 Classic Language Modelling Techniques

In this section, we will explore N-gram and Weighted Finite-state Automata (WFA) based language modelling approaches and their learning procedure.

2.3.1 Statistical N-grams

An N-gram is a sequence of N-items — if the items are words then a 2-gram (or bigram) is a two-word sequence of words, such as 'Another thesis', 'thesis has', or 'has been', and a 3-gram (or trigram) is a three-word sequence of words, such as 'Another thesis has', or 'thesis has been'.

In N-gram language modelling, the previous $N-1$ words are used to predict the current (N^{th}) word. N-gram models are used to estimate the probability of the last word of an N-gram given the previous $N-1$ words, and also to assign probabilities to entire sequences. For sentence level modelling, a model provides an estimation of a probability distribution over words that attempts to reflect how frequently the sequence of words occurs as a sentence. Without loss of generality, we can express the probability $p(O)$ of strings $O = (o_0, o_1, o_2, \ldots, o_T)$ as follows (utilising the chain rule and the Markov assumption),

$$\begin{aligned} p(O) &= p(o_0)p(o_1|o_0)p(o_2|o_0, o_1)\ldots p(o_T|o_0, o_1, \ldots, o_{T-1}) \\ &= \prod_{i=1}^{T} p(o_i|o_0, \ldots, o_{i-1}) \approx \prod_{i=1}^{T} p(o_i|o_{i-(N-1)}, \ldots, o_{i-1}) \end{aligned} \tag{2.5}$$

Here, based on the N^{th} order Markov property, it is assumed that the probability of observing the $(i)^{th}$ observation, o_i, with respect to the whole context history of the preceding $i-1$ observations can be approximated by the probability of observing it in the shortened context history of the preceding $N-1$ observations for a particular N-gram model.

Word co-occurrence statistics are used to train N-gram models. The co-occurrence statistics are based on N-gram context statistics. For example, if the length of the context history is two (trigram model), then the maximum likelihood estimate (MLE) can be used to train the model. The probability of a word o_i in the sequence O with respect to the two-word history context can be calculated by dividing the total number of three words sequences with o_i as the last in the sequence by the

total number of the two words context history in the corpus. If there is no context considered in the model training then the word frequency in isolation leads to a **unigram** model.

Unigram model

For $N = 3$, the trigram model prediction is

$$p_{MLE}(o_i|o_{i-2}, o_{i-1}) = \frac{c(o_{i-2}, o_{i-1}, o_i)}{\sum_w c(o_{i-2}, o_{i-1}, w)}$$

For $N = 1$, the unigram model prediction is

$$p_{MLE}(o_i) = \frac{c(o_i)}{\sum_w c(w)}$$

here c is the count of the occurrence of word patterns in the training corpus. Here, w denotes the words in the corpus. For $N = 3$, w correspond to the words in the corpus that have the specific context o_{i-2}, o_{i-1}.

The advantage of this model is simplicity and fast computation. The statistics and the probabilities of the N-grams can be computed beforehand thus fast computation is the advantage of using N-gram models. However, the number of possible N-grams increases *exponentially* with the length of the context. Even with large training texts, some N-grams will not appear or will occur too infrequently to provide a statistically meaningful probability. Therefore, although the N-grams calculate the probabilities efficiently, capturing the long term dependency is difficult due to *unavailability* of a large enough dataset that will contain all the larger context patterns. Thus, preventing these models from effectively capturing the long term dependency through the longer context patterns. Another problem with this model is the sparsity in context history. If one word in a sentence is less frequent or does not appear in a certain context in the corpus then the probability of the sentence will be zero for that word and that would make the probability of that sentence very low or zero. To solve this problem, different smoothing techniques have been proposed (Chen & Goodman 1996). Smoothing is solving the problem with the MLE learning that arises when the N-gram probability estimation is based on the unseen context (cases where the MLE will simply assign a probability of zero to the sequences). This is an inevitable problem for language modelling tasks because no matter how large the corpus is, it is very difficult for a dataset to contain all possibilities of N-grams from the language. The basic principle of smoothing is transferring some probability mass from some more frequent events (N-grams) to the less frequent or not occurring events (N-grams). The most prominent smoothing techniques are Laplace smoothing, Add-k smoothing, Good-Turing backoff, Kneser-Ney Smoothing and Modified Kneser-Ney Smoothing. It is confirmed in previous experiments (Goodman 2001) that

Requirement of a very large training dataset

among different smoothing techniques, the modified Kneser-Ney smoothing (Kneser & Ney 1995) (MKN) provides consistently the best results among smoothing techniques, specifically for the word language modelling task. The quality of the N-gram model thus highly depends on the choice of the order (the value of N), consequently on the quality and amount of data (Lesher et al. 1999) for meaningful statistics and the choice of smoothing techniques.

We noted that the number of possible N-grams increases exponentially with the length of the context (value of $N-1$). Here we will calculate a number of parameters for an N-gram model to explain this phenomenon. The number of parameters for an N-gram model can be expressed as $|V|^N$, where N is the length of the sequences and V is the number of observations (vocabulary size). So, for a 3-gram model with vocabulary size 20000, the number of parameters is $20000^3 = 8000000000000 = 8 \times 10^{12}$.

2.3.2 Weighted Finite-state Automata

WFA represent functions for mapping strings to real numbers. WFA include as special instances Deterministic Finite-state Automata (DFAs), Hidden Markov Models (HMMs), and Predictive State Representations (PSRs). Let Σ^* denote the set of strings over a finite alphabet Σ and let λ be the empty string. A WFA with n states is a tuple $A = \langle \alpha_1, \alpha_\infty, \{A_\sigma\} \rangle$ where $\alpha_1, \alpha_\infty \in \mathbf{R}^n$ are the initial and final weight vectors respectively, and $A_\sigma \in \mathbf{n} \times \mathbf{n}$ is the transition matrix for each symbol $\sigma \in \Sigma$. A WFA computes a function $f_A : \Sigma^* \to \mathbf{R}$ defined for each string $x = x_1 x_2 \dots x_k \in \Sigma^*$ by $f_{A(x)} = {\alpha_1}^\top A_{x_1} A_{x_2} \dots A_{x_k} \alpha_\infty$. Figure 2.1 shows an example of graph representation with 2 states and alphabet $\Sigma = a, b$. The corresponding algebraic representation of a weighted automaton over $\Sigma = a, b$ with 2 states in Fig. 2.1 is presented in Eq. 2.6 .

$$\alpha_1^\top = \begin{bmatrix} 1/2 & 1/2 \end{bmatrix} \quad \alpha_\infty^\top = \begin{bmatrix} 1 & -1 \end{bmatrix} \quad A_a = \begin{bmatrix} 3/4 & 0 \\ 0 & 1/3 \end{bmatrix} \quad A_b = \begin{bmatrix} 6/5 & 2/3 \\ 3/4 & 1 \end{bmatrix} \tag{2.6}$$

As described earlier in section 2.1.1, approximating distributions over strings is a hard learning problem. Learning WFA has exponential computational complexity (Mohri 2004). The recent advancement in learning WFA is based on spectral learning, which reduces the computation complexity of learning WFA (Balle et al. 2014).

Spectral algorithms for Weighted Finite-state Automata (WFA) use the information contained in the eigenvectors of a data affinity (i.e. item-item similarity)

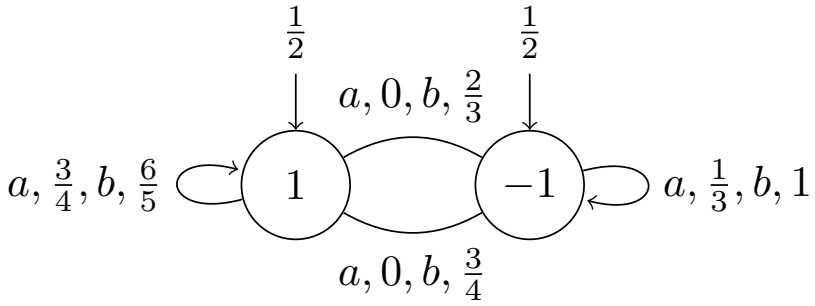


Figure 2.1: Example of a weighted automaton $A = \langle \alpha_1, \alpha_\infty, \{A_\sigma\} \rangle$ over $\Sigma = a, b$ with 2 states (graph representation)

matrix to detect low dimensional structure. Spectral algorithms use singular value decomposition (SVD) to retrieve the structure in the low dimensional space and use Method of Moments (MoM) based simple estimation techniques to compute the parameters in that low dimensional space. Although all the spectral algorithms use SVD as the basic operation, there is no unified learning method (e.g. MLE, EM, back-propagation) that works for many related models. Therefore, each learning problem has to be formulated independently as a matrix (or tensor) decompositions.

In recent literature, there are several approaches (different choice of prefixes and suffixes) for estimating WFAs have been proposed that are based on representing the function computed by a WFA using a Hankel matrix (Balle et al. 2014). A sequence labelled by the integers (instead of just the natural numbers) is called a bi-infinite sequence as it has infinitely many entries in both the positive and negative directions. The Hankel Matrix of a function is a bi-infinite matrix. A bi-infinite matrix refers to a representation of an operator that maps between two vector spaces of bi-infinite sequences. Given a sequence of output data, Hankel matrices are formed to retrieve a realisation of an underlying state-space (e.g. hidden Markov model). The singular value decomposition of the Hankel matrix provides a means of computing the matrices which define the state-space realisation.

The spectral-based parameter estimation follows from a duality result between minimal Weighted Finite-state Automata (WFA) and factorisation of Hankel matrices (H_f). A function $f : \Sigma^* \to \mathbb{R}$ is recognisable if it can be computed by a WFA. In this case, the rank of f is the number of states of a minimal WFA computing f. If f is not recognisable, we let $\text{rank}(f) = \infty$.

A Hankel-based method works as follows: given a set of pairs $(O, f(O))$,

where O is a sequence of observations in the support of some target function f over Σ^*, the goal is to learn an approximation of f. A Hankel based method provides a spectral solution to this problem. The method starts by choosing a set of prefixes P and suffixes S for selecting the *basis*. In the second step, the basis will be used to generate the Hankel matrix $H_f \in \mathbb{R}^{|P|\times|S|}$. The entries of the $H_f(p,s)$ for $p \in P$ and $s \in S$ is the target function on the sequence obtained by concatenating prefix p with suffix s. For example, Eq. 2.7 defines a Hankel matrix $\hat{H_S}(p,s) = \hat{f_S}(ps)$, when $\{aa, b, bab, a, b, a, ab, aa, ba, b, aa, a, aa, bab, b, aa\}$ is a sample of N i.i.d strings. Hankel matrix has rows correspond to the set $P = \{\lambda, a, b, ba\}$ which is the prefix and has columns correspond to the set $S = a, b$, which is the suffix. The entries in the Hankel matrix is frequency of strings (generated by concatenating the prefix and suffix) in the sample. The set P and S are called basis.

Hankel based spectral method in general.

$$\hat{H_S} = \begin{bmatrix} & a & b \\ \lambda & 0.19 & 0.25 \\ a & 0.31 & 0.06 \\ b & 0.06 & 0.00 \\ ba & 0.00 & 0.13 \end{bmatrix} \tag{2.7}$$

Then SVD will be applied on the H_f matrix to factorise it into three matrics (U, L, V). The factorisation $F = UL$, $B = V^\top$ and H_f are used to recover the parameters of the minimal WFA.

The method by Balle et al. (2014) provides an efficient algorithm that implements the ideas of the Lemma provided by Balle et al. (2014, Lemma 2) to find a rank factorisation of a complete sub-block H of H_f and obtains from it a minimal WFA for a function f which maps strings to real numbers. Balle et al. (2014) observed that a WFA $A = \langle \alpha_1, \alpha_\infty, \{A_\sigma\} \rangle$ for f with n states induces a factorisation of Hankel matrix H_f. They argued that similar phenomena can be observed for sub-blocks of H_f. Let H be a complete sub-block of H_f defined by an arbitrary basis $\mathcal{B} = (\mathcal{P}, \mathcal{S})$.

In practice, with limited amount of real datasets, the H and H_f are not known exactly. However, it is possible to apply the algorithm on approximate $\hat{H}$. Here, approximation means the $\hat{H}$ can be calculated from a number of strings in $W = \{\lambda, \Sigma\}$ rather than from all strings of $W = \{\lambda, \Sigma\}$.

The hyperparameters of the learning algorithm (Balle et al. 2014) for retrieving the parameters of the minimum WFA are the number of states n of the target WFA and the basis (i.e. sets of $\mathcal{P}$ and $\mathcal{S}$). This n is also a rank of the n-dimensional

reconstruction of the Hankel matrix when the best n dimensions of its SVD are used.

A basis that contains most frequent elements (substrings) observed in the sample can be chosen based on the work by Balle et al. (2012) as this approach was found computationally efficient. In that case, the rows and columns of the Hankel matrix correspond to the substrings, and the cells of the Hankel matrix contain the frequencies of the corresponding substrings. In this approach, the length of these substrings along rows (nR) and along column (nC) are also the hyperparameters of the spectral learning algorithm for WFA (Balle et al. 2014).

To summarise, the purpose of the spectral algorithm is to compute a minimal WFA for a function (f) defined on strings $f : \Sigma^* \rightarrow \mathbb{R}$ with finite rank n. The algorithm assumes that $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ is an arbitrary basis for f, and given that basis along with values of f on a set of strings or frequencies of substrings W, where $W = \{\lambda, \Sigma\}$, as input, the algorithm does a rank factorisation of a complete sub-block of the Hankel matrix to be able to apply the formulas given in Balle et al. (2014, Lemma 2). Using the formulas in Balle et al. (2014, Lemma 2), the probabilities of sequences can be determined as follows: $P(O_{1:T}) = (\alpha_1)^\top A_{o_1} \dots A_{o_T} \alpha_\infty$.

Spectral methods are appealing because they offer consistent estimators (and PAC-style guarantees of sample complexity) for several important models (WFA, HMM, etc). This is in contrast to the EM algorithm, which is an extremely successful approach, but which only has guarantees of reaching a local maximum of the likelihood function, it is slow to converge, and difficult to analyse.

Although spectral algorithms for WFA offer many desirable properties, there are some issues with the matrix decomposition. When the Hankel matrix-based spectral method is used to capture long-range dependencies, very large Hankel matrices have to be considered (Quattoni et al. 2017). Thus the computation of the SVD becomes a critical bottleneck (Gillis & Glineur 2011, Hillar & Lim 2013). Moreover, the spectral methods have convergence problems (estimated parameters go outside the parameter space) even for the two-dimensional matrix decomposition case (Liza & Grześ 2016) with limited training data.

2.4 Neural Models

In this section, we will explore neural network based language modelling approaches and their learning procedure. Before we do that, we will explain some of the basic

material related to Artificial Neural Networks (ANNs). ANNs were expected to replicate the architecture of the human brain, however until recently the only common feature between ANN and our brain was the similarity of their entities (e.g. neurons). The average human brain has about 100 billion neurons (or nerve cells) (Herculano-Houzel 2009). To replicate the biological neuron, classical neural network models approximate neurons as devices that sum their inputs and generate a non-zero output if the sum exceeds a threshold (e.g. activation function). For example, Figure 2.2 shows an artificial neuron. There are i input nodes connected to the j^{th} output node and the activation function $\varphi(.)$ will implement the threshold to show a particular predicted output. Based on the prediction N_j, an error e_j will be calculated and error back-propagation will be used to update the weights $w_j i$. This computation can be generalised for a larger neural network where many layers and many nodes are available as shown in the Fig. 2.3. From our current state of knowledge in neurobiology, it is easy to criticise these models as over-simplified (y Arcas et al. 2001). A neural network is thus a simplified model of the way the human brain processes information. It works by simulating a large number of interconnected processing units that resemble abstract versions of neurons.

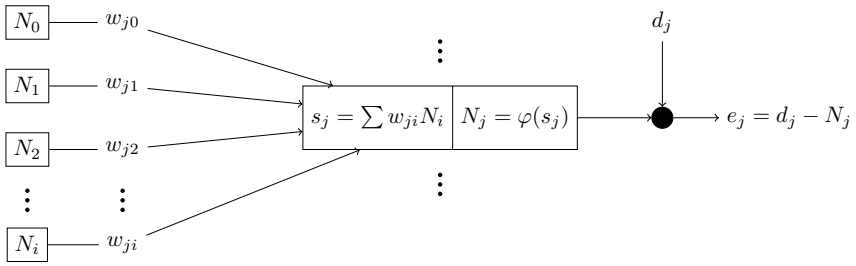


Figure 2.2: A Neuron Cell

The processing units are arranged in layers (e.g. Fig. 2.3). There are typically three parts in a neural network: an input layer, with units representing the input data; one or more hidden layers; and an output layer, with a unit or units representing the target data. The units are connected with varying connection strengths (or weights). Input data are presented to the first layer, and values are propagated or distributed from each neuron to every neuron in the next layer. Eventually, a result is delivered from the output layer. The network learns by examining individual instance/records, generating a prediction for each record, and making adjustments to the weights whenever it makes an incorrect prediction. This process is repeated many times, and the network continues to improve its predictions until one or more of the stopping criteria (i.e. convergence criteria) have been met.

A neural network learns using one of the three basic machine learning

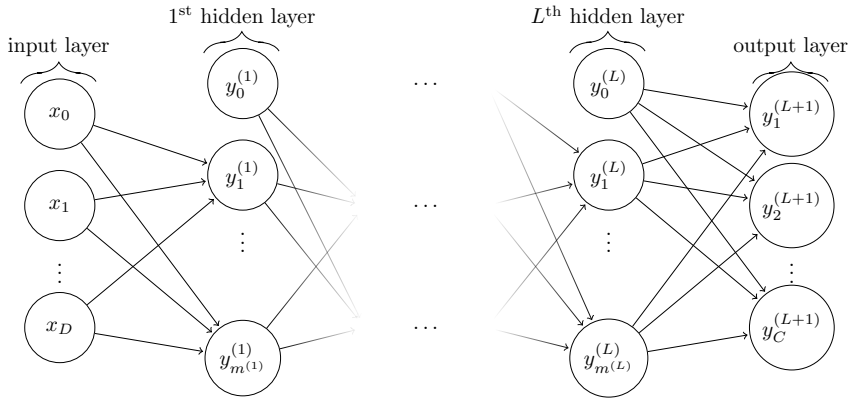


Figure 2.3: Network graph of a $(L+1)$-layer perceptron with D input units and C output units. The l^{th} hidden layer contains $m^{(l)}$ hidden units, source [a]

[a] https://github.com/davidstutz/latex-resources/blob/master/tikz-multilayer-perceptron/multilayer-perceptron.tex

paradigms: Supervised, Unsupervised and Semi-supervised. The majority of practical machine learning uses supervised learning. Supervised learning is where one has input variables x to process input data and an output variable Y to process target data, and an algorithm is used to learn the mapping function $f(.)$ from the input to the output: $Y = f(X)$. The goal is to approximate the mapping function so well that when unseen input data example is presented at input X, the function should predict the output Y for that data example. It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers, the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance (i.e. using a stopping criteria).

For training a neural network using the supervised machine learning paradigm, all weights needs to initialised randomly, and the answers that come out of the randomly initialised network are probably nonsensical, but the initialisation has important impact on the overall training. For example, if we initialise all weights with zero, the learning might not converge at all no matter how much data is available. After initialisation, labelled instances are repeatedly presented to the network, and the predictions it gives are compared to the known true target data. Information from this comparison is passed back through the network, gradually changing the weights using relevant optimisation techniques. As training progresses, the network becomes increasingly accurate in replicating the known outcomes. Once trained, the network

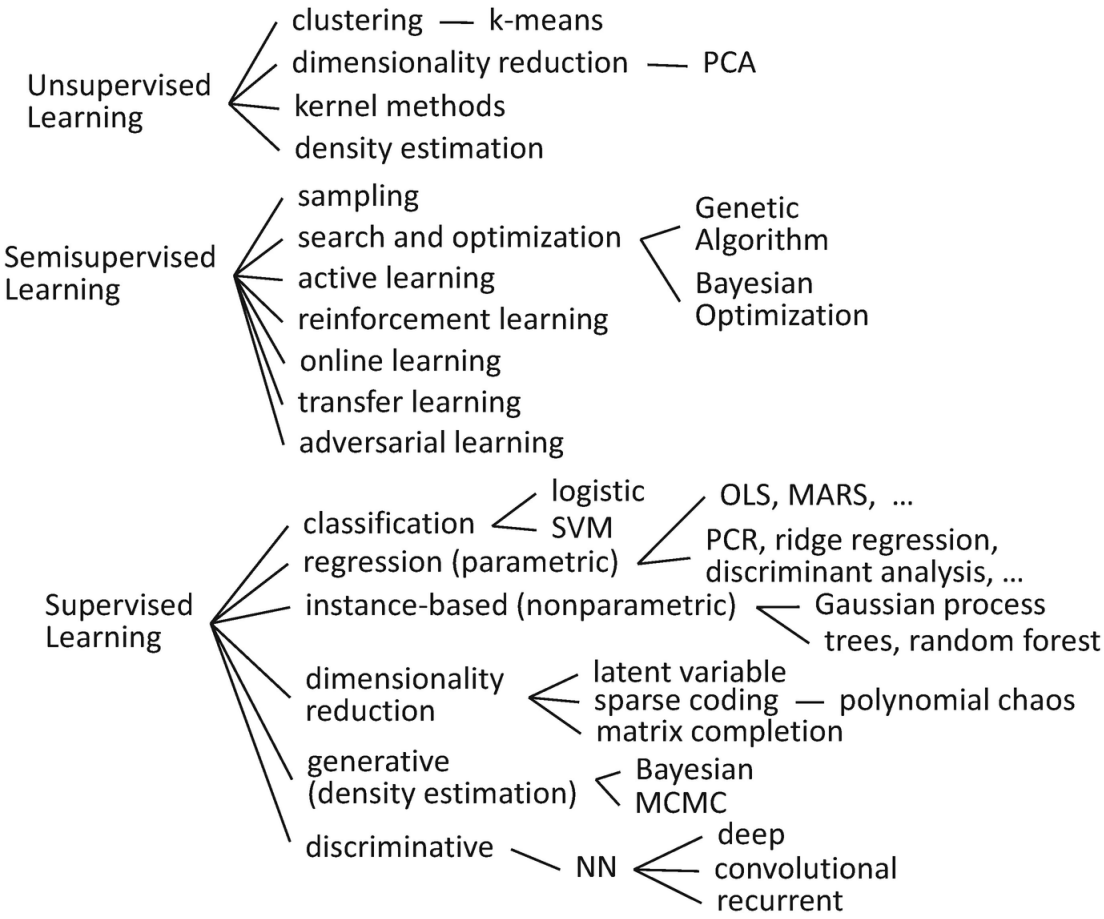


Figure 2.4: An approximate high-level taxonomy of machine learning methods (Boning et al. 2019). The research in this thesis is mostly under the category of discriminative supervised learning with deep recurrent neural networks.

can be applied to future cases where the true label is unknown.

Classification is considered an instance of supervised learning. The softmax function is often used in the final layer of a neural network-based classifier. Such networks are commonly trained under a cross-entropy regime, giving a non-linear variant of multinomial logistic regression. Multinomial logistic regression is a classification method that generalises logistic regression to multiclass problems, i.e. with more than two possible discrete target data. In mathematics, the softmax function is a function that takes as input a vector of real numbers, and normalises it into a probability distribution consisting of probabilities proportional to the exponentials of the input numbers.

Softmax Function

Contrary to the supervised learning, unsupervised learning is where only input data x are available and there are no corresponding target variables. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data or structure them meaningfully for a supervised task. These are called unsupervised learning because unlike supervised learning above there is no correct answers or target data, so there is no teacher supervision. Algorithms are designed to discover and present the interesting structure in the data without any supervision or target data.

Semi-supervised learning (SSL) is halfway between supervised and unsupervised learning. In addition to unlabelled data, the algorithm is provided with some supervision information through labelled data but not for all instances. The goal of SSL is to understand how combining labelled and unlabelled data may change the learning behaviour, and design algorithms that take advantage of such a combination. Semi-supervised learning is of great interest in machine learning and data mining because it can use readily available unlabelled data to improve supervised learning tasks when the labelled data are scarce or expensive. There are popular SSL models, including self-training, mixture models, co-training and multi-view learning, graph-based methods, and semi-supervised support vector machines. More detailed descriptions of these models can be found in (Zhu & Goldberg 2009).

There are different types of neural networks. Perceptron, the simplest and oldest model of neuron. These models take some weighted inputs, sum them up, apply activation function and pass them to the output layer. A neuron represents a linear model if we take the sum of weighted inputs and use that as an output of the neural model (Rosenblatt 1958). Perceptron can not learn a non-linear function. Multilayer perceptron (MLP) composed of several perceptron-like units arranged in multiple layers consists of an input layer, one or more hidden layers, and an output

layer. MLP can compute the non-linear function by computing a nonlinear transformation of the inputs utilising the nodes in the hidden layers. If a MLP does not have loops (i.e. no backward connections between layers), then MLP becomes a feedforward neural network (FFNN) (e.g. Fig. 2.3). A feedforward neural network with a special structure is called Convolutional Neural Network (CNN) (LeCun et al. 1999). The special structure is imposed by a sparse 'local' connectivity between layers (except the last output layer) and by shared weights (like a 'global' filter). The objective of these networks is to reduce the number of parameters to be learned. The global filters help capture the local properties of the input data.

For sequence modelling, Recurrent Neural Networks (RNNs) introduce recurrent connections between sequential input data at different time steps. The limitation of feedforward neural networks (FFNN) for sequence modelling is that they do not share features across different positions of the network. In other words, these models assume that all inputs (and outputs) are independent of each other. FFNN model would not work efficiently in sequence modelling since the previous inputs are inherently important in modelling the sequential data.

Compared to the N-gram and WFA models, the neural models have enhanced ability to generalise (Goldberg 2017, p109) for long and unobserved sequences. An example of enhanced generalisation with an unobserved sequence was provided by Goldberg (2017): "by observing that the words blue, green, red, black, etc. appear in similar contexts, the model will be able to assign a reasonable score to the event green car even though it never observed this combination in training, because it did observe blue car and red car.". Moreover, diverse neural model architectures can be trained using standard learning algorithms (back-propagation) and various optimisation techniques. As a result, neural networks can be seen as differentiable function approximators. More generally, the neural models are shown as universal function approximators. It is a well known and important result that neural networks are universal in the sense that any function can be approximated to arbitrary accuracy, as shown by Tikhomirov (1991), Hecht-Nielsen (1992), Hornik et al. (1990). The enhanced capability of learning representation and universal approximation make these models attractive for various research areas.

Memoryless neural models for sequences like autoregressive models predict the next term in a sequence from a fixed number of previous terms. Autoregressive models use shift registers of delays (delay taps) to retain successive values of the sequences. Feed-forward neural nets generalise autoregressive models by using one or more layers of non-linear hidden units. These models are not as effective as memory-based neural models (e.g. RNN) for sequence modelling. The memory-

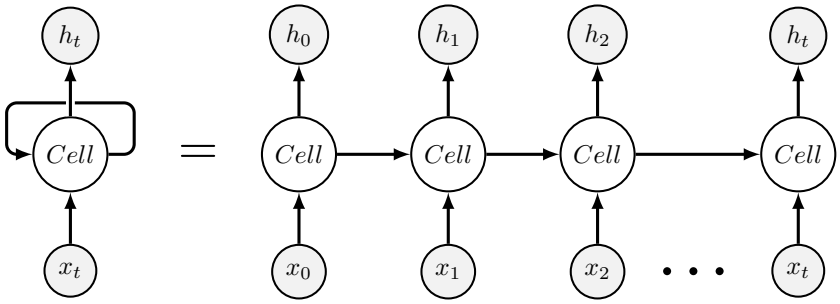


Figure 2.5: Recurrent Neural Network

based models have a distributed hidden state and recurrent connection that allows them to store past information efficiently. The memory-based models (e.g. RNN) are fundamentally different from feed-forward architectures in the sense that they operate on not only an input and output space (vertical arrows in Fig. 2.5) but also an internal state space (using recurrent connections, horizontal arrows in Fig. 2.5), and the internal state space enables the representation of longer sequential data. The RNN models with different cell types (e.g. Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU)) can store and retrieve information in a hidden state to solve computational issues. Many modifications of the conventional architecture have been proposed to make memory-based models more efficient for several tasks.

Learning in neural networks

The exact function that a neural network learns is determined by its architecture and a set of adaptable parameters, known as weights in parameter space. For example, in the network in Fig. 2.6, the total number of adaptable weights is 26. The calculation is as follows: there are $4+2=6$ neurons (not counting inputs), therefore, $[3x4]+[4x2]=20$ weights and $4+2=6$ biases making a total 26 trainable parameters.

The architecture and the training are determined by the hyperparameters (e.g. number of hidden nodes, the learning rate, etc). Hyperparameters are set before optimising the adaptable parameters. The adaptable parameters, for example a weight matrix θ in the parameter space Θ (i.e. $\theta \in \Theta$), for a given neural architecture learns the function using back-propagation and optimisation.

The neural network model is applied to some input x (training data) to produce some output of y (predicted value). More formally, $f : \Theta \times \mathcal{X} \rightarrow \mathcal{Y}$, where the domain of the function consists of the parameter space Θ and the input space $\mathcal{X}$, and the co-domain is the output space $\mathcal{Y}$.

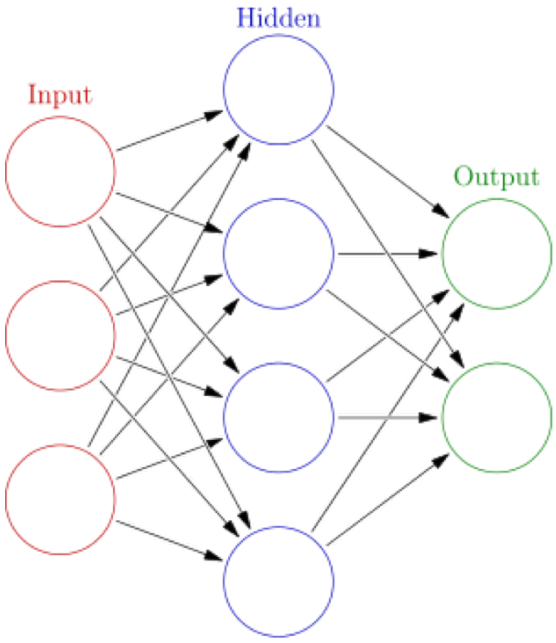


Figure 2.6: Parameters in a Neural Network (source: wikipedia)

Classical optimisation vs machine learning optimisation

The architecture of a neural network is typically designed by an expert, while the parameters Θ are algorithmically determined using continuous optimisation techniques (Nocedal & Wright 2006). Learning in machine learning, in general, is not just optimisation based on the given data, it is more about generalisation on the unseen data. Optimisation in a mathematical setting is a little different in machine learning in the sense that a mathematician assumes that an infinite amount of data is available, whereas in machine learning researchers assume that a finite segment of the data is available. This finite segment of data is known as training data and the objective of the machine learning algorithm is to optimise with respect to unseen data given the limited finite training data. Optimisation algorithms used for training deep models thus differ from traditional optimisation algorithms in several ways (Goodfellow et al. 2016, chapater 8). Machine learning usually acts indirectly compared to the traditional optimisation algorithm. In most machine learning scenarios, emphasis is given to some performance measure P, that is defined with respect to the test set (unseen examples) and may also be intractable. Thus optimisation of P is only done indirectly. A different cost function $J(.)$ which is based on the seen examples is reduced in the hope that doing so will improve P. This is in contrast to pure optimisation, where minimising J is the main goal. One aspect of machine learning algorithms that further separates them from traditional optimisation algorithms is that the cost function usually decomposes as a sum over the training examples.

Optimisation algorithms for machine learning typically compute each update to the parameters based on an expected value of the cost function estimated using only a subset of the terms of the full cost function.

For machine learning in general and for deep learning in particular, stochastic gradient descent (SGD) and its variants (Ruder 2016) are probably the most used optimisation algorithms. SGD is the incremental gradient descent based on the single sample or sub-sampled batch (mini-batch) instead of the whole batch in the batch gradient descent. This optimisation technique uses the gradient of the loss function to find the local minimum (if the objective is to minimise the loss function) by taking one step at a time toward the negative of the gradient. The size of the step in optimisation is known as the learning rate. Learning rate is the crucial parameter for the SGD algorithm. Learning rate is a hyper-parameter that controls how much the parameters Θ of the neural network to be adjusted with respect the gradient of the cost function. The lower the value, the slower training move along the downward slope of the cost function. While using a low learning rate might be a good idea so that no local minima be missed during the training, it could also mean that it will take a long time to converge — especially if training get stuck on a saddle point (Adolphs 2018).

If the true labels ($z \in \mathcal{Z}$) are available (i.e. supervised learning), we can formulate the desired behaviour of a function as a scalar loss function with $l : \mathcal{Y} \times \mathcal{Z} \rightarrow R$. For example, a widely used loss measure for supervised regression is the the sum of squares of the prediction errors or Mean Squared Error (MSE) (Gavish & Donoho 2017). The MSE can be written as $l(y, z) = \|z - y\|^2$, where $\|.\|^2$ denotes the Frobenius norm of a vector.

Complete loss $\mathcal{L}$ is dependent on l and f, thus PPL is also dependent on l and f.

The composition ($\circ$) of f and l are used to define a complete loss function $\mathcal{L} = l \circ f$ with $\mathcal{L} : \Theta \times \mathcal{X} \times \mathcal{Z} \rightarrow R$. If both, f and l are differentiable, the backpropagation algorithm and the gradient-based optimisation techniques can be used to find good parameters Θ after an application of the chain rule:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial f} \frac{\partial f}{\partial \theta}$$

.

This makes neural networks a adaptable tool for diverse tasks, since we can define the desired characteristics of a function in terms of some input observations without designing features for each input domain.

Training a neural network is inherently very difficult, specifically, it is prone

to fit the training data very well, it can even fit outliers (noise) due to co-adaptations and fail to generalise to new examples (test data). There are different mechanisms to avoid over-fitting. Specifically, dropout (Srivastava et al. 2014) was proposed to remove the co-adaptation. Dropout randomly drops units (along with their connections) during training and each unit is retained with fixed probability p, independent of other units. The hyper-parameter p is selected based on the dataset. When dropout is applied to a recurrent neural network, (Zaremba et al. 2014) proposed that dropout should not be applied to the recurrence layer. Later, (Gal & Ghahramani 2016*b*) showed how to apply dropout in the recurrence layer and theoretically proved dropout as a Bayesian-approximation to reason about model uncertainty (Gal & Ghahramani 2016*a*). The proposed dropout technique of Gal & Ghahramani (2016*b*) is called variational dropout.

Dropout and variational dropout

Other regularisation techniques have been applied to deep learning that are more common in the machine learning paradigm in general. Such mechanisms include adding regularisation term that penalises big weights. For example, the regularisation term added to the objective function that penalises big weights, weight decay coefficient determines the dominance of the regularisation during gradient computation and big weight decay coefficient are set for to impose big penalty for relatively large weights. There is another regularisation technique used in the deep learning paradigm is early stopping. This technique uses validation error (i.e. error calculated on some heldout data that has not been used in model training) to decide when to stop training. Training will be ceased once monitored validation error has not improved after several subsequent epochs.

Types and architectures of neural networks

As described earlier in this section, there are several types of neural network including perceptron, feed-forward neural network, recurrent neural network with several extensions in cell type including long short-term memory, gated recurrent unit, deep convolutional neural network, generative adversarial network, deep residual network, recurrent highway network. They differ in many ways including the cell types and the hidden layer connections. For language modelling, recurrent neural networks offer extensive modelling power through recurrent connections.

In this thesis, we will be mainly working on the RNN type of neural network with an LSTM cell and its particular generalisation Recurrent Highway Network (RHN). There are several proposed variants to the standard LSTM cell (see Fig. 2.7) including Peephole LSTM (Gers et al. 2002) and Gated Recurrent Unit (GRU) (Cho et al. 2014). However, the neural network based on the LSTM for sequence modelling

Why classical LSTM, not a modification to enhance performance?

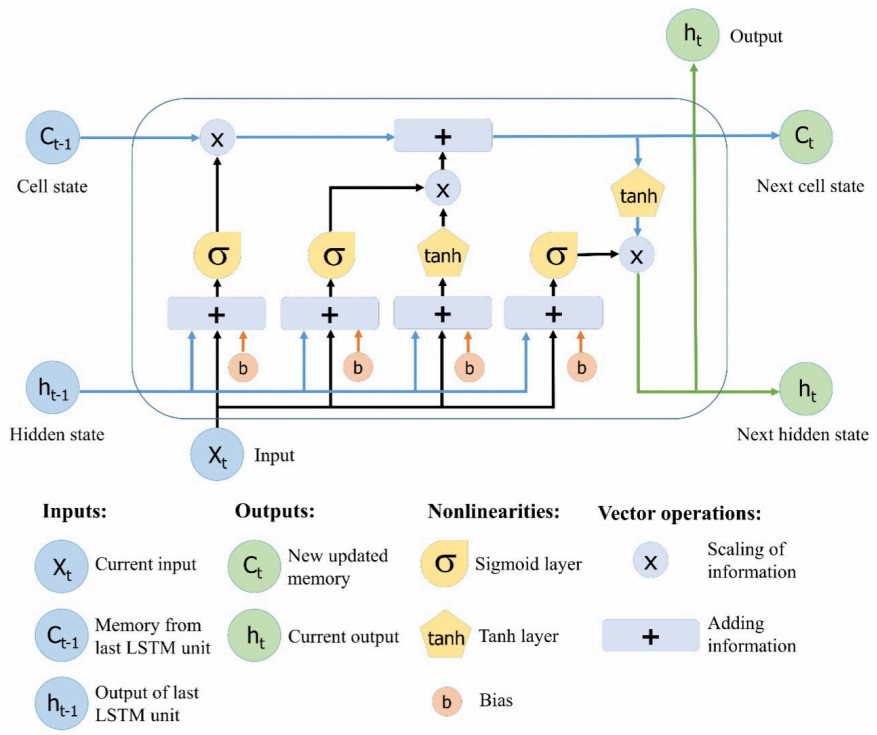


Figure 2.7: The structure of the Long Short-Term Memory (LSTM) cell. The figure is taken from Le et al. (2019). The network takes three inputs. X_t is the input of the current time step. h_{t-1} is the output from the previous LSTM unit and C_{t-1} is the "memory" of the previous unit, which is the most important input for sequential modelling. As for the outputs, h_t is the output of the current network. C_t is the memory of the current unit.

(specifically language modelling) has been found to be a superior architecture to other proposed modifications. Zaremba (2015) reported that an LSTM is better than one of its prominent alternative architectures, GRU, in a language modelling task. The experiments were based on ten thousand different RNN architectures. Britz et al. (2017) found that an LSTM is better than GRU for a machine translation task based on the variance values for several hundred experimental runs, corresponding to over 250,000 GPU hours on a standard English to German translation benchmark task held at a 'Workshop on statistical machine translation (WMT)'. Thus, conclusions of the language modelling task can be generalised to the other NLP tasks (machine translation), which have language modelling as the base component. Moreover, Greff et al. (2015) showed that none of the eight variants of LSTM neural cells on three representative tasks namely speech recognition, handwriting recognition, and polyphonic music modelling, can improve upon the standard LSTM with forget gate architecture significantly, and demonstrated that the forget gate (of an LSTM) and the output activation function are its most critical components, not the modification of the architectures. Their result is based on 5400 experimental runs ($\approx$ 15 years of CPU time [1]). These proposed novel architectures were developed to achieve improved performance that beat the state-of-the-art results. These extensions were shown as mandatory to achieve enhanced performance. However, in practice, an existing architecture with different hyperparameter settings could achieve or outperform the performance of the proposed new architecture.

RNNs give the ability of universal computation (Turing completeness, shown by Siegelmann & Sontag (1991)) to neural networks, in contrast to that of universal approximation (Hornik et al. 1989, Cybenko 1989) for feed-forward neural networks (FFNN). Turing completeness means that for any computable function, there exists a finite recurrent neural network (RNN) that can compute it. There are several extensions to the simple RNN models, such as Bidirectional recurrent neural networks (BI-RNN), etc, mainly to enhance the performance of a downstream application with limited data.

Although the RNN is Turing Complete and can approximate any process (algorithm), training an RNN model is challenging (Hochreiter 1991, Bengio et al. 1994). It was shown by Hochreiter (1998), Bengio et al. (1994) that the main reason for the training difficulty comes from the exploding and vanishing gradients. There are approaches (Pascanu, Mikolov & Bengio 2013) applied to reduce the severity of

[1] According to Greff et al. (2015, page 5) each of the 5400 experiments was run on one of 128 AMD Opteron CPUs at 2.5 GHz and took 24.3 h on average to complete. This sums up to a total single-CPU computation time of just below 15 years.

this problem. One of the most prominent is the LSTM (Hochreiter & Schmidhuber 1997).

Theoretically, a single layer RNN network should be able to approximate any computable function. However, it was observed *empirically* that deep RNNs work better than shallower ones in some tasks, specifically for natural language processing tasks. In particular, Sutskever et al. (2014) used a 4-layer deep architecture that was crucial in achieving good machine-translation performance in an encoder-decoder framework. To combat the computational challenges, a C++ implementation of deep LSTM was developed utilising 8 GPUs. Irsoy & Cardie (2014) also achieved improved results moving from a one-layer BI-RNN to an architecture with several layers. Many other works reports result using stacked layered RNN architectures, but most of them do not explicitly compare the improvement with a gradual increase in layers. In the next chapter, we will compare a single layer and two-layer neural models using publicly available datasets.

2.4.1 Neural Network Language Modelling

RNN based language modelling was pioneered by Mikolov et al. (2010), where recurrent neural networks were first introduced for language modelling. Since then, a number of improvements have been proposed. Zaremba et al. (2014) used a stack of Long Short-Term Memory (LSTM) layers trained with dropout applied on the outputs of every layer (except the recurrence layer), while Gal & Ghahramani (2016*b*) and Inan et al. (2017) further improved the perplexity score using variational dropout. Variational dropout applies to all layers, unlike the standard dropout. The RNN language model (Zaremba et al. 2014) is used by numerous other models including Press & Wolf (2016), Gal & Ghahramani (2016*b*). The model by Zaremba et al. (2014) was also used as a baseline in the very comprehensive (over ten thousand different RNN architectures) architecture search by Jozefowicz et al. (2015) in pursuit of a better architecture, and they failed to find architectures that were significantly better from the baselines.

