Exploiting Linguistic Task Hierarchies for Sequence Tagging in Natural Language Processing

Jonathan Godwin
Supervisors: Pontus Stenetorp & Sebastian Riedel

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Master of Science

of

Machine Learning.

Department of Computer Science

University College London

September 4, 2016

This report is submitted as part requirement for the MSc Degree in Machine Learning/CSML at University College London. It is substantially the result of my own work except where explicitly indicated in the text.

Code for this project can be found in the following public repository: https://github.com/jg8610/multi-task-project/

Abstract

This thesis investigates whether hierarchical structure in linguistic tasks can be used to improve the performance of Natural Language Processing algorithms. In particular, we investigate one such hierarchy (Part of Speech Tagging, Chunk Tagging and Language Modeling) in the context of a Neural Network algorithm, and find that adaptations to exploit the hierarchy improve performance.

Furthermore, we show that we can make use of the mixture of unsupervised (Language Modeling) and supervised tasks (Part of Speech Tagging and Chunk Tagging) within the hierarchy to conduct semi-supervised learning and, in doing so, improve the performance of the supervised tasks.

Acknowledgements

Thanks to my supervisor Pontus Stenetorp, who was incredibly generous with his time and without whom this project would never have gone very far. Thanks also to everyone in the Machine Reading group, who helped a huge amount with encouragement and endless tips.

Contents

List of Figures

List of Tables

Chapter 1

Introduction & Key Results

Natural Language Processing seeks to teach machines to understand language. If successful, NLP has a wide variety of use cases from language-based interaction with computers, to assisting drug discovery through reading biomedical data. However, many of the algorithms that perform Natural Language Processing tasks do not yet have the required level of accuracy to be applied widely. This thesis aims to introduce a method for improving performance, and demonstrate it with two example tasks.

Concretely, we investigate whether hierarchical structure in linguistic tasks can be used to improve the performance of Natural Language Processing algorithms. In particular, we investigate one such hierarchy (Part of Speech tagging, Chunk tagging and Language Modeling) in the context of a Neural Network algorithm. Furthermore, we show that we can make use of the mixture of unsupervised (Language Modeling[1]) and supervised tasks (Part of Speech tagging and Chunk tagging) within the hierarchy to conduct semi-supervised learning. Specifically, we investigate:

1. *Whether we can explicitly embed linguistically motivated hierarchical task structure in neural architectures for Part of Speech ('POS') tagging, Chunk tagging (also known as Noun-Phrase Chunking or Shallow Parsing) and Language Modeling, and what performance this explicit hierarchical structure brings.*

[1]Language Modeling is not strictly speaking unsupervised - the goal is predict the next word in a sequence. However, it does not require hand annotation and so we are terming the task unsupervised in this context.

2. *Whether, by incorporating an unsupervised auxiliary task (Language Modelling) to two supervised target tasks (POS tagging and Chunking), we can improve performance by conducting semi-supervised learning.*

We find that both designing a neural network around a task hierarchy, and including an auxiliary task improve the performance of our Chunk and POS taggers. Although we do not achieve State of the Art results, our performance is comparable to the best performing classifiers.

In this introduction we'll start by providing a deeper motivation for the project. Then we'll provide brief overview of POS tagging, Chunk tagging and Language Modeling and make the argument for using them as a task-set in multi-task learning. We'll then move on to discuss recent work on these problems in the NLP community, and finally outline our contribution in the context of both multi-task learning and semi-supervised learning.

Chapter 2 forms a literature review covering multi-task learning, semi-supervised learning and neural networks within NLP. We also go into more detail on POS tagging, Chunk tagging and Language Modeling. Chapter 3 documents our novel adaptations to existing algorithms.

Chapter 4 covers the data sets used in our experiments and in previous POS tagging, Chunk tagging and Language Modeling work; Chapter 5 covers experimental results and Chapter 6 is our conclusion and suggestions for further work.

1.1 Project Motivation

1.1.1 Linguistic Task Hierarchies

When we speak and understand language we are arguably doing many different linguistic tasks at once. At the top level we might be trying to formulate the best possible sequence of words given all of the contextual and prior information (like what has just been discussed and what the aims of the conversation are), but this requires us to do lower-level tasks like understand the syntactic and semantic roles of the words we choose in a specific context.

Computational linguists have tried to break down natural language understanding into these separate tasks, and design algorithms to do each of these separately. These tasks are numerous; they cover spotting named entities (names of people, for example), uncovering the full semantic structure of a sentence or predicting the sentiment of a document. An on-going area of research is how you might compose algorithms that accomplish each of these tasks separately, so that we have a single algorithm that might conduct multiple tasks at once just like a human (Collobert et al. (2011)), where each of the tasks provides additional information to the others to improve performance.

One such approach is to recognise that there is an implicit (and in some cases, explicit) linguistic task hierarchy. When teaching a computer to understand a sentence we might start with a task that figures out the syntactic properties of individual words ('Part of Speech tagging'). After that we might group these words into parts that have the same syntactic function ('Chunking'). Alongside these syntactic tasks we would also have to have a process that understands the meaning of the words of the sentence when put together ('Semantic Parsing'). Finally, these will feed into an overall task that assesses the overall meaning and correctness of the sentence and assigns it a score ('Language Modeling') .

This project seeks to provide an example of one such linguistic task hierarchy, and demonstrate that, by developing an algorithm that both exploits this structure and optimises all three jointly, we can improve performance. This research can then form a basis for similar approaches with other natural language processing tasks.

1.1.2 Semi-Supervised Learning and Linguistic Task Hierarchies

Most of the time each task in a task hierarchy will have a different number of data available in its training set [2]. If your goal is to try to write an algorithm that assigns a good score to well-formed sentences and a bad score to poorly-formed sentences in English, your data set is potentially billions of words long (you could just look at all the past issues of The Times!). Tasks such as these, where you can use raw input without any additional labelling, are called 'unsupervised' tasks. However, if

[2]In machine learning, the training set is the data the algorithm learns from.

you wanted to write an algorithm that picks out names of individuals based upon historical data you'd have to ask someone to hand-label a data set first. Tasks like these are called 'supervised' tasks.

Getting someone to hand-label data is expensive in time and cash. These considerations mean that for supervised tasks there are often only a very limited number of data, but for unsupervised tasks the number of data is huge. An interesting question is whether we can use some of the information we learn from conducting an unsupervised task to help the performance of our supervised task. This broad family of algorithms is called 'Semi-Supervised Learning'.

You can easily have a mixture of supervised and unsupervised tasks within a given linguistic task hierarchy. For this reason, it is a natural question for this project to see whether we can use a form of semi-supervised learning within our multi-task algorithm to improve performance.

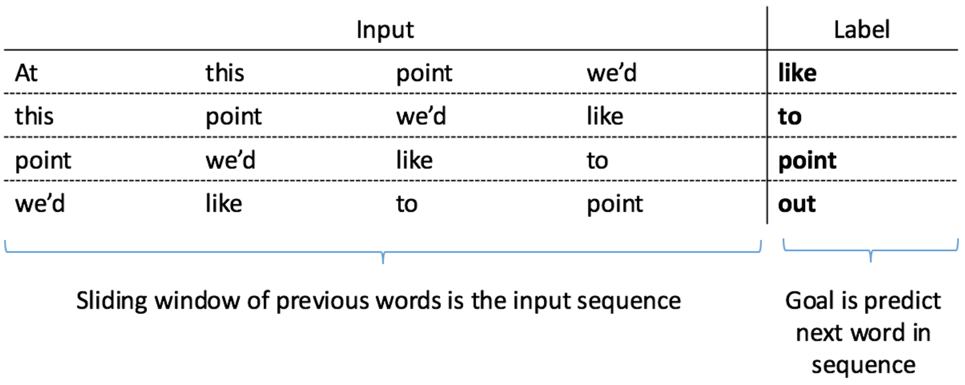
1.2 Example Task Hierarchy - POS tagging, Chunk tagging and Language Modeling

In our project we will examine a simple syntactic task hierarchy in detail in order to demonstrate the effectiveness of our proposed contributions. After the tasks have been introduced in this section, we will go on to discuss in more detail the different tasks and metrics for measuring performance starting in Section 2.5.

First, at the top of the hierarchy, we have language modeling – an unsupervised task that seeks to assign a probability to a sequence of words conditioned on prior information . In some ways, this task is the most similar to what a lay person might call Natural Language Understanding – most reading and speaking can be reformed as finding the most probable set of words given the correct prior information. In our model, we seek to predict the next word conditioned on the previous words in a sequence (Figure 1.1). [3]

In order to conduct language modeling effectively, knowledge of the syntactic

[3]Note: In our experiments, Language Modeling is used solely as an unsupervised auxilary task for the primary tasks of POS tagging and Chunking, and therefore we do not present results of the Language Model to be evaluated against existing benchmarks.



Input				Label
At	this	point	we'd	**like**
this	point	we'd	like	**to**
point	we'd	like	to	**point**
we'd	like	to	point	**out**

Sliding window of previous words is the input sequence
Goal is predict next word in sequence

Figure 1.1: Language Modeling Example. Here, we have a sliding window of words where the goal is predict the next word in the sequence. Ideally, we'd like to find the joint most likely sequence of words conditioned on all relevant prior information, but doing so is computationally intractable.

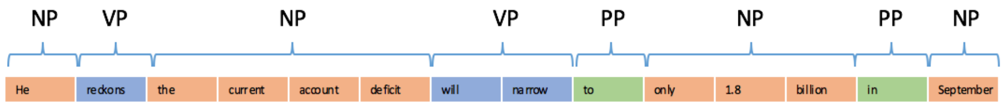
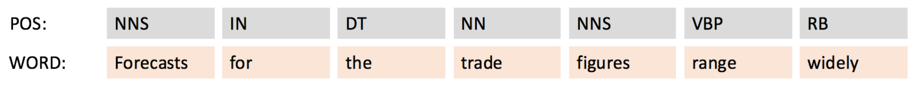


Figure 1.2: Chunk tagging Example. In the diagram, the words encompassed by each curly bracket are part of the same chunk. We have three types of Chunk in this sentence: 'NP' – Noun Phrase, 'VP' – Verb Phrase and 'PP' – Proposition.

structure of the sentence is very useful. For this reason our second task is putting words together into groups that have the same syntactic role - known as as a Chunk (Figure 1.2). This gives a 'shallow' parse tree of the sentence that can be used as a feature for language modeling. Since the Chunks of a sentence are not known before-hand, we need a hand annotated training set and so this task is a supervised task.

Finally, Chunking itself relies on knowledge about the role each word plays in the syntactic structure - for example, whether it is a noun, or a verb or an adjective. Labeling each word with this information is known as 'Parts of Speech' tagging , and forms the final task in our task hierarchy (Figure 1.3 shows a sentence that has been labeling with its Parts of Speech, Figure 1.4 shows a Chunking tree with POS tags). Again, the Part of Speech for each individual word requires hand annotation and so this is also a supervised task. A diagram of the task hierarchy can be seen in Figure 1.5.



POS:	NNS	IN	DT	NN	NNS	VBP	RB
WORD:	Forecasts	for	the	trade	figures	range	widely

Figure 1.3: Part of Speech tagging Example. Here, instead of groups of words being identified, each word has it's own label (its 'Part of Speech'). For this sentence we have 6 Parts of Speech: 'NNS' – Noun Plural, 'IN' – Preposition, 'DT' – Determiner, 'NN' – Noun Singular, 'VBP' – Verb Non-3rd Person Singular Present and 'RB' – Abverb.

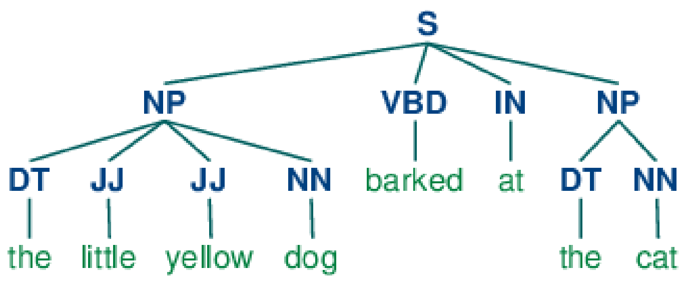


Figure 1.4: Pos and Chunk Relationship Example (Bird and Klein (2009)). In this figure, the chunks are at the first level in the tree (e.g. 'NP'), with the POS on the second level (e.g. 'DT').

At this point we'd like to emphasise that this task hierarchy has both supervised (POS tagging and Chunk tagging) and unsupervised (Language Modeling) tasks. This means that the size and variety of data available for training each task is very different - Language Modeling has significantly more data than POS tagging and Chunk tagging. A significant part of this project is investigating whether we can leverage the unlabelled data to make better predictions about POS and Chunk tags.

1.3 Summary of Relevant Work in Multi-Task Learning and Semi-Supervised Learning

Much recent work for multi-task learning in NLP has been based upon algorithms called 'Neural Networks'. In Section 2.2.2 we'll discuss these algorithms in detail, but for now we want to explain two important points. The first is that neural networks are essentially compositions of parametrised functions $F(X) = f_{\theta_1}(\sigma_{\phi_1} f_{\theta_2}(\ldots f_{\theta_n}(\sigma_{\phi_n} X))) \approx Y$ that minimises the loss – $L(F(X), Y)$ – between $F(X)$ and Y. The parameters for each of these functions are learnt using an algorithm called backpropagation. The second point that each of these functions is

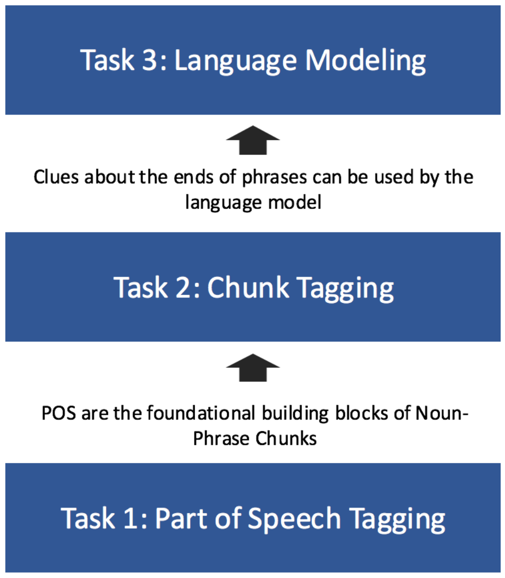

Task 3: Language Modeling
Clues about the ends of phrases can be used by the language model
Task 2: Chunk Tagging
POS are the foundational building blocks of Noun-Phrase Chunks
Task 1: Part of Speech Tagging

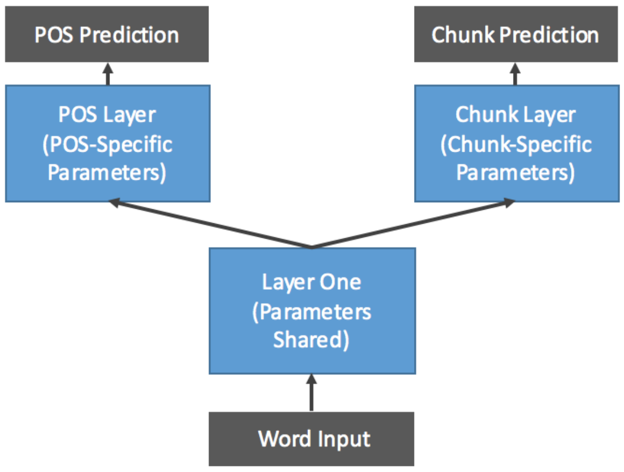
Figure 1.5: Task Hierarchy Rationale. Note that while at inference time information only flows up the hierarchy, during training senior tasks in the hierarchy can improve performance of junior tasks through the shared layer.

called a 'layer', and contains two parts - a linear transformation σ of the input followed by a non-linear function f applied to each dimension. Each layer learns a more abstract representation of the input than the last. [4]

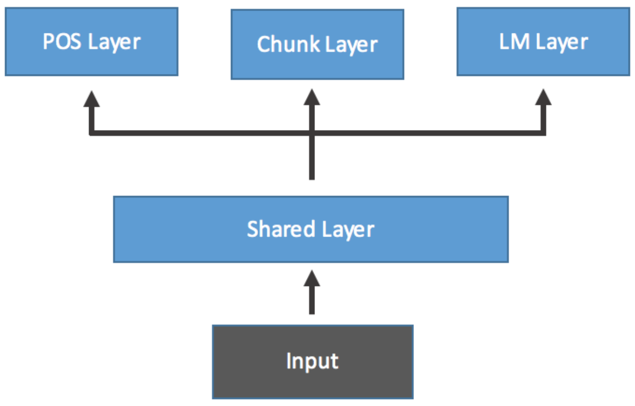
Multi-task learning in Neural Networks is achieved by sharing the parameters of one or more of these layers between different tasks, so that information can be shared between them. In the most recent multi-task learning work that trains POS and Chunk together (Yang et al. (2016)), POS and Chunk share parameters on the first layer, but have separate parameters on the second layer (a simplified diagram of the structure can be seen in Figure 1.6a). In this paper, optimising both POS and Chunk together rather than separately improves performance by 0.8 percentage points.

