View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Edinburgh Research Archive

THE UNIVERSITY *of* EDINBURGH

This thesis has been submitted in fulfilment of the requirements for a postgraduate degree (e.g. PhD, MPhil, DClinPsychol) at the University of Edinburgh. Please note the following terms and conditions of use:

This work is protected by copyright and other intellectual property rights, which are retained by the thesis author, unless otherwise stated.

A copy can be downloaded for personal non-commercial research or study, without prior permission or charge.

This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the author.

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author.

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given.

Natural Language Generation as Neural Sequence Learning and Beyond

Xingxing Zhang

Doctor of Philosophy
School of Informatics
University of Edinburgh
2017

Abstract

Natural Language Generation (NLG) is the task of generating natural language (e.g., English sentences) from machine readable input. In the past few years, deep neural networks have received great attention from the natural language processing community due to impressive performance across different tasks. This thesis addresses NLG problems with deep neural networks from two different modeling views. Under the first view, natural language sentences are modelled as sequences of words, which greatly simplifies their representation and allows us to apply classic sequence modelling neural networks (i.e., recurrent neural networks) to various NLG tasks. Under the second view, natural language sentences are modelled as dependency trees, which are more expressive and allow to capture linguistic generalisations leading to neural models which operate on tree structures.

Specifically, this thesis develops several novel neural models for natural language generation. Contrary to many existing models which aim to generate a single sentence, we propose a novel hierarchical recurrent neural network architecture to represent and generate multiple sentences. Beyond the hierarchical recurrent structure, we also propose a means to model context dynamically during generation. We apply this model to the task of Chinese poetry generation and show that it outperforms competitive poetry generation systems.

Neural based natural language generation models usually work well when there is a lot of training data. When the training data is not sufficient, prior knowledge for the task at hand becomes very important. To this end, we propose a deep reinforcement learning framework to inject prior knowledge into neural based NLG models and apply it to sentence simplification. Experimental results show promising performance using our reinforcement learning framework.

Both poetry generation and sentence simplification are tackled with models following the sequence learning view, where sentences are treated as word sequences. In this thesis, we also explore how to generate natural language sentences as tree structures. We propose a neural model, which combines the advantages of syntactic structure and recurrent neural networks. More concretely, our model defines the probability of a sentence by estimating the generation probability of its dependency tree. At each time step, a node is generated based on the representation of the generated subtree. We show experimentally that this model achieves good performance in language modeling and can also generate dependency trees.

Acknowledgements

Time flies. When I started to write this part, I realized I am very close to the finishing line of my PhD. The four years of PhD study in Edinburgh was a wonderful, happy and rewarding experience.

First and foremost I would like to thank my PhD supervisor, Mirella Lapata. I feel very fortunate to be her student. What I learned most from her is to think big and think critically in research. She is passionate and serious about research, which definitely influenced me a lot. She cares about how research is presented and constantly gave me feedbacks on writing and presentations, which makes me believe that doing research and presenting research properly are equally important. I would also like to thanks her for giving me enough freedom to work on what I am interested and dragging me back to the right track when I was going too wild. During my PhD, I had good times and bad times. She always encourages me and was being supportive. Everytime we were able to turn disappointments into even bigger excitements.

I would also like to thank my second supervisor, Adam Lopez for helpful discussions and brilliant feedback (especially on language modeling and dependency parsing). Thanks to Iain Murray for feedback to my first year review. Many thanks to my thesis examiners: Stephen Clark and Shay Cohen, whose insightful comments during the viva definitely improved this thesis. Many thanks to Bonnie Webber, Frank Keller, Sharon Goldwater, Rico Sennrich, Jianpeng Cheng, Li Dong and other members of ProbModels for helpful advice and comments to my research. Special thanks to my coauthors, Jianpeng Cheng and Liang Lu, from whom I learned so much.

I am thankful to my office mates: Li Dong, Jianpeng Cheng, Nicolas Collignon, Pablo León-Villagrá and John Torr. Thanks for creating such a great research environment and I benefit a lot from the daily discussions on research papers and other staff.

Finally, my deepest thanks to my parents for their love and support for so many years!

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Xingxing Zhang*)

Table of Contents

Chapter 1

Introduction

1.1 Motivation

There is a large amount of information available on the internet and it is still growing every day. Information is rendered in text but also images, video, databases, numbers and so on. To fulfill our information needs, we usually type several keywords to a search engine. Most search engine techniques are based on keyword matching and will often return relevant documents to the query (mostly in text format). Users' needs however are becoming increasingly muti-modal. For example, a user may search for "a cute cat" and he or she expect an image of "a cute cat" rather than a text description[1]. In this case, natural language generation technology can be used to transform images into text, and the generated text can be used for indexing in a search engine.

Besides being used in search engines, natural language generation technology can also be used as an authoring aid. A routine job for weather reporters is to write weather forecasts based on a weather database, which is time consuming and boring. We can use natural language generation methods to produce weather reports automatically (Goldberg et al., 1994). Similarly, natural language generation technology can also be used in automatic financial report writing. Natural language generation technology can even be used in creative composition tasks. For example, poem composition is a difficult task for most people. However, recent advances have shown that natural language generation can be used to compose poems by providing some keywords or sentences (He et al., 2012; Zhang and Lapata, 2014). The generated text may not be perfect and may need human post-editing, but it can facilitate the creative processing of poem writing.

[1] Google can return you *cute cat* images on 17 May, 2017.

Many people read news every day. But unfortunately most news articles are long and repetitive. Natural language generation technology (more specifically automatic summarization technology) can produce shorter versions of news articles. Besides, many sentences in a news article are very formal and long. Children and second language learners may not understand them easily. Natural language generation can also be used here to convert complex sentences into simpler ones (this technique is called *sentence simplification*).

We can see from the above examples that natural language generation is an important and useful task. In the next section, we formally define natural language generation.

1.2 Natural Language Generation

Natural Language Generation (NLG) is the task of generating natural language (e.g., English sentences) from machine readable input. A NLG system uses its knowledge of language and the specific (target) domain to transform system input into human understandable text. Early NLG systems typically operate on non-linguistic representations, which are easier for computers to understand and manipulate compared to natural language. For example, the FOG system (Goldberg et al., 1994) generates weather forcasts from a weather database, which contains numerical weather simulations produced by a supercomputer. The IDAS system (Reiter et al., 1995) produces hyper-text help messages for complex machinery from a machinery knowledge base.

Linguistic inputs can be also used as NLG system inputs. In this setting, linguistic input is first mapped into computer readable representations and then a "standard" NLG model can generate language according to these representations. This extension with linguistic inputs is quite interesting, since linguistic inputs are often much easier for users to provide compared to machine readable inputs. As a result, it enables researchers to do more challenging tasks such as generating stories (McIntyre and Lapata, 2009) or poems (He et al., 2012; Zhang and Lapata, 2014) using user provided keywords. It is also possible to use a sentence as the input for a NLG system. For example, sentence compression is the task of generating a shorter sentence for an input (long) sentence. For instance, Cohn and Lapata (2009) first transform the input sentence into a tree structure (which is easier for computers to understand) and then use a tree to tree transduction method to generate the compressed sentence. Sentence simplification is another example of using a sentence as input, which aims to rewrite a

sentence into its simpler version. Woodsend and Lapata (2011) extracts syntactic and lexical simplification rules using quasi-synchronous grammar and during generation sentences are reranked with linear integer programming.

Recent advances in representation learning have enabled computers to learn image representations very well with deep Convolutional Neural Networks (CNN; Krizhevsky et al. 2012; Simonyan and Zisserman 2015). In the task of object recognition, models based on CNN even surpass human performance (He et al., 2015, 2016). As a result, researchers can successfully generate text descriptions from images or video clips with features learned by deep CNNs (Vinyals et al., 2015; Venugopalan et al., 2015).

Although researchers keep advancing natural language processing and understanding, computer vision and representation learning in the past decades, the task of NLG remains the same, i.e., to generate high quality human readable language. Indeed, these advances enable an NLG system to generate from more diverse input. Despite taking in different input, most NLG systems share similar architectures. The first architecture we discuss in this thesis is a pipeline based architecture. It usually contains a *content planning* module, a *sentence planning* module, and a *surface realization* module. Note in some literature such as Reiter and Dale (2000), *content planning* is also called *document planning* and *sentence planning* is also called *microplanning*. We will also analyse the advantages and disadvantages of using this architecture. Later, due to the advances in deep learning, the pipelined architecture can be simplified into two modules: the representation module and the generation module. In the following, we will introduce these two architectures in detail.

1.3 The Architecture of Pipelined NLG Systems

Most early NLG systems follow a pipeline architecture (Goldberg et al., 1994; Reiter et al., 1995). Typically, there are three modules in this architecture: *Content Planning*, *Sentence Planning* and *Surface Realization* (Reiter and Dale, 1997). The content planning module focuses on *what to say*, while the sentence planning and surface realization modules are responsible for *how to say*. As shown in Figure 1.1, each module takes the output of the previous module as its input.

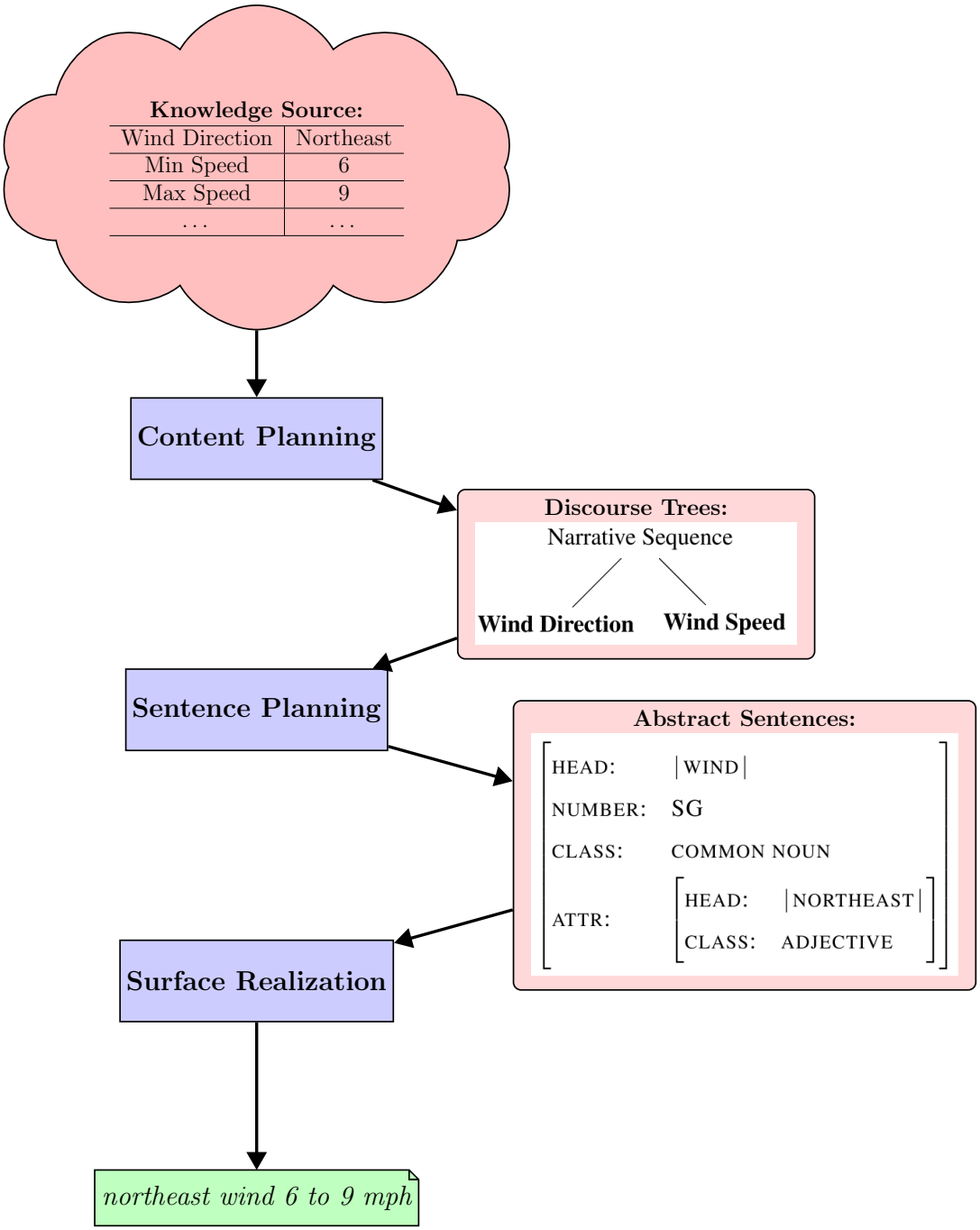
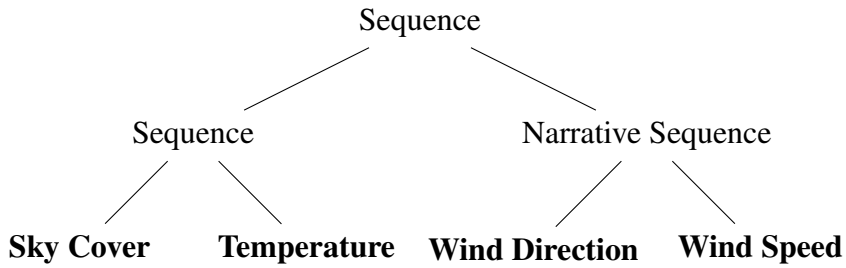


Figure 1.1: Architecture for a pipelined natural language generation system. The input is a knowledge source and the output is generated natural languge text. There are three modules inside this pipeline: *context planning*, *sentence planning* and *surface realization*.



Increasing clouds, with a low around 40. Northeast wind 6 to 9 mph.

Figure 1.2: A tree-structured document plan produced by a document planner (Hovy, 1993). Internal nodes correspond to RST relationships, while leaf nodes refer to parts of the input.

1.3.1 Content Planning

The *Content Planning* module determines which parts of the input (e.g., database) should be communicated in the final generated text. It is possible that the provided input information is redundant, and the content planning module needs to figure out which parts to include. For example, a hypothetical weather reporting system typically has access to weather data spanning an entire year. Faced with the task of generating a report for last week, it should determine which data is relevant and produce a high-level summary highlighting one or two typical days. This process is also called *content determination* (Reiter and Dale, 2000).

When multiple parts of the input are selected, there might be an additional process called *document planning*. Since in the generated text information is not presented in a random order, the content planner also needs to organize selected parts of the input in a logical and coherent manner. Note that sometimes *document planning* is omitted for simplicity (Angeli et al., 2010). The selected content produced by the content planner is usually presented in a tree structure. Figure 1.2 shows an example output of the content planning module. It is a tree structure, where internal nodes denote discourse information using Rhetorical structure theory (RST) structures (Mann and Thompson, 1988) and leaf nodes correspond to parts of the input chosen by a content determination process (Hovy, 1993).

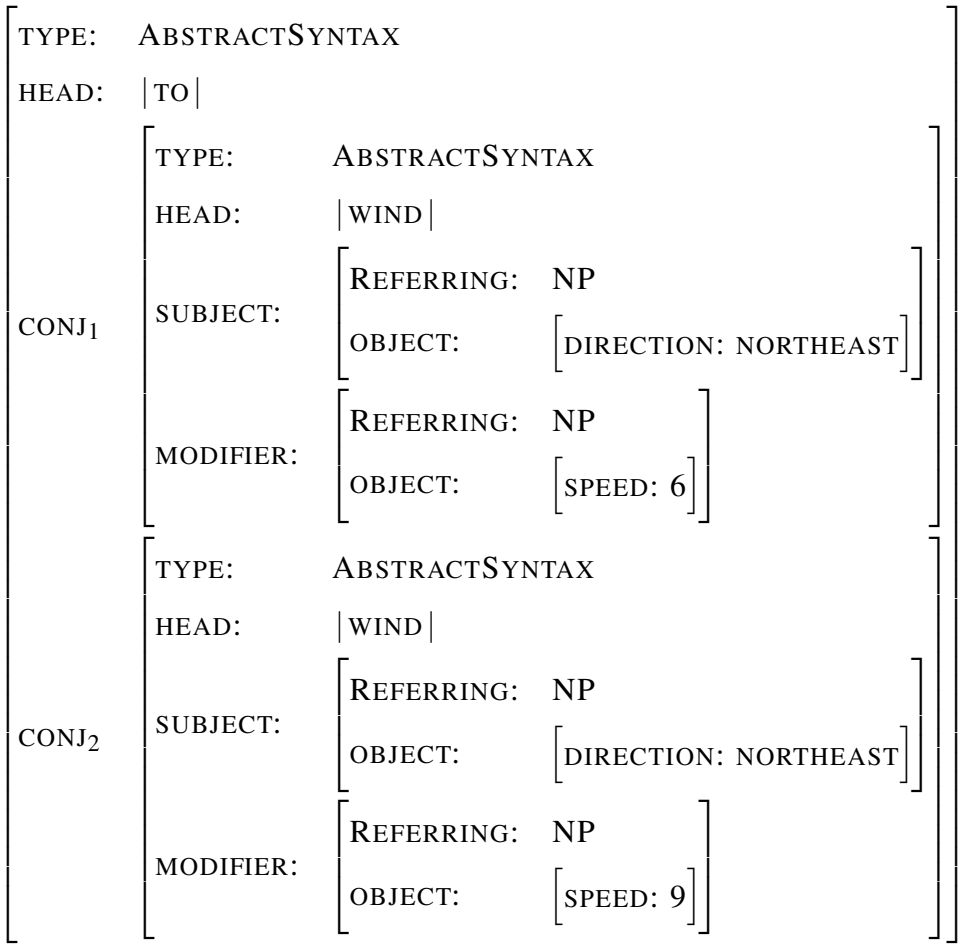


Figure 1.3: The abstract sentence representation of *Northeast wind 6 to 9 mph* in an Attribute Value Matrix (AVM). It is produced by a sentence planner in (Reiter and Dale, 2000).

1.3.2 Sentence Planning

The second module is *Sentence Planning*. In this module, the selected information produced by the content planner is grouped into different sentences, which is also called *sentence aggregation*. Note that the grouping of information may not be necessary, since the sentence planner can simply choose to present each piece of information in a single sentence. Then the *lexicalization* component chooses appropriate words or phrases to express the content selected by the content planner eariler. Sometimes grammar rules will be also applied during *lexicalization* (McIntyre and Lapata, 2009). After that, there is an optional process called *referring expression generation*, in which the domain entities are replaced. For example, in the weather report domain, the following sentence

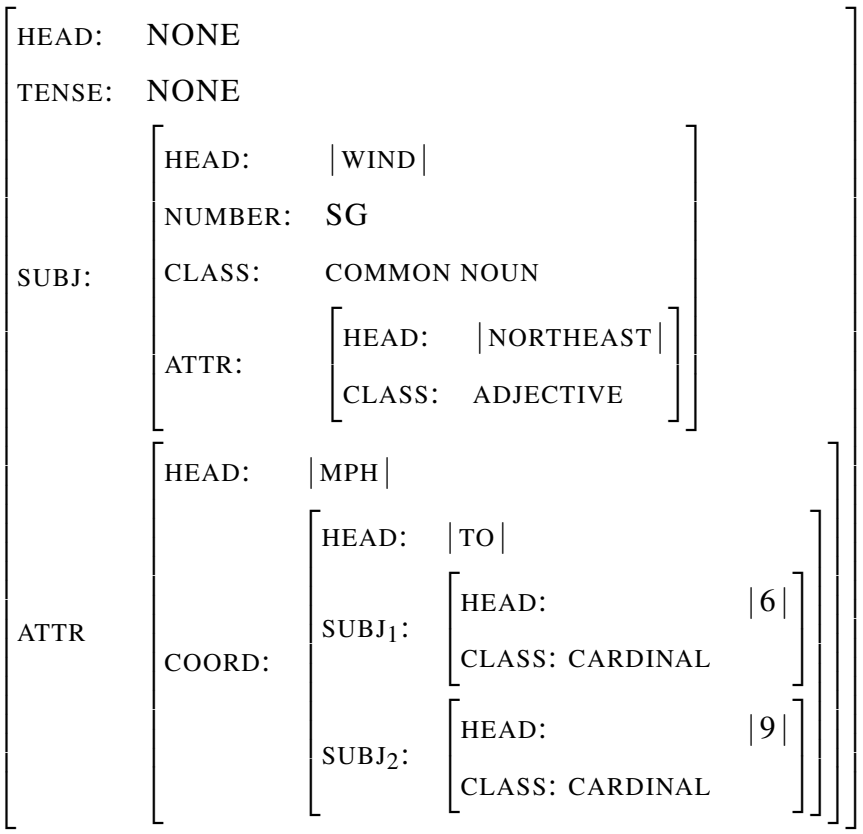


Figure 1.4: The Deep-Syntactic Structure representation of *Northeast wind 6 to 9 mph* in an Attribute Value Matrix (AVM). This representation is also the input for RealPro surface realizer (Lavoie and Rambow, 1997).

The temperature in LOC *today is between* TMP1 *and* TMP2.

where slots LOC, TMP1 and TMP2 will be filled with database records or their variants (according to the context). After all the above processes, we get abstract sentence representations. Figure 1.3 shows an example of the abstract sentence representation in abstract syntactic structure. Notable representations also include the Sentence Planning Language (Kasper, 1989), Functional Descriptions (Elhadad and Robin, 1996) and Deep-Syntactic Structure (Melčuk, 1988) ("DSyntS" for short). Figure 1.4 shows another example of abstract sentence representation in Deep-Syntactic Structures.

1.3.3 Surface Realization

The *surface realization* module transforms abstract sentence representations into syntactically and morphologically correct sentences. Compared to *Content Planning* and *Sentence Planning*, this module is more mature and there exist some off-the-shelf sur-

The family has the baby
The family has the baby. The baby is to seat the lady at the back. The baby sees the lady in the family. The family marries a lady for the triumph. The family quickly wishes the lady vanishes.
The giant guards the child
The giant guards the child. The child rescues the son from the power. The child begs the son for a pardon. The giant cries that the son laughs the happiness out of death. The child hears if the happiness tells a story.

Table 1.1: Two stories generated by the story generator in McIntyre and Lapata (2009). The system input is in italic.

face realizers such as PENMAN (Bateman, 1997), SURGE (Elhadad and Robin, 1996) and RealPro (Lavoie and Rambow, 1997). Figure 1.4 shows an example sentence produced by the RealPro realizer, which takes Deep-Syntactic Structured abstract sentence representations as input.

Now we will use an example to go through this pipeline. McIntyre and Lapata (2009) proposed a story generator by taking a user provided sentence or entities as input. Firstly, an event chain of the input entities (or entities in the input sentence) are extracted from a story corpus using a model which is similar to the one in Chambers and Jurafsky (2008). The event chain is in a graph structure (similar to a discourse tree). Each node in this graph is a (verb, relation) tuple. This corresponds to the *content planning* process described above. In the step of *sentence planning*, (verb, relation) tuples are mapped to grammar rules using lexicons (Korhonen et al., 2006) and dictionaries (Grishman et al., 1994). Then additional information expressing mood, agreement and argument roles is added to the grammar structures and these are transformed to a format compatible with the surface realizer. In *surface realization*, RealPro (Lavoie and Rambow, 1997) is used to produce candidate stories. Finally, the stories are ranked with a coherence model and an interestingness model. Table 1.1 shows two generated stories.

1.4 The Architecture of Data-driven NLG Systems

In the pipelined architecture, a NLG system is divided into well defined modules: *content planning*, *sentence planning* and *surface realization*. One advantage of this architecture is that some components may be reused across tasks. From a software engineering perspective, it is also easier to debug and maintain this architecture. However, a content planning module usually involves many manually engineered rules based on the analysis of a domain specific corpus. Similarly, sentence planning and surface realization modules also need significant engineering effort. Therefore, a system designed for one domain can not be easily used for another new domain. For example, the RealPro surface realizer (Lavoie and Rambow, 1997) relies on several linguistic knowledge bases and a great deal of grammar rules. RealPro only supports English and enabling RealPro to support other languages (e.g., French) would need significant effort.

The limitations of pipelined systems have led to the development of data-driven methods. Individual NLG components, even end-to-end systems can be learned automatically only using parallel training data. The architecture of data-driven systems is similar to that of pipelined systems discussed in Section 1.3 (they still have three modules). Compared to traditional systems, data-driven systems are domain independent, largely reducing the manual engineering effort required for developing NLG systems. Barzilay and Lapata (2005) learn a content selection module with a collective classification model. Duboue and McKeown (2002) propose a data-driven content planner using evolutionary algorithms. There are also automatically learned sentence planers (Barzilay and Lapata, 2006) and surface realizers (Knight and Hatzivassiloglou, 1995; Lu and Ng, 2011). The content planner, sentence planner and surface realizer can even be learned jointly as an end-to-end system (Angeli et al., 2010; Kim and Mooney, 2010; Konstas and Lapata, 2012).

In the next section, we introduce Neural NLG systems, which can also be viewed as end-to-end data-driven NLG models, but the classsic three-module architecture is further simplified.

1.5 The Architecture of Neural NLG Systems

Recently, an alternative architecture for NLG models has been gaining increasing attention. This architecture is shown in Figure 1.5. The input of the NLG model is firstly mapped into a vector space using a *representation* neural network and then the vector

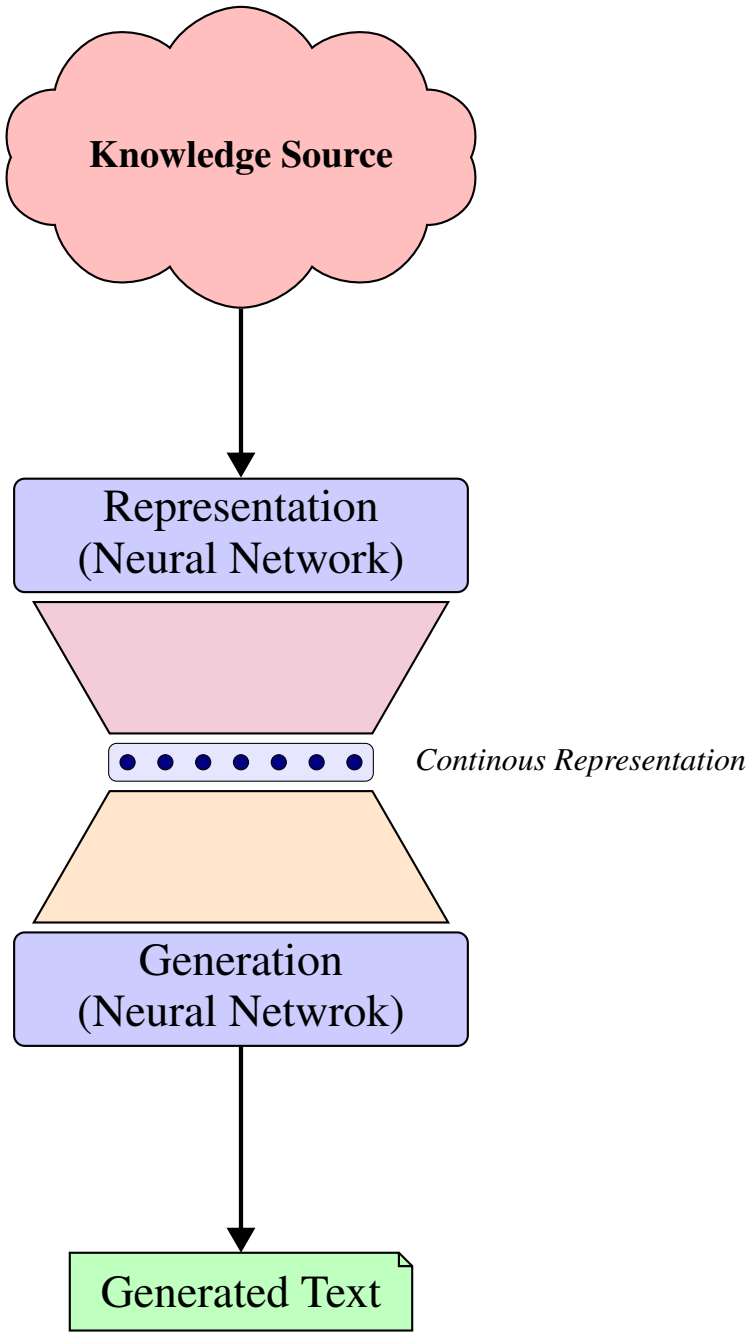


Figure 1.5: Architecture for a neural network based natural language generation system. The input is a knowledge source and the output is natural language text as in the pipelined architecture in Figure 1.1, but there are two modules inside this architecture (instead of three): a *representation* module (which is a neural network) and a *generation* module (which is usually a recurrent nerual network, since sentences can be viewed as a sequence of words). Note that these two modules can be trained jointly with gradient descent algorithms.

is decoded into text using a *genertation* neural network. We explain these two modules in more detail in the following section.

1.5.1 Representation

Thanks to the recent advances in representation and deep learning, computers can encode many different contents into a continuous vector space. For example, `word2vec` (Mikolov et al., 2013b) is a powerful model for mapping words into dense vectors, where similar words are close to each other in vector space. Sentences can also be represented as continous vector representations using recursive neural networks (Socher et al., 2011b), convolutional neural networks (Kim, 2014) and recurrent neural networks (Palangi et al., 2016). Even images (Krizhevsky et al., 2012) and knowledge bases (Bordes et al., 2011) can be encoded into vector spaces. As mentioned in the beginning of this chapter, a NLG system can take several different kinds of input, but now we can use neural networks to first map all these inputs into a vector (or several vectors). Then NLG is reduced to a problem of generating text from a vector (or several vectors). So the first module in a neural network based NLG system is simply a *representation* module.

1.5.2 Sequence Generation

Given the vector or vectors produced by the *representation* module, the *sequence generation* module is responsible for transforming vectors into text. A sentence or a couple of sentences can be viewed as a sequence of words (including punctuation). There is a special kind of neural network called recurrent neural network (RNN), which is designed for sequence prediction or generation. In natural language processing, perhaps the first successful application of (simple) RNNs is in language modeling (Mikolov et al., 2010). Later, Kalchbrenner and Blunsom (2013) applied (simple) RNNs to machine translation by adding source sentence representations (learned by a convolutional neural network) to RNNs. But simple RNNs are not strong enough to generate reasonable sentences and were used mostly for reranking. With the introduction of gated recurrent units and long short-term memory networks, the quality of the generated text improved dramatically (Cho et al., 2014; Sutskever et al., 2014). Recurrent neural networks especially long short-term memory networks completely revolutionized natural language generation and actually almost all recently proposed neural network based NLG models are based on recurrent neural networks. Specifically, in a neural natural

Figure 1.6: An example of image-caption pair in flickr8k dataset. The five descriptions for this image are *1. A child in a pink dress is climbing up a set of stairs in an entry way. 2. A girl going into a wooden building. 3. A little girl climbing into a wooden playhouse. 4. A little girl climbing the stairs to her playhouse. 5. A little girl in a pink dress going into a wooden cabin.*

language generation system, a recurrent neural network will predict one word at time by taking all previously generated words and the representation produced by the *representation* module into account. The generator stops until a special token indicating the end of sentence (or document) is produced. One can view the *sequence generation* module as a combination of the *content planning*, *sentence planning* and *surface realization* modules in the pipelined architecture.