The baseline architecture by Zaremba et al. (2014) is a two-layer stacked recurrent neural network. Stacking RNN layers (in space) inspired by the multilayer perceptron (MLP) allows for greater complexity by incorporating a complex feature representation of each layer (except the recurrent layer). There are approaches to increase the depth (in time) of the recurrence layer (Pascanu et al. 2014) contrary to the space extension like the stacked approach. RHN (Zilly et al. 2017) cells are one

such approach to increase modelling power utilising the recurrence layer and can be seen as a specific generalisation of LSTM. We will study RHNs in chapter 5.

2.4.2 The Problem of Very Slow Training Time

As mentioned in section 2.1.1, the two main problems with the RNN are long training time and limited capacity for effective long-term context capture. We will discuss the training time more elaborately here. Training an RNN is known to be very slow. For example, Mikolov et al. (2010) took several weeks to train the RNN model for language modelling, although the authors considered only about 17% of the New York Times (NYT) section of English Gigaword for training. The total training time is proportional to $2 \times P \times nH \times (nH + |V|)$ given a complexity of order $P \times nH \times (nH + |V|)$, where nH is the number of hidden units, P denotes the number of epochs needed to reach convergence and $|V|$ denotes the size of the vocabulary. Usually, it takes about 10–50 training epochs to achieve convergence (Mikolov, Kombrink, Burget, Cernocký & Khudanpur 2011, Mikolov, Deoras, Povey, Burget & Černockỳ 2011), although cases have been reported where even thousands of epochs were needed (Xu & Rudnicky 2000). Besides, the size of the vocabulary (i.e. $|V|$), which is usually very large for many languages, plays a crucial role in the real complexity of the training. With the use of GPUs, the training time has been considerably reduced from weeks to days and sometimes to hours. However, this reduction is still not sufficient for practical applications.

The main bottleneck for neural network training of the language modelling is the very long training time due to softmax normalisation at the output layer. The softmax function takes an vocabulary (V) sized dimensional vector of scores and limit the values into the range $[0, 1]$ as defined by the function:

$$p(w) = \frac{\exp(z_w)}{\sum_{w' \in V} \exp(z_{w'})}$$

where z_w is the logit corresponding to a word w. The logit is generally computed as an inner product $z_w = h^T e_w$ where h is a context vector generated by an deep neural network and e_w is a vector representation for w in the large parameter matrix in the output layer. The denominator of the softmax function is dependent on every element in the scores vector, and thus the time complexity of using the softmax function is $O(|V|)$. When using datasets with very large output spaces like language modelling where output space is equal to the size of the vocabulary(V), this can quickly become

a computational bottleneck during training time.

There are approaches to improve the training time by optimising the output layer including tree-based approaches such as hierarchical softmax (Morin & Bengio 2005), differentiated softmax (Chen et al. 2015), adaptive softmax (Grave, Joulin, Cissé, Jégou et al. 2017); sampling-based approaches such as importance sampling (Bengio & Sénécal 2003), adaptive importance sampling (Bengio & Senécal 2008), target sampling (Jean et al. 2014*a*), and self-normalising approaches such as noise contrastive estimation (Gutmann & Hyvärinen 2012), and self-normalisation (Andreas & Klein 2015*a*). Next, we will discuss the advantages and disadvantages of different approaches.

The tree-based approach, hierarchical softmax, is quite expensive during testing, requiring even more effort than softmax for computing the most likely next word (Chen et al. 2015), and requires a manual design for the intermediate latent nodes in the tree. Experimentally, more than two layers of a tree are computationally very expensive and do not computationally outperform softmax. Tree-based approaches are also very unfriendly to GPUs/mini-batching, every word in the batch has a different path and that makes GPU programming difficult. To address this problem, adaptive softmax (Grave, Joulin, Cissé, Jégou et al. 2017) has been proposed where word clusters are generated by utilising the unbalanced word distribution. This approach explicitly minimises computation complexity and makes the tree-based approaches more GPU friendly. This approach is based on clustering and the original paper uses between 2 and 5 clusters without specifying the impact of the number of clusters on the performance. For small numbers of clusters (between 2 and 5 in the original paper), significant degradation in performance compared to the full softmax was not observed. These approaches do not outperform softmax.

Differentiated Softmax (Chen et al. 2015) assigns a variable number of parameters to each word in the output layer. More parameters are assigned to frequent words compared to rare words, since more training occurrences allow for fitting more parameters. The vocabulary is partitioned according to word frequency and the words in each partition share the same embedding size, resulting in a sparse output layer (final weight matrix) which arranges the embeddings of the output words in blocks, each block corresponding to a separate partition. This method improves the train and test time complexities. The problem with this method is that by assigning fewer parameters to rare words the model does not model well the rare words and has the worst performance in modelling rare words.

The sampling-based approaches like importance sampling have stability and

consistency problems in practical applications. This approach is simple and theoretically related to Monte Carlo methods. Monte Carlo methods are computational algorithms that rely on repeated random sampling to obtain numerical results. A Monte Carlo method samples from a known distribution (proposal distribution) and corrects expectation of the actual distribution (true distribution) from the fact that the samples are from a wrong distribution (proposal distribution). However, the approach is **non-trivial** to use in practice because the **high variance** of the sampling estimates can make learning unstable (Mnih & Teh 2012). The variance tends to grow as learning progresses because the model (predicted) distribution moves away from the proposal distribution. When using neural language modelling and N-gram is chosen as a proposal distribution, Bengio & Senécal (2008) argue that the instability happens as neural language models and N-gram models learn very different distributions. Despite these difficulties, importance sampling-based approaches have been successfully applied in different domains, such as language modeling (Józefowicz et al. 2016), machine translation (Jean et al. 2014*b*) and computer vision (Joulin et al. 2016). None of these papers mentioned how the instability was handled to make the importance sampling method work well. Another sampling-based approach, target sampling, is faster and performs more iterations than softmax in a given time. However, its perplexity reduction per iteration is less than softmax (Chen et al. 2015). Thus, it takes longer than softmax to converge.

Self-normalised approaches aim at learning a naturally normalised classifier, to avoid computing the softmax normalisation. Popular methods are Noise Contrastive Estimation (Gutmann & Hyvärinen 2010, 2012), or a penalisation on the normalisation function (Andreas & Klein 2015*b*, Devlin et al. 2014). Noise Contrastive Estimation (Gutmann & Hyvärinen 2010) replaces the softmax by a binary classifier distinguishing the original (data) distribution form a noise distribution. While the original formulation still requires to compute the softmax normalisation, Mnih & Teh (2012), Vaswani et al. (2013), Zoph et al. (2016) showed that good performance can be achieved without this expensive computation and by setting the normalisation term to a predefined constant.

A common problem with sampling-based and self-normalising approaches is that they still require to evaluate the full softmax to be evaluated at test time. The main complexity of the neural language models, however, comes from training. Among many approaches, the Noise Contrastive Estimation (NCE) was shown not to be working with deep recurrent neural network in Chen et al. (2015). Theoretically, NCE (Gutmann & Hyvärinen 2010, 2012) is a consistent (convergent) estimator and it has better learning stability (the weights are always between 0 and 1) than

other sampling-based approaches including importance sampling. We will study the integration of NCE with deep neural models in chapter 4 and 5.

2.4.3 Limited Effective Context Size

We mentioned in section 2.1.1 that the two main problems with the RNN are longer training time and limited effective context capture. Now we will discuss the context capture capability of the neural models. To improve the context capture capability, many approaches have been proposed (Oualil et al. 2016, Wang & Cho 2016, Khandelwal et al. 2018, Peters et al. 2018). Specialised, multi-context integrated deep neural networks (Liao et al. 2018) were proposed when language modelling is used for location prediction. Theoretically, the basic RNN should capture all the previous context, but due to learning difficulty (the vanishing gradient problem), the RNN captures only a limited amount of context. Improving the learning process should enhance the model's capacity for capturing improved context information. We will study the context in chapter 4.

2.5 Generalisation and Regularisation

As we have described previously, optimisation in a mathematical setting is different from machine learning (ML). ML algorithms must perform well on new, previously unseen inputs, not just those on which our model was trained. The ability to perform well on previously unobserved inputs is called generalisation (Goodfellow et al. 2016). To measure the generalisation performance, we need both training and test performance measures usually through an error or accuracy metric. Concretely, when training a machine learning model, we have access to a training dataset; we can compute an error measure on the training set, called the training error; and we reduce this training error through optimisation. Until now, these steps are equivalent to a mathematical optimisation problem. What separates machine learning from a mathematical optimisation is that we want the test error (generalisation error) to be low as well. The generalisation error is a combination of two factors: bias and variance (Moody 1994, Girosi et al. 1995, Wahba et al. 1999).

The bias and variance of the ML models are related to model underfitting and overfitting. Underfitting corresponds to the fact that the model is not being able to obtain a sufficiently low training error and overfitting correspond to the large gap between the training error and test error (generalisation gap in Fig. 2.9). Models

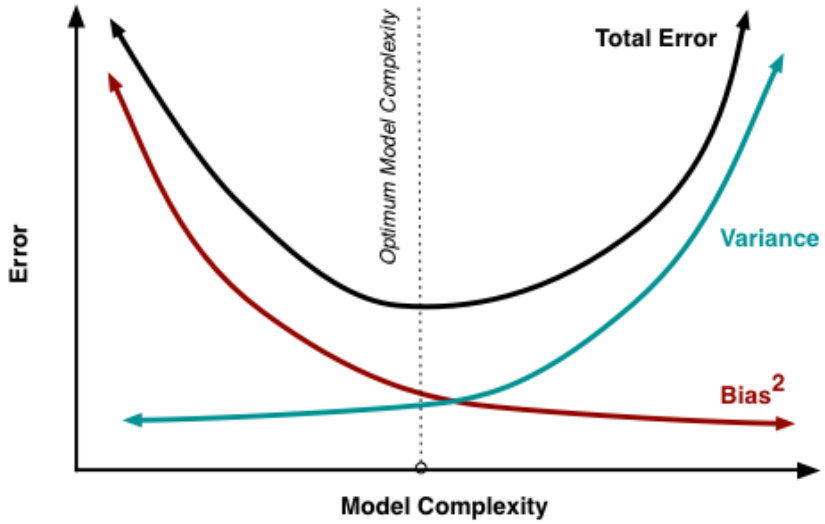


Figure 2.8: Learning curve: bias and variance contributing to total error (Neal et al. 2018).

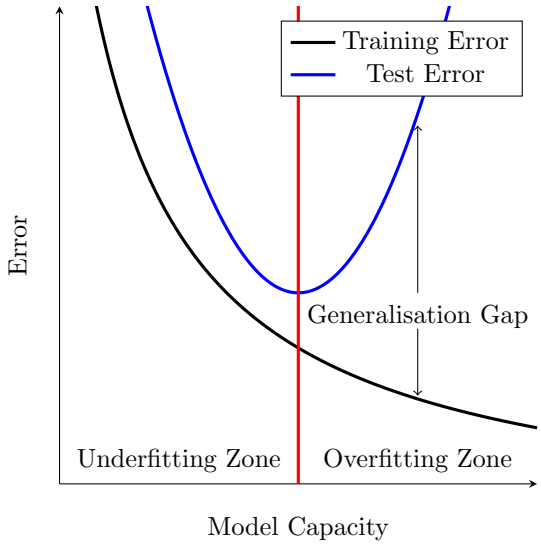


Figure 2.9: Typical relationship between model complexity (capacity) and error. Training and test error behave differently. At the left end of the graph, training error and test error are both high. This is the underfitting regime. As we increase capacity, training error decreases, but the gap between training and test error increases. Eventually, the size of this gap outweighs the decrease in training error, and we enter the overfitting regime, where capacity is too large, above the optimal capacity (Goodfellow et al. 2016, Fig. 5.3).

underfit when there exists high bias and overfit when there is high variance. Regularisation techniques 'regulate' complexity (i.e. discourage learning more complex than necessary) to avoid the risk of overfitting. Thus regularisation becoming to a process of modifying a learning algorithm to prevent overfitting. This generally involves imposing some sort of smoothness constraint on the learned model (Girosi et al. 1995). This constraint may be enforced explicitly, by fixing the number of parameters in the model, or by augmenting the cost function as in Tikhonov regularisation (also known as ridge regression (Hoerl & Kennard 1970), weight decay (Krogh & Hertz 1992)). Tikhonov regularisation along with many other regularisation schemes, fall under the umbrella of spectral regularisation (Engl et al. 1996). Early stopping (Morgan & Bourlard 1990, Finnoff et al. 1993) also belongs to this class of methods.

To explain underfitting, overfitting and regularisation with an example: let's consider that we are interested in developing functions to model data with quadratic form. If the data has quadratic form and we use a linear model then the model will underfit and have a high bias. For the same data, if we use a 50-degree polynomial model then the model will overfit and have high variance. For the later cases, the purpose of the regularisation techniques is to impose constraints to reduce the complexity of the 50-degree polynomial model to a simpler (smaller than 50-degree polynomial) one. We can test if models have high bias or variance through the use of learning curves (see Fig. 2.8, 2.9). We have utilised such learning curves in chapter 4 in our analysis.

2.5.1 Bias-Variance Tradeoff

The traditional view in machine learning is that increasingly complex models achieve lower bias at the expense of higher variance. This balance between underfitting (high bias) and overfitting (high variance) is commonly known as the bias-variance tradeoff (Moody 1994, Girosi et al. 1995, Wahba et al. 1999).

The bias-variance tradeoff of N-gram models is dependent on the choice of N. To choose a value for N in an N-gram model, it is necessary to find the right trade-off between the stability of the estimate against its appropriateness (bias-variance trade-off). A small N leads to a simpler (weaker) model, therefore causing more error due to bias. A larger N leads to a higher-order complex model causing an error due to variance. As a rule of thumb, a trigram is a common choice with large training corpora (e.g. millions of words in a language model training corpus), whereas a bigram is often used with smaller corpora.

Spectral learning for WFA is based on the law of large numbers' assumption. The parameters are often outside the parameter space resulting in negative probabilities. These models are theoretical and mathematical, and assume that an infinite dataset is available. However, in practical applications, this assumption does not hold. By infinite dataset, we mean the assumption of the optimisation in a mathematical setting that an infinite amount of data is available to learn the model parameters.

The bias-variance tradeoff for neural networks is discussed by Geman et al. (1992). The authors suggested that larger neural networks suffer from higher variance. This provided a intuition for how we think about the generalisation capabilities of large models and the intuition is supported by the learning theory. The learning theory-based study by Brutzkus et al. (2017) showed that the most classical and current bounds on generalisation error grow with the size of the networks. However, there is a growing amount of evidence that larger networks generalise better than their smaller counterparts in practice (Neyshabur et al. 2015*a*, Novak et al. 2018, Zhang, Bengio, Hardt, Recht & Vinyals 2017). This apparent mismatch between theory and practice is probably due to the use of worst-case analysis that depends only on the model class, is completely agnostic to data distribution (without knowledge of the data) and without taking optimisation (including the regularisation) into account. In practice, for domain-specific modelling, we know the data, and we also have access to the regularisation techniques which allows the larger models to generalise better. The larger models with regularisation can effectively be simpler models and this would align with the theory.

2.5.2 Generalisation in Deep Neural Networks

Deep artificial neural networks often have far more trainable model parameters than the number of samples they are trained on. Despite this fact, successful deep neural networks with this over-parametrisation can exhibit a remarkably small difference between training and test performance, thus demonstrating good generalisation. Nonetheless, some of the neural models demonstrate small generalisation error, whereas, at the same time, it is certainly easy to come up with natural model architectures that generalise poorly. It is not clear what makes some neural networks generalise better than others. Neural network generalisation has been extensively studied both in theory and in empirical research. The theoretical approaches to generalisation include classical notion such as Vapnik–Chervonenkis (VC) dimension (Baum & Haussler 1989, Harvey et al. 2017), Rademacher complexity (Sun et al. 2016), and

modern concepts such as stability (Hardt et al. 2016), robustness (Xu & Mannor 2012), compression (Arora et al. 2018*a*).

Existing studies found that optimisation and learning theory is unable to explain and predict the generalisation properties of deep neural networks (DNN). For example, from the earliest days of DNNs, it was suspected that VC theory does not apply to these systems (Vapnik et al. 1994). Vapnik et al. (1994) assumed that local minima in the loss surface were responsible for the inability of VC theory to describe neural networks, and the reason was justified by the fact that neural network training can get stuck to the local minima and that limits the number of possible functions realisable by the network. However, it was later realised that the presence of local minima in the loss function was not a problem in practice (LeCun et al. 1998, Duda et al. 2012). There is still no theoretically grounded approach to explain the generalisation of the deep neural network (Martin & Mahoney 2019). Overall, the generalisation of deep neural models is not completely understood in the literature.

2.5.2.1 Generalisation and Optimisation

There are studies on the relation between generalisation and properties of stochastic gradient descent (SGD) (Robbins & Monro 1951, Kiefer et al. 1952) algorithms (Zhang, Liao, Rakhlin, Sridharan, Miranda, Golowich & Poggio 2017, Soudry et al. 2018, Advani & Saxe 2017). Empirical studies (Zhang, Bengio, Hardt, Recht & Vinyals 2017) have shown that while deep neural networks are expressive enough to fit randomly labelled data, they can still generalise for data with structure. Smith & Le (2018) showed that the same phenomenon occurs in small linear models. Inspired by the results of Zhang, Bengio, Hardt, Recht & Vinyals (2017), Arpit et al. (2017) proposed that the data dependence of generalisation in neural networks can be explained because they tend to prioritise learning simple patterns first. The authors showed some empirical evidence supporting the statement (i.e. prioritise learning simple patterns first), and suggested that SGD might be the origin of this **implicit regularisation**. It was also shown that implicit regularisation induced by the optimisation method is playing a key role in generalisation of deep learning models (Neyshabur 2017).

Implicit Regularisation.

The stochastic gradient descent (SGD) demonstrated significantly better generalisation (i.e. maximises the test data accuracy) than batch methods (Bousquet & Bottou 2008). It was also shown by Wilson et al. (2017) that SGD generalises better than the adaptive optimisers (e.g. Adam (Kingma & Ba 2015), AdaGrad (Duchi et al. 2011)). The noise plays an important role in SGD convergence and generalisation of

neural network learning (Jim et al. 1995). It was also observed that noise introduced by the mini-batch helps SGD to generalise better given that optimum batch size is utilised (Smith & Le 2018). In chapter 4, we have empirically shown that correct utilisation of the 'search-then-converge' learning rate schedule in the SGD can result in improved generalisation when the output (linear + softmax) layer of the neural language model is approximated using NCE. The improved generalisation can be explained by the sampling noise induced by the NCE approximation.

2.5.2.2 Understanding Regularisation in Deep Neural Networks

Historical regularisation techniques usually reduce the effective number of degrees of freedom that are being fitted by the models resulting in effectively simpler models. For example, L1 regularisation aka LASSO (Least Absolute Shrinkage and Selection Operator) (Tibshirani 1996) can force some parameters/weights to be near zero and sparse PCA (Zou et al. 2006) applies LASSO, for example, to learn principal components (PCs) with sparse weights. LASSO is also used in low-rank regularisation (Hu et al. 2018, Tai et al. 2015) that forces the smallest singular values of the parameter matrix (or equivalently eigenvalues of its covariance matrix) to be near zero. When the eigenvalues become zero, the rank of the parameter matrix is reduced. When the smallest eigenvalues are not exactly zero but are sufficiently close to zero, approximate low rank is used to compare the regularisation effect of machine learning techniques. Another regularisation technique, L_2, pushes the parameters of the machine learning models towards a prior value, which is usually zero, to regularise them (Bengio 2012).

Low-rank regularisation.

The idea of having a bias towards simpler models and simpler patterns has a long history. For instance, the concepts of minimum description length (MDL) (Rissanen 1978), Blumer algorithms (Blumer et al. 1987, Wolpert 2018), and universal induction (Li & Vitányi 2013), are all based on a bias towards simple hypotheses. Very recent work of Pérez et al. (2019) has used a PAC-Bayes approach to explain the generalisation of deep neural networks on real-world problems by arguing that parameter-function map in a deep neural network is biased towards simple functions.

Probably Approximately Correct (PAC) learning framework.

Bengio (2012) argued that it is sufficient to regularise only the output layer (i.e. linear+softmax layer) weights to constrain capacity to simpler models. In general, adding a penalty term to the training criterion to regularise deep learning is common in the literature, specifically to the hidden nodes (Krogh & Hertz 1992). These penalty terms encourage the hidden units to be sparse, i.e. with values at or near 0. There

are many benefits of having a sparse representation in a deep neural network including disentangling the underlying factors of representation and regularise the overall network (Tibshirani 1996, Schmidhuber 2015). Thus increased sparsity can be compensated by a large number of hidden nodes (i.e. large neural models) resulting in an effectively simpler model. Increasing sparsity at the output layer can also result in overall simpler models and help regularise the whole network.

One of the reasons why deep learning methods work well, without explicit regularisation even in the strongly over-parametrised regime where classical learning theory would instead predict that they would severely overfit, maybe the fact that they allow for large numbers of parameters (through deep networks), but at the same time their implicit regularisation techniques (e.g. SGD (Lei et al. 2018), Dropout (Gal et al. 2017)) can exert certain simplicity constraints (through sparsity or other mechanisms) on the parameter space. For example, dropout (Srivastava et al. 2014) discourages co-adaptation of the neurons, which also leads to sparse activations (Srivastava et al. 2014); sparse activations also exist in sparse coding (Olshausen & Field 1996). In chapter 5, we have shown that NCE primarily works on output layers and can be seen as a regulariser for the deep neural language model.

Overall, from the discussion in this section, we can conclude that we do not completely understand (i.e. there are uses of the 'maybe' and 'can be' words in many sentences) the generalisation behaviours of the deep neural network. In an attempt to explain the generalisation behaviour observed in practice, several approaches have been proposed. For example, Neyshabur et al. (2015*b*), Keskar et al. (2016), Neyshabur, Bhojanapalli, McAllester & Srebro (2017), Bartlett et al. (2017), Neyshabur, Bhojanapalli & Srebro (2017), Golowich et al. (2017), Arora et al. (2018*b*) suggested different norm, margin and sharpness based measures. Dziugaite & Roy (2017) numerically evaluated a generalisation bound based on PAC-Bayes. As argued by (Neyshabur, Bhojanapalli, McAllester & Srebro 2017), complexity measures and generalisation bounds provided by Bartlett et al. (2017), Dziugaite & Roy (2017) increase with the increasing network size, thus fail to explain why over-parametrisation helps. Overall, existing complexity measures increase with the size of the network depending on the number of hidden units either explicitly, or the norms in their measures implicitly depend on the number of hidden units for the networks used in practice (Neyshabur, Bhojanapalli, McAllester & Srebro 2017). Generally, the existing literature cannot measure and explain the generalisation of deep neural networks. To be able to successfully explain the generalisation, we believe that we have to first understand the generalisation. In chapter 5, we have taken an empirical approach to understand the generalisation behaviour and propose a variance-based approach to

explain the regularisation.

2.6 Problem Analysis

Based on the discussion in this chapter so far, there are different research questions for which we do not know the answer yet. The first question that we ask is what is the best model for sequence modelling and specifically for language modelling. To achieve the goal of addressing this question, we will explore three methods (i.e. N-gram, WFA and Neural Networks) on 15 publicly available SPiCe sequence prediction competition datasets (Balle et al. 2017). As they are competition datasets, different methods have already been applied independently to discuss the best models. For example, one competition team has only experimented with neural network models and N-grams, whereas other team has used only WFAs. However, there was no study where these three methods were applied under a controlled experiment that would fulfil our goal of analysing the model and dataset characteristics. For example, there is no team that has applied these three methods on 15 datasets and reported the total 45 scores so that we could use the results to analyse the characteristics of the datasets and models. To analyse models and datasets, in chapter 3, we will compare three different families of models on 15 datasets. In chapter 3, we will also discuss the reasons for choosing these three particular families of models.

Based on the discussion in this chapter, we can conclude that deep neural networks require expressive power for sequence modelling specifically for language modelling where the observation space is large and regularities are complex. Historically, the main problem with deep neural models has been training difficulty. One of the reasons that make the training process inefficient is that it is very difficult to understand the overall training in the presence of many hyperparameters. There is no standard way of justifying the impact of a particular component in a neural network. It is also difficult to understand or analyse neural models as a whole due to the black box nature of the neural network-based modelling. To improve the training of deep neural language models, enhancing the understanding of neural models is crucial. To increase understanding of the neural models and potentially reduce the number of parameters of neural models, we need to explore the impact of different components in neural networks. There are many components in neural networks and it would be too ambitious to explore all the components in the PhD study. The neural network theoretical results show that one layer of the recurrent neural network is very powerful. However, it was also shown in the literature that multiple layers bring benefits in

RNN training for the sequence modelling task like language modelling. In chapter 3, we will explore the impact of multiple layers in neural networks that has been found in the literature as a key to success for many applications.

Even with the single layer RNN network, the main computational bottleneck of the neural training is the softmax function, and there exist different approaches to approximate the softmax operation. Among different approximation techniques, a common problem with sampling-based and self-normalising approaches is that they still require to evaluate the full softmax at test time. The main complexity of the neural language models, however, comes from training. Among many approaches, the Noise Contrastive Estimation (NCE) was shown to be not working with deep recurrent neural networks in Chen et al. (2015). It was also argued to be an inferior technique for neural language modelling because it optimises a binary rather than a multitask classification task (Józefowicz et al. 2016) and importance sampling was found to be a more appropriate for neural language modelling. Theoretically, NCE (Gutmann & Hyvärinen 2010, 2012) is a consistent (convergent) estimator and it has better learning stability (the weights are always between 0 and 1) than other sampling-based approaches including importance sampling. We will study the integration of NCE with deep neural models in chapters 4 and 5. Our objective is to analyse the neural language modelling learning to explore how a component (i.e. NCE) can be integrated into deep neural language models and explore the potential of the component for neural language modelling. Another important factor that we want to establish is that it is essential to work on the method which has good theoretical properties, but was found to be inefficient when integrated with deep neural network. In most fields, publication with negative results for a method get discouraged by the researchers to work on any further. In NLP this problem is exacerbated by the near-universal focus on improvements in benchmarks. A negative result is when the outcome of an experiment or a model is not what is expected or when a hypothesis does not hold. Despite being often overlooked in the scientific community, negative results are still results and they carry value. While this topic has been extensively discussed in other fields such as social sciences and biosciences, less attention has been paid to it in the deep learning community. Considering a negative result known in the literature we can formulate a research question to answer. We can ask how to make NCE work efficiently when it has been found inferior in the existing literature. Our investigation implies that before inventing a new method, it would be a good idea to check existing inefficient methods with rigorous analysis.

Much of the misunderstanding and contradictory results in the deep learning literature is due to the fact that we do not have tools to understand the neural

network training processes or learned models. We have seen in this chapter (see section 2.5.2) that generalisation in deep learning has not been understood completely and classical approaches to explain regularisation do not quite fit with the over-parametrised deep learning paradigm. We need to explore the overall impact of a component (e.g. NCE). Given the performance of a component on the improved generalisation capacity of the neural models, we need techniques to understand implicit regularisation. We believe that it is essential to design such analytical techniques for deep neural networks, and we will introduce novel techniques to improve the understanding of the neural models in chapter 5. As the proposed method will work on learned models, other methods in deep neural network that work as a implicit regularisation technique can be benefited from the proposed approaches in the chapter 5.

Conclusion

In this chapter, we have discussed the background and the relevant literature review for language modelling, deep learning and the fundamental concepts in machine learning that are exploited in this research. In the next chapter, we will compare language modelling methods on 15 different datasets.

3

A COMPARISON OF SEQUENCE PREDICTION METHODS

The previous chapter reviewed different methods for language modelling. Language modelling is a special and challenging case of a sequence prediction task where the observable symbols are linguistic units include characters, words, sentences, and whole documents (Rosenfeld 2000). In this chapter, we will compare the performance of sequence prediction methods including the language modelling methods described in the previous chapter.

Comparing different sequence prediction methods specifically for the language modelling task is very difficult, if not impossible, just by comparing their theoretical description. Each language modelling technique is well motivated and some of them have theoretical guarantees on their optimality under certain assumptions. The main problem with such methods is that the assumptions are not usually satisfied with real datasets (Quattoni et al. 2017, section 1.1).

The sequence prediction task is more general than language modelling in the sense that the observable symbols can be from any domain and the regularities involved in the process can be diverse. The different regularities help us understand the different requirements for a dataset to make accurate predictions. For example, Natural Language Processing (NLP) datasets have long term dependency for prediction, whereas Software Engineering datasets (e.g. datasets to evaluate the effectiveness of various verification and validation approaches on reactive systems) do not have a similar requirement. Moreover, when we do not use the domain knowledge associated with the data, we cannot facilitate the process of learning by utilising prior knowledge (Yu et al. 2010). Although incorporating domain knowledge is challenging, successful incorporation of prior knowledge improves both the quality of the result produced by the learning process and the efficiency of the learning process itself. When comparing methods for different domains, to incorporate prior knowledge is very challenging. It takes many years of experience to become a domain expert and multiple domain expertise is a rare calibre. Thus, it is difficult to be fair for all domains when incorporating prior knowledge by a person who has more expertise in one domain than another. Using diverse domain synthetic and real datasets without

incorporating prior knowledge we thus evaluate methods based on the data-driven approach, revealing the inherent strength and weakness of the methods. Applying multiple methods on the same dataset also helps us understand one method with the interpretability of another method. For example, neural networks are difficult to analyse but Weighted Finite-state Automata (WFA) and N-grams are easier to analyse, and applying these methods on the same dataset help us understand different aspects of the neural network models and the learning process.

Data driven approach

Therefore, in this chapter, we will use publicly available competition data from different domains, including NLP, Biology and Software Engineering, to evaluate methods. We will compare results from models generated using three methods (N-gram, WFA and neural networks) on sequence prediction tasks of the SPiCe sequence prediction competition (Balle et al. 2017) using the 15 publicly available datasets that include five NLP datasets. Based on the discussion in the previous chapter, we will use N-grams with smoothing, a spectral algorithm for WFA, and LSTM (see section 2.4) in the neural models. The competition data were preprocessed to hide the domain-related information by the competition organisers (Balle et al. 2017) and observable symbols were replaced with unidentifiable common symbols (e.g. integer values). The goal is to identify the best model that works for different datasets from different domains without using domain knowledge. To the best of our knowledge, there is no previous work where diverse datasets were explored (and analysed) with these methods (N-gram, WFA and Neural Network) through a controlled experiment.

The description of these 15 datasets is given in section 3.1. The task of the competition and its evaluation criteria are described in section 3.2. The related methods used in the competition are described in section 3.3 and all the models were evaluated using unseen test data and according to the SPiCe evaluation metrics. Our approach to the task is described in section 3.4, the experimental setting is described in section 3.5 and the result and analysis in section 3.6.

The main contributions of this chapter are filling the knowledge gap by :

1. a comprehensive evaluation of the spectral WFA method on real datasets and a comparison with N-gram and two variations of recurrent neural network models (RNN). There is no previous comprehensive evaluation available (including hyperparameter search) of spectral WFA on synthetic and real datasets. Moreover, we present the results using both single-layer and two-layer RNN neural networks. An empirical investigation of the diverse domain datasets with the recurrent neural models (emphasising the impact of the number of layers in the

results) has not been done in the past.

2. exploring spectral based WFA algorithms in a combination of a real and synthetic context to understand the hidden state relationship with multiple layers in recurrent neural networks. In the existing literature, multiple layers are chosen for getting improved accuracy without explaining the reason behind such a choice. Our experiments and analysis show that the hidden states of a process can be modelled efficiently using multiple layers in recurrent neural models.

3.1 Data Description

In this section, we introduce the 15 datasets used in the Sequence PredIction ChallengE (SPiCe) in 2016. The datasets consist of 8 (fully or partially) synthetic and 7 real-world datasets. Among the synthetic datasets, four are generated artificially and four are partially synthetic based on real data. These datasets are publicly available[1] and their descriptions can be found in (Balle et al. 2017). The synthetic datasets 1, 2, and 3 were artificially generated based on a Hidden Markov Model (HMM) (Balle et al. 2017). HMM sequences were generated with n states and non-stationary transition probabilities were obtained by partitioning the unit interval [0,1) into n equal length sub-intervals and letting the states evolve as $h_{t+1} = h_t + \Phi \bmod 1$, for some irrational number Φ. [2] The emission probabilities were sampled from a Dirichlet distribution. Another synthetic dataset, 12, consists of synthetic data generated using the PAutomaC data generator (Verwer, Eyraud & dela Higuera 2014). Partially synthetic datasets 6 and 9 are based on software engineering and come from the challenge RERS 2013 (Howar et al. 2014). Partially synthetic datasets 14 and 15 contain synthetic data generated from two Deterministic Finite-state Automata learned using the ALERGIA algorithm (Carrasco & Oncina 1994) based on the NLP datasets 4 and 5, respectively.

Synthetic and partly synthetic datasets

Real datasets 4 (English Verbs from Penn Treebank), 5 (Character Language Modelling benchmark from Penn Treebank), and 8 (POS from Ancora) all correspond to NLP problems from Penn Treebank (Marcus et al. 1993*a*) and the Spanish Ancora corpus (Taulé et al. 2008). Dataset 11 (lemmatisation) was created from a lemmatised version of the Flickr-8k dataset (Hodosh et al. 2013). Real dataset 13 (spelling correction) was derived from a Twitter spelling correction corpus (*TYPO CORPUS* 2010). Real datasets 7 and 10 are protein family sequences taken from the Pfam database

Real dataset

[1] http://spice.lif.univ-mrs.fr/data.php

[2] Multiples of an irrational number modulo 1 cover the unit interval uniformly (Katz 2004).

Table 3.1: Dataset Descriptions — The column # gives a number to the dataset, the column Sym gives the number of different symbols, Train and Test provide respectively the number of elements in the training and test sets, and Type details the source of the data.

#	Sym	Train	Test	Type
1	20	20000	5000	synthetic (non-stationary HMM with 2 states)
2	10	20000	5000	synthetic (non-stationary HMM with 2 states)
3	10	20000	5000	synthetic (non-stationary HMM with 4 states)
4	33	5987	749	NLP (English verbs, character level, Penn Treebank)
5	49	33654	4207	NLP (character level language modeling, Penn Treebank)
6	60	5000	5000	partly synthetic, software engineering (RERS 2013 problem 34)
7	20	65438	5000	biology (protein family PF13855, full set, Pfam)
8	48	13903	1738	NLP (Spanish simplified POS sentences, Ancora)
9	11	5000	5000	partly synthetic, software engineering (RERS 2013 problem 42)
10	20	54932	4848	biology (protein family PF00400, RP15 subset, Pfam)
11	6722	32384	4048	NLP (English lemmas from Flickr-8000)
12	21	200000	3000	synthetic (PAutomaCgenerator)
13	702	26544	3318	NLP (English spelling correction from Twitter Typos Corpus)
14	27	10000	5000	partly synthetic (ALERGIA, DFA based on problem 4)
15	32	50000	5000	partly synthetic (ALERGIA, DFA based on problem 5)

(Finn et al. 2015).

3.2 Problem Statement and Evaluation Criteria

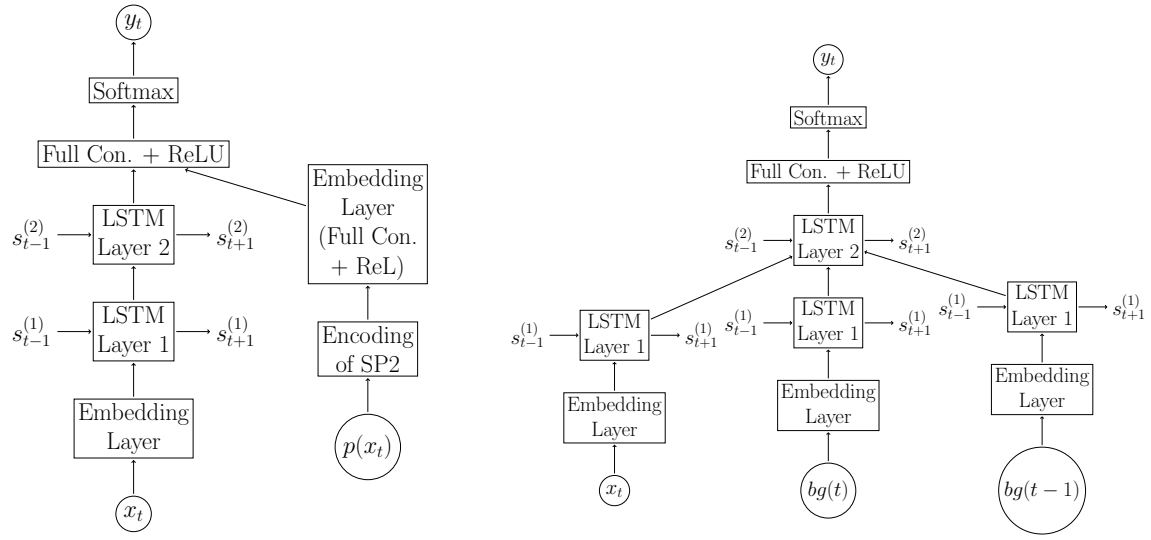
The Sequence PredictIction ChallengE (SPiCe) (Balle et al. 2017) was an on-line competition to predict the next element of a sequence. The competition scored methods on their performance on both real and synthetic data (see section 3.1). Training datasets consist of whole sequences and the aim is to learn a model that allows the ranking of potential next symbols for a given test sequence (prefix or context), that is, the most likely options for a single next symbol. Once rankings for all prefixes were submitted by the participants, the score (normalised discounted cumulative gain NDCG_5, explained below) of the submission was computed. The score is a ranking metric based on normalised discounted cumulative gain computed from the ranking of 5 potential next symbols starting from the most probable one. Suppose the test set is made of prefixes $y_1, \ldots, y_M$ and the distinct next symbols ranking submitted for y_i is $(\hat{a}_1^i, \ldots, \hat{a}_5^i)$ sorted from more likely to least likely. The target probability distribution of possible next symbols given the prefix y_i, $p(.|y_i)$, was known to the organisers. Thus, the exact measure for prefix y_i could be computed using the following equation:

$$\mathrm{NDCG}_5(\hat{a}_1^i, \ldots, \hat{a}_5^i) = \frac{\sum_{k=1}^{5} \frac{p(\hat{a}_k^i|y_i)}{\log_2(k+1)}}{\sum_{k=1}^{5} \frac{p_k}{\log_2(k+1)}}$$

where $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_5$ are the top 5 values in the distribution $p(.|y_i)$. More details on this evaluation can be found in (Balle et al. 2017).

Why a new evaluation metric?

The competition was open for all methods, and that requires a generalised evaluation metric. In the previous chapter, we have seen that perplexity is an evaluation metric for language modelling. However, the spectral based solutions do not necessarily assign a non-negative number to each sequence, much less adds up to one when summed overall sequence, although both properties are satisfied in the limit (Balle et al. 2014). In practice, this is a problem when trying to evaluate these methods using perplexity as an accuracy measure. Therefore, the above-mentioned evaluation criterion was used in the competition.



(a) SP2 model (b) Bigram model

Figure 3.1: The best neural architectures for the datasets in the literature

3.3 Related Works

To find the most efficient methods for the 15 datasets, the total of 26 teams implemented a wide range of methods including different types of neural network models, boosting, spectral and classical state-merging algorithms for learning weighted automata, and ensemble methods that combined several techniques. Among them, the top six teams presented their work as a report. The top score was achieved by Shibata & Heinz (2017). This score was achieved by using a combination of novel RNN models where the state vector is augmented with an indicator vector representing the previous N-gram in history. Specifically, they have used three models, the basic model was a stacked LSTM followed by an all-connected layer. The second and third models used the combined representation learnt by stacked LSTM and input vectors corresponding to the states of particular deterministic finite-state automata (DFA). The second model, SP2 model (see Fig. 3.1a), used a vector embedded with the current state of a Strictly 2-Piecewise DFA (Heinz & Rogers 2010) and the third model, bigram model (see Fig. 3.1b), used a vector embedding the state of a Strictly 2-Local DFA (bigram model) (McNaughton & Papert 1971). The third model, basic model (see Fig. 3.3a), is a two-layer stacked LSTM network.

The second-best team used an ensemble of Multilayer Perceptron, Convolutional Neural Network, LSTM and N-gram models (Zhao et al. 2017). The third team also used an ensemble method of N-gram, spectral, RNN and tree boosting (Sato et al.

2017). The fourth team (Liza & Grześ 2017) used a combination of a spectral model with N-gram models. The fifth team (Xi & Zhuang 2017) used model compression techniques. Team seven (Hammerschmidt et al. 2016) has used (non-)probabilistic deterministic finite-state automata for the given datasets. In this chapter, we will compare our proposed methods with the top-scoring method proposed by Shibata & Heinz (2017).

3.4 Method Used For the dataset

In the previous section, we have seen that numerous approaches have been applied to the sequence prediction task. In this section, we will describe the specific approaches for the three methods and their learning approach that were applied to the datasets.

3.4.1 N-gram with smoothing

We have described N-gram models and smoothing (Chen & Goodman 1996) techniques in the previous chapter (section 2.3.1). Based on that discussion, for the datasets in section 3.1, we have used the N-gram with modified Kneser-Ney (MKN) smoothing (Kneser & Ney 1995). The MKN smoothing is a recursive interpolation with lower order models, making use of different discount values for more or less frequently observed events.

3.4.1.1 Relevant parameters

The main hyperparameter for the N-gram model is the value of N, defining the length of the sequences to consider in estimating the probability. In the previous chapter, we have seen that for N-gram models, the number of parameters to be estimated increases exponentially with the hyperparameter N. We have also discussed that the impact of the choice of N is related to the bias-variance trade-off. The performance dependency relative to N is dependent on the training data available for the prediction. The additional hyperparameters for the MKN smoothing are, D_1, D_2 and D_{3+}, defining the discount parameters for N-gram with one, two and greater than three counts respectively. These discount parameters are used in linear interpolation of higher and lower order N-gram probabilities to preserve probability mass.