Semi-Supervised learning for Neural Networks in NLP is most often achieved

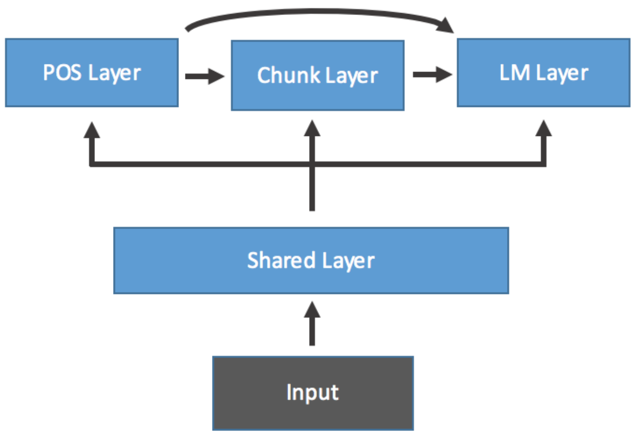
[4]Interestingly, it has been shown that you can approximate any function using one layer as long as you have a suitable non-linear function (for more details on what 'suitably non-linear function' means see Hornik et al. (1989)), and the dimensionality of the linear-transformation is sufficiently large. This raises the interesting question of why we need multiple layers in the first place, or any improvements on simple one-layer architectures. Part of the answer is that layer decisions embed prior beliefs about the task (e.g. that you know that the first layer should represent one feature, like relationships between nearby pixels in the case of a convolutional neural net) that often make them easier to train.


POS Prediction
Chunk Prediction
POS Layer (POS-Specific Parameters)
Chunk Layer (Chunk-Specific Parameters)
Layer One (Parameters Shared)
Word Input

(a) Simplified Structure from Yang et al. (2016). This is the most recent architecture used for multi-task learning in NLP.



(b) Our Baseline Model. Here, we've added Language Modeling as a task to improve the ability to learn good predictive structure. Notice also that this model is without an explicit hierarchy in the tasks.



(c) Our Model. This model is the same as above, except here we have created connections within the tasks according to the linguistic task hierarchy. This addition improves performance.

Figure 1.6: This figure outlines the different architectures motivated by our contribution. Note that the Baseline Model and Our Model are capable of semi-supervised learning because they conduct Language Modeling. In our experiments we conduct semi-supervised learning by alternating batches of unsupervised and supervised data.

through initialising some of the parameters of the supervised model with parameters pre-trained on an unsupervised model. In particular, it is common to represent each word as a dense vector (also known as 'word embedding'), and for these vectors to be trained using a neural network. The reason you'd want to represent words as vectors is because you could then represent semantic relationships between words as relationships in the vector space - for example, 'dog' and 'Labrador' would be close together in the vector space. Once this structure has been learned by an unsupervised algorithm (using lots of data), it can be exploited by a supervised algorithm (with less data) to achieve better results. In Section 2.3 we'll cover this topic in significantly more detail.

For our experiments we construct a baseline model based upon this existing work in multi-task and semi-supervised learning. In particular, we use a simplified [5] version of the architecture developed in Yang et al. (2016), but with the addition of the language modeling task (Figure 1.6b) and pre-trained word vector representations.

1.4 Our Contribution

A full discussion of our contribution can be found in Chapter 3.

There are two key things to note about the existing work on Multi-Task learning and Semi-Supervised learning discussed in the previous section. First, the architectures make no use of the implicit linguistic hierarchy with Parts of Speech and Chunking. The second is that, while semi-supervised learning exists in the form of word-vectors, there is no semi-supervised learning on the training corpus or related corpora that may improve results further.

Our contribution to the current approaches for POS and Chunk tagging are then twofold: first, we explicitly create a hierarchical structure for POS, Chunking and Language Modelling that ensures information from lower tasks in the hierarchy can be used by those higher up; second, by incorporating the Language Model we can train unsupervised representations using the training corpus and related corpora

[5]Specifically, we use simpler functions for layers one and two.

	Test $F_{\beta=1}$ (CoNLL)	Test $F_{\beta=1}$ (Genia)
Baseline Model	91.95 %	85.47 %
Explicit Task Hierarchy	**92.24** % (+ 0.29 %)	85.80 % (+ 0.37 %)

Table 1.1: Hierarchy Summary Results - Chunk

	Test Accuracy (CoNLL)	Test Accuracy (Genia)
Baseline Model	96.2 %	97.3 %
Explicit Task Hierarchy	**96.4** % (+ 0.2 %)	**97.6** % (+ 0.3 %)

Table 1.2: Hierarchy Summary Results - POS

that improve both the POS and Chunking performance (Figure 1.6c). For a more detailed description of both the hierarchical links and the method of semi-supervised learning, please see Chapter 3.

1.5 Summary of Experimental Results

For a full discussion of our experimental results, please see Chapter 5.

We test our results on two data sets – the CoNLL 2000 Newspaper text corpus and the Genia Biomedial text corpus. This is because Chunking for Biomedical text has been found to be significantly more difficult than Chunking for Newspaper text.

First, we run our baseline model (Figure 1.6b) to establish benchmarking results. These results can be seen in Table 1.1.

1.5.1 Explicitly Representing Task Hierarchy

As disucssed, the first of our contributions is explicitly embedding the linguistic task hierarchy in the neural architecture (Figure 1.6c). This addition results in a 0.3 percentage point increase in Chunking F1 Score on the CoNLL test set compared to a baseline model initialised with pre-trained word embeddings. Full results of these experiments can be seen in Tables 1.1 and 1.2.

Furthermore, embedding an explicit task hierarchy allows us to learn a vector representation for each POS and Chunk tag. When projected into low-dimensional space we can observe semantic regularities, particularly in the chunk tags where chunks are grouped together according prefix. A full discussion of the Task Hierarchy experimental results can be found in Section 5.2.1.

	No Pre-Trained Embeddings		With Pre-Trained Word Embeddings	
	Test $F_{\beta=1}$ (CoNLL)	Test $F_{\beta=1}$ (Genia)	Test $F_{\beta=1}$ (CoNLL)	Test $F_{\beta=1}$ (Genia)
Baseline Model	86.2 %	85.3 %	91.9 %	85.8 %
Semi-Supervised	**88.3** % (+ 2.1 %)	**85.7** % (+ 0.4 %)	**92.3** % (+ 0.4 %)	**85.8** % (+ 0.0 %)

Table 1.3: Semi-Supervised Summary Results - Chunk

	No Pre-Trained Embeddings		With Pre-Trained Word Embeddings	
	Test $F_{\beta=1}$ (CoNLL)	Test $F_{\beta=1}$ (Genia)	Test $F_{\beta=1}$ (CoNLL)	Test $F_{\beta=1}$ (Genia)
Baseline Model	92.4 %	97.5 %	96.2 %	97.3 %
Semi-Supervised	**94.2** % (+ 1.8 %)	**97.6** % (+ 0.1 %)	**96.2** % (+ 0.0 %)	**97.6** % (+ 0.3 %)

Table 1.4: Semi-Supervised Summary Results - POS

1.5.2 Semi-Supervised Learning

The second of our contributions is incorporating a (proportionally) large unlabelled data set into our training schedule to learn unsupervised representations of word tokens and avoid over-fitting to the training set. Experimental results with pre-trained word embeddings show a small improvement of 0.35 percentage points F1 score on Chunking against a baseline model. In addition, the model trains significantly faster (without unsupervised data our model converges after c.40 epochs, with unsupervised data it converges in c.10 epochs).

However, the largest improvements can be seen in models that do not utilise transfer learning in the form of pre-trained word embeddings. Here, there is no pre-trained unsupervised word representations - all of the work is undertaken by the auxiliary task. Here, we see a performance improvement of over 2 percentage points in Chunking F1 score in the CoNLL 2000 Chunking data set. Interestingly, the biomedical data set benefits least from both word embeddings and semi-supervised learning. We hypothesise that this is partly due to the little overlap between training and test vocabulary, and the distinct style and vocabulary for biomedical text.

The semi-supervised learning effect has interesting applications for situations where there are not pre-trained word embeddings (for example, in languages with smaller corpora). In this case, good results can be achieved without the need for corpora of over a billions tokens or months of training time. A set of summary results for this experiment can be seen in Tables 1.3 and 1.4. A full discussion of all semi-supervised experiments can be seen in Section 5.2.2.

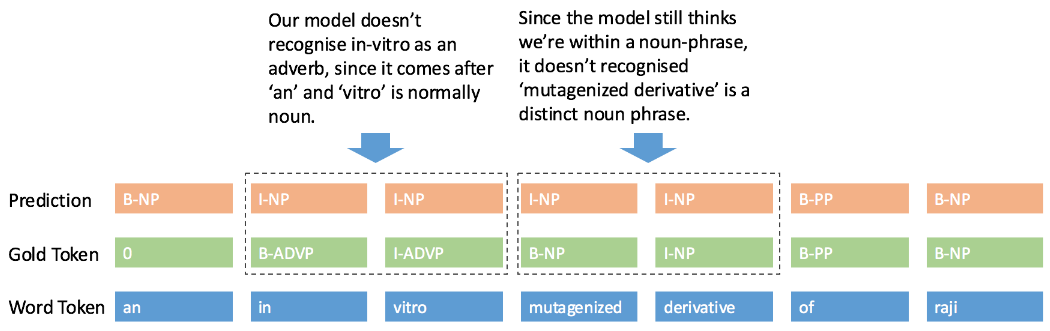

Our model doesn't recognise in-vitro as an adverb, since it comes after 'an' and 'vitro' is normally noun.
Since the model still thinks we're within a noun-phrase, it doesn't recognised 'mutagenized derivative' is a distinct noun phrase.
Prediction
Gold Token
Word Token
B-NP
I-NP
I-NP
I-NP
I-NP
B-PP
B-NP
0
B-ADVP
I-ADVP
B-NP
I-NP
B-PP
B-NP
an
in
vitro
mutagenized
derivative
of
raji

Figure 1.7: Example of Incorrect Sentence in Biomedical Text

1.5.3 Example Incorrect Sentence

Further insight into the challenges of our model can be generated by examining sentences in which mistakes were made. In Figure 1.7 we provide an example incorrect sentence from the Genia Biomedical corpus. In this sentence the model gets confused by unusual usage (Latin-esque) of the word 'in' as part of the adverb 'in vitro', and so over-extends the noun-phrase Chunk. In particular, it shows the difficulty of tagging unusual vocabularies and writing styles. More examples can be found Chapter 5.

1.5.4 Summary Analysis & Further Work

At the beginning of this introduction we introduced two questions we hoped to answer. We'll revisit each of those in turn, briefly describing the results of our investigation and suggestions for further work.

1. Whether we can explicitly embed hierarchical task structure in neural architectures for Part of Speech ('POS') tagging, Chunk tagging (also known as Noun-Phrase Chunking or Shallow Parsing) and Language Modeling, and what performance this explicit hierarchical structure brings.

We found that explicit embedding of hierarchical task structure can improve performance and offer valuable insight into the vector representation of low-level tasks learnt by the neural network.

Further work could investigate whether this performance improvement can be observed in other putative task hierarchies in Natural Language Processing and other machine learning disciplines. Furthermore, additional work could be conducted to quantify how much senior tasks are using the learnt representations of junior tasks.

2. Whether, by incorporating an unsupervised auxiliary task (Language Modeling) into a model with to two supervised target tasks (POS tagging and Chunking), we can improve performance by conducting semi-supervised learning.

We found that semi-supervised learning as described can improve performance both on models initialised with pre-trained word embeddings. However, the best results were found in situations with no prior pre-training, where our auxiliary task can achieve much of the unsupervised representation learning achieved with far larger corpora and training resources in a fraction of the time and with far fewer data.

Further research would investigate two outstanding problems in this area. The first is the problem of individual layers 'forgetting' their role when specific tasks are not optimised for large periods of time. This happens in situations where you might only have a small number of supervised data for one task, but a large number of supervised data for another. Solutions to this problem could involve selectively fixing weights for epochs without a training signal, or by providing a form of regularisation that trades off learning new information and forgetting old information.

The second problem is that of choosing the unsupervised task. Alternative tasks to language modelling could involve a variation of Generative Adversarial net or Actor Critic algorithms. Investigating which task improves the fully-supervised tasks the most would be a key area of follow-on research.

Chapter 2

Literature Review

2.1 Multi-Task Learning & Semi-Supervised Learning

2.1.1 Overview

For many applications in Machine Learning there are far more unlabelled data than there are labelled data. Ideally, we'd like to be able to use these unlabelled data to improve the performance of an algorithm for labelled data. This is the case for the tasks in our project - POS tagging and Chunking on the CoNLL 2000 task have 250k tokens, whereas unlabelled corpora for training word embeddings can have 6 billion tags Pennington et al.. This approach, where we leverage unlabelled data for a supervised task, is called Semi-Supervised Learning.

One approach to semi-supervised learning is to try and find a good predictive structure using unlabelled data. So, for example, we might try and find a good dense vector representation of a word using unlabelled data, and then leverage these representations when training a labelled task. This is one motivation behind word vector representations, covered in Section 2.3.

A related approach would be to jointly minimise two or more tasks that share some parameters. The intuition behind this is that each task provides a sample of the 'base' functional predictive structure, and that by sampling from many related tasks, we get a better estimate of what that base structure should be. If this intuition is correct, we should expect that by sharing parameters we can improve results on all tasks. The general approach of training a model to perform two tasks is called 'multi-task' learning (e.g. Evgeniou and Pontil (2004)), with the specific approach in which a predictive structure is uncovered by sharing parameters often called 'structure learning' or 'inductive transfer' Ando and Zhang (2005).

2.1.1.1 Structural Learning Outline

This outline is based on Ando and Zhang (2005).

In supervised learning we want an algorithm that maps an input $x \in X$ to an output $y \in Y$, where we assume the existence of some generating process D. We want to select a function $\hat{f}$ from a set of possible functions H_θ parametrised by θ, called a 'hypothesis space', that best approximates D, measured by minimising the expected error. The expected error is often approximated by the empirical error - if you minimise this error you doing what's known as Empirical Risk Minimisation:

$$\hat{f} = \underset{f}{argmin} \sum_{(x,y)=1}^{n} L(f(x),y)$$

where $L(f(X),y)$ is the error. Now assume we have M tasks each with N_m samples drawn from generating processes D_m, and hypothesis spaces $H_{m,\theta}$ with shared parameters θ. Then, if we are conducting Empirical Risk Minimisation we want to find $\hat{f}$ such that:

$$\hat{f}_{m,\theta} = \underset{f}{argmin} \sum_{i=1}^{n_m} L(f(x_n^m),y_n^m)$$

The aim is to find the optimal tied parameters θ such that the average empirical error on each task is minimised. We can see immediately that the more tasks we add, the better our estimate of the optimal tied parameters will be, since we have more data. In our project, our hypothesis space will be a neural network with tied parameters on specific layers.

It is also possible to make an intuitive argument for why related tasks should benefit from being trained jointly. Something you expect with a supervised learning algorithm is that similar inputs should have similar outputs - that the predictor is smooth in the input space. But this concept of similarity is often difficult to define.

Consider, for example, words - it isn’t clear that the correct way to measure the similarity of ’Apple’ and ’Orange’ even in a dense word embedding. By looking at multiple tasks for Natural Language Processing at the same time the distance measures between inputs intrinsic in language should be more discoverable than looking at one individually.

Relevant recent work in Multi-Task learning for Natural Language Processing has focused on training multiple, related tasks at the same time. In particular, Yang et al. (2016) shares first layer parameters for a two-layer neural network that optimises POS and Chunk jointly.

Word vector representations are the major semi-supervised contribution to Natural Language Processing. Here, word vector representations that encode some degree of semantic knowledge are learned on an unlabelled corpus (of billions of tokens), and then are used to initialise a supervised network. An in-depth discussion of word vectors is provided in Section 2.3.

2.2 Artificial Neural Networks (‘Neural Networks’)

2.2.1 Overview

Neural Networks have a long history. They were first introduced in the 1950s with the Perceptron Rosenblatt (1958) and the Multi-layer Perceptron algorithms, but were difficult to train (partly because they were not end-to-end differentiable). In the 1980s, Rumelhart, Hinton and Williams Rumelhart et al. demonstrated that using an algorithm called Back Propagation (’BP’), based upon the chain rule of calculus, one could train a fully differentiable neural network efficiently.