As in Section 1.3, we will also use examples to go through this architecture. Vinyals et al. (2015) proposed a model following the architecture in Figure 1.5 for caption generation. Caption generation aims to generate a text description for a given image. An example image from the Flickr8k dataset (Hodosh et al., 2013) is shown in Figure 1.6 and one of its five gold standard captions is *A child in a pink dress is climbing up a*

Figure 1.7: An example of infobox to text generation. The description for this infobox is *Ludwig van Beethoven (baptised 17 December 1770[1] – 26 March 1827) was a German composer and pianist.*

set of stairs in an entry way (other gold captions are shown in Figure 1.6). In their model, a 22-layer convoluational neural network (Szegedy et al., 2015) is used to map images into vectors (so their *representation* module is a very deep convolutional neural network). Their *sequence generation* module is a long short-term memory network (Hochreiter and Schmidhuber, 1997), which is initialized with the image feature vector learned by the *representation* module. In their initial attempt, they did not update the *representation* module to avoid overfitting to the image caption dataset. Note the size of the dataset used to train the deep convolutional neural network is much larger than the size of the image caption dataset. But in later work (Vinyals et al., 2017), they found that careful tunning of these two parts can lead to even better performance.

Another example of a neural NLG model generates descriptions for Wikipedia infoboxes in the biography domain (Lebret et al., 2016), where the infobox for a person is given, and a model is expected to generate a description for it (see Figure 1.7). In their model, a infobox (or a table in general) is mapped to continuous vector space with embedding matrices of field names and field values (i.e., words). Then, conditioned on the representation of the infobox, they use a feed-forward neural *n*gram language model to generate the description. They also observe that some of the generated words are from actually *copied* from the infobox. For example, *composer* and *pianist* appear both in the infobox and the description. Therefore, a *copy* operation is explicitly modeled by augmenting the output word vocabulary with all words in the infobox as well.

1.6 Thesis Contributions

There are at least three advantages for the neural NLG architecture. Firstly, deep neural networks are usually more expressive than many shallow models and therefore often lead to better performance. Furthermore, the *representation* and the *sequence generation* modules can be trained jointly in an end-to-end fashion (that is another nice property of neural networks). This means that if you have a parallel corpus for the input and corresponding natural language output, then you can train your NLG model with it. All you need to do is design an appropriate neural module for the input representation and another neural module to map the representations into word sequences. The training loss for the whole generation module is directly learned from the output. Therefore, there is no need to train the representation module and generation modules separately with different loss functions. Lastly, it is easier to do transfer learning in neural networks, which means that one task (with less training data) can benefit from another task (with much more training data) by parameter sharing. For example, in many NLP neural models, the word embedding matrices can usually be initialized with pre-trained embeddings such as word2vec (Mikolov et al., 2013b) or Glove (Pennington et al., 2014) vectors, where these word vectors are trained on large amounts of unlabeled data. This initialization usually leads to better performance. Therefore, the models proposed in this thesis are all based on the neural NLG architecture due to the flexibility and powerfulness of neural networks.

Hierarchical Recurrent Generation Model. Most neural based generation models aim to generate a single sentence as the output, which is a better *representation*

network for the neural language generation architecture. In this thesis, we propose a *hierarchical recurrent neural network* architecture to model and generate multiple sentences. Most recurrent neural network based generation models rely on static context during generation, we propose a way to make the context dynamic. Intuitively, in different generate stage, the context needed should be different.

Deep Reinforcement Learning Generation Model. Neural based NLG models are very powerful, when there is a lot of good-quality data. However, there are many NLG sub-tasks for which data is sparse or noisy. When the training data is not sufficient or noisy, prior knowledge for the task becomes important. How to model this kind of prior knowledge effectively is very important for generating good output. Therefore, we propose a deep reinforcement learning framework for this kind of generation tasks and model task specific prior knowledge as rewards, which is a better training method for the neural language generation architecture.

Tree Structured Neural Generation Model. Most existing neural NLG work uses sequence models (i.e., recurrent neural networks) as the *generation* module. In a sequence based model, a sentence is viewed as a sequence of words. However, in computational linguistics, sentences can also be represented in a tree structure such as dependency trees or constituency trees. In a linear sentence of length N, the longest dependency for a word is at length $N-1$, while by using the tree structure, the longest dependency length can be reduced to rougely $\log(N)$. By reducing the dependency length, we can probably train the model better. We propose a neural language generator (a better *generation* network) called *top-down tree long short-term memory network*, which generates dependency trees rather than linear structured sentences. The final sentences can be generated using in-order traversal of dependency trees.

1.7 Thesis Outline

In this chapter, we have introduced the task of Natural Language Generation, two popular natural language generation architectures and my contribution to neural based natural language generation models. The reminder of this thesis is structured as follows:

- **Chapter 2.** In Chapter 2, we introduce the very basics of a neural network based natural language generation model, namely (recurrent) neural network based language models. We first introduce *language modeling*, then present the details

of a *feed-forward neural language model* (Bengio et al., 2003) and finally we introduce recurrent neural networks based language models including *simple recurrent neural network language models* and *long short-term memory network language models*. Next, we discuss *context sensitive language modelling* (which is almost a neural language generation model), *sequence to sequence learning* (Sutskever et al., 2014) and the *attention mechanism* (Bahdanau et al., 2015).

- **Chapter 3.** In Chapter 3, we introduce our *hierarchical recurrent neural network* model, which is designed to generate multiple sentences. The model is tested in Chinese poetry generation (specifically *quatrain* generation). In a quatrain, there are four sentences and each sentence has five or seven words. Given several keywords, the first sentence of a *quatrain* is generated with a template based method (He et al., 2012). Subsequent sentences are generated with a *hierarchical recurrent neural network*. To make the generator more robust, during decoding, the neural network is interpolated with an *n*gram language model and several machine translation features. The proposed model outperforms a statistical machine translation based poetry generator (He et al., 2012) and a summarization based poetry generator (Yan et al., 2013).

- **Chapter 4.** In Chapter 4, we introduce our *deep reinforcement learning based generation model*. As mentioned earlier, this model can incorporate task specific prior knowledge for a neural based language generation model. *Sentence Simplification* is used as a test bed for this model, a challenging task, for which relatively little training data is available. *Sentence Simplification* aims to rewrite a complex sentence into a simpler one. This is a typical sequence to sequence learning task and therefore the RNN encoder-decoder model is used as a starting point. However, the RNN encoder-decoder does not solve the simplification problem very well. The output is quite boring, i.e., it always repeats the complex sentence or only makes a few trivial changes. Simplified sentences should be *simpler*, *meaning preserving* and *grammatical*. If we can model these three properties well, we can naturally solve the repetition curse (repeating the complex sentence will not make the output simpler!). We choose the reinforcement learning framework, since it can naturally incorporate non-differentiable objectives (we simply model these objectives as rewards). Our reinforcement learning based model outperforms several strong simplification systems (Wubben et al., 2012; Narayan and Gardent, 2014; Xu et al., 2016).

- **Chapter 5.** Most neural language generation models view sentences as sequences of words and use sequence models (e.g., long short-term memory networks) to generate the target sentences. Despite superior performance in many applications, neural language generation models essentially predict sequences of words. Many NLP tasks, however, exploit syntactic information operating over tree structures (e.g., dependency or constituent trees). In this chapter we develop a novel neural network model which combines the advantages of the LSTM architecture and syntactic structure. Our model estimates the probability of a sentence by estimating the generation probability of its dependency tree. Instead of explicitly encoding tree structure as a set of features, we use four LSTM networks to model four types of dependency edges which altogether specify how the tree is built. At each time step, one LSTM is activated which predicts the next word conditioned on the sub-tree generated so far. To learn the representations of the conditioned sub-tree, we force the four LSTMs to share their hidden layers. Application of our model to the MSR sentence completion challenge achieves results beyond the current state of the art. Our model is also capable of generating trees just by sampling from a trained model and can be seamlessly integrated with text generation applications.

- **Chapter 6.** This chapter focuses on dependency parsing. Although not the main focus of this thesis, better dependency parsing performance can potentially boost the performance of dependency tree based models (e.g., the *top-down TreeLSTM* described in Chapter 5). Conventional graph-based dependency parsers guarantee a tree structure both during training and inference. Instead, we formalize dependency parsing as the problem of independently selecting the head of each word in a sentence. Our model which we call DENSE (as shorthand for **De**pendency **N**eural **Se**lection) produces a distribution over possible heads for each word using features obtained from a bidirectional recurrent neural network. Without enforcing structural constraints during training, DENSE generates (at inference time) trees for the overwhelming majority of sentences, while non-tree outputs can be adjusted with a maximum spanning tree algorithm. We evaluate DENSE on four languages (English, Chinese, Czech, and German) with varying degrees of non-projectivity. Despite the simplicity of the approach, our parsers are on par with the state of the art. We also applied this parser to the *TreeLSTM* models proposed in Chapter 5 and improved their performance.

- **Chapter 7.** In chapter 7, we conclude this thesis and discuss directions for future research.

Some of the work presented here has been previously published in Zhang and Lapata (2014) (Chapter 3), Zhang and Lapata (2017) (Chapter 4), Zhang et al. (2016) (Chapter 5) and Zhang et al. (2017) (Chapter 6).

Chapter 2

Background

As introduced in Chapter 1, there are two modules in a neural based natural language generation system, namely the *representation* module and the *generation* module. Both of them are neural networks. The *representation* neural network transforms system input into continuous vectors and the generation neural network generates one or more sentences based on the vectors produced by the *representation* network. Since incorporating the vectors learned by the *representation* network is trivial (simply treating them as additional input for the generation network), we will first introduce the generation network, a neural language model. Then, we will discuss how the representation network interacts with the generation network.

We will introduce the background knowledge for neural network based natural language generation, which covers Neural Network Language Models, Recurrent Neural Networks and Encoder-Decoder Models. A language model (LM) can be used to predict the next word given its preceding context. Traditional language models employ count based methods coupled with smoothing techniques (Katz, 1987; Kneser and Ney, 1995) to estimate the current word given its history. However, it is difficult to jointly model other content (e.g., images or audio), which is not count based, in a language model. Bengio et al. (2003) introduced a neural network based language model, where words are mapped into continuous space. Indeed, after introducing neural networks into language modeling, they are able to build a much better language model especially when a lot of training data is available. But this is far from the most interesting part. Neural networks are capable of mapping words or other content (such as video and images) into a continuous space (i.e., a vector) and different neural networks can be connected to each other simply by stacking and they can even be trained jointly in an end to end fashion. For example, Kiros et al. (2014) intro-

duced a conditional neural network language model by taking image features learned by a seven layer convolution neural network (Krizhevsky et al., 2012) into account. The model is capable of generating descriptions for given images. In fact, most recent text generation models are conditional language models! Among all these neural language models, long short-term memory network language models (Hochreiter and Schmidhuber, 1997; Sundermeyer et al., 2012) are usually more powerful. In natural language processing, conditional language models that are conditioned on another sentence are particularly popular, because many tasks such as machine translation and text rewriting can be modeled this way. Cho et al. (2014) and Sutskever et al. (2014) proposed RNN Encoder-Decoder models for this class of problems and later an important variant called Attention-based Encoder-Decoder Model (Bahdanau et al., 2015) was developed, which makes Encoder-Decoder models much more powerful. Although encoder-decoder models (and their attention-based variants) were initially proposed to predict a sequence with the decoder network based on another sequence learned by the encoder network, the encoder network can also encode non-sequential data such as image patches or even database records. In the remainder of this chaper, we will introduce the models mentioned above in more detail.

2.1 Neural Network Language Models

A language model is used to estimate the probability of a sentence. Usually, the probability is decomposed into probabilities of all words given preceding words in the sentence $S = (w_0 = \text{BOS}, w_1, w_2, \ldots, w_N = \text{EOS})$[1]:

$$P(S) = \prod_{i=1}^{N} P(w_i | w_{0:i-1}) \tag{2.1}$$

However, estimating the term $P(w_i|w_{0:i-1})$ directly is difficult, as the context of w_i (i.e., $w_{0:i-1}$) can be very long. To simplify this problem, a Markov assumption can be applied and we can assume that w_i is only related to the previous $n-1$ words, which is the $n$gram model. Therefore, $P(S)$ (the probability of S) becomes:

$$P(S) = \prod_{i=1}^{N} P(w_i | w_{i-n+1:i-1}) \tag{2.2}$$

Note that the "$i-n+1$" in $w_{i-n+1:i-1}$ may be smaller than 0 (e.g., $i = 1$ and $n = 3$). A common solution for this is simply viewing $w_i (i \leq 0)$ as a BOS token (Bengio

[1] As a convention in language modeling, we assume every sentence starts with an additional BOS (**B**egin **O**f **S**entence) token and ends with an EOS (**E**nd **O**f **S**entence) token.

et al., 2003). Bengio et al. (2003) use a feed-forward neural network to estimate $P(w_i|w_{i-n+1:i-1})$. The preceding $n-1$ words $w_{i-n+1:i-1}$ are mapped to a continuous space with a word embedding matrix $\mathbf{W}_e \in \mathbb{R}^{d\times|V|}$ ($|V|$ is the vocabulary size and d is the word embedding dimension):

$$\mathbf{h}_0 = \begin{bmatrix} \mathbf{W}_e\, e(w_{i-n+1}) \\ \dots \\ \mathbf{W}_e\, e(w_{i-1}) \end{bmatrix} \tag{2.3}$$

Where $e(w)$ is a one hot vector for word w. Then, usually there are one or more hidden layers to add more abstractions to $\mathbf{h}_0$ (here for simplicity we assume there is only one hidden layer):

$$\mathbf{h}_1 = f(\mathbf{W}_{h0}\,\mathbf{h}_0 + \mathbf{b}_{h0}) \tag{2.4}$$

Where $\mathbf{W}_{h0} \in \mathbb{R}^{d\times(n-1)d}$ and f is an activation function such as tanh or ReLU (Glorot et al., 2011). To reduce over-fitting, one can also apply Dropout (Srivastava et al., 2014) after the hidden activation function. With the context at hand, the probability distribution of the next word is:

$$P(w_i|w_{i-n+1:i-1}) = \text{softmax}(\mathbf{W}_o\,\mathbf{h}_1 + \mathbf{b}_o) \tag{2.5}$$

where $\mathbf{W}_o \in \mathbb{R}^{|V|\times d}$. Usually, the model is trained by minimizing the negative log-likelihood of the training data with stochastic gradient descent. At this point, a language model can only leverage fixed length context. In the next part, we will introduce how to overcome this limitation.

2.2 Recurrent Neural Network Language Models

Recurrent Neural Networks (RNNs) can in theory take unlimited context into account. In language modeling, RNNs can estimate the probability of a word given any number (rather than a limited number) of preceding words (i.e., $P(w_i|w_{i-n+1:i-1})$). Like in a feed-forward neural network, the preceding words are encoded into a vector, then a softmax function is used to estimate the probability distribution for the next word (see Equation 2.5). Next, we will introduce the two most popular RNN architectures: the Elman RNN (Elman, 1990), which is also called Simple Recurrent Neural Network and the Long Short-Term Memory Network (Hochreiter and Schmidhuber, 1997).

2.2.1 Simple Recurrent Neural Networks

Similar to the neural network language model described earlier, a word embedding matrix $\mathbf{W}_e$ is used to map words to continuous vectors. Note that the original RNN language model described in Mikolov et al. (2010) has no word embedding matrix, but we still include the word embedding matrix as most work does (Cho et al., 2014; Sutskever et al., 2014). Suppose we have seen the first t words $w_1, w_2, \ldots, w_t$ and we are trying to estimate the probability distribution for the next word w_{t+1}. We use t to refer to time steps as a convention in RNNs. We assume $\mathbf{h}_t$ is the hidden state for time step $t \in [0, N]$. $\mathbf{h}_0$ is usually initialized to either $\mathbf{0}$ or $a * \mathbf{1}$ (where a is a small value such as 0.1). The computation for time step $t \geq 1$ is as follows (we ignore all bias terms for simplicity):

$$\mathbf{x}_t = \mathbf{W}_e \, e(w_t) \tag{2.6}$$

$$\mathbf{h}_t = f(\mathbf{W}_{ih} \, \mathbf{x}_t + \mathbf{W}_{hh} \, \mathbf{h}_{t-1}) \tag{2.7}$$

$$P(w_{t+1} | w_{1:t}) = \text{softmax}(\mathbf{W}_{ho} \, \mathbf{h}_t) \tag{2.8}$$

The main difference between a feed-forward neural network is that the RNN has a notion of memory due to $\mathbf{h}_t$. f is usually a hyperbolic tangent (tanh) or sigmoid (σ) function. f can also be a rectified linear unit (ReLU) function (Glorot et al., 2011), but the recurrent matrix $\mathbf{W}_{hh}$ needs to be initialized as an identity matrix (Le et al., 2015).

RNNs are usually trained with back propagation through time (BPTT) (Rumelhart et al., 1988; Werbos, 1988). Training RNNs can be difficult due to the exploding and vanishing gradient problems (Bengio et al., 1994).

2.2.2 Long Short-Term Memory Networks

Long Short-Term Memory Networks (LSTMs; Hochreiter and Schmidhuber 1997) are a special kind of RNN designed for solving the vanishing gradient problem in regular RNNs. The biggest difference between a simple RNN lies in the computation of the hidden state $\mathbf{h}_t$ (see the computation in RNNs in Equation 2.7), where gating mechanisms are introduced. In addition to the hidden state $\mathbf{h}_t$, in each time step, a memory cell $\mathbf{c}_t$ is also introduced. $\mathbf{h}_0$ and $\mathbf{c}_0$ are all initialized to $\mathbf{0}$. At time step t, we firstly compute the information update $\mathbf{u}_t$:

$$\mathbf{u}_t = \tanh(\mathbf{W}_{ux} \, \mathbf{x}_t + \mathbf{W}_{uh} \, \mathbf{h}_{t-1}) \tag{2.9}$$

Then an input gate $\mathbf{i}_t$ is introduced to control how much information in $\mathbf{u}_t$ will flow into the new memory cell $\mathbf{c}_t$ and a forget gate $\mathbf{f}_t$ is introduced to control how much information in the previous memory cell $\mathbf{c}_{t-1}$ should be *remembered* (rather than forgotten):

$$\mathbf{i}_t = \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1}) \tag{2.10}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fh}\mathbf{h}_{t-1}) \tag{2.11}$$

$$\mathbf{c}_t = \mathbf{i}_t \odot \mathbf{u}_t + \mathbf{f}_t \odot \mathbf{c}_{t-1} \tag{2.12}$$

where σ is a sigmoid function. Finally, yet another gate, the output gate $\mathbf{o}_t$ is also introduced to determine which part of the memory cell $\mathbf{c}_t$ should flow into the hidden state $\mathbf{h}_t$:

$$\mathbf{o}_t = \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1}) \tag{2.13}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \tag{2.14}$$

In the context of language modeling, we can simply put a softmax layer on the top of $\mathbf{h}_t$, as shown in Equation (2.8).

Although training RNNs is difficult due to the exploding and vanishing gradient problems (Bengio et al., 1994), the vanishing gradient problem can be alleviated by using LSTMs and the exploding gradient problem can be alleviated by rescaling the gradients $g = \frac{\eta g}{|g|}$ when the norm exceeds a threshold η (e.g., $\eta = 5$) (Pascanu et al., 2013).

There are also other kinds of neural language models based on Gated Recurrent Units (GRUs; Cho et al. 2014) and Memory Networks (Sukhbaatar et al., 2015). A GRU unit can be viewed as a simpler version of the LSTM unit, where there are only two gates and no memory cells. The Memory Network is similar to a feed-forward neural network, but it has additional addressing and attention mechanisms. We refer the interested readers to Cho et al. (2014); Chung et al. (2014); Sukhbaatar et al. (2015) for more details.

2.2.3 Generation with a Language Model

Given a neural language model, a sentence can be generated by sampling from the distribution of the LM until an EOS[2] token is produced. As mentioned earlier, a neural language model can be conditioned on other modules (Kiros et al., 2014; Vinyals et al., 2015; Mei et al., 2016) and generate sentences according to this conditioning context.

[2]EOS is a shorthand for **E**nd **O**f **S**entence.

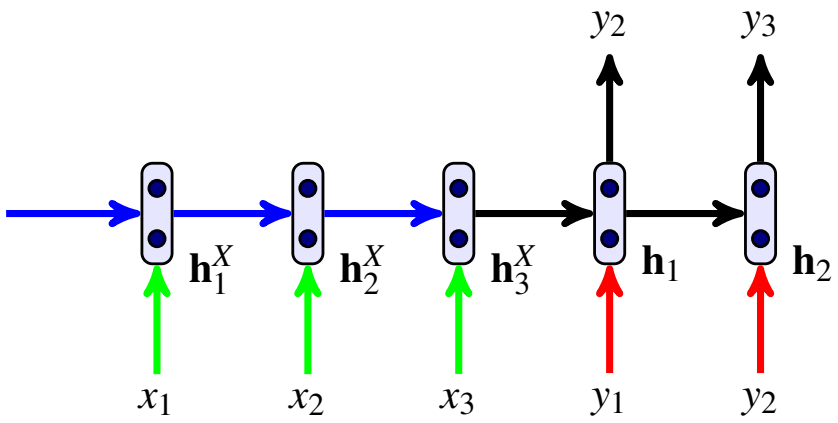


Figure 2.1: RNN Encoder-Decoder Model (without Attention Mechanism).

For example, neural language models can be used to generate image descriptions by conditioning on image features (Kiros et al., 2014; Vinyals et al., 2015); they can also be used in concept to text generation by conditioning on database records (Mei et al., 2016).

2.3 RNN Encoder Decoder Models

Many NLP tasks can be framed as text-to-text generation tasks (e.g., machine translation, text rewriting, dialog generation), where the input is text and the output is also text. Text-to-text tasks can be conceptualized as conditional language modeling where output text is conditioned on some input text. The RNN encoder-decoder model (Kalchbrenner and Blunsom, 2013; Cho et al., 2014; Sutskever et al., 2014) is proposed to solve this kind of problem. Specifically, an *encoder*, which is a recurrent neural network, encodes the input sentence into a list of hidden states. Then, a *decoder*, which is another recurrent neural network (language model), generates the output sentence based on the hidden states learned from the encoder.

The original encoder-decoder model was initially applied in machine translation without however outperforming more traditional translation models (Durrani et al., 2014; Sutskever et al., 2014) until the attention model was introduced (Bahdanau et al., 2015; Jean et al., 2015). We will first discuss the basic encoder-decoder architecture and then go to its attention-based extension.

Suppose we have a sentence pair (X,Y) where $X = (x_1,\ldots,x_{|X|})$ and $Y = (y_1,\ldots,y_{|Y|})$. The task is to use X to predict Y. We use an LSTM Encoder to transform X to

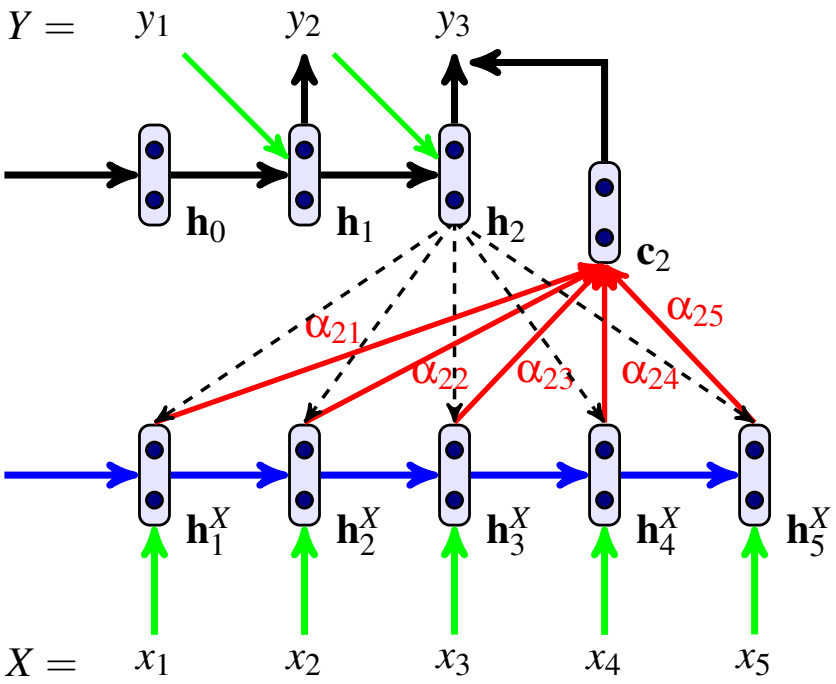


Figure 2.2: RNN Encoder-Decoder Model with Attention Mechanism.

$(\mathbf{h}_1^X, \ldots, \mathbf{h}_{|X|}^X)$ (sometimes the encoder is a bidirectional LSTM[3]). The decoder is an LSTM language model. In a vanilla encoder-decoder model, we can simply use $\mathbf{h}_{|X|}^X$ to initialize the decoder LSTM (see Figure 2.1). This setup assumes a single vector (i.e., $\mathbf{h}_{|X|}^X$) can encode the whole source sentence. However, it is difficult in practice to do so (Sutskever et al., 2014; Bahdanau et al., 2015). In the attention-based encoder-decoder model, the model assumes that at different steps of the generation, the decoder should focus on different parts of the input, while how much each part of the input should be focused upon is determined by the attention mechanism (Bahdanau et al., 2015). Specifically, as shown in Figure 2.2, at each time step t, in addition to the word embedding of y_t, we also give a dynamic context vector $\mathbf{c}_t$ to the LSTM (note we use $\text{LSTM}(\cdot)$ to summarize Equation (2.9) to Equation (2.14)):

$$\mathbf{h}_t = \text{LSTM}([\mathbf{W}_e^D e(y_t); \mathbf{c}_t], \mathbf{h}_{t-1}) \tag{2.15}$$

where $\mathbf{W}_e^D$ is the decoder word embeding matrix, $e(y_t)$ is the one-hot encoding of word

[3]The bidirectional LSTM (Schuster and Paliwal, 1997) encoder has two LSTMs (a forward LSTM and a backward LSTM). The forward LSTM encodes X into a list of forward hidden states $(\overrightarrow{\mathbf{h}_1^X}, \overrightarrow{\mathbf{h}_2^X}, \ldots, \overrightarrow{\mathbf{h}_{|X|}^X})$. Similarly, the backward LSTM encodes X into $(\overleftarrow{\mathbf{h}_1^X}, \overleftarrow{\mathbf{h}_2^X}, \ldots, \overleftarrow{\mathbf{h}_{|X|}^X})$. The final hidden states for X is $(\mathbf{h}_1^X, \ldots, \mathbf{h}_{|X|}^X)$, where $\mathbf{h}_t^X$ is the concatenation of $\overrightarrow{\mathbf{h}_t^X}$ and $\overleftarrow{\mathbf{h}_t^X}$.

y_t, $\mathbf{c}_t$ is dynamically constructed as follows:

$$\mathbf{c}_t = \sum_{i=1}^{|X|} \alpha_{ti} \mathbf{h}_i^X \tag{2.16}$$

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_i \exp(e_{ti})} \tag{2.17}$$

$$e_{ti} = \mathbf{v}^\top \tanh(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{h}_i^X) \tag{2.18}$$

where α_{ti} is the attention score.

As in Equation 2.8, the final word distribution for y_{t+1} is:

$$P(y_{t+1}|y_{1:t}, X) = \text{softmax}(\mathbf{W}_{ho}\mathbf{h}_t) \tag{2.19}$$

Although the encoder-decoder model (and its attention variant) described above was initially proposed to predict a sequence based on another sequence, the encoder network can also encode non-sequential data such as image patches (Xu et al., 2015a), video frames (Venugopalan et al., 2015) or even database records (Mei et al., 2016).

2.4 Summary

In this chapter, we introduced the basis of neural language generation models, namely conditional language models. The most popular conditional language models are recurrent neural network language models. We also introduced the encoder-decoder framework and the attention mechanism, which were initially proposed in the context of neural machine translation, but have been used in natural language generation. Most neural network based natural language generation models aim to generate a single sentence. In the next chapter, we will introduced a model for multiple sentence generation.

Chapter 3

Generating Multiple Sentences in Sequence based Generation

Most neural network based natural language generation models aim to generate a single sentence. To generate multiple sentences, we propose a hierarchical neural generation model. The proposed hierarchical model is composed of two recurrent neural networks stacked in a hierarchical structure. Unlike a non-hierarchical recurrent network (which only handles word level recurrence), our model can handle not only the word level recurrence (the generation within a sentence) but also the sentence level recurrence (the context compression for all previous sentences). We assume the hierarchical model can memorize generated words/sentences better than its non-hierarchical counterpart, since it is conditioned on larger context and the sentence level recurrent network can enforce sentence coherence. We apply this model to the task of Chinese poetry generation. We use Chinese poems rather than English poems as our test bed because we believe Chinese poems are easier to generate (it has only four lines).

The neural based language generation architecture we introduced in chapter 1 has two main components: the *representation* network and the generation network. What we propose in this chapter is actually a better *representation* network.

The neural network generator *jointly* performs content selection ("what to say") and surface realization ("how to say") by learning representations of individual characters, and their combinations into one or more lines as well as how these mutually reinforce and constrain each other. Poem lines are generated incrementally by taking into account the entire history of what has been generated so far rather than the limited horizon imposed by the previous line or lexical n-grams. Experimental results show that our model outperforms competitive Chinese poetry generation systems using both

相 思
Missing You
红 豆 生 南 国， (* Z P P Z)
Red berries born in the warm southland.
春 来 发 几 **枝**？ (P P Z Z P)
How many branches flush in the spring?
愿 君 多 采 撷， (* P P Z Z)
Take home an armful, for my sake,
此 物 最 相 **思**。 (* Z Z P P)
As a symbol of our love.

Table 3.1: An example of a 5-char *quatrain* exhibiting one of the most popular tonal patterns. The tone of each character is shown at the end of each line (within parentheses); P and Z are shorthands for *Ping* and *Ze* tones, respectively; * indicates that the tone is not fixed and can be either. Rhyming characters are shown in boldface.

automatic and manual evaluation methods.[1]

3.1 Introduction

Classical poems are a significant part of China's cultural heritage. Their popularity manifests itself in many aspects of everyday life, e.g., as a means of expressing personal emotion, political views, or communicating messages at festive occasions as well as funerals. Amongst the many different types of classical Chinese poetry, *quatrain* and *regulated verse* are perhaps the best-known ones. Both types of poem must meet a set of structural, phonological, and semantic requirements, rendering their composition a formidable task left to the very best scholars.