Our approach to parameter tuning

We have increased the values of N gradually, starting from the unigram and used best found value for N when the performance on the public testing dataset (i.e. validation dataset) stopped improving. The final score was based on a separate private test datasets only available to the competition organiser. The values of the smoothing parameters D_1, D_2 and D_{3+} can be set efficiently using the following approach (Koehn 2010, chap 7):

$$Y = \frac{N_1}{N_1 + 2N_2}$$
$$D_1 = 1 - 2Y\frac{N_2}{N_1}$$
$$D_2 = 2 - 3Y\frac{N_3}{N_2}$$
$$D_{3+} = 3 - 4Y\frac{N_4}{N_3}$$

Here, N_c are the counts of N-gram with exactly c count in training dataset. We have used this approach to set the values of these hyperparameters.

3.4.2 WFA

In the previous chapter, we described WFA as a good approximator for sequence prediction. Approximating distributions over strings is a hard learning problem and in the previous chapter, we have discussed the learning difficulty of WFA. The recent advancement in learning for WFA is based on spectral learning which reduces the computation complexity. In the previous chapter we have seen the spectral algorithm (section 2.3.2) for WFA (Balle et al. 2014). Next, we will briefly describe the steps of the Hankel matrix-based spectral algorithm for WFA:

S1. Basis Selection: Choose a set of prefixes P and suffixes S

S2. Build a Hankel matrix: The Hankel matrix ($H_f \in \mathbf{R}^{\Sigma^* \times \Sigma^*}$) associated with a function $f : \Sigma^* \to \mathbf{R}$ is a bi-infinite matrix. In practice, one deals with finite sub-blocks of the Hankel matrix based on the chosen basis in step S1, thus $B = (P, S) \subset \Sigma^* \times \Sigma^*$. The corresponding sub-block of the Hankel matrix is denoted by $H \in R^{|P| \times |S|}$. The entry $H(p, s)$ is the value of the target function on the sequence obtained by concatenating prefix p with suffix s. Among all possible basis, we are particularly interested in the ones with the same rank as f. We say that a basis is complete if $rank(H) = rank(f) = rank(H_f)$.

S3. Perform SVD on $H = U\Sigma V^\top$.

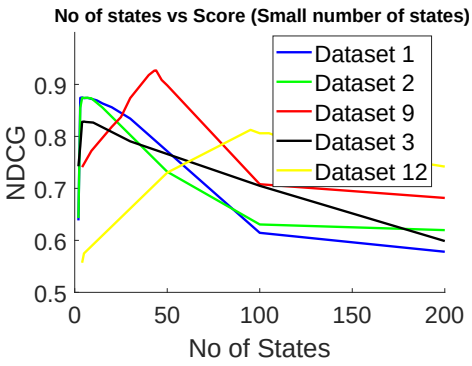
S4. Use the factorization $F = U\Sigma$, $B = V^\top$ and H to recover the parameters of the minimal WFA, following (Hsu et al. 2012, see section 2.3) and the description of the previous chapter (see section 2.3.2).

3.4.2.1 Relevant Parameters

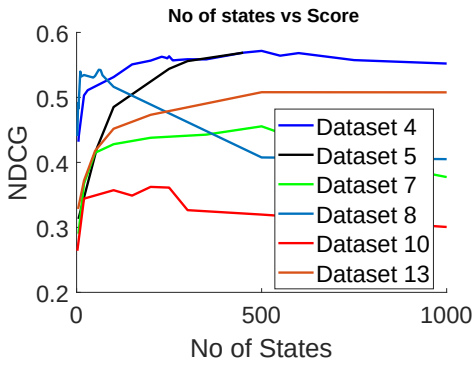
The algorithm receives as input the number of states n of the target WFA. This n is a hyperparameter for this algorithm and defines the rank of the Hankel matrix which defines the number of hidden (latent) states in the low dimensional state space. If we relate the literature of subspace identification methods for linear systems with the spectral methods, then we are trying to find a basis for the state space such that operators in the new basis are related to observable quantities.

The noise and the insufficient (not sufficiently informative) data can make the rank of $\widehat{H}$ different from the true rank of H_f. When the value of n is selected through the cross-validation procedure, a truncated SVD of $\widehat{H}$ up to dimension n is used to get the factorisation. By truncating some small singular values of $\widehat{H}$ are ignored. The algorithm has to ignore some small singular values of $\widehat{H}$, which correspond to zeros in the original matrix, as the zeros in the original matrix do not have much impact on the final result. Bailly (2011) has shown that when empirical Hankel matrices are sufficiently accurate, singular values of $\widehat{H}$ can yield accurate estimates of the number of states n in the target.

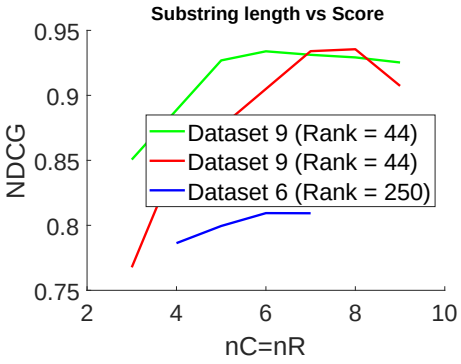
The other important parameter to choose when using the algorithm is the basis. In practice, choosing a basis can be done in the form $P = S = \Sigma^{\leqslant k}$ for some $k > 0$ (Hsu et al. 2012, Siddiqi et al. 2009). Another approach is to choose a basis that contains the most frequent elements observed in the sample, which can be either strings, prefixes, suffixes, or substrings (Balle et al. 2012). We used the second approach in our method with substrings. Note that the basis vector can be chosen from a subblock of the Hankel matrix where the rows and columns of the Hankel matrix correspond to the substrings and the cells of the Hankel matrix contain the frequencies of the corresponding substrings. In the algorithm, we also choose the length of these substrings. Instead of using substrings, it is also possible to use the prefixes as a row and suffixes as a column. In such a case, the cell of the Hankel matrix can be calculated as the frequencies of the corresponding strings. If the data is informative enough and the frequencies are high enough, the Hankel matrix gives a complete basis without the costly need to look at all possible rows and columns.



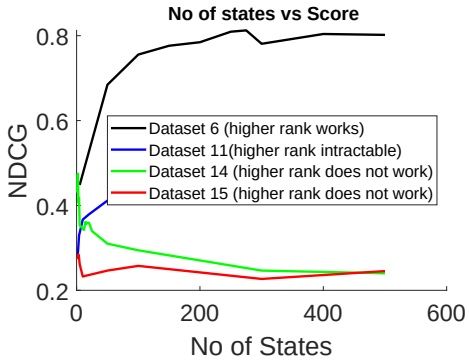
(a) Impact of basis on score



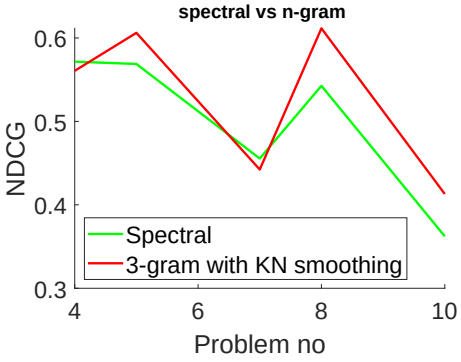
(b) Low Score (SM)



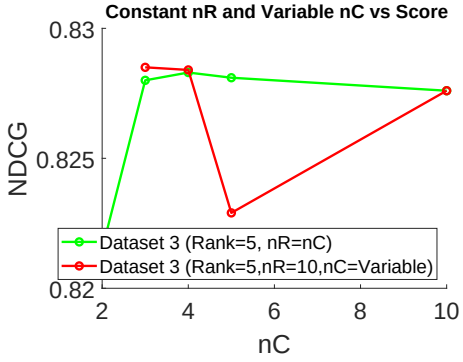
(c) Impact of basis on score



(d) No of states vs Score



(e) SM vs n-gram



(f) Same and different nR, nC

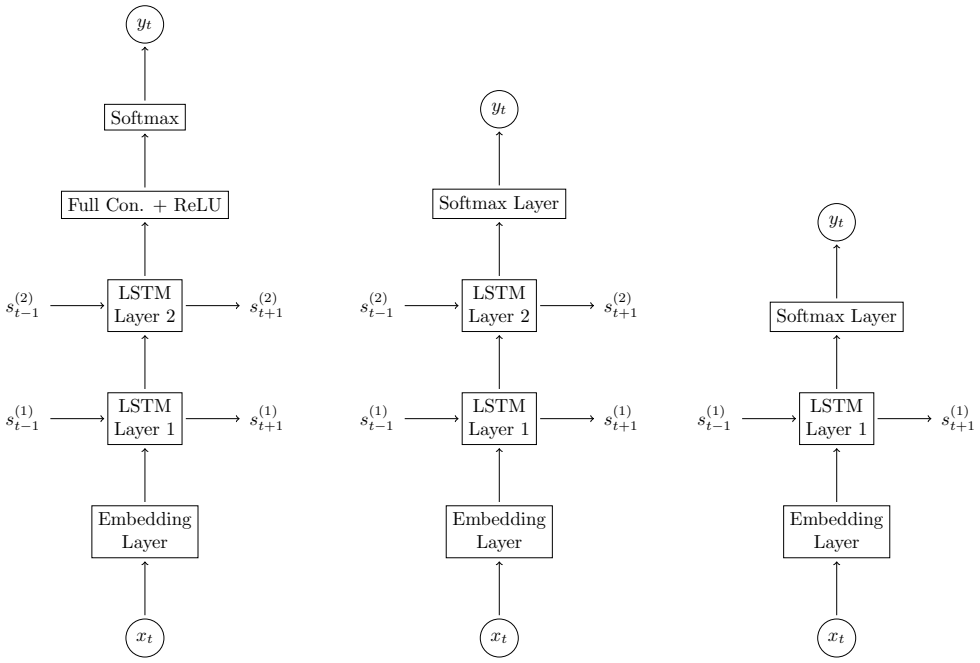
Figure 3.2: Spectral Learning parameter learning

Our Approach to Parameter Tuning

Our main task now is to find the best hyperparameters of the spectral algorithm of WFA. Specifically, we had to select values for three parameters: the number of states, n, the maximum length of the substring considered in the row of the Hankel matrix, nR, and the maximum length of the substring considered in the column of the Hankel matrix, nC. The objective is to maximise the score 'Normalised Discounted Cumulative Gain' ($NDCG_5$) on the datasets (section 3.1).

For the hyperparameter search, we did not apply any of the more sophisticated methods for hyperparameter tuning—such as random search (Bergstra & Bengio 2012), Bayesian optimisation (Bergstra et al. 2011) or grid-based search (Larochelle et al. 2007)—because the method quickly becomes infeasible when the rank, i.e. the number of states or the length of the substrings is too high, and we had to stop many experiments manually. Similar to Larochelle et al. (2007), our method included a combination of multi-resolution search, coordinate ascent, and manual search, with a significant utilisation of the last method on public test data (i.e. validation dataset).

On all problems, our method first initialises nR and nC to 4 and n to 5. Note that the number of rows (columns) in the Hankel matrix is much larger than nR (nC). In the second step, the algorithm starts the process of tuning the number of states n because this was the most important hyperparameter in our preliminary experiments. A random walk is used to select new values of n with the step size being dependent on the size of the domain, i.e. the number of observations and the number of sequences. Thus, when nR and nC were kept constant, the value of n was increased or decreased randomly based on the $NDCG_5$ score (see Figure 3.2b), i.e. a form of coordinate ascent was performed on n. After the highest score was achieved by tuning n, n was frozen, and the algorithm used the same randomised procedure to tune nR (see Figure 3.2c). Finally, the same procedure was executed to tune the parameter nC (see Figure 3.2f). After tuning nR, and nC, we did not tune n again because, for a given n, a very small improvement was usually observed after tuning nR and nC. On some problems, increasing the values of n, nR and nC to a large number was not possible as the algorithm was becoming intractable. All these hyperparameter tuning were done on public test data (i.e. validation dataset). The final scores in the result section are based on the organisers' private test data which was not available to contestants during hyperparameter tuning stage.



(a) Basic SPiCe (b) Basic 2-layer (c) Basic 1-layer

Figure 3.3: Neural architectures for the datasets

3.4.3 Neural Network

In the SPiCe competition, there were three neural models (SP2 model, bigram model and basic model) explored by Shibata & Heinz (2017) that achieved the winning accuracy. Among those models, the basic model is a two-layer stacked LSTM network. There is an all-connected non-linear layer with a Rectified Linear Unit (ReLU) activation function used on top of a stacked LSTM (Fig. 3.3a). The two-layer LSTM stack was placed on top of the embedding layer that is used to embed each symbol x_t of the sequence at position t. The output layer consists of the softmax layer implementing the softmax activation, which outputs the network's prediction of the next symbol of the sequence $y_t = x_{t+1}$.

In this chapter, we have simplified the basic architecture in two ways. First, we removed the fully-connected non-linear layer and introduced dropout at all non-recurrent layers (Fig. 3.3b). Second, we further simplified the model by using just a single layer (Fig. 3.3c) with dropout at non-recurrent layer. The motivation for removing the all-connected layer is to reduce the number of parameters and the state-of-the-art sequence prediction models (Zaremba et al. 2014) do not usually have an all-connected non-linear layer on top of the stacked LSTM. Stacked LSTMs are expres-

sive enough to capture most of the regularities without an additional all-connected non-linear layer. The motivation for applying the dropout at all non-recurrent layers is to regularise the whole networks (Zaremba et al. 2014) instead of regularising based on some particular layers (Shibata & Heinz 2017). We compare against a single-layer LSTM in the results section.

Following Shibata & Heinz (2017) a 'start' symbol and an 'end' symbol were added to both sides of each training sentence. Symbols are fed into the model from the 'start' symbol.

3.4.3.1 Relevant Parameters

Neural models have more hyperparameters than the other models (e.g. N-gram, WFA). Setting the values of hyperparameters can be seen as model selection, i.e. choosing which model to use from the hypothesised set of possible models. Hyperparameters are often set by hand, selected by some search algorithm as described in the previous section (section 3.4.2.1), or optimised by utilising a hyper-learner or meta-learning approach. For the deep learning models, there are mainly three types of hyperparameters. First, model design hyperparameters: these parameters specify the building blocks of the model including the number of layers, types of activations, types of optimisation techniques. Second, the learning and regularising hyperparameters: these parameters include the learning rate, dropout rate. Third, the model hyperparameters: these hyperparameters help to learn the model parameters (weights of the network learned from the data) with stability including the weight initialisation scheme, gradient thresholding (gradient norm).

Our Approach to Parameter Tuning

In this chapter, we will experiment with model design hyperparameters and the learning and regularising hyperparameters. We will use one layer and two layer networks to see the impact of increased modelling power on the performance. We will also experiment with dropout at different layers. The rest of the hyperparameters are used on the previous study by Shibata & Heinz (2017). Our work in this chapter can be extended by experimenting with the other hyperparameters, however, our goal has been served by the tuning of the above-mentioned hyperparameters.

3.5 Experimental Settings

For the N-gram models, we have used the 3-gram model after experimenting with uni-gram till 5-grams. The MKN3 smoothing was used and the relevant hyperparameters (D_1, D_2, D_{3+}) were set according to the unigram, bigram and trigram statistics of the respective datasets. The hyperparameters (See Tab. 3.4) for the WFA were tuned based on the grid search approach (described in the section 3.4.2.1) to find the best results obtained in this chapter. The specific values can be found in the Tab. 3.4.

For the neural models, the weights were initialised with Gaussian samples, each of which has zero mean and deviation $\sqrt{\frac{1}{\texttt{in\_size}}}$, where $\texttt{in\_size}$ is the dimension of input vectors. The LSTM has 600 hidden nodes, the size of embedding vectors was set to 100, and when a non-linear layer was used in between the LSTM and Softmax layer (for the baseline replication), the output dimension was set to 300. These values were set based on the baseline study.

We have used two-layer and one layer LSTM networks for the dataset. The dropout rate was set to 0.5 for all non-recurrent layers, which is known to be close to optimal for a wide range of networks and tasks (Srivastava et al. 2014).

Following baseline study, for optimisation, we used the momentum stochastic gradient descent (SGD) with the momentum 0.9. The learning rate decreased gradually from 0.1 to 0.001, where the number of iterations was 45 and the mini-batch size was 128.

3.6 Results and General Analysis

In this section, we will report the experimental results which are based on the evaluation score described in section 3.2 and analyse the results. Tab. 3.3 reports the scores of the three methods.

In this chapter, for the neural models (in Tab. 3.3 and Tab. 3.5), we report average accuracy (mean statistics) and standard deviation (SD) (Garg & Mohanty 2013) in the 'Mean (SD)' format. Each experiment was repeated 10 times with different random seeds and the results were averaged. Tab. 3.2 reports the results of the three neural models (Fig. 3.1a, 3.1b, 3.3a) proposed by Shibata & Heinz (2017). From the scores, we can see that the performance of the basic model is quite competitive with the sp2 and bigram models. The sp2 model scored better than the basic model in

Table 3.2: Comparison of Scores Between SPiCE Neural Models (Shibata & Heinz 2017, Tab. 2)

Dataset	basic(600)	sp2(600)	bigram(600)
1	0.909(0.002)	0.915(0.000)	0.769(0.003)
2	0.920(0.000)	0.920(0.000)	0.838(0.004)
3	0.888(0.001)	0.886(0.001)	0.831(0.001)
4	0.619(0.002)7	0.616(0.002)	0.634(0.001)
6	0.863(0.001)	0.867(0.001)	0.828(0.002)
7	0.736(0.000)	0.736(0.001)	0.747(0.001)
8	0.645(0.001)	0.644(0.001)	0.614(0.001)
9	0.962(0.000)	0.962(0.000)	0.959(0.000)
10	0.574(0.001)	0.573(0.001)	0.570(0.002)
11	0.520(0.001)	0.519(0.001)	-
12	0.799(0.002)	0.807(0.001)	0.713(0.001)
13	0.592(0.001)	0.590(0.001)	0.581(0.000)
14	0.350(0.002)	0.351(0.002)	0.333(0.002)
15	0.263(0.001)	0.263(0.001)	0.258(0.001)

datasets 1,6,12,14 in small-scale (i.e. accuracy improvement is $\leqslant$ 0.008). Similarly, the bigram model scored better than the basic model in datasets 4,7 in small scale (i.e. accuracy improvement is $\leqslant$ 0.015). From these results, we can see that the increased model complexity (additional parameters) have little impact on the resulting accuracy. We will compare the SPiCe basic model with the proposed basic models in Tab. 3.5.

In Tab 3.3, the scores of the datasets (14, 15) are low for all used methods, specifically, none of the methods could score at least 0.5. In fact, in the literature, there are no known methods for these two datasets that have scored at least 0.5. Therefore, we are excluding the analysis of the result for those datasets. We assume that the datasets are too complex or not informative enough or the given datasets are just too small for any methods to predict meaningfully.

As described in the previous chapter (chapter 2), the N-gram models are still the baseline for the sequence prediction task, specifically for the language modelling task. Therefore, WFA and the neural models will be compared with N-gram models. We will compare and analyse the results of these three methods in the following paragraphs.

Table 3.3: Comparing Three Methods on the Sequence Prediction Task

Dataset	N-gram	WFA	Two Layers NN Mean (SD)
1	0.8424	0.8789	0.9179 (0.0004)
2	0.8204	0.8731	0.9208 (0.0001)
3	0.7719	0.8248	0.8937 (0.0006)
4	0.5540	0.5272	0.6103 (0.0019)
5	0.6142	0.5688	0.8116 (0.0007)
6	0.7003	0.8096	0.8709 (0.0016)
7	0.5020	0.4474	0.7303 (0.0020)
8	0.6233	0.5426	0.6616 (0.0017)
9	0.8662	0.9324	0.9666 (0.0004)
10	0.3965	0.3623	0.5523 (0.0027)
11	0.4092	0.4147	0.5525 (0.0019)
12	0.6992	0.8113	0.8512 (0.0013)
13	0.4737	0.4990	0.6014 (0.0013)
14	0.3552	0.4649	0.3549 (0.0038)
15	0.2524	0.2978	0.2659 (0.0011)

Comparing N-grams and Neural models: When comparing between the neural models and the N-gram models (see Tab. 3.3), neural models have outperformed the N-gram models in all datasets. It is expected, as we have described in the previous chapter that N-gram models require much more data than neural models to achieve the same performance.

Comparing N-grams and WFA models: From the experimental results in Tab. 3.3, the WFA model performed worse than the N-gram models in five datasets (4,5,7,8 and 10) and better than the N-gram models in the eight datasets (1,2,3,6,9,11,12, and 13).

We will analyse the results of these three methods (N-gram, WFA and neural network) in the following paragraphs.

Datasets where N-gram models are better than WFA models: We will first discuss the results where N-gram models did better than WFA models. When compared with the WFA models in Tab 3.3, for datasets 4,5,7,8 and 10, the N-gram models perform better than the WFA models. Three of these datasets are from the NLP domain (4, 5 and 8) and two from biology (7, 10). These domains are known

Table 3.4: The hyperparameters for the WFSA

Dataset	Rank (n)	lrows (nR)	lcolumns (nC)	Score
1	4	5	5	0.8789916635
2	6	5	5	0.8731489778
3	5	10	3	0.8248148561
4	500	5	5	0.5272911191
5	450	5	5	0.5688513217
6	90	6	7	0.8096061349
7	500	4	4	0.4474728703
8	60	5	5	0.5426079151
9	57	8	7	0.9324635267
10	200	5	5	0.362334120
11	100	5	5	0.4147772193
12	95	4	4	0.8113699555
13	500	5	5	0.4990697801
14	2	10	10	0.4649848044
15	3	6	6	0.2899561226

to have long-distance dependencies on context or history for sequence prediction. Although an N-gram performs better than a WFA, we have described previously that the N-gram models scored low compared to the neural models. This is expected as we have argued in the previous chapter that neural models can capture long term dependency efficiently and learning N-gram and WFA models requires larger datasets than neural models to converge.

WFA-based sequence modelling requires three hyperparameters (n, nC, nR). Among them, n defines the rank or cardinality of the hidden space, whilst nC and nR define how much history to consider. From Fig. 3.2b, it can be seen that for the datasets 4,5,8,10, and 13, only increasing the number of hidden states (n) does not show significant improvement in the score. For these datasets, the value for the hidden state n has been increased until 1000 and no performance increment is observed after 400.

Based on the other two hyperparameters (nC, nR), two types of observation are made: the learning does not converge (low score) or the learning becomes intractable. To elaborate on the above observation, we will now analyse the results. If the datasets have a small number of training examples, increasing the length of the substring does not improve the score. Dataset 4 (Sym =33, # of training examples =

5987) have this characteristic. For this dataset, the substring length of 5 gave the best result (0.5272) with rank 500. Increasing the rank to 856 and the length up to 70 does not improve the score. *We assume, one of the reasons for such an observation can be related to the size of the dataset. Dataset 4 is small and increasing the substring length will result in a Hankel matrix with many zero entries. We need the Hankel matrix to be informative (non-zero entries) enough to make meaningful predictions (learning convergence).* In such a case, as learning has not converged, we do not know if the data has a long term dependency or not.

While small training examples impose the convergence problem, a large dataset can make the method intractable (computationally infeasible), specifically, if the dataset has a long-distance dependency on the context to make predictions. Datasets 5, 7, 8, 10 have comparatively larger numbers of training examples than dataset 4 but increasing the length of the substring makes the method intractable. For dataset 5, using the number of states (n) $[5, 55, 100, 250, 300, 350, 450]$ gave the evaluation scores of $[0.313, 0.421, 0.485, 0.544, 0.555, 0.560, 0.568]$ with nC and nR of size 5. Increasing the number of hidden states, the score improves slightly. However, increasing the string length to 6 with a gradual increase in the number of hidden states to 350 makes the method intractable. Similarly, for dataset 7 (Sym =20, # of training examples = 65438), the best score achieved by WFA is 0.4474 with rank 500 and string length 4. Increasing the string length to 5 makes the learning process computationally infeasible. For dataset 8 (Sym = 48, # of training examples = 13903), the reported score was found with $n = 60$ and $nC = 5$, $nR = 5$, increasing the nC and nR to 6 makes the method intractable. Note that with $n = 55$ and $nR = nC = 6$ the score was 0.485. Following a similar pattern, for dataset 10, the best-achieved score is 0.3623 with rank 200 and string length 5. Increasing the length over 5 made the method intractable. To summarise, for predictive modelling of these datasets, we have to consider the use of a large string length in the Hankel matrix. Increasing the values of the length increases the model complexity and makes the learning process infeasible for WFA. Therefore, to model datasets with long term dependency, WFA suffers from computational complexity. In the previous chapter, we have seen that spectral learning of WFA has better computation complexity than the non-spectral counterpart, however, for datasets with long term dependency, the learning process is still intractable (computationally infeasible) with spectral algorithms.

Datasets where WFA models are better than N-gram models: We will analyse the datasets where N-gram does worse than WFA. For datasets 1,2,3,6,9,11,12, and 13 WFA performed better than N-gram models. Although WFA has done better than N-gram in dataset 11 and 13, the scores of both methods are low (< 0.5)

and we are excluding our analysis for these datasets. Datasets 1, 2, 3, 6, 9, and 12 where the spectral algorithms for WFA led to good scores (see Tab. 3.4), have common characteristics. All these datasets are generated synthetically (partially or fully), the data are not from the NLP domain, have small numbers of hidden states (can be expressed through low-rank approximation) and with short length substring, the scores are competitive. From the above discussion, it can be concluded that for synthetic datasets where the generating distribution is known or definable and prediction is dependent less on the longer distance context, WFA is a good method for sequence prediction.

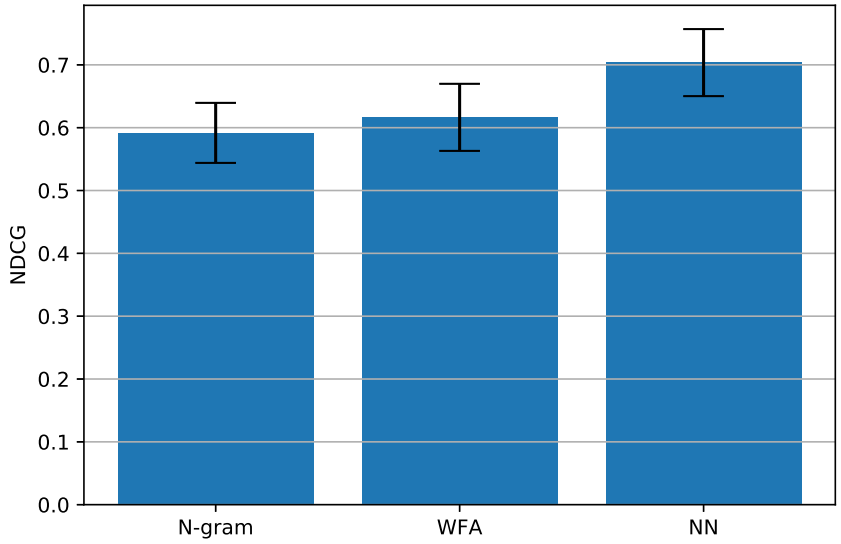


Figure 3.4: A bar chart with (standard) error bars (shown as black lines) of the models NDCG scores in Tab. 3.3. The (standard) error bars was calculated using the standard error of the mean (SEM). SEM = $\frac{\sigma}{\sqrt{15}}$, where σ is the standard deviation of the NDCG scores. The height of the bar represents the mean value of the NDCG scores.

Overall, neural models did better than N-gram and WFA models: From the above discussion, we have seen that neural models did better than N-gram and WFA in all datasets. To conclude, datasets where modelling requires longer-distance context (i.e. datasets that have long term dependency) are difficult to model with WFA and N-gram models. For such datasets, neural models are more appropriate. Evidently, from Tab 3.3, we have seen that a two-layer neural model performed better than WFA and the N-gram model in all datasets, including the ones which have long term dependency in the prediction.

To check the statistical significance based on the 15 scores of three families of models, we will use Tab. 3.3 which reports the scores of the three methods and Fig. 3.4

which shows the (standard) error bars. From the Tab. 3.3 and Fig. 3.4, we can say that accuracy of neural networks over the other two methods has large magnitude. The error bars in Fig. 3.4 indicate that the neural networks can be better than N-gram with statistical significance, however, neural networks is probably not better than WFA with statistical significance, but the increase in magnitude is not negligible.

3.7 Specific Analysis: The Impact of Multiple Layers in NN

In this section, we will do a specific analysis to understand the impact of multiple layers in recurrent neural networks. In this chapter, we used two types of RNN models (one is a single-layer and another is a two-layer stacked RNN network) and a WFA. Using the fifteen datasets and the results from the two types of neural model and WFA models, we will attempt to answer the following question "what is the impact of multiple RNN layers in sequence modelling?". To answer this question, we contrasted the impact of the LSTM layers in RNNs with the rank (i.e. the number of hidden states) in the corresponding WFA models.

Why Deep Recurrent Neural Networks? In chapter 2 we have seen that multilayer RNNs are advantageous for efficient sequence modelling (Zaremba et al. 2014, Jozefowicz et al. 2015). However, it is hard to analyse such models theoretically. As a result, in spite of competitive empirical results, it is not clear what kind of additional modelling power is gained by a deep architecture (i.e. more than one hidden layer in RNNs). Stacking RNN layers (in space) is inspired by the multilayer perceptron (MLP) and can be motivated with respect to the hypothesis by Bengio et al. (2009) that multiple layers allow the model to have greater complexity by incorporating complex feature representations of each time step. For the non-recurrent networks, Bengio et al. (2009) hypothesise that a deep, hierarchical model can be exponentially more efficient at representing some functions than a shallow one. Theoretical (Le Roux & Bengio 2010, Delalleau & Bengio 2011, Pascanu, Montufar & Bengio 2013) and empirical (Goodfellow et al. 2013, Hinton et al. 2012) work on non-recurrent networks agrees with the above hypothesis. Based on these results, Pascanu et al. (2014) assumed that the MLP-based hypothesis proposed by Bengio et al. (2009) is also true for recurrent neural networks. The earlier work attempted to capture large context and reducing the training time by using multilayer RNNs. For example, El Hihi & Bengio (1996) assumed that the layers increase the capacity of learning the context by capturing the improved long-term history, whereas Schmidhuber (2008) argues

that the stacked RNN requires less computation per time-step and far fewer training sequences than a single-layer RNN.

Depth of a Recurrent Neural Network Elman (1990) introduced the notion of 'memory' to capture non-fixed long-term contexts through the recurrent layer. When stacking the RNNs, the transition between the consecutive recurrent layers is still shallow (Pascanu et al. 2014). Thus, stacking the RNNs does not extend the hypothesis of (Bengio et al. 2009) to the recurrent layer that is dedicated to long-term context capture. The empirical results of Zaremba et al. (2014), Jozefowicz et al. (2015) suggested that multilayer RNNs improve sequence modelling. We show empirical evidence that indicates that a multilayer RNN does capture better context as shown by El Hihi & Bengio (1996), but that is achieved across stacked layers instead of the time scale (i.e. instead of the recurrent layer). Better learning is dependent on both capturing improved input representation at each time step and capturing improved long-term dependency from the previous time-steps in a sequence. In this chapter, we investigate RNN learning from the formal language perspective using WFA models, and we show that adding more layers may not be sufficient if the model has to deal with long-term dependencies.

WFA and RNN Weighted Finite-state Automata (WFA) and Recurrent Neural Networks (RNNs) provide a general framework for the representation of functions that map strings (i.e. sequential data) to real numbers. As we have mentioned earlier WFA-based models are used for both theoretical studies and sequence prediction tasks including language modelling (Buchsbaum et al. 1998). However, they are computationally expensive to train. On the other hand, RNNs are remarkably expressive models but their theoretical analysis is difficult even for a single-layer RNN. Evaluating their performance on real and synthetic data can help us to understand the WFA model's hidden state relationship with the recurrent neural network layers. In the existing literature, multiple layers are used in RNNs to obtain improved accuracy on sequence prediction tasks, but this is done without deeply-justified reasons for such a choice. Our experiments and analysis show that the hidden states of a process can be modelled efficiently using multiple layers.

The main goal of our empirical investigation is to show that correlating the impact of multiple layers in RNN-based neural models with the number of hidden states (quantified using a rank of the SVD) of finite-state automata, we can increase the understanding of deep neural networks. This way we aim to explain the role of multiple RNN layers in sequence modelling. In this discussion, we will refer

to Tab. 3.5 that reports the scores of the three neural network models described in section 3.4.3 and to Tab. 3.4 that reports the scores of the WFA models described in section 3.4.2.

Table 3.5: Comparing Sequence Prediction Scores (Basic Architecture)

Dataset	2 Layers (SPiCe) Mean (SD)	Two Layers Mean (SD)	One Layer Mean (SD)
1	0.909 (0.002)	0.9179 (0.0004)	0.8524 (0.0022)
2	0.920 (0.000)	0.9208 (0.0001)	0.9183 (0.0008)
3	0.888 (0.001)	0.8937 (0.0006)	0.8821 (0.0018)
4	0.619 (0.002)	0.6103 (0.0019)	0.6142 (0.0041)
5	0.8100 (0.001)	0.8116 (0.0007)	0.7972 (0.0013)
6	0.863 (0.001)	0.8709 (0.0016)	0.7806 (0.0023)
7	0.736 (0.000)	0.7303 (0.0020)	0.7113 (0.0015)
8	0.645 (0.001)	0.6616 (0.0017)	0.6506 (0.0017)
9	0.962 (0.000)	0.9666 (0.0004)	0.9538 (0.0009)
10	0.574 (0.001)	0.5523 (0.0027)	0.5615 (0.0025)
11	0.520 (0.001)	0.5525 (0.0019)	0.5413 (0.0006)
12	0.799 (0.002)	0.8512 (0.0013)	0.7127 (0.0010)
13	0.592 (0.001)	0.6014 (0.0013)	0.5384 (0.0017)
14	0.350 (0.002)	0.3549 (0.0038)	0.3594 (0.0036)
15	0.263 (0.001)	0.2659 (0.0011)	0.2661 (0.0014)

Table 3.6 shows that the proposed one-layer neural network model has achieved competitive results compared with the two-layer stacked network on many of the SPiCe datasets. The additional layer in a two-layer model improved the score most significantly on dataset 12, where the score improved from 0.711 to 0.851 (0.14 units). Another dataset where improvement was observed was dataset 6, where the score improved by 0.088 units. On datasets 1 and 13, the improvements were 0.065 and 0.065 units. In addition to those bigger improvements, a two-layer stacked RNN achieved slight improvement in the score ($\leqslant$ 0.02) on datasets 2, 3, 5, 7, 8, 9, 10, and 11. Still, a one-layer network did better than at least one of the two-layer networks (i.e. SPiCe and the one proposed in this chapter) on datasets 4, 8, 10, and 11. Overall, we can see that a one-layer RNN would be a better choice for some of our datasets, although using multiple layers leads to better predictions on other datasets. This means that to gain deeper insight into the behaviour of the methods, it is useful to investigate individual datasets in detail and include WFA in our analysis. For this reason, to shed some light on the impact of multiple layers in RNNs, we will analyse

Table 3.6: The hyperparameters of WFA, the scores of WFA and neural models (2 Layers (2L) and 1 Layer (1L)), and the score improvement by the best neural model compared to WFA

Data	WFA				Neural Models (NN)			RNN Improv.
#	n	nR	nC	WFA	SPiCe NN	RNN (2L)	RNN (1L)	Gain in score
1	4	5	5	0.8789	0.909	**0.9180**	0.8521	0.0391
2	6	5	5	0.8731	0.920	**0.9210**	0.9183	0.0479
3	5	10	3	0.8248	0.888	**0.8938**	0.8819	0.0690
4	500	5	5	0.5272	**0.619**	0.6131	0.6142	0.0918
5	450	5	5	0.5688	0.8100	**0.8107**	0.7988	0.2419
6	90	6	7	0.8096	0.863	**0.8690**	0.7815	0.0594
7	500	4	4	0.4474	**0.736**	0.7258	0.7176	0.2886
8	60	5	5	0.5426	0.645	**0.6614**	0.6521	0.1188
9	57	8	7	0.9324	0.962	**0.9674**	0.9546	0.0350
10	200	5	5	0.3623	**0.574**	0.5526	0.5604	0.2117
11	100	5	5	0.4147	0.520	**0.5535**	0.5412	0.1388
12	95	4	4	0.8113	0.799	**0.8508**	0.7116	0.0395
13	500	5	5	0.4990	0.592	**0.6007**	0.5357	0.1017
14	2	10	10	**0.4649**	0.350	0.3496	0.3616	-
15	3	6	6	**0.2899**	0.263	0.2655	0.2651	-

datasets 12 and 5 in the subsequent paragraphs. The reason for this choice is that on dataset 12, the score improved significantly using two layers, whereas on dataset 5 a similar improvement was not observed.

Dataset 12 and High Rank The synthetic dataset 12 was the biggest and arguably the most challenging problem in SPiCe 2016. It was initially generated for another competition (PAutomaC) using the PAutomaC data generator (Verwer, Eyraud & De La Higuera 2014). The best performing WFA scored 0.8113 on this dataset with $n = 95$ and $nR = nC = 4$. Although, WFA is a Markov model[3] (i.e. a model that may require l-th order representation, which makes predictions based on l the most recent observations, to learn long-range dependencies (Bengio & Frasconi 1995*b*, Hinton et al. 2001, Kakade et al. 2016)), on dataset 12, WFA was as good as the RNN models. Our one-layer neural model scored 0.7116 and the two-layer neural model

[3]Note that WFA is a generalisation of a Markov model where the next state depends only on the current state (Penagarikano & Bordel 2004); every Markov model can be represented as a WFA.

improved the result to 0.8508. So, we can see that on this large dataset, two layers improve the results. We argue that we can use WFA results to explain the improvement of our two-layer neural model. For that, we will focus on the rank of WFA (i.e. parameter n) and the maximum length of substrings in its basis (i.e. nR and nC). To score high on dataset 12, WFA had to use 95 hidden states, which is a large number of hidden states for a traditional Baum-Welch algorithm (Siddiqi et al. 2007). This means that to solve this problem, a Markov model requires a relatively large number of states. This fact can explain why our two-layer neural model outperformed a single-layer model because the second LSTM layer increased the number of hidden states in the neural model. Moreover, the obtained WFA's score is based on short substrings (i.e. $nR = nC = 4$) in its basis. Therefore, it is fair to expect that dataset 12 does not have long-term dependencies since short substring statistics are sufficient to capture the data-generating distribution in this model. We believe that we can make this claim because our WFA with short substrings works very well on this data. All this means that dataset 12 requires a relatively large number of hidden states, but it does not have long-term dependencies. Our two-layer neural model is sufficient for such problems because two layers increase the number of hidden states, whereas the long-term dependencies are not an issue.

To support our discussion above, we should add that in (Rabusseau et al. 2018), the hidden units of the second-order RNN were shown to be related to the rank or the hidden states of WFA. Note that second-order and higher-order RNNs have their recurrence depth increased by explicit, higher-order multiplicative interactions between the hidden states at previous time steps and input at the current time step. It was shown that any function that can be computed by a linear second-order RNN (Giles et al. 1992) with n hidden units on sequences of one-hot vectors (i.e. canonical basis vectors) can be computed by a WFA with n states. A higher-order RNN has additional connections from multiple previous time steps, whereas the classic RNN has connections from one previous time step only. Higher-order RNNs allow a direct mapping to a finite-state machine (Giles et al. 1992, Omlin & Giles 1996). However, a similar connection is not available for classic RNNs and WFA, and more importantly for multilayer RNNs and WFA. Based on these theoretical results and our empirical investigation, we can conjecture that the improved score on dataset 12 by using two LSTM layers indicates that the multiple layers helped to model the hidden states more efficiently.

Low Rank To support our arguments about the rank (i.e. the number of hidden states) in our discussion about dataset 12, we can identify a complementary relation-

ship in other results. In particular, when we consider all datasets on which WFA did well having a small rank (this is in contrast to dataset 12 which required a high rank for WFA), a two-layer network does not lead to significant improvement. This pattern can be seen on datasets 1, 2, and 3, and this complements our previous arguments about dataset 12.

Dataset 5 and Long Context Dataset 5 is a real dataset on which the best performing WFA model scored 0.568 with rank $n = 450$ and substring lengths $nR = nC = 5$. This dataset is large for spectral learning and increasing nR and nC above 5 made the method intractable. Our one-layer neural model scored 0.7988 and a two-layer model showed a small improvement scoring 0.8107. This means that on this dataset adding more layers did not change the score significantly. We will attempt to explain the lack of a big improvement of a two-layer neural model using our WFA results.

Dataset 5 corresponds to the NLP character language modelling benchmark from Penn Treebank (Marcus et al. 1993*b*). The other NLP datasets are 4, 8, 11, and 13. Similar to dataset 5, increasing the number of RNN layers did not significantly improve the score on those datasets. Most NLP data (including dataset 5) have long-term dependencies because there are many training examples of word agreements (with different long-range regularities) which span a large portion of a sentence (Brown & Hinton 2001). WFA with discrete states has limited memory capacity which gets consumed by having to deal with all the intervening regularities in the sequence. We can see this in our results, because, in our experiments on WFA, we have many hidden states ($n = 450$). We can see that a large number of hidden states was not sufficient to solve this problem using WFA when $nR = nC = 5$, i.e. when substrings are short. To capture long-term dependencies, our WFA would need to be trained on longer substrings (higher nR and nC), but this is infeasible to do on this large dataset because the method becomes intractable. This problem requires the learning algorithm to take care of the long-term context.

We can provide a theoretical justification as to why long substrings (i.e. prefixes and suffixes that define the basis of a Hankel matrix) can lead to a better model given a particular number of hidden states, n. Note that the number of hidden states n corresponds to the number of dimensions that are kept after the SVD of the Henkel matrix. This means that n most informative latent dimensions (i.e. those that carry the most variance) are used as hidden states. If, given a particular value of n, one model has better performance than another model, it means that it's best n dimensions capture more variance than the best n latent dimensions of the alternative model. This argument explains why one basis of a Hankel matrix can lead to a better

model than another basis. Intuitively, it is also natural to expect that long substrings can lead to a better set of hidden states because they can capture longer interactions between input symbols, which should naturally lead to more informative hidden states. If it was computationally feasible to evaluate dataset 5 with larger substring lengths, we could investigate the spectral norm of the empirical Hankel matrix with the increasing length of substrings (i.e. nR, nC). This would shed some light on the quality of the first hidden state in compared models.