Limitations in the hardware of the day meant that deep networks were not trainable, and so early promising results failed to materialise into breakthroughs. Recent developments in Graphical Processing Units (GPUs) have meant that neural networks can be trained effectively, and they have proceeded to achieve state of the art results in many disciplines. One of the first breakthrough result of this era was the 2012 ImageNet competition Krizhevsky et al. (2012) in which a deep convolutional neural net beat the previous state of the art by c.12 percentage points. Neural

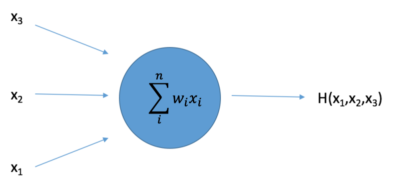


Figure 2.1: Basic Neuron

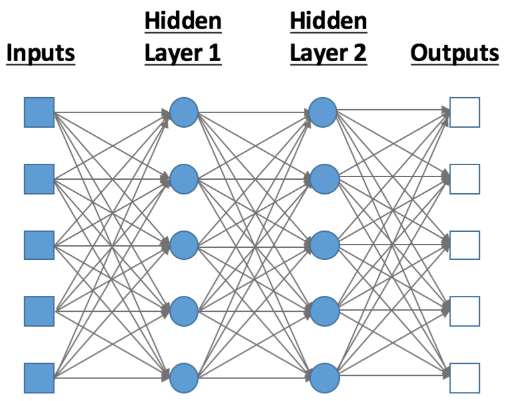


Figure 2.2: Basic Neural Net

Network models hold state of the art results in all of the tasks in this project.

2.2.2 Basic Neural Network

As the name suggests, a neural network is a network of neurons. A neuron is simply a function that takes in some inputs values x, and outputs a continuous value y. A basic schematic where the function is a linear combination of the inputs can be seen in Figure 2.1.

You can stack lots of these neurons with the same inputs together to make a ‘layer’, and you can put many layers together to make a ‘deep neural network’ (Figure 2.2). Depending on which functions you use for the neurons, a neural net can approximate any arbitrary function (Cybenko (1989)). This means that, theoretically, neural nets can learn arbitrary complex decision functions for classification tasks, which makes them very powerful models.

We can train a fully-differentiable neural network using back propagation. Here, we give a simple demonstration of the back propagation algorithm and its benefits (diagrams from Olah (2015)). For a more detailed derivation please see Rumelhart et al..

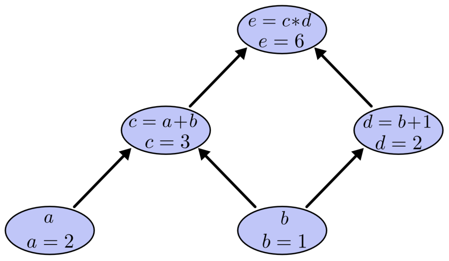


Figure 2.3: Example Computation Graph

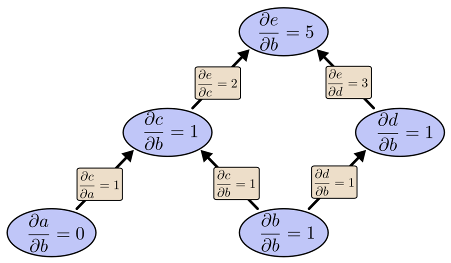

$\frac{\partial e}{\partial b} = 5$
$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$
$\frac{\partial c}{\partial b} = 1$
$\frac{\partial d}{\partial b} = 1$
$\frac{\partial c}{\partial a} = 1$
$\frac{\partial c}{\partial b} = 1$
$\frac{\partial d}{\partial b} = 1$
$\frac{\partial a}{\partial b} = 0$
$\frac{\partial b}{\partial b} = 1$

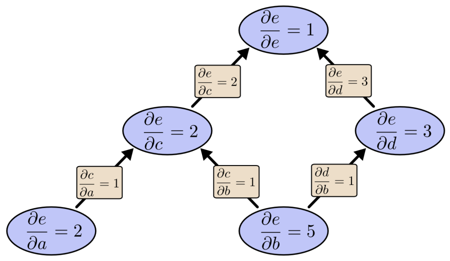
Figure 2.4: Forward Differentiation

The essence of back propagation is efficiently calculating the derivative of the outputs (the classification error) of the neural network with respect to the inputs, and then using standard methods to minimise the outputs. We will use the computation in Figure 2.3 to demonstrate how this is achieved.

A naive way to calculate the derivatives is to start with each input, and calculate the derivative of each output with respect to this input (Figure 2.4). In our computation, this would require four calculations for b. We then require two calculations for a - a total of 6 calculations. You can imagine how, if were to add significantly more inputs, the number of calculations would grow enormously. In situations like computer vision, where pictures may have millions of pixels, this method becomes impractical.

We can alternatively start at the outputs, and find the derivatives of the inputs with respect to the outputs. Utilising the tree structure of the computation graph, this can be computed in 5 computations - fewer than forward diff. Importantly, if we added another input we would only increase the number of computations by at most two, whereas for forward differentiation the additional computations could be 4.

For tasks such as computer vision classification tasks, where the number of



$\frac{\partial e}{\partial e} = 1$
$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$
$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$
$\frac{\partial c}{\partial a} = 1$
$\frac{\partial c}{\partial b} = 1$
$\frac{\partial d}{\partial b} = 1$
$\frac{\partial e}{\partial a} = 2$
$\frac{\partial e}{\partial b} = 5$

Figure 2.5: Backwards Differentiation

inputs is far larger than the number of outputs, backwards differentiation can be significantly faster. However, if the opposite is the case (i.e. fewer inputs than outputs) then forward differentiation would be significantly faster.

The final step is to use an optimisation algorithm such as Stochastic Gradient Descent to optimise the output function. A review of two optimisation algorithms, and our choice for this project, is in Section 2.2.4.

2.2.3 Cross Entropy Loss for Categorical Predictions

Categorical data in neural networks are often represented as one-hot vectors. These are vectors that are the length n (the number of labels) where every index is zero apart from i, the index of the label. For example, if you had 10 labels and you wanted to represent the label ‘3’, your vector would be of length 10 with zeros everywhere but the third index.

A neural network will predict the probability that an input x should be labelled i for each possible label, giving a probability distribution over the possible labels q (Figure 2.6). We then need to find a suitable loss function that represents how close our prediction is to the true distribution p, our one-hot vector.

An initial candidate could be the KL Divergence (Barber (2012)) which is a way to measure the ‘difference’ between two different distributions:

$$KL(p|q) = -\sum_x p(x) log q(x) + \sum_x p(x) log p(x) \tag{2.1}$$

$$= H(P,Q) - H(P) \geq 0 \tag{2.2}$$

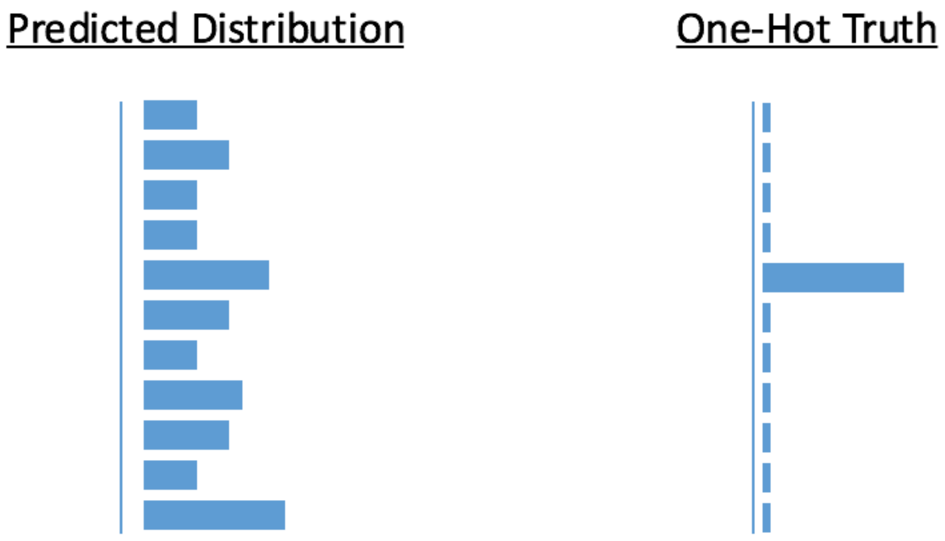


Figure 2.6: Predicted Distribution vs. Gold Distribution. Here we have a visualisation of the distribution predicted by the Neural Network (on the left), and a visualisation of the one-hot gold vector (on the right). The challenge is finding a suitable loss function.

Where $H(P,Q)$ is called the cross entropy between P and Q and $H(P)$ is called the entropy of P. The KL divergence is zero when the two distributions P and Q are identical, and so a natural objective for the Neural Network would be to minimise the KL divergence between the prediction and the gold token. However, we notice that only the cross entropy term $H(P,Q)$ is dependent upon the prediction Q, and so it is sufficient to minimise the cross entropy only. This is known as the cross entropy loss.

2.2.4 Optimisation Algorithms

2.2.4.1 Stochastic Gradient Descent

Gradient Descent is a very simple algorithm that minimises a function $f_\theta(\mathbf{x})$. In gradient descent, you take the derivative of the function at the current value of θ, $\nabla f_\theta(\mathbf{x})$ and subtract a proportion of it (called the learning rate) from the current value - $\theta := \theta - \alpha \nabla f_\theta(x)$, repeated for each time step until convergence. For a proof of convergence under certain conditions please see Barber (2012). A variety of methods using second-order derivatives, such as Newton's method, can be used to improve the performance but are typically not suitable for high-dimensionality spaces. For a fuller discussion again see Barber (2012).

A separate problem with gradient descent occurs when there are too many data to take the derivative of the model (or evaluate it) on all data points efficiently simultaneously. This could be, for example, in a large corpus of 1 billion words. In this case, we can make an approximation to the gradient by taking the gradient of a small batch of data - this is called Stochastic Gradient Descent and n this case, the update is $\theta = \theta - \alpha\nabla f_{\theta}(x_{batch})$. Again, this algorithm converges to a (possibly local) minimum depending on learning rate assumptions.

2.2.4.2 Learning Rate Annealing

If you choose a learning rate that is too large then the jumps in the objective function will also be too big - this means you may overshoot a minimum, or oscillate around a minimum. However, if your learning rate is too low, your training never really 'gets started' - it will converge so slowly that reaching an optimum will be inpractical.

To combat this, we often start off with a large learning rate and reduce it over time - known as 'learning rate annealing'. Particularly for algorithms without adaptive learning rates (Section 2.2.4.4), this hack can significantly improve results.

2.2.4.3 Momentum

Stochastic Gradient Descent can occasionally get stuck in saddle points, where we have a gradient of zero but aren't at a minimum. It is this problem that Momementum is designed to address.

The idea behind momentum is to update your current parameters using a moving average of your current gradient and previous gradients (called a 'velocity' vector), so that when you are in an area of zero gradient you still make a in the direction of the weighted average of your previous updates. This weighted average is termed 'momentum'. Specifically, we replace the Stochastic Gradient Descent update with:

$$v_{t+1} = \mu v_t - \varepsilon \nabla f(\theta_t) \tag{2.3}$$

$$\theta_{t+1} = \theta_t + v_t \tag{2.4}$$

An adaptation of momentum, called Nesterov’s Accelerated Gradient (‘NAG’) changes the update to the velocity vector. This update ‘looks ahead’ by taking the gradient of your velocity vector plus the current parameters, which significantly improves convergences. The update then becomes:

$$v_{t+1} = \mu v_t - \varepsilon \nabla f(\theta_t + \mu v_t) \tag{2.5}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{2.6}$$

Adding momentum to your optimiser can significantly improve convergence speed. For a full discussion please see Sutskever et al..

2.2.4.4 Adaptive Gradients

Stochastic Gradient Descent has another key problem - a constant learning for each parameter rate will emphasise highly varying parameters over less-varying parameters. To achieve the best results, the learning rate has to adapt per dimension to compensate for this bias. Alternatives to Stochastic Gradient Descent that try to address this problem like AdaGrad (Duchi et al. (2011)) weight old gradient updates and new gradient updates equally. This is a problem in non-stationary environments where we’d like to favour new gradient updates. One approach to solving both of these problems at the same time, the algorithm called *Adam*, is described below.

In 2014, an algorithm called Adam was introduced that addresses the key problems associated with Stochastic Gradient Descent (Kingma and Ba (2014)). First, it adapts the the update to emphasise less frequently varying parameters, it then adds a bias to ensure that there is less variance in the update. For a full derivation of the algorithm please see the original paper Kingma and Ba (2014). The full algorithm can be seen in Algorithm 1.

Given these improvements over Stochastic Gradient Descent (even with Momentum) demonstrated by Adam, we have decided to use Adam as the optimiser for this project.

```
Input: $\alpha$: Step size learning rate
Input: $\beta_1 \beta_2 \in [0,1)$: Exponential decay rates for the moment estimates
Input: $f(\theta)$: Stochastic objective function with parameters $\theta$
Input: $\theta_0$: Initial Parameter Vector
$m_0 \leftarrow 0$ (Initialise 1st moment vector) ;
$v_0 \leftarrow 0$ (Initialise 2nd moment vector) ;
$t \leftarrow 0$ (Initialise timestep) ;
while $\theta_t$ not converged do
    $t \leftarrow t+1$;
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t stochastic objective at time step t) ;
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$ (Update biased first moment estimate) ;
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$ (Update biased second raw moment
      estimate) ;
    $\hat{m}_t \leftarrow m_t \ (1-\beta_1^t)$(Compute bias-corrected first moment estimate) ;
    $\hat{v}_t \leftarrow v_t \ (1-\beta_2^t)$ (Compute bias-corrected second raw moment estimate)
      ;
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t/(\sqrt{\hat{v}_t} + \varepsilon)$ (Update parameters) ;
end
```

Algorithm 1: Adam

2.2.5 Recurrent Neural Networks ('RNNs')

In a story, the words and sentences at the beginning affect the words and sentences that come at the end - this is an example of sequential data, where the predictions of the future are dependent on what has come before. Much of the analysis we would like to do with Neural Networks is sequential; language and video are two examples.

Neural Networks of the type that we discussed in Section 2.2.2 can't deal with this sort of data - they evaluate one input at a time, unconditioned on the previous

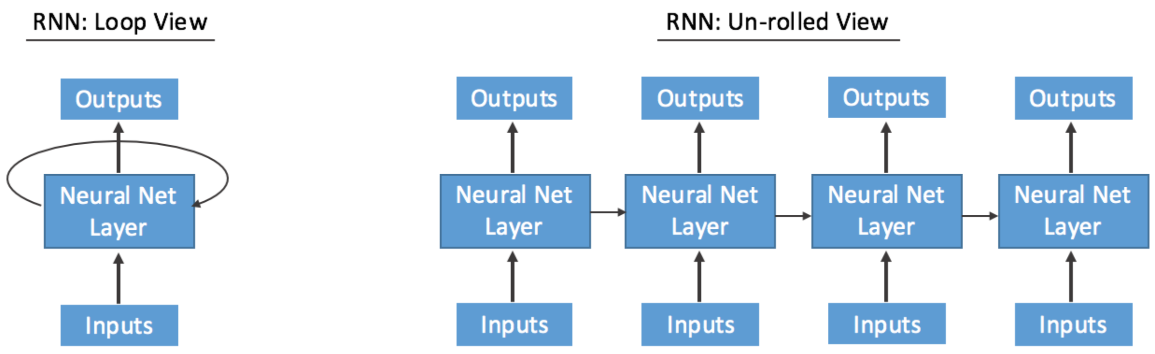


Figure 2.7: Two RNN Views

inputs. Recurrent Neural Networks are an adaptation of the basic Neural Network that allow us to deal with these data. Formally, for a sequence of inputs $x_1,\ldots,x_N$ and labels $y_1,\ldots,y_N$, an RNN will predict $\hat{y}_n = q_\theta(x_n|x_{n-1},\ldots,x_1)$ (where θ are the parameters of the RNN) and minimise $\frac{1}{n}\sum_n Loss(y_n,\hat{y}_n)$. The algorithm to minimise this loss, called Back Propagation Through Time (‘BPTT’) was introduced in Werbos (1990).