An example of a quatrain is shown in Table 3.1. Quatrains have four lines, each five or seven characters long. Throughout this chapter, we will use "character" and "word" interchangeably[2]. Characters in turn follow specific phonological patterns, within each line and across lines. For instance, the final characters in the second, fourth

[1] Our code is available at `https://github.com/XingxingZhang/rnnpg`.

[2] In ancient Chinese, each character has its own meaning and it is almost a word in modern Chinese (or other languages). There are 6773 different characters in our training set (see details in Section 3.4.1).

and (optionally) first line must rhyme, whereas there are no rhyming constraints for the third line. Moreover, poems must follow a prescribed tonal pattern. In traditional Chinese, every character has one tone, *Ping* (level tone) or *Ze* (downward tone). The poem in Table 3.1 exemplifies one of the most popular tonal patterns (Wang, 2002). Besides adhering to the above formal criteria, poems must exhibit concise and accurate use of language, engage the reader/hearer, stimulate their imagination, and bring out their feelings.

In this chapter we are concerned with generating traditional Chinese poems automatically. Although computers are no substitute for poetic creativity, they can analyze very large online text repositories of poems, extract statistical patterns, maintain them in memory and use them to generate many possible variants. Furthermore, while amateur poets may struggle to remember and apply formal tonal and structural constraints, it is relatively straightforward for the machine to check whether a candidate poem conforms to these requirements. Poetry generation has received a fair amount of attention over the past years (see the discussion in Section 3.2), with dozens of computational systems written to produce poems of varying sophistication. Beyond the long-term goal of building an autonomous intelligent system capable of creating meaningful poems, there are potential short-term applications for computer generated poetry in the ever growing industry of electronic entertainment and interactive fiction as well as in education. An assistive environment for poem composition could allow teachers and students to create poems subject to their requirements, and enhance their writing experience.

We propose a model for Chinese poem generation based on recurrent neural networks. Our generator *jointly* performs content selection ("what to say") and surface realization ("how to say"). Given a large collection of poems, we learn representations of individual characters, and their combinations into one or more lines as well as how these mutually reinforce and constrain each other. Our model generates lines in a poem probabilistically: it estimates the probability of the current line given the probability of all previously generated lines. We use a recurrent neural network to learn the representations of the lines generated so far which in turn serve as input to a recurrent language model (Mikolov et al., 2010, 2011b,a) which generates the current line. In contrast to previous approaches (Greene et al., 2010; Jiang and Zhou, 2008), our generator makes no Markov assumptions about the dependencies of the words within a line and across lines.

We evaluate our approach on the task of quatrain generation (see Table 3.1 for

a human-written example). Experimental results show that our model outperforms competitive Chinese poetry generation systems using both automatic and manual evaluation methods.

3.2 Related Work

Automated poetry generation has been a popular research topic over the past decades (see Colton et al. (2012) and the references therein). Most approaches employ templates to construct poems according to a set of constraints (e.g., rhyme, meter, stress, word frequency) in combination with corpus-based and lexicographic resources. For example, the Haiku poem generator presented in Wu et al. (2009) and Tosa et al. (2008) produces poems by expanding user queries with rules extracted from a corpus and additional lexical resources. Netzer et al. (2009) generate Haiku with Word Association Norms, Agirrezabal et al. (2013) compose Basque poems using patterns based on parts of speech and WordNet (Fellbaum, 1998), and Oliveira (2012) presents a generation algorithm for Portuguese which leverages semantic and grammar templates.

A second line of research uses genetic algorithms for poem generation (Manurung, 2003; Manurung et al., 2012; Zhou et al., 2010). Manurung et al. (2012) argue that at a basic level all (machine-generated) poems must satisfy the constraints of grammaticality (i.e., a poem must syntactically well-formed), meaningfulness (i.e., a poem must convey a message that is meaningful under some interpretation) and poeticness (i.e., a poem must exhibit features that distinguishes it from non-poetic text, e.g., meter). Their model generates several candidate poems and then uses stochastic search to find those which are grammatical, meaningful, and poetic.

A third line of research draws inspiration from statistical machine translation (SMT) and related text-generation applications such as summarization. Greene et al. (2010) infer meters (stressed/unstressed syllable sequences) from a corpus of poetic texts which they subsequently use for generation together with a cascade of weighted finite-state transducers interpolated with IBM Model 1. Jiang and Zhou (2008) generate Chinese couplets (two line poems) using a phrase-based SMT approach which translates the first line to the second line. He et al. (2012) extend this algorithm to generate four-line quatrains by sequentially translating the current line from the previous one. Yan et al. (2013) generate Chinese quatrains based on a query-focused summarization framework. Their system takes a few keywords as input and retrieves the most relevant poems from a corpus collection. The retrieved poems are segmented into their

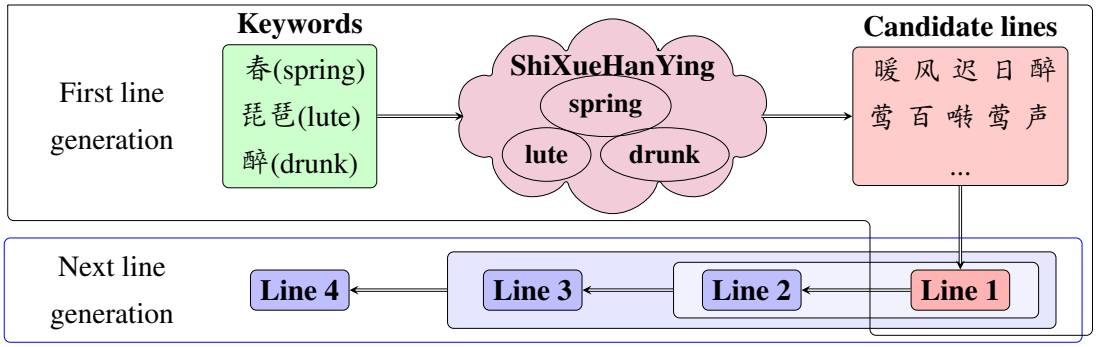


Figure 3.1: Poem generation with keywords *spring*, *lute*, and *drunk*. The keywords are expanded into phrases using a poetic taxonomy. Phrases are then used to generate the first line. Following lines are generated by taking into account the representations of all previously generated lines.

constituent terms which are then grouped into clusters. Poems are generated by iteratively selecting terms from clusters subject to phonological, structural, and coherence constraints.

Our approach departs from previous work in two important respects. Firstly, we model the tasks of surface realization and content selection jointly using recurrent neural networks. Structural, semantic, and coherence constraints are captured naturally in our framework, through learning the representations of individual characters and their combinations. Secondly, generation proceeds by taking into account multi-sentential context rather than the immediately preceding sentence. Our work joins others in using continuous representations to express the meaning of words and phrases (Socher et al., 2012; Mikolov et al., 2013b) and how these may be combined in a language modeling context (Mikolov and Zweig, 2012). More recently, continuous translation models based on recurrent neural networks have been proposed as a means to map a sentence from the source language to sentences in the target language (Auli et al., 2013; Kalchbrenner and Blunsom, 2013). These models are evaluated on the task of rescoring n-best lists of translations. We use neural networks more directly to perform the actual poem generation task.

3.3 The Poem Generator

As common in previous work (Yan et al., 2013; He et al., 2012) we assume that our generator operates in an interactive context. Specifically, the user supplies keywords

(e.g., *spring*, *lute*, *drunk*) highlighting the main concepts around which the poem will revolve. As illustrated in Figure 3.1, our generator expands these keywords into a set of related phrases. We assume the keywords are restricted to those attested in the *ShiXueHanYing* poetic phrase taxonomy (He et al., 2012; Yan et al., 2013). The latter contains 1,016 manual clusters of phrases (Liu, 1735); each cluster is labeled with a keyword id describing general poem-worthy topics. The generator creates the first line of the poem based on these keywords. Subsequent lines are generated based on all previously generated lines, subject to phonological (e.g., admissible tonal patterns) and structural constraints (e.g., whether the quatrain is five or seven characters long).

To create the first line, we select all phrases corresponding to the user's keywords and generate all possible combinations satisfying the tonal pattern constraints. We use a language model to rank the generated candidates and select the best-ranked one as the first line in the poem. In implementation, we employ a character-based recurrent neural network language model (Mikolov et al., 2010) interpolated with a Kneser-Ney trigram and find the n-best candidates with a beam search style stack decoder (see Section 3.3.5 for details). Therefore, once the input keywords are given, the generation of the first line is deterministic[3]. We then generate the second line based on the first one, the third line based on the first two lines, and so on. Our generation model computes the probability of line $S_{i+1} = w_1, w_2, \ldots, w_m$, given all previously generated lines $S_{1:i} (i \geq 1)$ as:

$$P(S_{i+1}|S_{1:i}) = \prod_{j=1}^{m-1} P(w_{j+1}|w_{1:j}, S_{1:i}) \tag{3.1}$$

Equation (3.1), decomposes $P(S_{i+1}|S_{1:i})$ as the product of the probability of each character w_j in the current line given all previously generated characters $w_{1:j-1}$ and lines $S_{1:i}$. This means that $P(S_{i+1}|S_{1:i})$ is sensitive to previously generated content and currently generated characters.

The estimation of the term $P(w_{j+1}|w_{1:j}, S_{1:i})$ lies at the heart of our model. We learn representations for $S_{1:i}$, the context generated so far, using a recurrent neural network whose output serves as input to a second recurrent neural network used to estimate $P(w_{j+1}|w_{1:j}, S_{1:i})$. Figure 3.2 illustrates the generation process for the $(j+1)$th character w_{j+1} in the $(i+1)$th line S_{i+1}. First, lines $S_{1:i}$ are converted into vectors $v_{1:i}$ with a *convolutional sentence model* (CSM; described in Section 3.3.1). Next, a *recurrent context model* (RCM; see Section 3.3.2) takes $v_{1:i}$ as input and outputs u_i^j, the

[3]However, it is also possible to introduce randomness into first line generations. Instead of doing beam search during decoding, we can sample a character at each step.

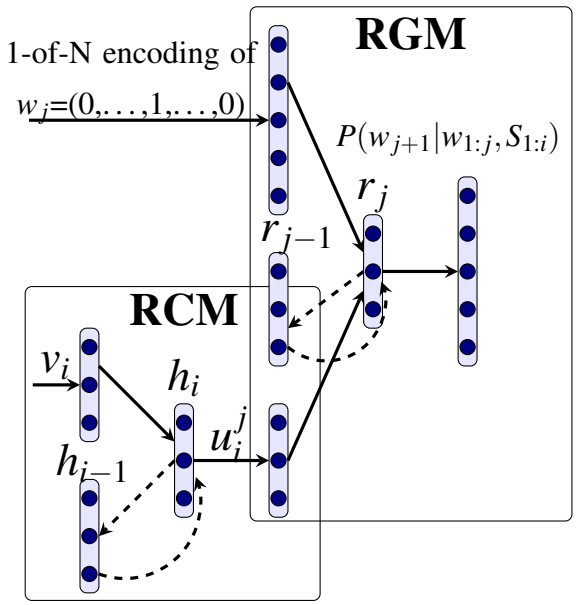


Figure 3.2: Generation of the $(j+1)$th character w_{j+1} in the $(i+1)$th line S_{i+1}. The recurrent context model (RCM) takes i lines as input (represented by vectors $v_1,\ldots,v_i$) and creates context vectors for the recurrent generation model (RGM). The RGM estimates the probability $P(w_{j+1}|w_{1:j},S_{1:i})$.

representation needed for generating $w_{j+1} \in S_{i+1}$. Finally, $u_i^1, u_i^2, \ldots, u_i^j$ and the first j characters $w_{1:j}$ in line S_{i+1} serve as input to a *recurrent generation model* (RGM) which estimates $P(w_{j+1} = k|w_{1:j},S_{1:i})$ with $k \in V$, the probability distribution of the $(j+1)$th character over all characters in the vocabulary V. More formally, to estimate $P(w_{j+1}|w_{1:j},S_{1:i})$ in Equation (3.1), we apply the following procedure:

$$v_i = \text{CSM}(S_i) \tag{3.2a}$$

$$u_i^j = \text{RCM}(v_{1:i}, j) \tag{3.2b}$$

$$P(w_{j+1}|w_{1:j},S_{1:i}) = \text{RGM}(w_{1:j+1}, u_i^{1:j}) \tag{3.2c}$$

We obtain the probability of the $(i+1)$th sentence $P(S_{i+1}|S_{1:i})$, by running the RGM in (3.2c) above $m-1$ times (see also Equation (3.1)). In the following, we describe how the different components of our model are obtained.

3.3.1 Convolutional Sentence Model (CSM)

The CSM converts a poem line into a vector. In principle, any model that produces vector-based representations of phrases or sentences could be used (Mitchell and La-

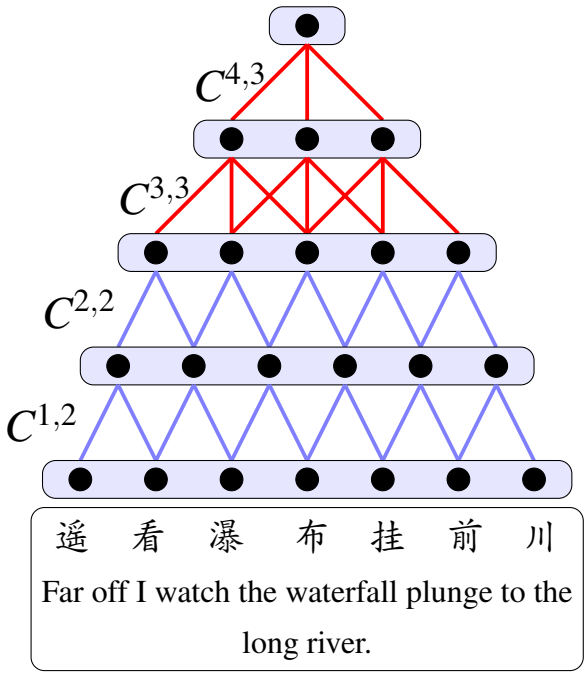


Figure 3.3: Convolutional sentence model for 7-char quatrain. The first layer has seven vectors, one for each character. Two neighboring vectors are merged to one vector in the second layer with weight matrix $C^{1,2}$. In other layers, either two or three neighboring vectors are merged.

pata, 2010; Socher et al., 2012). We opted for the convolutional sentence model proposed in Kalchbrenner and Blunsom (2013) as it is n-gram based and does not make use of any parsing, POS-tagging or segmentation tools which are not available for Chinese poems. Their model computes a continuous representation for a sentence by sequentially merging neighboring vectors (see Figure 3.3).

Let V denote the character vocabulary in our corpus; $L \in \mathbb{R}^{q\times|V|}$ denotes a character embedding matrix whose columns correspond to character vectors (q represents the hidden unit size). Such vectors can be initialized randomly or obtained via a training procedure (Mikolov et al., 2013b). Let w denote a character with index k; $e(w) \in \mathbb{R}^{|V|\times 1}$ is a vector with zero in all positions except $e(w)_k = 1$; $T^l \in \mathbb{R}^{q\times N^l}$ is the sentence representation in the lth layer, where N^l is the number of columns in the lth layer ($N^l = 1$ in the top layer); $C^{l,n^l} \in \mathbb{R}^{q\times n^l}$ is an array of weight matrices which compress neighboring n^l columns in the lth layer to one column in the $(l+1)$th layer. Given a sentence $S = w_1, w_2, \ldots, w_m$, the first layer is represented as:

$$\begin{aligned} T^1 &= [L\cdot e(w_1), L\cdot e(w_2), \ldots, L\cdot e(w_m)] \\ N^1 &= m \end{aligned} \tag{3.3}$$

The $(l+1)$th layer is then computed as follows:

$$\begin{aligned} T^{l+1}_{:,j} &= \sigma(\sum_{i=1}^{n^l} T^l_{:,j+i-1} \odot C^{l,n^l}_{:,i}) \\ N^{l+1} &= N^l - n^l + 1 \\ 1 &\leq j \leq N^{l+1} \end{aligned} \tag{3.4}$$

where T^l is the representation of the previous layer l, C^{l,n^l} a weight matrix[4], $\odot$ element-wise vector product, and σ a non-linear function. We compress two neighboring vectors in the first two layers and three neighboring vectors in the remaining layers (i.e., $n^1 = n^2 = 2, n^3 = n^4 = 3$). Specifically, for quatrains with seven characters, we use $C^{1,2}$, $C^{2,2}$, $C^{3,3}$, $C^{4,3}$ to merge vectors in each layer (see Figure 3.3); and for quatrains with five characters we use $C^{1,2}$, $C^{2,2}$, $C^{3,3}$.

3.3.2 Recurrent Context Model (RCM)

The RCM takes as input the vectors representing the i lines generated so far and reduces them to a single context vector which is then used to generate the next character (see Figure 3.2). We compress the i previous lines to one vector (the hidden layer) and then decode the compressed vector to different character positions in the current line. The output layer consists thus of several vectors (one for each position) connected together. This way, different aspects of the context modulate the generation of different characters. Note that RCM tries to produce dynamic context vectors for different positions and this dynamic context can also be created with the attention mechanism (Bahdanau et al., 2015)[5].

Let $v_1, \ldots, v_i \quad (v_i \in \mathbb{R}^{q\times 1})$ denote the vectors of the previous i lines (they are obtained by the CSM; see Section 3.3.1); $h_i \in \mathbb{R}^{q\times 1}$ is their compressed representation (hidden layer) which is obtained with matrix $M \in \mathbb{R}^{q\times 2q}$; matrix U_j decodes h_i to $u^j_i \in \mathbb{R}^{q\times 1}$ in the $(i+1)$th line. The computation of the RCM proceeds as follows:

$$\begin{aligned} h_0 &= \mathbf{0} \\ h_i &= \sigma(M \cdot \begin{bmatrix} v_i \\ h_{i-1} \end{bmatrix}) \\ u^j_i &= \sigma(U_j \cdot h_i) \qquad 1 \leq j \leq m-1 \end{aligned} \tag{3.5}$$

[4] In some literatures, C^{l,n^l} is also called the kernel matrix.

[5] When we were doing this project, the attention mechanism (Bahdanau et al., 2015) has not yet been invented.

where σ is a non-linear function such as sigmoid and m the line length. Advantageously, lines in classical Chinese poems have a fixed length of five or seven characters. Therefore, the output layer of the recurrent context model only needs two weight matrices (one for each length) and the number of parameters still remains tractable.

3.3.3 Recurrent Generation Model (RGM)

As shown in Figure 3.2, the RGM estimates the probability distribution of the next character (over the entire vocabulary) by taking into account the context vector provided by the RCM and the 1-of-N encoding of the previous character. The RGM is essentially a recurrent neural network language model (Mikolov et al., 2010) with an auxiliary input layer, i.e., the context vector from the RCM. Similar strategies for encoding additional information have been adopted in related language modeling and machine translation work (Mikolov and Zweig, 2012; Kalchbrenner and Blunsom, 2013; Auli et al., 2013).

Let $S_{i+1} = w_1, w_2, \ldots, w_m$ denote the line to be generated. The RGM must estimate $P(w_{j+1}|w_{1:j}, S_{1:i})$, however, since the first i lines have been encoded in the context vector u_i^j, we compute $P(w_{j+1}|w_{1:j}, u_i^j)$ instead. Therefore, the probability $P(S_{i+1}|S_{1:i})$ becomes:

$$P(S_{i+1}|S_{1:i}) = \prod_{j=1}^{m-1} P(w_{j+1}|w_{1:j}, u_i^j) \tag{3.6}$$

Let $|V|$ denote the size of the character vocabulary. The RGM is specified by a number of matrices. Matrix $H \in \mathbb{R}^{q\times q}$ (where q represents the hidden unit size) transforms the context vector to a hidden representation; matrix $X \in \mathbb{R}^{q\times|V|}$ transforms a character to a hidden representation, matrix $R \in \mathbb{R}^{q\times q}$ implements the recurrent transformation and matrix $Y \in \mathbb{R}^{|V|\times q}$ decodes the hidden representation to weights for all words in the vocabulary. Let w denote a character with index k in V; $e(w) \in \mathbb{R}^{|V|\times 1}$ represents a vector with zero in all positions except $e(w)_k = 1$, r_j is the hidden layer of the RGM at step j, and y_{j+1} the output of the RGM, again at step j. The RGM proceeds as follows:

$$r_0 = \mathbf{0} \tag{3.7a}$$

$$r_j = \sigma(R \cdot r_{j-1} + X \cdot e(w_j) + H \cdot u_i^j) \tag{3.7b}$$

$$y_{j+1} = Y \cdot r_j \tag{3.7c}$$

where σ is a nonlinear function (e.g., sigmoid).

The probability of the $(j+1)$th word given the previous j words and the previous i lines is estimated by a *softmax* function:

$$P(w_{j+1} = k | w_{1:j}, u_i^j) = \frac{\exp(y_{j+1,k})}{\sum_{k=1}^{|V|} \exp(y_{j+1,k})} \tag{3.8}$$

We obtain $P(S_{i+1}|S_{1:i})$ by multiplying all the terms in the right hand-side of Equation (3.6).

3.3.4 Training

The objective for training is minimizing the cross entropy errors[6] of the predicted character distribution and the actual character distribution in our corpus. An l_2 regularization term is also added to the objective. The model is trained with back propagation through time (Rumelhart et al., 1988; Werbos, 1988) with sentence length being the time step. The objective is minimized by stochastic gradient descent. During training, the cross entropy error in the output layer of the RGM is back propagated to its hidden and input layers, then to the RCM and finally to the CSM. The same number of hidden units ($q = 200$) is used throughout (i.e., in the RGM, RCM, and CSM). In our experiments all parameters were initialized randomly, with the exception of the word embedding matrix in the CSM which was initialized with *word2vec* embeddings (Mikolov et al., 2013b) obtained from our poem corpus (see Section 3.4 for details on the data we used). The initialization with *word2vec* vectors helps our model reduce the perplexity from 96 to 93 (see perplexity results in Table 3.3).

To speed up training, we employed word-classing (Mikolov et al., 2011b). To compute the probability of a character, we estimate the probability of its class and then multiply it by the probability of the character conditioned on the class. In our experiments we used 82 (square root of $|V|$) classes which we obtained by applying hierarchical clustering on character embeddings. This strategy outperformed better known frequency-based classing methods (Zweig and Makarychev, 2013) on our task.

Simple recurrent neural networks are successful at language modeling (Mikolov et al., 2010), but not that successful at generating words or characters (Kalchbrenner and Blunsom, 2013)[7]. Kalchbrenner and Blunsom (2013) employed recurrent neural networks for machine translation, but their models were used in reranking of `cdec`

[6]The "cross entropy error" is also called negative log-likelihood (NLL).

[7]Later, many researchers found long short-term memory networks (Hochreiter and Schmidhuber, 1997) are better options for sequence or language generation (Sutskever et al., 2014; Vinyals et al., 2015; Venugopalan et al., 2015).

(Dyer et al., 2010) top 1000 candidate translations. Statistical machine translation models (Jiang and Zhou, 2008; He et al., 2012) have been particularly successful at generating poems (especially short poems like couplets). To enhance the generation capability of our model, we thus interpolate our model with three machine translation features (i.e., inverted phrase translation model feature, inverted lexical weight feature and a Kneser-Ney trigram language model). Therefore, during the generation decoding, we have four features (three SMT features and our model probability output) in total for the stack decoder (Koehn et al., 2003). At the time when we were doing our project (2013 to 2014), integrating a neural network with a statistical machine translation model is a very effective way to improve performance for machine translation (Gao et al., 2014; Devlin et al., 2014).

Throughout our experiments, we use the RNNLM toolkit to train the character-based recurrent neural network language model (Mikolov et al., 2010). Kneser-Ney n-grams were trained with KenLM (Heafield, 2011).

3.3.5 Decoding

Our decoder is a stack decoder similar to Koehn et al. (2003). In addition, it implements the tonal pattern and rhyming constraints necessary for generating well-formed Chinese quatrains. Once the first line in a poem is generated, its tonal pattern is determined. During decoding, phrases violating this pattern are ignored. As discussed in Section 3.1, the final characters of the second and the fourth lines must rhyme. We thus remove during decoding fourth lines whose final characters do not rhyme with the second line. Finally, we use MERT training (Och, 2003)[8] to learn feature weights for the decoder.

3.4 Experimental Design

3.4.1 Data

We created a corpus of classical Chinese poems by collating several online resources: *Tang Poems*, *Song Poems*, *Song Ci*, *Ming Poems*, *Qing Poems*, and *Tai Poems*. The corpus consists of 284,899 poems in total. 78,859 of these are quatrains and were

[8]Minimum Error Rate Training (MERT) is an automatic way to tune feature weights for statistical machine translation systems on a development set.

	Poems	Lines	Characters
QTRAIN	74,809	299,236	2,004,460
QVALID	2,000	8,000	48,000
QTEST	2,050	8,200	49,200
POEMLM	280,849	2,711,034	15,624,283

Table 3.2: Dataset partitions of our poem corpus.

used for training and evaluating our model.[9] Table 3.2 shows the different partitions of this dataset (POEMLM) into training (QTRAIN)[10], validation (QVALID) and testing (QTEST). Half of the poems in QVALID and QTEST are 5-char quatrains and the other half are 7-char quatrains. All poems except QVALID and QTEST were used for training the character-based language models (see row POEMLM in Table 3.2). We also trained *word2vec* embeddings on POEMLM. In our experiments, we generated quatrains following the eight most popular tonal patterns according to Wang (2002).

3.4.2 Perplexity Evaluation

Evaluation of machine-generated poetry is a notoriously difficult task. Our evaluation studies were designed to assess Manurung et al.'s (2012) criteria of grammaticality, meaningfulness, and poeticness. As a sanity check, we first measured the perplexity of our model with respect to the goldstandard. Intuitively, a better model should assign larger probability (and therefore lower perplexity) to goldstandard poems.

3.4.3 BLEU-based Evaluation

We also used BLEU to evaluate our model's ability to generate the second, third and fourth line given previous goldstandard lines. A problematic aspect of this evaluation is the need for multiple human-authored references (for a partially generated poem) which we only have one[11]. We obtain references automatically following the method proposed in He et al. (2012). The main idea is that if two lines share a similar topic, the lines following them can be each other's references. Let A and B denote two adjacent lines in a poem, with B following A. Similarly, let line B' follow line A' in another

[9] The data used in our experiments can be downloaded from `http://homepages.inf.ed.ac.uk/mlap/index.php?page=resources`.

[10] Singleton characters in QTRAIN (6,773 in total) were replaced by <R> to reduce data sparsity.

[11] If we use only one reference, all systems will get nearly zero BLEU scores.

poem. If lines A and A' share some keywords in the same cluster in the *Shixuehanying* taxonomy, then B and B' can be used as references for both A and A'. We use this algorithm on the *Tang Poems* section of our corpus to build references for poems in the QVALID and QTEST data sets. As a convention in machine translation community, poems in QVALID (with auto-generated references) were used for MERT training and poems in QTEST (with auto-generated references) were used for BLEU evaluation.

3.4.4 Human Evaluation

Finally, we also evaluated the generated poems by eliciting human judgments. Specifically, we invited 30 experts[12] on Chinese poetry to assess the output of our generator (and comparison systems). These experts were asked to rate the poems using a 1–5 scale on four dimensions: *fluency* (is the poem grammatical and syntactically well-formed?), *coherence* (is the poem thematically and logically structured?), *meaningfulness* (does the poem convey a meaningful message to the reader? does the poem reflect the meaning of input keywords?) and *poeticness* (does the text display the features of a poem?). We also asked our participants to evaluate system outputs by ranking the generated poems relative to each other as a way of determining overall poem quality (Callison-Burch et al., 2012).

Participants rated the output of our model and three comparison systems. These included He et al.'s (2012) SMT-based model (SMT), Yan et al.'s (2013) summarization-based system (SUM), and a random baseline which creates poems by randomly selecting phrases from the *ShiXueHanYing* taxonomy given some keywords as input. We also included human written poems whose content matched the input keywords. All systems were provided with the same keywords (i.e., the same cluster names in the *ShiXueHanYing* taxonomy). In order to compare all models on equal footing, we randomly sampled 30 sets of keywords (with three keywords in each set) and generated 30 quatrains for each system according to two lengths, namely 5-char and 7-char. Overall, we obtained ratings for 300 ($5\times30\times2$) poems.

3.5 Results

The results of our perplexity evaluation are summarized in Table 3.3. We compare our RNN-based poem generator (RNNPG) against Mikolov's (Mikolov et al., 2010)

[12] 27 participants were professional or amateur poets and three were Chinese literature students who had taken at least one class on Chinese poetry composition.

Models	Perplexity
KN5	172
RNNLM	145
RNNPG	**93**

Table 3.3: Perplexities for different models.

Models	$1 \rightarrow 2$		$2 \rightarrow 3$		$3 \rightarrow 4$		Average	
	5-char	7-char	5-char	7-char	5-char	7-char	5-char	7-char
SMT	0.0559	0.0906	0.0410	0.1837	0.0547	**0.1804**	0.0505	0.1516
RNNPG	**0.0561**	**0.1868**	**0.0515**	**0.2102**	**0.0572**	0.1800	**0.0549**	**0.1923**

Table 3.4: BLEU-2 scores on 5-char and 7-char quatrains. Given i goldstandard lines, BLEU-2 scores are computed for the next $(i+1)$th lines.

recurrent neural network language model (RNNLM) and a 5-gram language model with Kneser-Ney smoothing (KN5). All models were trained on QTRAIN and tuned on QVALID. The perplexities were computed on QTEST. Note that the RNNPG estimates the probability of a poem line given at least one previous line. Therefore, the probability of a quatrain assigned by the RNNPG is the probability of the last three lines. For a fair comparison, RNNLM and KN5 only leverage the last three lines of each poem during training, validation and testing. The results in Table 3.3 indicate that the generation ability of the RNNPG is better than KN5 and RNNLM. Note that this perplexity-style evaluation is not possible for models which cannot produce probabilities for gold standard poems. For this reason, other related poem generators (Yan et al., 2013; He et al., 2012) are not included in the table.

The results of our evaluation using BLEU-2 are summarized in Table 3.4. Here, we compare our system against the SMT-based poem generation model of He et al. (2012).[13] Their system is a linear combination of two translation models (one with five features and another one with six). Our model uses three of their features, namely the inverted phrase translation model feature, the lexical weight feature, and a Kneser-Ney trigram feature. Unfortunately, it is not possible to evaluate Yan et al.'s (2013) summarization-based system with BLEU, as it creates poems as a whole and there

[13]Our re-implementation of their system delivered very similar scores to He et al. (2012). For example, we obtained an average BLEU-1 of 0.167 for 5-char quatrains and 0.428 for 7-char quatrains compared to their reported scores of 0.141 and 0.380, respectively.