Theoretically, a one-layer RNN model can capture infinite context (Siegelmann & Sontag 1991), but due to training difficulties (e.g. the vanishing gradient problem), the context capture capacity of RNNs is limited. Despite this difficulty, it was shown in the literature that RNNs can capture the previous context of up to 200 tokens (Khandelwal et al. 2018). However, other researchers (Pascanu et al. 2014) argue that stacking RNN layers (like in our two-layer model) does not increase the capacity of the model to capture longer contexts. This means that our two-layer model cannot deal with long contexts even when we add more layers. Since WFA did not perform well on dataset 5 having short context and a large number of hidden states, we conjecture that this dataset requires a long context and for this reason, two-layers in a neural model do not help. Our results are consistent with other results where long-term contexts are captured by the recurrent layer (Bengio & Frasconi 1995*a*). According to the distributed hypothesis (Bengio et al. 2009) stacking multiple layers allows for learning distributed features but not for capturing long-term contexts. Consequently, we assume that the long-term contexts are more important for dataset 5 to make efficient prediction than the pure increase in the number of hidden states across space.

Theoretical Considerations The relationship between the number of types of hidden states (discrete, or distributed), long-term dependency and the sequence prediction has been explored by Hinton et al. (2001), Bengio & Frasconi (1995*b*). For example, the hidden state of a single HMM (a specific version of WFA) can only convey $\log_2 K$ bits of information about the recent history. Instead, if a generative model had a distributed hidden state representation (Williams & Hinton 1991) consisting of M variables each with K alternative states, it could convey $M \log_2 K$ bits of information. This means that the information bottleneck scales linearly with the number of variables and only logarithmically with the number of alternative states of each variable (Hinton et al. 2001). However, the link between the hidden state modelling and the number of recurrent neural network layers had not been explored before. From this theoretical analysis, we can see that if we have access to a large dataset then

increasing the number of layers helps in modelling the hidden states more accurately (as seen in dataset 12), but it does not have to help to capture long-term contexts (dataset 5). In the latter case, one has to use models with high recurrence depth (Zilly et al. 2017, Pascanu, Montufar & Bengio 2013), but we leave their exploration for future work since in this chapter we wanted to focus on traditional LSTM layers.

To conclude this section, Recurrent Neural Networks (RNNs) are a powerful tool for sequence modelling. However, RNNs are non-linear models, which makes them difficult to analyse theoretically. In this chapter, we empirically analysed two RNN models (single-layer and two-layer RNNs) to understand the impact of the additional LSTM layers. We used Weighted Finite-state Automata (WFA) trained using the Hankel-based spectral learning algorithm. Based on fifteen benchmark datasets from the SPiCe 2016 competition, our empirical analyses indicate that multiple layers in RNNs help learning distributed hidden states through improved hidden space modelling but have a lesser impact on the ability to learn long-term dependencies.

Conclusion

In this chapter, we have used three different methods to model 15 datasets. We have shown that for the NLP data where prediction given context (language modelling) has long distanced context-dependency, RNN (with LSTM cell) based neural models perform better than N-gram and WFA. We have also analysed the impact of the multiple neural layers on the modelling. Although neural models perform better in modelling sequence prediction, training neural models is computationally expensive compared with N-gram models. In the next chapter, we will propose methods to improve the predictive accuracy of the neural models while improving the training time.

4

NEURAL LANGUAGE MODELS WITH APPROXIMATE NORMALISATION

In chapter 3, we have seen that compared to other models neural network models have enhanced capacity in incorporating longer distance context information in prediction. The N-gram models are fast and scalable, considered as the state-of-the-art models for language modelling, however when long term dependency is required, the performance gain of N-gram diminishes (Williams et al. 2015). We have seen in chapter 2 that neural language models do not scale well when the vocabulary is large and we have discussed different approaches to scale the neural network language models. Among them, one of the approaches is Noise Contrastive Estimation (NCE). NCE is a sampling-based method that allows for fast learning with large vocabularies. Although NCE has shown promising performance in neural machine translation, its full potential has not been demonstrated in the neural language modelling literature. A sufficient investigation of the hyperparameters in the NCE-based neural language models was also missing. In section 3.4.3.1, we have described the relevent hyperparameters for training a neural network model. In this chapter, we show that NCE can be a very successful approach in neural language modelling when the hyperparameters of a neural network are used appropriately. We introduce the 'search-then-converge' learning rate schedule for NCE and design a heuristic that specifies how to use this schedule. The impact of the other important hyperparameters, such as the dropout rate and the weight initialisation range, is also demonstrated. Using a popular benchmark, we show that appropriate tuning of NCE in neural language models outperforms the state-of-the-art single-model methods based on the standard LSTM recurrent neural networks.

4.1 Introduction

Neural network language models that apply various neural architectures (Bengio et al. 2003, Mikolov et al. 2010, Józefowicz et al. 2016) have recently demonstrated significant achievements. As we have discussed in section 2.4, the large vocabulary

is one of the main training bottlenecks for neural network language models. In real applications, the vocabulary size is large and the language models have to estimate a probability distribution over many words. The need for a normalised probability distribution becomes a computational bottleneck because the normalisation constant (i.e. the partition function) has to be computed for the output layer.

We have seen in section 2.4 that many solutions have been proposed to address the computational complexity of the partition function to resolve the softmax bottleneck. In this chapter, we investigate noise contrastive estimation (NCE) because of its statistical consistency, and the fact that its potential has not been sufficiently explored in the literature on neural language models (LMs). NCE has also achieved promising results in machine translation (Vaswani et al. 2013, Baltescu & Blunsom 2015), which indicates that its performance on language models could be better than what is known in current research.

NCE was first proposed in (Gutmann & Hyvärinen 2010) as an estimation principle for unnormalised statistical models (self-normalising approach without a guarantee to be normalised). It is also proposed as a self-normalising discriminative model (Goldberger & Melamud 2018). Unnormalised statistical models compute values which, in contrast to formal probabilities, do not add up to one. In order to normalise those values so that they become valid probabilities, they can be divided by the partition function (i.e. a normalising constant). However, the partition function is computationally expensive to compute when the number of outcomes is large. Therefore, instead of calculating the partition function, NCE converts the original estimation problem into a nonlinear logistic regression problem which discriminates between the noise samples generated from a known (noise) distribution and the original data samples. NCE is statistically consistent and more stable than other Monte Carlo methods such as importance sampling (Mnih & Teh 2012). In (Gutmann & Hyvärinen 2010), NCE achieved the best trade-off between computational and statistical efficiency when compared against importance sampling, contrastive divergence (Hinton 2002), and score matching (Hyvärinen 2005). This method has also been applied in language modelling and machine translation (Mnih & Teh 2012, Vaswani et al. 2013, Baltescu & Blunsom 2015, Zoph et al. 2016).

Many features of NCE are not understood, especially the hyperparameters in deep learning when NCE is used at the output layer. Also, comparisons against standard LSTM-based single-model softmax have never shown that NCE can compete with softmax on those tasks on which softmax is feasible. Our results are surprising in the face of the existing literature which generally indicates that NCE is an inferior method. For example, studying language modelling in (Józefowicz et al. 2016), the

authors argued that importance sampling (IS) may be better than NCE as IS optimises a multiclass classification task whereas NCE is solving a binary task. In (Józefowicz et al. 2016), the authors managed to improve the results on language modelling using IS, whereas similar improvements for NCE were not found. Overall, the current literature does not have substantial, empirical evidence that NCE is a powerful method for neural network language modelling.

Another example that demonstrates the weak performance of NCE on language modelling is the research by Chen et al. (2015). The authors explain that a limited number (50) of noise samples in NCE does not allow for frequent sampling of every word in a large vocabulary. Note that the number of noise samples has to be relatively small to make the method feasible. In their experiments, NCE performed better than softmax only on billionW, a dataset on which softmax is very slow due to a very large vocabulary. So, NCE was better only because softmax was not feasible on a large vocabulary. In this chapter, we show for the first time that NCE can outperform softmax in a situation when softmax is feasible and it is known to perform very well. To demonstrate that, we used the Penn Treebank (PTB) dataset[1] (Marcus et al. 1993*b*), which is a popular language modelling benchmark with a vocabulary size of 10K words. Softmax is known for competitive performance on this data, and it is feasible to apply it to this data using graphic processing units (GPUs).

There exist papers in which the researchers tried to create conditions which make softmax feasible to be executed on large datasets. For example, the experiments in (Baltescu & Blunsom 2015) are based on a few billions of training examples and vocabulary with over 100k tokens. To manage the softmax computation, the authors partitioned the vocabulary into K classes. Under those conditions, the authors showed that NCE performed almost as well as softmax. Softmax in their comparisons was approximate, however, due to partitioning. In this chapter, the goal is to compete with the original softmax without any approximations. Our aims are justified by the following reasoning. In theory, NCE, being a statistically consistent method, converges to the maximum likelihood estimation method when the number of noise samples is increased. However, the fact that NCE solves a different optimisation problem means that stochastic gradient descent applied to neural networks with NCE may find a different, better local optimum than when it is applied to networks with softmax. Therefore, when the objective function is highly non-convex, NCE can beat softmax even though it is only an approximation to softmax.

Tuning hyperparameters has been an important element of neural networks

[1] http://www.fit.vutbr.cz/ imikolov/rnnlm/simple-examples.tgz

research (Bengio 2012). The main contribution of this chapter is based on a carefully designed hyperparameter tuning strategy for NCE. The separate 'search-and-convergence' phases proposed by Darken & Moody (1991) for controlling the learning rate have never been applied to NCE-based neural networks. Also, the importance of the search phase and better convergence thanks to the randomness attributed to NCE was never observed in the literature. They appeared to be the key components in this research, and they allowed NCE to outperform softmax on a problem on which softmax is known to have competitive performance and to be computationally feasible.

Some researchers, e.g., Chen et al. (2015), concluded that NCE *'is not among the best techniques for minimising perplexity'*, and this was probably the reason why more sophisticated mechanisms, such as the 'search-and-convergence' phases for controlling the learning rate, were not used with NCE. This seems to be a common pattern in deep learning research. For example, in 2006, the community used unsupervised learning to initialise supervised learning for neural networks, whereas today, the appropriate resources and engineering practices allow feedforward networks to perform very well without unsupervised initialisation (Goodfellow et al. 2016, Ch. 6). Analogously, this chapter shows that appropriate techniques exist to turn NCE into a very successful method for neural network language modelling.

The chapter is organised as follows: section 4.2 introduces the NCE model with a deep neural architecture, and section 4.3 describes our approach to NCE-based neural language modelling (NCENLM). Sections 4.4 and 4.5 describe the experimental design and the results showing that the proposed method improves the state-of-the-art results on the Penn Treebank dataset using language modelling based on a standard LSTM (Hochreiter & Schmidhuber 1997, Gers 2001).

4.2 Background

We study language models where given a sequence of words $W = (w_0, w_1, w_2, \ldots, w_T)$ over the vocabulary V, we model sequence probability

$$p(W) = \prod_{i=0}^{T-1} p(w_{i+1}|w_0, \ldots, w_i) = \prod_{i=0}^{T-1} p(w_{i+1}|c_i). \tag{4.1}$$

Here, for a given word w_{i+1}, $c_i =< w_0, w_1, \ldots, w_i >$ represents its full, non-truncated context. In many applications, one is interested in $p(w_{i+1}|c_i)$. Recurrent neural networks try to model such probabilities that depend on a sequence of words c_i. The recurrent connections introduce a notion of 'memory', which can remember a

substantial part of word's context c_i. However, due to the gradient vanishing and exploding problems (Pascanu, Mikolov & Bengio 2013), it is challenging to optimise standard recurrent neural networks even though their expressive power is sufficient in many situations. For this reason, long short term memory (LSTM) was introduced to improve learning with a long context, c_i (Hochreiter & Schmidhuber 1997, Gers 2001). LSTM introduces the concept of memory cells that are used to create layers. Several layers can be stacked into larger blocks (similar to layers of neurons in the multilayer perceptron). The blocks of those layers are then unrolled for several time steps during learning.

When n is the last hidden layer, and i is the last unrolled time step, v_i^n (see Figure. 4.1) is the activation vector that results after the sequence of words c_i has been presented to the network. Then, the final output layer has one vector θ_j for every word w_j in the vocabulary, and the probability of the next word can be computed using the softmax function:

$$P_\theta^{SOFT}(w_{i+1}|c_i) = \frac{\exp(\theta_{i+1}^\top v_i^n)}{\sum_{j=1}^{|V|} \exp\left(\theta_j^\top v_i^n\right)} = \frac{\exp(\theta_{i+1}^\top v_i^n)}{Z}. \tag{4.2}$$

Here, $P_\theta^{SOFT}(w_{i+1}|c_i)$ is the probability of word w_{i+1} given context c_i, θ_{i+1} is the weight vector corresponding to the word w_{i+1} at the output layer, θ_j is the weight vector for the word w_j in the vocabulary, and $|V|$ is the vocabulary size. The normalising term Z is known as the **partition function**. Note that unnormalised products $\theta_{i+1}^\top v_i^n$ are not sufficient to evaluate the words.

The softmax-based training of recurrent neural networks that uses stochastic gradient descent (SGD) and backpropagation (BP) maximises the loglikelihood or equivalently minimises the cross-entropy of the training sequence containing N words. This objective can be formally expressed as

$$J_{SOFT}(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \ln P_\theta^{SOFT}(w_{i+1}|c_i). \tag{4.3}$$

The gradient used for updating the parameters θ is

$$\frac{\partial J_{SOFT}(\theta)}{\partial \theta} = -\frac{1}{N} \sum_{i=1}^{N} \left[\frac{\partial(\theta_{i+1}^\top v_i^n)}{\partial \theta} - \sum_{j=1}^{|V|} P_\theta^{SOFT}(w_j|c_i) \frac{\partial(\theta_j^\top v_i^n)}{\partial \theta}\right]. \tag{4.4}$$

Gradient computation is usually time-consuming because when the vocabulary is large, the partition function in P_θ^{SOFT} creates the performance bottleneck for the training and testing phases. It is advantageous to avoid this expensive normalisation

term. Noise contrastive estimation (NCE) bypasses this calculation by converting the original optimisation problem to a binary classification task.

In NCE, we see the corpus as a new dataset of n words of the following format:

$$((c_1, w_2), D_1)), \ldots, ((c_n, w_{n+1}), D_n)$$

where c_i represents the context, w_{i+1} represents the next word after c_i, and a random variable D is set to one when w_{i+1} is from the training corpus (true data distribution) and D is set to zero when w_{i+1} is from a known chosen noise distribution, P_n. For a given context c_i, the NCE-based neural language model (NCENLM) models data samples (from the corpus) as if they were generated from a mixture of two distributions (P_θ^{NCE} and P_n). The mixture is normalised; hence, the requirement for the normalisation term is satisfied implicitly as shown in Eq. (4.5).

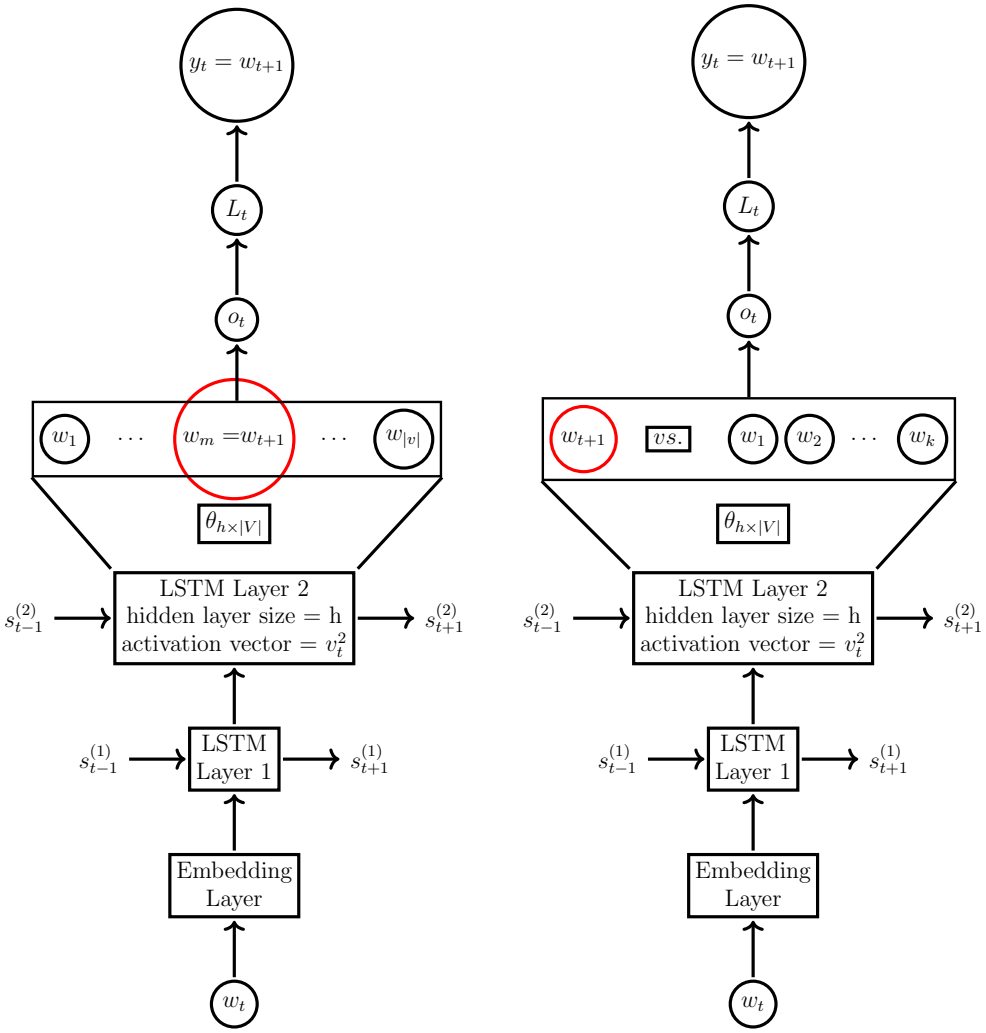
The posterior probability of a sample word w_{i+1} generated from the mixture of the P_θ^{NCE} and the noise distribution P_n are as follows:

$$\begin{aligned} P(D = 1|w_{i+1}, c_i) &= \frac{P_\theta^{NCE}(w_{i+1}|c_i)}{P_\theta^{NCE}(w_{i+1}|c_i) + kP_n(w_{i+1}|c_i)} \\ P(D = 0|\widetilde{w_{i+1}}, c_i) &= \frac{kP_n(\widetilde{w_{i+1}}|c_i)}{P_\theta^{NCE}(\widetilde{w_{i+1}}|c_i) + kP_n(\widetilde{w_{i+1}}|c_i)} \end{aligned} \tag{4.5}$$

where $\widetilde{w_{i+1}}$ is a word sampled from a known noise distribution P_n (e.g., a uniform distribution) and k is the ratio of the number of noise samples to the number of the data samples. A general assumption is that noise samples are k times more frequent than data samples. Based on this posterior distribution, NCE minimises the following objective function:

$$J_{NCE}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\Big[\ln P(D = 1|w_{i+1}, c_i) + \sum_{j=1}^{k} \ln P(D = 0|\widetilde{w_{i+1,j}}, c_i)\Big]. \tag{4.6}$$

which is the same objective function that is minimised by the traditional logistic regression. Here, for every word w_{i+1} that comes from a true data distribution, k noise samples $\widetilde{w_{i+1,j}}$ are generated from a known noise distribution, P_n. Mnih & Teh (2012) showed that for large k NCE-based parameter estimation is a close approximation of the maximum likelihood estimation.



(a) Illustration of the softmax output layer. (b) Illustration of the NCE output layer.

Figure 4.1: Illustration of the output layer: the computational graph to compute the training loss of a stacked recurrent network that maps an input sequence of values w to a corresponding sequence of output values o. A loss L measures how far each o is from the corresponding training target y. The loss L internally computes Eq. 4.3 or approximate it using Eq. 4.6. The mini-batched stochastic gradient descent optimisation will use Eq. 4.4 and Eq. 4.7 to find the parameters including the θ.

The gradient of the objective function is as follows:

$$\frac{\partial J_{NCE}(\theta)}{\partial \theta} = -\frac{1}{N}\sum_{i=1}^{N}\Bigg[\underbrace{\frac{kP_n(w_{i+1}|c_i)}{P_\theta^{NCE}(w_{i+1}|c_i)+kP_n(w_{i+1}|c_i)}\times\underbrace{\frac{\partial}{\partial\theta}\ln P_\theta^{NCE}(w_{i+1}|c_i)}_{\text{gradient}}}_{P_1} - \underbrace{\sum_{j=1}^{k}\frac{P_\theta^{NCE}(\widetilde{w_{i+1,j}}|c_i)}{P_\theta^{NCE}(\widetilde{w_{i+1,j}}|c_i)+kP_n(\widetilde{w_{i+1,j}}|c_i)}\times\underbrace{\frac{\partial}{\partial\theta}\ln P_\theta^{NCE}(\widetilde{w_{i+1,j}}|c_i)}_{\text{gradient}}}_{P_2}\Bigg] \tag{4.7}$$

where,

$$P_\theta^{NCE}(w_{i+1}|c_i) = \frac{\exp\left(\theta_{i+1}^\top v_i^n\right)}{Z}$$
$$P_\theta^{NCE}(\widetilde{w_{i+1,j}}|c_i) = \frac{\exp\left(\theta_j^\top v_i^n\right)}{Z}. \tag{4.8}$$

In the softmax gradient in Eq. 4.4, the normalisation term Z is required to compute P_θ^{SOFT}, which is a problem during training because Z has to be computed for every gradient calculation. Studying the NCE gradient in Eq. 4.7, one can see a subtraction of two products (see P1 and P2 in Eq. 4.7). The first terms of those products are normalised implicitly regardless of whether P_θ^{NCE} is normalised or not. Thus, the only terms in which normalisation can matter are the gradients in Eq. 4.7. It has been argued in the literature, however, that as far as the gradient is concerned, Z can be learnt as a parameter (Gutmann & Hyvärinen 2010, Labeau & Allauzen 2018) or it can be seen as a constant. In literature, the normalisation constant was set to one for all contexts in (Mnih & Teh 2012, Vaswani et al. 2013, Zoph et al. 2016). This is the precise reason why the partition function Z does not have to be computed in every iteration in NN training when the softmax is approximated by NCE. In our implementation, Z was learnt as a parameter following (Gutmann & Hyvärinen 2010, Labeau & Allauzen 2018). In the future, we plan to investigate the effect of different values of Z when NCE is used for a downstream application.

4.3 Our Approach

In this section, we present our training procedure with special hyperparameter tuning for NCE-based neural language models (NCENLM). Stochastic gradient descent (SGD) (Robbins & Monro 1951, Kiefer et al. 1952) along with the Backpropagation algorithm is the common training algorithm for neural networks (Goodfellow et al.

2016). Given a neural network with parameter θ, the gradient descent optimisation finds the value of θ that minimises the loss function $J(\theta)$ using the repeated update of the following expression until convergence:

$$\theta_j \leftarrow \theta_j - \eta \frac{\partial J(\theta)}{\partial \theta_j}$$

Here, η is the learning rate, which represents the size of the step taken at each iteration by stochastic gradient descent, j ranges from 0 to the number of parameters in the neural model.

Stochastic gradient descent requires only first order partial derivatives (gradient) of the loss function, where the derivative is efficiently calculated by Backpropagation, and it is simple and efficient compared to other first order and second order optimisation algorithms including Quasi-Newton methods and Newton's method. For SGD, the learning rate plays an important role in learning, whereas the second order methods and some other first order methods (e.g. Broyden-Fletcher-Goldfarb-Shanno (BFGS) (Broyden 1970, Fletcher 1970, Goldfarb 1970, Shanno 1970)) do not need the learning rate. Although these methods do not need the learning rate, they are computationally expensive and memory inefficient. The second-order optimisation algorithms require the partial derivative or a Hessian matrix. Although the algorithm makes efficient updates, we have to calculate the second order partial derivatives matrix for every parameter with respect to every other parameter, which makes it computationally costly and highly ineffective in terms of memory. Quasi-Newton methods like BFGS replaces that Hessian in Newton's method with an approximate Hessian but requires storing the approximate Hessian (allocating space to store it). For very large scale problems, with millions of parameters, these methods are difficult to implement. These criteria make SGD a favourable optimisation method for deep neural networks, which usually have a large number of parameters. We have seen in section 2.5.2 that for training a model, SGD is used because SGD yields significantly better generalisation than the batch gradient descent (Bousquet & Bottou 2008) and other adaptive optimisers (e.g. Adam (Kingma & Ba 2015), AdaGrad (Duchi et al. 2011)).

Why care about methods that need the learning rate?

The SGD optimisation requires a learning rate, which is a hyperparameter and has an important impact on the learning process (Bengio 2012). In this chapter, we will show that an appropriate learning rate schedule is important for faster convergence and better generalisation in NCE-based neural language models. Overall, the following hyperparameters were found to be important for the NCENLM: the

learning rate schedule, the weight (parameter) initialisation strategy, and sampling techniques.

4.3.1 Learning rate

The learning rate is one of the most prominent hyperparameters in deep network training (Bengio 2012). The 'search-then-converge' learning rate schedule for SGD usually has the form of $\eta(t) = \eta_0(1 + \frac{t}{\tau})^{-1}$ (Darken & Moody 1991), where t is the epoch number, η_0 is the initial learning rate, τ is a parameter, and $\eta(t)$ is the learning rate for epoch t. This allows the learning rate to stay high during the 'search period' $t \leqslant \tau$. It is expected that during this period the parameters will hover around a good minimum. Then, for $t > \tau$, the learning rate decreases as $\frac{1}{t}$, and the parameters converge to a local minima (Robbins & Monro 1951).

In our implementation, we used the 'search-then-converge' learning rate schedule of the form:

$$\eta(t) = \eta_0 \times \left(\frac{1}{\psi}\right)^{\max(t+1-\tau, 0.0)}, \tag{4.9}$$

which previously appeared in other studies that involve neural networks (Zaremba et al. 2014). The hyperparameter ψ is kept constant in our experiments and its value was set according to (Zaremba et al. 2014). During the search period ($t \leqslant \tau$ epochs), the learning rate is constant and equal to η_0, and during the convergence period the learning rate is decreased by a factor of $\frac{1}{\psi}$. The initial learning rate η_0 is one in (Zaremba et al. 2014), and we use the same value in our experiments.

Our investigation has shown that learning with NCE is more sensitive to the length of the search period ($t \leqslant \tau$) when comparing with softmax. Choosing appropriate τ is, therefore, crucial for convergence of NCE-based learning with SGD. Our research suggests that, when NCE is used, τ should be between 1 and two-thirds of the total number of training epochs. For instance, if we need 40 training epochs then τ could be between 1 and 26. This was one of the most important insights that allowed us to improve the performance of NCE.

Our intuition behind the investigation is based on the two facts. First, learning the normalising constant in NCE is a part of the overall training of the neural models. This normalisation constant learning is more dependent on the search phase of the optimisation and longer search phase would facilitate this learning. Second, the convergence rate of the stochastic gradient descent on smooth convex function is given by $O(L/T + \sigma/\sqrt{T})$ (Sutskever et al. 2013), where σ is the variance in the gradient estimates, T is the number of iterations and L is the Lipschitz coefficient of

∇J. For convex objectives, the search phase of the SGD optimisation is dominated by the term $\frac{L}{T}$ (Sutskever et al. 2013). In our investigation, the search phase epochs showed training error higher than the corresponding softmax training (see Fig. 4.8) and the NCE gradient range is larger than the softmax gradient range (see Fig. 4.9). To understand the range of the large gradients, we use the concept of the gradient with respect to the Lipschitz coefficient for a differentiable function.

The Lipschitz coefficient can be understood using a simple observation. The slope m of the linear function $f(w) = mw + c$ determines how much the function values change as the input values w change. The larger $|m|$ is, the steeper the line is, and the more the function values change for a given change in input w. The concept of Lipschitz continuity is designed to measure the change of function values versus change in the independent variable for a general function $f : I \rightarrow Q$, where I is a set of rational numbers. If w_1 and w_2 are two numbers in I, then $|w_2 - w_1|$ is the change in the input and $|f(w_2) - f(w_1)|$ is the corresponding change in the output. We say that f is Lipschitz continuous with Lipschitz constant L on I if there is a (non-negative) constant L such that

$$|f(w_2) - f(w_1)| \leqslant L|w_2 - w_1|, \forall w_1, w_2 \in I. \tag{4.10}$$

A differentiable function $f(.)$ is L-smooth or L-gradient Lipschitz (Jin et al. 2017) if

$$|\nabla f(w_2) - \nabla f(w_1)| \leqslant L|w_2 - w_1|, \forall w_1, w_2 \in I. \tag{4.11}$$

If L is small, then $\nabla f(w)$ may change only a little with a small change of w, while if L is large, then $\nabla f(w)$ may change a lot on only a small change of w. To summarise, L may vary from small to large depending on the function f.

We have used the above theory to understand the observation that the NCE gradient range is larger than the softmax gradient range and we assume that the NCE objective function has a larger value for L compared to the softmax objective function. Based on this understanding, we assume that to converge to the softmax range (which presumably has smaller L), NCE-based training needs more iterations (i.e. T in $\frac{L}{T}$) in search phase where the learning rate is constant compared to softmax.

In the 'convergence phase', NCE takes fewer epochs to get similar results as softmax. However, in this phase $\sigma/\sqrt{T}$ is the dominant term (i.e., when the optimisation problem resembles estimation problem) (Sutskever et al. 2013). In our inves-

tigation, in the convergence phase, the range of the gradient estimate is smaller (i.e. smaller variance) than the softmax. Perhaps this is the reason that NCE takes fewer iterations to converge. Moreover, the NCE formulation reduces the softmax (multi-class classification) objective function into a binary classification objective function. Thus, perhaps induced convexity helps in faster convergence.

It is also interesting to observe that the Eq. 4.11 is referring to the change of the change of the first derivative, which is essentially the second derivative. The second derivative tells us how the first derivative will change as we vary the input. This is important because it tells us whether a gradient step will cause as much of an improvement as we would expect based on the gradient alone. We can think of the second derivative as characterisation of the curvature. If a function has a second derivative of zero, then there is no curvature. It is a perfectly flat line, and its value can be predicted using only the gradient. If the gradient is 1, then we can make a step of size η along the negative gradient, and the cost function will decrease by η. If the second derivative is negative, the function curves downward, so the cost function will actually decrease by more than η. Finally, if the second derivative is positive, the function curves upward, so the cost function can decrease by less than η. If L is small, the function has flat curvature and L defines more upward curvature. In the future, we are planning to investigate and understand the behaviour of the NCE in terms of the second derivative.

4.3.2 Weight Initialisation

In neural networks, the initial weights are usually drawn from a uniform distribution (Glorot & Bengio 2010), unit Gaussian (Sutskever et al. 2013) or a general Gaussian distribution (He et al. 2015). For our NCE training, we used a uniform distribution for the weight initialisation. All three distributions described above were compared, but the uniform distribution led to slightly better results. *However, regardless of which distribution is used, we found that NCE works better when the initial weights are within a smaller range, i.e. when the variance of the initial weights is smaller than what is suggested in (Glorot & Bengio 2010). This was another insight that led to significant improvements in NCE performance.*

The commonly used heuristic (Glorot & Bengio 2010) for initialising the weights θ_{ij} at each layer is:

$$\text{Weight} \sim \mathcal{U}\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$$

where $\mathcal{U}[-a, a]$ is the uniform distribution in the interval $(-a, a)$ and n is the size of the previous layer.

Glorot & Bengio (2010) observed that the variance of the gradient on the weights is the same for all layers in feedforward neural networks, but the variance of the back-propagated gradient might still vanish or explode in the deeper networks. A similar phenomenon was observed by Bengio et al. (1994) when studying recurrent neural networks.

Glorot & Bengio (2010) argued that the standard initialisation gives rise to variance and causes the variance of the back-propagated gradient to be dependent on the layer (and decreasing). They assumed that the normalisation factor may, therefore, be important when initialising deep networks because of the multiplicative effect through layers, and they suggested the following initialisation procedure to approximately satisfy their objectives of maintaining activation variances and back-propagated gradients variance as one moves up or down the network.

$$\text{Weight} \sim \mathcal{U}\left(-\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}\right)$$

In our investigation, we found that NCE has a larger gradient range than the softmax (see Fig. 4.9). This indicates that we need a mechanism to reduce the gradient range. The larger gradient range is probably due to the fact that in NCE the entire weight matrix of the output layer does not change during the backward pass of the back-propagation algorithm. Only the part of the weight matrix corresponding to the sampled noise data gets updated in each iteration. Thus the weights of the output layer are less influenced by the co-adaptation compared to softmax. The following smaller initialisation (with some manual choices for ranges in Table 4.4) satisfies our objectives of maintaining the variance of the back-propagated gradient for the NCELM.

$$\text{Weight} \sim \mathcal{U}\left(-\frac{\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}}{4}, \frac{\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}}{4}\right)$$

4.3.3 Sampling Techniques

For NCE, the noise distribution (i.e., P_n in Eq. 4.5) plays an important role. The proposed NCE method (Gutmann & Hyvärinen 2010) is dependant on the noise sample generated from the distribution to contrast the original data generating distribution. For efficient training, the noise distribution should be close to the data distribution (Gutmann & Hyvärinen 2010). In practice, we do not know the data generating distribution, therefore it is very difficult to design a noise distribution that is a good match with the data distribution. For language modelling, one can assume that sampling noise from an N-gram model, particularly an N-gram model with $N \geqslant 5$, would give a good representation of the noise distribution. However, according to Bengio & Senécal (2008), neural language models and N-gram models learn very different distributions.

For efficient noise distribution, we have come up with a noise distribution that efficiently describes characteristic of the word distribution (if not language model distribution). As word distributions follows Zipf's law (Piantadosi 2014), we have used noise samples from a power law distribution (i.e., Zipfian (log-uniform) distribution) (Tullo & Hurford 2003, Chierichetti et al. 2017). To be able to use samples from that distribution, the words in the vocabulary have to be sorted in the descending order of the frequency. The unigram distribution can be another good representative for the noise distribution. Sampling the noise samples from the unigram distribution when the words in the vocabulary are sorted in an ascending or a descending order gave competitive empirical results compared to sampling from the power-law distribution. We have also used a uniform distribution for comparisons in section 4.4.

4.4 Experimental Methodology and Implementation

We aim at showing that NCE can outperform alternative methods for language modelling. In particular, we investigate its performance in the context of softmax because NCE approximates softmax being consistent with softmax in the limit. Our implementation of NCE follows our approach presented in section 4.3.

In our experiments, we focus on the popular perplexity measure (PPL) using the Penn Treebank (PTB) dataset. It is feasible to run 'exact' softmax on this dataset, and the large literature that uses it allows for comparisons with other approaches (see Tab. 4.3).

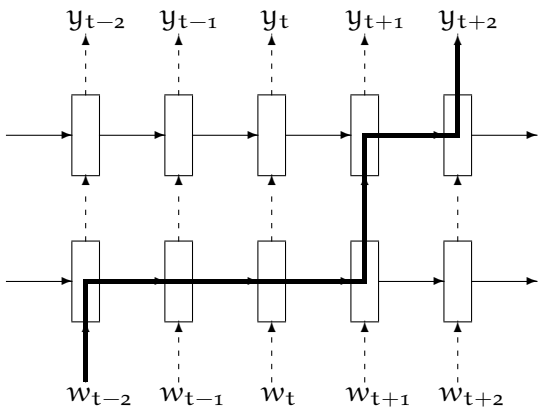


Figure 4.2: The unfolded stacked LSTM network: the thick line shows a typical path of information flow in the LSTM. The information is affected by dropout $L + 1$ times, where L is the depth of the network. The dashed arrows indicate connections where dropout is applied, and the solid lines indicate connections where dropout is not applied. The figure is adapted from (Zaremba et al. 2014, Figure 3).

The PTB dataset consists of 929k training words, 73k validation words, and 82k test words. The vocabulary size is 10k. Softmax usually becomes inefficient when the vocabulary size exceeds 10k words.

All models were implemented in Tensorflow[2] and executed on NVIDIA K80 GPUs. The standard components of our models follow (Zaremba et al. 2014), where excellent results on this dataset were reported. The words were represented with dense vectors trained using the skip-gram model with negative sampling (Mikolov, Sutskever, Chen, Corrado & Dean 2013) on the Wikipedia corpus downloaded on December 2016. This word representation was used across all our experiments.

To perform more experiments, we designed models of three sizes: small (S), medium (M) and large (L). The small model is non-regularised whereas the medium and large models are dropout regularised with 50% and 60% dropout rate on the non-recurrent connections (See Fig. 4.2) in the medium and larger models correspondingly. This led to the best empirical results after investigating different dropout rates in the suggested ranges (Srivastava et al. 2014). All the models have two LSTM layers with the hidden layer size of 200 (S), 650 (M), and 1500 (L). The LSTM was

[2] https://www.tensorflow.org/

unrolled for 20-time steps for the small model and 35-time steps for the medium and large models. We used mini-batch SGD for training where the mini batch size was 20.

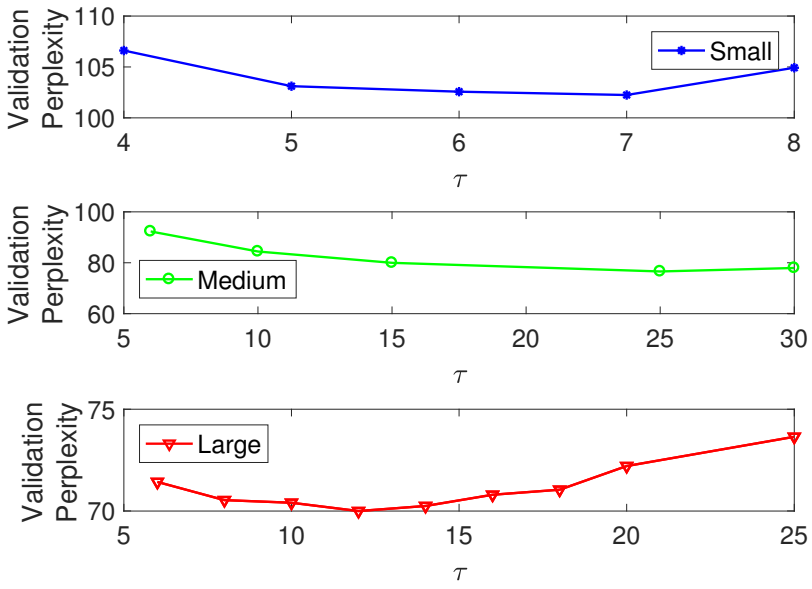


Figure 4.3: Selection of the learning rate parameter τ

For sampling the initial weights, a smaller range than the one suggested in (Glorot & Bengio 2010) turned out to be beneficial for NCE. We tested several initialisation heuristics which are described in the corresponding column in Tab. 4.4. Row number 1 shows the formula suggested in (Glorot & Bengio 2010). Note that U denotes a uniform distribution with its minimum and maximum values, and n_i denotes the number of hidden nodes in layer i. Row number 2 shows our updated formula that reduces the first range by a factor of 4, and row number 3 shows the range values that led to the best results in our experiments.

The learning rate was scheduled using Eq. 5.3. The search time limit τ was chosen empirically using Fig. 5.4. As a result, τ was set to 7, 25 and 12 for the small, medium and larger models correspondingly. During the convergence period, the parameter ψ was set to 2, 1.2 and 1.15 for the small, medium and larger models as suggested by (Zaremba et al. 2014). We trained the models for 20, 39 and 55 epochs respectively.

The norm of the gradients (which was normalised by the mini batch size) was clipped at 5 and 10 for the medium and large models correspondingly. To com-

pute validation and testing perplexity, we used softmax to guarantee the accuracy of our comparisons.

Table 4.1: Impact of noise sample distribution on medium models

Noise Distribution Type	Test PPL
Uniform	96.046
Unigram	76.621
Power law	75.286

Table 4.2: Impact of noise sample size on medium models

Sample Size	Time	Test PPL
50	1 hr 50 min	79.383
100	1 hr 56 min	77.539
150	1 hr 56 min	77.207
300	1 hr 57 min	76.090
600	1 hr 57 min	75.286
800	2 hr 2 min	75.822
1200	2 hr 8 min	75.397

In NCE, we used 600 noise samples. The noise samples were generated from the power law distribution. Table 4.1 reports the Test Perplexity of three noise distributions for medium model and power-law distribution resulted efficiently compared to unigram and uniform distribution. We evaluated different noise sample sizes (50, 100, 150, 300, 600, 800 and 1200), and 600 had the best trade-off between quality and processing time (see Tab. 4.2). We observed that when a GPU implementation is used, it is possible to increase the sample size within a reasonable range without dramatically increasing the computational complexity. Our softmax-based language model was implemented and parametrised according to (Zaremba et al. 2014), which achieved state-of-the-art results using the standard LSTM network.

The two models that we implemented, i.e. softmax- and NCE-based language models, used a standard LSTM network (Hochreiter & Schmidhuber 1997, Gers 2001). Many extensions to the LSTM architecture exist, e.g., Recurrent Highway Networks (RHN) (Zilly et al. 2017), that may improve LSTM's capabilities in capturing long term dependencies. In this chapter, we aimed at comparing NCE and softmax on standard LSTM networks, but our results could generalise to other, potentially more advanced types of LSTM cells. We have shown in chapter 5 such generalisation with RHN, but other architectures should show similar outcomes in

terms of performance. It should be noted, however, that our NCE implementation with standard LSTM outperforms some language models which use more advanced versions of LSTM as shown in Tab. 4.3.