There are two complementary ways to view a recurrent neural network, both seen in Figure 2.7. The first is a Neural Network with a loop in it - so information (‘cell state’) from the previous time step is fed through to the current time step. In this view it’s really easy to see the simplicity of the RNN architecture. The second view is the ‘unrolled’ view - where we show the net for each time step. In this view you can easily see that each time step has it’s own loss, and that the RNN has to optimise the joint loss over each time step because the parameters are tied. This is the reason behind the average loss in the above paragraph.

Because RNNs can be very long in the horizontal direction, and also 2-3 layers deep, RNNs are particularly subject to the ‘vanishing gradient’ problem in which gradients become very small for many of the parameters in the network (which prevents training from taking place), or the ’exploding gradients’ problem, where gradients can become too large and oscillate. For tasks where you want to learn long-term dependencies (e.g. in text), this is a significant problem (For a full theoretical discussion of this problem please see Hochreiter et al. and Bengio et al. (1994)). One solution to this is to use specialist ‘cells’ that are designed to handle

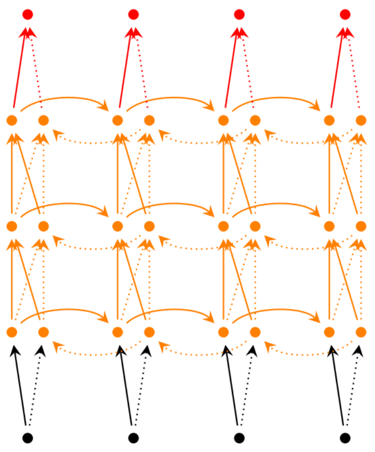


Figure 2.8: Bi-directional RNN

long-term dependencies such as the 'Long Short Term Memory' cell and the 'Gated Recurrent Unit'.

2.2.6 Bi-directional RNNs

Recall that, in a Part of Speech Tagger, for input word n_t an RNN will output a probability distribution over the POS-tag vocabulary conditioned on all previous words $q_\theta(p_t|n_{t-1},\dots,n_1)$. However, since we have the whole of the sentence it makes sense to try and use the information on what comes next to predict p_t. This is the motivation behind a Bi-directional RNN, and they have achieved state of the art in Chunking (Yang et al. (2016)) and POS tagging (Andor et al. (2016)). A diagram of a Bi-directional RNN can be seen in Figure 2.8.

In a Bi-directional RNN we run left-to-right through the sentence, and then we run right-to-left. This gives us two hidden states at time step t - one from the left-to-right pass, and one from the right-to-left pass. These are then combined to provide a probability distribution for the POS token conditioned on all of the other words in the sentence $q_\theta(p_t|n_T,,\dots,n_{t+1},\dots,n_{t-1},n_1)$. The same procedure can be used on other sequence-tagging tasks, such as Chunking. The strong point of Bi-directional RNNs, that it looks forwards as well as backwards, is also its weak point since it prevents online or continuous prediction (Lipton et al. (2015)).

2.2.7 Long Short-Term Memory Cells (‘LSTMs’)

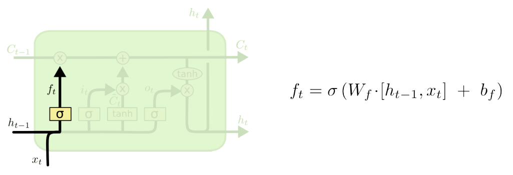
LSTMs, introduced in Hochreiter and Schmidhuber (1997) are specially designed to learn long-term dependencies (and so avoid the vanishing gradient problem). An LSTM replaces a single neuron in a traditional network with a series of 4 ‘gates’ which control what information is passed from time step to time step in the RNN and which information is used for prediction. Because prediction is based upon this ’cell state’ and not the gradient, LSTMs are more stable than traditional RNNs. Empirical evidence shows that the LSTM can learn things as sophisticated as when tokens are within or outside parentheses (Karpathy (2015)). The diagrams for this section (Figure 2.9) are taken from Olah (2015).

The idea behind an LSTM is that you have a four-stage process in a cell: you first figure out which information is not relevant from the previous time step (‘Forget’ - Figure 2.9a), you then figure out which information is relevant from the current time step (‘Input’ - Figure 2.9b), you then update the cell state (‘Update’ - Figure 2.9c) and then make your prediction or input to the next layer - (‘Output’ - Figure 2.9d).

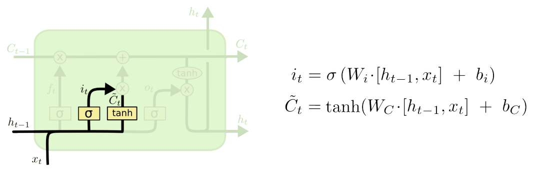
The Forget Gate applies a sigmoid function to a linear combination of the hidden state and input representing which parts of the previous hidden state should be forgotten. The Forget Gate was not originally part of the LSTM, and was introduced in Gers et al. (2000). The equation for this gate can be seen in Figure 2.9a and below:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.7}$$

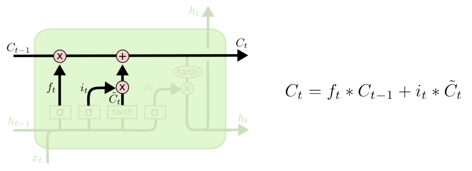
The Input Gate has two parts. The first part is a sigmoid function that decides (as continuous value between 0 and 1) which parts of the cell state need additions. The second part is a tanh layer that decides which information to add to the cell state. The equations for these gates can be seen in Figures 2.9b and below:


$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$

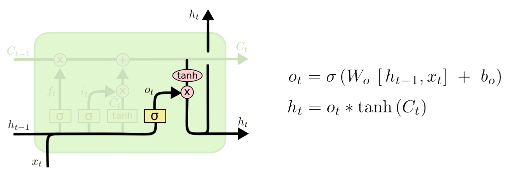
(a) Forget


$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$
$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$

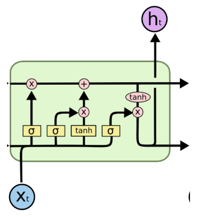
(b) Input



(c) Update Gate


$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$
$h_t = o_t * \tanh\left(C_t\right)$

(d) Output



(e) Full LSTM Cell

Figure 2.9: LSTM

$$i_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_i) \quad (2.8)$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad (2.9)$$

The Update Gate combines the results of the Input Gate and the Forget Gate to update the Cell State with simple multiplication and addition. Importantly, the Cell State is fully passed on to the next time step, hindering the vanishing gradients problem. The equation for these gates can be seen in Figure 2.9c and below:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.10}$$

Finally, we decide what to output. This is simply done with a combination of a sigmoid function on the input multiplied by a tanh function on the Cell state (Figure 2.9d).

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.11}$$

$$h_t = o_t * tanh(C_t) \tag{2.12}$$

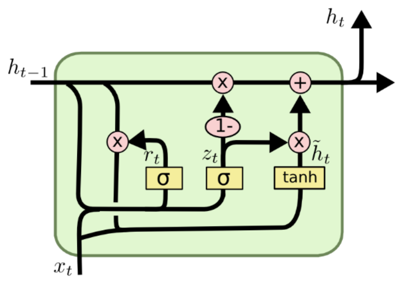
Note that by stacking the weights of all the separate functions, the LSTM can be computed in one vector product. As such, it can be considered the activation function for one layer.

2.2.8 Gated Recurrent Units (GRUs)

A Gated Recurrent Unit (introduced in 2014 by Cho et al. (2014)) is a cell inspired by the LSTM, but is simpler to compute and implement. They have been shown (Chung et al. (2014)) to have comparable performance to the LSTM on sequence modelling tasks. Indeed, an adaptation of the GRU achieves state of the art performance for Chunking (Yang et al. (2016)).

The essence of the GRU is to merge the hidden (output) state with the cell state, and make the forget and input functions more compact. A diagram of a GRU can be seen in Figure 2.10.

The GRU combines the 'forget' and 'input' gates in the LSTM by a singe 'reset' (a sigmoid activation function) gate on a linear combination of the output of the previous time step h_{t-1}, and the current input x_t. When the reset gate is set to zero, the cell will 'forget' all previous information. This gate allows the gate to decide which information is useful and which isn't.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 2.10: GRU Diagram

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \tag{2.13}$$

Using the reset gate we generate a new candidate hidden state $\tilde{h}_t$ from the 'remembered' information from the previous state h_{t-1} using a tanh (or alternative non-linear) activation function:

$$\tilde{h}_t = tanh(W \cdot [r_t * h_{t-1}, x_t]) \tag{2.14}$$

Finally, we create an update gate that provides a weight between the previous hidden state h_{t-1} and the candidate new hidden state $\tilde{h}_t$. We then combine these in a weighted sum to get the new hidden state:

$$\tilde{z}_t = tanh(W_z \cdot [h_{t-1}, x_t]) \tag{2.15}$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{2.16}$$

2.3 Vector Representations of Words

2.3.1 Overview

We use words like 'Apple' and 'Orange' in similar contexts because they both refer to similar things - they have a semantic relationship. A perfect NLP system would understand the semantic relationship between words, ideally with these semantic relationships embedded in the representation of the word itself. This is the motivation behind Vector Representations of Words.

The classical NLP representation of a word as an index or a one-hot unitary vector fails in this objective since all words are orthogonal to one another - in this representation, ‘Apple’ and ‘Orange’ are as alike (or dis-alike) as ‘Apple’ and ‘Car’. This sort of representation is referred to as *Bag of Words*.

An alternative approach is to represent each words as a dense, continuous vector known as a *Word Embedding*. In this case we have an intuitive way to measure similarity - their distance from each other (however measured). Any NLP model that has used pre-trained word embeddings during its own training will adapt to vocabulary outside its training set, since the vector representation of the out of training vocabulary words will be similar to those inside the training vocabulary.

Word Embeddings have a long history in NLP (Hinton (1984)), but have only recently succeeded in capturing the desired semantic relationships (Pennington et al., Mikolov et al. (2013b)).

A key way of improving performance in NLP tasks is to initialised your algorithm with pre-trained word embeddings. This is one of the most popular forms of semi-supervised learning (also called ‘transfer learning’), and so significant time in this literature review is devoted to exploring these approaches in detail. In particular, we note here that the Language Modeling task in our task hierarchy can be viewed as a ‘local’ way of training word embeddings, and so is very related to some of the methods discussed below.

2.3.2 Different Training Techniques

At the heart of training Word Embeddings is the *distributional semantics hypothesis*. This states that a word’s meaning is defined by its context, and so words with *similar meanings will have similar distributions* (Harris (1954)). All of the varying methods for training word embeddings start with this assumption about the origins of word meaning.

There are two paradigms in training word embeddings:

Table 2.1: Example Co-occurrence Table

Probability	$k = solid$	$k = gas$	$k = water$	$k = fashion$
P(k—ice)	1.9×10^4	6.6×10^5	3.0×10^3	1.7×10^{-5}
P(k—steam)	2.2×10^5	7.8×10^4	2.2×10^3	1.8×10^5
p(k—ice)/p(k—steam)	8.9	8.5×10^2	1.36	0.96

1. **Local Models.** Local Models are designed to aid in making predictions given an individual context window. Typically, they are trained by asking a Neural Network to predict either the context for a given word ('Skip-Gram'), or a word for a given context ('CBOW'). These models, including the *word2vec* (Mikolov et al. (2013a)), have performed well on semantic relationship tasks and exhibit surprising regularities in the vector space (e.g. $king - man + woman = queen$ (Mikolov et al. (2013b))). These models are not able to make use of word or phrase repetition in a corpus.

2. **Global Models.** Global Models leverage the statistical information of an entire corpus. Typically, these models build co-occurrence matrices (For an example please see Table 2.1) for each word in the corpus vocabulary and attempt matrix-factorisation to recover a vector for each word. These models have traditionally suffered from poor performance on semantic relationship tasks, and computational complexity. Recently, however, new models such as 'Global Vectors for Word Representation' ('GloVe') (Pennington et al.) have addressed some of these concerns.

The following sections, explore the main models in more detail, and make a recommendation that we use GLoVe embeddings.

2.3.2.1 Local Models

The following explanations are based upon Mikolov et al. (2013a).

Neural Net Language Modeling ('NNLM'). In 2003 Yoshua Bengio introduced Neural Language Models (Bengio et al. (2003)), in which a Neural Network outputs a probability distribution over the vocabulary for the next word in a sequence conditioned on the previous words. The model has one non-linearity following by a linear projection. Each word is represented by a dense vector, which is trainable by back propagation since the whole network is differentiable.

Neural Net Language Modeling is very computationally expensive. Consider the computational complexity per training example (Mikolov et al. (2013a)) (D is the word vector dimension, N is the length of the context, H is the hidden non-linear layer size and V is the size of the vocabulary):

$$C = (N \times D) + (N \times D \times H) + (H \times V)$$

For a corpus with a vocabulary of potentially hundreds of thousands of words, and a non-linear layer that may be 500 neurons wide, this cost term is dominated by $H \times V$. Even with adaptations to improve this performance, such as a Hierarchical Softmax[1] (Morin and Bengio (2005)), the computation cost can only decrease V to $log_2 V$. The computational cost of a network of this size with a non-linearity prohibits it from being trained efficiently on corpora of billions of words, the size of the available data. Alternative models seek to maintain performance with architectures that do not have any non-linearities.

Many of the POS and Chunk tagging results use a pre-trained embeddings matrix called 'SENNA' that has been trained using a method similar to the NNLM (Collobert et al. (2011)).

Continuous Bag of Words ('CBOW') The CBOW architectures makes simplifying assumptions to the NNLM model in two ways. First, it removes the non-linearity. Second, it averages the word-vectors before the word prediction. The averaging function removes the information on the order of the context words, and so the model is termed a 'bag of words' model. A diagram of the CBOW model can be seen in Figure 2.11.

By still using a Hierarchical Softmax, we derive the complexity of CBOW as (Mikolov et al. (2013a)):

$$C = (N \times D) + (D \times log_2(V))$$

[1]When you take a normal Softmax in a language model, your partition function has to sum over the whole vocabulary. The idea behind a Hierarchical Softmax is that you only sum over the 'relevant' context for the word.

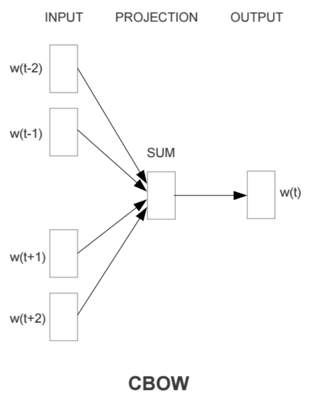


Figure 2.11: CBOW Architecture

Skip-Gram Model The skip gram model does the reverse of the CBOW task - it asks a neural network to predict the context given a target word vector. A diagram of the Skip-Gram Model can be seen in Figure 2.12.

Specifically, each current word embedding is put through a log-linear classifier, which predicts the words within a specified distance before and after the current word. Like the CBOW model, we do not have non-linearities. Experiment results have shown that the longer the context prediction, the better the word embeddings but the higher the computational cost.

Again, using a Hierarchical Softmax we arrive at the complexity of the Skip-Gram Model (Where *Context* is the context length). This model's computational complexity again allows it to be trained of corpora of billions of words :

$$C = Context \times (D + D \times log_2(V))$$

2.3.2.2 Global Models

Global Models (unlike Local Models) use global statistical information about the entire corpus, primarily in the form of co-occurrence relationships. Historically they have suffered from computational complexity and poor performance on semantic relationship tasks, but recent algorithms have addressed these concerns. We therefore use GloVE embeddings for our experiments.

Latent Semantic Analysis ('LSA'). LSA is a popular historical method of creating word embeddings (Deerwester et al. (1990)). LSA starts by constructing

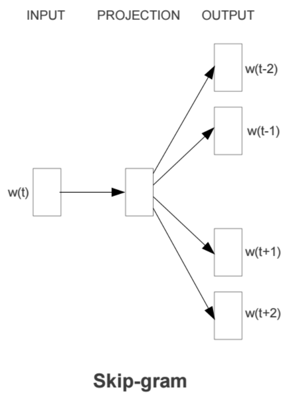


Figure 2.12: Skip Gram Architecture

a co-occurrence matrix (for an example see Table 2.1) for an entire corpus, and then finds a low-rank approximation to the co-occurrence matrix using Singular Value Decomposition ('SVD'). For example, say C is the co-occurrence matrix with SVD $C = U\Sigma V^T$. We then take the r (where r is the desired size of your word embeddings) largest singular values to create Singular Value Matrix $\tilde{\Sigma}$, and create the low-rank approximation $\tilde{C} = U\tilde{\Sigma}V^T$. It can be shown that the SVD minimises the frobenius norm error for low-rank matrix approximation. Since the complexity of computing the SVD is cubic in the size of the vocabulary, this method suffers at large scale.