Models	Fluency		Coherence		Meaning		Poeticness		Rank	
	5-char	7-char	5-char	7-char	5-char	7-char	5-char	7-char	5-char	7-char
Random	2.52	2.18	2.22	2.16	2.02	1.93	1.77	1.71	0.31	0.26
SUM	1.97	1.91	2.08	2.33	1.84	1.98	1.66	1.73	0.25	0.22
SMT	2.81	3.01	2.47	2.76	2.33	2.73	2.08	2.36	0.43	0.53
RNNPG	**4.01****	**3.44***	**3.18****	**3.12***	**3.20****	**3.02**	**2.80****	**2.68***	**0.73****	**0.64***
Human	4.31^{+}	4.19^{++}	3.81^{++}	4.00^{++}	3.61^{+}	3.91^{++}	3.29^{++}	3.49^{++}	0.79	0.84^{++}

Table 3.5: Mean ratings elicited by humans on 5-char and 7-char quatrains. Diacritics ** ($p < 0.01$) and * ($p < 0.05$) indicate our model (RNNPG) is significantly better than all other systems except Human. Diacritics $^{++}$ ($p < 0.01$) and $^{+}$ ($p < 0.05$) indicate Human is significantly better than all other systems.

白鹭窥鱼立,	满怀风月一枝春,
Egrets stood, peeping fishes.	Budding branches are full of romance.
青山照水开.	未见梅花亦可人.
Water was still, reflecting mountains.	Plum blossoms are invisible but adorable.
夜来风不动,	不为东风无此客,
The wind went down by nightfall,	With the east wind comes Spring.
明月见楼台.	世间何处是前身.
as the moon came up by the tower.	Where on earth do I come from?

Table 3.6: Example output produced by our model (RNNPG).

is no obvious way to generate next lines with their algorithm. The BLEU scores in Table 3.4 indicate that, given the same context lines, the RNNPG is better than SMT at generating what to say next. BLEU scores should be, however, viewed with some degree of caution. Aside from being an approximation of human judgment (Callison-Burch et al., 2012), BLEU might be unnecessarily conservative for poem composition which by its very nature is a creative endeavor.

The results of our human evaluation study are shown in Table 3.5. Each column reports mean ratings for a different dimension (e.g., fluency, coherence). Ratings for 5-char and 7-char quatrains are shown separately. The last column reports rank scores for each system (Callison-Burch et al., 2012). In a ranked list of N items ($N = 5$ here), the score of the ith ranked item is $\frac{(N-i)}{(N-1)}$. The numerator indicates how many times a systems won in pairwise comparisons, while the denominator normalizes the score.

With respect to 5-char quatrains, RNNPG is significantly better than Random,

SUM and SMT on fluency, coherence, meaningfulness, poeticness and ranking scores (using a t-test). On all dimensions, human-authored poems are rated as significantly better than machine-generated ones, with the exception of overall ranking. Here, the difference between RNNPG and Human is not significant. We obtain similar results with 7-char quatrains. In general, RNNPG seems to perform better on the shorter poems. The mean ratings are higher and the improvements over other systems are larger. Also notice that the score margins between the human- and machine-written poems become larger for 7-char quatrains. This indicates that the composition of 7-char quatrains is more difficult compared to 5-char quatrains. Table 3.6 shows two example poems (5-char and 7-char) produced by our model which received high scores with respect to poeticness.

Although the RNNPG model used in both BLEU an human evaluation includes additional statistical machine translation features, it outperforms a statistical machine translation model in both evaluations. This indicates the RNNPG probability feature is very important for producing good poems.

Interestingly, poems generated by SUM[14] are given ratings similar to Random. In fact SUM is slightly worse (although not significantly) than Random on all dimensions, with the exception of coherence. In the human study reported in Yan et al. (2013), SUM is slightly better than SMT. There are several reasons for this discrepancy. We used a more balanced experimental design: all systems generated poems from the same keywords which were randomly chosen. We used a larger dataset to train the SMT model compared to Yan et al. (284,899 poems vs 61,960). The Random baseline is not a straw-man; it selects phrases from a taxonomy of meaningful clusters edited by humans and closely related to the input keywords.

3.6 Conclusions

In this chapter we have presented a model for Chinese poem generation based on recurrent neural networks. Our model jointly performs content selection and surface realization by learning representations of individual characters and their combinations within and across poem lines. Previous work on poetry generation has mostly leveraged contextual information of limited length (e.g., one sentence). In contrast, we introduced two recurrent neural networks (the recurrent context model and recurrent

[14] We made a good-faith effort to re-implement their poem generation system. We are grateful to Rui Yan for his help and technical advice.

generation model) which naturally capture multi-sentential content.

Experimental results show that our model yields high-quality poems compared to the state of the art. Perhaps unsurprisingly, our human evaluation study revealed that machine-generated poems lag behind human-generated ones. It is worth bearing in mind that poetry composition is a formidable task for humans, let alone machines. And that the poems against which our output was compared have been written by some of the most famous poets in Chinese history!

To our knowledge, the work described in this chapter is one of the first few work to introduce recurrent neural networks (or neural networks) to natural language generation and the first work to introduce neural networks in poetry generation. We also noticed in recent years there are poetry generation work extending our model with long short-term memory networks and attention mechanism (Yan, 2016; Wang et al., 2016).

Avenues for future work are many and varied. Generate poems across different languages and genres (e.g., English sonnets or Japanese Haiku) would be a possible direction.

Chapter 4

Adding Prior Knowledge to Sequence based Generation

Neural based NLG models are very powerful, when there is a lot of high-quality data. However, there are many NLG tasks for which data is sparse or noisy. When the training data is not sufficient or noisy, prior knowledge for the task becomes important. In some cases, people have an expectation for a task, but maybe because of the training data or limitations of current machine learning models, the trained models do not behave as we expected. In these circumstances, modeling task specific prior knowledge is also important. How to model this kind of prior knowledge effectively is very important for generating good output. In this chapter, we explore the deep reinforcement learning framework for modeling prior knowledge and task specific constraints and test the framework on *sentence simplification*.

The neural based language generation architecture we introduced in chapter 1 has two main components: the *representation* network and the generation network. In this chapter, we actually propose a better way to train these neural networks.

Sentence simplification aims to make sentences easier to read and understand. Most recent approaches draw on insights from machine translation to learn simplification rewrites from monolingual corpora of complex and simple sentences. We address the simplification problem with an encoder-decoder model coupled with a deep reinforcement learning framework. Our model, which we call DRESS (as shorthand for **D**eep **RE**inforcement **S**entence **S**implification), explores the space of possible simplifications while learning to optimize a reward function that encourages outputs which are simple, fluent, and preserve the meaning of the input. Experiments on three datasets

demonstrate that our model outperforms competitive simplification systems.[1]

4.1 Introduction

The main goal of *sentence simplification* is to reduce the linguistic complexity of text, while still retaining its original information and meaning. The simplification task has been the subject of several modeling efforts in recent years due to its relevance for NLP applications and individuals alike (Siddharthan, 2014; Shardlow, 2014). For instance, a simplification component could be used as a preprocessing step to improve the performance of parsers (Chandrasekar et al., 1996), summarizers (Beigman Klebanov et al., 2004), and semantic role labelers (Vickrey and Koller, 2008; Woodsend and Lapata, 2014). Automatic simplification would also benefit people with low-literacy skills (Watanabe et al., 2009), such as children and non-native speakers as well as individuals with autism (Evans et al., 2014), aphasia (Carroll et al., 1999), or dyslexia (Rello et al., 2013).

The most prevalent rewrite operations which give rise to simplified text include substituting rare words with more common words or phrases, rendering syntactically complex structures simpler, and deleting elements of the original text (Siddharthan, 2014). Earlier work focused on individual aspects of the simplification problem. For example, several systems performed syntactic simplification only, using rules aimed at sentence splitting (Carroll et al., 1999; Chandrasekar et al., 1996; Vickrey and Koller, 2008; Siddharthan, 2004) while others turned to lexical simplification by substituting difficult words with more common WordNet synonyms or paraphrases (Devlin, 1999; Inui et al., 2003; Kaji et al., 2002).

Recent approaches view the simplification process more holistically as a monolingual text-to-text generation task borrowing ideas from statistical machine translation. Simplification rewrites are learned automatically from examples of complex-simple sentences extracted from online resources such as the ordinary and simple English Wikipedia. For example, Zhu et al. (2010) draw inspiration from syntax-based translation and propose a model similar to Yamada and Knight (2001) which additionally performs simplification-specific rewrite operations (e.g., sentence splitting). Woodsend and Lapata (2011) formulate simplification in the framework of Quasi-synchronous grammar (Smith and Eisner, 2006) and use integer linear programming to score the candidate translations/simplifications. Wubben et al. (2012) propose a two-

[1] Our code and data are publicly available at `https://github.com/XingxingZhang/dress`.

stage model: initially, a standard phrase-based machine translation (PBMT) model is trained on complex-simple sentence pairs. During inference, the K-best outputs of the PBMT model are reranked according to their dis-similarity to the (complex) input sentence. The hybrid model developed in Narayan and Gardent (2014) also operates in two phases. Initially, a probabilistic model performs sentence splitting and deletion operations over discourse representation structures assigned by Boxer (Curran et al., 2007). The resulting sentences are further simplified by a model similar to Wubben et al. (2012). Xu et al. (2016) train a syntax-based machine translation model on a large scale paraphrase dataset (Ganitkevitch et al., 2013) using simplification-specific objective functions and features to encourage simpler output.

In this paper we propose a simplification model which draws on insights from neural machine translation (Bahdanau et al., 2015; Sutskever et al., 2014). Central to this approach is an encoder-decoder architecture implemented by recurrent neural networks. The encoder reads the source sequence into a list of continuous-space representations from which the decoder generates the target sequence. Although our model uses the encoder-decoder architecture as its backbone, it must also meet constraints imposed by the simplification task itself, i.e., the predicted output must be simpler, preserve the meaning of the input, and be grammatical. To incorporate this knowledge, the model is trained in a reinforcement learning framework (Williams, 1992): it explores the space of possible simplifications while learning to maximize an expected reward function that encourages outputs which meet simplification-specific constraints. Reinforcement learning has been previously applied to extractive summarization (Ryang and Abekawa, 2012), information extraction (Narasimhan et al., 2016), dialogue generation (Li et al., 2016), machine translation, and image caption generation (Ranzato et al., 2016).

We evaluate our system on three publicly available datasets collated automatically from Wikipedia (Zhu et al., 2010; Woodsend and Lapata, 2011) and human-authored news articles (Xu et al., 2015b). We experimentally show that the reinforcement learning framework is the key to successful generation of simplified text bringing significant improvements over strong simplification models across datasets.

4.2 Neural Encoder-Decoder Model

We will first define a basic encoder-decoder model for sentence simplification and then explain how to embed it in a reinforcement learning framework. Given a (complex)

source sentence $X = (x_1, x_2, \ldots, x_{|X|})$, our model learns to predict its simplified *target* $Y = (y_1, y_2, \ldots, y_{|Y|})$. Inferring the target Y given the source X is a typical sequence to sequence learning problem, which can be modeled with attention-based encoder-decoder models (Bahdanau et al., 2015; Luong et al., 2015b). Sentence simplification is slightly different from related sequence transduction tasks (e.g., compression) in that it can involve splitting operations. For example, a long source sentence (*In 1883, Faur married Marie Fremiet, with whom he had two sons.*) can be simplified as two sentences (*In 1883, Faur married Marie Fremiet. They had two sons.*). Nevertheless, we still view the target as a sequence, i.e., two or more sequences concatenated with full stops.

The encoder-decoder model has two parts (see left hand side in Figure 4.1). The *encoder* transforms the source sentence X into a sequence of hidden states $(\mathbf{h}_1^S, \mathbf{h}_2^S, \ldots, \mathbf{h}_{|X|}^S)$ with a Long Short-Term Memory Network (LSTM; Hochreiter and Schmidhuber 1997), while the *decoder* uses another LSTM to generate one word y_{t+1} at a time in the simplified target Y. Generation is conditioned on all previously generated words $y_{1:t}$ and a dynamically created context vector $\mathbf{c}_t$, which encodes the source sentence:

$$P(Y|X) = \prod_{t=1}^{|Y|} P(y_t|y_{1:t-1}, X) \tag{4.1}$$

$$P(y_{t+1}|y_{1:t}, X) = \text{softmax}(g(\mathbf{h}_t^T, \mathbf{c}_t)) \tag{4.2}$$

where $g(\cdot)$ is a one-hidden-layer neural network with the following parametrization:

$$g(\mathbf{h}_t^T, \mathbf{c}_t) = \mathbf{W}_o \tanh(\mathbf{U}_h \mathbf{h}_t^T + \mathbf{W}_h \mathbf{c}_t) \tag{4.3}$$

where $\mathbf{W}_o \in \mathbb{R}^{|V| \times d}$, $\mathbf{U}_h \in \mathbb{R}^{d \times d}$, and $\mathbf{W}_h \in \mathbb{R}^{d \times d}$; $|V|$ is the output vocabulary size and d the hidden unit size. $\mathbf{h}_t^T$ is the hidden state of the decoder LSTM which summarizes $y_{1:t}$, i.e., what has been generated so far:

$$\mathbf{h}_t^T = \text{LSTM}(y_t, \mathbf{h}_{t-1}^T) \tag{4.4}$$

The dynamic context vector $\mathbf{c}_t$ is the weighted sum of the hidden states of the source sentence:

$$\mathbf{c}_t = \sum_{i=1}^{|X|} \alpha_{ti} \mathbf{h}_i^S \tag{4.5}$$

whose weights α_{ti} are determined by an *attention* mechanism:

$$\alpha_{ti} = \frac{\exp(\mathbf{h}_t^T \cdot \mathbf{h}_i^S)}{\sum_i \exp(\mathbf{h}_t^T \cdot \mathbf{h}_i^S)} \tag{4.6}$$

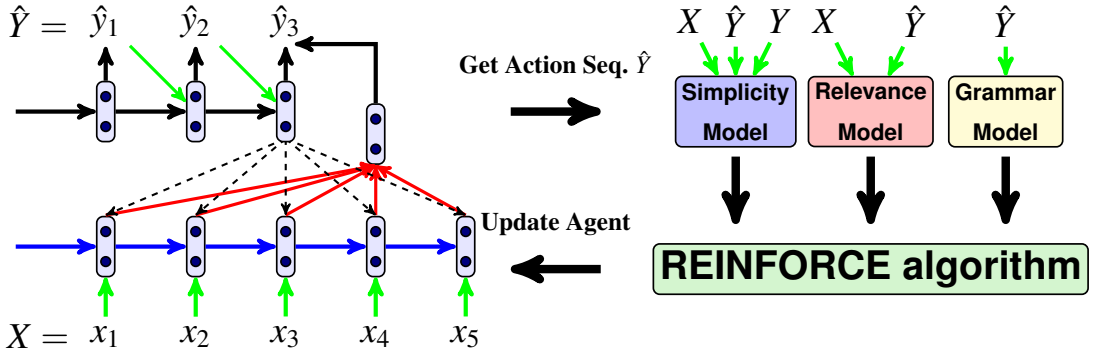


Figure 4.1: Deep reinforcement learning simplification model. X is the complex sentence, Y the reference (simple) sentence and $\hat{Y}$ the action sequence (simplification) produced by the encoder-decoder model.

where $\cdot$ is the dot product between two vectors. We use the dot product here mainly for efficiency reasons; alternative ways to compute attention scores have been proposed in the literature and we refer the interested reader to Luong et al. (2015b). The model sketched above is usually trained by minimizing the negative log-likelihood of the training source-target pairs.

4.3 Reinforcement Learning for Sentence Simplification

In this section we present DRESS, our **D**eep **RE**inforcement **S**entence **S**implification model. Despite successful application in numerous sequence transduction tasks (Jean et al., 2015; Chopra et al., 2016; Xu et al., 2015a), a vanilla encoder-decoder model is not ideal for sentence simplification. Although a number of rewrite operations (e.g., copying, deletion, substitution, word reordering) can be used to simplify text, copying is by far the most common. We empirically found that 73% of the target words are copied from the source in the Newsela dataset. This number further increases to 83% when considering Wikipedia-based datasets (we provide details on these datasets in Section 4.5.1). As a result, a generic encoder-decoder model learns to copy all too well at the expense of other rewrite operations, often parroting back the source or making only a few trivial changes.

To encourage a wider variety of rewrite operations while remaining fluent and faithful to the meaning of the source, we employ a reinforcement learning framework (see Figure 4.1). We view the encoder-decoder model as an agent which first reads the source sentence X; then at each step, it takes an action $\hat{y}_t \in V$ (where V is the output

vocabulary) according to a policy $P_{RL}(\hat{y}_t|\hat{y}_{1:t-1},X)$ (see Equation (4.2)). The agent continues to take actions until it produces an **E**nd **O**f **S**entence (EOS) token yielding the action sequence $\hat{Y} = (\hat{y}_1,\hat{y}_2,\ldots,\hat{y}_{|\hat{Y}|})$, which is also the simplified output of our model. A reward r is then received and the REINFORCE algorithm (Williams, 1992) is used to update the agent[2]. In the following, we first introduce our reward and then present the details of the REINFORCE algorithm.

4.3.1 Reward

The reward $r(\hat{Y})$ for system output $\hat{Y}$ is the weighted sum of the three components aimed at capturing key aspects of the target output, namely simplicity, relevance, and fluency:

$$r(\hat{Y}) = \lambda^S r^S + \lambda^R r^R + \lambda^F r^F \tag{4.7}$$

where $\lambda^S,\lambda^R,\lambda^F \in [0,1]$; $r(\hat{Y})$ is a shorthand for $r(X,Y,\hat{Y})$ where X is the source, Y the reference (or target), and $\hat{Y}$ the system output. r^S, r^R, and r^F are shorthands for simplicity $r^S(X,Y,\hat{Y})$, relevance $r^R(X,\hat{Y})$, and fluency $r^F(\hat{Y})$. We provide details for each reward summand below.

4.3.1.1 Simplicity

To encourage the model to apply a wide range of simplification operations, we use SARI (Xu et al., 2016), a recently proposed metric which compares **S**ystem output **A**gainst **R**eferences and against the **I**nput sentence. SARI is the arithmetic average of n-gram precision and recall of three rewrite operations: addition (*add*), copying (*keep*), and deletion (*del*). It rewards addition operations where system output was not in the input but occurred in the references. Analogously, it rewards words retained/deleted in both the system output and the references. Specifically, SARI is defined as follows:

$$SARI = d_1 F_{add} + d_2 F_{keep} + d_3 P_{del} \tag{4.8}$$

where $d_1 = d_2 = d_3 = \frac{1}{3}$ and

$$P_{operation} = \frac{1}{k} \sum_{n=[1,2,\ldots,k]} p_{opteration}(n) \tag{4.9}$$

[2]In some reinforcement environments (e.g., Atari games), an agent can receive an immediate reward after each action. But here our reward is delayed and our agent only receive rewards after the action that produces an EOS token.

$$R_{operation} = \frac{1}{k} \sum_{n=[1,2,\ldots,k]} r_{opteration}(n) \tag{4.10}$$

$$F_{operation} = \frac{2 \times P_{operation} \times R_{operation}}{P_{operation} + R_{operation}} \tag{4.11}$$

$$operation \in [del, keep, add] \tag{4.12}$$

As in BLEU, k is the highest $n$gram order and is set to 4.

The $n$gram precision and recall of the addition operation ($p_{add}(n)$ and $r_{add}(n)$) are defined as follows:

$$p_{add}(n) = \frac{\sum_{g \in \hat{Y}} \min(\max(0, \#_g(\hat{Y}) - \#_g(X)), \#_g(Y))}{\sum_{g \in \hat{Y}} \max(0, \#_g(\hat{Y}) - \#_g(X))} \tag{4.13}$$

$$r_{add}(n) = \frac{\sum_{g \in \hat{Y}} \min(\max(0, \#_g(\hat{Y}) - \#_g(X)), \#_g(Y))}{\sum_{g \in \hat{Y}} \max(0, \#_g(Y) - \#_g(X))} \tag{4.14}$$

where n is the $n$gram order and $\#_g(\cdot)$ is the occurrence of $n$grams g in a given set.

The $n$gram precision and recall of the copy operation ($p_{keep}(n)$ and $r_{keep}(n)$) are defined as follows:

$$p_{keep}(n) = \frac{\sum_{g \in X} \min(\min(\#_g(X), \#_g(\hat{Y})), \min(\#_g(X), \#_g(Y)))}{\sum_{g \in X} \min(\#_g(X), \#_g(\hat{Y}))} \tag{4.15}$$

$$r_{keep}(n) = \frac{\sum_{g \in X} \min(\min(\#_g(X), \#_g(\hat{Y})), \min(\#_g(X), \#_g(Y)))}{\sum_{g \in X} \min(\#_g(X), \#_g(Y))} \tag{4.16}$$

The $n$gram precision of the deletion operation ($p_{del}(n)$) is defined as follows:

$$p_{del}(n) = \frac{\sum_{g \in X} \min(\max(\#_g(X) - \#_g(\hat{Y}), 0)), \max(\#_g(X) - \#_g(Y), 0))}{\sum_{g \in X} \max(\#_g(X) - \#_g(\hat{Y}), 0)} \tag{4.17}$$

Note that SARI scores are not symmetric. For example, assume the original sentence is "About 95 species are currently accepted .", the reference is "95 species are now accepted ." and the system output is "About 95 you now get in .". $\text{SARI}(X, \hat{Y}, Y) = 33.52$ and $\text{SARI}(X, \hat{Y}, Y) = 33.47$.

In experimental evaluation Xu et al. (2016) demonstrate that SARI correlates well with human judgments of simplicity, whilst correctly rewarding systems that both make changes and simplify the input.

One caveat with using SARI as a reward is the fact that it relies on the availability of multiple references which are rare for sentence simplification. Xu et al. (2016) provide eight references for 2,350 sentences, but these are primarily for system tuning and evaluation rather than training. The majority of existing simplification datasets

(see Section 4.5.1 for details) have a single reference for each source sentence. Moreover, they are unavoidably noisy as they are mostly constructed automatically, e.g., by aligning sentences from the ordinary and simple English Wikipedias. When relying solely on a single reference, SARI will try to reward accidental n-grams that should never have occurred in it. To counter the effect of noise, we apply $\text{SARI}(X,\hat{Y},Y)$ in the expected direction, with X as the source, $\hat{Y}$ the system output, and Y the reference as well as in the reverse direction with Y as the system output and $\hat{Y}$ as the reference. Assuming our system can produce reasonably good simplifications, by swapping the output and the reference, reverse SARI can be used to estimate how good a reference is with respect to the system output. Our first reward is therefore the weighted sum of SARI and reverse SARI:

$$r^S = \beta\,\text{SARI}(X,\hat{Y},Y) + (1-\beta)\,\text{SARI}(X,Y,\hat{Y}) \tag{4.18}$$

4.3.1.2 Relevance

While the simplicity-based reward r^S tries to encourage the model to make changes, the relevance reward r^R ensures that the generated sentences preserve the meaning of the source. We use a LSTM sentence encoder to convert the source X and the predicted target $\hat{Y}$ into two vectors $\mathbf{q}_X$ and $\mathbf{q}_{\hat{Y}}$ (both $\mathbf{q}_X$ and $\mathbf{q}_{\hat{Y}}$ are last hidden states of a LSTM sentence encoder). The relevance reward r^R is simply the cosine similarity between these two vectors:

$$r^R = \cos(\mathbf{q}_X, \mathbf{q}_{\hat{Y}}) = \frac{\mathbf{q}_X \cdot \mathbf{q}_{\hat{Y}}}{||\mathbf{q}_X||\,||\mathbf{q}_{\hat{Y}}||} \tag{4.19}$$

We use a sequence auto-encoder (SAE; Dai and Le 2015) to train the LSTM sentence encoder on both the complex and simple sentences. Specifically, the SAE uses sentence $X = (x_1,\ldots,x_{|X|})$ to infer itself via an encoder-decoder model (without an attention mechanism). Firstly, an encoder LSTM converts X into a sequence of hidden states $(\mathbf{h}_1,\ldots,\mathbf{h}_{|X|})$. Then, we use $\mathbf{h}_{|X|}$ to initialize the hidden state of the decoder LSTM and recover/generate X one word at a time.

4.3.1.3 Fluency

Xu et al. (2016) observe that SARI correlates less with fluency compared to other metrics such as BLEU (Papineni et al., 2002). The fluency reward r^F models the well-formedness of the generated sentences explicitly. It is the normalized sentence

probability assigned by an LSTM language model trained on simple sentences:

$$r^F = \exp\left(\frac{1}{|\hat{Y}|}\sum_{i=1}^{|\hat{Y}|}\log P_{LM}(\hat{y}_i|\hat{y}_{0:i-1})\right) \tag{4.20}$$

We take an exponential outside to ensure that $r^F \in [0,1]$ as is the case with r^S and r^R.

Note that we did not use FKGL (Flesch-Kincaid Grade Level) or BLEU as our reward, although we used them in evaluation. There are several reasons for this. FKGL only measures the simplicity of output sentences and totally ignores their relevance to original sentences. But SARI takes both into account. Apart from the three rewards we introduced earlier, we also additional tried FKGL as part of our reward, but we did not observe extra performance gain. BLEU is not a good evaluation metric for simplification and it conflicts with SARI. Xu et al. (2016) shows that the original complex sentences yield highest BLEU (but very low human evaluation scores) compared to other simplification systems.

4.3.2 The REINFORCE Algorithm

The goal of the REINFORCE algorithm is to find an agent that maximizes the expected reward. The training loss for one sequence is its negative expected reward:

$$\mathcal{L}(\theta) = -\mathbb{E}_{(\hat{y}_1,\ldots,\hat{y}_{|\hat{Y}|})\sim P_{RL}(\cdot|X)}[r(\hat{y}_1,\ldots,\hat{y}_{|\hat{Y}|})] \tag{4.21}$$

where P_{RL} is our policy, i.e., the distribution produced by the encoder-decoder model (see Equation(4.2)) and $r(\cdot)$ is the reward function of an action sequence $\hat{Y} = (\hat{y}_1,\ldots,\hat{y}_{|\hat{Y}|})$, i.e., a generated simplification. Unfortunately, computing the expectation term is prohibitive, since there is an infinite number of possible action sequences. In practice, we approximate this expectation with a single sample from the distribution of $P_{LR}(\cdot|X)$. We refer to Williams (1992) for the full derivation of the gradients. The gradient of $\mathcal{L}(\theta)$ is:

$$\nabla\mathcal{L}(\theta) \approx$$
$$\sum_{t=1}^{|\hat{Y}|}\nabla\log P_{RL}(\hat{y}_t|\hat{y}_{1:t-1},X)[r(\hat{y}_{1:|\hat{Y}|}) - b_t]$$

To reduce the variance of gradients, we also introduce a baseline linear regression model b_t to estimate the expected future reward at time t (Ranzato et al., 2016). b_t takes the concatenation of $\mathbf{h}_t^T$ and $\mathbf{c}_t$ as input and outputs a real value as the expected reward. The parameters of the regressor are trained by minimizing mean squared error. We do not back-propagate this error to $\mathbf{h}_t^T$ or $\mathbf{c}_t$ during training (Ranzato et al., 2016).

4.3.3 Learning

Presented in its original form, the REINFORCE algorithm starts learning with a random policy. This assumption can make model training challenging for generation tasks like ours with large vocabularies (i.e., action spaces). We address this issue by pre-training our agent (i.e., the encoder-decoder model) with a negative log-likelihood objective (see Section 4.2), making sure it can produce reasonable simplifications, thereby starting off with a policy which is better than random. We follow prior work (Ranzato et al., 2016) in adopting a curriculum learning strategy. In the beginning of training, we give little freedom to our agent allowing it to predict the last few words for each target sentence. For every target sequence, we use negative log-likelihood to train the first L (initially, $L = 24$) tokens and apply the reinforcement learning algorithm to the $(L+1)$th tokens onwards. Every two epochs, we set $L = L-3$ and the training terminates when L is 0.

4.4 Lexical Simplification

Lexical substitution, the replacement of complex words with simpler alternatives, is an integral part of sentence simplification (Specia et al., 2012). The model presented so far learns lexical substitution and other rewrite operations *jointly*. In some cases, words are predicted because they seem natural in the their context, but are poor substitutes for the content of the complex sentence. To counter this, we learn lexical simplifications *explicitly* and integrate them with our reinforcement learning-based model.

We use an *pre-trained* encoder-decoder model (which is trained on a parallel corpus of complex and simple sentences) to obtain probabilistic word alignments, aka attention scores (see α_t in Equation (4.6)). Let $X = (x_1, x_2, \ldots, x_{|X|})$ denote a source sentence and $Y = (y_1, y_2, \ldots, y_{|Y|})$ a target sentence. We convert X into $|X|$ hidden states $(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{|X|})$ with an LSTM. Note that $\mathbf{v}_t \in \mathbb{R}^{d\times 1}$ corresponds to the context dependent representation of x_t. Let α_t denote the alignment scores $\alpha_{t1}, \alpha_{t2}, \ldots, \alpha_{t|X|}$. The lexical simplification probability of y_t given the source sentence and the alignment scores is:

$$P_{LS}(y_t|X, \alpha_t) = \text{softmax}(\mathbf{W}_l \mathbf{s}_t) \tag{4.22}$$

where $\mathbf{W}_l \in \mathbb{R}^{|V|\times d}$ and $\mathbf{s}_t$ represents the source:

$$\mathbf{s}_t = \sum_{i=1}^{|X|} \alpha_{ti} \mathbf{v}_i \tag{4.23}$$

The lexical simplification model on its own encourages lexical substitutions, without taking into account what has been generated so far (i.e., $y_{1:t-1}$) and as a result fluency could be compromised. A straightforward solution is to integrate lexical simplification with our reinforcement learning trained model (Section 4.3) using linear interpolation, where $\eta \in [0,1]$:

$$\begin{aligned} P(y_t|y_{1:t-1},X) = (1-\eta)\, P_{RL}(y_t|y_{1:t-1},X) \\ +\eta\, P_{LS}(y_t|X,\alpha_t) \end{aligned} \tag{4.24}$$

4.5 Experimental Setup

In this section we present our experimental setup for assessing the performance of the simplification model described above. We give details on our datasets, model training, evaluation protocol, and the systems used for comparison.