4.5 Results and discussion

Table 4.3: Comparison with the state-of-the-art results of different models on the PTB dataset

Model Description	Valid. PPL	Test PPL
Classic RNN and LSTM		
Deep RNN (Pascanu et al. 2014)	-	107.5
Sum-Prod Net (Cheng et al. 2014)	-	100.0
RNN-LDA + KN-5 + cache (Mikolov & Zweig 2012)	-	92.0
Conv.+Highway+ regularized LSTM (Kim et al. 2016)	-	78.9
Non regularised LSTM with Softmax (Zaremba et al. 2014)	120.7	114.5
Medium regularised LSTM with Softmax (Zaremba et al. 2014)	86.2	82.7
Large regularised LSTM with Softmax (Zaremba et al. 2014)	82.2	78.4
Non regularised LSTM with NCE (our method)	106.196	102.245
Medium regularised LSTM with NCE (our method)	78.762	75.286
Large regularised LSTM with NCE (our method)	72.726	**69.995**
Extended or Improved LSTM		
Variational LSTM (Gal & Ghahramani 2016*b*)	77.3	75.0
Variational LSTM + Weight Tying(Press & Wolf 2016)	75.8	73.2
Pointer Sentinel LSTM (Merity et al. 2016)	72.4	70.9
Variational LSTM + Weight Tying + augmented loss (Inan et al. 2017)	71.1	68.5
Variational RHN (Zilly et al. 2017)	71.2	68.5
Variational RHN + Weight Tying (Zilly et al. 2017)	67.9	65.4
Neural Architecture Search with base 8 and shared embeddings	-	62.4
Model Averaging/ Ensembles		
38 large regularized LSTMs (Zaremba et al. 2014)	71.9	68.7
Model averaging with dynamic RNNs and n-gram models (Mikolov & Zweig 2012)	-	72.9

The results in Tab. 4.3 compare our best NCE-based result with other state-of-the-art methods. Our result is the best in the class of single-model methods that use a standard LSTM; the large model achieved the perplexity of **69.995** after 55

Table 4.4: Weight initialisation ranges for the uniform distribution (U) and the corresponding test perplexity (PPL)

No.	Initialisation Heuristic	Small Model	Medium Model	Large Model
1	$\mathrm{U}\left(-\frac{\sqrt{6}}{\sqrt{n_i+n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i+n_{i+1}}}\right)$	U(-0.1225, 0.1225) PPL = 104.449	U(-0.0679, 0.0679) PPL = 75.960	U(-0.04472, 0.04472) PPL = 71.184
2	$\mathrm{U}\left(-\frac{\frac{\sqrt{6}}{\sqrt{n_i+n_{i+1}}}}{4}, \frac{\frac{\sqrt{6}}{\sqrt{n_i+n_{i+1}}}}{4}\right)$	U(-0.031, 0.031) PPL = 102.245	U(-0.0169, 0.0169) PPL = 75.959	U(-0.011180, 0.011180) PPL = 70.444
3	Empirically Tuned Ranges	U(-0.0153, 0.0153) PPL = 102.237	$\mathrm{U}(-0.00849, 0.00849)$ PPL = 75.286	U(-0.00625, 0.00625) PPL = 69.995

Table 4.5: Comparison of softmax and NCE

Large Model	Time	Valid. PPL	Test PPL
Softmax (55 epochs)	9 h 11 min	82.588	78.196
Softmax (20 epochs)	3 h 40 min	79.798	76.935
NCE (55 epochs)	7 h 34 min	72.726	69.995
NCE (20 epochs)	2 h 36 min	76.268	74.129

training epochs. This result outperforms all known single-model algorithms that use the same kinds of LSTM cells. The total time for training, validating and testing our large NCE-based model was 7 hours 34 minutes (see Tab. 4.5). The 55 epochs of softmax took 9 hours 11 minutes, and the testing perplexity was 78.826. Note that, all the experiments were run on the same machine. Early stopping, which is a common regularisation method (Goodfellow et al. 2016), allowed softmax to achieve a testing perplexity of 76.935. So, softmax was overfitted after 55 epochs. The same overfitting was not observed in NCE as can be seen in Tab. 4.5 and Fig. 4.4.

Below, we present additional results that explain the good performance achieved by NCE and provide further insights into its properties.

Figure 4.4 presents the validation perplexity (Y axis) of a large model for different dropout rates as a function of an epoch number (X axis) at the convergence stage of learning. One can see that softmax with a dropout rate of 60% overfitted since the 21st epoch. Increasing the dropout rate to 70% allowed softmax to avoid overfitting, but the asymptotic performance was not as good as in NCE. The asymptotic convergence of NCE was superior across a range of dropout rates. In NCE, the gradients (Eq. 4.7) are different and more noisy than in softmax (Eq. 4.4). We know that SGD leads to better generalisation than batch gradient descent because of the in-

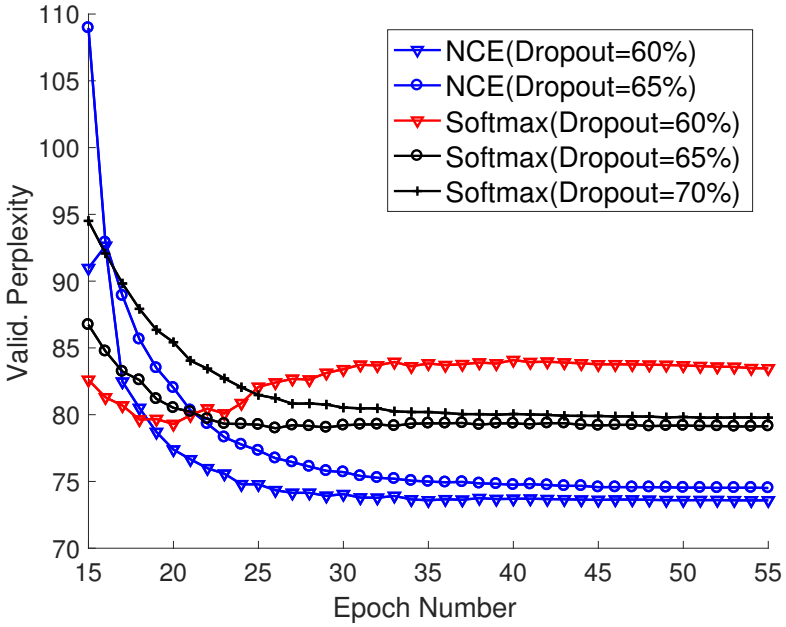


Figure 4.4: Convergence phase in the large model

duced noise by updating the parameters from a single example (Bousquet & Bottou 2008). A similar property of NCE could justify its robust generalisation in Fig. 4.4.

Figures 4.5 and 4.6 show the validation perplexity (Y axis) for selected values of τ as a function of an epoch number (X axis) at the convergence stage of learning. These figures demonstrate the critical impact of the learning rate schedule on NCE. Figure 4.5 shows that NCE requires a long search period (large $\tau = 25$) to achieve competitive asymptotic convergence on the medium model. Figure 4.6 for the large model has additional evidence that a long search period is required because larger $\tau = 12$, in addition to having better asymptotic convergence, has poor (i.e. high) perplexity in the initial phase. This poor perplexity indicates that the algorithm explores widely at this stage, but by doing that it can avoid converging to the nearest local optima. High initial perplexity is even more pronounced in Fig. 4.7 which is for all epochs of the medium model (note that perplexity is on a log scale here). Although difficult to see in the figure, the asymptotic validation perplexity is the best for NCE at $\tau = 25$. There was also a difference in test performance between NCE and softmax: NCE with $\tau = 25$ scored 75.959, NCE with $\tau = 6$ scored 83.858, softmax with $\tau = 25$ scored 79.906, and softmax with $\tau = 6$ achieved 78.567. NCE with high τ was the best; and increasing τ from 6 to 25 reduced perplexity from 83.858 to 75.959, which confirms the significance of our arguments in section 4.3. Thanks to the noise samples, NCE can explore better than softmax when the exploration phase is long

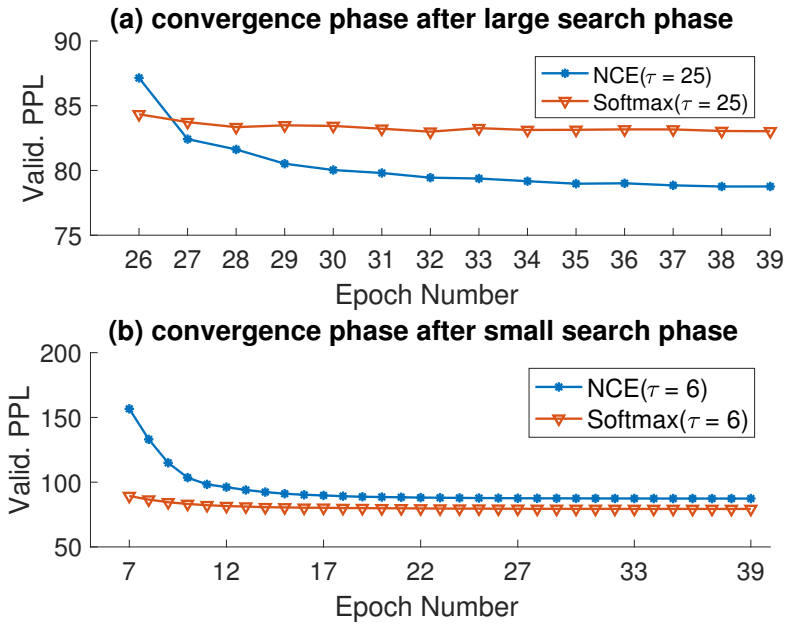


Figure 4.5: Convergence phase in the medium model

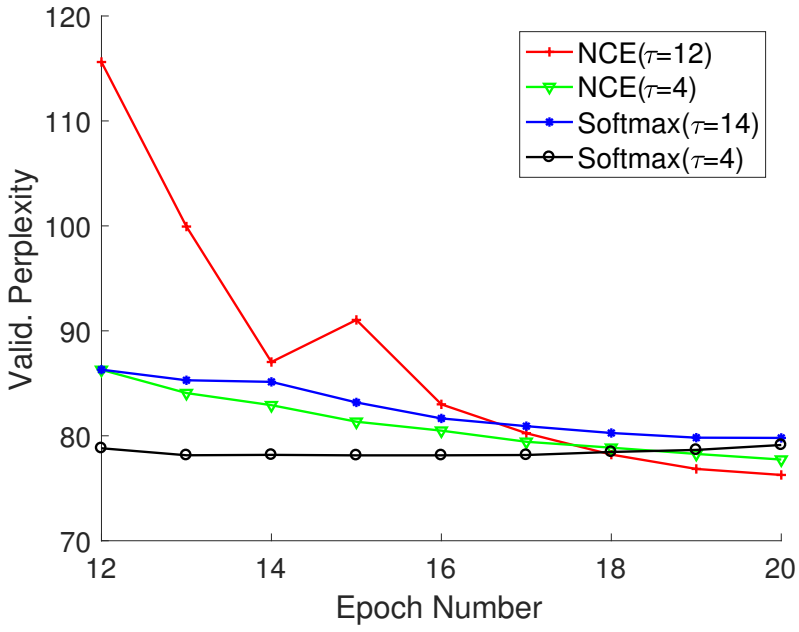


Figure 4.6: Convergence phase in the large model

enough; which is confirmed through high perplexity in the initial stage of learning. This means that NCE can find a better solution potentially for the same reasons which make stochastic gradient descent better than batch gradient descent (Bottou 2010).

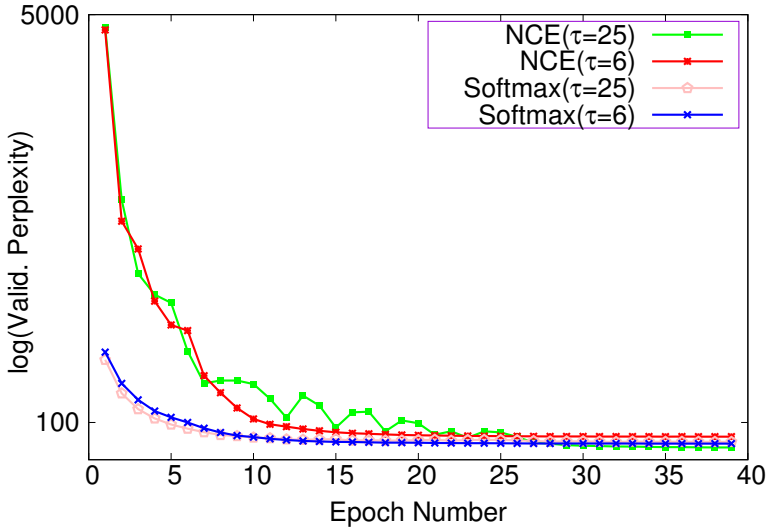


Figure 4.7: Validation perplexity of the medium model during all epochs of learning

Our results in Fig. 4.6 and 4.7 indicated that good NCE results could be attributed to its high error, i.e., high perplexity, in the early stages of learning, which may allow for broad exploration. We tried to enforce similar behaviour in softmax using a large learning rate in the search period. Figure 4.8(a) presents the validation perplexity in the log scale for a large model with softmax (Y-axis) as a function of a training epoch (X-axis) and the learning rate (LR) which was increased to 1, 2, and 3 during search time. This arrangement increased the validation perplexity for the first few epochs, but the asymptotic convergence of softmax was not improved. When, in Fig. 4.8(b), we compare the increased initial softmax perplexity with NCE perplexity during the progression of the first epoch, we can see that NCE has much larger perplexity at this stage, even though its learning rate is not larger than one. It might be a distinct characteristic of the NCE that helps to converge to a **better local optimum** due to the initial high training error.

The numerical entries in Tab. 4.4, i.e., in all cells in the bottom right part of the table, contain both the intervals $\mathcal{U}$ used to sample initial weights and the resulting perplexity (PPL) on a corresponding model. The results on the large model show that weight initialisation with lower variance led to better results, where the best perplexity of 69.995 was the best result that NCE achieved in our experiments.

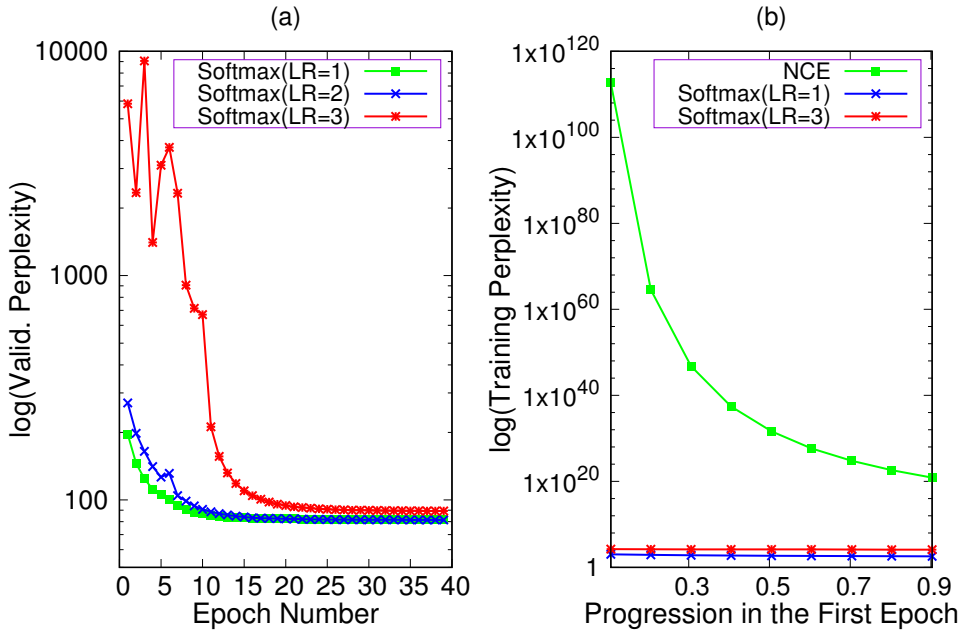


Figure 4.8: High learning rate (LR) to increase the initial softmax perplexity (a). NCE and softmax initial perplexities in the first epoch; note that only training perplexity is available within one epoch (b).

4.5.1 Gradient Analysis

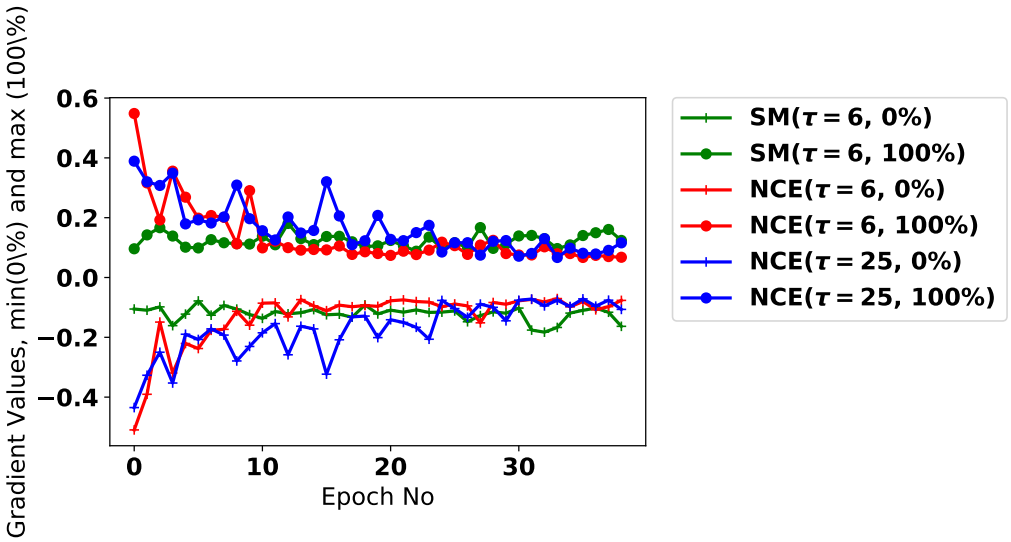


Figure 4.9: The gradient range

Figure 4.9 reports the gradient range (Y-axis) of medium models for different τ as a function of an epoch number (X-axis) for both the search phase and the convergence phase. For example, when $\tau = 6$ for softmax (SM) or noise contrastive estimation (NCE) then the search phase ends at epoch 6 and the convergence phase starts from epoch 7 and ends at epochs 39. Here 0% and 100% respectively define the lower and upper limit of the gradient values for an epoch. Concretely, there are no (0%) gradient values observed below the 0% line plot and all (100%) gradient values observed below the 100% line plot. In Figure 4.9, we can see that in the initial epochs (i.e. the search phase) of NCE, the gradients of the loss function with respect to the output layer (linear layer with parameter matrix θ) parameters have larger values than softmax, but at the later epochs (i.e. the convergence phase) have smaller values than softmax. We have used this observation during our investigation and designed the method described in section 4.3.

4.5.2 Consistency Analysis

The exact results in this chapter may vary depending on the random weight initialisation (in NCE and softmax) and sampling noise (in NCE). To check the consistency, we repeated our training multiple times (independently) and the results are consistent. For example, when NCE was executed four times using the large model, we obtained the following testing perplexities [70.082, 70.064, 69.975, 69.995]. The standard deviation is 0.045. Similarly, with softmax we obtained [78.394, 78.288, 78.196, 78.226]. The standard deviation is 0.075. Overall, these values show that our results are consistent. Also, the results reported in the Figures were compared across the four runs, and the trends that we observed were consistent. Most papers that we cite in this chapter present one value for every experiment and repeated runs are not mentioned, except (Gal 2016) where the experiments were repeated for three times.

4.6 Conclusion

Language modelling techniques can use Noise Contrastive Estimation (NCE) to deal with the partition function problem during learning. Although it was known that NCE can outperform softmax (which computes the exact partition function) on large problems which are too big for softmax, its performance has never been shown to outperform softmax or other methods on tasks on which softmax is feasible and works well. In this chapter, we showed that NCE can beat all the previous best

results in the class of single-model methods based on a standard LSTM achieving perplexity of 69.995. Our result establishes a new standard on the Penn Treebank dataset reducing the perplexity of the best existing method in this class by 8.405.

5

GENERALISATION IN DEEP NEURAL LANGUAGE MODEL THROUGH SELF-NORMALISATION

We have seen in chapters 2 and 4 that to deal with large vocabularies (e.g. over 10K words), researchers have developed various approximations that make learning computationally feasible. These approximations (e.g. NCE by Gutmann & Hyvärinen) usually modify the final layer of the neural network. Note that the changes to the final layer have a profound impact on the lower layers during learning because the gradients from the final layer are backpropagated to all previous layers. As a result, the model that deals with large vocabularies using approximations in the final layer (i.e. output layer) can improve the quality of the entire, deep model. Thus two benefits can be achieved through the approximated output layer: training time can be reduced and better performance on test data (i.e. improved generalisation) can be achieved by the approximation method compared to the exact method. Such results were shown in chapter 4 where noise contrastive estimation (NCE) (Gutmann & Hyvärinen 2010) is used as an approximation method, and the model was efficiently trained with less time and outperformed 'less approximate' approaches.

In chapter 4, we have seen an integration of noise contrastive estimation (NCE) (Gutmann & Hyvärinen 2010) with stacked RNN neural networks, and NCE based model outperformed (faster training and improved generalisation) 'less approximate' softmax models. Improved generalisation means that NCE models were more accurate in terms of the perplexity metric compared to the Softmax model on unseen (i.e. test) data.

To provide tangible evidence that the potential of NCE is more general and NCE can improve performance **regardless of the lower layers in the neural networks (representation layer)**, we provide a detailed analysis that explains the positive impact of NCE on language models based on neural networks. Deep neural networks are over-parametrised (i.e. high capacity) models (Arora et al. 2018*a*, Pérez et al. 2019). We have seen in section 2.5.2 that it is sufficient to regularise only the output layer (i.e. the all connected linear + softmax activation) weights in order to constrain capacity (Bengio 2012). Moreover, Bengio et al. (2012) argued that large capacity in

Why regularise the output layer?

the output layer does not help when training a very large neural network. So, limiting the output layer can help training the whole network efficiently. As approximating the output layer using NCE reduces generalisation error, the question is, can we see NCE as a regulariser? In the most basic context of deep supervised learning, generalisation error is the gap between the error on the training and test sets drawn from the same distribution. According to Goodfellow et al. (2016, ch. 7) regularisation can be defined as any modification we make to a learning algorithm that is intended to reduce its generalisation error but not its training error. According to Neyshabur (2017), generalisation can be explained by the test error being close to the training error, even when minimising the training error. According to the state-of-the-art definitions, to qualify NCE as a regulariser, we have to look into both training error and test error; and investigate the gap between these two errors. Experimental results in the last chapter show that NCE demonstrated lower test error than Softmax. In this chapter, we perform experiments to verify that NCE is a regulariser based on these definitions. We find that NCE is indeed a **regulariser**. Seeing NCE as a regulariser as per these definitions raises some more research questions. **First**, why is it that NCE induces regularisation even though the number of parameters is unchanged compared to Softmax. Usually, a regulariser reduces the number of parameters (i.e. capacity) of the model. **Second**, can we define regularisation and resulting generalisation more robustly because seeing the regularisation in terms of the error gap does not tell us much about the trained model. Understanding regularisation in terms of a trained model would help us designing robust, domain-specific regularisation techniques, and importantly, better understand overall neural learning. Therefore, we will attempt to answer these research questions in this chapter. To answer these questions, we have to understand generalisation from the deep neural networks perspective.

Generally, deep networks generalise well in supervised learning tasks even with over-parametrised (i.e. high capacity) models (Arora et al. 2018*a*, Pérez et al. 2019). In spite of recent research efforts in deep learning, the problem of understanding generalisation remains far from solved. With an over-parametrised model (e.g. deep neural network), to generalise well on unseen data, we need a mechanism to reduce overfitting, and regularisation techniques are used to get the model from the overfitting regime to fitted regime (see section 2.5.2). However, it is not clear how or why a particular regularisation technique does that in a large neural model. Current research challenges include understanding the data-dependency of the gap (i.e. the gap between train and test error), the role of increasing network depth (in space; e.g. stacked network and in time; e.g. large recurrent depth), and the role of implicit (e.g. implicit self-regularisation by Martin & Mahoney (2018)) and explicit

regularisation. Overall, as we discussed in section 2.5.2, there is no established and theoretical formulation or measurement of generalisation for deep neural networks. Therefore, in this chapter, answering the research questions would help fill the gap in knowledge.

The results in chapter 4 are limited to classic, stacked LSTM networks with standard dropout. Training stacked LSTM networks requires the learning algorithm to calculate the error contribution across both space and time, which is difficult in practice. For this reason, many models are usually restricted to only two or three layers (Graves 2013, Zaremba et al. 2014). Stacking layers in LSTM shows performance improvement on real datasets (Graves 2016, Godin et al. 2017, Merity et al. 2017) at the expense of increased computational complexity. Moreover, in chapter 3 we have seen that for NLP tasks where long term dependency has more impact on the performance than modelling of the hidden space, increasing the recurrence depth is beneficial for long term dependency modelling.

Stacked LSTM networks have limited modelling power due to the vanishing gradient problem (Srivastava et al. 2015, Zilly et al. 2017). Recurrent Highway Networks (RHNs) (Zilly et al. 2017) and other models that use large recurrence depth (Pascanu et al. 2014) were proposed to address the modelling challenge imposed by the stacked LSTM network. In this chapter, we show that NCE leads to significant gains in performance in one of such RNN models that have large recurrence depth (Pascanu et al. 2014). We focus on RHNs (Zilly et al. 2017) because they are a specific generalisation of the classic LSTM and have long credit assignment paths which are not just in time but also in space (per time step). The credit assignment paths are the chain of transformations from input to output of a given network. Large credit assignment paths enable improved modelling power for a neural model, however, this requires appropriate handling of the vanishing gradient problem. RHN uses the key mechanisms of the LSTM to deal with the vanishing gradient problem (Zilly et al. 2017, see section 4). There is no similar mechanism available for stacked networks.

The main goal of this chapter is not to argue whether RHNs or stacked LSTMs are better. Showing that NCE works for a particular generalised LSTM network (i.e. RHN) is appealing as other models that were derived from LSTM could have a similar benefit. Since NCE explicitly modifies learning in the output layer only, our findings could apply to other recurrent neural models, for example, (Melis et al. 2017).

We have seen in chapter 4 that the existing literature lacks sufficient understanding of NCE in conjunction with deep neural networks, which leaves NCE in a

less favourable position where its potential is hidden. We aim for a more detailed analysis of NCE in the context of RNNs with large recurrence depth as this is beneficial for long-term dependency modelling. Most of the existing analyses of NCE (Chen et al. 2015, Botev et al. 2017) are based on architectures that do not use deep RNN models (large recurrence depth) or do not demonstrate sufficient potential of NCE (Józefowicz et al. 2016). Ultimately, the existing literature does not see NCE as a regulariser. It has been mainly seen as an approximation to softmax in language modelling.

To summarise, in this chapter, we study NCE for a model that has a large recurrence depth (i.e. RHNs) to analyse the impact of NCE on deep neural networks. Specifically, we show that NCE acts as a reguraliser for the neural model because it learns an output layer of approximately lower rank (Hu et al. 2018, Tai et al. 2015) than softmax. We use spectral, sparsity, gradient, and generalised variance analysis to understand the generalisation properties. Understanding the generalisation properties is challenging in deep neural networks because of the non-linear activation and SGD optimisation. As a result, it is difficult to analyse the learned weight matrix. We propose ways to analyse the learned models that can be used to measure sparsity and low rankness which can lead us to understand the generalisation properties. To explain generalisation based on the learned model (i.e. the weight matrix) in the context of deep neural language models, we use generalised variance for the first time and provide justification in terms of classical learning theory. In our justification, we use a quantitative method based on the variance of the learned weights in the output layer of the deep neural language model. Utilising variance is appealing as this can enhance our knowledge of neural learning in terms of classical learning theory (e.g. bias-variance tradeoff). In chapter 4, the search phase of the optimisation has an impact on learning, and our analysis suggested that improved optimisation is due to the NCE induced noise in the search phase, and in this chapter, we argue that it improves generalisation by inducing low-rank regularisation through sparse learning that reduced the variance. Our results and analysis are based on the models trained in chapter 4 and RHNs trained in this chapter.

In this chapter, our contributions are as follows:

- We show that NCE can be seen as a regulariser because it leads to output layers that have an approximately lower rank than softmax, our analysis is also supported by spectral analysis.
- We show that generalised variance can be used to explain the reasoning behind the generalisation for a deep neural model.

- In the previous chapter, we argued that noise-induced by NCE accelerates the convergence in deep neural language models. In this chapter, we use variational dropout which adds more noise by dropping from the recurrent layers. Moreover, NCE uses a noise distribution and its learning of the output layer is randomised. We show that even if more randomisation in the recurrent layer is added through variational dropout, NCE still provides improvements because its randomisation in the output layer is informed by the data.
- Our analysis of gradients of the loss function in NCE and softmax indicates that NCE makes RHNs more resistant to the vanishing gradient problem (Bengio et al. 2012). As a result, it provides better regularisation for the overall network. This is confirmed through a clear correlation between the gradients in RHNs and the generalisation error.
- We show that NCE in conjunction with Recurrent Highway Networks (RHNs) (Zilly et al. 2017) leads to better generalisation than softmax. This means that NCE improves not only the standard LSTM as shown in the previous chapter, but also its specific generalisation, RHN.

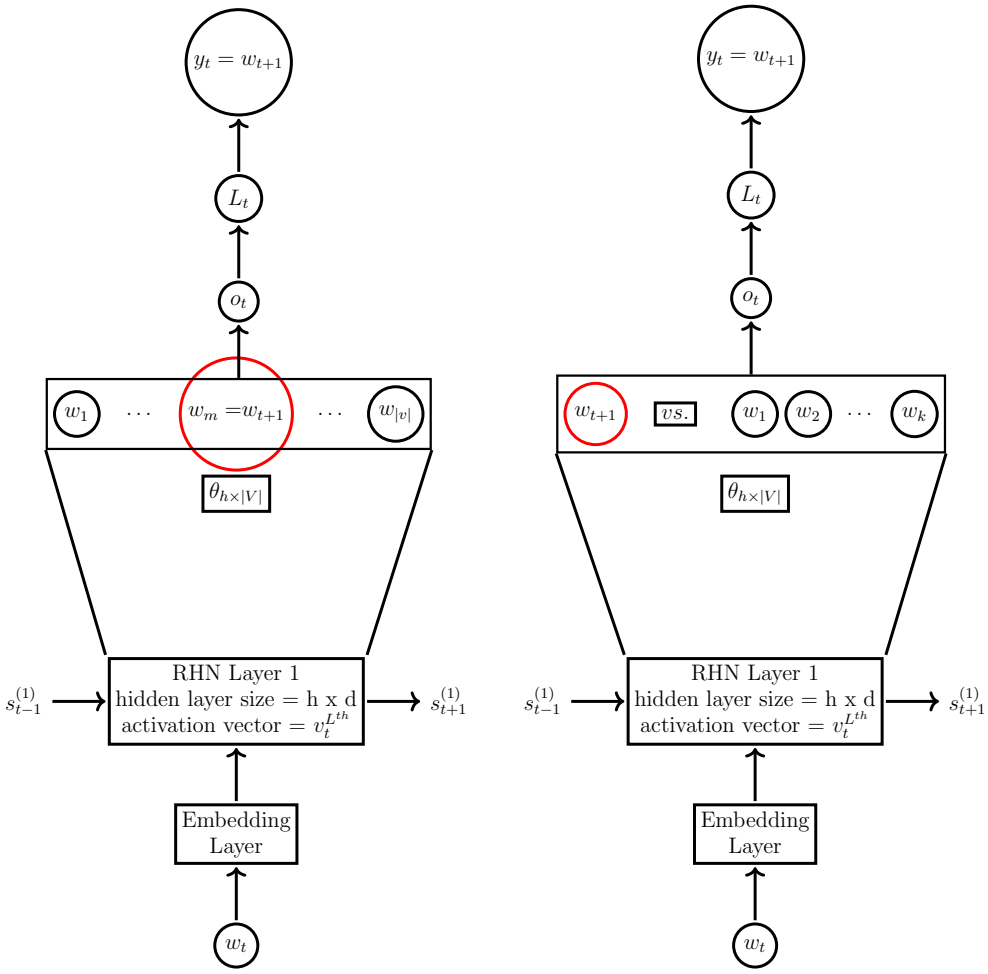
5.1 Background

The fundamental architecture of LSTM was extended and improved in various ways. In this chapter, we study Recurrent Highway Networks (RHNs) (Zilly et al. 2017) because they are an explicit generalisation of LSTM, and our goal is to verify whether NCE can improve more general LSTM-based models.

In a standard LSTM, every time step is atomic. RHNs generalise LSTM allowing every time step to be additionally divided into d microsteps, where d is a parameter. This means that the RHN model gains parameters along the dimension that corresponds to time, which increases its recurrence depth. Conceptually, when $d = 1$, an RHN reduces to one LSTM layer.

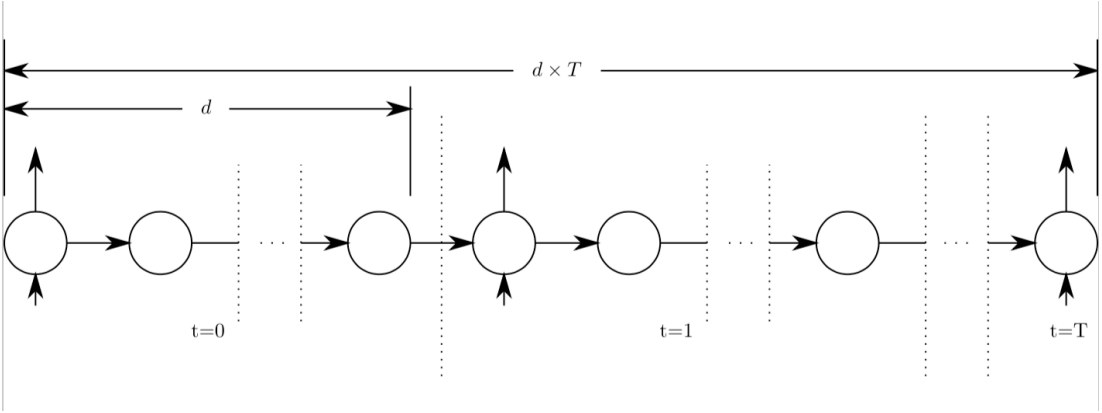
Microsteps enhance modelling power, but they also make learning challenging. For this reason, RHNs use the gating mechanism (i.e. the transform and carry gates) similar to LSTM.

In the last chapter, we have seen that two classic LSTM layers can be stacked into large blocks (similar to layers of neurons in the multilayer perceptron). The blocks of those layers are then unrolled for several time steps during learning.

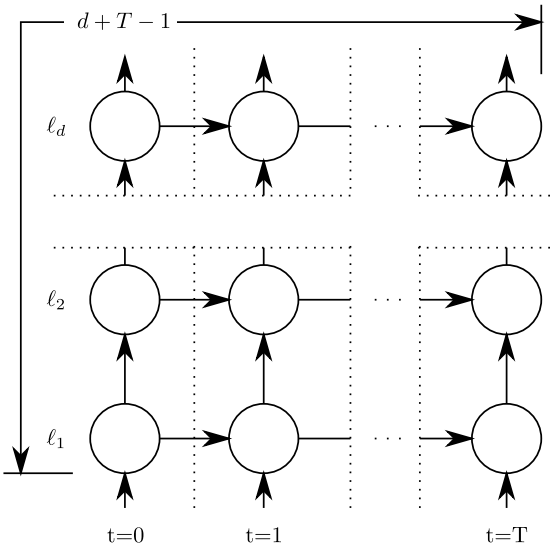


(a) Illustration of the softmax output layer. (b) Illustration of the NCE output layer.

Figure 5.1: Illustration of the output layer with RHN: the computational graph to compute the training loss of a recurrent highway network that maps an input sequence of w values to a corresponding sequence of output o values. A loss function d measures how far each o is from the corresponding training target y. The loss function d internally computes Eq. 4.3 or approximate it using Eq. 4.6. The mini-batched stochastic gradient descent optimisation will use Eq. 4.4 and Eq. 4.7 to find the parameters including the θ.



(a) Increasing recurrence depth along the time steps (Zilly et al. 2017, Fig. 1(b))



(b) Increasing recurrence depth along the space and the time steps (Zilly et al. 2017, Fig. 1(a))

Figure 5.2: Illustration of the recurrence depth of one time step in Fig. 5.1: comparison of (a) RHN of recurrence depth d and (b) stacked RNN with depth d, both operating on a sequence of T time steps. The longest learning path between hidden states T time steps is $d \times T$ for RHN and $d - T + 1$ stacked RNN .

In the existing implementations of RHNs, there is usually only one hidden layer, i.e. there is no stacking of hidden layers, but stacking is possible too.

Similar to stacked LSTM networks, RHNs are unrolled for several time steps during learning. When every physical time step in an RHN is split into d microsteps, the last d^{th} microstep is the one that defines the output of the time step. When i is the last unrolled time step, $v_i^{d^{th}}$ is the activation vector that results after the context c_i has been presented to the recurrent network. Then, the final output layer (that uses either NCE or softmax) has one vector θ_j of parameters for every word o_j in the vocabulary. We denote the matrix that consists of all the vectors θ_j stacked as rows by θ. The matrix θ of the parameter vectors θ_j will be investigated in our results section.

We can recall that when we study language models where given a sequence of words $O = (o_0, o_1, o_2, \ldots, o_T)$ over the vocabulary V, the sequence probability is

$$p(O) = \prod_{i=0}^{T-1} p(o_{i+1}|o_0, o_1, \ldots, o_i) = \prod_{i=0}^{T-1} p(o_{i+1}|c_i). \tag{5.1}$$

Here, for a given word o_{i+1}, $c_i =< o_0, o_1, \ldots, o_i >$ represents its full, non-truncated context. The probability of the next word o_{i+1} can be computed using the softmax function for the RHN:

$$P_\theta^{SOFT}(o_{i+1}|c_i) = \frac{\exp\left(\theta_{i+1}^\top v_i^{d^{th}}\right)}{\sum_{j=1}^{|V|} \exp\left(\theta_j^\top v_i^{d^{th}}\right)} = \frac{\exp\left(\theta_{i+1}^\top v_i^{d^{th}}\right)}{Z}. \tag{5.2}$$

Here, $P_\theta^{SOFT}(o_{i+1}|c_i)$ is the probability of word o_{i+1} given context c_i. θ_{i+1} is the weight vector corresponding to the word o_{i+1} in the output layer, θ_j is the weight vector for the word o_j in vocabulary, and $|V|$ is the vocabulary size. The normalising term Z is known as the partition function. Note that unnormalised products $\theta_{i+1}^\top v_i^{d^{th}}$ are not sufficient to evaluate the words.

If we compare Eq. 4.2 and Eq. 5.2, the difference is v_i^n and $v_i^{d^{th}}$ (the outputs of the representation layer). The objective and gradient formulation for softmax and NCE are similar (see section 4.2 for details) for both representations (i.e. stacked LSTM and RHN).

5.1.1 Regularisation and Generalisation

In section 2.5.2, we have seen that explaining theoretically the generalisation of a deep neural network is difficult. Several approaches have been proposed to explain the generalisation of the deep neural network (see section 2.5.2).

Both NCE and softmax learn the matrix θ in the output layer of the model. The size of the matrix is identical in NCE and softmax when they are applied to the same model and data. This means that NCE does not reduce the number of parameters in a specific comparison; the initial modelling power of NCE and softmax is identical. Using the concept of approximate low rank (Hu et al. 2018) and generalised variance, we show that NCE leads to better regularisation of the parameter matrix θ than softmax and leads to better generalisation performance.

5.2 Methods and Objectives

To determine that NCE acts as a regulariser and show that generalised variance can explain the generalisation of the deep neural model, we tested Recurrent Highway Networks (RHNs) with softmax and NCE. The parameter (weight) matrix θ learned by the algorithms (see sections 5.1, 4.2) and the learning processes were analysed. In this section, we introduce the algorithmic design used to incorporate NCE into RHN, the analysis techniques to verify that NCE is a regulariser, and an understanding technique for generalisation.

5.2.1 NCE in Recurrent Highway Networks

In this section, we have mostly used the learning approach described in chapter 4. Following section 4.3, we controlled the learning rate, $\eta(t)$, using the 'search-then-converge' procedure of the form:

$$\eta(t) = \eta_0 \times \left(\frac{1}{\psi}\right)^{\max(t+1-\tau,0.0)}. \tag{5.3}$$

The hyperparameter ψ is kept constant in our experiments, and its value was set according to (Zilly et al. 2017). The equation defines two stages of learning: the search period for $t \leqslant \tau$, and the convergence period for $t > \tau$. We used a long search period (i.e. large τ) following (Liza & Grześ 2018). The learning rate, $\eta(t)$, is constant and equal to η_0 during the search period.

Weights were initialised according to the heuristics suggested by Glorot & Bengio (2010) with a smaller range as discussed in the last chapter (see. 4.3). Precisely, the initial weights were sampled from the following distribution:

$$U\left(-\frac{\frac{\sqrt{6}}{\sqrt{n_i+n_{i+1}}}}{4}, \frac{\frac{\sqrt{6}}{\sqrt{n_i+n_{i+1}}}}{4}\right) \tag{5.4}$$

where U is a uniform distribution and n_i is the number of nodes in the layer i.

We sampled noise from the power-law distribution based on the analysis in section 4.3.3.

Adding more noise using variational dropout

In chapter 4, we have seen that noise helps in improving generalisation. We used standard dropout (Hinton et al. 2012) in that chapter. The standard dropout can be interpreted as a way of regularising a neural network by adding noise to its hidden units (Srivastava et al. 2014). They showed that adding noise is not only useful for unsupervised feature learning as shown in Vincent et al. (2008, 2010), but can also be extended to supervised learning problems. The variational dropout (Gal & Ghahramani 2016*b*) is an extension of the standard dropout which has a theoretical grounding in the application and requires the dropout to be applied in all layers in RNN (including recurrent layer). Following Zaremba et al. (2014), in the last chapter, we did not apply dropout in the recurrent layer. In this chapter, in all our experiments, we used RHNs with variational dropout (Gal & Ghahramani 2016*b*) because we expect that this will improve performance of both models (i.e. models trained using softmax and NCE) due to the enhanced incorporation of noise. The enhanced noise is due to the fact that hidden nodes in the recurrent layer will be dropped randomly and the resulting randomisation will induce noise in the recurrent layer. We also expect that this will be more beneficial for softmax as it has less provision for inducing noise compared to NCE.