Latent Dirichlet Allocation ('LDA'). LDA is generative model where a corpus is viewed as a mixture of latent topics. First introduced as a graphical model in 2003 (Blei et al. (2003)) (A diagram can be seen in Figure 2.13), the model has been very influential in topic modelling.

LDA proceeds by defining each topic as a distribution over the vocabulary of a corpus. If you have k topics, then it is simple to construct a word embedding - you simply take the probability $p(word|topic)$ for each of the words and topics to create a topic-word matrix. The word vector is then simply a row in the matrix.

This method has the nice property that each of the dimensions in a word embedding have a specific, human interpretable value. However, it fails to provide a good way to represent semantic relationships - the distance between words in the vector space does not relate to their similarity.

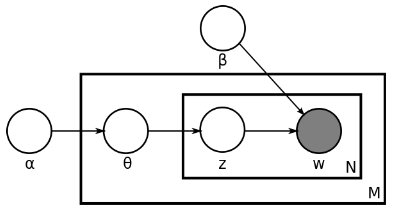


Figure 2.13: LDA Graphical Model. α is is the parameter of the Dirichlet prior on the per-document topic distributions, β is the parameter of the Dirichlet prior on the per-topic word distribution, θ is the topic distribution, γ is the word distribution, Z is the topic matrix, W is the word matrix, N is the number of words and M is the number of documents.

GloVe. GloVe (Pennington et al.) makes two key contributions to Global Word Embedding training models that address important drawbacks in historical approaches. First, it recognises that the ratios between co-occurrences are more important than the absolute values. This means that noise in non-discriminatory words (unrelated to the target word in question) is cancelled out, and the remaining words are discriminatory (See Table 2.1). The second is that the low-rank approximation of the co-occurrence matrix scales quadratically on the number of non-zero entries in the co-occurrence matrix, significantly enhancing computation time and performance.

Specifically, GloVe seeks to approximate the log ratio co-occurrence matrix with the formula $w_i^T \tilde{w}_j + b_i + \tilde{b}_j = log(X_{i,j})$, minimising the following weighted least squares objective:

$$J = \sum_{i,j=1}^{V} f(X_{i,j}) \left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - log(X_{i,j}) \right)^2 \tag{2.17}$$

Where $f(X_{i,j})$ is some weighting function, $w_i, \tilde{w}_i$ are word vectors, $b_i, \tilde{b}_i$ are their biases and $X_{i,j}$ is the ratio co-occurrence matrix. The parameters, w, $\tilde{w}$, b, and $\tilde{b}$ are learned with a neural network. Here the way in which the matrix is only trained on the non-zero elements is clear, since zeroed elements contribute nothing to the loss. Empirical evidence shows a large majority of the entries of the matrix are zero. Glove Word Embeddings exhibit the same linear substructure (e.g. King

Table 2.2: Local & Global Model Comparative Results (300 dim, trained on 6bn data points) (Pennington et al.)

Model	Semantic-Syntactic Word Relationship Test Set		MSR Word Relatedness
Architecture	Semantic Accuracy [%]	Syntactic Accuracy [%]	Test Set
CBOW	63.6	67.4	65.7
Skip-Gram	73.0	66.0	69.1
GloVe	77.4	67.0	71.7

- Man + Woman = Queeen) as Skip-Gram models. Since GloVe trains globally and has state of the art results on the Semantic-Syntactic Word Relationship test, it is used in our experimental results.

2.3.2.3 Comparative Results

As we see in Table 2.2, the glove model has the best overall performance on the Semantic-Syntactic Word Relationship Test, trained on the same number of tokens and with the same embedding size. For this reason, we use the GloVe embeddings as the primary choice in our architecture.

2.4 Improving Neural Network Performance

2.4.1 Word Embedding Shrinkage & Fine Tuning

It is common practise to initialise a neural network with pre-trained word embeddings to improve performance (e.g. Yang et al. (2016)). However, there are often specific regularities associated with your task or corpus that mean that transformation or further training of the word embeddings is required. In these cases, we have two approaches: (1) fine-tuning the word vectors by making them trainable parameters within our Neural Network (called fine tuning, used in Yang et al. (2016)), or (2) adding a trainable non-linear projection from the word embeddings to the first hidden layer (called word embedding shrinkage).

One key consideration is whether you want to train embeddings for words not in your pre-trained corpus. This is particularly the case in biomedical text for instance, where many words may not be found in the pre-trained word embeddings, but you'd like to learn something about their semantic relationship to other words.

[1]No Comparative Results available for SENNA word embeddings

In this case, you'd want to use fine tuning.

In other cases, fine tuning of pre-trained word vectors with large size (the largest GloVe vectors are 300-dimensional) can lead to over-fitting. In these cases, we could use a projection from the 300-dimensional space down to a smaller dimension. This projection forces the model to use only a portion of the information contained in the word embedding and as such encourages generalisation.

In the existing work on POS tagging and Chunking, fine-tuning is the dominant method of word embedding initialisation, and so this is used for our experimental results.

2.4.2 Sentence Sequences & Window Sequences

Some tasks in natural language processing benefit from information not contained in a sentence. For example, if you are trying to predict the next word in a paragraph, you might want to condition on all of the words in the paragraph rather than just those in the sentence. Algorithms that take a fixed-sized window before the word are called 'window' sequence methods.

For other tasks, you might want to give hints to your algorithm to look at the position of the word in the sentence. This could be the case for POS tagging and Chunking, since they are trying to find structure in the sentence. In this case, it makes sense to break up your corpus into sentences, and feed these into the algorithm one by one, rather than as an arbitrary window. These methods are called 'sentence' sequence methods.

Since Language Modeling is the dominant task in our hierarchy, we are using window batching - the established batching method for this task.

2.4.3 Dropout

Neural Networks are very powerful function approximators, which often means that they are prone to over-fitting. One possible way to prevent over-fitting is to average multiple different models to get an expected prediction. However, it is computationally infeasible to exactly average large numbers neural networks (there are 2^n possible networks for n nodes), since it can take many days for a model to

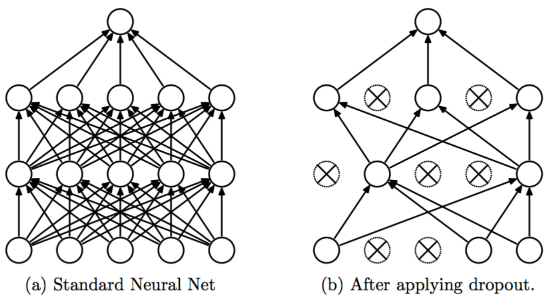


Figure 2.14: Dropout Example. On the left hand side you have a fully connected network. On the right hand side you have a 'thinned' network where a random number of connections are missing. (Srivastava et al. (2014))

train. Dropout can be seen as a way to provide an approximation of the average of many different networks.

Dropout (Srivastava et al. (2014)) does this in a very simple, but very effective manner for Neural Networks. The basic idea is that you randomly 'drop out' connections in your neural network during training, using 'thinned' neural network (Figure 2.14). At test time, we want to average the weights, which we approximate by rescaling. In practice, it improves performance of neural networks by a significant margin (e.g. 0.2 percentage points on the MNIST data set (Srivastava et al. (2014))).

2.5 Part of Speech Tagging

2.5.1 Introduction

A Part of Speech is a label for a group of words that have similar grammatical properties, given to a word based upon its definition and where it appears in the sentence. Simplified Parts of Speech should be familiar (*noun, verb, adjective*) but a Part of Speech tag set that covers the variety of grammatical properties is far larger, incorporating, for example, tokens for punctuation and numbers. POS tagging is the task of labelling every word or token within a corpus with its POS tag. Figure 2.15 demonstrates an example tagged sentence.

The difficulty in writing a POS tagger arises because a single word can have different roles within a sentence dependent upon the context. For example, the

POS:	NNS	IN	DT	NN	NNS	VBP	RB
WORD:	Forecasts	for	the	trade	figures	range	widely

Figure 2.15: Part of Speech Tagging Example. Here, instead of groups of words being identified, each word has it's own label (its 'Part of Speech'). For this sentence we have 6 Parts of Speech: 'NNS' – Noun Plural, 'IN' – Preposition, 'DT' – Determiner, 'NN' – Noun Singular, 'VBP' – Verb Non-3rd Person Singular Present and 'RB' – Abverb. (Repeat of figure in Introduction).

word 'entertaining' in the sentence 'The Prime Minister was entertaining guests last night' is a verb, but in the sentence 'The Prime Minister is an entertaining woman' the word 'entertaining' is an adjective. As such, a successful POS tagger has to take into account the context surrounding the target word.

There is ongoing debate as to the correct level of abstraction for a Part of Speech, reflecting the ambiguity of the task. The original corpus for Part of Speech Tagging is the Brown Corpus (by W. N. Francis and Kuera (1979), Greene (1981)) created in 1964. In subsequent years many other corpora were compiled, each with a different Part of Speech tagging scheme, for example the LOB: Lancaster-Oslo/Bergen corpus (Atwell (1982)), SEC: Spoken English Corpus (Taylor (1988)) , PoW: Polytechnic of Wales (Souter (1989)) corpus and UPenn: University of Pennsylvania (Santorini (1990)) corpus. These competing tagging schemes reflect the ongoing debate about the correct level of abstraction for Parts of Speech. Our project uses the UPenn tag set as this has emerged as the benchmark for comparing algorithmic performance (the complete tag set for Part of Speech can be found in Appendix A).

2.5.2 Metrics & State of the Art Table

The performance of Part of Speech taggers are measured using simple accuracy - the proportion of words correctly predicted. The benchmark Part of Speech corpus is the Wall Street Journal Corpus. A table of state of the art results can be seen in Table 2.3. As you can see from the table, advances in Part of Speech tagging performance are measured in the sub-percentage point scales - the total improvement over the past 13 years has been 0.54 percentage points.

Table 2.3: POS State of the Art Results

	Percent Accuracy
Toutanova et al. (2003)	97.24 %
Shen et al. (2007)	97.33 %
Sgaard et al. (2011)	97.50 %
Collobert et al. (2011)	97.29 %
Huang et al. (2015)	97.55 %
Ling et al. (2015)	**97.78** %
Yang et al. (2016)	97.55 %
Andor et al. (2016)	97.77 %

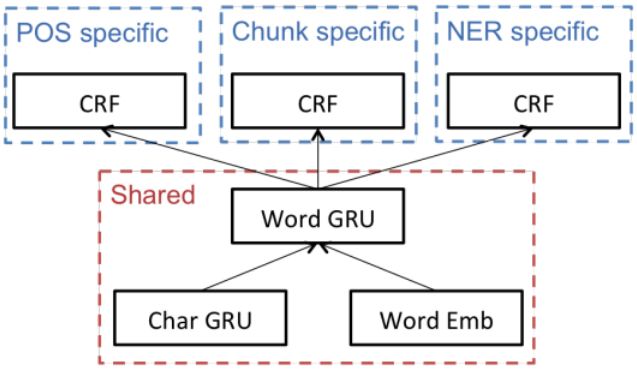
2.5.3 Recent Work in Part of Speech Tagging

Recent work on POS tagging has focused on using variants of Neural Networks called Recurrent Neural Networks ('RNNs') to achieve state of the art performance. The two current best-performing models for Part of Speech tagging (Ling et al. (2015) and Andor et al. (2016)) are task-specific, and so do not incorporate any Chunking data. The third best model, Yang et al. (2016), uses a multi-task network for POS and Chunk and is more similar to our proposed structure. While all three models train on supervised data, they make use of unsupervised representation learning in the form of vector representations of words. We will go into further detail on word embeddings, and the different training regimes in Section 2.3.

The best performing classifier, Ling et al. (2015), uses a variant of Recurrent Neural Networks called Long-Short-Term Memory (LSTM) combined with character-level vector representations to achieve state of the art results. In this model, a novel way of combining character-level vector representations into word-level representations gives a performance boost. It achieves 97.78% accuracy on the Penn Tree Bank corpus.

The second best performing classifier, Andor et al. (2016), uses a neural-network in an incremental transition-based parser Nivre (2006). In this model, a neural network estimates the 'score' of taking an action (predicting the next pos token) given the current state (the previous words and predictions). It achieves 97.77% accuracy on the Penn Tree Bank corpus.

Our model is most similar to the third best performing classifier, Yang et al.



(a) Multi-Task Joint Training

Figure 2.16: Yang et al. (2016) Structure. (Repeat of figure in Introduction).

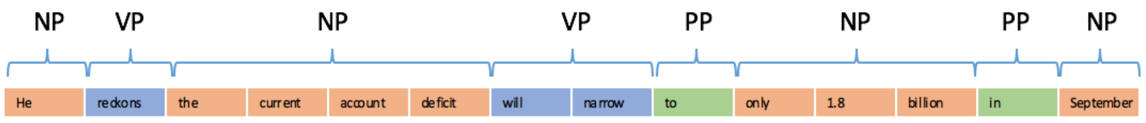


Figure 2.17: Chunk Tagging Example. In the diagram, the words encompassed by each curly bracket are part of the same chunk. We have three types of Chunk in this sentence: 'NP' – Noun Phrase, 'VP' – Verb Phrase and 'PP' – Proposition. (Repeat of figure in Introduction).

(2016), in which POS tagging and Chunk tagging are trained jointly, with a shared component and a task specific component (a diagram of the network can be seen in Figure 2.16). In this model, a novel RNN cell that combines character-level and word-level vector representations called a 'Hierarchical GRU' is used to generate a hidden latent state, with a Conditional Random Field ('CRF') used to create the predictions. This models achieves 97.55 % accuracy on the Penn Tree Bank corpus.

2.6 Chunking

2.6.1 Introduction

The task of Noun-Phrase Chunking ('Chunking') is to find 'chunks' of sentences that correspond to individual noun-phrases. The best way to develop an intuition for what this means is to analyse an example sentence, demonstrated in Figure 2.17 for the sentence *'[He] [reckons] [the current account deficit] [will narrow] [to] [only 1.8 billion] [in] [September]'*.

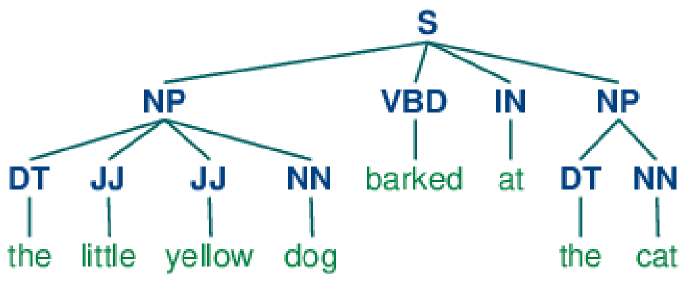


Figure 2.18: POS and Chunk Relationship Example (Bird and Klein (2009)). In this figure, the chunks are at the first level in the tree (e.g. 'NP'), with the POS on the second level (e.g. 'DT'). (Repeat of figure in Introduction).

Using the terminology of the original Chunking paper (Abney (1991)), a chunk consists of a 'content' word at the root (noun), and surrounding 'function' words. For example, in the chunk 'the current account deficit' contained in the example sentence, the 'content' would be the 'deficit' with function words being 'the' and 'current account'. These chunks form the basis for building a tree of the individual noun-phrases, and ultimately a parse tree of the sentence.

The relationship between Chunking and Part of Speech tagging is explicit. In the Chunking introductory paper, 'Parsing by Chunks' - (Abney (1991)), Abney describes a three stage process by which you might parse a sentence. First, you read in the words and assign a Part of Speech Tag, you then use a 'Chunker' to group these words together into chunks depending on the context and the Parts of Speech, and finally you build a parse tree on top of the chunks. In Figure 2.18 an example sentence that has been parsed explicitly using Parts of Speech and Chunking can be seen.

This relationship is a key motivation for using POS and Chunking as two tasks in a multi-task learning architecture - there is an explicit rationale that suggests that information from a POS prediction should improve the performance of your Chunk predictor. In fact, the benchmark data set for Chunking provides Part of Speech tags explicitly.

Chunking as an NLP task was introduced to the academic literature in 1991 by Steven Abney, and has been attempted with Machine Learning methods since 1995 (Ramshaw and Marcus (1995)). In 2000, the SIGNLL Conference on Com-

Table 2.4: Chunk State of the Art Results

	$F_{\beta=1}$ Score
Kudo and Matsumoto (2001)	93.91 %
Shen and Sarkar (2005)	94.01 %
Sun et al. (2008)	94.34 %
Collobert et al. (2011)	94.32 %
Huang et al. (2015)	94.46 %
Yang et al. (2016)	95.41 %

putational Natural Language Learning ('CoNLL') developed a corpus of POS and Chunk-tagged words derived from the same Wall Street Journal text as the Penn Tree Bank (Tjong Kim Sang and Buchholz (2000)). This corpus has become the benchmark for chunk-tagging algorithms, and so is used in our experimental results.