4.5.1 Datasets

We conducted experiments on three simplification datasets. *WikiSmall* Zhu et al. (2010) is a parallel corpus which has been extensively used as a benchmark for evaluating text simplification systems (Wubben et al., 2012; Woodsend and Lapata, 2011; Narayan and Gardent, 2014; Zhu et al., 2010). It contains automatically aligned complex and simple sentences from the ordinary and simple English Wikipedias. The test set consists of 100 complex-simple sentence pairs. The training set contains 89,042 sentence pairs (after removing duplicates and test sentences). We randomly sampled 205 pairs for development and used the remaining sentences for training.

We also constructed *WikiLarge*, a larger Wikipedia corpus by combining previously created simplification corpora. Specifically, we aggregated the aligned sentence pairs in Kauchak (2013), the aligned and revision sentence pairs in Woodsend and Lapata (2011), and Zhu's (2010) WikiSmall dataset described above. We used the development and test sets created in Xu et al. (2016). These are complex sentences taken from WikiSmall paired with simplifications provided by Amazon Mechanical Turk workers. The dataset contains 8 (reference) simplifications for 2,359 sentences partitioned into 2,000 for development and 359 for testing. After removing duplicates and sentences in development and test sets, the resulting training set contains 296,402 sentence pairs.

Our third dataset is *Newsela*, a corpus collated by Xu et al. (2015b) who argue

that Wikipedia-based resources are suboptimal due to the automatic sentence alignment which unavoidably introduces errors, and their uniform writing style which leads to systems that generalize poorly. Newsela[3] consists of 1,130 news articles, each rewritten four times by professional editors for children at different grade levels (0 is the most complex level and 4 is simplest). Xu et al. (2015b) provide multiple aligned complex-simple pairs within each article. We removed sentence pairs corresponding to levels 0–1, 1–2, and 2–3, since they were too similar to each other. The first 1,070 documents were used for training (94,208 sentence pairs), the next 30 documents for development (1,129 sentence pairs) and the last 30 documents for testing (1,076 sentence pairs).[4] We are not aware of any published results on this dataset.

4.5.2 Training Details

We trained our models on an Nvidia GPU card. We used the same hyper-parameters across datasets. We first trained an encoder-decoder model, and then performed reinforcement learning training (Section 4.3), and trained the lexical simplification model (Section 4.4). Encoder-decoder parameters were uniformly initialized to $[-0.1, 0.1]$. We used Adam (Kingma and Ba, 2014) to optimize the model with learning rate 0.001; the first momentum coefficient was set to 0.9 and the second momentum coefficient to 0.999. The gradient was rescaled when the norm exceeded 5 (Pascanu et al., 2013). Both encoder and decoder LSTMs have two layers with 256 hidden neurons in each layer. We regularized all LSTMs with a dropout rate of 0.2 (Zaremba et al., 2014). We initialized the encoder and decoder word embedding matrices with 300 dimensional Glove vectors (Pennington et al., 2014).

During reinforcement training, we used plain stochastic gradient descent with a learning rate of 0.01. We set $\beta = 0.1$, $\lambda_S = 1$, $\lambda_R = 0.25$ and $\lambda_F = 0.5$.[5] Training details for the lexical simplification model are identical to the encoder-decoder model except that word embedding matrices were randomly initialized. The weight of the lexical simplification model was set to $\eta = 0.1$.

To reduce vocabulary size, named entities were tagged with the Stanford CoreNLP (Manning et al., 2014) and anonymized with a NE@N token, where NE $\in \{$PER, LOC, ORG, MISC$\}$ and N indicates NE@N is the N-th distinct NE typed entity. For exam-

[3] https://newsela.com

[4] If a sentence has multiple references in the development or test set, we use the reference with highest simplicity level.

[5] Weights were tuned on the development set of the Newsela dataset and kept fixed for the other two datasets.

ple, "John and Bob are ..." becomes "PER@1 and PER@2 are ...". At test time, we de-anonymize NE@*N* tokens in the output by looking them up in their source sentences. Note that the de-anonymization may fail, but the chance is small (around 2% of the time on the Newsela development set). We replaced words occurring three times or less in the training set with UNK. At test time, when our models predict UNK, we adopt the UNK replacement method proposed in Jean et al. (2015) (i.e., we replace UNK with the word in the complex sentence which is assigned the largest attention score).

4.5.3 Evaluation

Following previous work (Woodsend and Lapata, 2011; Xu et al., 2016) we evaluated system output automatically adopting metrics widely used in the simplification literature. Specifically, we used BLEU[6] (Papineni et al., 2002) to assess the degree to which generated simplifications differed from gold standard references and the Flesch-Kincaid Grade Level index (FKGL; Kincaid et al. 1975) to measure the readability of the output (lower FKGL[7] implies simpler output). In addition, we used SARI (Xu et al., 2016), which evaluates the quality of the output by comparing it against the source and reference simplifications.[8] As a convention in machine translation community, BLEU, FKGL, and SARI are all measured at corpus level[9]. Because average of scores at sentence level may bias to shorter sentences. We also evaluated system output by eliciting human judgments via Amazon's Mechanical Turk. Specifically (self-reported) native English speakers were asked to rate simplifications on three dimensions: *Fluency* (is the output grammatical and well formed?), *Adequacy* (to what extent is the meaning expressed in the original sentence preserved in the output?) and *Simplicity* (is the output simpler than the original sentence?). All ratings were obtained using a five point Likert scale.

4.5.4 Comparison Systems

We compared our model against several systems previously proposed in the literature. These include PBMT-R, a monolingual phrase-based machine translation system

[6]With the default `mtevalv13a.pl` settings.
[7]FKGL implementation at `http://goo.gl/OHP7k3`.
[8]We used he implementation of SARI in Xu et al. (2016).
[9]We have to use sentence level SARI during training because reward is measured for each sequence.

with a reranking post-processing step[10] (Wubben et al., 2012) and Hybrid, a model which first performs sentence splitting and deletion operations over discourse representation structures and then further simplifies sentences with PBMT-R (Narayan and Gardent, 2014). Hybrid[11] is state of the art on the WikiSmall dataset. Comparisons with SBMT-SARI, a syntax-based translation model trained on PPDB (Ganitkevitch et al., 2013) and tuned with SARI (Xu et al., 2016), are problematic due to the size of PPDB which is considerably larger than any of the datasets used in this work (it contains 106 million sentence pairs with 2 billion words). Nevertheless, we compare[12] against SBMT-SARI, but only models trained on Wikilarge, our largest dataset.

4.6 Results

Since Newsela contains high quality simplifications created by professional editors, we performed the bulk of our experiments on this dataset. Specifically, we set out to answer two questions: (a) which neural model performs best and (b) how do neural models which are resource lean and do not have access to linguistic annotations fare against more traditional systems. We therefore compared the basic attention-based encoder-decoder model (EncDecA), with the deep reinforcement learning model (DRESS; Section 4.3), and a linear combination of DRESS and the lexical simplification model (DRESS-LS; Section 4.4). Neural models were further compared against two strong baselines, PBMT-R and Hybrid. Table 4.4 shows example output of all models on the Newsela dataset.

The top block in Table 4.1 summarizes the results of our automatic evaluation. As can be seen, all neural models obtain higher BLEU, lower FKGL and higher SARI compared to PBMT-R. Hybrid has the lowest FKGL and highest SARI. Compared to EncDecA, DRESS scores lower on FKGL and higher on SARI, which indicates that the model has indeed learned to optimize the reward function which includes SARI. Integrating lexical simplification (DRESS-LS) yields better BLEU, but slightly worse FKGL and SARI.

The results of our human evaluation are presented in the top block of Table 4.2. We elicited judgments for 100 randomly sampled test sentences. Aside from comparing system output (PBMT-R, Hybrid, EncDecA, DRESS, and DRESS-LS), we also elicited

[10] We made a good-faith effort to re-implement their system following closely the details in Wubben et al. (2012).

[11] We are grateful to Shashi Narayan for running his system on our three datasets.

[12] The output of SBMT-SARI is publicly available.

Newsela	BLEU	FKGL	SARI
PBMT-R	18.19	7.59	15.77
Hybrid	14.46	**4.01**	**30.00**
EncDecA	21.70	5.11	24.12
DRESS	23.21	4.13	27.37
DRESS-LS	**24.30**	4.21	26.63

WikiSmall	BLEU	FKGL	SARI
PBMT-R	46.31	11.42	15.97
Hybrid	**53.94**	9.20	**30.46**
EncDecA	47.93	11.35	13.61
DRESS	34.53	**7.48**	27.48
DRESS-LS	36.32	7.55	27.24

WikiLarge	BLEU	FKGL	SARI
PBMT-R	81.11	8.33	38.56
Hybrid	48.97	**4.56**	31.40
SBMT-SARI	73.08	7.29	**39.96**
EncDecA	**88.85**	8.41	35.66
DRESS	77.18	6.58	37.08
DRESS-LS	80.12	6.62	37.27

Table 4.1: Automatic evaluation on Newsela, WikiSmall, and WikiLarge test sets.

ratings for the gold standard Reference as an upper bound. We report results for Fluency, Adequacy, and Simplicity individually and in combination (All is the average rating of the three dimensions). As can be seen, DRESS and DRESS-LS outperform PBMT-R and Hybrid on Fluency, Simplicity, and overall. The fact that neural models (EncDecA, DRESS and DRESS-LS) fare well on Fluency, is perhaps not surprising given the recent success of LSTMs in language modeling and neural machine translation (Zaremba et al., 2014; Jean et al., 2015).

Neural models obtain worse ratings on Adequacy but are closest to the human references on this dimension. DRESS-LS (and DRESS) are significantly better ($p < 0.01$) on Simplicity than EncDecA, PBMT-R, and Hybrid which indicates that our reinforcement learning based model is effective at creating simpler output. Combined ratings (All) for DRESS-LS are significantly different compared to the other models but not to DRESS and the Reference. Nevertheless, integration of the lexical simplification

Newsela	Fluency	Adequacy	Simplicity	All
PBMT-R	3.56	**3.58****	2.09**	3.08**
Hybrid	2.70**	2.51**	2.99	2.73**
EncDecA	3.63	2.99	2.56**	3.06**
DRESS	3.65	2.94	**3.10**	3.23
DRESS-LS	**3.71**	3.07	3.04	**3.28**
Reference	3.90	2.81**	3.42**	3.38

WikiSmall	Fluency	Adequacy	Simplicity	All
PBMT-R	3.91	**3.74****	2.80**	3.48*
Hybrid	3.26**	3.42	2.82**	3.17**
DRESS-LS	**3.92**	3.36	**3.55**	**3.61**
Reference	3.74*	3.34	3.13**	3.41**

WikiLarge	Fluency	Adequacy	Simplicity	All
PBMT-R	3.68	**3.63***	2.70**	3.34*
Hybrid	2.60**	2.42**	**3.52**	2.85**
SBMT-SARI	3.34**	3.51*	2.77**	3.21**
DRESS-LS	**3.70**	3.28	3.42	**3.46**
Reference	3.79	3.72**	2.86**	3.46

Table 4.2: Mean ratings elicited by humans on Newsela, WikiSmall, and WikiLarge test sets. Ratings significantly different from DRESS-LS are marked with * ($p < 0.05$) and ** ($p < 0.01$). Significance tests were performed using a student t-test.

model boosts performance as ratings increase almost across the board (Simplicity is slightly worse). Returning to our original questions, we find that neural models are more fluent than comparison systems, while performing non-trivial rewrite operations (see the SARI scores in Table 4.1) which yield simpler output (see the Simplicity column in Table 4.2). Based on our judgment elicitation study, neural models trained with reinforcement learning perform best, with DRESS-LS having a slight advantage.

We further analyzed model performance by computing various statistics on the simplified output. We measured average sentence length and the degree to which DRESS and comparison systems perform rewriting operations. We approximated the latter with Translation Error Rate (TER; Snover et al. 2006), a measure commonly used to automatically evaluate the quality of machine translation output. We used TER to compute the (average) number of edits required to change an original complex sentence to

Models	Len	TER	Ins	Del	Sub	Shft
PBMT-R	23.1	0.13	0.68	0.68	1.50	0.09
Hybrid	12.4	0.90	0.01	10.19	0.12	0.41
EncDecA	17.0	0.36	0.13	5.96	1.69	0.09
DRESS	14.2	0.46	0.07	8.53	1.37	0.11
DRESS-LS	14.4	0.44	0.07	8.38	1.11	0.09
Reference	12.7	0.67	0.40	10.26	3.44	0.73

Table 4.3: Output length (average number of tokens), TER scores and number of edits by type (Ins*ertions*, Del*etions*, Sub*stitutions*, Sh*ifts*) on the Newsela test set. Higher TER means that more rewriting operations are performed.

simpler output. We also report the number of edits by type, i.e., the number of insertions, substitutions, deletions, and shifts needed (on average) to convert complex to simple sentences.

As shown in Table 4.3, Hybrid obtains the highest TER, followed by our models (DRESS and DRESS-LS), which indicates that they actively perform rewriting. Perhaps Hybrid is too aggressive when simplifying a sentence, it obtains low Fluency and Adequacy scores in human evaluation (Table 4.2). There is a strong correlation between sentence length and number of deletion operations (i.e., more deleteions lead to shorter sentences) and PBMT-R performs very few deletions. Overall, reinforcement learning encourages deletion (see DRESS and DRESS-LS), while performing a reasonable amount of additional operations (e.g., substitutions and shifts) compared to EncDecA and PBMT-R.

The middle blocks in Tables 4.1 and 4.2 report results on the WikiSmall dataset. FKGL and SARI follow a similar pattern as on Newsela. BLEU scores for PBMT-R, Hybrid, and EncDecA are much higher compared to DRESS and DRESS-LS. Hybrid obtains best BLEU and SARI scores, while DRESS and DRESS-LS do very well on FKGL. In human evaluation, we elicited judgments on the entire WikiSmall test set (100 sentences). We compared DRESS-LS, with PBMT-R, Hybrid, and gold standard Reference simplifications. As human experiments are time consuming and expensive, we did not include other neural models besides DRESS-LS based on our Newsela study which showed that EncDecA is inferior to variants trained with reinforcement learning and that DRESS-LS is the better performing model (however, we do compare *all* models in Table 4.1). DRESS-LS is significantly better on Simplicity than PBMT-R, Hybrid, and the Reference. It performs on par with PBMT-R on Fluency and worse

on Adequacy (but still closer to the human Reference than PBMT-R or Hybrid). When combining all ratings (All in Table 4.2), DRESS-LS is significantly better than PBMT-R, Hybrid, and the Reference.

The bottom blocks in Tables 4.1 and 4.2 report results on Wikilarge. We compared our models with PBMT-R, Hybrid, and SBMT-SARI (Xu et al., 2016). The FKGL follows a similar pattern as in the previous datasets. PBMT-R and our models are best in terms of BLEU while SBMT-SARI outperforms all other systems on SARI.[13] Because there are 8 references for each complex sentence in the test set, BLEU scores are much higher compared to Newsela and WikiSmall. In human evaluation, we again elicited judgments for 100 randomly sampled test sentences. We randomly selected one of the 8 references as the Reference upper bound. On Simplicity, DRESS-LS is significantly better than all comparison systems, except Hybrid. On Adequacy, it is better than Hybrid but significantly worse than other comparison systems. On Fluency, it is on par with PBMT-R[14] but better than Hybrid and SBMT-SARI. On All dimension DRESS-LS significantly outperforms all comparison systems.

The reward we used is a weighted sum of three different rewards: simplicity reward, relevance reward and fluency reward (see Equation 4.7). From their relative weights (see Section 4.5.2), we can see the most important reward is the simplicity reward, which has the highest weight. In addition to the simplicity reward, introducing fluency and relevance reward lead to small changes in BLEU, SARI and FKGL metrics. However, we do observe their positive impact to system outputs. By adding the fluency reward, we observe some system outputs with grammar mistakes are corrected. For example, “She said the FFA ’s public speak team pushed her beyond her .” becomes “She said the FFA ’s public speak team pushed her .”. “Sierra McKenzie , a fourth-year student , told her team , acceptable your stuff .” becomes “Sierra McKenzie , a fourth-year student , told her team , Remember your stuff .”. By adding the relevance reward, some severely shorten sentences are fixed. For example, the original sentence is “Not only is Dewey their teacher , she ’s also occasionally their ride home or the source of a meal .”. Without the relevance reward, the output is “She ’s also sometimes their ride home .” and with the relevance reward, it becomes “She ’s also sometimes their ride home or the source of a meal .”.

[13] BLEU and SARI scores reported in Xu et al. (2016) are 72.36 and 37.91, and measured at sentence-level.

[14] We used more data to train PBMT-R and maybe that is why PBMT-R performs better than Xu et al. (2016) reported.

4.7 Conclusions

We developed a reinforcement learning-based text simplification model, which can jointly model simplicity, grammaticality, and semantic fidelity to the input. We also proposed a lexical simplification component that further boosts performance. Overall, we find that reinforcement learning offers a great means to inject prior knowledge to the simplification task achieving good results across three datasets. In the future, we would like to explicitly model sentence splitting and simplify entire documents (rather than individual sentences). Beyond sentence simplification, the reinforcement learning framework presented here is potentially applicable to generation tasks such as sentence compression (Chopra et al., 2016), generation of programming code (Ling et al., 2016), or poems (Zhang and Lapata, 2014).

Complex	There's just one major hitch: the primary purpose of education is to develop citizens with a wide variety of skills.
Reference	The purpose of education is to develop a wide range of skills.
PBMT-R	It's just one major hitch: the purpose of education is to **make people** with a wide variety of skills.
Hybrid	one hitch the purpose is to develop citizens.
EncDecA	The **key** of education is to develop **people** with a wide variety of skills.
DRESS	There's just one major hitch: the **main goal** of education is to develop **people** with **lots of** skills.
DRESS-LS	There's just one major hitch: the **main goal** of education is to develop citizens with **lots of** skills.
Complex	"They were so burdened by the past they couldn't think about the future," said Barnet, 62, who was president of Columbia Records, the No.1 record label in the United States, before joining Capitol.
Reference	Capitol was stuck in the past. It could not think about the future, Barnett said.
PBMT-R	"They were so **affected** by the past they couldn't think about the future," said Barnett, 62, was president of Columbia Records, before joining Capitol **building**.
Hybrid	'They were so burdened by the past they couldn't think about the future," said Barnett, 62, who was Columbia Records, president of the No.1 record label in the united states, before joining Capitol.
EncDecA	"They were so burdened by the past they couldn't think about the future," said Barnett, who was president of Columbia Records, the No.1 record labels in the United States.
DRESS	"They were so **sicker** by the past they couldn't think about the future," said Barnett, who was president of Columbia Records.
DRESS-LS	"They were so burdened by the past they couldn't think about the future," said Barnett, who was president of Columbia Records.

Table 4.4: System output for two sentences (Newsela development set). Substitutions are shown in bold.

Chapter 5

Beyond Sequence Learning: Structure based Generation

Most existing neural NLG work uses sequence models (i.e., recurrent neural networks) as the *generation* module. In such a model, a sentence is viewed as a sequence of words. However, in computational linguistics, sentences can also be represented in a tree structure such as dependency trees or constituency trees. In a linear sentence of length N, the longest dependency for a word is at length $N-1$, while by using the tree structure, the longest dependency length can be reduced to rougely $\log(N)$. By reducing the dependency length, we can probably train our model better. In this chapter, we propose a neural language generator called *top-down tree long short-term memory network*, which generates dependency trees rather than linear structured sentences. The final sentences can be generated using in-order traversal of dependency trees.

The neural based language generation architecture we introduced in chapter 1 has two main components: the *representation* network and the generation network. The model we proposed in this chapter is essentially a better generation network.

Long Short-Term Memory (LSTM) networks, a type of recurrent neural network with a more complex computational unit, have been successfully applied to a variety of sequence modeling tasks. In this paper we develop Tree Long Short-Term Memory (TREELSTM), a neural network model based on LSTM, which is designed to predict a tree rather than a linear sequence. TREELSTM defines the probability of a sentence by estimating the generation probability of its dependency tree. At each time step, a node is generated based on the representation of the generated sub-tree. We further enhance the modeling power of TREELSTM by explicitly representing the correlations between left and right dependents. Application of our model to the MSR sentence completion

challenge achieves results beyond the current state of the art. We also report results on dependency parsing reranking achieving competitive performance.

5.1 Introduction

Neural language models have been gaining increasing attention as a competitive alternative to n-grams. The main idea is to represent each word using a real-valued feature vector capturing the contexts in which it occurs. The conditional probability of the next word is then modeled as a smooth function of the feature vectors of the preceding words and the next word. In essence, similar representations are learned for words found in similar contexts resulting in similar predictions for the next word. Previous approaches have mainly employed feed-forward (Bengio et al., 2003; Mnih and Hinton, 2007) and recurrent neural networks (Mikolov et al., 2010; Mikolov, 2012) in order to map the feature vectors of the context words to the distribution for the next word. Recently, RNNs with Long Short-Term Memory (LSTM) units (Hochreiter and Schmidhuber, 1997; Hochreiter, 1998) have emerged as a popular architecture due to their strong ability to capture long-term dependencies. LSTMs have been successfully applied to a variety of tasks ranging from machine translation (Sutskever et al., 2014), to speech recognition (Graves et al., 2013), and image description generation (Vinyals et al., 2015).

Despite superior performance in many applications, neural language models essentially predict sequences of words. Many NLP tasks, however, exploit syntactic information operating over tree structures (e.g., dependency or constituent trees). In this paper we develop a novel neural network model which combines the advantages of the LSTM architecture and syntactic structure. Our model estimates the probability of a sentence by estimating the generation probability of its dependency tree. Instead of explicitly encoding tree structure as a set of features, we use four LSTM networks to model four types of dependency edges which altogether specify how the tree is built. At each time step, one LSTM is activated which predicts the next word conditioned on the sub-tree generated so far. To learn the representations of the conditioned sub-tree, we force the four LSTMs to share their hidden layers. Our model is also capable of generating trees just by sampling from a trained model and can be seamlessly integrated with text generation applications.

Our approach is related to but ultimately different from recursive neural networks (Pollack, 1990) a class of models which operate on structured inputs. Given a (binary)

parse tree, they recursively generate parent representations in a bottom-up fashion, by combining tokens to produce representations for phrases, and eventually the whole sentence. The learned representations can be then used in classification tasks such as sentiment analysis (Socher et al., 2011b) and paraphrase detection (Socher et al., 2011a). Tai et al. (2015) learn distributed representations over syntactic trees by generalizing the LSTM architecture to tree-structured network topologies. The key feature of our model is not so much that it can learn semantic representations of phrases or sentences, but its ability to predict tree structure and estimate its probability.

Syntactic language models have a long history in NLP dating back to Chelba and Jelinek (2000) (see also Roark (2001) and Charniak (2001)). These models differ in how grammar structures in a parsing tree are used when predicting the next word. Other work develops dependency-based language models for specific applications such as machine translation (Shen et al., 2008; Zhang, 2009; Sennrich, 2015), speech recognition (Chelba et al., 1997) or sentence completion (Gubbins and Vlachos, 2013). All instances of these models apply Markov assumptions on the dependency tree, and adopt standard n-gram smoothing methods for reliable parameter estimation. Emami et al. (2003) and Sennrich (2015) estimate the parameters of a structured language model using feed-forward neural networks (Bengio et al., 2003). Mirowski and Vlachos (2015) re-implement the model of Gubbins and Vlachos (2013) with RNNs. They view sentences as sequences of words over a dependency tree. While they ignore the edge types, we model them explicitly.

Our model shares with other structured-based language models the ability to take dependency information into account. It differs in the following respects: (a) it does not artificially restrict the depth of the dependencies it considers and can thus be viewed as an infinite order dependency language model; (b) it not only estimates the probability of a string given its tree structure but is also capable of generating dependency trees with additional classifiers; (c) finally, contrary to previous dependency-based language models which encode syntactic information as features, our model takes tree structure into account more directly via representing different types of dependency edges explicitly using LSTMs. Therefore, there is no need to manually determine which dependency tree features should be used or how large the feature embeddings should be.

We evaluate our model on the MSR sentence completion challenge, a benchmark language modeling dataset. Our results outperform the best published results on this dataset. Since our model is a general tree estimator, we also use it to rerank the top K

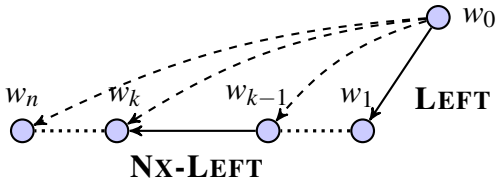


Figure 5.1: LEFT and NX-LEFT edges. Dotted line between w_1 and w_{k-1} (also between w_k and w_n) indicate that there may be ≥ 0 nodes inbetween.

dependency trees from the (second order) MST Pasrser and obtain performance on par with recently proposed dependency parsers.

5.2 Tree Long Short-Term Memory Networks

We seek to estimate the probability of a sentence by estimating the generation probability of its dependency tree. Syntactic information in our model is represented in the form of dependency paths. In the following, we first describe our definition of dependency path and based on it explain how the probability of a sentence is estimated.

5.2.1 Dependency Path

Generally speaking, a dependency path is the path between ROOT and w consisting of the nodes on the path and the edges connecting them. To represent dependency paths, we introduce four types of edges which essentially define the "shape" of a dependency tree. Let w_0 denote a node in a tree and $w_1, w_2, \ldots, w_n$ its left dependents. As shown in Figure 5.1, LEFT edge is the edge between w_0 and its first left dependent denoted as (w_0, w_1). Let w_k (with $1 < k \leq n$) denote a non-first left dependent of w_0. The edge from w_{k-1} to w_k is a NX-LEFT edge (NX stands for NEXT), where w_{k-1} is the right adjacent sibling of w_k. Note that the NX-LEFT edge (w_{k-1}, w_k) replaces edge (w_0, w_k) (illustrated with a dashed line in Figure 5.1) in the original dependency tree. The modification allows information to flow from w_0 to w_k through $w_1, \ldots, w_{k-1}$ rather than directly from w_0 to w_k. RIGHT and NX-RIGHT edges are defined analogously for right dependents.

Given these four types of edges, dependency paths (denoted as $\mathcal{D}(w)$) can be defined as follows bearing in mind that the first right dependent of ROOT is its only dependent and that w^p denotes the parent of w. We use $(\ldots)$ to denote a sequence, where $()$ is an empty sequence and $\|$ is an operator for concatenating two sequences.

(1) if w is ROOT, then $\mathcal{D}(w) = ()$

(2) if w is a **left dependent** of w^p

(a) if w is the first left dependent, then $\mathcal{D}(w) = \mathcal{D}(w^p) \| (\langle w^p, \text{LEFT} \rangle)$

(b) if w is not the first left dependent and w^s is its right adjacent sibling, then $\mathcal{D}(w) = \mathcal{D}(w^s) \| (\langle w^s, \text{NX-LEFT} \rangle)$

(3) if w is a **right dependent** of w^p

(a) if w is the first right dependent, then $\mathcal{D}(w) = \mathcal{D}(w^p) \| (\langle w^p, \text{RIGHT} \rangle)$

(b) if w is not the first right dependent and w^s is its left adjacent sibling, then $\mathcal{D}(w) = \mathcal{D}(w^s) \| (\langle w^s, \text{NX-RIGHT} \rangle)$

A dependency tree can be represented by the set of its dependency paths which in turn can be used to reconstruct the original tree.[1]

Dependency paths for the first two levels of the tree in Figure 5.2 are as follows (ignoring for the moment the subscripts which we explain in the next section). $\mathcal{D}(\text{sold}) = (\langle \text{ROOT}, \text{RIGHT} \rangle)$ (see definitions (1) and (3a)), $\mathcal{D}(\text{year}) = \mathcal{D}(\text{sold}) \| (\langle \text{sold}, \text{LEFT} \rangle)$ (see (2a)), $\mathcal{D}(\text{manufacturer}) = \mathcal{D}(\text{year}) \| (\langle \text{year}, \text{NX-LEFT} \rangle)$ (see (2b)), $\mathcal{D}(\text{cars}) = \mathcal{D}(\text{sold}) \| (\langle \text{sold}, \text{RIGHT} \rangle)$ (see (3a)), $\mathcal{D}(\text{in}) = \mathcal{D}(\text{cars}) \| (\langle \text{cars}, \text{NX-RIGHT} \rangle)$ (according to (3b)).

5.2.2 Sentence Probability

In this section, we estimate the probability of sentence S given its corresponding tree scaffolding T, $P(S|T)$. T is a "scaffolding" tree structure with nodes waiting to be filled with words. For example, the scaffolding tree structure of the sentence *The luxury auto manufacturer last year sold 1,214 cars in the U.S.* is shown in Figure 5.3 (its dependency tree is shown in Figure 5.2). We view the probability computation of a sentence given its scaffolding tree as a generation process and when the generation process is done, we get a dependency tree. Specifically, we assume dependency trees are constructed top-down, in a breadth-first manner. Generation starts at the ROOT node. For each node at each level, first its left dependents are generated from closest to farthest and then the right dependents (again from closest to farthest). The same process is applied to the next node at the same level or a node at the next level. Figure 5.2 shows the breadth-first traversal of a dependency tree.

[1] Throughout this paper we assume all dependency trees are projective.

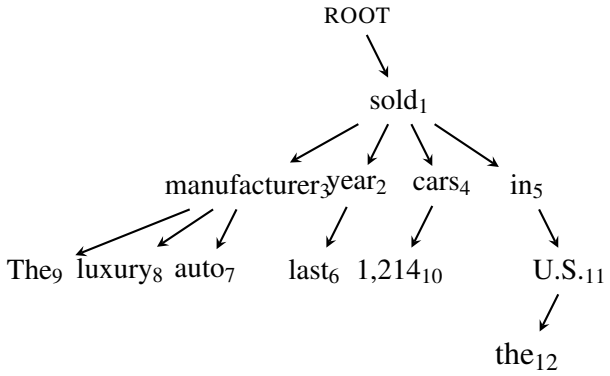


Figure 5.2: Dependency tree of the sentence *The luxury auto manufacturer last year sold 1,214 cars in the U.S.* Subscripts indicate breadth-first traversal. ROOT has only one dependent (i.e., *sold*) which we view as its first right dependent.

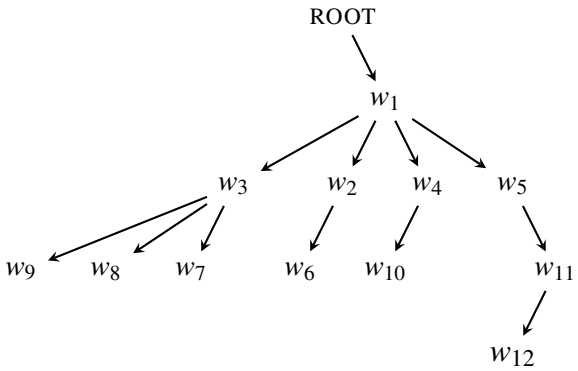


Figure 5.3: Tree scaffolding of the sentence *The luxury auto manufacturer last year sold 1,214 cars in the U.S.* Subscripts indicate breadth-first traversal. $w_i (1 \leq i \leq 12)$ can be any word in the vocabulary.