5.2.2 NCE as a Regulariser

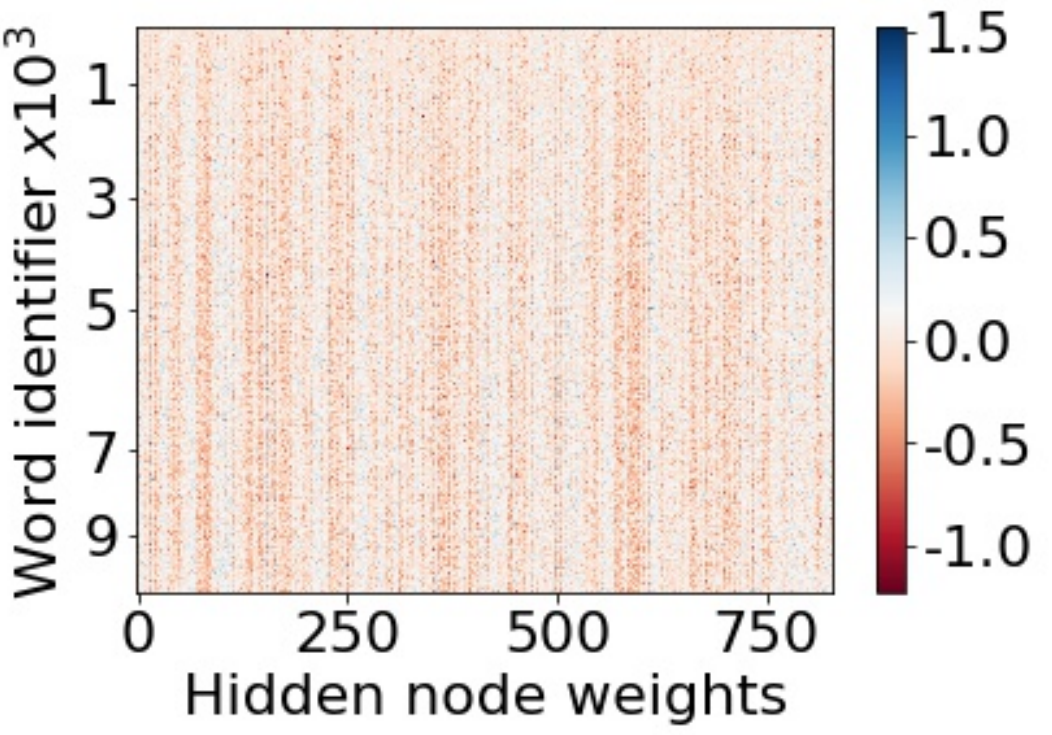


Figure 5.3: NCE on PTB

Our intuition about NCE as a regulariser has come from the visual analysis of the parameter matrix (see Fig. 5.3). The structured pattern in the parameter matrix indicates that appropriate integration of NCE with neural models perhaps fosters sparse and low-rank structures (Sanyal et al. 2018, Langeberg et al. 2019) in the parameter matrix. In practice, these structures are observed after applying the regularisation that promotes sparsity and low-rankness (Langeberg et al. 2019, Figure 1). The observed better generalisation by NCE can potentially be related to the structured sparsity regularisation and the low-rank regularisation induced by NCE in the neural training.

In general, structured sparsity regularisation extends and generalises the variable selection problem that characterises sparsity regularisation (Yuan & Lin 2006, Obozinski et al. 2011). Sparsity and low-rank regularisation are related and can co-occur in a model. For example, simultaneous sparsity and low-rankness are observed in linear models (Langeberg et al. 2019). Moreover, the effective sparsity of the vector of singular values of a matrix is called the effective rank of such matrix (Langeberg et al. 2019).

To increase sparsity, weights pruning and model pruning techniques have been used in the literature (Han et al. 2015, Lebedev & Lempitsky 2016, Zhou et al. 2016, Li et al. 2017). Motivation for weight pruning is to make efficient the over-parametrised neural network which has significant parameter redundancy (Denil et al. 2013). For example, Han et al. (2015) proposed a three-step method that prunes redundant connections. An interesting visual observation that they have presented was a sparsity pattern that was revealed after applying pruning (Han et al. 2015, figure 4). With our visual analysis, we assume that we get the similar weight pruning effect as a by-product of NCE approximation without applying any specific weight pruning methods. It is perhaps worth investigating in the future how each of these sparse sub-networks (i.e. corresponding to the dark band and light bands) could be trained to learn a specific task (Golkar et al. 2019).

Quantifying sparsity is challenging in neural networks

Now, the question is if the weight matrix is sparse and has low rank, why don't we just calculate the rank to verify the low-rankness. The problem is that when SGD is used as an optimisation method for deep neural network, SGD does not yield actual zeros for the parameters but values hovering around zeros (Bengio 2012). Thus the classical verification of low rankness and thus sparsity measurement is difficult to apply for deep neural network parameter matrices, as deep neural network parameter matrices are usually full ranked. Based on this understanding, we further investigate the weight matrix and in this section, we are going to describe quantitative methods (i.e. spectral analysis and gradient analysis) to measure sparsity and low-rankness to justify our claim that NCE works as a regulariser for deep neural language models and to enhance the understanding of the generalisation performance based on learned models. To measure the generalisation performance based on the existing measures (i.e. error gap (Goodfellow et al. 2016, Neyshabur 2017)), the perplexity of the model based on training and test data for each epoch are calculated and the relevant results are described in section 5.4. The rest of the section is organised as follows: approximately low rank is explained in the section 5.2.2.1. Subsequently, we introduce the gradient analysis in section 5.2.2.2, which allowed us to see a clear correlation between generalisation and the generated gradients values during training. As our objective is to see the generalisation in terms of learned models, the gradient-based results provide more evidence that NCE can be seen as a regulariser while providing an understanding of the change in the learning process that results in improved regularisation.

5.2.2.1 Low Rank Analysis

Sparsity and low-rank regularisation are important features of machine learning algorithms because simpler models are less likely to overfit (Hinton et al. 2012, Hu et al. 2018). Algorithms that learn weight matrices of (at least approximately) lower rank have more structured parameter spaces (Langeberg et al. 2019). In our analysis, we focus on the parameter matrix θ that has the same size in NCE and softmax, which means that NCE does not reduce the number of parameters in any way, but we can show that it learns θ of approximately lower rank than softmax. In many applications, low-rank regularisation (Hu et al. 2018) uses L1-norm (aka LASSO) to reduce the approximate rank of the parameter matrix. Similarly, nuclear norm regularisation is used to promote low-rankness (Langeberg et al. 2019). In this chapter, we show that everything else being equal, NCE-based learning without any explicit regularisation leads to approximately lower rank than softmax. This means that without using LASSO or nuclear norm regularisation, NCE has the effect of low-rank regularisation although the matrices are of full rank.

Our arguments presented in the results section use several techniques. First, the visual analysis (see Fig. 5.6) of the matrices θ shows that NCE's matrices are less random, which means that they have more structure and this becomes qualitative evidence of (approximately) lower rank. Second, we perform a quantitative analysis using the results of the Singular Value Decomposition (SVD) of normalised θ (where the columns of θ are normalised) or equivalently Principal Component Analysis (PCA) of its correlation matrix. Having SVD/PCA results, we compute the cumulative variance carried by an increasing number of the strongest principal components. This is a classic method (Jolliffe 2002) to argue that the model with faster growth of the cumulative variance is more structured. Additional evidence is provided by the reconstruction error when the θ matrices were reconstructed using the results of SVD and a reduced number of dimensions. θ that can be reconstructed with smaller error given the same number of principal components is of approximately lower rank (Manning et al. 2008, section 18.3).

Quantitative approach is more robust compared to qualitative approach.

The approximate lower rank and structured sparsity improve training and generalisation of the deep neural network (Wen et al. 2016, Zhu et al. 2018). To explain the generalisation, we have used the generalised variance (Wilks 1960) in section 5.2.3, which also provides evidence to support our argument about NCE regularisation.

5.2.2.2 Gradient Analysis

To understand why NCE converges to a better solution, in this section, we analysed the relationship between gradient propagation and generalisation. We investigated the distribution of the gradient values in hidden cells at two microsteps (precisely, the first, h_1, and the last, h_9, microsteps) of the RHN. This selection of microsteps is motivated by the fact that the vanishing gradients will have the smallest impact on h_9 and the largest impact on h_1 because they are the furthest (h_1) and the closet (h_9) distance from the output layer. We use h_1 instead of h_0 when $L = 10$, because h_1 and h_9 have the same activation (tanh) and the same numbers of units. Insightful conclusions are drawn in our results section because the relationship between the gradient in the selected microsteps and perplexity depends on the output layer of the entire network, i.e., whether NCE or softmax is used. To the best of our knowledge, there was no previous work where these two relationships were explored. This analysis also shows that with NCE, the entire network receives better regularisation because NCE has a positive impact on the gradients in the lower layers of the network.

5.2.3 Understanding and Explaining Generalisation

The theoretical formulation and measurement of generalisation in the context of deep neural learning is still an open problem (Neyshabur et al. 2019, Martin & Mahoney 2019). Despite existing work on ensuring generalisation of neural networks in terms of scale sensitive complexity measures, such as norms, margin and sharpness (see section 2.5.2), these complexity measures do not offer an explanation of why neural networks generalise better with over-parametrisation. For example, in our experiments, the parameter matrices are the same for both Softmax and NCE, however, NCE offers improved generalisation. We can ask ourselves **why softmax overfits compared to NCE having the same number of model parameters**.

Our discussion in section 5.2.2 is to argue that NCE works as a regulariser for deep neural language models and generalises better than softmax because NCE learns the output layer weight matrix of approximately lower rank. In this section, we will describe a fundamentally inspired empirical approach to understand generalisation for deep networks, especially for comparing learned models. One can argue that, when we have the training perplexity and test perplexity and we can see the gap between these two metrics to measure the generalisation, then why do we need a new technique. Although the existing approach indicates generalisation, it does not say much about the generalisation in terms of the learned weights and thus does

not offer an understanding of the learned model. The existing approach requires us to rely on the test dataset, we don't know what is exactly happening in the training model (i.e. weight matrix) that leads to better or worse generalisation.

Black box model.

Classical Learning Theory and Weight Values In section 2.5, we have discussed the bias-variance tradeoff for neural networks. Geman et al. (1992) suggested that large neural networks suffer from higher variance which results in poor generalisation. Generally, neural networks learn a set of weights that best map inputs to outputs. A network with large network weights can be a sign of an unstable network where small changes in the input can lead to large changes in the output (Reed & MarksII 1999). This causes a large variance in models. If a model has large weights which result in poor generalisation on test data, we refer to that model as having a large variance and a small bias. That is, the model is sensitive to the specific examples, the statistical noise, in the training dataset. This can be a sign that the network has overfitted the training dataset (including the statistical noise) and will likely perform poorly when making predictions on new data (i.e. test data). Thus the magnitude of weight is related to the model variance and test performance (i.e. generalisation).

A model with large weights, hence is overly specialised (including the statistical noise) to training data, is more complex (i.e. has higher capacity than needed to model the task) than a simpler model with smaller weights. The regularisation methods reduce the variance by keeping the weights small and improve the generalisation on test data by reducing the generalisation error (Goodfellow et al. 2016). Traditional wisdom in learning suggests that using models with increasing capacity will result in overfitting to the training data. Therefore, in the traditional approaches, the capacity of the models is generally controlled either by limiting the size of the model (number of parameters) or by adding an explicit regularisation, to prevent from overfitting to the training data by keeping the weights smaller. Many traditional regularisation approaches are based on limiting the complexity (i.e. capacity) of models by adding a penalty to the objective function. For example, weight decay regularisation aims to prevent overfitting by extending the objective function to include the goal of model simplicity. This aim is fulfilled by penalising the size of weights by adding to the objective function each of the weights squared, multiplied by some regularisation parameter (Bishop 2006).

Complex vs. simple, traditional modelling vs black box modelling

Controlling the complexity of the deep neural model is not a simple task, because finding the model of the right size, with the right number of parameters is difficult for neural network generalisation (Goodfellow et al. 2016). Moreover, deep

models work better with an over-parametrised model. Therefore, the traditional understanding of generalisation is not sufficient and requires rethinking (Zhang, Bengio, Hardt, Recht & Vinyals 2017). In practice, for deep learning, the best-fitted model (in the sense of minimising generalisation error) is a large model (i.e. over-parametrised) that has been regularised appropriately (Goodfellow et al. 2016). On the contrary to the traditional regularisation approach that we have discussed in the last paragraph, in the case of neural networks increasing the model size only helps in improving the generalisation error, even when the networks are trained without any explicit regularisation (e.g. weight decay or early stopping) (Lawrence et al. 1998, Srivastava et al. 2014, Neyshabur et al. 2015*a*). This contradiction makes the theoretical analysis of generalisation difficult for a neural model and there is no standard implicit regularisation measurement metric for over-parametrised large models. In this section, we are proposing a new explanation technique of generalisation based on the generalised variance.

Regularisation measurement for implicit regularisation.

Generalised Variance The generalised variance (GV) of a p-dimensional random vector variable X is defined as the determinant of its variance-covariance matrix and has several interesting interpretations (Sengupta 2004). The generalised variance can be seen as a measure of overall multidimensional scatter. It can be seen as the expected volume in which data represented by a two-dimensional matrix (e.g. weight matrix θ) is located. Smaller generalised variance means that the data is located in a smaller volume. GV was introduced by Wilks (Wilks 1932) as a scalar measure of overall multidimensional scatter. The determinant of the covariance matrix could be considered a generalisation of the variance, in that it is analogous to the scalar variance in the case of dimension one.

The intuition behind the generalised variance as a generalisation measurement tool is related to the weight magnitude described in this section. From the discussion in this section, we understand that having large weights is related to poor generalisation and results in overfitted models (i.e. poor generalisation). Therefore, we don't want large weights, however, defining 'large' is subjective and specific to the model in consideration. Generalised variance, on the other hand, gives an indicative overall measure of the size of the weights in a parameter matrix. Intuitively, the variance for one-dimensional weight variable defines how far each weight in the weight vector is from the mean, the generalised variance will represent the distance (size) of weights from the mean value of the weight matrix. Therefore, large generalised variance indicates the presence of large weights in the weight matrix. Based

on this understanding, we are proposing generalised variance of the weight matrix to explain the generalisation.

Given two weight matrices, the one with smaller generalised variance is better regularised and expected to generalise better on unseen data.

We are proposing this method primarily to compare the regularisation of models. For example, between two models, the model with a smaller generalised variance will define better regularisation and will result in improved generalisation. We have tested this claim in section 5.4.4.

5.3 Experimental Setup

To show the potential of NCE for language modelling, we investigate its performance in comparison with softmax because NCE approximates softmax, being consistent with softmax in the limit. Our implementation follows the configuration choices presented in section 5.2.

Since we use softmax in our evaluation, it is necessary to have a dataset on which it is feasible to run softmax without any approximations. Therefore we use a popular benchmark, Penn Treebank (PTB), that was also used in the previous chapter. WikiText-2 (WT2) is another benchmark that is known for long-term dependencies, as the dataset is composed of full articles. It has less preprocessing than PTB as it does not remove the original case, punctuation, and numerical tokens. We have not done extensive hyperparameter tuning for the WT2 dataset, and we have used most of the hyperparameters based on tuning on the PTB dataset. Hyperparameter tuning on the WT2 dataset was left for future work. WT2 is over two times larger than PTB and has a larger vocabulary. As shown in the Tab. 5.1, vocabulary sizes are 10,000 and 33,278 words in PTB and WT2 respectively. The training, validation, and testing datasets contain 887521, 70390, and 78669 tokens in PTB and 2088628, 217646, and 245569 tokens in WT2 [1].

All our models were implemented in Tensorflow[2] and executed on NVIDIA P100 GPUs. The standard components of our models follow (Zilly et al. 2017) where excellent results on PTB were reported. In our experiments, the model has 1 hidden RHN layer and every time step is divided into L = 10 microsteps. Every microstep contains 830 hidden nodes. Variational dropout (Gal & Ghahramani 2016*b*) was used

[1]https://einstein.ai/research/the-wikitext-long-term-dependency-language-modeling-dataset
[2]https://www.tensorflow.org/

Table 5.1: Summary of datasets

	Penn Treebank (PTB)			WikiText-2		
	Training	Validation	Testing	Training	Validation	Testing
Tokens	887,521	70,390	78,669	2,088,628	217,646	245,569
Vocabulary size	10,000			33,278		

for regularisation. For robust training, it adds noise at every layer (input, hidden, output) of the network, and it can be seen as an implementation of approximate variational inference. Weight decay (with a value of 1e-7) and weight tying were implemented following (Zilly et al. 2017, Inan et al. 2017, Press & Wolf 2016). We have run the experiment without the weight decay and the result was similar to the one with the weight decay. The impact of weight decay was not observed on the final performance.

We used two sets of dropout rates for the input, output, embedding, and hidden layers respectively: $[0.65, 0.65, 0.20, 0.20]$ for PTB, and $[0.55, 0.55, 0.20, 0.20]$ for WT2. The RHN was unrolled for 35 time steps. We used mini-batch stochastic gradient descent for training where the mini-batch size was 20. The initial weights were sampled using the uniform distribution shown in Eq. 5.4.

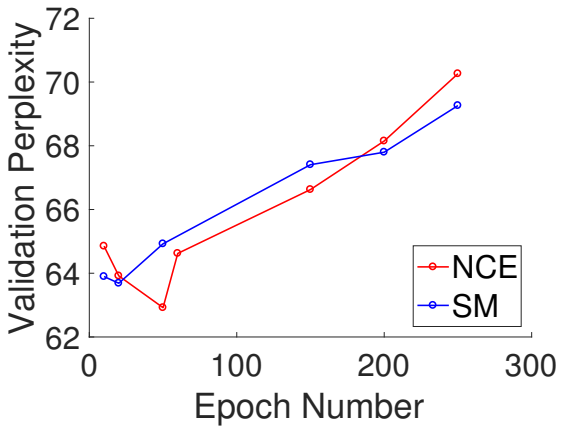


Figure 5.4: Selection of the learning rate parameter τ for the PTB dataset

The learning rate was scheduled using Eq. 5.3. The search time limit τ was chosen empirically using Fig. 5.4. As a result, τ was set to 20 and 50 for softmax and NCE respectively, which means that during the first τ epochs, the learning rate was equal to η_o, where $\eta_o = 0.2$. Afterwards, during the convergence period, the learning rate was decreased by the factor of $1/\psi$ where ψ was set to 1.02. We trained the models for 500 epochs.

The norm of the gradients (which was normalised by the mini-batch size) was clipped at 10. In NCE, we used 600 noise samples for PTB based on the experiments in the last chapter and 1200 noise samples for WT2 as the vocabulary is larger than PTB. The noise samples were generated from the power-law distribution.

When the model is learned using NCE, its output layer activations may not represent a probability distribution that is normalised. For this, whenever we use validation or testing data to compute perplexities for models learned using NCE, we normalise those probabilities explicitly to make sure that our results are accurate. Note that explicit normalisation is not used during NCE learning.

We have seen in section 5.2.3 that generalised variance can be calculated using the determinant of the covariance matrix. Due to the computational problems with calculating determinants on covariance with a large matrix (Bai et al. 1996), the direct calculation of determinant on covariance matrix is problematic. We have calculated generalised variance using the eigenvalues of the covariance matrix. Concretely, if C is an $n \times n$ covariance matrix of the weight matrix θ, then we know that the product of the n eigenvalues of C is equal to the determinant of C. This is based on the following theorem[3]:

Theorem 1 *If A is a $n \times n$ matrix, then the sum of the n eigenvalues of A is the trace of A and the product of the n eigenvalues is the determinant of A.*

5.4 Results and Discussion

This section addresses our objectives stated in section 5.2. First, we will analyse the results of the generalisation, second, we will analyse the impact of NCE regularisation on the generalisation, and third, we will explain the generalisation using the generalised variance.

[3]https://www.adelaide.edu.au/mathslearning/play/seminars/evalue-magic-tricks-handout.pdf

Table 5.2: Comparing the running time of softmax and NCE

Dataset	Model	Time
PTB	softmax	1d 8h 58m 52s
WikiText-2	softmax	4d 11h 29m 45s
PTB	NCE	1d 5h 15m 19s
WikiText-2	NCE	3d 8h 40m 8s

Table 5.3: Results on PTB using neural language models with variational dropout, recurrence depth $L = 10$ in Recurrent Highway Networks, and hidden size 830

Internal memory based model	τ	Valid. PPL	Test PPL
Variational LSTM + Weight Tying (WT) (Press & Wolf 2016)		75.8	73.2
Variational RHN with WT (Zilly et al. 2017)	20	67.9	**65.4**
AWD-LSTM-3-layer LSTM (WT) (Merity et al. 2017)		60.0	57.3
Variational RHN with softmax with WT	20	63.686 (0.12)	61.982 (0.06)
Variational RHN with NCE with WT	50	63.074 (0.09)	**60.502** (0.04)
Variational RHN with softmax without WT	50	69.558 (0.18)	67.154 (0.08)
Variational RHN with NCE without WT	50	66.296 (0.10)	63.014 (0.05)
External memory (Continuous Cache(CC)) equipped model	τ	Valid. PPL	Test PPL
AWD-LSTM-3-layer LSTM with CC pointer (Merity et al. 2017)		53.9	**52.8**

Table 5.4: Results on WiKiText-2 using neural language models with variational dropout, recurrence depth L = 10 in Recurrent Highway Networks, and hidden size 830

Internal memory based model	τ	Valid. PPL	Test PPL
Variational LSTM (Merity et al. 2016)		101.7	96.3
Zoneout + Variational LSTM (Merity et al. 2016)		108.7	100.9
AWD-LSTM - 3-layer LSTM (WT) (Merity et al. 2017)		68.6	**65.8**
Variational RHN with softmax with WT	20	74.182 (0.17)	71.791 (0.09)
Variational RHN with NCE with WT	50	70.647 (0.11)	**68.054** (0.05)
Variational RHN with softmax without WT	20	79.712 (0.16)	77.753 (0.10)
Variational RHN with NCE without WT	50	75.643 (0.20)	73.080 (0.09)
External memory equipped model (Continuous Cache(CC))	τ	Valid. PPL	Test PPL
LSTM CC (size = 100) (Grave, Joulin & Usunier 2017)		-	81.6
LSTM CC (size = 1000) (Grave, Joulin & Usunier 2017)		-	**68.9**
AWD-LSTM - 3-layer LSTM with CC pointer (Merity et al. 2017)		53.8	52.0

5.4.1 Improving Generalisation of RHNs

The results in Tab. 5.3 and 5.4 compare NCE with other state-of-the-art methods on Penn TreeBank (PTB) and WikiText-2 (WT2) datasets. This chapter focuses on neural networks that do not use external memory, but we include the existing results of such networks (Grave, Joulin & Usunier 2017, Merity et al. 2017) in our tables for comparison. The results of this chapter could generalise to those models with external memory, and NCE's improvement could be found there as well. Each experiment was repeated three times and the table reports the results in the mean (standard

deviation) format.

On PTB, NCE achieved perplexity of 60.502, which is better than the current state-of-the-art solution with the Recurrent Highway Network model (Zilly et al. 2017) by 4.898. To achieve this result, NCE took 3 hours 43 minutes less time than softmax. Thus, NCE is both faster and more accurate.

As described in section 5.3, we did not tune our methods extensively for the WT2 dataset due to time constraints (the dataset is relatively large). However, we can see in Tab. 5.4 that NCE performs better than softmax by 3.737 with weight tying (WT) and by 4.673 without WT. WT regularisation reduces the difference by 0.936, but it does not account for the improvement brought by NCE. So, NCE can improve on top of other well-known regularisation techniques. NCE being an approximation to softmax has better perplexity than softmax and took 1 day 2 hours and 50 minutes less time than softmax. It is crucial to remember that as the vocabulary size becomes larger (e.g. in WT2), the training time difference becomes larger, and the gap is expected to increase on larger vocabularies. Exact softmax becomes infeasible in those cases. In our results, on PTB with the vocabulary size of 10K words, the time difference was 3 hours 43 minutes. In WT2 the vocabulary size was only 3.3 times larger, but the time difference was over 6 times higher (see Tab. 5.2). The times in Tab. 5.2 are sums of training, validation, and testing times.

Observing the performance improvement compared to softmax, in the subsequent sections, we analyse NCE's impact on the learning process in more detail, and we show that NCE can be seen as a regulariser for deep recurrent neural networks.

5.4.2 NCE As A Regulariser

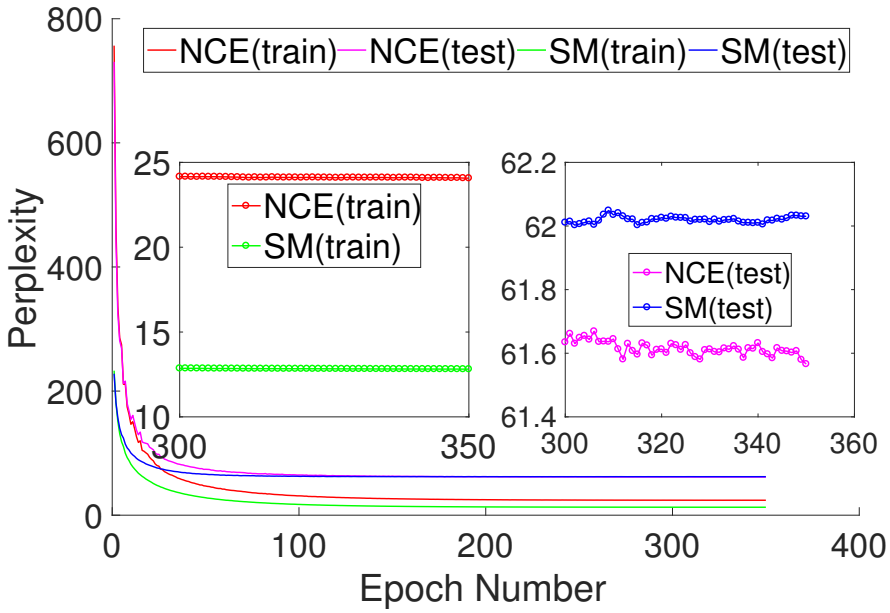


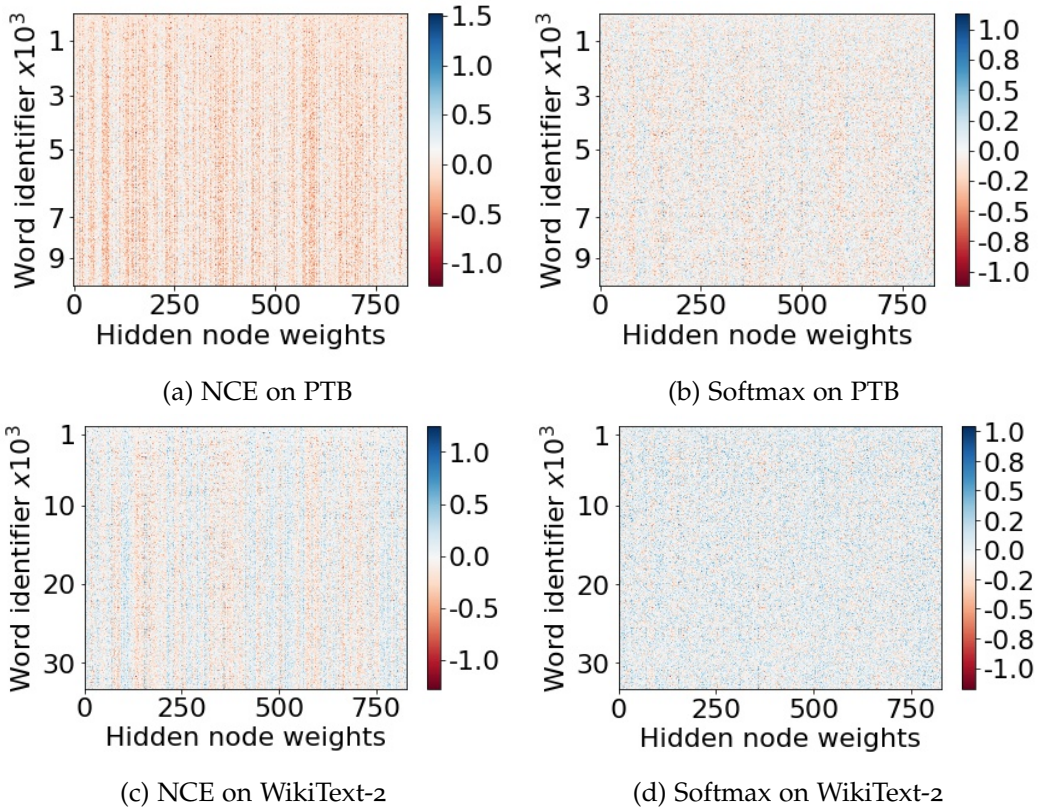
Figure 5.5: NCE as a regulariser

As we have discussed before, according to Goodfellow et al. (2016), Neyshabur (2017), a regulariser should reduce the generalisation error but not the training error. To see the NCE's impact on the whole deep neural network, in Fig. 5.5, we plotted the test and train perplexities of NCE and softmax. In the figure, the x axis corresponds to the epoch number and y axis corresponds to the perplexity of the model. We can see that NCE's training perplexity is higher than that of softmax whereas the testing (generalisation) perplexity is lower using NCE. Thus, NCE decreases overfitting reducing the gap between the training error and the testing error from 49 to 37.6 for the final model, which is the evidence of regularisation (Goodfellow et al. 2016, Neyshabur 2017). The improved generalisation of NCE reported so far shows that NCE has a regularisation effect. Subsequent sections investigate the reason for the regularisation observed without the explicit parameter reduction in terms of low-rank approximation.

5.4.3 Low Rank Analysis / Regularisation

In section 5.1, we denoted the matrix of parameter values that are learned in the output layer by θ. For a particular model and dataset, the size of the matrix, θ, is the same regardless of NCE or softmax being used for learning. Therefore, having

all the other elements of the models equal, we can compare the matrices θ learned with NCE and softmax. Given that the numbers of parameters of the models are the same, we are trying to answer the question of why NCE works as a regulariser. Our intuition is that if we can show that NCE makes the weight matrix more structured (or less random), then we can explain the regularisation in terms of low-rank approximation. So, *our goal is to investigate their approximate ranks because approximately lower rank is a standard objective of low-rank regularisation (Hu et al. 2018)*. Mathematically, the weight matrix θ is full rank for NCE and Softmax. Many applications of low-rank approximation in machine learning problems are puzzling as the matrices are in general full rank, although they can be computationally feasible or more efficient if being approximated using lower rank (Udell & Townsend 2019). For the deep neural network weight matrix, this statement is true because SGD does not yield actual zeros but values hovering around zeros (Bengio 2012). However, we see the low-rank structures from the visual analysis perspective. As we have described before that in practice these structures are observed after applying the regularisation that promote the sparsity and low-rankness (Langeberg et al. 2019, Figure 1). Therefore we will attempt to explain the low-rank regularisation empirically using visual and spectral analysis.



(a) NCE on PTB

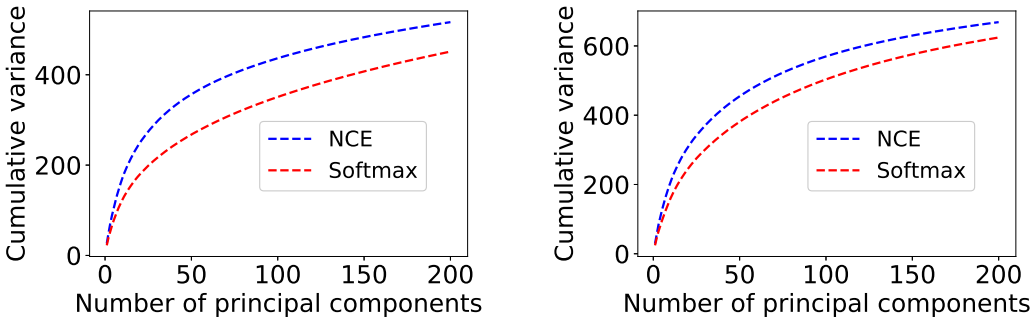
(b) Softmax on PTB

(c) NCE on WikiText-2

(d) Softmax on WikiText-2

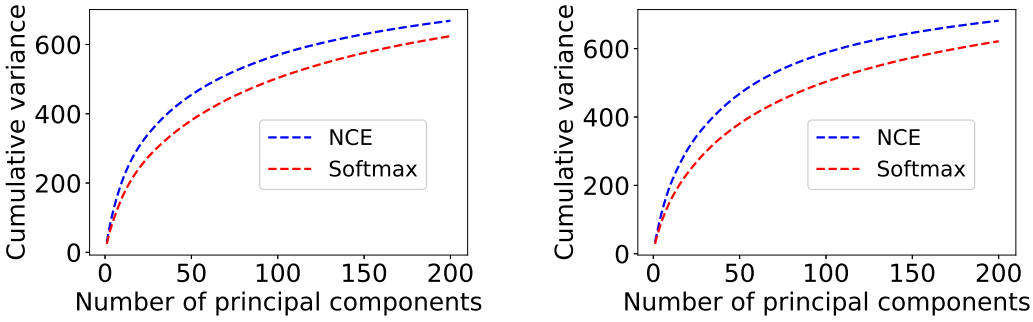
Figure 5.6: Parameters of the output layer (i.e., matrix θ) learned for the PTB and the WikiText-2 datasets

The empirical analysis uses the matrices shown in Fig. 5.6. In the matrix θ, there is one column for every hidden unit in the output layer and one row for every word in the vocabulary. Thus, in Fig. 5.6a, the matrix is 830 hidden nodes by 10K words. A qualitative visual analysis of Fig. 5.6a and 5.6b indicates that NCE learns a more structured (i.e. less random) matrix of weights, θ. Individual columns of the matrix are more pronounced in NCE, which indicates that a smaller number of principal components would be sufficient to represent the information contained in the entire matrix. The matrix learned by softmax is more random. NCE's better structure is also noticeable in Fig. 5.6c and 5.6d that show the matrices, θ, learned for WT2.



(a) PTB datase (Stacked LSTM), $n = 650$ (b) PTB dataset (RHN), $n = 830$

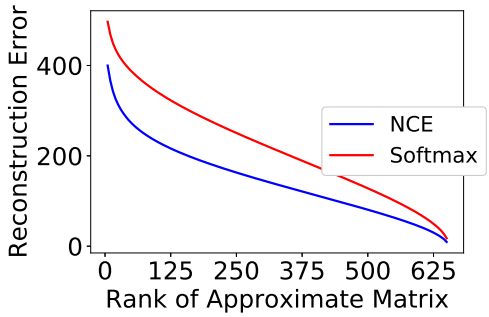
Figure 5.7: Cumulative variance analysis of the output layer (i.e., matrix θ), on Stacked LSTM and RHN



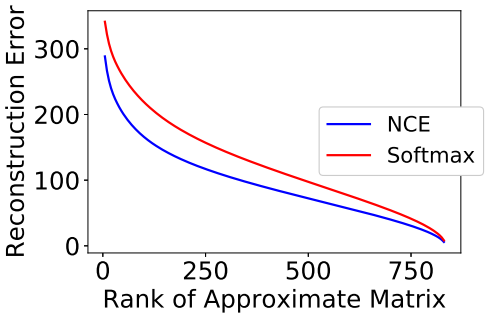
(a) PTB dataset (b) WikiText-2 dataset

Figure 5.8: Cumulative variance analysis of the output layer (i.e., matrix θ), on PTB and Wiki-Text2

Besides the qualitative analysis, as shown in section 5.2.2, we can now compute quantitative metrics to argue that NCE's matrices θ are of approximately lower rank. The first measure is the cumulative variance that is carried out by a growing number of principal components (PCs). The results presented in Fig. 5.14a and 5.14b show that in NCE the same number of PCs carries more information than in softmax. This is one of the ways to show that NCE's approximate rank is lower than in softmax.

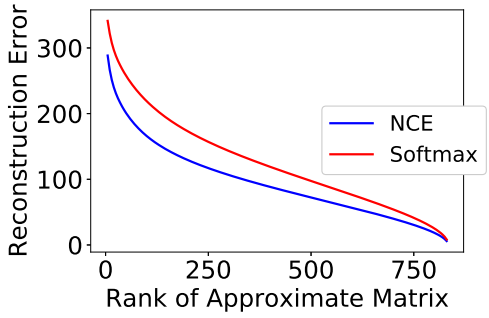


(a) PTB datase (Stacked LSTM), $n = 650$

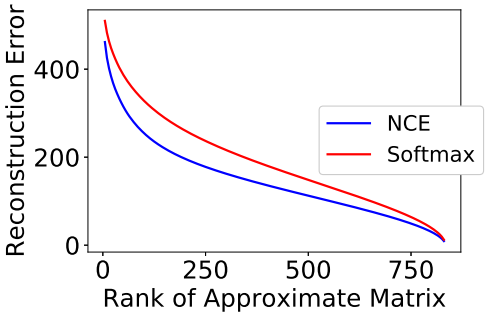


(b) PTB dataset (RHN), $n = 830$

Figure 5.9: Reconstruction error of the output layer (i.e., matrix θ) Stacked LSTM vs RHN



(a) PTB dataset



(b) WikiText-2 dataset

Figure 5.10: Reconstruction error of the output layer (i.e., matrix θ), on PTB and Wiki-Text2

To support the validity of our spectral variance based low-rank arguments, we will now describe the low-rank reconstructions error, the error between the original matrices θ and their low-rank reconstructions (Eckart & Young 1936). We use reconstructions derived from the SVD of θ, and we investigate different ranks of the reconstructed matrices (the horizontal axis in Fig. 5.9, 5.10, 5.15). The quality of the reconstruction (the vertical axes in Fig. 5.9, 5.10, 5.15) is measured as the Frobenius distance between the original and the reconstructed matrices. These figures clearly show that for a specific rank (i.e. for a specific point on the horizontal axis) NCE's matrices θ are reconstructed with higher accuracy, which confirms that NCE's matrices θ have approximately lower rank than softmax.

Our discussion above suggests that NCE acts as a low-rank regulariser. We have described in section 5.2.3 that for deep neural networks, the classical methods

of understanding generalisation do not work and we describe a generalised variance-based approach to understand the generalisation based on the trained models. In the next section, we will describe generalisation due to the low-rank regularisation induced by NCE.

5.4.4 Explaining Generalisation: Generalised Variance

Table 5.5: Generalised Variance and Generalisation Performance

Model	Generalised Variance	PPL
Stacked LSTM (NCE) on PTB	1.4×10^{-1265}	69.9
Stacked LSTM (SM) on PTB	1.0×10^{-1050}	78.4
RHN (NCE) on PTB	2.1×10^{-2002}	63.0
RHN (SM) on PTB	4.4×10^{-1805}	67.0
RHN (NCE) on WK2	6.9×10^{-2121}	73.1
RHN (SM) on Wk2	2.5×10^{-1941}	77.8

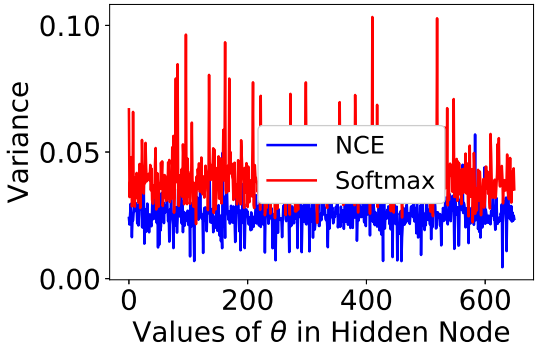


Figure 5.11: Weight Variance (Stacked LSTM)

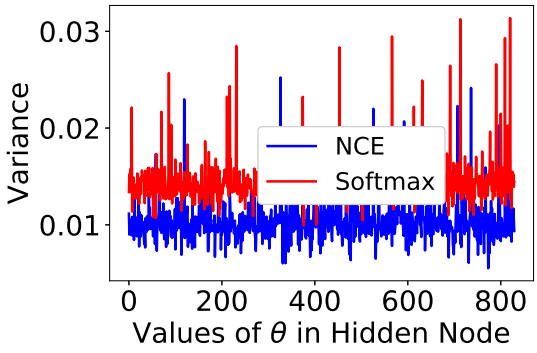


Figure 5.12: Weight Variance (RHN and PTB dataset)

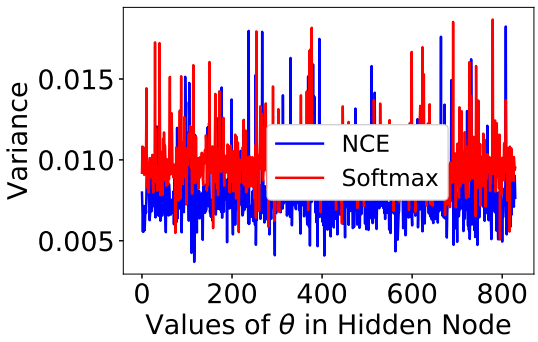


Figure 5.13: Weight Variance (RHN and Wiki-text2 dataset)

We have discussed in section 5.2.3 that generalised variance can provide generalisation explanation that can improve our understanding of the learned model based on classical learning theory (i.e. the bias-variance tradeoff). In this section, we will report the results obtained from the experiments. Tab. 5.5 reports the generalised variance of the θ matrix for NCE and softmax models. On PTB, the generalised variance is 6.1×10^{-1604} for NCE and 4.3×10^{-1520} for softmax, which means that NCE's volume is 84 orders of magnitude smaller. On WT2, the values are 5.1×10^{-1547} and 5.0×10^{-1449} respectively. We have also calculated the generalised variance for the stacked LSTM models from chapter 4. On, PTB, the values are 2.6×10^{-1263} and 1.3×10^{-1050} respectively. Thus the total amount of multidimensional variance is smaller in NCE and this gives a generalisation explanation of the trained model. These results indicate that the low-rank regularisation that we have observed in the last section can be explained with a metric (i.e. generalised variance).

As this generalisation measurement for comparing two methods is based on the actual weights of the learned models (not on the gap between training and test

accuracy), we can describe the generalisation based on the variance in the weights matrix. The generalised variance of NCE is smaller than SM and thus the weights of the matrices θ have less variance and are less likely to have large weights compared to softmax. The Test PPL is also aligned to the results. We want to see the actual variance of the weights in each hidden node for words. In Fig. 5.11 and 5.13 we see that the variance is smaller in RHN compared to Stacked models, NCE has smaller variance compared to softmax and the weights are smaller in NCE.