2.6.2 Metrics & State of the Art Table

Chunking performance is measured using the $F_{\beta=1}$ score. The $F_{\beta=1}$ score is the harmonic mean between the precision (the proportion of predicted chunks that are correct) and the recall (the proportion of chunks that were predicted), where β controls the weighting.

$$F_{\beta=1} = 2 * \frac{precision\ *\ recall}{precision\ +\ recall}$$

The $F_{\beta=1}$ score is evaluated on the chunk level rather than the token level. The key benchmark data set is the CoNLL 2000 shared task (a subsection of the Wall Stree Journal tokens). A list of state of the art results can be seen in Table 2.4. Again, not that advances over the past 15 years of Chunking have only led to a 1.5 percentage point increase in F1 score.

2.6.3 Recent Work in Chunk Tagging

State-of-the-art performance in chunk tagging, like POS tagging, is dominated by neural models. In particular, the current state of the art is the multi-task architecture of Yang et al. (2016) described in Section 2.5.3 using a variation of the Recurrent Neural Net. This model achieves 95.41% F1 score. The second best result in Chunk tagging is Huang et al. (2015), a very similar architecture to the state of the art, except that it is task-specific, uses LSTMs rather than Hierarchical GRUs and uses hand-crafted task-specific features. This model achieves 94.46% F1 score. Both use unsupervised representation learning in the form of vector representations of words.

One key point is that the multi-task architecture exploited in Yang et al. (2016) does not explicitly represent the hierarchical relationship of the tasks discussed in the previous section. In this model, the information learnt by the POS CRF layer in Figure 2.16 can not be used by the chunk-specific CRF layer. This means that a portion of the relevant information for Chunking is not available.

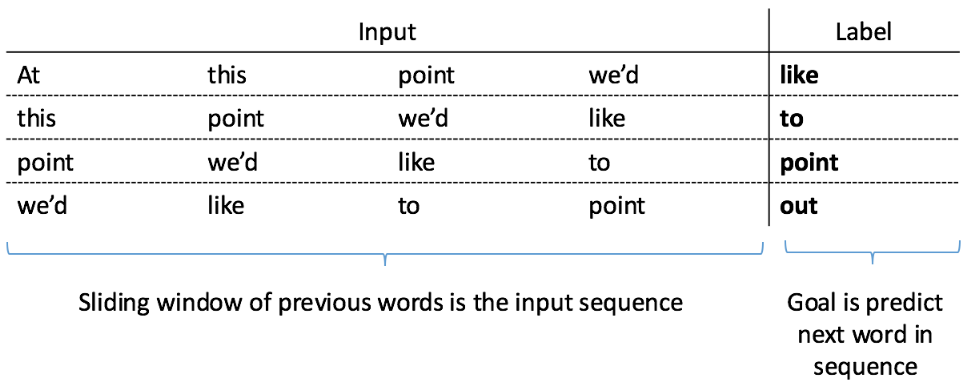
2.7 Language Modeling

2.7.1 Introduction

In machine translation or speech recognition we often want to find a way of evaluating a word-by-word prediction on a sentence level, or, if we are building a piece predictive text software we want to find a way for predicting the next word in a sentence. Both of these tasks are referred to as 'Statistical Language Modeling'. It is considered one of the essential tasks in Natural Language Processing.

As hinted at, Statistical Language Modeling commonly refers to two tasks: the task of assigning a probability to a sequence of words $p(W) = p(w_1, w_2, w_3, w_4)$, and the task of estimating probability of the next word in a sequence, conditioned on the previous words $p(w_5|w_4, w_3, w_2, w_1)$. We can see that these tasks are intimately related by using bayes' rule to decompose the joint probability of a sequence: $p(W) = p(w_1, w_2, w_3, w_4) = p(w_1)p(w_2|w_1)p(w_3|w_2, w_1)p(w_4|w_3, w_2, w_1)$ – in fact, the conditional task is normally used as a way of generating a sentence W with a high joint probability. Figure 2.19 is an example of the conditional language modelling task.

Evaluating $\max\limits_{w_1, w_2, \ldots, w_n} p(w_1, w_2, \ldots, w_n)$ directly is in general computationally intractable. Consider that the number of values each word could take is the size of the language vocabulary (perhaps millions of words) and that the sentence lengths can be hundreds of words long. Without any prior knowledge about the factorisation



Input				Label
At	this	point	we'd	**like**
this	point	we'd	like	**to**
point	we'd	like	to	**point**
we'd	like	to	point	**out**

Sliding window of previous words is the input sequence
Goal is predict next word in sequence

Figure 2.19: Language Modeling Example. (Repeat of figure in Introduction)

of the joint probability distribution, the best running time for the computation is exponential.

Language Modeling is the final task in our task hierarchy. To see why this is the case consider again the parse tree in Figure 2.18. Here you can see that knowledge of the incomplete parse tree, particularly where a noun-phrase ends and another begins, can be helpfully used to inform the probability distribution of the following word. For example, knowing that 'dog' is the end of the noun-phrase that describes the little dog gives you a clue that the next word may be part of the a verb phrase, for example 'barked'.

Language Modelling, unlike POS and Chunk Tagging, does not require labelling and so can be considered a form of unsupervised learning. This means that the corpora used for language modeling are orders of magnitude larger than those used for POS and Chunk tagging. Corpora used for language modeling include the Penn Tree Bank (Taylor et al. (2003)) and the Thomson Reuters News Corpus (Lewis et al. (2004)), each an order of magnitude larger than the CoNLL-2000 POS and Chunk tagging corpus.

In our experiments, Language Modeling is used solely as an unsupervised auxiliary task for the primary tasks of POS tagging and Chunking, and therefore we do not present results of the Language Model to be evaluated against existing benchmarks.

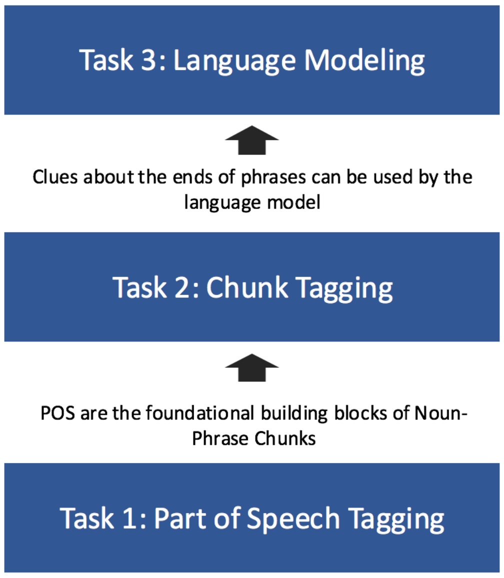

Task 3: Language Modeling
Clues about the ends of phrases can be used by the language model
Task 2: Chunk Tagging
POS are the foundational building blocks of Noun-Phrase Chunks
Task 1: Part of Speech Tagging

Figure 2.20: Task Hierarchy Rationale. Note that while at inference time information only flows up the hierarchy, during training senior tasks in the hierarchy can improve performance of junior tasks through the shared layer. (Repeat of figure from Introduction).

2.8 Task Hierarchy Summary

Now that each of the three tasks (POS tagging, Chunk tagging and Language Modeling) have been introduced, we'll restate the argument for treating them as a task hierarchy[2]. First, the formulation of Chunk Tagging explicitly makes use of POS tags as the foundational information from which to build a noun-phrase chunk, so these two tasks are obvious candidates for tasks 1 and 2. Second, the noun-phrase structure of a sentence gives information to the language model about the probability distribution of the next word and so is a strong candidate for the third task. A schematic of the task hierarchy can be found in Figure 2.20.

[2]For the sake of clarity, a task hierarchy is a group of tasks where the outputs of tasks lower down the hierarchy provide useful information for tasks higher up in the hierarchy.

Chapter 3

Our Contribution

The approaches discussed in the previous Chapter only implicitly represent the relationship between POS tagging, Chunking and Language Modeling. Most recognise the impact of language modeling (or related tasks) on POS and Chunking through the use of Word Embeddings, but only one trains POS and Chunk jointly.

As argued in the introduction, POS, Chunking and Language Modeling all form part of an example linguistic hierarchy, and that this hierarchy could be embedded in the model architecture. Our contribution is to show through this example that exploiting hierarchical structure in linguistic tasks can improve performance of neural network algorithms.

Our contribution to the current approaches for POS and Chunk tagging are then twofold:

1. We explicitly create a hierarchical structure for the POS, Chunking and Language Modelling that ensures information from lower tasks in the hierarchy can be used by those higher up; and

2. Second, by incorporating the Language Model we can trained unsupervised representations that improve both the POS and Chunking performance.

Figure 3.1 is a diagram of our model structure. The following two subsections will expand on these contributions in more detail.

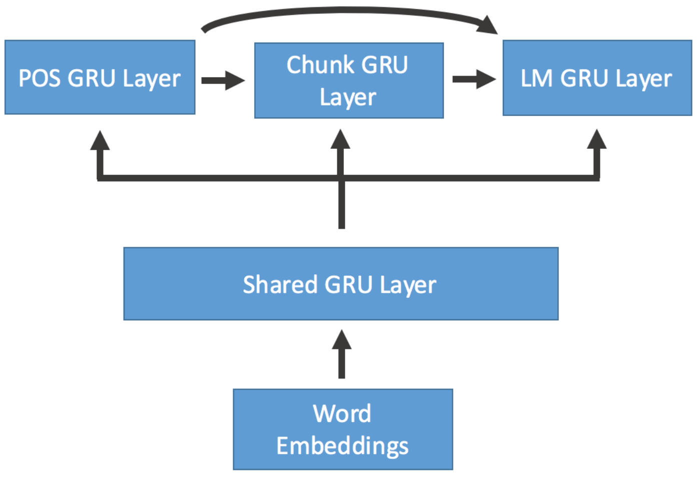


Figure 3.1: Our Model

3.1 Explicit Task Hierarchy

Our model embeds an explicit hierarchy in the multi-task network by feeding the outputs from each task specific layer into the inputs of the tasks higher up the hierarchy.

Concretely, the output of the POS GRU layer (the base task in the hierarchy) is a probability distribution over the POS tags, and is represented by a vector $\mathbf{P}^T$. This vector is multiplied with a matrix $\mathbf{W_{POS}}$ of where each row corresponds to a vector representation of a POS tag. The result of the calculation $\mathbf{P}^T\mathbf{W_{POS}}$ is then concatenated with the hidden state of the shared layer to be fed into the Chunk-specific layer - $concatenate(\mathbf{POSVector}, \mathbf{HiddenState})$. A similar process is conducted for Chunk tagging, with both the POS vector and Chunk vector representations fed into the Language Model. A diagram of this structure for POS and Chunk can be seen in Figure 3.2.

This type of connection, where we learn a vector representation of POS tags and Chunk tags, is motivated by two factors. The first is that, by representing POS tags as vectors we have the opportunity to learn regularities within POS tags - for example, to find which POS tags are used in similar contexts - and to analyse that qualitatively. The second is that we are able to give this richer, continuous representation of the POS output to the Chunking layer.

As we've previously discussed, given the explicit hierarchical relationship be-

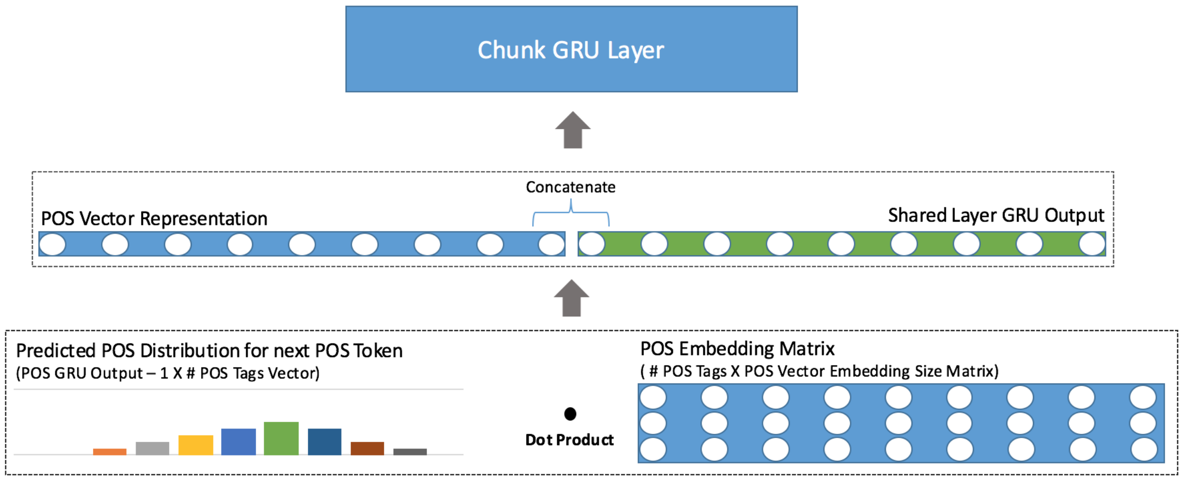

Chunk GRU Layer
Concatenate
POS Vector Representation
Shared Layer GRU Output
Predicted POS Distribution for next POS Token
(POS GRU Output – 1 X # POS Tags Vector)
Dot Product
POS Embedding Matrix
(# POS Tags X POS Vector Embedding Size Matrix)

Figure 3.2: Hierarchical Link Example

tween POS and Chunking, and the implicit hierarchical relationship between POS, Chunking and Language Modeling, explicit connections between tasks should improve the performance of Chunking and Language Modelling - tasks 2 and 3 in the hierarchy respectively.

3.2 Semi-Supervised Learning

Semi-supervised learning describes a family of techniques in which an unsupervised task (where there are typically many data) is optimised to give a set of parameters which improve the performance of a related supervised task. These techniques are particularly useful in situations where there labelling data is particularly costly. A full definition of semi-supervised learning can be found in Section 2.1.

Our model employs semi-supervised learning by alternately optimising the joint loss of POS, Chunking and Language Modeling on a supervised corpus, and then optimising just the Language Modeling loss on an unsupervised corpus. Since there is an explicit shared representation at the bottom layer of our model, and the outputs of Chunk and POS and directly connected to the Language Modelling specific layer, the error from the unsupervised task is back propagated through the the weights of the Chunk and POS layers. In the semi-supervised learning terminology of Ando and Zhang (2005) the unsupervised Language Modeling task is the ‘auxiliary’ task, and the supervised Chunking and POS tagging are ‘target’ tasks.

Task	Unsupervised/Supervised
POS Tagging	Supervised
Chunking	Supervised
Language Modeling	Unsupervised

Figure 3.3: Table of Supervised and Unsupervised Tasks

We now present the intuition for why explicit hierarchical connections between tasks should mean that our model benefits more from the semi-supervised training schedule. One of the challenges in machine learning is over-fitting - this is where the model 'fits' too closely to the probability distribution of the training set, rather than the 'underlying' probability distribution of the task in general. In our model, the language model makes use of the POS and Chunk predictions during the unsupervised-only training, where the data have a different probability distribution to the training set. This acts as a penalty to attempts by the POS and Chunk network to memorise the training set, since those predictions would adversely affect the unsupervised task.

Chapter 4

Data Sets

Our experiments seek to demonstrate that our contributions improve performance in two domains: Newspaper Text and Biomedical Text. As our experiments require a large unsupervised corpus for each of the two domains as well as a supervised corpus, we require 4 data sets in total. A summary of the data sets we use can be seen in Table 4.1.

4.1 Newspaper Text

4.1.1 CoNLL 2000 Chunking Task

In 2000, the SIGNLL Conference on Computational Natural Language Learning (‘CoNLL’) published a shared task for predicting the chunks of Newspaper text. This has emerged as the benchmark for evaluating Chunking performance. However, in our experiment we use it both for POS tagging and Chunking.

The CoNLL 2000 Chunking task contains three columns: the first column is the word token, the second column is a POS token predicted by the Brill tagger (Brill (1992)), and the final column is the Noun-Phrase Chunk (an example can be seen in Table 4.2). Note that the Chunk tag contains a prefix (‘B’ for beginning, ‘I’ for intermediate) which is not evaluated at test time.