Under the assumption that each word w in a dependency tree is *only* conditioned on its *dependency path*, the probability of a sentence S given its dependency tree T is:

$$P(S|T) = \prod_{w \in \text{BFS}(T) \setminus \text{ROOT}} P(w|\mathcal{D}(w)) \tag{5.1}$$

where $\mathcal{D}(w)$ is the dependency path of w. Note that each word w is visited according to its breadth-first search order (BFS(T)) and the probability of ROOT is ignored since every tree has one. The role of ROOT in a dependency tree is the same as the begin of sentence token (BOS) in a sentence. When computing $P(S|T)$ (or $P(S)$), the probability of ROOT (or BOS) is ignored (we assume it always exists), but is used to predict other words. We explain in the next section how TREELSTM estimates $P(w|\mathcal{D}(w))$.

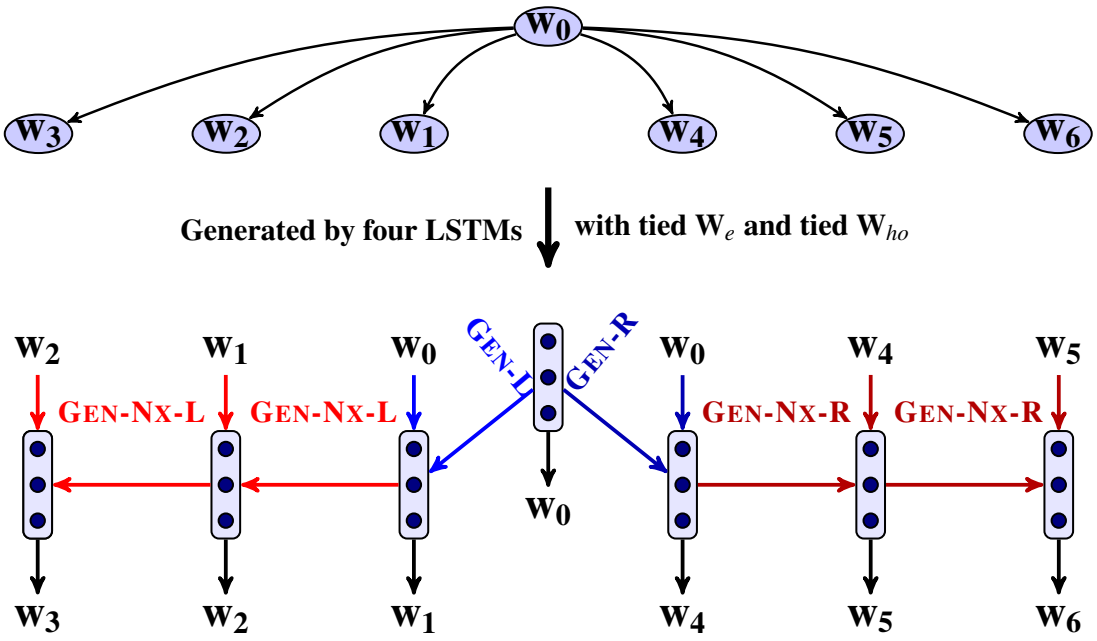


Figure 5.4: Generation process of left (w_1, w_2, w_3) and right (w_4, w_5, w_6) dependents of tree node w_o (top) using four LSTMs (GEN-L, GEN-R, GEN-NX-L and GEN-NX-R). The model can handle an arbitrary number of dependents due to GEN-NX-L and GEN-NX-R.

5.2.3 Tree LSTMs

A dependency path $\mathcal{D}(w)$ is subtree which we denote as a sequence of $\langle word, edge\text{-}type \rangle$ tuples. Our innovation is to learn the representation of $\mathcal{D}(w)$ using four LSTMs. The four LSTMs (GEN-L, GEN-R, GEN-NX-L and GEN-NX-R) are used to represent the four types of edges (LEFT, RIGHT, NX-LEFT and NX-RIGHT) introduced earlier. GEN, NX, L and R are shorthands for GENERATE, NEXT, LEFT and RIGHT. At each time step, an LSTM is chosen according to an edge-type; then the LSTM takes a word as input and predicts/generates its dependent or sibling. This process can be also viewed as adding an edge and a node to a tree. Specifically, LSTMs GEN-L and GEN-R are used to generate the first left and right dependent of a node (w_1 and w_4 in Figure 5.4). So, these two LSTMs are responsible for going deeper in a tree. While GEN-NX-L and GEN-NX-R generate the remaining left/right dependents and therefore go wider in a tree. As shown in Figure 5.4, w_2 and w_3 are generated by GEN-NX-L, whereas w_5 and w_6 are generated by GEN-NX-R. Note that the model can handle any number of left or right dependents by applying GEN-NX-L or GEN-NX-R multiple times.

We assume time steps correspond to the steps taken by the breadth-first traversal of

the dependency tree and the sentence has length n. At time step t ($1 \leq t \leq n$), let $\langle w_{t'}, z_t \rangle$ denote the last tuple in $\mathcal{D}(w_t)$. Subscripts t and t' denote the breadth-first search order of w_t and $w_{t'}$, respectively. $z_t \in \{\text{LEFT}, \text{RIGHT}, \text{NX-LEFT}, \text{NX-RIGHT}\}$ is the edge type (see the definitions in Section 5.2.1). Let $\mathbf{W}_e \in \mathbb{R}^{s \times |V|}$ denote the word embedding matrix and $\mathbf{W}_{ho} \in \mathbb{R}^{|V| \times d}$ the output matrix of our model, where $|V|$ is the vocabulary size, s the word embedding size and d the hidden unit size. We use tied $\mathbf{W}_e$ and tied $\mathbf{W}_{ho}$ for the four LSTMs to reduce the number of parameters in our model. Without tying $\mathbf{W}_{ho}$, the training and inference will be significantly slower, since one mini-batch may be split up into four (thus influencing parallelization). We also observe on a small scale dataset, tying $\mathbf{W}_{ho}$ can lead to slightly better performance. The four LSTMs also share their hidden states. The hidden states updated by one LSTM can be used as input hidden states for other LSTMs. Let $\mathbf{H} \in \mathbb{R}^{d \times (n+1)}$ denote the *shared* hidden states of all time steps and $e(w_t)$ the one-hot vector of w_t. Then, $\mathbf{H}[:,t]$ represents $\mathcal{D}(w_t)$ at time step t, and the computation[2] is:

$$\mathbf{x}_t = \mathbf{W}_e \cdot e(w_{t'}) \tag{5.2a}$$

$$\mathbf{h}_t = \text{LSTM}^{z_t}(\mathbf{x}_t, \mathbf{H}[:,t']) \tag{5.2b}$$

$$\mathbf{H}[:,t] = \mathbf{h}_t \tag{5.2c}$$

$$\mathbf{y}_t = \mathbf{W}_{ho} \cdot \mathbf{h}_t \tag{5.2d}$$

where the initial hidden state $\mathbf{H}[:,0]$ is initialized to a vector of small values such as 0.01. According to Equation (5.2b), the model selects an LSTM based on edge type z_t. We describe the details of LSTM^{z_t} in the next paragraph. The probability of w_t given its dependency path $\mathcal{D}(w_t)$ is estimated by a *softmax* function:

$$P(w_t|\mathcal{D}(w_t)) = \frac{\exp(\mathbf{y}_{t,w_t})}{\sum_{k'=1}^{|V|} \exp(\mathbf{y}_{t,k'})} \tag{5.3}$$

We must point out that although we use four jointly trained LSTMs to encode the hidden states, the training and inference complexity of our model is no different from a regular LSTM, since at each time step only one LSTM is working.

We implement LSTM^z in Equation (5.2b) using a deep LSTM (to simplify notation, from now on we write z instead of z_t). The inputs at time step t are $\mathbf{x}_t$ and $\mathbf{h}_{t'}$ (the hidden state of an earlier time step t') and the output is $\mathbf{h}_t$ (the hidden state of current time step). Let L denote the layer number of LSTM^z and $\hat{\mathbf{h}}_t^l$ the internal hidden state of the l-th layer of the LSTM^z at time step t, where $\mathbf{x}_t$ is $\hat{\mathbf{h}}_t^0$ and $\mathbf{h}_{t'}$ is $\hat{\mathbf{h}}_{t'}^L$. The LSTM architecture

[2]We ignore all bias terms for notational simplicity.

introduces multiplicative gates and memory cells $\hat{\mathbf{c}}_t^l$ (at l-th layer) in order to address the *vanishing gradient* problem which makes it difficult for the standard RNN model to learn long-distance correlations in a sequence. Here, $\hat{\mathbf{c}}_t^l$ is a linear combination of the current input signal $\mathbf{u}_t$ and an earlier memory cell $\hat{\mathbf{c}}_{t'}^l$. How much input information $\mathbf{u}_t$ will flow into $\hat{\mathbf{c}}_t^l$ is controlled by input gate $\mathbf{i}_t$ and how much of the earlier memory cell $\hat{\mathbf{c}}_{t'}^l$ will be forgotten is controlled by forget gate $\mathbf{f}_t$. This process is computed as follows:

$$\mathbf{u}_t = \tanh(\mathbf{W}_{ux}^{z,l} \cdot \hat{\mathbf{h}}_t^{l-1} + \mathbf{W}_{uh}^{z,l} \cdot \hat{\mathbf{h}}_{t'}^l) \tag{5.4a}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_{ix}^{z,l} \cdot \hat{\mathbf{h}}_t^{l-1} + \mathbf{W}_{ih}^{z,l} \cdot \hat{\mathbf{h}}_{t'}^l) \tag{5.4b}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{fx}^{z,l} \cdot \hat{\mathbf{h}}_t^{l-1} + \mathbf{W}_{fh}^{z,l} \cdot \hat{\mathbf{h}}_{t'}^l) \tag{5.4c}$$

$$\hat{\mathbf{c}}_t^l = \mathbf{f}_t \odot \hat{\mathbf{c}}_{t'}^l + \mathbf{i}_t \odot \mathbf{u}_t \tag{5.4d}$$

where $\mathbf{W}_{ux}^{z,l} \in \mathbb{R}^{d \times d}$ ($\mathbf{W}_{ux}^{z,l} \in \mathbb{R}^{d \times s}$ when $l = 1$) and $\mathbf{W}_{uh}^{z,l} \in \mathbb{R}^{d \times d}$ are weight matrices for $\mathbf{u}_t$, $\mathbf{W}_{ix}^{z,l}$ and $\mathbf{W}_{ih}^{z,l}$ are weight matrices for $\mathbf{i}_t$ and $\mathbf{W}_{fx}^{z,l}$, and $\mathbf{W}_{fh}^{z,l}$ are weight matrices for $\mathbf{f}_t$. σ is a sigmoid function and $\odot$ the element-wise product.

Output gate $\mathbf{o}_t$ controls how much information of the cell $\hat{\mathbf{c}}_t^l$ can be seen by other modules:

$$\mathbf{o}_t = \sigma(\mathbf{W}_{ox}^{z,l} \cdot \hat{\mathbf{h}}_t^{l-1} + \mathbf{W}_{oh}^{z,l} \cdot \hat{\mathbf{h}}_{t'}^l) \tag{5.5a}$$

$$\hat{\mathbf{h}}_t^l = \mathbf{o}_t \odot \tanh(\hat{\mathbf{c}}_t^l) \tag{5.5b}$$

Application of the above process to all layers L, will yield $\hat{\mathbf{h}}_t^L$, which is $\mathbf{h}_t$. Note that in implementation, all $\hat{\mathbf{c}}_t^l$ and $\hat{\mathbf{h}}_t^l$ ($1 \leq l \leq L$) at time step t are stored, although we only care about $\hat{\mathbf{h}}_t^L$ ($\mathbf{h}_t$).

5.2.4 Left Dependent Tree LSTMs

TREELSTM computes $P(w|\mathcal{D}(w))$ based on the dependency path $\mathcal{D}(w)$, which ignores the interaction between left and right dependents on the same level. In many cases, TREELSTM will use a verb to predict its object directly without knowing its subject. For example, in Figure 5.2, TREELSTM uses $\langle$ROOT, RIGHT$\rangle$ and $\langle$ *sold*, RIGHT $\rangle$ to predict *cars*. This information is unfortunately not specific to *cars* (many things can be sold, e.g., *chocolates*, *candy*). Considering *manufacturer*, the left dependent of *sold* would help predict *cars* more accurately.

In order to jointly take left and right dependents into account, we employ yet another LSTM, which goes from the furthest left dependent to the closest left dependent

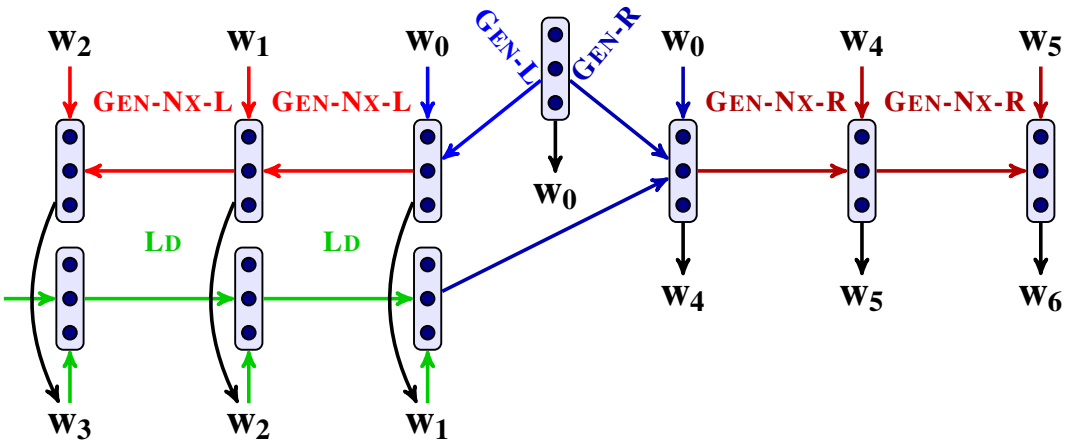


Figure 5.5: Generation of left and right dependents of node w_0 according to LDTREELSTM.

(LD is a shorthand for left dependent). As shown in Figure 5.5, LD LSTM learns the representation of all left dependents of a node w_0; this representation is then used to predict the first right dependent of the same node. Non-first right dependents can also leverage the representation of left dependents, since this information is injected into the hidden state of the first right dependent and can percolate all the way. Note that in order to retain the generation capability of our model (Section 5.3.4), we only allow right dependents to leverage left dependents (they are generated before right dependents).

The computation of the LDTREELSTM is almost the same as in TREELSTM except when $z_t = \text{GEN-R}$. In this case, let $\mathbf{v}_t$ be the corresponding left dependent sequence with length K ($\mathbf{v}_t = (w_3, w_2, w_1)$ in Figure 5.5). Then, the hidden state ($\mathbf{q}_k$) of $\mathbf{v}_t$ at each time step k is:

$$\mathbf{m}_k = \mathbf{W}_e \cdot e(\mathbf{v}_{t,k}) \tag{5.6a}$$

$$\mathbf{q}_k = \text{LSTM}^{\text{LD}}(\mathbf{m}_k, \mathbf{q}_{k-1}) \tag{5.6b}$$

where $\mathbf{q}_K$ is the representation for all left dependents. Then, the computation of the current hidden state becomes (see Equation (5.2) for the original computation):

$$\mathbf{r}_t = \begin{bmatrix} \mathbf{W}_e \cdot e(w_{t'}) \\ \mathbf{q}_K \end{bmatrix} \tag{5.7a}$$

$$\mathbf{h}_t = \text{LSTM}^{\text{GEN-R}}(\mathbf{r}_t, \mathbf{H}[:, t']) \tag{5.7b}$$

where $\mathbf{q}_K$ serves as additional input for $\text{LSTM}^{\text{GEN-R}}$. All other computational details are the same as in TreeLSTM (see Section 5.2.3).

5.2.5 Model Training

On small scale datasets we employ Negative Log-likelihood (NLL) as our training objective for both TREELSTM and LDTREELSTM:

$$\mathcal{L}^{\text{NLL}}(\theta) = -\frac{1}{|\mathcal{S}|}\sum_{S\in\mathcal{S}}\log P(S|T) \tag{5.8}$$

where S is a sentence in the training set $\mathcal{S}$, T is the dependency tree of S and $P(S|T)$ is defined as in Equation (5.1).

On large scale datasets (e.g., with vocabulary size of 65K), computing the output layer activations and the *softmax* function with NLL would become prohibitively expensive. Instead, we employ Noise Contrastive Estimation (NCE; Gutmann and Hyvärinen (2012), Mnih and Teh (2012)) which treats the normalization term $\hat{Z}$ in $\hat{P}(w|\mathcal{D}(w_t)) = \frac{\exp(\mathbf{W}_{ho}[w,:]\cdot\mathbf{h}_t)}{\hat{Z}}$ as constant. The intuition behind NCE is to discriminate between samples from a data distribution $\hat{P}(w|\mathcal{D}(w_t))$ and a known noise distribution $P_n(w)$ via binary logistic regression. Assuming that noise words are k times more frequent than real words in the training set (Mnih and Teh, 2012), then the probability of a word w being from our model $P_d(w,\mathcal{D}(w_t))$ is $\frac{\hat{P}(w|\mathcal{D}(w_t))}{\hat{P}(w|\mathcal{D}(w_t))+kP_n(w)}$. We apply NCE to large vocabulary models with the following training objective:

$$\mathcal{L}^{\text{NCE}}(\theta) = -\frac{1}{|\mathcal{S}|}\sum_{T\in\mathcal{S}}\sum_{t=1}^{|T|}\Bigg(\log P_d(w_t,\mathcal{D}(w_t)) + \sum_{j=1}^{k}\log[1-P_d(\tilde{w}_{t,j},\mathcal{D}(w_t))]\Bigg)$$

where $\tilde{w}_{t,j}$ is a word sampled from the noise distribution $P_n(w)$. We use smoothed unigram frequencies (exponentiating by 0.75) as the noise distribution $P_n(w)$ (Mikolov et al., 2013b). We initialize $\ln\hat{Z} = 9$ as suggested in Chen et al. (2015), but instead of keeping it fixed we also learn $\hat{Z}$ during training (Vaswani et al., 2013). We set $k = 20$.

5.3 Experiments

We assess the performance of our model on two tasks: the Microsoft Research (MSR) sentence completion challenge (Zweig and Burges, 2012), and dependency parsing reranking. We also demonstrate the tree generation capability of our models with the help of classifiers predicting scaffolding tree structures. Note that in the first two experiments and the training stage of the third experiment, we all need a dependency

parser to first parse the text to be processed (we assume the depedency trees are given), but during the test stage of experiment three, our models can predict tree structures on the fly. In the following, we first present details on model training and then present our results. We implemented our models using the Torch library (Collobert et al., 2011) and our code is available at `https://github.com/XingxingZhang/td-treelstm`.

5.3.1 Training Details

We trained our model with back propagation through time (Rumelhart et al., 1988) on an Nvidia GPU Card with a mini-batch size of 64. The objective (NLL or NCE) was minimized by stochastic gradient descent. Model parameters were uniformly initialized in $[-0.1, 0.1]$. We used the NCE objective on the MSR sentence completion task (due to the large size of this dataset) and the NLL objective on dependency parsing reranking. We used an initial learning rate of 1.0 for all experiments and when there was no significant improvement in log-likelihood on the validation set (usually around the 10th epoch), the learning rate was divided by 2 per epoch until convergence (Mikolov et al., 2010). To alleviate the exploding gradients problem, we rescaled the gradient g when the gradient norm $||g|| > 5$ and set $g = \frac{5g}{||g||}$ (Pascanu et al., 2013; Sutskever et al., 2014). Dropout (Srivastava et al., 2014) was applied to the 2-layer TREELSTM and LDTREELSTM models. The word embedding size was set to $s = d/2$ where d is the hidden unit size.

5.3.2 Microsoft Sentence Completion Challenge

The task in the MSR Sentence Completion Challenge (Zweig et al., 2012) is to select the correct missing word for 1,040 SAT-style test sentences when presented with five candidate completions. Here is an example:

I have seen on him , and could ____ to it .
a) write b) migrate c) climb d) swear e) contribute

The training set contains 522 novels from the Project Gutenberg which we preprocessed as follows. After removing headers and footers from the files, we tokenized and parsed the dataset into dependency trees with the Stanford Core NLP toolkit (Manning et al., 2014). The resulting training set contained 49M words. We converted all words to lower case and replaced those occurring five times or less with UNK. The resulting

vocabulary size was 65,346 words. We randomly sampled 4,000 sentences from the training set as our validation set.

The literature describes two main approaches to the sentence completion task based on word vectors and language models. In vector-based approaches, all words in the sentence and the five candidate words are represented by a vector; the candidate which has the highest average similarity with the sentence words is selected as the answer. For language model-based methods, the LM computes the probability of a test sentence with each of the five candidate words, and picks the candidate completion which gives the highest probability. Our model belongs to this class of models.

Table 5.1 presents a summary of our results together with previously published results. The best performing word vector model is IVLBL (Mnih and Kavukcuoglu, 2013) with an accuracy of 55.5, while the best performing single language model is LBL (Mnih and Teh, 2012) with an accuracy of 54.7. Both approaches are based on the log-bilinear language model (Mnih and Hinton, 2007). A combination of several recurrent neural networks and the skip-gram model holds the state of the art with an accuracy of 58.9 (Mikolov et al., 2013a). To fairly compare with existing models, we restrict the layer size of our models to 1. We observe that LDTREELSTM consistently outperforms TREELSTM, which indicates the importance of modeling the interaction between left and right dependents. In fact, LDTREELSTM ($d = 400$) achieves a new state-of-the-art on this task[3], despite being a single model (previous state-of-the-art (Mikolov et al., 2013a) used a combination of several models). We also implement LSTM and bidirectional LSTM language models.[4] An LSTM with $d = 400$ outperforms its smaller counterpart ($d = 300$), however performance decreases with $d = 450$. The bidirectional LSTM is worse than the LSTM (see Mnih and Teh (2012) for a similar observation). The best performing LSTM is worse than a LDTREELSTM ($d = 300$). The input and output embeddings ($\mathbf{W}_e$ and $\mathbf{W}_{ho}$) dominate the number of parameters in all neural models except for RNNME, depRNN+3gram and ldepRNN+4gram, which include a ME model that contains 1 billion sparse n-gram features (Mikolov, 2012; Mirowski and Vlachos, 2015). The number of parameters in TREELSTM and LDTREELSTM is not much larger compared to LSTM due to the tied $\mathbf{W}_e$ and $\mathbf{W}_{ho}$ matrices.

[3]Now the state-of-the-art is 69.2 achieved by the recurrent memory network (Tran et al., 2016).

[4]LSTMs and BiLSTMs were also trained with NCE ($s = d/2$; hyperparameters were tuned on the development set).

Model	d	$\|\theta\|$	**Accuracy**
Word Vector based Models			
LSA	—	—	49.0
Skip-gram	640	102M	48.0
IVLBL	600	96.0M	55.5
Language Models			
KN5	—	—	40.0
UDepNgram	—	—	48.3
LDepNgram	—	—	50.0
RNN	300	48.1M	45.0
RNNME	300	1120M	49.3
depRNN+3gram	100	1014M	53.5
ldepRNN+4gram	200	1029M	50.7
LBL	300	48.0M	54.7
LSTM	300	29.9M	55.00
LSTM	400	40.2M	57.02
LSTM	450	45.3M	55.96
Bidirectional LSTM	200	33.2M	48.46
Bidirectional LSTM	300	50.1M	49.90
Bidirectional LSTM	400	67.3M	48.65
Model Combinations			
RNNMEs	—	—	55.4
Skip-gram + RNNMEs	—	—	58.9
Our Models			
TREELSTM	300	31.6M	55.29
LDTREELSTM	300	32.5M	57.79
TREELSTM	400	43.1M	56.73
LDTREELSTM	400	44.7M	**60.67**

Table 5.1: Model accuracy on the MSR sentence completion task. The results of KN5, RNNME and RNNMEs are reported in Mikolov (2012), LSA and RNN in Zweig et al. (2012), UDepNgram and LDepNgram in Gubbins and Vlachos (2013), depRNN+3gram and depRNN+4gram in Mirowski and Vlachos (2015), LBL in Mnih and Teh (2012), Skip-gram and Skip-gram+RNNMEs in Mikolov et al. (2013a), and IVLBL in Mnih and Kavukcuoglu (2013); d is the hidden size and $|\theta|$ the number of parameters in a model.

Parser	Development		Test	
	UAS	LAS	UAS	LAS
MSTParser-2nd	92.20	88.78	91.63	88.44
TREELSTM	92.51	89.07	91.79	88.53
TREELSTM*	92.64	89.09	91.97	88.69
LDTREELSTM	**92.66**	**89.14**	**91.99**	**88.69**
NN parser*	92.00	89.70	91.80	89.60
S-LSTM*	**93.20**	**90.90**	**93.10**	**90.90**

Table 5.2: Performance of TREELSTM and LDTREELSTM on reranking the top dependency trees produced by the 2nd order MSTParser (McDonald and Pereira, 2006). Results for the NN and S-LSTM parsers are reported in Chen and Manning (2014) and Dyer et al. (2015), respectively. * indicates that the model is initialized with pre-trained word vectors.

5.3.3 Dependency Parsing

In this section we demonstrate that our model can be also used for parse reranking. This is not possible for sequence-based language models since they cannot estimate the probability a sentence given its (possible) tree structure. A dependency parser estimates the probability of a tree T given a sentence S, $P(T|S)$, while our model estimates $P(S|T)$. S is known, therefore

$$P(T|S) \propto P(S|T)P(T) \tag{5.9}$$

We do not have much prior information on T and therefore we assume T is uniformly distributed among all possible trees. Then, we have

$$T^* = \arg\max_T P(T|S) = \arg\max_T P(S|T)P(T) = \arg\max_T P(S|T) \tag{5.10}$$

Therefore, we can use our model to rerank dependency parsing. We use our models to rerank the top K dependency trees produced by the second order MSTParser (McDonald and Pereira, 2006).[5] We follow closely the experimental setup of Chen and Manning (2014) and Dyer et al. (2015). Specifically, we trained TREELSTM and LDTREELSTM on Penn Treebank sections 2–21. We used section 22 for development and section 23 for testing. We adopted the Stanford basic dependency representations

[5] http://www.seas.upenn.edu/ strctlrn/MSTParser

(De Marneffe et al., 2006); part-of-speech tags were predicted with the Stanford Tagger (Toutanova et al., 2003). We trained TREELSTM and LDTREELSTM as language models (singletons were replaced with UNK) and did not use any POS tags, dependency labels or composition features, whereas these features are used in Chen and Manning (2014) and Dyer et al. (2015). We tuned d, the number of layers, and K on the development set.

Table 5.2 reports unlabeled attachment scores (UAS) and labeled attachment scores (LAS) for the MSTParser, TREELSTM ($d = 300$, 1 layer, $K = 2$), and LDTREELSTM ($d = 200$, 2 layers, $K = 4$). We also tried larger K, but did not get better performance. Probably because our model only see sentences with their gold trees during training and thus still lack the capability to assign proper probabilities to sentences given trees with mistakes. We also include the performance of two neural network-based dependency parsers; Chen and Manning (2014) use a neural network classifier to predict the correct transition (NN parser); Dyer et al. (2015) also implement a transition-based dependency parser using LSTMs to represent the contents of the stack and buffer in a continuous space. As can be seen, both TREELSTM and LDTREELSTM outperform the baseline MSTParser, with LDTREELSTM performing best. We also initialized the word embedding matrix $\mathbf{W}_e$ with pre-trained GLOVE vectors (Pennington et al., 2014). We obtained a slight improvement over TREELSTM (TREELSTM* in Table 5.2; $d = 200$, 2 layer, $K = 4$) but no improvement over LDTREELSTM. Finally, notice that LDTREELSTM is slightly better than the NN parser in terms of UAS but worse than the S-LSTM parser. In the future, we would like to extend our model so that it takes labeled dependency information into account.

5.3.4 Tree Generation

This section demonstrates how to use a trained LDTREELSTM to generate tree samples. The generation starts at the ROOT node. At each time step t, for each node w_t, we add a new edge and node to the tree. Unfortunately during generation, we do not know which type of edge to add. We therefore use four binary classifiers (ADD-LEFT, ADD-RIGHT, ADD-NX-LEFT and ADD-NX-RIGHT) to predict whether we should add a LEFT, RIGHT, NX-LEFT or NX-RIGHT edge.[6] These binary classifiers actually try to

[6]It is possible to get rid of the four classifiers by adding START/STOP symbols when generating left and right dependents as in Eisner (1996). We refrained from doing this for computational reasons. For a sentence with N words, this approach will lead to $2N$ additional START/STOP symbols (with one START and one STOP symbol for each word). Consequently, the computational cost and memory consumption during training will be three times as much rendering our model less scalable.

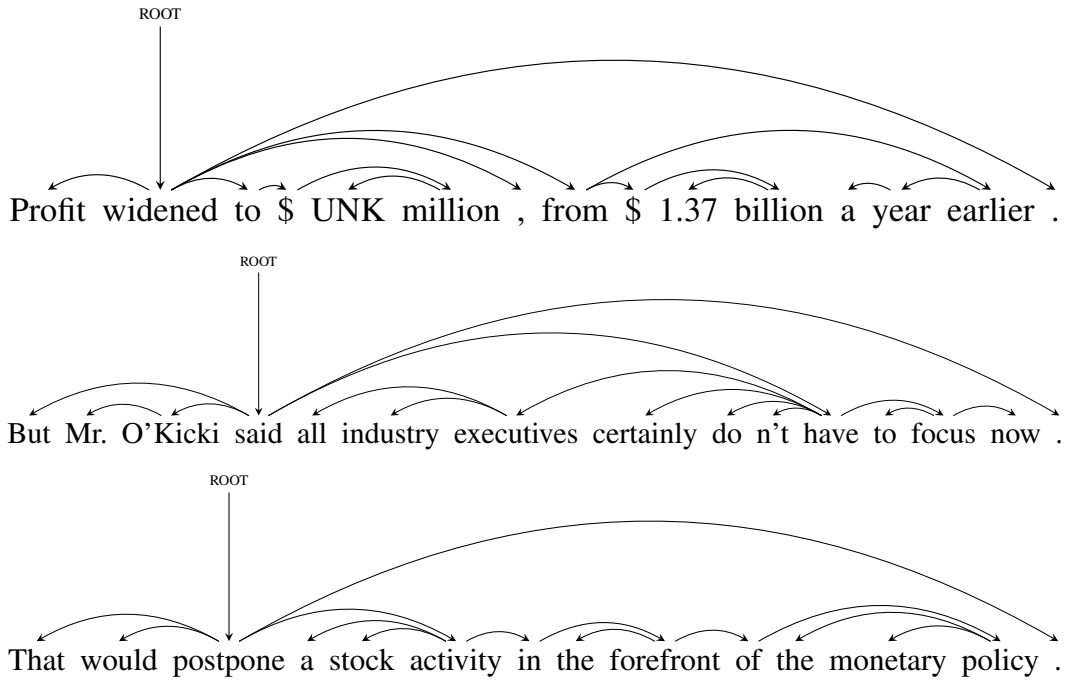


Figure 5.6: Generated dependency trees with LDTREELSTM trained on the PTB.

predict the scaffolding tree structure described in Section 5.2.2. Then when a classifier predicts true, we use the corresponding LSTM to generate a new node by sampling from the predicted word distribution in Equation (6.5). The four classifiers take the previous hidden state $\mathbf{H}[:,t']$ and the output embedding of the current node $\mathbf{W}_{ho} \cdot e(w_t)$ as features.[7] Specifically, we use a trained LDTREELSTM to go through the training corpus and generate hidden states and embeddings as input features; the corresponding class labels (true and false) are "read off" the training dependency trees. We use two-layer rectifier networks (Glorot et al., 2011) as the four classifiers with a hidden size of 300. We use the same LDTREELSTM model as in Section 5.3.3 to generate dependency trees. The classifiers were trained using AdaGrad (Duchi et al., 2011) with a learning rate of 0.01. The accuracies of ADD-LEFT, ADD-RIGHT, ADD-NX-LEFT and ADD-NX-RIGHT are 94.3%, 92.6%, 93.4% and 96.0%, respectively. Figure 5.6 shows examples of generated trees.