Based on the above analysis we can see that NCE works as a regulariser and generalised variance can be used to explain the generalisation based on the weight matrix. In the next section, we will analyse NCE based on reduced co-adaptation and induced sparsity.

5.4.5 Reducing Co-adaptation and Inducing Sparsity

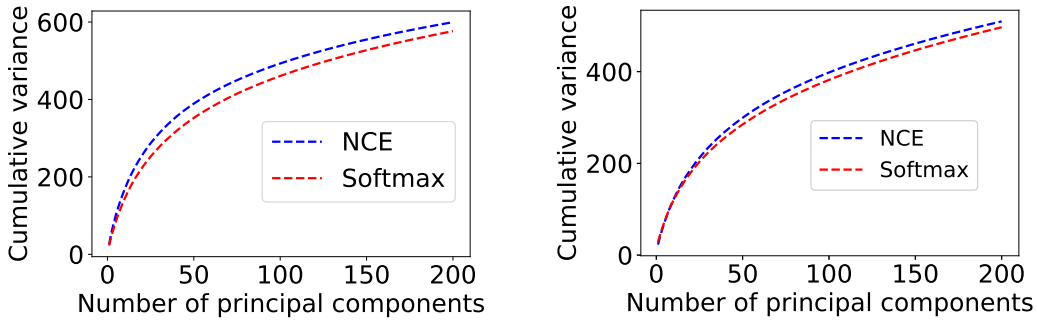
The qualitative and quantitative results presented above have an intuitive explanation based on the existing regularisation techniques. The dropout is a regularisation technique for deep neural models. This regularisation reduces the co-adaptation of the over-parametrised neural models (Hinton et al. 2012, Srivastava et al. 2014). We are going to analyse how NCE reduces the co-adaptation in weight matrix θ. In softmax, the entire weight matrix, θ, needs to be updated in every training iteration, which leads to co-adaptation in the weight matrix θ. NCE does not update all rows of the matrix in every iteration because only words that are sampled from the noise distribution (P_n in section 5.1) and the true labels of the mini-batch are updated. To explain this elaborately, we will consider Fig. 5.1. For softmax, the output is a V-dimensional probability vector where V is the vocabulary size. The forward pass involves multiplication of the h-dimensional internal representation with the $\theta^{h \times V}$ matrix and normalisation of the result. The NCE model, on the other hand, only needs the scores of the correct word and k additional noise sample words during training. This involves extracting $k+1$ rows from the $\theta^{h \times V}$ matrix, multiplying the h-dimensional internal representation with the resulting $(k+1) \times h$ matrix and no normalisation. Thus in NCE, not all rows of the θ matrix get updated at each iteration. Therefore, NCE's selected updates can arguably reduce co-adaptation and lead to more structured vectors of parameters. This is what we demonstrated in section 5.4.3. The explanation is based on the reduced co-adaptation that also justifies dropout regularisation (Srivastava et al. 2014).

Dropout is expected to reduce co-adaptation in the entire network, whereas

our results show that NCE additionally reduces co-adaptation in the output layer (i.e. θ weight matrix). We could conjecture that NCE can achieve that due to informed noise sampling that is based on the power-law distribution over the vocabulary. Dropout's sampling is in contrast not informed by the data.

Dropout regularisation was explained by sparsity. Specifically, Srivastava et al. (2014) described that dropout affects the sparsity of hidden unit activations, resulting in reduced co-adaptation and induced regularisation. We have seen in section 5.1 that sparsity is an important regularisation technique for deep neural networks. We also know that low-rank reconstruction is related to sparsity (Wen et al. 2018, Chen 2018). We believe that the low-rank regularisation induced by NCE is perhaps related to the induced sparsity. As previously stated, all our experiments used variational dropout proposed by (Gal & Ghahramani 2016*a*). As long as dropout can lead to the sparsification of the entire network, its impact on the softmax output layer in Fig. 5.6 was smaller than that of NCE, showing that the reduced co-adaptation and induced sparsity present in NCE can explain the regularisation of the output layer.

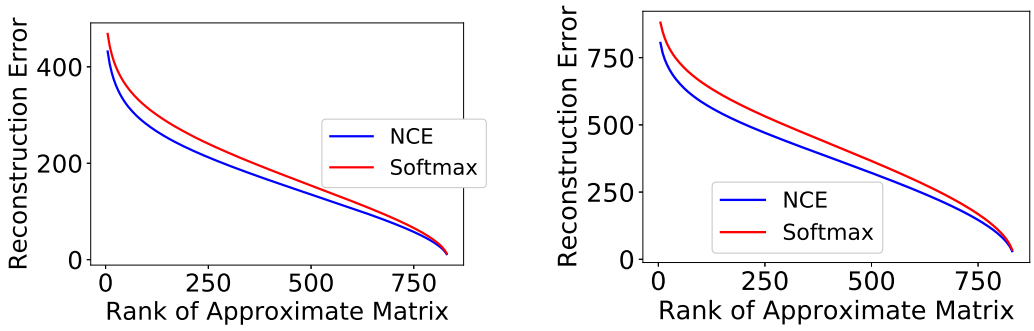
5.4.6 Reduced Parametrisation



(a) PTB dataset (WT)

(b) WikiText-2 dataset (WT)

Figure 5.14: Cumulative variance analysis of the output layer (i.e., matrix θ), on PTB and Wiki-Text2 with weight tying (WT)



(a) PTB dataset (WT)

(b) WikiText-2 dataset (WT)

Figure 5.15: Reconstruction error of the output layer (i.e., matrix θ), on PTB and Wiki-Text2 with weight tying (WT)

Table 5.6: Generalised Variance and Generalisation Performance (WT)

Model	Generalised Variance	PPL
RHN (NCE) on PTB	1.5×10^{-1605}	60.502
RHN (SM) on PTB	2.8×10^{-1521}	61.982
RHN (NCE) on WK2	1.0×10^{-1492}	68.054
RHN (SM) on Wk2	2.0×-1408	71.791

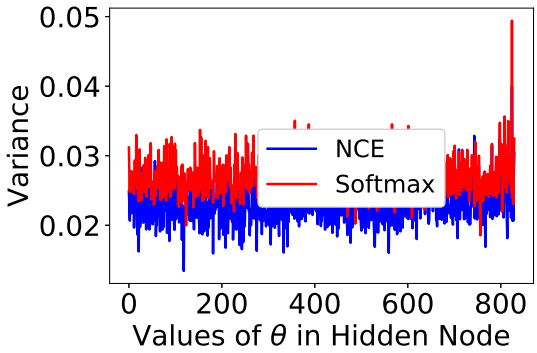


Figure 5.16: Weight Variance (RHN and PTB dataset)

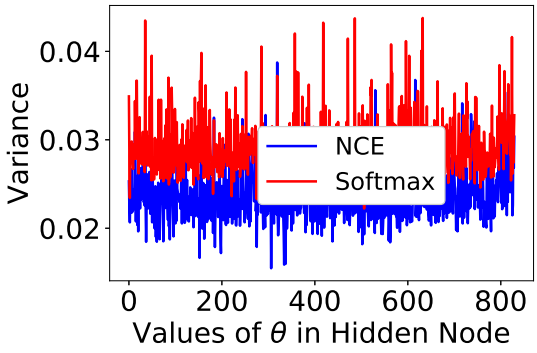
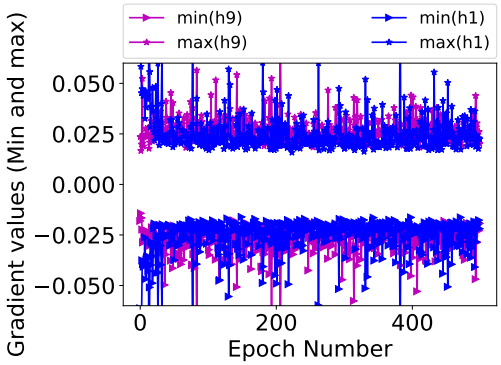


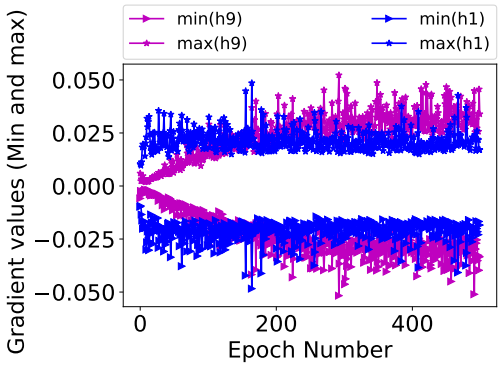
Figure 5.17: Weight Variance (RHN and Wiki-text2 dataset)

Weight tying reduces the number of parameters to be learned, which usually improves generalisation and convergence speed (Press & Wolf 2016, Zhang et al. 2018). For this reason, we explored the cumulative variance of the θ matrices in networks with weight tying (WT). From Fig. 5.14, we can see that with weight tying NCE has faster growth of the cumulative variance. The reconstruction error is also lower in NCE (Fig. 5.15). This means that NCE provides complimentary regularisation of the output layer even when dropout and weight tying are used. The regularisation can be explained by the generalised variance as we see in Tab. 5.6 that NCE based model has lower generalised variance than Softmax when WT was used for both models. This impact can be noticed in terms of perplexity in Tab. 5.4 for WT2, where weight tying improves perplexity from 77.753 to 71.791 in softmax and from 73.080 to 68.054 in NCE (keeping the configuration the same except for the weight tying). These differences show that softmax benefits from dropout much more than NCE, which confirms that NCE delivers better regularisation than softmax. Similar improvement was also observed in Tab. 5.3 for PTB.

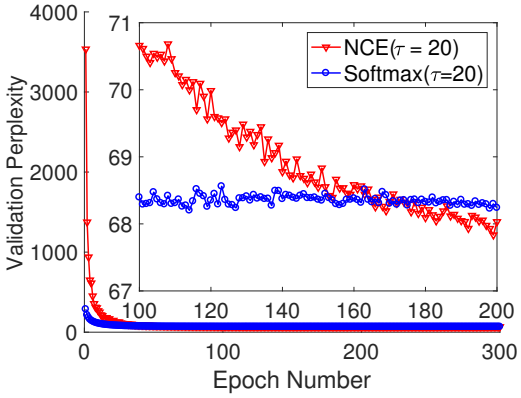
5.4.7 Gradient Analysis



(a) Softmax



(b) NCE



(c) Convergence

Figure 5.18: Gradients with respect to microsteps h_1 and h_9 versus asymptotic convergence of NCE and softmax. Fig. 5.18a and Fig. 5.18b show the gradients with respect to h_1 and h_9 for softmax and NCE respectively. In Fig. 5.18c, NCE's convergence improves when the gradient of h_9 increases in Fig. 5.18b.

In this section, we show that in the RHN models NCE leads to different gradients than softmax, and the differences are correlated with the convergence. Our goal is to show deeper insights as to what happens when NCE outperforms softmax in our experiments. We consider the microsteps h_1 and h_9 (for the reasons explained in section 5.2.2), where h_9 is closer to the output layer than h_1. We run the same model with NCE and softmax, and we compare the magnitude of the gradients of the loss function with respect to the hidden vectors of the microsteps h_1 and h_9. Figures 5.18a and 5.18b show the maximum and minimum values of those gradients across the epochs. We can see that in the initial epochs of NCE, the gradients of the loss function with respect to the higher microstep, h_9, have smaller values than h_1, but increase as learning progresses. The gradient values of the loss function with respect to h_1 are the same across all the epochs. This difference between microsteps h_1 and h_9 is, however, not visible in softmax.

The observations that we made above gain relevance to our study when we include Fig. 5.18c in our analysis. Specifically, when in Fig. 5.18b the gradient becomes larger with respect to h_9 around the epochs number 150–200, the validation perplexity of NCE becomes better than the validation perplexity of softmax in Fig. 5.18c. The graphs show a clear relationship between the magnitude of gradients and the learning convergence. Using this evidence and noting that h_9 is closer to the output layer than h_1, one could conjecture that NCE is more robust against the vanishing gradient problem than softmax since it allows for higher gradients when compared against softmax in the same conditions. The rationale is also the fact that the increase in the solution quality begins at the time when the values of h_9 gradients start increasing. These results show that the NCE's regularisation effect can go beyond the output layer.

5.5 Conclusion

In this chapter, we showed that NCE improves language modelling using RHNs, which are a particular generalisation of LSTM. We analysed the positive impact of NCE and observed that NCE can be seen as a regulariser because it learns models of approximately lower rank than softmax. Both qualitative and quantitative justification was provided, and it was also demonstrated that NCE improves results even when weight tying is used. For understanding the regularisation, generalised variance was introduced, and the technique gave us learning insights about trained models. NCE's impact on gradient propagation was shown to be correlated with its

better generalisation, which is another evidence of its effective regularisation.

Our results can raise interesting theoretical questions for future work. For example, dropout was recently shown to cause a sharp reduction in Rademacher complexity (Gao & Zhou 2016). Since NCE is a randomised method as well, we assume the NCE may have a similar effect on Rademacher complexity, which could be an interesting question to answer for theoretical machine learning.

6

CONCLUSION AND FUTURE WORKS

We have studied sequence modelling, particularly, language modelling to improve performance in terms of accuracy, speed, and model understanding. We have given the emphasis on the standard baseline models and showed that there are opportunities for improvement of the baseline models. Moreover, our investigations have emphasised the output layer and its impact on the representation layer. We hope that our results will motivate further research on improving the output layers. Although it was known that optimising the output layer is important, there was no extensive previous research that would show the impact of the output layer on the quality of the entire network.

We have also shown the links between neural models and weighted finite-state automata (WFA) through language modelling. This has improved the understanding of the models and data properties, and also opened up a new avenue for future research in that direction — specifically, we hope that more theoretical research will be conducted to improve the neural network understanding through comparisons with WFAs.

To summarise, we believe that the results and analyses of the thesis will enhance the research emphasis on standard baseline neural models before designing new architectures for improving the modelling performance. We also believe that the results of this research will encourage researchers to optimise the output layer and design methods to improve the understanding of a neural model. The main contributions of the thesis are summarised next:

1. We have contributed with extensive experiments to investigate models' capacity for capturing the long term dependency and found that in a datasets with long-distance context dependency (e.g. NLP, Biology), deep neural language models perform superiorly compared to other state-of-the-art models. Comparison of advanced techniques shows that RNN language models reach state-of-the-art performance on several benchmark datasets, mainly on the well-known benchmark corpus PennTreebank and PAutomaC. This contribution is based on the

results and analyses of chapters 3, 4, and 5.

2. We have found that spectral algorithms for WFA cannot capture long term dependency due to the computation intractability with large substring statistics, but they can be used as an empirical probing tool to understand deep neural models and dataset characteristics. This insight is novel. This contribution is based on the results and analyses of chapter 3.

3. In the literature, RNNs are stacked to increase the depth to get better accuracy for a language modelling task. For the sequence modelling task, it is not clear what benefit the depth actually brings. We have shown that layers in neural models are efficient in capturing the hidden state representation and less efficient in capturing long-term dependency. This contribution is based on the results and analysis of chapter 3. Furthermore, we have seen in chapters 4 and 5 that increasing the recurrence depth has improved the accuracy for benchmark datasets compared to increasing the depth by stacking layers of RNNs.

4. In the literature, it is shown that NCE does not work efficiently with deep RNNs. In chapters 4 and 5, we have proposed new approaches to integrate NCE with deep RNN language models when trained using stochastic gradient descent and backpropagation through time; and we have shown the effectiveness of the proposed approaches.

5. The NCE, which approximates softmax; works on the output layer of the RNN. The main purpose of the approximation is to improve the training speed. We have shown that the proper use of NCE can enhance generalisation performance (i.e. testing accuracy) while improving the training speed. This conclusion is based on the results and analyses of chapters 4 and 5.

6. The generalisation of deep neural networks is difficult to explain with the classical learning theory. We have explained generalisation in terms of trained models, using the concept of generalised variance. Intuitively, reduced generalised variance moves a model from an overfitted region to a fitted region. This would help us to connect deep learning with classical learning theory in future and could provide new theoretical research directions. This conclusion is based on chapter 5.

7. We have provided an explanation about why deep neural models work better with over-parameterisation. We have found that performance enhancement is related to low-rank regularisation. We have related the explanation with sparsity and a reduced number of effective parameters. This contribution is based on chapter 5.

There is of course rooms for future improvements. Language modelling techniques are usually compared based on the following properties: accuracy, speed, size and implementation complexity. Improving accuracy while reducing training time is challenging. Although we have proposed new approaches that can achieve that, there are rooms for improving the accuracy and reducing the training time that will be explored in future research. Understanding neural models is difficult and crucial for knowledge advancement. The next obvious step is thus getting the neural models to learn to reason. Future research that would aim to improve accuracy and training time includes but is not limited to:

1. Exploring other training algorithms (e.g. Hessian-Free Optimisation) for recurrent neural networks and NCE.
2. Improving noise sampling (e.g. interpolating different noise samples) for NCE with RNNs.
3. Exploring approaches that can be used to learn context-based noise samples with a GPU implementation.
4. Exploring approaches that will make neural models perform efficiently with low resource datasets (i.e. small datasets).
5. Exploring representations based on different levels of textual data (character-level, subword-level, word-level, phrase-level etc.) to allow models to access information more easily from a distant history.

The findings of the thesis are based on the empirical results and experiments conducted on a few benchmark datasets. More theoretical investigations are needed to understand neural networks. Our findings can also be extended to explore the theoretical research. Most of the deep learning research is empirical and there are many contradictory empirical findings that make the researchers puzzled. Theoretical research will help to establish the knowledge contribution. We plan to explore the theoretical aspect of the study in future work. The potential theoretical questions are:

1. Is that possible to find a theory on how much data is needed for a particular neural model?
2. As the neural models work better with over-parameterisation, is there anything special about deep learning architectures or a particular component that re-

duces the VC-dimension significantly? Can we get any theoretical explanation of generalisation in that direction?

3. Gao & Zhou (2016) showed that dropout reduces the Rademacher complexity. Does incorporating NCE into deep neural networks reduce the Rademacher complexity of those networks?

4. How to connect weighted finite-state automata and recurrent neural networks to explain the results of chapter 3 from the theoretical perspective?

Although language modelling has received much attention since 1980s, the main problems (e.g. data sparsity or curse of dimensionality problems Bengio et al. (2003)) are still far from being solved. This is due to the complexity of the task, as often the easiest way to obtain good results is to choose trained models based on as much data as possible. This strategy is however not getting any closer to solving the main problems, rather avoiding them as much as possible. At present, for many tasks involving the neural network training, the amount of available training data (e.g. one billion words benchmark dataset for language modelling) is so huge that further progress by adding more data is not very likely. Although such models are very interesting as they can be used for transfer learning and knowledge extraction, the problem of language modelling is still unsolved.

It is commonly known that most of the published papers report only negligible improvements over baselines. For an example, if we consider speech recognition systems, even the best technique rarely affects the word error rate of the system by more than 10% relatively and that is not a meaningful difference from users perspectives. However, even a small difference can be a stepping stone to a huge difference in the long term — competitions are often won by a slight margin. Also, even if the improvements are small and hardly observable, it is likely that in longer term, the majority of users will tend to prefer the best system.

The next best step for the advancement of neural language models is to understand the trained models. For example, it is essential to understand the patterns that have been learned by a model so that we can improve the model to incorporate the patterns that have not been learned. The first step to understand the patterns that have been learned by a neural model would be extracting automata from trained RNN models (Weiss et al. 2018, Okudono et al. 2019) and analyse the automata. Training deep models with the biggest possible dataset to achieve better accuracy is important for practical applications, but for advancing the knowledge, we need to understand the models and their generalisation capacity.

Appendices

A

Appendix: A

The classical formulation of cross-entropy as a relationship between two distributions and the Eq. A.1 are not immediately obvious. This section will relate the Perplexity's relation to Entropy which is adapted from (Jurafsky & Martin 2009, section 4.6). Given a random variable W ranging over the observations we are predicting (words, letters, parts of speech, the set of which we'll call $\mathcal{W}$) and with a particular probability function, call it $p(w)$, the entropy of the random variable W is:

$$H(W) = -\sum_{w \in \mathcal{W}} p(w) \log_2 p(w)$$

The log can, in principle, be computed in any base. If we use log base 2, the resulting value of entropy will be measured in bits. The above formulation computes the entropy of a single variable. But in NLP we use entropy that involves sequences. For a grammar, for example, we will be computing the entropy of some sequence of words $W = w_0, w_1, w_2, \ldots, w_n$. One way to do this is to have a variable that ranges over sequences of words. For example we can compute the entropy of a random variable that ranges over all finite sequences of words of length L in some language K as follows:

$$H(w_1, \ldots, w_L) = -\sum_{w_1^L \in K} p(W_1^L) \log p(W_1^L)$$

We could define the entropy rate (we could also think of this as the per-word entropy) as the entropy of this sequence divided by the number of words:

$$\frac{1}{L} H(w_1, \ldots, w_L) = -\frac{1}{L} \sum_{w_1^L \in K} p(W_1^L) \log p(W_1^L)$$

But to measure the true entropy of a language, we need to consider sequences of infinite length. If we think of a language as a stochastic process K that

produces a sequence of words, its entropy rate $H(K)$ is defined as

$$H(K) = -\lim_{L\to\infty} \frac{1}{L} H(w_1, \ldots, w_L)$$

$$H(K) = -\lim_{L\to\infty} \frac{1}{L} \sum_{W\in K} p(w_1, \ldots, w_L) \log_2 p(w_1, \ldots, w_L)$$

The Shannon-McMillan-Breiman theorem (Algoet & Cover 1988, Cover & Thomas 2012) states that if the language is regular in certain ways (to be exact, if it is both stationary and ergodic),

$$H(K) = \lim_{L\to\infty} -\frac{1}{L} \log_2 p(w_1, \ldots, w_L)$$

That is, we can take a single sequence that is long enough instead of summing over all possible sequences. The intuition of the Shannon-McMillan-Breiman theorem is that a long-enough sequence of words will contain in it many other shorter sequences and that each of these shorter sequences will reoccur in the longer sequence according to their probabilities.

A stochastic process is said to be stationary if the probabilities it assigns to a sequence are invariant with respect to shifts in the time index. In other words, the probability distribution for words at time t is the same as the probability distribution at time $t+1$. Markov models, and hence N-grams, are stationary. For example, in a bigram, P_i is dependent only on P_{i-1}. So if we shift our time index by x, P_{i+x} is still dependent on P_{i+x-1}. But natural language is not stationary, since the probability of upcoming words can be dependent on events that were arbitrarily distant and time dependent. Thus, our statistical models only give an approximation to the correct distributions and entropies of natural language.

To summarise, by making some incorrect but convenient simplifying assumptions, we can compute the entropy of some stochastic process by taking a very long sample of the output and computing its average log probability.

The cross-entropy is useful when we don't know the actual probability distribution p that generated some data. It allows us to use some q, which is a model of p (i.e. an approximation to p). The cross-entropy of q on p is defined by

$$H(p,q) = \lim_{L\to\infty} -\frac{1}{n} \sum_{W\in K} p(w_1,\ldots,w_L) \log_2 q(w_1,\ldots,w_L)$$

That is, we draw sequences according to the probability distribution p, but sum the log of their probabilities according to q.

Again, following the Shannon-McMillan-Breiman theorem, for a stationary ergodic process:

$$H(p,q) = \lim_{L\to\infty} -\frac{1}{L} \log_2 q(w_1 \ldots w_L)$$

This means that, as for entropy, we can estimate the cross-entropy of a model q on some distribution p by taking a single sequence that is long enough instead of summing over all possible sequences.

What makes the cross-entropy useful is that the cross-entropy $H(p,q)$ is an upper bound on the entropy $H(p)$. For any model q:

$$H(p) \leqslant H(p,q)$$

This means that we can use some simplified model q to help estimate the true entropy of a sequence of symbols drawn according to probability p. The more accurate q is, the closer the cross-entropy $H(p,q)$ will be to the true entropy $H(p)$. Thus, the difference between $H(p,q)$ and $H(p)$ is a measure of how accurate a model is. Between two models q_1 and q_2, the more accurate model will be the one with lower cross-entropy.

We are finally ready to see the relation between perplexity and cross-entropy. Cross-entropy is defined in the limit, as the length of the observed word sequence goes to infinity. We will need an approximation to cross-entropy, relying on a (sufficiently long) sequence of fixed length. This approximation to the cross-entropy of a model $Q = P(w_i|w_{i-N+1}\ldots w_{i-1})$ on a sequence of words W is

$$H(W) = -\frac{1}{L} \log_2 q(w_1 \ldots w_L)$$

$$\begin{aligned} H(w) &= -\frac{1}{L}\log_2 q(w_1,\ldots,w_L) \\ &= -\frac{1}{L}\log_2 \prod_{i=1}^{L} q(w_i|w_1,\ldots,w_{i-1}) \end{aligned} \tag{A.1}$$

From equation 2.2 and A.1, the relation between the perplexity and the cross entropy is as follows:

$$\begin{aligned} 2^{H(w)} &= 2^{-\frac{1}{L}\log_2 q(w_1,\ldots,w_L)} \\ &= 2^{\log_2 q(w_1,\ldots,w_L)^{-\frac{1}{L}}} \\ &= q(w_1,\ldots,w_L)^{-\frac{1}{L}} \\ &= \sqrt[L]{q(w_1,\ldots,w_L)^{-1}} \\ &= \sqrt[L]{\frac{1}{q(w_1,\ldots,w_L)}} \\ &= \sqrt[L]{\frac{1}{\prod_{i=1}^{L} q(w_i|w_1,\ldots,w_{i-1})}} \\ &= PPL(w) \end{aligned} \tag{A.2}$$

$$\boxed{PPL(w) = 2^{H(w)}}$$

Bibliography

Adolphs, L. (2018), Non convex-concave saddle point optimization, Master's thesis, ETH Zurich. 28

Advani, M. S. & Saxe, A. M. (2017), 'High-dimensional dynamics of generalization error in neural networks', *CoRR* **abs/1710.03667**. 40

Algoet, P. H. & Cover, T. M. (1988), 'A sandwich proof of the Shannon-McMillan-Breiman theorem', *The annals of probability* pp. 899–909. 140

Allison, B., Guthrie, D. & Guthrie, L. (2006), Another look at the data sparsity problem, *in* 'International Conference on Text, Speech and Dialogue', Springer, pp. 327–334. 4

Andreas, J. & Klein, D. (2015*a*), When and why are log-linear models self-normalizing?, *in* 'Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies', pp. 244–249. 34

Andreas, J. & Klein, D. (2015*b*), When and why are log-linear models self-normalizing?, *in* 'Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies', pp. 244–249. 35

Angluin, D. (1978), 'On the complexity of minimum inference of regular sets', *Information and control* **39**(3), 337–350. 11

Angluin, D. (1988), 'Queries and concept learning', *Machine learning* **2**(4), 319–342. 11

Arora, S., Ge, R., Neyshabur, B. & Zhang, Y. (2018*a*), Stronger generalization bounds for deep nets via a compression approach, *in* J. Dy & A. Krause, eds, 'Proceedings of the 35th International Conference on Machine Learning', Vol. 80 of *Proceedings of Machine Learning Research*, PMLR, Stockholmsmässan, Stockholm Sweden, pp. 254–263. 40, 98, 99

Arora, S., Ge, R., Neyshabur, B. & Zhang, Y. (2018*b*), 'Stronger generalization bounds for deep nets via a compression approach', *arXiv preprint arXiv:1802.05296* . 42

Arpit, D., Jastrzebski, S., Ballas, N., Krueger, D., Bengio, E., Kanwal, M. S., Maharaj, T., Fischer, A., Courville, A., Bengio, Y. et al. (2017), A closer look at memorization in deep networks, *in* 'Proceedings of the 34th International Conference on Machine Learning-Volume 70', JMLR. org, pp. 233–242. 40

Bahl, L. R., Jelinek, F. & Mercer, R. L. (1983), 'A maximum likelihood approach to continuous speech recognition', *IEEE transactions on pattern analysis and machine intelligence* (2), 179–190. 10

Bai, Z., Fahey, G. & Golub, G. (1996), 'Some large-scale matrix computation problems', *Journal of Computational and Applied Mathematics* **74**(1-2), 71–89. 116

Bailly, R. (2011), 'Quadratic weighted automata: Spectral algorithm and likelihood maximization', *Journal of Machine Learning Research* **20**, 147–162. 54

Balle, B., Carreras, X., Luque, F. M. & Quattoni, A. (2014), 'Spectral learning of weighted automata', *Machine learning* **96**(1-2), 33–63. 11, 17, 18, 19, 20, 50, 53

Balle, B., Eyraud, R., Luque, F. M., Quattoni, A. & Verwer, S. (2017), Results of the sequence prediction challenge (spice): a competition on learning the next symbol in a sequence, *in* 'International Conference on Grammatical Inference', pp. 132–136. 43, 47, 48, 50

Balle, B. & Mohri, M. (2012), Spectral learning of general weighted automata via constrained matrix completion, *in* 'Advances in neural information processing systems', pp. 2159–2167. 11

Balle, B. & Mohri, M. (2015), Learning weighted automata, *in* A. Maletti, ed., 'Algebraic Informatics', Springer International Publishing, Cham, pp. 1–21. 11

Balle, B., Quattoni, A. & Carreras, X. (2012), Local loss optimization in operator models: A new insight into spectral learning., *in* 'ICML', icml.cc / Omnipress. 20, 54

Baltescu, P. & Blunsom, P. (2015), Pragmatic neural language modelling in machine translation., *in* R. Mihalcea, J. Y. Chai & A. Sarkar, eds, 'HLT-NAACL', The Association for Computational Linguistics, pp. 820–829. 74, 75

Bartlett, P. L., Foster, D. J. & Telgarsky, M. J. (2017), Spectrally-normalized margin bounds for neural networks, *in* 'Advances in Neural Information Processing Systems', pp. 6240–6249. 42

Baum, E. B. & Haussler, D. (1989), What size net gives valid generalization?, *in* 'Advances in neural information processing systems', pp. 81–90. 39

Bengio, Y. (2012), *Practical Recommendations for Gradient-Based Training of Deep Architectures*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 437–478. 41, 76, 81, 82, 98, 109, 121

Bengio, Y., Boulanger-Lewandowski, N. & Pascanu, R. (2012), 'Advances in optimizing recurrent networks', *CoRR* **abs/1212.0901**.
URL: *http://arxiv.org/abs/1212.0901* 98, 102

Bengio, Y., Courville, A. & Vincent, P. (2013), 'Representation learning: A review and new perspectives', *IEEE transactions on pattern analysis and machine intelligence* **35**(8), 1798–1828. 3

Bengio, Y., Ducharme, R., Vincent, P. & Jauvin, C. (2003), 'A neural probabilistic language model', *Journal of machine learning research* **3**(Feb), 1137–1155. 11, 12, 73, 137

Bengio, Y. & Frasconi, P. (1995*a*), 'Diffusion of context and credit information in markovian models', *Journal of Artificial Intelligence Research* **3**, 249–270. 71

Bengio, Y. & Frasconi, P. (1995*b*), Diffusion of credit in markovian models, *in* G. Tesauro, D. S. Touretzky & T. K. Leen, eds, 'Advances in Neural Information Processing Systems 7', MIT Press, pp. 553–560. 68, 71

Bengio, Y. & Sénécal, J.-S. (2003), Quick training of probabilistic neural nets by importance sampling, *in* 'Proceedings of the conference on Artificial Intelligence and Statistics (AISTATS)'. 34

Bengio, Y. & Senécal, J.-S. (2008), 'Adaptive importance sampling to accelerate training of a neural probabilistic language model', *IEEE Transactions on Neural Networks* **19**(4), 713–722. 34, 35, 86

Bengio, Y., Simard, P., Frasconi, P. et al. (1994), 'Learning long-term dependencies with gradient descent is difficult', *IEEE transactions on neural networks* **5**(2), 157–166. 31, 85

Bengio, Y. et al. (2009), 'Learning deep architectures for ai', *Foundations and trends in Machine Learning* **2**(1), 1–127. 65, 66, 71

Bergstra, J. & Bengio, Y. (2012), 'Random search for hyper-parameter optimization', *Journal of Machine Learning Research* **13**(Feb), 281–305. 56

Bergstra, J. S., Bardenet, R., Bengio, Y. & Kégl, B. (2011), Algorithms for hyper-parameter optimization, *in* 'Advances in neural information processing systems', pp. 2546–2554. 56

Bishop, C. M. (2006), *Pattern recognition and machine learning*, springer. 112

Blumer, A., Ehrenfeucht, A., Haussler, D. & Warmuth, M. K. (1987), 'Occam's razor', *Information processing letters* **24**(6), 377–380. 41

Boning, D. S., Elfadel, I. A. M. & Li, X. (2019), *A Preliminary Taxonomy for Machine Learning in VLSI CAD*, Springer International Publishing, Cham, pp. 1–16. ix, 23

Boots, B., Gretton, A. & Gordon, G. J. (2013), Hilbert space embeddings of predictive state representations, *in* 'Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence', UAI'13, AUAI Press, Arlington, Virginia, United States, pp. 92–101.
URL: *http://dl.acm.org/citation.cfm?id=3023638.3023648* 11

Botev, A., Zheng, B. & Barber, D. (2017), Complementary sum sampling for likelihood approximation in large scale classification, *in* 'Proc. of AISTATS', pp. 1030–1038. 101

Bottou, L. (2010), Large-scale machine learning with stochastic gradient descent, *in* 'Proceedings of COMPSTAT'2010', Springer, pp. 177–186. 94

Bousquet, O. & Bottou, L. (2008), The tradeoffs of large scale learning, *in* 'Advances in neural information processing systems', pp. 161–168. 40, 81, 92

Brants, T., Popat, A. C., Xu, P., Och, F. J. & Dean, J. (2007), Large language models in machine translation, *in* 'Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)', pp. 858–867. 2

Britz, D., Goldie, A., Luong, M.-T. & Le, Q. (2017), Massive exploration of neural machine translation architectures, *in* 'Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing', Association for Computational Linguistics, pp. 1442–1451.
URL: *http://aclweb.org/anthology/D17-1151* 31

Brown, A. D. & Hinton, G. E. (2001), Products of hidden markov models., *in* 'AISTATS'. 70

Brown, P. F., Desouza, P. V., Mercer, R. L., Pietra, V. J. D. & Lai, J. C. (1992), 'Class-based n-gram models of natural language', *Computational linguistics* **18**(4), 467–479. 10

Broyden, C. G. (1970), 'The convergence of a class of double-rank minimization algorithms 1. general considerations', *IMA Journal of Applied Mathematics* **6**(1), 76–90. 81

Brutzkus, A., Globerson, A., Malach, E. & Shalev-Shwartz, S. (2017), 'Sgd learns over-parameterized networks that provably generalize on linearly separable data', *arXiv preprint arXiv:1710.10174* . 39

Buchsbaum, A., Giancarlo, R. & Westbrook, J. R. (1998), Shrinking language models by robust approximation, *in* 'Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181)', Vol. 2, IEEE, pp. 685–688. 66

Carrasco, R. C. & Oncina, J. (1994), Learning stochastic regular grammars by means of a state merging method, *in* 'International Colloquium on Grammatical Inference', Springer, pp. 139–152. 48

Chen, S. F. & Goodman, J. (1996), An empirical study of smoothing techniques for language modeling, *in* 'Proceedings of the 34th Annual Meeting on Association for Computational Linguistics', ACL '96, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 310–318.
URL: *http://dx.doi.org/10.3115/981863.981904* 16, 52

Chen, W. (2018), 'Simultaneously sparse and low-rank matrix reconstruction via nonconvex and nonseparable regularization', *IEEE Transactions on Signal Processing* **66**(20), 5313–5323. 128

Chen, W., Grangier, D. & Auli, M. (2015), 'Strategies for training large vocabulary neural language models', *CoRR* **abs/1512.04906**.
URL: *http://arxiv.org/abs/1512.04906* 13, 34, 35, 44, 75, 76, 101

Cheng, W.-C., Kok, S., Pham, H. V., Chieu, H. L. & Chai, K. M. A. (2014), Language modeling with sum-product networks, *in* 'Fifteenth Annual Conference of the International Speech Communication Association'. 90

Chierichetti, F., Kumar, R. & Pang, B. (2017), On the power laws of language: Word frequency distributions, *in* 'Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval', ACM, pp. 385–394. 86

Cho, K., van Merrienboer, B., Bahdanau, D. & Bengio, Y. (2014), On the properties of neural machine translation: Encoder–decoder approaches, *in* 'Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation',

Association for Computational Linguistics, Doha, Qatar, pp. 103–111.
URL: *https://www.aclweb.org/anthology/W14-4012* 29

Church, K. W. (1989), A stochastic parts program and noun phrase parser for unrestricted text, *in* 'International Conference on Acoustics, Speech, and Signal Processing,', IEEE, pp. 695–698. 10

Ciampi, A. & Lechevallier, Y. (2007), *Statistical Models and Artificial Neural Networks: Supervised Classification and Prediction Via Soft Trees*, Birkhäuser Boston, Boston, MA, pp. 239–261. 9

Cortes, C., Haffner, P. & Mohri, M. (2004), 'Rational kernels: Theory and algorithms', *Journal of Machine Learning Research* **5**(Aug), 1035–1062. 11

Cover, T. M. & Thomas, J. A. (2012), *Elements of information theory*, John Wiley & Sons. 140

Cybenko, G. (1989), 'Approximation by superpositions of a sigmoidal function', *Mathematics of Control, Signals, and Systems (MCSS)* **2**(4), 303–314. 31

Dai, A. M. & Le, Q. V. (2015), Semi-supervised sequence learning, *in* 'Advances in neural information processing systems', pp. 3079–3087. 1

Darken, C. & Moody, J. E. (1991), Note on learning rate schedules for stochastic optimization, *in* 'Advances in neural information processing systems', pp. 832–838. 76, 82

De Vine, L., Zuccon, G., Koopman, B., Sitbon, L. & Bruza, P. (2014), Medical semantic similarity with a neural language model, *in* 'Proceedings of the 23rd ACM international conference on conference on information and knowledge management', ACM, pp. 1819–1822. 2

Delalleau, O. & Bengio, Y. (2011), Shallow vs. deep sum-product networks, *in* 'Advances in Neural Information Processing Systems', pp. 666–674. 65

Denil, M., Shakibi, B., Dinh, L., Ranzato, M. & de Freitas, N. (2013), 'Predicting parameters in deep learning', *CoRR* **abs/1306.0543**. 109

Devlin, J., Zbib, R., Huang, Z., Lamar, T., Schwartz, R. & Makhoul, J. (2014), Fast and robust neural network joint models for statistical machine translation, *in* 'Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)', Vol. 1, pp. 1370–1380. 35

Dong, L., Yang, N., Wang, W., Wei, F., Liu, X., Wang, Y., Gao, J., Zhou, M. & Hon, H. (2019), 'Unified language model pre-training for natural language understanding and generation', *CoRR* **abs/1905.03197**. 3

Droste, M. & Kuich, W. e. (2009), *Handbook of weighted automata*, EATCS Monographs on Theoretical Computer Science. 11

Duchi, J., Hazan, E. & Singer, Y. (2011), 'Adaptive subgradient methods for online learning and stochastic optimization', *Journal of Machine Learning Research* **12**(Jul), 2121–2159. 40, 81

Duda, R. O., Hart, P. E. & Stork, D. G. (2012), *Pattern classification*, John Wiley & Sons. 40

Dupont, P., Denis, F. & Esposito, Y. (2005), 'Links between probabilistic automata and hidden markov models: probability distributions, learning models and induction algorithms', *Pattern recognition* **38**(9), 1349–1371. 11

Dziugaite, G. K. & Roy, D. M. (2017), 'Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data', *arXiv preprint arXiv:1703.11008* . 42

Eckart, C. & Young, G. (1936), 'The approximation of one matrix by another of lower rank', *Psychometrika* **1**(3), 211–218. 124

Edwards, P. N. (2016), 'Michael D. Gordin. Scientific Babel: How Science Was Done before and after Global English.', *The American Historical Review* **121**(5), 1636–1637. **URL:** *https://doi.org/10.1093/ahr/121.5.1636* 2

Eilenberg, S. (1974), *Automata, Languages, and Machines*, Academic Press, Inc., Orlando, FL, USA. 11

El Hihi, S. & Bengio, Y. (1996), Hierarchical recurrent neural networks for long-term dependencies, *in* 'Advances in neural information processing systems', pp. 493–499. 65, 66

Elman, J. L. (1990), 'Finding structure in time', *COGNITIVE SCIENCE* **14**(2), 179–211. 12, 66

Engl, H. W., Hanke, M. & Neubauer, A. (1996), *Regularization of inverse problems*, Vol. 375, Springer Science & Business Media. 38

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P. & Bengio, S. (2010), 'Why does unsupervised pre-training help deep learning?', *Journal of Machine Learning Research* **11**(Feb), 625–660. 1

Filippova, K., Alfonseca, E., Colmenares, C. A., Kaiser, L. & Vinyals, O. (2015), Sentence compression by deletion with lstms, *in* 'Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing', pp. 360–368. 1

Finn, R. D., Coggill, P., Eberhardt, R. Y., Eddy, S. R., Mistry, J., Mitchell, A. L., Potter, S. C., Punta, M., Qureshi, M., Sangrador-Vegas, A. et al. (2015), 'The pfam protein families database: towards a more sustainable future', *Nucleic acids research* **44**(D1), D279–D285. 50

Finnoff, W., Hergert, F. & Zimmermann, H. G. (1993), 'Improving model selection by nonconvergent methods', *Neural Networks* **6**(6), 771–783. 38

Fischer, A. & Bauckhage, C. (2018), 'Language modeling with recurrent neural networks'. 2

Fletcher, R. (1970), 'A new approach to variable metric algorithms', *The Computer Journal* **13**(3), 317–322. 81

Gal, Y. (2016), Uncertainty in Deep Learning, PhD thesis, University of Cambridge. 96

Gal, Y. & Ghahramani, Z. (2016*a*), Dropout as a Bayesian approximation: Representing model uncertainty in deep learning, *in* 'Proc. of ICML', pp. 1050–1059. 29, 128

Gal, Y. & Ghahramani, Z. (2016*b*), A theoretically grounded application of dropout in recurrent neural networks, *in* 'Proc. of NIPS', pp. 1019–1027. 29, 32, 90, 107, 114