	Labelled	**Unlabelled**
Newspaper Text	CoNLL 2000 Chunking Task (259,104 tokens)	Penn Tree Bank Word Tokens (929,589 tokens)
Biomedical Text	Genia Dataset (410,026 tokens)	Pubmed Abstract Scrape (2,990,385 tokens)

Table 4.1: Summary of Data Sets By Domain

Word Token	POS Token	Chunk Token
He	PRP	B-NP
reckons	VBZ	B-VP
the	DT	B-NP
current	JJ	I-NP

Table 4.2: Example Data

Table 4.3: Chunk State of the Art Results

	$F_{\beta=1}$ Score
Kudo and Matsumoto (2001)	93.91 %
Shen and Sarkar (2005)	94.01 %
Sun et al. (2008)	94.34 %
Collobert et al. (2011)	94.32 %
Huang et al. (2015)	94.46 %
Yang et al. (2016)	95.41 %

The word tokens for training are sections 15-18 from the Wall Street Corpus, the word tokens for testing is section 20.

The CoNLL 2000 Chunking Task is a recognised benchmark for Chunking only, and not for POS (since the POS tokens are derived from the Brill Tagger). A table of state of the art results for Chunking can be found in Table 4.3.

4.1.2 Penn Tree Bank Word Tokens

The Penn Tree Bank Word Tokens corpus comprises c. 1 million word tokens taken from the Wall Street Journal. The full Penn Tree Bank includes the Parts of Speech as well as the parse trees, but for our purposes we just use the raw word tokens.

The data is preprocessed to be in the same format as the CoNLL 2000 chunking task, but with a ‘null’ token for POS and Chunk.

4.2 Biomedical Text

4.2.1 The Genia Corpus

The GENIA corpus is a collection of Medline abstracts (intended to represent the literature of molecular biology) which has been annotated with POS and Chunk tags (among others).

It was shown in Tsuruoka et al. (2005) that taggers designed for Newspaper text can perform poorly on Biomedical text. This is because biomedical text has a lot of specialist vocabulary, often with a different writing style to that found in normal text. For this reason, we use the Genia corpus to test the robustness of our taggers.

The current state of the art for the Genia corpus POS tagging is 98.49 % (Tsuruoka et al. (2005)). The performance on chunking is not reported as a benchmark.

4.2.2 Word Tokens from Pubmed Abstracts

There are no standard corpora for unsupervised learning in biomedical text. As such, we had to create our own. To do so, we downloaded the first 3m tokens of the abstracts on the Pubmed [1] database matching the keywords used for the original Genia Corpus.

As for the Penn Tree Bank Word Tokens, we pre-process the data into the same format as CoNLL 2000 but with null tokens instead of POS and Chunk.

[1] http://www.ncbi.nlm.nih.gov/pubmed

Chapter 5

Experimental Results

In this Chapter we examine the architecture impact of our contributions on the performance of both Part of Speech tagging and Chunking. First, we create baseline results using a simple multi-task architecture for POS, Chunking and Language Modeling. We highlight here that we are unable to replicate the results achieved by Huang et al. (2015) with a very similar architecture.

We then introduce our contributions - the explicit task hierarchy and semi-supervised learning and demonstrate the situations in which performance improvement can be seen under a base case hyper-parameter setting.

Finally, we run through a selection of the hyper-parameters in the model and demonstrate the performance sensitivity for each hyper-parameter. We then present results with our best hyper-parameter setting and compare them to state of the art.

5.1 Baseline Model

Our Baseline Model is a basic multi-task architecture, based upon the the model in Yang et al. (2016). In our architecture, we have one shared GRU layer of size 256, then each task has an individual GRU layer, again of size 256. Finally, we project this onto a tag vector and take the Softmax Cross-Entropy loss. A diagram of the baseline model can be seen in Figure 5.1.

We train this model jointly for 100 epochs. We use Genia 300-dimensional word embeddings following the conclusions of section 2.3. We have batches of size 64 x 64, with window sequences.

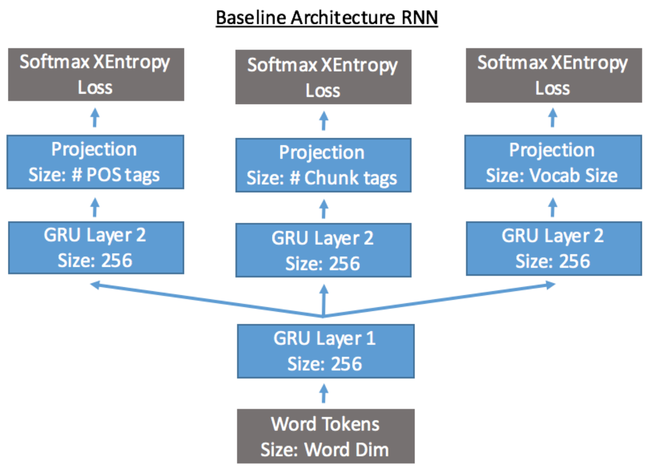


Figure 5.1: Baseline Architecture

	Test $F_{\beta=1}$ (CoNLL)	Test $F_{\beta=1}$ (Genia)
Baseline Model	91.95 %	85.3 %

Table 5.1: Baseline Results - Chunk

We run the baseline model on both the CoNLL 2000 and Genia data sets. Baseline results for Chunking can be seen in Table 5.1, and baseline results for POS tagging can be seen in Table 5.2.

5.2 Our Contributions

In this section we discuss the improvements in performance attributable to our contributions. We show that building a hierarchical model not only improves the Chunk F1 score of our baseline model by 0.3 % on ConLL 2000, but provides a new way to visualise the learnt representations of POS and Chunking. We then move on to discuss semi-supervised learning and show that, while gains can be seen when the model is initialised with pre-trained word embeddings, the real gains are found when the model is not pre-initialised.

	Test Accuracy (CoNLL)	Test Accuracy (Genia)
Baseline Model	96.2 %	97.3 %

Table 5.2: Baseline Results - POS

	Test $F_{\beta=1}$ (CoNLL)	Test $F_{\beta=1}$ (Genia)
With Connection	**92.24** %	**85.82** %
Without Connection	91.95 %	85.79 %

Table 5.3: Hierarchy Test Results - Chunk

	Validation CoNLL	Test Genia
With Connection	**96.4** %	**97.8** %
Without Connection	96.2 %	97.6 %

Table 5.4: Hierarchy Test Results - POS

5.2.1 Hierarchical Model

Our first contribution is to explicitly represent the task hierarchy in the structure of the neural network. In our experiments, we test our baseline model both with and without the connections between tasks, and find an improvement in performance. The results of this experiment can be found in Tables 5.3 and 5.4.

One of the additional advantages of such an architectures is that we learn a dense-vector representation of POS and Chunk tags, similar to the word vector representations discussed in Section 2.3. In Figure 5.2 we present the TSNE visualisation (Maaten and Hinton (2008)), where we can see that the neural network has encoded the difference between tags for the beginning of chunks (prefixed with a ‘b’) and tags for intermediate chunks (prefixed with an ‘i’).

5.2.2 Semi-Supervised Learning

Our second contribution is achieving semi-supervised learning by training our language model on an unlabelled corpora. In our semi-supervised training schedule we mix unlabelled and labelled batches probabilistically according to some value $0 < \alpha < 1$. We only consider a training epoch complete once all of the PTB and CoNLL data has been seen by the model.

The intuition behind this mix percentage is that, if the model sees too many unsupervised tasks in a row, it will start to ‘forget’ the Chunk-specific and POS-specific layers. However, if the model sees to many supervised tasks in a row some of the benefit of the unsupervised task will be lost.The results of the experiment with this parameter can be seen in Tables 5.5 and 5.6.

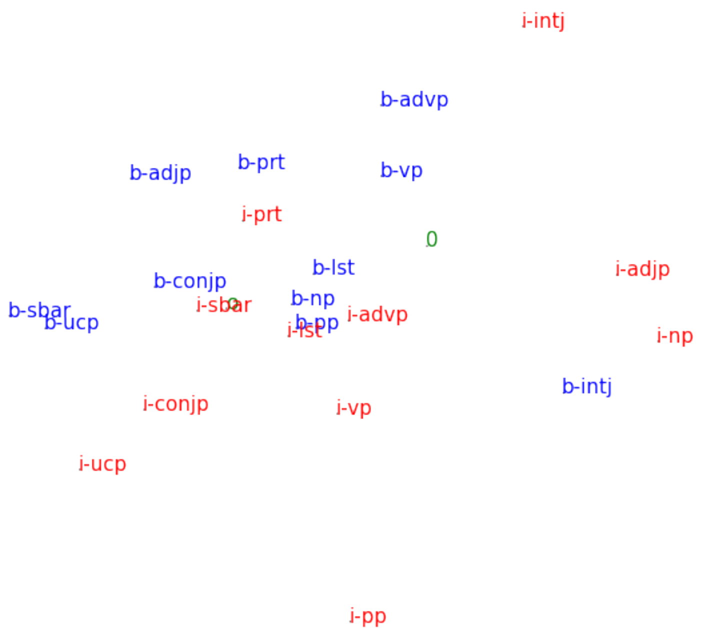


Figure 5.2: TSNE Visualisation of Chunk Tags. Here we can see that the vector representation has encoded the difference between a beginning of chunk and the middle/end of a chunk.

	Validation $F_{\beta=1}$ (CoNLL)	Validation $F_{\beta=1}$ (Genia)
40 %	92.6% (-0.2 %)	86.1 % (+0.6 %)
50 %	92.8 % (+0.0 %)	86.4 % (+0.9 %)
60 %	92.8 % (+0.0 %)	86.3 % (+0.8 %)
70 %	**93.0** % (+0.2%)	**86.4** % (+0.9 %)
100 %	92.8 %	85.5 %

Table 5.5: Mix Percentage - Chunk

In Tables 5.5 and 5.6 we present the results of varying the mix percentage on the validation set, while keeping the total amount of labelled and unlabelled data constant. As we can see, there is a minor impact associated with varying the mix percentage, with the best results being achieved in the 60 % to 70 % range. While the overall impact on performance of the unsupervised data in this example was small, there was a significant improvement in the speed of convergence as can be seen in Figure 5.3.

The language modeling task is similar in spirit to the tasks used to generate

	Validation CoNLL	Validation Genia
40 %	95.8 % (- 0.6 %)	97.3 % (+ 0.0 %)
50 %	96.3 % (- 0.1 %)	97.3 % (+ 0.0 %)
60 %	96.3 % (- 0.1 %)	**97.6** % (+ 0.3 %)
70 %	**96.4** % (+ 0.0 %)	97.5 % (+ 0.2 %)
100 %	96.4 %	97.3 %

Table 5.6: Mix Percentage - POS

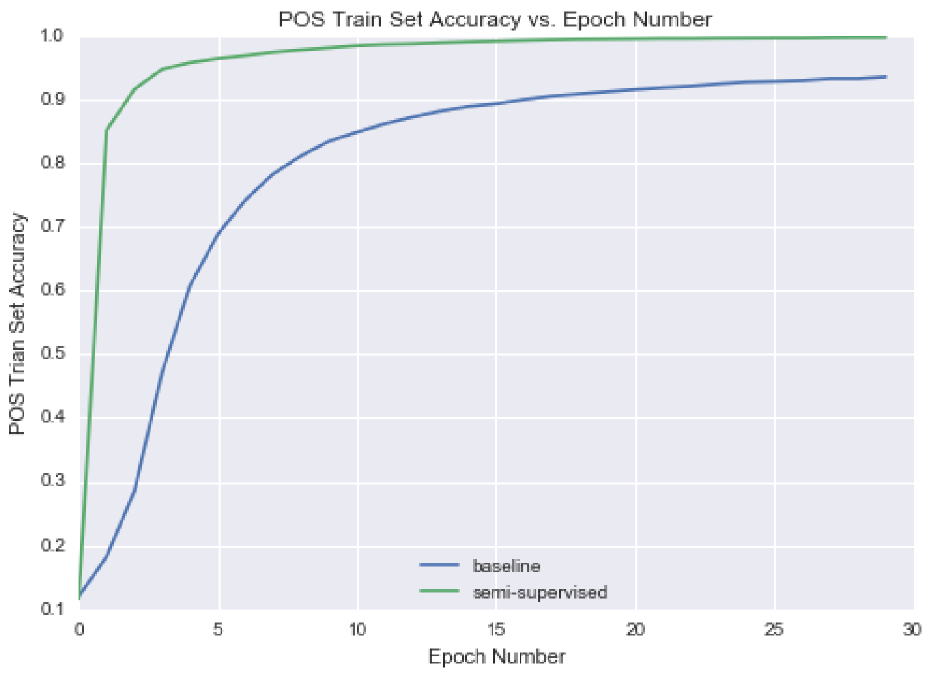


Figure 5.3: POS Accuracy on Training Set by Epoch. Here we can see that, by incorporating unlabelled data, we converge more swiftly.

the word embeddings discussed in Section 2.3, and for this reason at least some of the benefit of using the unlabelled task is already provided by initialising our embeddings matrix with pre-trained vectors.

For this reason, we demonstrate the impact of using semi-supervised learning in a situations without any pre-trained word embeddings (Tables 5.7 and 5.8). Here, we can see more than a 2 percentage point increase in performance on the Chunk F1 score for CoNLL 2000 vs. no unsupervised data, a performance improvement comparable to the increase seen by using pre-trained word embeddings. This is a promising result, since it hints that for tasks in languages where there are too few data to train word embeddings (or where the cost is too great), a comparable

	No Pre-Trained Embeddings		With Pre-Trained Word Embeddings	
	Test $F_{\beta=1}$ (CoNLL)	Test $F_{\beta=1}$ (Genia)	Test $F_{\beta=1}$ (CoNLL)	Test $F_{\beta=1}$ (Genia)
Baseline Model	86.2 %	85.3 %	91.9 %	85.8 %
Semi-Supervised	**88.3** % (+ 2.1 %)	**85.7** % (+ 0.4 %)	**92.3** % (+ 0.4 %)	**85.8** % (+ 0.0 %)

Table 5.7: No Embeddings Semi-Supervised Summary Results - Chunk

	No Pre-Trained Embeddings		With Pre-Trained Word Embeddings	
	Test $F_{\beta=1}$ (CoNLL)	Test $F_{\beta=1}$ (Genia)	Test $F_{\beta=1}$ (CoNLL)	Test $F_{\beta=1}$ (Genia)
Baseline Model	92.4 %	97.5 %	96.2 %	97.3 %
Semi-Supervised	**94.2** % (+ 1.8 %)	**97.6** % (+ 0.1 %)	**96.2** % (+ 0.0 %)	**97.6** % (+ 0.3 %)

Table 5.8: Semi-Supervised Summary Results - POS

performance increase can be attained by adding an auxiliary unsupervised tasks.

5.3 Hyper-parameter Settings

Hyper-parameter optimisation can have an large impact on machine learning models, in some cases improving performance by 3-4 %. For this reason, this section is devoted to describing some of the most important hyper-parameters in our model and investigating the sensitivity of the performance metrics to each hyper-parameter.

The parameter settings are chosen by 1-fold cross-validation. This is where we hold back a portion of the training data to 'test' our hyper parameters on (known as the 'Validation Set'). We then optimise our model for the validation set, and then evaluate the overall performance on the test set. With machine learning algorithms that are not so expensive to train as Neural Nets, it is common to do 'x-fold' cross-validation in which the experiment is run x times with a different training set and hyper-parameters are chosen by majority.

It's important to note that flexing only one hyper-parameter at a time, and then combining them into a 'best setting' will in general not provide the best combined hyper-parameter combination. This is because the settings of individual hyper-parameters have an effect on the rest of the model. For example, if you flex the width and depth of a neural network this increase the sum of all the weights, meaning that any regularisation that acts upon this total sum may need to change.

However, conducting a grid search on all hyper-parameters is not something that is computationally reasonable to do. For this reason, and for exposition reasons,

	Sequence Length					
	Validation $F_{\beta=1}$ (CoNLL)			Validation $F_{\beta=1}$ (Genia)		
	16	32	64	16	32	64
16	91.44 %	92.38 %	**92.89** %	84.99 %	85.82 %	**86.27** %
32	91.63 %	92.24 %	92.34 %	85.17 %	85.49 %	85.84 %
64	91.29 %	92.24 %	92.73 %	84.89 %	85.45 %	86.20 %

Table 5.9: Batch Size Results - Chunk

we are investigating each hyper-parameter separately. Once we have investigated the importance of the hyper-parameters, we will fix them in the 'best setting' and present our final results.

The baseline model for these hyper-parameter experiments includes connections between tasks and unsupervised learning with the mix percentage at 50 %.

5.3.1 Batch Size

In Stochastic Gradient Descent the data is fed in batches. The size of these batches will have an effect on the convergence rate and the performance of the model. In Table 5.9 we find that the larger sequence sizes boost performance, but that larger batch sizes have a very slight detrimental performance. This detrimental performance may be too small to consider statistically significant.