5.4 Conclusions

In this paper we developed TREELSTM (and LDTREELSTM), a neural network model architecture, which is designed to predict tree structures rather than linear sequences.

[7]The input embeddings have lower dimensions and therefore result in slightly worse classifiers.

Experimental results on the MSR sentence completion task show that LDTREELSTM is superior to sequential LSTMs. Dependency parsing reranking experiments highlight our model's potential for dependency parsing. Finally, the ability of our model to generate dependency trees holds promise for text generation applications such as sentence compression and simplification (Filippova et al., 2015). Although our experiments have focused exclusively on dependency trees, there is nothing inherent in our formulation that disallows its application to other types of tree structure such as constituent trees or even taxonomies.

Chapter 6

More on Structure based Generation

We have shown in the previous chapter that our dependency tree strucuture based models (i.e. TREELSTM and LDTREELSTM) can outperform sequence based models. Both TREELSTM and LDTREELSTM rely on a pretrained dependency parser. This chapter explores whether better dependency parsers can further improve the performance of our tree based models. We first propose a new dependency parser and then apply it to our TREELSTM and LDTREELSTM models.

Conventional graph-based dependency parsers guarantee a tree structure both during training and inference. Instead, we formalize dependency parsing as the problem of independently selecting the head of each word in a sentence. Our model which we call DENSE (as shorthand for **De**pendency **N**eural **Se**lection) produces a distribution over possible heads for each word using features obtained from a bidirectional recurrent neural network. Without enforcing structural constraints during training, DENSE generates (at inference time) trees for the overwhelming majority of sentences, while non-tree outputs can be adjusted with a maximum spanning tree algorithm. We evaluate DENSE on four languages (English, Chinese, Czech, and German) with varying degrees of non-projectivity. Despite the simplicity of the approach, our parsers are on par with the state of the art.[1] We empirically found that better dependency parsers are indeed helpful for our TREELSTM and LDTREELSTM models.

6.1 Introduction

Dependency parsing plays an important role in many natural language applications, such as relation extraction (Fundel et al., 2007), machine translation (Carreras and

[1] Our code is available at `http://github.com/XingxingZhang/dense_parser`.

Collins, 2009), language modeling (Chelba et al., 1997; Zhang et al., 2016) and ontology construction (Snow et al., 2005). Dependency parsers represent syntactic information as a set of head-dependent relational arcs, typically constrained to form a tree. Practically all models proposed for dependency parsing in recent years can be described as graph-based (McDonald et al., 2005a) or transition-based (Yamada and Matsumoto, 2003; Nivre et al., 2006b).

Graph-based dependency parsers are typically arc-factored, where the score of a tree is defined as the sum of the scores of all its arcs. An arc is scored with a set of local features and a linear model, the parameters of which can be effectively learned with online algorithms (Crammer and Singer, 2001, 2003; Freund and Schapire, 1999; Collins, 2002). In order to efficiently find the best scoring tree during training *and* decoding, various maximization algorithms have been developed (Eisner, 1996, 2000; McDonald et al., 2005b). In general, graph-based methods are optimized globally, using features of single arcs in order to make the learning and inference tractable. Transition-based algorithms factorize a tree into a set of parsing actions. At each transition state, the parser scores a candidate action conditioned on the state of the transition system and the parsing history, and selects high-scoring actions to execute. This score is typically obtained with a classifier based on non-local features defined over a rich history of parsing decisions (Yamada and Matsumoto, 2003; Zhang and Nivre, 2011).

Regardless of the algorithm used, most well-known dependency parsers, such as the MST-Parser (McDonald et al., 2005b) and the MaltPaser (Nivre et al., 2006a), rely on extensive feature engineering. Feature templates are typically manually designed and aim at capturing head-dependent relationships which are notoriously sparse and difficult to estimate. More recently, a few approaches (Chen and Manning, 2014; Pei et al., 2015; Kiperwasser and Goldberg, 2016) apply neural networks for learning dense feature representations. The learned features are subsequently used in a conventional graph- or transition-based parser, or better designed variants (Dyer et al., 2015).

In this work, we propose a simple neural network-based model which learns to select the head for each word in a sentence without enforcing tree structured output. Our model which we call DENSE (as shorthand for **De**pendency **N**eural **Se**lection) employs bidirectional recurrent neural networks to learn feature representations for words in a sentence. These features are subsequently used to predict the head of each word. Although there is nothing inherent in the model to enforce tree-structured output, when tested on an English dataset, it is able to generate trees for 95% of the sentences, 87% of which are projective. The remaining non-tree (or non-projective) outputs are

post-processed with the Chu-Liu-Edmond (or Eisner) algorithm. DENSE uses the head selection procedure to estimate arc weights during training. During testing, it essentially reduces to a standard graph-based parser when it fails to produce tree (or projective) output.

We evaluate our model on benchmark dependency parsing corpora, representing four languages (English, Chinese, Czech, and German) with varying degrees of non-projectivity. Despite the simplicity of our approach, experiments show that the resulting parsers are on par with the state of the art.

6.2 Related Work

6.2.1 Graph-based Parsing

Graph-based dependency parsers employ a model for scoring possible dependency graphs for a given sentence. The graphs are typically factored into their component arcs and the score of a tree is defined as the sum of its arcs. This factorization enables tractable search for the highest scoring graph structure which is commonly formulated as the search for the maximum spanning tree (MST). The Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967; McDonald et al., 2005b) is often used to extract the MST in the case of non-projective trees, and the Eisner algorithm (Eisner, 1996, 2000) in the case of projective trees. During training, weight parameters of the scoring function can be learned with margin-based algorithms (Crammer and Singer, 2001, 2003) or the structured perceptron (Freund and Schapire, 1999; Collins, 2002). Beyond basic first-order models, the literature offers a few examples of higher-order models involving sibling and grand parent relations (Carreras, 2007; Koo et al., 2010; Zhang and McDonald, 2012). Although more expressive, such models render both training and inference more challenging.

6.2.2 Transition-based Parsing

As the term implies, transition-based parsers conceptualize the process of transforming a sentence into a dependency tree as a sequence of transitions. A transition system typically includes a stack for storing partially processed tokens, a buffer containing the remaining input, and a set of arcs containing all dependencies between tokens that have been added so far (Nivre, 2003; Nivre et al., 2006b). A dependency tree is constructed by manipulating the stack and buffer, and appending arcs with predetermined

operations. Most popular parsers employ an *arc-standard* (Yamada and Matsumoto, 2003; Nivre, 2004) or *arc-eager* transition system (Nivre, 2008). Extensions of the latter include the use of non-local training methods to avoid greedy error propagation (Zhang and Clark, 2008; Huang and Sagae, 2010; Zhang and Nivre, 2011; Goldberg and Nivre, 2012).

6.2.3 Neural Network-based Features

Neural network representations have a long history in syntactic parsing (Mayberry and Miikkulainen, 1999; Henderson, 2004; Titov and Henderson, 2007). Recent work uses neural networks in lieu of the linear classifiers typically employed in conventional transition- or graph-based dependency parsers. For example, Chen and Manning (2014) use a feed forward neural network to learn features for a transition-based parser, whereas Pei et al. (2015) do the same for a graph-based parser. Lei et al. (2014) apply tensor decomposition to obtain word embeddings in their syntactic roles, which they subsequently use in a graph-based parser. Dyer et al. (2015) redesign components of a transition-based system where the buffer, stack, and action sequences are modeled separately with stack long short-term memory networks. The hidden states of these LSTMs are concatenated and used as features to a final transition classifier. Kiperwasser and Goldberg (2016) use bidirectional LSTMs to extract features for a transition- and graph-based parser, whereas Cross and Huang (2016) build a greedy arc-standard parser using similar features.

In our work, we formalize dependency parsing as the task of finding for each word in a sentence its most probable head. Both head selection and the features it is based on are learned using neural networks. The idea of modeling child-parent relations independently dates back to Hall (2007) who use an edge-factored model to generate k-best parse trees which are subsequently reranked using a model based on rich global features. Later Smith (2010) show that a head selection variant of their loopy belief propagation parser performs worse than a model which incorporates tree structure constraints. Our parser is conceptually simpler: we rely on head selection to do most of the work and decode the best tree *directly* without using a reranker. In common with recent neural network-based dependency parsers, we aim to alleviate the need for hand-crafting feature combinations. Beyond feature learning, we further show that it is possible to simplify the training of a graph-based dependency parser in the context of bidirectional recurrent neural networks.

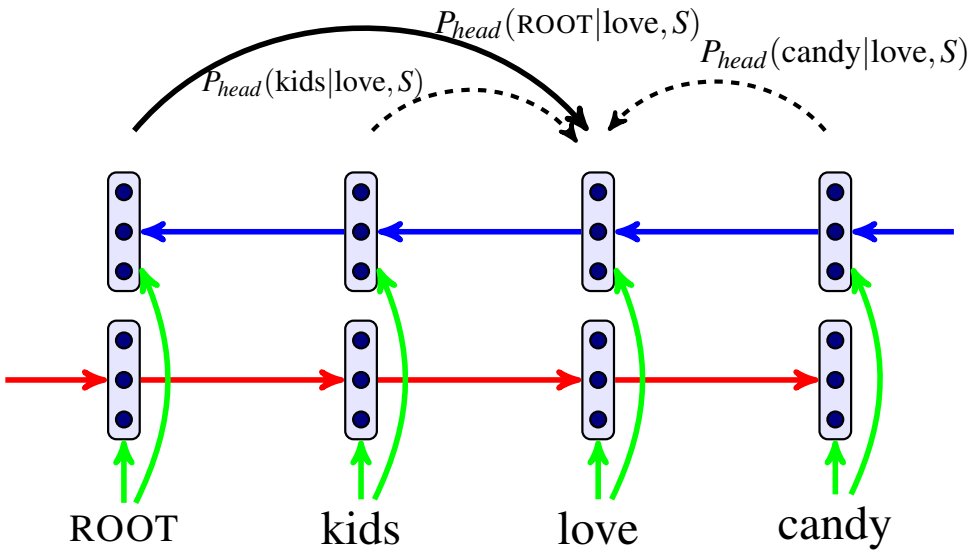


Figure 6.1: DENSE estimates the probability a word being the head of another word based on bidirectional LSTM representations for the two words. $P_{head}(\text{ROOT}|\text{love}, S)$ is the probability of ROOT being the head of *love* (dotted arcs denote candidate heads; the solid arc is the goldstandard).

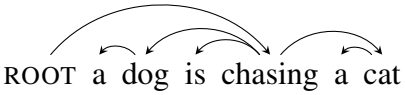
6.3 Dependency Parsing as Head Selection

In this section we present our parsing model, DENSE, which tries to predict the head of each word in a sentence. Specifically, the model takes as input a sentence of length N and outputs N $\langle$head, dependent$\rangle$ arcs. We describe the model focusing on unlabeled dependencies and then discuss how it can be straightforwardly extended to the labeled setting. We begin by explaining how words are represented in our model and then give details on how DENSE makes predictions based on these learned representations. Since there is no guarantee that the outputs of DENSE are trees (although they mostly are), we also discuss how to extend DENSE in order to enforce projective and non-projective tree outputs. Throughout this chapter, lowercase boldface letters denote vectors (e.g., $\mathbf{v}$ or $\mathbf{v}_i$), uppercase boldface letters denote matrices (e.g., $\mathbf{M}$ or $\mathbf{M}_b$), and lowercase letters denote scalars (e.g., w or w_i).

6.3.1 Word Representation

Let $S = (w_0, w_1, \ldots, w_N)$ denote a sentence of length N; following common practice in the dependency parsing literature (Kübler et al., 2009), we add an artificial ROOT token represented by w_0. Analogously, let $A = (\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_N)$ denote the representation

of sentence S, with $\mathbf{a}_i$ representing word w_i $(0 \leq i \leq N)$. Besides encoding information about each w_i in isolation (e.g., its lexical meaning or POS tag), $\mathbf{a}_i$ must also encode w_i's positional information within the sentence. Such information has been shown to be important in dependency parsing (McDonald et al., 2005a). For example, in the following sentence:



the head of the first *a* is *dog*, whereas the head of the second *a* is *cat*. Without considering positional information, a model cannot easily decide which *a* (nearer or farther) to assign to *dog*.

Long short-term memory networks (Hochreiter and Schmidhuber, 1997; LSTMs), a type of recurrent neural network with a more complex computational unit, have proven effective at capturing long-term dependencies. In our case LSTMs allow to represent each word on its own and within a sequence leveraging long-range contextual information. As shown in Figure 6.1, we first use a forward LSTM (LSTM^F) to read the sentence from left to right and then a backward LSTM (LSTM^B) to read the sentence from right to left, so that the entire sentence serves as context for each word:[2]

$$\mathbf{h}_i^F, \mathbf{c}_i^F = \text{LSTM}^F(\mathbf{x}_i, \mathbf{h}_{i-1}^F, \mathbf{c}_{i-1}^F) \tag{6.1}$$

$$\mathbf{h}_i^B, \mathbf{c}_i^B = \text{LSTM}^B(\mathbf{x}_i, \mathbf{h}_{i+1}^B, \mathbf{c}_{i+1}^B) \tag{6.2}$$

where $\mathbf{x}_i$ is the feature vector of word w_i, $\mathbf{h}_i^F \in \mathbb{R}^d$ and $\mathbf{c}_i^F \in \mathbb{R}^d$ are the hidden states and memory cells for the ith word w_i in LSTM^F and d is the hidden unit size. $\mathbf{h}_i^F$ is also the representation for $w_{0:i}$ (w_i and its left neighboring words) and $\mathbf{c}_i^F$ is an internal state maintained by LSTM^F. $\mathbf{h}_i^B \in \mathbb{R}^d$ and $\mathbf{c}_i^B \in \mathbb{R}^d$ are the hidden states and memory cells for the backward LSTM^B. Each token w_i is represented by $\mathbf{x}_i$, the concatenation of two vectors corresponding to w_i's lexical and POS tag embeddings:

$$\mathbf{x}_i = [\mathbf{W}_e \cdot e(w_i); \mathbf{W}_t \cdot e(t_i)] \tag{6.3}$$

where $e(w_i)$ and $e(t_i)$ are one-hot vector representations of token w_i and its POS tag t_i; $\mathbf{W}_e \in \mathbb{R}^{s \times |V|}$ and $\mathbf{W}_t \in \mathbb{R}^{q \times |T|}$ are the word and POS tag embedding matrices, where $|V|$ is the vocabulary size, s is the word embedding size, $|T|$ is the POS tag set size,

[2]For more detail on LSTM networks, see e.g., Graves (2012) or Goldberg (2016).

and q the tag embedding size. The hidden states of the forward and backward LSTMs are concatenated to obtain $\mathbf{a}_i$, the final representation of w_i:

$$\mathbf{a}_i = [\mathbf{h}_i^F; \mathbf{h}_i^B] \quad i \in [0, N] \tag{6.4}$$

Note that bidirectional LSTMs are one of many possible ways of representing word w_i. Alternative representations include embeddings obtained from feed-forward neural networks (Chen and Manning, 2014; Pei et al., 2015), character-based embeddings (Ballesteros et al., 2015), and more conventional features such as those introduced in McDonald et al. (2005a).

6.3.2 Head Selection

We now move on to discuss our formalization of dependency parsing as head selection. We begin with unlabeled dependencies and then explain how the model can be extended to predict labeled ones.

In a dependency tree, a head can have multiple dependents, whereas a dependent can have only one head. Based on this fact, dependency parsing can be formalized as follows. Given a sentence $S = (w_0, w_1, \ldots, w_N)$, we aim to find for each word $w_i \in \{w_1, w_2, \ldots, w_n\}$ the most probable head $w_j \in \{w_0, w_1, \ldots, w_N\}$. For example, in Figure 6.1, to find the head for the token *love*, we calculate probabilities $P_{head}(\text{ROOT}|\text{love}, S)$, $P_{head}(\text{kids}|\text{love}, S)$, and $P_{head}(\text{candy}|\text{love}, S)$, and select the highest. More formally, we estimate the probability of token w_j being the head of token w_i in sentence S as:

$$P_{head}(w_j|w_i, S) = \frac{\exp(g(\mathbf{a}_j, \mathbf{a}_i))}{\sum_{k=0}^{N} \exp(g(\mathbf{a}_k, \mathbf{a}_i))} \tag{6.5}$$

where $\mathbf{a}_i$ and $\mathbf{a}_j$ are vector-based representations of w_i and w_j, respectively (described in Section 6.3.1); $g(\mathbf{a}_j, \mathbf{a}_i)$ is a neural network with a single hidden layer that computes the associative score between representations $\mathbf{a}_i$ and $\mathbf{a}_j$:

$$g(\mathbf{a}_j, \mathbf{a}_i) = \mathbf{v}_a^\top \cdot \tanh(\mathbf{U}_a \cdot \mathbf{a}_j + \mathbf{W}_a \cdot \mathbf{a}_i) \tag{6.6}$$

where $\mathbf{v}_a \in \mathbb{R}^{2d}$, $\mathbf{U}_a \in \mathbb{R}^{2d \times 2d}$, and $\mathbf{W}_a \in \mathbb{R}^{2d \times 2d}$ are weight matrices of g. Note that the candidate head w_j can be the ROOT, while the dependent w_i cannot. Equations (6.5) and (6.6) compute the probability of adding an arc between two words, in a fashion similar to the neural attention mechanism in sequence-to-sequence models (Bahdanau et al., 2015).

We train our model by minimizing the negative log likelihood of the gold standard ⟨head, dependent⟩ arcs in all training sentences:

$$J(\theta) = -\frac{1}{|\mathcal{T}|} \sum_{S \in \mathcal{T}} \sum_{i=1}^{N_S} \log P_{head}(h(w_i)|w_i, S) \tag{6.7}$$

where $\mathcal{T}$ is the training set, $h(w_i)$ is w_i's gold standard head[3] within sentence S, and N_S the number of words in S (excluding ROOT). During inference, for each word w_i ($i \in [1, N_S]$) in S, we greedily choose the most likely head w_j ($j \in [0, N_S]$):

$$w_j = \underset{w_j : j \in [0, N_S]}{\arg\max}\ P_{head}(w_j|w_i, S) \tag{6.8}$$

Note that the prediction for each word w_i is made independently of the other words in the sentence.

Given our greedy inference method, there is no guarantee that predicted ⟨head, dependent⟩ arcs form a tree (maybe there are cycles). However, we empirically observed that most outputs during inference are indeed trees. For instance, on an English dataset, 95% of the arcs predicted on the development set are trees, and 87% of them are projective, whereas on a Chinese dataset, 87% of the arcs form trees, 73% of which are projective. This indicates that although the model does not explicitly model tree structure during training, it is able to figure out from the data (which consists of trees) that it should predict them.

So far we have focused on unlabeled dependencies, however it is relatively straightforward to extend DENSE to produce labeled dependencies. We basically train an additional classifier to predict labels for the arcs which have been already identified. The classifier takes as input features $[\mathbf{a}_i; \mathbf{a}_j; \mathbf{x}_i; \mathbf{x}_j]$ representing properties of the arc $\langle w_j, w_i \rangle$. These consist of $\mathbf{a}_i$ and $\mathbf{a}_j$, the LSTM-based representations for w_i and w_j (see Equation (6.4)), and their word and part-of-speech embeddings, $\mathbf{x}_i$ and $\mathbf{x}_j$ (see Equation (6.3)). Specifically, we use a trained DENSE model to go through the training corpus and generate features and corresponding dependency labels as training data. We employ a two-layer rectifier network (Glorot et al., 2011) for the classification task.

6.3.3 Maximum Spanning Tree Algorithms

As mentioned earlier, greedy inference may not produce well-formed trees. In this case, the output of DENSE can be adjusted with a maximum spanning tree algorithm.

[3]Note that $h(w_i)$ can be ROOT.

We use the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967) for building non-projective trees and the Eisner (Eisner, 1996) algorithm for projective ones.

Following McDonald et al. (2005a), we view a sentence $S = (w_0 = \text{ROOT}, w_1, \ldots, w_N)$ as a graph $G_S = \langle V_S, E_S \rangle$ with the sentence words and the dummy root symbol as vertices and a directed edge between every pair of distinct words and from the root symbol to every word. The directed graph G_S is defined as:

$$\begin{aligned} V_S &= \{w_0 = \text{ROOT}, w_1, \ldots, w_N\} \\ E_S &= \{\langle i, j \rangle : i \neq j, \langle i, j \rangle \in [0, N] \times [1, N]\} \\ s(i, j) &= P_{head}(w_i | w_j, S) \quad \langle i, j \rangle \in E_S \end{aligned}$$

where $s(i, j)$ is the weight of edge $\langle i, j \rangle$ and $P_{head}(w_i | w_j, S)$ is known. The problem of dependency parsing now boils down to finding the tree with the highest score which is equivalent to finding a MST in G_S (McDonald et al., 2005b).

Non-projective Parsing To build a non-projective parser, we solve the MST problem with the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967). The algorithm selects for each vertex (excluding ROOT) the in-coming edge with the highest weight. If a tree results, it must be the maximum spanning tree and the algorithm terminates. Otherwise, there must be a cycle which the algorithm identifies, contracts into a single vertex and recalculates edge weights going into and out of the cycle. The greedy inference strategy described in Equation (6.8) is essentially a sub-procedure in the Chu-Liu-Edmonds algorithm with the algorithm terminating after the first iteration. In implementation, we only run the Chu-Liu-Edmonds algorithm through graphs with cycles, i.e., non-tree outputs.

Projective Parsing For projective parsing, we solve the MST problem with the Eisner (Eisner, 1996) algorithm. The time complexity of the Eisner algorithm is $O(N^3)$, while checking if a tree is projective can be done reasonably faster, with a $O(N \log N)$ algorithm. Therefore, we only apply the Eisner algorithm to the non-projective output of our greedy inference strategy. Finally, it should be noted that the *training* of our model does not rely on the Chu-Liu-Edmonds or Eisner algorithm, or any other graph-based algorithm. MST algorithms are only used at *test* time to correct non-tree outputs which are a minority; DENSE acquires underlying tree structure constraints from the data without an explicit learning algorithm.

6.4 Parsing Experiments

We evaluated our parser in a projective and non-projective setting. In the following, we describe the datasets we used and provide training details for our models. We also present comparisons against multiple previous systems and analyze the parser's output.

6.4.1 Datasets

In the projective setting, we assessed the performance of our parser on the English Penn Treebank (PTB) and the Chinese Treebank 5.1 (CTB). Our experimental setup closely follows Chen and Manning (2014) and Dyer et al. (2015).

For English, we adopted the Stanford basic dependencies (SD) representation (De Marneffe et al., 2006).[4] We follow the standard splits of PTB, sections 2–21 were used for training, section 22 for development, and section 23 for testing. POS tags were assigned using the Stanford tagger (Toutanova et al., 2003) with an accuracy of 97.3%. For Chinese, we follow the same split of `CTB5` introduced in Zhang and Clark (2008). In particular, we used sections 001–815, 1001–1136 for training, sections 886–931, 1148–1151 for development, and sections 816–885, 1137–1147 for testing. The original constituency trees in CTB were converted to dependency trees with the Penn2Malt tool.[5] We used gold segmentation and gold POS tags as in Chen and Manning (2014) and Dyer et al. (2015).

In the non-projective setting, we assessed the performance of our parser on Czech and German, the largest non-projective datasets released as part of the CoNLL 2006 multilingual dependency parsing shared task[6]. Since there is no official development set in either dataset, we used the last 374/367 sentences in the Czech/German training set as development data.[7] Projective statistics of the four datasets are summarized in Table 6.1.

6.4.2 Training Details

We trained our models on an Nvidia GPU card; training takes one to two hours. Model parameters were uniformly initialized to $[-0.1, 0.1]$. We used Adam (Kingma and

[4] We obtained SD representations using the Stanford parser v.3.3.0.

[5] `http://stp.lingfil.uu.se/~nivre/research/Penn2Malt.html`

[6] The reason we choose the largest two datasets is because our model, which is based on LSTMs, might be data hungry. Vania et al. (2017) applies a similar model as ours (the main difference is they use character embeddings) to other non-projective languages and get pretty good results.

[7] We make the number of sentences in the development and test sets comparable.

Dataset	# Sentences	(%) Projective
English	39,832	99.9
Chinese	16,091	100.0
Czech	72,319	76.9
German	38,845	72.2

Table 6.1: Projective statistics on four datasets. Number of sentences and percentage of projective trees are calculated on the training set.

Ba, 2014) to optimize our models with hyper-parameters recommended by the authors (i.e., learning rate 0.001, first momentum coefficient 0.9, and second momentum coefficient 0.999). To alleviate the exploding gradient problem, we rescaled the gradient when its norm exceeded 5 (Pascanu et al., 2013). Dropout (Srivastava et al., 2014) was applied to our model with the strategy recommended in the literature (Zaremba et al., 2014; Semeniuta et al., 2016). On all datasets, we used two-layer LSTMs and set $d = s = 300$, where d is the hidden unit size and s is the word embedding size.

As in previous neural dependency parsing work (Chen and Manning, 2014; Dyer et al., 2015), we used pre-trained word vectors to initialize our word embedding matrix $\mathbf{W}_e$. For the PTB experiments, we used 300 dimensional pre-trained GloVe[8] vectors (Pennington et al., 2014). For the CTB experiments, we trained 300 dimensional GloVe vectors on the Chinese Gigaword corpus which we segmented with the Stanford Chinese Segmenter (Tseng et al., 2005). For Czech and German, we did not use pre-trained word vectors. The POS tag embedding size was set to $q = 30$ in the English experiments, $q = 50$ in the Chinese experiments and $q = 40$ in both Czech and German experiments.

6.4.3 Results

For both English and Chinese experiments, we report unlabeled (UAS) and labeled attachment scores (LAS) on the development and test sets; following Chen and Manning (2014) punctuation is excluded from the evaluation.

Experimental results on PTB are shown in Table 6.2. We compared our model with several recent papers following the same evaluation protocol and experimental settings. The first block in the table contains mostly graph-based parsers which do not use neural networks: Bohnet10 (Bohnet, 2010), Martins13 (Martins et al., 2013),

[8] http://nlp.stanford.edu/projects/glove/

Parser	Dev		Test	
	UAS	LAS	UAS	LAS
Bohnet10	—	—	92.88	90.71
Martins13	—	—	92.89	90.55
Z&M14	—	—	93.22	91.02
Z&N11	—	—	93.00	90.95
C&M14	92.00	89.70	91.80	89.60
Dyer15	93.20	90.90	93.10	90.90
Weiss15	—	—	93.99	92.05
Andor16	—	—	**94.61**	**92.79**
K&G16 *graph*	—	—	93.10	91.00
K&G16 *trans*	—	—	93.90	91.90
DENSE-Pei	90.77	88.35	90.39	88.05
DENSE-Pei+E	91.39	88.94	91.00	88.61
DENSE	94.17	91.82	94.02	91.84
DENSE+E	**94.30**	**91.95**	94.10	91.90

Table 6.2: Results on English dataset (PTB with Stanford Dependencies). +E: we post-process non-projective output with the Eisner algorithm.

and Z&M14 (Zhang and McDonald, 2014). Z&N11 (Zhang and Nivre, 2011) is a transition-based parser with non-local features. Accuracy results for all four parsers are reported in Weiss et al. (2015).

The second block in Table 6.2 presents results obtained from neural network-based parsers. C&M14 (Chen and Manning, 2014) is a transition-based parser using features learned with a feed forward neural network. Although very fast, its performance is inferior compared to graph-based parsers or strong non-neural transition based parsers (e.g., Z&N11). Dyer15 (Dyer et al., 2015) uses (stack) LSTMs to model the states of the buffer, the stack, and the action sequence of a transition system. Weiss15 (Weiss et al., 2015) is another transition-based parser, with a more elaborate training procedure. Features are learned with a neural network model similar to C&M14, but much larger with two layers. The hidden states of the neural network are then used to train a structured perceptron for better beam search decoding. Andor16 (Andor et al., 2016) is similar to Weiss15, but uses a globally normalized training algorithm instead.

Unlike all models above, DENSE does not use any kind of transition- or graph-

Parser	Dev		Test	
	UAS	LAS	UAS	LAS
Z&N11	—	—	86.00	84.40
Z&M14	—	—	**87.96**	**86.34**
C&M14	84.00	82.40	83.90	82.40
Dyer15	87.20	**85.90**	87.20	85.70
K&G16 *graph*	—	—	86.60	85.10
K&G16 *trans*	—	—	87.60	86.10
DENSE-Pei	82.50	80.74	82.38	80.55
DENSE-Pei+E	83.40	81.63	83.46	81.65
DENSE	87.27	85.73	87.63	85.94
DENSE+E	**87.35**	85.85	87.84	86.15

Table 6.3: Results on Chinese dataset (CTB). +E: we post-process non-projective outputs with the Eisner algorithm.