Gal, Y., Hron, J. & Kendall, A. (2017), Concrete dropout, *in* 'Advances in Neural Information Processing Systems', pp. 3581–3590. 42

Gambhir, M. & Gupta, V. (2017), 'Recent automatic text summarization techniques: a survey', *Artificial Intelligence Review* **47**(1), 1–66. 1

Gao, W. & Zhou, Z.-H. (2016), 'Dropout rademacher complexity of deep neural networks', *Science China Information Sciences* **59**(7), 072104. 133, 137

Garg, P. K. & Mohanty, D. (2013), 'Mean (standard deviation) or mean (standard error of mean): time to ponder', *World Journal of Surgery* **37**(4), 932–932. 59

Gavish, M. & Donoho, D. L. (2017), 'Optimal shrinkage of singular values', *IEEE Transactions on Information Theory* **63**(4), 2137–2152. 28

Geman, S., Bienenstock, E. & Doursat, R. (1992), 'Neural networks and the bias/variance dilemma', *Neural computation* **4**(1), 1–58. 39, 112

Gers, F. (2001), Long short-term memory in recurrent neural networks, PhD thesis, Universität Hannover. 76, 77, 89

Gers, F. A., Schraudolph, N. N. & Schmidhuber, J. (2002), 'Learning precise timing with lstm recurrent networks', *Journal of machine learning research* **3**(Aug), 115–143. 29

Giles, C. L., Miller, C. B., Chen, D., Chen, H. H., Sun, G. Z. & Lee, Y. C. (1992), 'Learning and extracting finite state automata with second-order recurrent neural networks', *Neural Comput.* **4**(3), 393–405. 69

Gillis, N. & Glineur, F. (2011), 'Low-rank matrix approximation with weights or missing data is np-hard', *SIAM Journal on Matrix Analysis and Applications* **32**(4), 1149–1165. 20

Girosi, F., Jones, M. & Poggio, T. (1995), 'Regularization theory and neural networks architectures', *Neural computation* **7**(2), 219–269. 36, 38

Glorot, X. & Bengio, Y. (2010), Understanding the difficulty of training deep feedforward neural networks., *in* 'Proc. of AISTATS', Vol. 9, pp. 249–256. 84, 85, 88, 107

Godin, F., Dambre, J. & Neve, W. D. (2017), 'Improving language modeling using densely connected recurrent neural networks', *CoRR* **abs/1707.06130**. 100

Gold, E. M. (1978), 'Complexity of automaton identification from given data', *Information and control* **37**(3), 302–320. 11

Goldberg, Y. (2017), 'Neural network methods for natural language processing', *Synthesis Lectures on Human Language Technologies* **10**(1), 1–309. 25

Goldberger, J. & Melamud, O. (2018), 'Self-normalization properties of language modeling', *CoRR* **abs/1806.00913**. 74

Goldfarb, D. (1970), 'A family of variable-metric methods derived by variational means', *Mathematics of computation* **24**(109), 23–26. 81

Golkar, S., Kagan, M. & Cho, K. (2019), 'Continual learning via neural pruning', *CoRR* **abs/1903.04476**.
URL: *http://arxiv.org/abs/1903.04476* 109

Golowich, N., Rakhlin, A. & Shamir, O. (2017), 'Size-independent sample complexity of neural networks', *arXiv preprint arXiv:1712.06541* . 42

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep Learning*, The MIT Press. x, 27, 36, 37, 76, 80, 91, 99, 109, 112, 113, 120

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A. & Bengio, Y. (2013), Maxout networks, *in* 'Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28', ICML'13, JMLR.org, pp. III–1319–III–1327. 65

Goodman, J. T. (2001), 'A bit of progress in language modeling', *Computer Speech & Language* **15**(4), 403–434. 13, 16

Grave, E., Joulin, A., Cissé, M., Jégou, H. et al. (2017), Efficient softmax approximation for gpus, *in* 'Proceedings of the 34th International Conference on Machine Learning-Volume 70', JMLR. org, pp. 1302–1310. 34

Grave, E., Joulin, A. & Usunier, N. (2017), 'Improving neural language models with a continuous cache', *Proc. of ICLR* . 118

Graves, A. (2013), 'Generating sequences with recurrent neural networks', *CoRR* **abs/1308.0850**. 100

Graves, A. (2016), 'Adaptive computation time for recurrent neural networks', *CoRR* **abs/1603.08983**.
URL: *http://arxiv.org/abs/1603.08983* 100

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R. & Schmidhuber, J. (2015), 'LSTM: A search space odyssey', *CoRR* **abs/1503.04069**.
URL: *http://arxiv.org/abs/1503.04069* 31

Guedj, B. (2019), 'A primer on pac-bayesian learning', *arXiv preprint arXiv:1901.05353* . 11

Gulcehre, C., Firat, O., Xu, K., Cho, K. & Bengio, Y. (2017), 'On integrating a language model into neural machine translation', *Computer Speech & Language* **45**, 137–148. 1

Gutmann, M. & Hyvärinen, A. (2010), Noise-contrastive estimation: A new estimation principle for unnormalized statistical models., *in* 'AISTATS', Vol. 1, pp. 297–304. 35, 44, 74, 80, 86, 98

Gutmann, M. U. & Hyvärinen, A. (2012), 'Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics', *Journal of Machine Learning Research* **13**(Feb), 307–361. 34, 35, 44

Hammerschmidt, C., Loos, B. L., Verwer, S. et al. (2016), 'Flexible state-merging for learning (p) dfas in python'. 52

Han, S., Pool, J., Tran, J. & Dally, W. (2015), Learning both weights and connections for efficient neural network, *in* 'Advances in neural information processing systems', pp. 1135–1143. 109

Hardt, M., Recht, B. & Singer, Y. (2016), Train faster, generalize better: Stability of stochastic gradient descent, *in* 'Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48', ICML'16, JMLR.org, pp. 1225–1234. 40

Harvey, N., Liaw, C. & Mehrabian, A. (2017), Nearly-tight vc-dimension bounds for piecewise linear neural networks, *in* 'Conference on Learning Theory', pp. 1064–1068. 39

He, K., Zhang, X., Ren, S. & Sun, J. (2015), Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, *in* 'Proceedings of the IEEE international conference on computer vision', pp. 1026–1034. 84

Hecht-Nielsen, R. (1992), Theory of the backpropagation neural network, *in* 'Neural networks for perception', Elsevier, pp. 65–93. 25

Heeman, P. A. (1999), Pos tags and decision trees for language modeling, *in* '1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora'. 10, 12

Heinz, J. & Rogers, J. (2010), Estimating strictly piecewise distributions, *in* 'Proceedings of the 48th annual meeting of the association for computational linguistics', Association for Computational Linguistics, pp. 886–896. 51

Herculano-Houzel, S. (2009), 'The human brain in numbers: a linearly scaled-up primate brain', *Frontiers in human neuroscience* **3**, 31. 21

Hillar, C. J. & Lim, L.-H. (2013), 'Most tensor problems are np-hard', *J. ACM* **60**(6), 45:1–45:39. 20

Hinton, G., Brown, A. & London, Q. S. (2001), 'Training many small hidden markov models', *Proc. of the Workshop on Innovation in Speech Processing* . 68, 71

Hinton, G. E. (2002), 'Training products of experts by minimizing contrastive divergence', *Neural computation* **14**(8), 1771–1800. 74

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. R. (2012), 'Improving neural networks by preventing co-adaptation of feature detectors', *arXiv preprint arXiv:1207.0580* . 65, 107, 110, 127

Hochreiter, S. (1991), 'Untersuchungen zu dynamischen neuronalen netzen', *Diploma, Technische Universität München* **91**(1). 31

Hochreiter, S. (1998), 'The vanishing gradient problem during learning recurrent neural nets and problem solutions', *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **6**(02), 107–116. 31

Hochreiter, S. & Schmidhuber, J. (1997), 'Long short-term memory', *Neural computation* **9**(8), 1735–1780. 32, 76, 77, 89

Hodosh, M., Young, P. & Hockenmaier, J. (2013), 'Framing image description as a ranking task: Data, models and evaluation metrics', *Journal of Artificial Intelligence Research* **47**, 853–899. 48

Hoerl, A. E. & Kennard, R. W. (1970), 'Ridge regression: Biased estimation for nonorthogonal problems', *Technometrics* **12**(1), 55–67. 38

Hornik, K., Stinchcombe, M. & White, H. (1989), 'Multilayer feedforward networks are universal approximators', *Neural networks* **2**(5), 359–366. 31

Hornik, K., Stinchcombe, M. & White, H. (1990), 'Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks', *Neural networks* **3**(5), 551–560. 25

Howar, F., Isberner, M., Merten, M., Steffen, B., Beyer, D. & Pasareanu, C. S. (2014), 'Rigorous examination of reactive systems - the RERS challenges 2012 and 2013', *STTT* **16**(5), 457–464. 48

Hsu, D., Kakade, S. M. & Zhang, T. (2012), 'A spectral algorithm for learning hidden markov models', *Journal of Computer and System Sciences* **78**(5), 1460–1480. 11, 54

Hu, Z., Nie, F., Tian, L., Wang, R. & Li, X. (2018), Low rank regularization: a review, Vol. abs/1808.04521.
URL: *http://arxiv.org/abs/1808.04521* 41, 101, 106, 110, 121

Hutchins, W. J. (1995), Machine translation: A brief history, *in* 'Concise history of the language sciences', Elsevier, pp. 431–445. 2

Hyvärinen, A. (2005), 'Estimation of non-normalized statistical models by score matching', *Journal of Machine Learning Research* **6**(Apr), 695–709. 74

Inan, H., Khosravi, K. & Socher, R. (2017), 'Tying word vectors and word classifiers: A loss framework for language modeling', *Proc. of ICLR* **abs/1611.01462**.
URL: *http://arxiv.org/abs/1611.01462* 32, 90, 115

Irsoy, O. & Cardie, C. (2014), Opinion mining with deep recurrent neural networks, *in* 'Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)', pp. 720–728. 32

Jean, S., Cho, K., Memisevic, R. & Bengio, Y. (2014*a*), 'On using very large target vocabulary for neural machine translation', *arXiv preprint arXiv:1412.2007* . 34

Jean, S., Cho, K., Memisevic, R. & Bengio, Y. (2014*b*), 'On using very large target vocabulary for neural machine translation', *CoRR* **abs/1412.2007**.
URL: *http://arxiv.org/abs/1412.2007* 35

Jelinek, F. (1997), *Statistical methods for speech recognition*, MIT press. 10

Jim, K., Horne, B. G. & Giles, C. L. (1995), Effects of noise on convergence and generalization in recurrent networks, *in* 'Advances in neural information processing systems', pp. 649–656. 41

Jin, C., Netrapalli, P. & Jordan, M. I. (2017), 'Accelerated gradient descent escapes saddle points faster than gradient descent', *CoRR* **abs/1711.10456**.
URL: *http://arxiv.org/abs/1711.10456* 83

Jolliffe, I. (2002), *Principal component analysis*, John Wiley & Sons. 110

Jordan, M. (1986), Serial order: a parallel distributed processing approach. technical report, june 1985-march 1986, Technical report, California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science. 12

Joulin, A., van der Maaten, L., Jabri, A. & Vasilache, N. (2016), Learning visual features from large weakly supervised data, *in* 'European Conference on Computer Vision', Springer, pp. 67–84. 35

Józefowicz, R., Vinyals, O., Schuster, M., Shazeer, N. & Wu, Y. (2016), 'Exploring the limits of language modeling', *CoRR* **abs/1602.02410**.
URL: *http://arxiv.org/abs/1602.02410* 1, 35, 44, 73, 74, 75, 101

Jozefowicz, R., Zaremba, W. & Sutskever, I. (2015), An empirical exploration of recurrent network architectures, *in* 'International Conference on Machine Learning', pp. 2342–2350. 32, 65, 66

Jurafsky, D. (2000), *Speech & language processing*, Pearson Education India. 13, 14

Jurafsky, D. & Martin, J. H. (2009), *Speech and Language Processing (2nd Edition)*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA. 139

Kakade, S. M., Liang, P., Sharan, V. & Valiant, G. (2016), 'Prediction with a short memory', *CoRR* **abs/1612.02526**. 68

Katz, N. M. (2004), 'Notes on g2, determinants, and equidistribution', *Finite Fields and Their Applications* **10**(2), 221–269. 48

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M. & Tang, P. T. P. (2016), 'On large-batch training for deep learning: Generalization gap and sharp minima', *arXiv preprint arXiv:1609.04836* . 42

Khandelwal, U., He, H., Qi, P. & Jurafsky, D. (2018), Sharp nearby, fuzzy far away: How neural language models use context, *in* 'Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)', Association for Computational Linguistics, pp. 284–294. 36, 71

Kiefer, J., Wolfowitz, J. et al. (1952), 'Stochastic estimation of the maximum of a regression function', *The Annals of Mathematical Statistics* **23**(3), 462–466. 40, 80

Kim, Y., Jernite, Y., Sontag, D. & Rush, A. M. (2016), Character-aware neural language models., *in* 'AAAI', pp. 2741–2749. 5, 90

Kingma, D. P. & Ba, J. (2015), Adam: A method for stochastic optimization, *in* 'Proc. of ICLR'. 40, 81

Kneser, R. & Ney, H. (1995), Improved backing-off for m-gram language modeling, *in* '1995 International Conference on Acoustics, Speech, and Signal Processing', Vol. 1, pp. 181–184 vol.1. 17, 52

Koehn, P. (2010), *Statistical Machine Translation*, 1st edn, Cambridge University Press, New York, NY, USA. 53

Krogh, A. & Hertz, J. A. (1992), A simple weight decay can improve generalization, *in* 'Advances in neural information processing systems', pp. 950–957. 38, 41

Labeau, M. & Allauzen, A. (2018), Learning with noise-contrastive estimation: Easing training by learning to scale, *in* 'Proceedings of the 27th International Conference on Computational Linguistics (COLING)', pp. 3090–3101. 80

Lafferty, J., Sleator, D. & Temperley, D. (1992), *Grammatical trigrams: A probabilistic model of link grammar*, Vol. 56, School of Computer Science, Carnegie Mellon University. 10

Langeberg, P., Balda, E. R., Behboodi, A. & Mathar, R. (2019), 'On the effect of low-rank weights on adversarial robustness of neural networks', *arXiv preprint arXiv:1901.10371* . 108, 110, 121

Larochelle, H., Erhan, D., Courville, A., Bergstra, J. & Bengio, Y. (2007), An empirical evaluation of deep architectures on problems with many factors of variation, *in* 'Proceedings of the 24th international conference on Machine learning', ACM, pp. 473–480. 56

Lawrence, S., Giles, C. L. & Tsoi, A. C. (1998), What size neural network gives optimal generalization? convergence properties of backpropagation, Technical report, Technical report, U. of Maryland. 113

Le Roux, N. & Bengio, Y. (2010), 'Deep belief networks are compact universal approximators', *Neural computation* **22**(8), 2192–2207. 65

Le, X.-H., Ho, H. V., Lee, G. & Jung, S. (2019), 'Application of long short-term memory (LSTM) neural network for flood forecasting', *Water* **11**(7), 1387. ix, 30

Lebedev, V. & Lempitsky, V. (2016), Fast convnets using group-wise brain damage, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 2554–2564. 109

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. et al. (1998), 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE* **86**(11), 2278–2324. 40

LeCun, Y., Haffner, P., Bottou, L. & Bengio, Y. (1999), Object recognition with gradient-based learning, *in* 'Shape, contour and grouping in computer vision', Springer, pp. 319–345. 25

Lei, D., Sun, Z., Xiao, Y. & Wang, W. Y. (2018), 'Implicit regularization of stochastic gradient descent in natural language processing: Observations and implications', *CoRR* **abs/1811.00659**. 42

Lesher, G. W., Moulton, B. J. & Higginbotham, D. J. (1999), 'Effects of ngram order and training text size on word prediction'. 17

Li, H., Kadav, A., Durdanovic, I., Samet, H. & Graf, H. P. (2017), 'Pruning filters for efficient convnets', *In Proc. of ICLR* . 109

Li, M. & Vitányi, P. (2013), *An introduction to Kolmogorov complexity and its applications*, Springer Science & Business Media. 41

Liao, J., Liu, T., Liu, M., Wang, J., Wang, Y. & Sun, H. (2018), 'Multi-context integrated deep neural network model for next location prediction', *IEEE Access* **6**, 21980–21990. 36

Liza, F. F. & Grześ, M. (2016), Estimating the accuracy of spectral learning for hmms, *in* 'Proc. of International Conference on Artificial Intelligence: Methodology, Systems, and Applications', Springer, pp. 46–56. 7, 20

Liza, F. F. & Grześ, M. (2017), A spectral method that worked well in the spice'16 competition, *in* 'Proc. of International Conference on Grammatical Inference', pp. 143–148. 7, 52

Liza, F. F. & Grześ, M. (2018), Improving language modelling with noise-contrastive estimation, *in* 'Proc. of AAAI'. 7, 106

Liza, F. F. & Grzes, M. (2019), Relating RNN layers with the spectral WFA ranks in sequence modelling, *in* 'Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges', Association for Computational Linguistics, Florence, pp. 24–33.
URL: *https://www.aclweb.org/anthology/W19-3903* 7

Lothaire, M. (2005), *Applied Combinatorics on Words*, Encyclopedia of Mathematics and its Applications, Cambridge University Press. 11

Luong, T., Kayser, M. & Manning, C. D. (2015), Deep neural language models for machine translation, *in* 'Proceedings of the Nineteenth Conference on Computational Natural Language Learning', pp. 305–309. 1, 2

Manning, C. D., Manning, C. D. & Schütze, H. (1999), *Foundations of statistical natural language processing*, MIT press. 10

Manning, C. D., Raghavan, P. & Schütze, H. (2008), *Introduction to Information Retrieval*, Cambridge University Press, New York, NY, USA. 110

Marcus, M. P., Marcinkiewicz, M. A. & Santorini, B. (1993*a*), 'Building a large annotated corpus of english: The penn treebank', *Comput. Linguist.* **19**(2), 313–330. 48

Marcus, M. P., Marcinkiewicz, M. A. & Santorini, B. (1993*b*), 'Building a large annotated corpus of english: The penn treebank', *Computational linguistics* **19**(2), 313–330. 70, 75

Martin, C. H. & Mahoney, M. W. (2018), 'Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning', *arXiv preprint arXiv:1810.01075* . 99

Martin, C. H. & Mahoney, M. W. (2019), 'Traditional and heavy-tailed self regularization in neural network models', *arXiv preprint arXiv:1901.08276* . 40, 111

McNaughton, R. & Papert, S. A. (1971), *Counter-Free Automata (M.I.T. Research Monograph No. 65)*, The MIT Press. 51

Melis, G., Dyer, C. & Blunsom, P. (2017), 'On the state of the art of evaluation in neural language models', *CoRR* **abs/1707.05589**.
URL: *http://arxiv.org/abs/1707.05589* 100

Merity, S., Keskar, N. S. & Socher, R. (2017), 'Regularizing and optimizing LSTM language models', *CoRR* **abs/1708.02182**.
URL: *http://arxiv.org/abs/1708.02182* 100, 117, 118

Merity, S., Xiong, C., Bradbury, J. & Socher, R. (2016), 'Pointer sentinel mixture models', *CoRR* **abs/1609.07843**.
URL: *http://arxiv.org/abs/1609.07843* 90, 118

Mikolov, T. (2012), Statistical Language Models Based on Neural Networks., PhD thesis, Faculty of Information Technology. 13

Mikolov, T., Deoras, A., Povey, D., Burget, L. & Černockỳ, J. (2011), Strategies for training large scale neural network language models, *in* '2011 IEEE Workshop on Automatic Speech Recognition & Understanding', IEEE, pp. 196–201. 13, 33

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J. & Khudanpur, S. (2010), Recurrent neural network based language model., *in* T. Kobayashi, K. Hirose & S. Nakamura, eds, 'INTERSPEECH', ISCA, pp. 1045–1048. 1, 12, 32, 33, 73

Mikolov, T., Kombrink, S., Burget, L., Cernocký, J. & Khudanpur, S. (2011), Extensions of recurrent neural network language model., *in* 'ICASSP', IEEE, pp. 5528–5531. 33

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. & Dean, J. (2013), Distributed representations of words and phrases and their compositionality, *in* C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani & K. Q. Weinberger, eds, 'Advances in Neural Information Processing Systems 26', Curran Associates, Inc., pp. 3111–3119. 1, 12, 87

Mikolov, T., Yih, W.-t. & Zweig, G. (2013), Linguistic regularities in continuous space word representations., *in* 'HLT-NAACL', pp. 746–751. 1, 12

Mikolov, T. & Zweig, G. (2012), Context dependent recurrent neural network language model., *in* 'SLT', pp. 234–239. 13, 90

Mnih, A. & Teh, Y. W. (2012), A fast and simple algorithm for training neural probabilistic language models, *in* 'Proc. of ICML'. 35, 74, 78, 80

Mohri, M. (1997*a*), 'Finite-state transducers in language and speech processing', *Computational linguistics* **23**(2), 269–311. 11

Mohri, M. (1997*b*), 'On the use of sequential transducers in natural language processing', *Finite-State Language Processing* p. 355. 11

Mohri, M. (2004), Weighted finite-state transducer algorithms. an overview, *in* 'Formal Languages and Applications', Springer, pp. 551–563. 11, 17

Mohri, M. & Pereira, F. C. (1998), Dynamic compilation of weighted context-free grammars, *in* 'Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 2', Association for Computational Linguistics, pp. 891–897. 11

Moody, J. (1994), Prediction risk and architecture selection for neural networks, *in* 'From statistics to neural networks', Springer, pp. 147–165. 36, 38

Morgan, N. & Bourlard, H. (1990), Generalization and parameter estimation in feedforward nets: Some experiments, *in* 'Advances in neural information processing systems', pp. 630–637. 38

Morin, F. & Bengio, Y. (2005), Hierarchical probabilistic neural network language model., *in* 'Aistats', Vol. 5, Citeseer, pp. 246–252. 34

Neal, B., Mittal, S., Baratin, A., Tantia, V., Scicluna, M., Lacoste-Julien, S. & Mitliagkas, I. (2018), 'A modern take on the bias-variance tradeoff in neural networks', *CoRR* **abs/1810.08591**. ix, 37

Neyshabur, B. (2017), Implicit Regularization in Deep Learning, PhD thesis. 40, 99, 109, 120

Neyshabur, B., Bhojanapalli, S., McAllester, D. & Srebro, N. (2017), Exploring generalization in deep learning, *in* 'Advances in Neural Information Processing Systems', pp. 5947–5956. 42

Neyshabur, B., Bhojanapalli, S. & Srebro, N. (2017), 'A pac-bayesian approach to spectrally-normalized margin bounds for neural networks', *arXiv preprint arXiv:1707.09564* . 42

Neyshabur, B., Li, Z., Bhojanapalli, S., LeCun, Y. & Srebro, N. (2019), 'Towards understanding the role of over-parametrization in generalization of neural networks', *Proc. of ICLR* . 111

Neyshabur, B., Tomioka, R. & Srebro, N. (2015*a*), 'In search of the real inductive bias: On the role of implicit regularization in deep learning', *Proc of ICLR workshop track* . 39, 113

Neyshabur, B., Tomioka, R. & Srebro, N. (2015*b*), Norm-based capacity control in neural networks, *in* 'Conference on Learning Theory', pp. 1376–1401. 42

Nocedal, J. & Wright, S. J. (2006), *Numerical Optimization*, second edn, Springer, New York, NY, USA. 27

Novak, R., Bahri, Y., Abolafia, D. A., Pennington, J. & Sohl-Dickstein, J. (2018), 'Sensitivity and generalization in neural networks: an empirical study', *International Conference on Learning Representations* . 39

Obozinski, G., Jacob, L. & Vert, J.-P. (2011), 'Group lasso with overlaps: the latent group lasso approach', *arXiv preprint arXiv:1110.0413* . 108

Okudono, T., Waga, M., Sekiyama, T. & Hasuo, I. (2019), 'Weighted automata extraction from recurrent neural networks via regression on state spaces', *arXiv preprint arXiv:1904.02931* . 137

Olshausen, B. A. & Field, D. J. (1996), 'Emergence of simple-cell receptive field properties by learning a sparse code for natural images', *Nature* **381**(6583), 607. 42

Omlin, C. W. & Giles, C. L. (1996), 'Constructing deterministic finite-state automata in recurrent neural networks', *Journal of the ACM (JACM)* **43**(6), 937–972. 69

Oualil, Y., Singh, M., Greenberg, C. & Klakow, D. (2016), Long-short range context neural networks for language modeling, *in* J. Su, X. Carreras & K. Duh, eds, 'Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016', The Association for Computational Linguistics, pp. 1473–1481.
URL: *http://aclweb.org/anthology/D/D16/* 36

Pascanu, R., Gülçehre, Ç., Cho, K. & Bengio, Y. (2014), 'How to construct deep recurrent neural networks', *In the Proc. of ICLR* . 32, 65, 66, 71, 90, 100

Pascanu, R., Mikolov, T. & Bengio, Y. (2013), On the difficulty of training recurrent neural networks, *in* 'Proc. of ICML', Atlanta, Georgia, USA, pp. 1310–1318.
URL: *http://proceedings.mlr.press/v28/pascanu13.html* 31, 77

Pascanu, R., Montufar, G. & Bengio, Y. (2013), 'On the number of response regions of deep feed forward networks with piece-wise linear activations', *arXiv preprint arXiv:1312.6098* . 65, 72

Penagarikano, M. & Bordel, G. (2004), Layered Markov models: a new architectural approach to automatic speech recognition, *in* 'Proceedings of the 2004 14th IEEE Signal Processing Society Workshop Machine Learning for Signal Processing, 2004.', IEEE, pp. 305–314. 68

Pérez, G. V., Louis, A. A. & Camargo, C. Q. (2019), 'Deep learning generalizes because the parameter-function map is biased towards simple functions', *Proc. of ICLR* . 41, 98, 99

Peters, J. & Klakow, D. (1999), Compact maximum entropy language models, *in* 'Proceedings of the IEEE workshop on automatic speech recognition and understanding'. 11, 12

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. & Zettlemoyer, L. (2018), Deep contextualized word representations, *in* 'Proc. of NAACL'. 36

Piantadosi, S. T. (2014), 'Zipf's word frequency law in natural language: A critical review and future directions', *Psychonomic Bulletin & Review* **21**(5), 1112–1130.
URL: *http://dx.doi.org/10.3758/s13423-014-0585-6* 86

Pierce, J. R. & Carroll, J. B. (1966), *Language and Machines: Computers in Translation and Linguistics*, National Academy of Sciences/National Research Council, Washington, DC, USA. 2

Pitt, L. & Warmuth, M. K. (1993), 'The minimum consistent dfa problem cannot be approximated within any polynomial', *Journal of the ACM (JACM)* **40**(1), 95–142. 11

Potamianos, G. & Jelinek, F. (1998), 'A study of n-gram and decision tree letter language modeling methods', *Speech Communication* **24**(3), 171–192. 10

Prabhavalkar, R., Rao, K., Sainath, T. N., Li, B., Johnson, L. & Jaitly, N. (2017), A comparison of sequence-to-sequence models for speech recognition., *in* 'Interspeech', pp. 939–943. 1

Press, O. & Wolf, L. (2016), 'Using the output embedding to improve language models', *In the Proc. of EACL* **2**, 157–163. 32, 90, 115, 117, 130

Quattoni, A., Carreras, X. & Gallé, M. (2017), 'A maximum matching algorithm for basis selection in spectral learning', *CoRR* **abs/1706.02857**.
URL: *http://arxiv.org/abs/1706.02857* 20, 46

Rabusseau, G., Li, T. & Precup, D. (2018), 'Connecting weighted automata and recurrent neural networks through spectral learning', *arXiv preprint arXiv:1807.01406* . 69

Radford, A., Narasimhan, K., Salimans, T. & Sutskever, I. (2018), 'Improving language understanding by generative pre-training', *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/language-unsupervised/language_ understanding_paper. pdf* . 1

Rashkin, H., Choi, E., Jang, J. Y., Volkova, S. & Choi, Y. (2017), Truth of varying shades: Analyzing language in fake news and political fact-checking, *in* 'Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing', pp. 2931–2937. 1

Razborov, A. A. (1992), On small depth threshold circuits, *in* 'Scandinavian Workshop on Algorithm Theory', Springer, pp. 42–52. 9

Reed, R. & MarksII, R. J. (1999), *Neural smithing: supervised learning in feedforward artificial neural networks*, Mit Press. 112

Rei, M. & Yannakoudakis, H. (2017), 'Auxiliary objectives for neural error detection models', *arXiv preprint arXiv:1707.05227* . 1

Rissanen, J. (1978), 'Modeling by shortest data description', *Automatica* **14**(5), 465–471. 41

Robbins, H. & Monro, S. (1951), 'A stochastic approximation method', *The annals of mathematical statistics* pp. 400–407. 40, 80, 82

Rosenblatt, F. (1958), 'The perceptron: a probabilistic model for information storage and organization in the brain.', *Psychological review* **65**(6), 386. 24

Rosenfeld, R. (1994), Adaptive statistical language modeling; a maximum entropy approach, Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE. 11

Rosenfeld, R. (2000), 'Two decades of statistical language modeling: Where do we go from here?', *Proceedings of the IEEE* **88**(8), 1270–1278. 1, 2, 46

Ruder, S. (2016), 'An overview of gradient descent optimization algorithms', *CoRR* **abs/1609.04747**.
URL: *http://arxiv.org/abs/1609.04747* 28

Salomaa, A. & Soittola, M. (1978), *Automata: Theoretic Aspects of Formal Power Series*, Springer-Verlag, Berlin, Heidelberg. 11

Sanyal, A., Kanade, V. & Torr, P. H. (2018), 'Low rank structure of learned representations', *arXiv preprint arXiv:1804.07090* . 108

Sato, I., Chubachi, K. et al. (2017), Evaluation of machine learning methods on spice, *in* 'International Conference on Grammatical Inference', pp. 149–153. 51

Schäfer, A. M. & Zimmermann, H. G. (2006), Recurrent neural networks are universal approximators, *in* 'International Conference on Artificial Neural Networks', Springer, pp. 632–640. 13

Schmidhuber, J. (2008), 'Learning complex, extended sequences using the principle of history compression', *Learning* **4**(2). 65

Schmidhuber, J. (2015), 'Deep learning in neural networks: An overview', *Neural Networks* **61**, 85 – 117.
URL: *http://www.sciencedirect.com/science/article/pii/S0893608014002135* 42

Schwenk, H. & Gauvain, J.-L. (2002), 'Connectionist language modeling for large vocabulary continuous speech recognition', *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing* **1**, I–765–I–768. 4

Sengupta, A. (2004), 'Generalized variance', *Encyclopedia of statistical sciences* . 113

Shanno, D. F. (1970), 'Conditioning of quasi-newton methods for function minimization', *Mathematics of computation* **24**(111), 647–656. 81

Shannon, C. E. (1948), 'A mathematical theory of communication', *Bell system technical journal* **27**(3), 379–423. 13

Shibata, C. & Heinz, J. (2017), Predicting sequential data with lstms augmented with strictly 2-piecewise input vectors, *in* 'International Conference on Grammatical Inference', pp. 137–142. xiii, 51, 52, 57, 58, 59, 60

Siddiqi, S. M., Boots, B. & Gordon, G. J. (2009), 'Reduced-rank hidden markov models', *CoRR* **abs/0910.0902**. 54

Siddiqi, S. M., Gordon, G. J. & Moore, A. W. (2007), Fast state discovery for hmm model selection and learning, *in* 'Artificial Intelligence and Statistics', pp. 492–499. 69

Siegelmann, H. T. & Sontag, E. D. (1991), 'Turing computability with neural nets', *Applied Mathematics Letters* **4**, 77–80. 31, 71

Smith, S. L. & Le, Q. V. (2018), 'A bayesian perspective on generalization and stochastic gradient descent', *Proc. of ICLR* . 40, 41

Soudry, D., Hoffer, E., Nacson, M. S., Gunasekar, S. & Srebro, N. (2018), 'The implicit bias of gradient descent on separable data', *The Journal of Machine Learning Research* **19**(1), 2822–2878. 40

Spithourakis, G. P., Augenstein, I. & Riedel, S. (2016), 'Numerically grounded language models for semantic error correction', *arXiv preprint arXiv:1608.04147* . 1

Spithourakis, G. P., Petersen, S. E. & Riedel, S. (2016), 'Clinical text prediction with numerically grounded conditional language models', *arXiv preprint arXiv:1610.06370* . 2

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014), 'Dropout: a simple way to prevent neural networks from overfitting.', *Journal of Machine Learning Research* **15**(1), 1929–1958. 29, 42, 59, 87, 107, 113, 127, 128

Srivastava, R. K., Greff, K. & Schmidhuber, J. (2015), 'Highway networks', *arXiv preprint arXiv:1505.00387* . 100

Sun, S., Chen, W., Wang, L., Liu, X. & Liu, T.-Y. (2016), On the depth of deep neural networks: A theoretical view, *in* 'Thirtieth AAAI Conference on Artificial Intelligence'. 39

Sutskever, I., Martens, J., Dahl, G. E. & Hinton, G. E. (2013), 'On the importance of initialization and momentum in deep learning.', *ICML (3)* **28**, 1139–1147. 82, 83, 84

Sutskever, I., Vinyals, O. & Le, Q. V. (2014), Sequence to sequence learning with neural networks, *in* 'Advances in neural information processing systems', pp. 3104–3112. 32

Tai, C., Xiao, T., Zhang, Y., Wang, X. et al. (2015), 'Convolutional neural networks with low-rank regularization', *arXiv preprint arXiv:1511.06067* . 41, 101

Taulé, M., Martí, M. A. & Recasens, M. (2008), Ancora: Multilevel annotated corpora for Catalan and Spanish, *in* 'LREC', European Language Resources Association. 48

Tibshirani, R. (1996), 'Regression shrinkage and selection via the lasso', *Journal of the Royal Statistical Society. Series B (Methodological)* pp. 267–288. 41, 42

Tikhomirov, V. M. (1991), *On the Representation of Continuous Functions of Several Variables as Superpositions of Continuous Functions of one Variable and Addition*, Springer Netherlands, Dordrecht, pp. 383–387. 25

Tullo, C. & Hurford, J. (2003), Modelling zipfian distributions in language, *in* 'Proceedings of language evolution and computation workshop/course at ESSLLI', pp. 62–75. 86

Turing, A. M. (1950), 'Computing machinery and intelligence', *Mind* **59**(236), 433–460. **URL:** *http://www.jstor.org/stable/2251299* 3

TYPO CORPUS (2010), http://luululu.com/tweet/. [Online; accessed 05-Feb-2019]. 48

Udell, M. & Townsend, A. (2019), 'Why are big data matrices approximately low rank?', *SIAM Journal on Mathematics of Data Science* **1**(1), 144–160. 121

Vapnik, V., Levin, E. & Cun, Y. L. (1994), 'Measuring the vc-dimension of a learning machine', *Neural computation* **6**(5), 851–876. 40

Vaswani, A., Zhao, Y., Fossum, V. & Chiang, D. (2013), Decoding with large-scale neural language models improves translation., *in* 'EMNLP', ACL, pp. 1387–1392. 35, 74, 80

Verwer, S., Eyraud, R. & De La Higuera, C. (2014), 'Pautomac: a probabilistic automata and hidden markov models learning competition', *Machine learning* **96**(1-2), 129–154. 68

Verwer, S., Eyraud, R. & dela Higuera, C. (2014), 'Pautomac: a probabilistic automata and hidden markov models learning competition', *Machine Learning* **96**(1), 129–154. 48

Vincent, P., Larochelle, H., Bengio, Y. & Manzagol, P.-A. (2008), Extracting and composing robust features with denoising autoencoders, *in* 'Proceedings of the 25th international conference on Machine learning', ACM, pp. 1096–1103. 107

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. & Manzagol, P.-A. (2010), 'Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion', *Journal of machine learning research* **11**(Dec), 3371–3408. 107

Wahba, G., Lin, X., Gao, F., Xiang, D., Klein, R. & Klein, B. (1999), The bias-variance tradeoff and the randomized gacv, *in* 'Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II', MIT Press, Cambridge, MA, USA, pp. 620–626.
URL: *http://dl.acm.org/citation.cfm?id=340534.340761* 36, 38

Wang, S., Schuurmans, D., Peng, F. & Zhao, Y. (2005), 'Combining statistical language models via the latent maximum entropy principle', *Machine Learning* **60**(1), 229–250.
URL: *https://doi.org/10.1007/s10994-005-0928-7* 11

Wang, T. & Cho, K. (2016), Larger-context language modelling with recurrent neural network, *in* 'Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)', Association for Computational Linguistics, pp. 1319–1329. 13, 36

Wang, T., Yuan, X. & Trischler, A. (2017), 'A joint model for question answering and question generation', *arXiv preprint arXiv:1706.01450* . 1

Warner, B. & Misra, M. (1996), 'Understanding neural networks as statistical tools', *The American Statistician* **50**(4), 284–293.
URL: *http://www.jstor.org/stable/2684922* 9

Weiss, G., Goldberg, Y. & Yahav, E. (2018), Extracting automata from recurrent neural networks using queries and counterexamples, *in* 'International Conference on Machine Learning', pp. 5244–5253. 137

Wen, F., Chu, L., Liu, P. & Qiu, R. C. (2018), 'A survey on nonconvex regularization-based sparse and low-rank recovery in signal processing, statistics, and machine learning', *IEEE Access* **6**, 69883–69906. 128

Wen, W., Wu, C., Wang, Y., Chen, Y. & Li, H. (2016), Learning structured sparsity in deep neural networks, *in* 'Advances in neural information processing systems', pp. 2074–2082. 110

Wilks, S. (1960), 'Multidimensional statistical scatter', *Contributions to probability and statistics* **2**, 486. 110

Wilks, S. S. (1932), 'Certain generalizations in the analysis of variance', *Biometrika* pp. 471–494. 113

Williams, C. K. & Hinton, G. E. (1991), Mean field networks that learn to discriminate temporally distorted strings, *in* 'Connectionist Models', Elsevier, pp. 18–22. 71

Williams, W., Prasad, N., Mrva, D., Ash, T. & Robinson, T. (2015), Scaling recurrent neural network language models, *in* '2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)', IEEE, pp. 5391–5395. 73

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N. & Recht, B. (2017), The marginal value of adaptive gradient methods in machine learning, *in* 'Advances in Neural Information Processing Systems', pp. 4148–4158. 40

Wolpert, D. H. (2018), The relationship between pac, the statistical physics framework, the bayesian framework, and the vc framework, *in* 'The mathematics of generalization', CRC Press, pp. 117–214. 41

Xi, D. & Zhuang, D. (2017), Model selection of sequence prediction algorithms by compression, *in* 'International Conference on Grammatical Inference', pp. 160–163. 52

Xu, H. & Mannor, S. (2012), 'Robustness and generalization', *Machine Learning* **86**(3), 391–423. 40

Xu, W. & Rudnicky, A. (2000), Can artificial neural networks learn language models?, *in* 'Sixth International Conference on Spoken Language Processing'. 33

y Arcas, B. A., Fairhall, A. L. & Bialek, W. (2001), What can a single neuron compute?, *in* 'Advances in neural information processing systems', pp. 75–81. 21

Young, T., Hazarika, D., Poria, S. & Cambria, E. (2018), 'Recent trends in deep learning based natural language processing', *ieee Computational intelligenCe magazine* **13**(3), 55–75. 12

Yu, T., Simoff, S. & Jan, T. (2010), 'Vqsvm: A case study for incorporating prior domain knowledge into inductive machine learning', *Neurocomputing* **73**(13-15), 2614–2623. 46

Yuan, M. & Lin, Y. (2006), 'Model selection and estimation in regression with grouped variables', *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **68**(1), 49–67. 108

Zaremba, W. (2015), 'An empirical exploration of recurrent network architectures'. 31

Zaremba, W., Sutskever, I. & Vinyals, O. (2014), 'Recurrent neural network regularization', *arXiv preprint arXiv:1409.2329* . x, 29, 32, 57, 58, 65, 66, 82, 87, 88, 89, 90, 100, 107

Zhang, C., Bengio, S., Hardt, M., Recht, B. & Vinyals, O. (2017), 'Understanding deep learning requires rethinking generalization', *International Conference on Learning Representations* . 39, 40, 113

Zhang, C., Liao, Q., Rakhlin, A., Sridharan, K., Miranda, B., Golowich, N. & Poggio, T. (2017), Musings on deep learning: Properties of sgd, Technical report, Center for Brains, Minds and Machines (CBMM). 40

Zhang, D., Wang, H., Figueiredo, M. & Balzano, L. (2018), Learning to share: Simultaneous parameter tying and sparsification in deep learning, *in* 'Proc. of ICLR'. **URL:** *https://openreview.net/forum?id=rypT3fbob* 130

Zhao, Y., Chu, S., Zhou, Y. & Tu, K. (2017), Sequence prediction using neural network classiers, *in* 'International Conference on Grammatical Inference', pp. 164–169. 51

Zhou, H., Alvarez, J. M. & Porikli, F. (2016), Less is more: Towards compact CNNs, *in* 'European Conference on Computer Vision', Springer, pp. 662–677. 109

Zhu, X. & Goldberg, A. B. (2009), 'Introduction to semi-supervised learning', *Synthesis lectures on artificial intelligence and machine learning* **3**(1), 1–130. 24

Zhu, X., Zhou, W. & Li, H. (2018), Improving deep neural network sparsity through decorrelation regularization., *in* 'IJCAI', pp. 3264–3270. 110

Zilly, J. G., Srivastava, R. K., Koutník, J. & Schmidhuber, J. (2017), Recurrent highway networks, *in* 'Proceedings of the 34th International Conference on Machine Learning-Volume 70', JMLR. org, pp. 4189–4198. 32, 72, 89, 90, 100, 102, 104, 106, 114, 115, 117, 119

Zoph, B., Vaswani, A., May, J. & Knight, K. (2016), Simple, fast noise-contrastive estimation for large rnn vocabularies, *in* 'HLT-NAACL'. 35, 74, 80

Zou, H., Hastie, T. & Tibshirani, R. (2006), 'Sparse principal component analysis', *Journal of computational and graphical statistics* **15**(2), 265–286. 41