5.3.2 Word Embeddings

Word embeddings are the most common form of semi-supervised learning in Natural Language Processing. In this subsection we compare fine-tuning no pre-trained word embeddings, SENNA, GloVe 50 dimensional, GloVe 100 dimensional and GloVe 300 dimensional word embeddings without any projections and see that SENNA 50 dimensional embeddings perform best. Results can be seen in Tables 5.10 and 5.11.

This is surprising result, since GloVe embeddings are trained on a larger corpus, are global vectors, and have performed better on semantic similarity tasks. One potential explanation for these results it that the training task for SENNA word embeddings is based upon a multi-task sequence tagging architecture. For more information, please see Section 2.3.

	Validation $F_{\beta=1}$ (CoNLL)	Validation $F_{\beta=1}$ (Genia)
No Word Embeddings	88.3 %	86.1 %
SENNA	**93.0** % (+ 4.7 %)	85.8 % (- 0.3 %)
Glove 50	91.5 % (+ 3.2 %)	85.3 % (- 0.8 %)
Glove 100	92.4 % (+ 4.1 %)	85.6 % (- 0.5 %)
Glove 200	92.8 % (+ 4.5 %)	**86.3** % (+ 0.2 %)
Glove 300	92.8 % (+ 4.5 %)	86.1 % (+ 0.0 %)

Table 5.10: Word Embedding Results - Chunk

	Validation CoNLL	Validation Genia
No Word Embeddings	94.2 %	97.6 %
SENNA	**96.6** % (+2.4 %)	**97.7** % (+0.1 %)
Glove 50	96.0 % (+1.8 %)	97.5 % (-0.1 %)
Glove 100	96.5 % (+2.3 %)	97.6 % (+0.0 %)
Glove 200	96.4% (+2.2 %)	97.6 % (+0.0 %)
Glove 300	96.1 % (+1.9 %)	97.5 % (-0.1 %)

Table 5.11: Word Embedding Results - POS

5.3.3 Optimiser

As discussed in Section 2.2.4, the choice of optimisation technique can have a significant effect on the final accuracy. The two best performing algorithms for Chunking use Adagrad with an annealing learning rate. In Tables 5.12 and 5.13 we compare this optimiser to Adam, and find that Adam has the better performance.

5.3.4 Dropout

Dropout is a method to reduce overfitting in neural networks (Srivastava et al. (2014)). It operates by randomly removing connections in the neural network, helping to reduce co-adaptation of artificial neurons. The probability of removal is a key

	Validation $F_{\beta=1}$ (CoNLL)	Validation $F_{\beta=1}$ (Genia)
Adagrad	89.6 %	85.4 %
Adam	**92.5** % (+ 2.9 %)	**86.3** % (+ 0.9 %)

Table 5.12: Optimisation Results - Adam and Adagrad

	Validation CoNLL	Validation Genia
Adagrad	94.5 %	96.2 %
Adam	**96.2** % (+ 1.8 %)	**97.5** % (+ 1.3 %)

Table 5.13: Optimisation Results - Adam and Adagrad.

	Validation $F_{\beta=1}$ (CoNLL)	Validation $F_{\beta=1}$ (Genia)
0.4	92.75 %	85.85 %
0.5	**92.94** %	**86.02** %
0.6	95.72 %	85.91 %

Table 5.14: Dropout Results - Chunk

	Validation CoNLL	Validation Genia
0.4	96.2 %	97.5 %
0.5	**96.0** %	**97.3** %
0.6	96.0 %	97.3 %

Table 5.15: Dropout Results - POS

regularisation parameter in the model, tested in Tables 5.14 and 5.15. Here we find that a 50 % chance of removal is optimum.

5.4 Best Model

In this section we present the model performance for Genia and CoNLL using the best hyper-parameter settings (Tables 5.16 and 5.17). While these results are not state of the art, they demonstrate that good performance can be achieved using our contributions. Further work will investigate how to make additional architecture changes to reach state of the art performance.

5.5 Qualitative Analysis of Error Sentences

Insight into our model can be generated from sentences where predictions are incorrect. For example, in Figure 5.4 we see a sentence from the CoNLL Newspaper data set which our model has failed to predict accurately. In this case, the word 'short' is part of the Noun-Phrase 'a short attention span', but our model predicts as

	Test $F_{\beta=1}$ (CoNLL)	Test $F_{\beta=1}$ (Genia)
Baseline Model	92.7 %	85.9 %

Table 5.16: Optimised Results - Chunk

	Test Accuracy (CoNLL)	Test Accuracy (Genia)
Baseline Model	96.4 %	97.7 %

Table 5.17: Optimised Results - POS

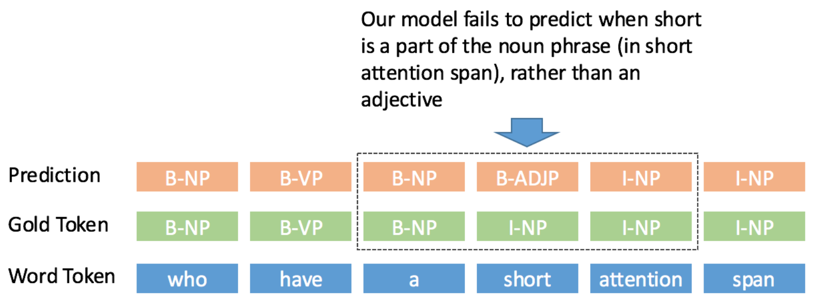

Our model fails to predict when short is a part of the noun phrase (in short attention span), rather than an adjective
Prediction
B-NP
B-VP
B-NP
B-ADJP
I-NP
I-NP
Gold Token
B-NP
B-VP
B-NP
I-NP
I-NP
I-NP
Word Token
who
have
a
short
attention
span

Figure 5.4: Example of Sentence our Model gets Wrong in Newspaper Text

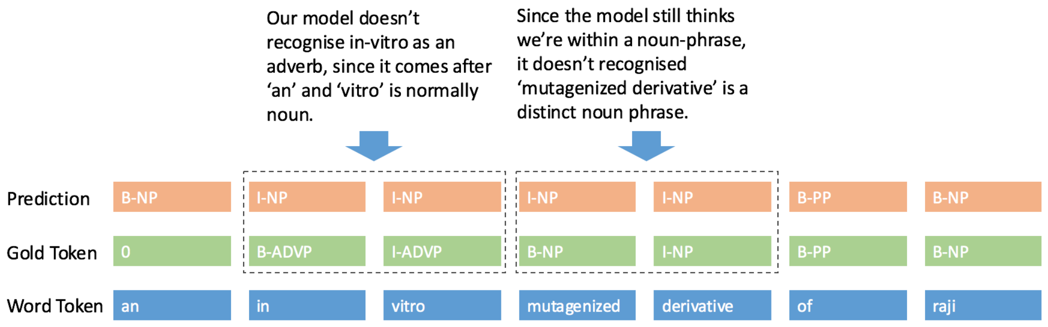

Our model doesn't recognise in-vitro as an adverb, since it comes after 'an' and 'vitro' is normally noun.
Since the model still thinks we're within a noun-phrase, it doesn't recognised 'mutagenized derivative' is a distinct noun phrase.
Prediction
B-NP
I-NP
I-NP
I-NP
I-NP
B-PP
B-NP
Gold Token
0
B-ADVP
I-ADVP
B-NP
I-NP
B-PP
B-NP
Word Token
an
in
vitro
mutagenized
derivative
of
raji

Figure 5.5: Example of Sentence our Model gets Wrong in Biomedical Text

an adjective - presumably because that is the dominant label in the corpus.

We can also look at an example of a sentence that the model predicts incorrectly from the Genia Biomedical data set (Figure 5.5). Here we see the impact of many infrequent words, and their ambiguous usage. For example, 'in vitro' should be regarded as a single adverb chunk, however our model gets this wrong because it requires an unusual use of the word 'in'. These Latin phrases can greatly complicate the Chunking task.

Chapter 6

Conclusions & Suggestions for Further Work

6.1 Summary

This thesis sought to investigate whether linguistically motivated task hierarchies in Machine Learning could improve algorithm performance. In particular, we were interested in demonstrating improvements for particular example - POS, Chunking and Language Modeling. We showed that explicitly representing the task hierarchy can bring small gains, but provide valuable representations of junior tasks. Additionally, we showed an effective way to combine tasks with varying numbers of data through semi-supervised learning.

Concretely, the beginning of this thesis we introduced two questions we hoped to answer. We'll revisit each of those in turn briefly describing the results of our investigation and suggestions for further work.

1. Whether we can explicitly embed hierarchical task structure in neural architectures for Part of Speech ('POS') tagging, Chunk tagging (also known as Noun-Phrase Chunking or Shallow Parsing) and Language Modeling, and what performance this explicit hierarchical structure brings.

We found that explicit embedding of hierarchical task structure can improve performance and offer valuable insight into the vector representation of low-level tasks learnt by the neural network.

Further work would investigate whether (1) this performance improvement can be observed in other putative task hierarchies in Natural Language Processing and other machine learning disciplines; (2) ways to quantify how much transfer there is between each of the tasks.

2. Whether, by incorporating an unsupervised auxiliary task (Language Modelling) to two supervised target tasks (POS tagging and Chunking), we can improve performance by conducting semi-supervised learning.

We found that semi-supervised learning as described can improve performance both on models initialised with pre-trained word embeddings. However, the best results were found in situations with no prior pre-training, where our auxiliary task can achieve much of the unsupervised representation learning achieved with far larger corpora and training resources in of the time and with far fewer data.

6.2 Critique

While our results are promising, there remain a number of outstanding issues with our algorithm. In this section we will discuss three.

The first critique is that task hierarchies only provide very small improvements in performance, and don't seem to hold up well across domains. This, however, should be seen in the context of only a 0.5 percentage point improvement in POS accuracy over the past 13 years.

The second critique is that these models have the requirement to know beforehand which tasks are hierarchically related, rather than this relationship being learnt, which presents a barrier to these models being used more generally. A naive solution, which is to build lateral connections between all tasks, would increase model capacity exponentially but with huge redundancy. At present there doesn't seem to be an obvious manner to compress the information flow or prune the model adaptively (Rusu et al. (2016)) to address redundancy.

The third critique is that our model is still prone to catastrophic forgetting of previously learnt tasks during training of a senior task if there is no reservoir of persistent training data. For example, our attempt to address this by alternating batches requires us to have the POS and Chunk training data to hand. There are not many situations in which this could be the case, and we certainly do not want to have to train every single task when learning a new one. One naive solution - fixing the parameters of previously learnt weights as in Rusu et al. (2016) - means that additional generalisation from exposure to new tasks can't take place, an unsatisfactory result.

6.3 Suggestions For Further Work

Further research would first investigate the outstanding problems mentioned above. Concretely: (1) it would investigate methods for information compression and adaptive pruning of networks so that only utilised capacity in the network was kept, (2) it would investigate methods for avoiding catastrophic forgetting of previously learned tasks, potentially through parameter fixing (albeit with the problems mentioned above), but also through methods such as distillation (Rasmus et al. (2015)).

The final area of research would be on alternative tasks for achieving semi-supervised learning. These might include methods inspired by using Generative Adversarial Networks as an auxiliary task in Computer Vision experiments (Salimans et al. (2016)), or alternative unsupervised tasks in Natural Language Processing.

Appendix A

UPenn Part of Speech Tag Set

Table A.1: UPenn Tag Set

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential there
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	to
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

Bibliography

Abney, S. P.
1991. Parsing by chunks. In *Principle-based parsing*, Pp. 257–278. Springer.

Ando, R. K. and T. Zhang
2005. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6(Nov):1817–1853.

Andor, D., C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and M. Collins
2016. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042*.

Atwell, E.
1982. Ob corpus tagging project: Post-edit handbook. Department of Linguistics and Modern English Language, University of Lancaster.

Barber, D.
2012. *Bayesian Reasoning and Machine Learning*. Cambridge University Press.

Bengio, Y., R. Ducharme, P. Vincent, and C. Jauvin
2003. A neural probabilistic language model. *journal of machine learning research*, 3(Feb):1137–1155.

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

Bird, Steven, E. L. and E. Klein

2009. *Natural Language Processing with Python*. O'Reilly Media Inc.

Blei, D. M., A. Y. Ng, and M. I. Jordan

2003. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Brill, E.

1992. A simple rule-based part of speech tagger. In *Proceedings of the workshop on Speech and Natural Language*, Pp. 112–116. Association for Computational Linguistics.

by W. N. Francis, C. and H. Kuera

1964, 1971, 1979. A Standard Corpus of Present-Day Edited American English, for use with Digital Computers (Brown).

Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio

2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Chung, J., C. Gulcehre, K. Cho, and Y. Bengio

2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Collobert, R., J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa

2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

Cybenko, G.

1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Deerwester, S., S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman

1990. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391.

Duchi, J., E. Hazan, and Y. Singer

2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Evgeniou, T. and M. Pontil

2004. Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, Pp. 109–117. ACM.

Gers, F. A., J. Schmidhuber, and F. Cummins

2000. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471.

Greene, Barbara/Rubin, G.

1981. Automatic grammatical tagging of english. Providence, RI: Department of Linguistics, Brown University.

Harris, Z. S.

1954. Distributional structure. *Word*, 10(2-3):146–162.

Hinton, G. E.

1984. Distributed representations.

Hochreiter, S., Y. Bengio, P. Frasconi, and J. Schmidhuber

. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

Hochreiter, S. and J. Schmidhuber

1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hornik, K., M. Stinchcombe, and H. White

1989. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Huang, Z., W. Xu, and K. Yu

2015. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Karpathy, A.

2015. The unreasonable effectiveness of rnns.

Kingma, D. and J. Ba

2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A., I. Sutskever, and G. E. Hinton

2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, Pp. 1097–1105.

Lewis, D. D., Y. Yang, T. G. Rose, and F. Li

2004. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397.

Ling, W., T. Luís, L. Marujo, R. F. Astudillo, S. Amir, C. Dyer, A. W. Black, and I. Trancoso

2015. Finding function in form: Compositional character models for open vocabulary word representation. *arXiv preprint arXiv:1508.02096*.

Lipton, Z. C., J. Berkowitz, and C. Elkan

2015. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*.

Maaten, L. v. d. and G. Hinton

2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605.

Mikolov, T., K. Chen, G. Corrado, and J. Dean

2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., I. Sutskever, K. Chen, G. Corrado, and J. Dean
2013b. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*.

Morin, F. and Y. Bengio
2005. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, Pp. 246–252. Citeseer.

Nivre, J.
2006. *Inductive dependency parsing*. Springer.

Olah, C.
2015. Calculus on computational graphs: Backpropagation.

Pennington, J., R. Socher, and C. D. Manning
. Glove: Global vectors for word representation.

Ramshaw, L. A. and M. P. Marcus
1995. Text chunking using transformation-based learning. *arXiv preprint cmp-lg/9505040*.

Rasmus, A., M. Berglund, M. Honkala, H. Valpola, and T. Raiko
2015. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, Pp. 3546–3554.

Rosenblatt, F.
1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams
. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

Rusu, A. A., N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell
2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Salimans, T., I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen
2016. Improved techniques for training gans. *arXiv preprint arXiv:1606.03498*.

Santorini, B.
1990. Part-of-speech tagging guidelines for the penn treebank project. technical report ms-cis-9047. University of Pennsylvania: Department of Computer and Information Science.

Souter, C.
1989. A short handbook to the polytechnic of wales corpus. bergen university. Norway: ICAME, The Norwegian Computing Centre for the Humanities.

Srivastava, N., G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov
2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Sutskever, I., J. Martens, G. E. Dahl, and G. E. Hinton
. On the importance of initialization and momentum in deep learning.

Taylor, A., M. Marcus, and B. Santorini
2003. The penn treebank: an overview. In *Treebanks*, Pp. 5–22. Springer.

Taylor, Lolita/Knowles, G.
1988. Manual of information to accompany the sec corpus: The machine readable corpus of spoken english. University of Lancaster: Unit for Computer Research on the English Language.

Tjong Kim Sang, E. F. and S. Buchholz
2000. Introduction to the conll-2000 shared task: Chunking. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*, Pp. 127–132. Association for Computational Linguistics.

Tsuruoka, Y., Y. Tateishi, J.-D. Kim, T. Ohta, J. McNaught, S. Ananiadou, and

J. Tsujii

2005. Developing a robust part-of-speech tagger for biomedical text. In *Panhellenic Conference on Informatics*, Pp. 382–392. Springer.

Werbos, P. J.

1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

Yang, Z., R. Salakhutdinov, and W. Cohen

2016. Multi-task cross-lingual sequence tagging from scratch. *arXiv preprint arXiv:1603.06270*.