Parser	PTB		CTB	
	Dev	Test	Dev	Test
C&M14	43.35	40.93	32.75	32.20
Dyer15	51.94	50.70	**39.72**	**37.23**
DENSE	51.24	49.34	34.74	33.66
DENSE+E	**52.47**	**50.79**	36.49	35.13

Table 6.4: UEM results on PTB and CTB.

based algorithm during training and inference. Nonetheless, it obtains a UAS of 94.02%. Around 95% of the model's outputs after inference are trees, 87% of which are projective. When we post-process the remaining 13% of non-projective outputs with the Eisner algorithm (DENSE+E), we obtain a slight improvement on UAS (94.10%).

Kiperwasser and Goldberg (2016) extract features from bidirectional LSTMs and feed them to a graph- (K&G16 *graph*) and transition-based parser (K&G16 *trans*). Their LSTMs are jointly trained with the parser objective. DENSE yields very similar performance to their transition-based parser while it outperforms K&G16 *graph*. A key difference between DENSE and K&G16 lies in the training objective. The objective of DENSE is log-likelihood based *without* tree structure constraints (the model is trained to produce a distribution over possible heads for each word, where each head selec-

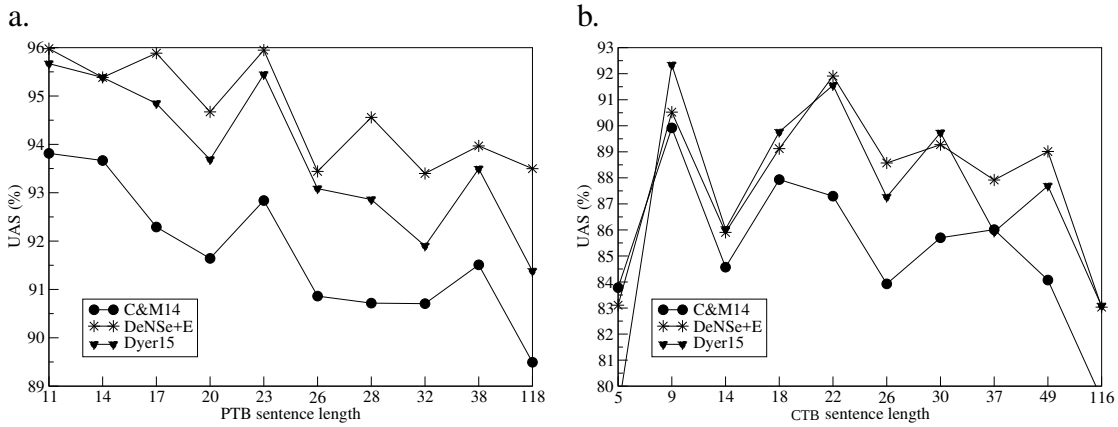


Figure 6.2: UAS against sentence length on PTB and CTB (development set). Sentences are sorted by length in ascending order and divided equally into 10 bins. The horizontal axis is the length of the last sentence in each bin.

tion is independent), while K&G16 employ a max-margin objective *with* tree structure constraints. Although our probabilistic objective is non-structured, it is perhaps easier to train compared to a margin-based one.

We also assessed the importance of the bidirectional LSTM on its own by replacing our LSTM-based features with those obtained from a feed-forward network. Specifically, we used the 1-order-atomic features introduced in Pei et al. (2015) which represent POS-tags, modifiers, heads, and their relative positions. As can be seen in Table 6.2 (DENSE-Pei), these features are less effective compared to LSTM-based ones and the contribution of the MST algorithm (Eisner) during decoding is more pronounced (DENSE-Pei+E). We observe similar trends in the Chinese, German, and Czech datasets (see Tables 6.3 and 6.5).

Results on CTB follow a similar pattern. As shown in Table 6.3, DENSE outperforms all previous neural models (see the test set columns) on UAS and LAS. DENSE performs competitively with Z&M14, a non-neural model with a complex high order decoding algorithm involving cube pruning and strategies for encouraging diversity. Post-processing the output of the parser with the Eisner algorithm generally improves performance (by 0.21%; see last row in Table 6.3). Again we observe that 1-order-atomic features (Pei et al., 2015) are inferior compared to the LSTM. Table 6.4 reports unlabeled sentence level exact match (UEM) in Table 6.4 for English and Chinese. Interestingly, even when using the greedy inference strategy, DENSE yields a UEM comparable to Dyer15 on PTB. Finally, in Figure 6.2 we analyze the performance of

Parser	Czech		German	
	UAS	LAS	UAS	LAS
MST-1st	86.18	—	89.54	—
MST-2nd	87.30	—	90.14	—
Turbo-1st	87.66	—	90.52	—
Turbo-3rd	90.32	—	**92.41**	—
RBG-1st	87.90	—	90.24	—
RBG-3rd	**90.50**	—	91.97	—
DENSE-Pei	86.00	77.92	89.42	86.48
DENSE-Pei+CLE	86.52	78.42	89.52	86.58
DENSE	89.60	81.70	92.15	89.58
DENSE+CLE	89.68	81.72	92.19	89.60

Table 6.5: Non-projective results on the CoNLL 2006 dataset. +CLE: we post-process non-tree outputs with the Chu-Liu-Edmonds algorithm.

our parser on sentences of different length. On both PTB and CTB, DENSE has an advantage on long sentences compared to C&M14 and Dyer15.

For Czech and German, we closely follow the evaluation setup of CoNLL 2006. We report both UAS and LAS, although most previous work has focused on UAS. Our results are summarized in Table 6.5. We compare DENSE against three non-projective graph-based dependency parsers: the MST parser (McDonald et al., 2005a), the Turbo parser (Martins et al., 2013), and the RBG parser (Lei et al., 2014). We show the performance of these parsers in the first order setting (e.g., MST-1st) and in higher order settings (e.g., Turbo-3rd). The results of MST-1st, MST-2nd, RBG-1st and RBG-3rd are reported in Lei et al. (2014) and the results of Turbo-1st and Turbo-3rd are reported in Martins et al. (2013). We show results for our parser with greedy inference (see DENSE in the table) and when we use the Chu-Liu-Edmonds algorithm to post-process non-tree outputs (DENSE+CLE).

As can been seen, DENSE outperforms all other first (and second) order parsers on both German and Czech. As in the projective experiments, we observe slight a improvement (on both UAS and LAS) when using a MST algorithm. On German, DENSE is comparable with the best third-order parser (Turbo-3rd), while on Czech it lags behind Turbo-3rd and RBG-3rd. This is not surprising considering that DENSE is a first-order parser and only uses words and POS tags as features. Comparison systems

Dataset	#Sent	Before MST Tree	Before MST Proj	After MST Tree	After MST Proj
PTB	1,700	95.1	86.6	100.0	100.0
CTB	803	87.0	73.1	100.0	100.0
Czech	374	87.7	65.5	100.0	72.7
German	367	96.7	67.3	100.0	68.1

Table 6.6: Percentage of trees and projective trees on the development set before and after DENSE uses a MST algorithm. On PTB and CTB, we use the Eisner algorithm and on Czech and German, we use the Chu-Liu-Edmonds algorithm.

use a plethora of hand-crafted features and more sophisticated high-order decoding algorithms. Finally, note that a version of DENSE with features in Pei et al. (2015) is consistently worse (see the second block in Table 6.5).

Our experimental results demonstrate that using a MST algorithm during inference can slightly improve the model's performance. We further examined the extent to which the MST algorithm is necessary for producing dependency trees. Table 6.6 shows the percentage of trees before and after the application of the MST algorithm across the four languages. In the majority of cases DENSE outputs trees (ranging from 87.0% to 96.7%) and a significant proportion of them are projective (ranging from 65.5% to 86.6%). Therefore, only a small proportion of outputs (14.0% on average) need to be post-processed with the Eisner or Chu-Liu-Edmonds algorithm.

6.5 Dependency Language Modeling Experiments

We have shown that on PTB, the DENSE parser performs well. In this section, we would like to assess the performance of our parser on downstream tasks. Specifically, we apply it to dependency language models and see if language modeling performance can be improved on the MSR Sentence Completion Challenge, which was used in Chapter 5 to assess the performance our tree based language models (TREELSTM AND LDTREELSTM).

The task in the MSR Sentence Completion Challenge (Zweig et al., 2012) is to select the correct missing word for 1,040 SAT-style test sentences when presented with five candidate completions. The training set contains 522 novels from the Project Gutenberg which we preprocessed as follows. After removing headers and footers

Model	PPL	Accuracy
TREELSTM + C&M14	67.84	56.73
TREELSTM + DENSE	64.90	57.69
LDTREELSTM + C&M14	57.24	60.67
LDTREELSTM + DENSE	56.17	59.13

Table 6.7: Model accuracies on the MSR sentence completion task. TREELSTM + C&M14 and LDTREELSTM + C&M14 use the C&M14 parser (Chen and Manning, 2014). As their names indicate, TREELSTM + DENSE and LDTREELSTM + DENSE use the DENSE dependency parser.

from the files, we tokenized the dataset with the Stanford Core NLP toolkit (Manning et al., 2014). Sentences are parsed into dependency trees with the C&M14 parser (Chen and Manning, 2014) and with our DENSE parser, respectively. We therefore obtained two preprocessed versions of the MSR dataset. Following the settings in Zhang et al. (2016), we converted all words to lower case and replaced those occurring five times or less with UNK. The resulting vocabulary size was 65,346 words. We randomly sampled 4,000 sentences from the training set as our validation set.

As shown in Table 6.7, with a better parser, both TREELSTM and LDTREELSTM obtain lower perplexities (see TREELSTM + DENSE and LDTREELSTM + DENSE rows). With DENSE, TREELSTM improves the final accuracy of sentence completions over C&M14. However, this is not the case for LDTREELSTM. This is perhaps due to the fact that LDTREELSTM is more robust to parsing errors. Compared to TREELSTM, in addition to parent nodes, right dependents also rely on left dependents. Therefore, when a parse assigns a wrong parent to a right dependent, TREELSTM is not likely to make a correct prediction, while LDTREELSTM may still make a correct prediction based on its left dependents (maybe some of its left dependents are parsed correctly).

6.6 Conclusions

In this work we presented DENSE, a neural dependency parser which we train without a transition system or graph-based algorithm. Experimental results show that DENSE achieves competitive performance across four different languages and can seamlessly transfer from a projective to a non-projective parser simply by changing the post-

processing MST algorithm during inference. We also found DENSE can improve tree structured based language models such as Zhang et al. (2016). In the future, we plan to increase the coverage of our parser by using tri-training techniques (Li et al., 2014) and multi-task learning (Luong et al., 2015a).

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In 1990s and early 2000s, most natural language generation models adopted a pipelined architecture. This architecture typically involved three modules, namely *content planning*, *sentence planning* and *surface realization*. With the recent resurgence of deep neural networks and their applications to natural language processing, natural language generation (especially single sentence generation) models have been greatly simplified. Neural based generation models usually have two neural networks: the *Representation* network for learning the input representations and the *Generation* network for generating natural language sentences by using the learned representations. These two networks are usually trained jointly. In this thesis, we focus on natural language generation problems, which we model by developing neural network models which adopt two different views: sequence learning and tree structured learning.

In sequence based generation models, sentences are viewed as sequences of words. Recurrent neural networks, which are designed for sequence predictions, are naturally employed in these models. However, most recurrent network based generation models are targeting at generating single sentences. A hierarchical recurrent model is proposed in Chapter 3, which is a better *Representation* network. It is designed to generation paragraph level text. Specifically, a word level recurrent neural network is used to generate the current sentence and a sentence level recurrent neural network is used to model sentences which have been generated before. The model is applied to the task of Chinese poetry generation and is shown to outperform non-hierarchical recurrent neural networks in terms of perplexity and it is also superior to a statistical machine translation based model (He et al., 2012) and a summarization based (Yan et al., 2013)

poetry generator.

Most neural based generation models are data-driven and rely heavily on end to end training. However, end to end training can be a double edged sword. On the one hand, it greatly simplifies the training process in a fully data-driven setting. At the same time, it is difficult to incorporate task-specific constraints. Therefore, in Chapter 4, we proposed a deep reinforcement learning based framework to model task specific prior knowledge. In essence, we proposed a better training method for neural language generation models. The framework was applied to the task of sentence simplification, which aims to produce a simplified form for a complex sentence. We used the sequence to sequence with attention model as a base model and modelled prior knowledge of this task (i.e. the simplified sentences should be simplier, meaning preserving and grammatical) as different rewards in a deep reinforcement learning framework. Moreover, since lexical simplification is quite important, we designed a neural lexical simplification model, which further boosts the performance. Experimentally, we demonstrated the reinforcement learning framework is key to producing simpler sentences. This framework can also be used in other generation tasks such as summarization or concept-to-text generation.

Generating sentences as sequences of words is a great simplification for natural language generation tasks. This simplification makes NLG tasks easier, since there are many existing sequence prediction models (e.g. Recurrent Neural Networks) can be used. However, is generating a sentence as a sequence of words the only way (or the best way) to generate a sentence? Chapter 5 tries to answer this question by proposing a tree structured generation model called *top-down tree long short-term memory network*, which is actually a better *Generation* network. It generates dependency trees rather than linear structure sentences. We empirically demonstrate that this model outperforms LSTMs in the context of language modeling (MSR sentence completion challenge) and it can also generate sentences in the form of dependency trees. However, generating (dependency) trees is difficult. Chapter 6 tries to explore whether better dependency parsing can improve the tree structured model we proposed in Chapter 5 and the answer is yes.

7.2 Future Work

Avenues for future work are many and varied. The poetry generation model we proposed in this thesis is targets the multiple sentence generation problem. It may also be

used in potential tasks such as writing an essay or news article given text descriptions (e.g., keywords) or article structures. However, these tasks are challenging and even hierarchical recurrent neural networks may not handle them well. It will be interesting to model discourse structure explicitly in hierarchical recurrent neural networks.

Sequence to sequence model with attention (Bahdanau et al., 2015) becomes a very strong baseline in machine translation and many (single sentence) generation tasks. However, tree representations in NLP still have advantages as we have demonstrated in chapter 6. Therefore, a sequence-to-tree or tree-to-tree model for machine translation and natural language generation should be explored. However, generating trees is not easy (since structure constraints need to be taken into account) and the resulting model may generate ill-formed trees because of lacking global constraints. We can leverage reinforcement learning to integrate global constraints during training. We may also put the sequence (or tree) to tree model in a generative adversarial network (GAN) framework (Goodfellow et al., 2014), where the sequence (or tree) to tree model is the generator, while a dependency parser can be used as a discriminator. This tree-to-tree model is likely to work on the sentence simplification task, where simplification rules apply on the syntactic level going beyond mere word substitution.

Text rewriting tasks (e.g. sentence simplification or sentence compression) are a special class of generation problems, where the generated sentence should always preserve the meaning of the original sentence. Many existing models treat these problems as a standard sequence to sequence learning problem or variants (e.g. coupled with reinforcement learning in Chapter 4). However, the meaning preservation constraint can be modelled from another prospective. For example, this constraint can be captured in an auto-encoding framework (Hinton and Salakhutdinov, 2006; Kingma and Ba, 2014). Besides using a simplification (or compression) model to produce a simpler (or shorter) sentence, we can also use the sentence produced by the rewriting model to recover the original sentence. This model can be trained by minimizing the reconstruction error. The auto-encoding framework can be used in a semi-supervised setting (Miao and Blunsom, 2016) or even a unsupervised setting.

Bibliography

Agirrezabal, M., Arrieta, B., Astigarraga, A., and Hulden, M. (2013). POS-Tag Based Poetry Generation with WordNet. In *Proceedings of the 14th European Workshop on Natural Language Generation*, pages 162–166, Sofia, Bulgaria.

Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., Petrov, S., and Collins, M. (2016). Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany.

Angeli, G., Liang, P., and Klein, D. (2010). A simple domain-independent probabilistic approach to generation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 502–512. Association for Computational Linguistics.

Auli, M., Galley, M., Quirk, C., and Zweig, G. (2013). Joint Language and Translation Modeling with Recurrent Neural Networks. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1044–1054, Seattle, Washington, USA.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *ICLR 2015*.

Ballesteros, M., Dyer, C., and Smith, N. A. (2015). Improved transition-based parsing by modeling characters instead of words with LSTMs. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 349–359, Lisbon, Portugal.

Barzilay, R. and Lapata, M. (2005). Collective content selection for concept-to-text generation. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 331–338. Association for Computational Linguistics.

Barzilay, R. and Lapata, M. (2006). Aggregation via set partitioning for natural language generation. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 359–366. Association for Computational Linguistics.

Bateman, J. A. (1997). Enabling technology for multilingual natural language generation: the kpml development environment. *Natural Language Engineering*, 3(1):15–55.

Beigman Klebanov, B., Knight, K., and Marcu, D. (2004). Text simplification for information-seeking applications. In *Proceedings of ODBASE*, volume 3290 of *Lecture Notes in Computer Science*, pages 735–747, Agia Napa, Cyprus. Springer.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155.

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

Bohnet, B. (2010). Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 89–97, Beijing, China.

Bordes, A., Weston, J., Collobert, R., and Bengio, Y. (2011). Learning structured embeddings of knowledge bases. In *Conference on artificial intelligence*, number EPFL-CONF-192344.

Callison-Burch, C., Koehn, P., Monz, C., Post, M., Soricut, R., and Specia, L. (2012). Findings of the 2012 Workshop on Statistical Machine Translation. In *Proceedings of the 7th Workshop on Statistical Machine Translation*, pages 10–51, Montréal, Canada.

Carreras, X. (2007). Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 957–961, Prague, Czech Republic.

Carreras, X. and Collins, M. (2009). Non-projective parsing for statistical machine translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 200–209, Singapore.

Carroll, J., Minnen, G., Pearce, D., Canning, Y., Devlin, S., and Tait, J. (1999). Simplifying text for language-impaired readers. In *Proceedings of the 9th EACL*, pages 269–270, Bergen, Norway.

Chambers, N. and Jurafsky, D. (2008). Unsupervised learning of narrative event chains. In *ACL*, volume 94305, pages 789–797. Citeseer.

Chandrasekar, R., Doran, C., and Srinivas, B. (1996). Motivations and methods for text simplification. In *Proceedings of the 16th COLING*, pages 1041–1044, Copenhagen, Denmark.

Charniak, E. (2001). Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 124–131. Association for Computational Linguistics.

Chelba, C., Engle, D., Jelinek, F., Jimenez, V., Khudanpur, S., Mangu, L., Printz, H., Ristad, E., Rosenfeld, R., Stolcke, A., et al. (1997). Structure and performance of a dependency language model. In *EUROSPEECH*. Citeseer.

Chelba, C. and Jelinek, F. (2000). Structured language modeling. *Computer Speech and Language*, 14(4):283–332.

Chen, D. and Manning, C. (2014). A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar. Association for Computational Linguistics.

Chen, X., Liu, X., Gales, M., and Woodland, P. (2015). Recurrent neural network language model training with noise contrastive estimation for speech recognition. In *In 40th IEEE International Conference on Accoustics, Speech and Signal Processing*, pages 5401–5405, Brisbane, Australia.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Chopra, S., Auli, M., and Rush, A. M. (2016). Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference*

of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 93–98, San Diego, California.

Chu, Y.-J. and Liu, T.-H. (1965). On shortest arborescence of a directed graph. *Scientia Sinica*, 14(10):1396.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Cohn, T. and Lapata, M. (2009). Sentence compression as tree transduction. *Journal of Artificial Intelligence Research*, 34:637–674.

Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8.

Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011). Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376.

Colton, S., Goodwin, J., and Veale, T. (2012). Full-FACE Poetry Generation. In *Proceedings of the International Conference on Computational Creativity*, pages 95–102, Dublin, Ireland.

Crammer, K. and Singer, Y. (2001). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292.

Crammer, K. and Singer, Y. (2003). Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991.

Cross, J. and Huang, L. (2016). Incremental parsing with minimal features using bi-directional LSTM. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 32–37, Berlin, Germany.

Curran, J., Clark, S., and Bos, J. (2007). Linguistically motivated large-scale NLP with C&C and Boxer. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 33–36, Prague, Czech Republic.

Dai, A. M. and Le, Q. V. (2015). Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, pages 3079–3087.

De Marneffe, M.-C., MacCartney, B., Manning, C. D., et al. (2006). Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454.

Devlin, J., Zbib, R., Huang, Z., Lamar, T., Schwartz, R., and Makhoul, J. (2014). Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1370–1380. Association for Computational Linguistics.

Devlin, S. (1999). *Simplifying Natural Language for Aphasic Readers*. PhD thesis, University of Sunderland.

Duboue, P. A. and McKeown, K. R. (2002). Content planner construction via evolutionary algorithms and a corpus-based fitness function. In *Proceedings of INLG 2002*, pages 89–96.

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

Durrani, N., Haddow, B., Koehn, P., and Heafield, K. (2014). Edinburgh's phrase-based machine translation systems for WMT-14. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 97–104. Association for Computational Linguistics Baltimore, MD, USA.

Dyer, C., Ballesteros, M., Ling, W., Matthews, A., and Smith, N. A. (2015). Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343, Beijing, China. Association for Computational Linguistics.

Dyer, C., Weese, J., Setiawan, H., Lopez, A., Ture, F., Eidelman, V., Ganitkevitch, J., Blunsom, P., and Resnik, P. (2010). cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of*

the ACL 2010 System Demonstrations, pages 7–12. Association for Computational Linguistics.

Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71(4):233–240.

Eisner, J. (2000). Bilexical grammars and their cubic-time parsing algorithms. In *Advances in probabilistic and other parsing technologies*, pages 29–61. Springer.

Eisner, J. M. (1996). Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345. Association for Computational Linguistics.

Elhadad, M. and Robin, J. (1996). An overview of surge: A reusable comprehensive syntactic realization component. Technical report, Technical Report 96-03, Ben Gurion University, Dept. of Computer Science, Beer Sheva, Israel.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

Emami, A., Xu, P., and Jelinek, F. (2003). Using a connectionist model in a syntactical based language model. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 372–375, Hong Kong, China.

Evans, R., asan, C. O., and Dornescu, I. (2014). An evaluation of syntactic simplification rules for people with autism. In *Proceedings of the 3rd Workshop on Predicting and Improving Text Readability for Target Reader Populations (PITR)*, pages 131–140, Gothenburg, Sweden.

Fellbaum, C., editor (1998). *WordNet: An Electronic Database*. MIT Press, Cambridge, MA.

Filippova, K., Alfonseca, E., Colmenares, C. A., Kaiser, L., and Vinyals, O. (2015). Sentence compression by deletion with LSTMs. In *EMNLP*, pages 360–368.

Freund, Y. and Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296.

Fundel, K., Küffner, R., and Zimmer, R. (2007). Relation extraction using dependency parse trees. *Bioinformatics*, 23(3):365–371.

Ganitkevitch, J., Van Durme, B., and Callison-Burch, C. (2013). PPDB: The paraphrase database. In *Proceedings of NAACL-HLT*, pages 758–764, Atlanta, Georgia. Association for Computational Linguistics.

Gao, J., He, X., Yih, W.-t., and Deng, L. (2014). Learning continuous phrase representations for translation modeling. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 699–709. Association for Computational Linguistics.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323.

Goldberg, E., Driedger, N., and Kittredge, R. I. (1994). Using natural-language processing to produce weather forecasts. *IEEE Expert*, 9(2):45–53.

Goldberg, Y. (2016). A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420.

Goldberg, Y. and Nivre, J. (2012). A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*. Studies in Computational Intelligence. Springer.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE.

Greene, E., Bodrumlu, T., and Knight, K. (2010). Automatic Analysis of Rhythmic Poetry with Applications to Generation and Translation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 524–533, Cambridge, MA.

Grishman, R., Macleod, C., and Meyers, A. (1994). Comlex syntax: Building a computational lexicon. In *Proceedings of the 15th conference on Computational linguistics-Volume 1*, pages 268–272. Association for Computational Linguistics.

Gubbins, J. and Vlachos, A. (2013). Dependency language models for sentence completion. In *EMNLP*, pages 1405–1410, Seattle, Washington, USA. Association for Computational Linguistics.

Gutmann, M. U. and Hyvärinen, A. (2012). Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *The Journal of Machine Learning Research*, 13(1):307–361.

Hall, K. (2007). K-best spanning tree parsing. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 392–399, Prague, Czech Republic.

He, J., Zhou, M., and Jiang, L. (2012). Generating Chinese Classical Poems with Statistical Machine Translation Models. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, pages 1650–1656, Toronto, Canada.

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

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

Heafield, K. (2011). KenLM: Faster and Smaller Language Model Queries. In *Proceedings of the EMNLP 2011 Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, United Kingdom.

Henderson, J. (2004). Discriminative training of a neural network statistical parser. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 95–102, Barcelona, Spain.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

Hochreiter, S. (1998). Vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, 6(2):107–116.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

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

Hovy, E. H. (1993). Automated discourse generation using discourse structure relations. *Artificial intelligence*, 63(1):341–385.

Huang, L. and Sagae, K. (2010). Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086, Uppsala, Sweden.

Inui, K., Fujita, A., Takahashi, T., Iida, R., and Iwakura, T. (2003). Text simplification for reading assistance: A project note. In *Proceedings of the Second International Workshop on Paraphrasing*, pages 9–16, Sapporo, Japan. Association for Computational Linguistics.

Jean, S., Firat, O., Cho, K., Memisevic, R., and Bengio, Y. (2015). Montreal neural machine translation systems for WMT15. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 134–140.

Jiang, L. and Zhou, M. (2008). Generating Chinese Couplets using a Statistical MT Approach. In *Proceedings of the 22nd International Conference on Computational Linguistics*, pages 377–384, Manchester, UK.

Kaji, N., Kawahara, D., Kurohashi, S., and Sato, S. (2002). Verb paraphrase based on case frame alignment. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 215–222, Philadelphia, Pennsylvania, USA.

Kalchbrenner, N. and Blunsom, P. (2013). Recurrent Continuous Translation Models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, Seattle, Washington.

Kasper, R. T. (1989). A flexible interface for linking applications to penman's sentence generator. In *Proceedings of the workshop on Speech and Natural Language*, pages 153–158. Association for Computational Linguistics.

Katz, S. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE transactions on acoustics, speech, and signal processing*, 35(3):400–401.

Kauchak, D. (2013). Improving text simplification language modeling using unsimplified text data. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1537–1546, Sofia, Bulgaria. Association for Computational Linguistics.

Kim, J. and Mooney, R. J. (2010). Generative alignment and semantic parsing for learning from ambiguous supervision. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 543–551. Association for Computational Linguistics.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751. Association for Computational Linguistics.

Kincaid, J. P., Fishburne Jr, R. P., Rogers, R. L., and Chissom, B. S. (1975). Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel. Technical report, DTIC Document.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kiperwasser, E. and Goldberg, Y. (2016). Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Kiros, R., Salakhutdinov, R., and Zemel, R. S. (2014). Multimodal neural language models. In *ICML*, volume 14, pages 595–603.

Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184. IEEE.

Knight, K. and Hatzivassiloglou, V. (1995). Two-level, many-paths generation. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 252–260. Association for Computational Linguistics.

Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical Phrase-based Translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54, Edmonton, Canada.

Konstas, I. and Lapata, M. (2012). Concept-to-text generation via discriminative reranking. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 369–378, Jeju Island, Korea. Association for Computational Linguistics.

Koo, T., Rush, A. M., Collins, M., Jaakkola, T., and Sontag, D. (2010). Dual decomposition for parsing with non-projective head automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1288–1298, Cambridge, MA.

Korhonen, A., Krymolowski, Y., and Briscoe, T. (2006). A large subcategorization lexicon for natural language processing applications. In *Proceedings of LREC*, volume 6.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Kübler, S., McDonald, R., Nivre, J., and Hirst, G. (2009). *Dependency Parsing*. Morgan and Claypool Publishers.

Lavoie, B. and Rambow, O. (1997). A fast and portable realizer for text generation systems. In *Proceedings of the fifth conference on Applied natural language processing*, pages 265–268. Association for Computational Linguistics.

Le, Q. V., Jaitly, N., and Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*.

Lebret, R., Grangier, D., and Auli, M. (2016). Neural text generation from structured data with application to the biography domain. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1203–1213. Association for Computational Linguistics.

Lei, T., Xin, Y., Zhang, Y., Barzilay, R., and Jaakkola, T. (2014). Low-rank tensors for scoring dependency structures. In *Proceedings of the 52nd Annual Meeting of the*

Association for Computational Linguistics (Volume 1: Long Papers), pages 1381–1391, Baltimore, Maryland.

Li, J., Monroe, W., Ritter, A., Jurafsky, D., Galley, M., and Gao, J. (2016). Deep reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1192–1202, Austin, Texas. Association for Computational Linguistics.

Li, Z., Zhang, M., and Chen, W. (2014). Ambiguity-aware ensemble training for semi-supervised dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 457–467, Baltimore, Maryland.

Ling, W., Blunsom, P., Grefenstette, E., Hermann, K. M., Kočiský, T., Wang, F., and Senior, A. (2016). Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 599–609, Berlin, Germany.

Liu, W. (1735). *ShiXueHanYing*.

Lu, W. and Ng, H. T. (2011). A probabilistic forest-to-string model for language generation from typed lambda calculus expressions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1611–1622. Association for Computational Linguistics.

Luong, M.-T., Le, Q. V., Sutskever, I., Vinyals, O., and Kaiser, L. (2015a). Multi-task sequence to sequence learning. In *Proceedings of the 4th International Conference on Learning Representations*, San Juan, Puerto Rico.

Luong, T., Pham, H., and Manning, C. D. (2015b). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

Mann, W. C. and Thompson, S. A. (1988). Rhetorical structure theory: Toward a functional theory of text organization. *Text-Interdisciplinary Journal for the Study of Discourse*, 8(3):243–281.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

Manurung, R. (2003). *An Evolutionary Algorithm Approach to Poetry Generation*. PhD thesis, University of Edinburgh.

Manurung, R., Ritchie, G., and Thompson, H. (2012). Using Genetic Algorithms to Create Meaningful Poetic Text. *Journal of Experimental & Theoretical Artificial Intelligence*, 24(1):43–64.

Martins, A., Almeida, M., and Smith, N. A. (2013). Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria.

Mayberry, M. R. and Miikkulainen, R. (1999). SardSrn: A neural network shift-reduce parser. In *In Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 820–825, Stockholm, Sweden.

McDonald, R., Crammer, K., and Pereira, F. (2005a). Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 91–98, Ann Arbor, Michigan.

McDonald, R., Pereira, F., Ribarov, K., and Hajic, J. (2005b). Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada.

McDonald, R. T. and Pereira, F. C. (2006). Online learning of approximate dependency parsing algorithms. In *EACL*.

McIntyre, N. and Lapata, M. (2009). Learning to tell tales: A data-driven approach to story generation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 217–225, Singapore.

Mei, H., Bansal, M., and Walter, M. R. (2016). What to talk about and how? selective generation using LSTMs with coarse-to-fine alignment. In *Proceedings of the 2016*

Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 720–730, San Diego, California. Association for Computational Linguistics.

Melčuk, I. A. (1988). *Dependency syntax: theory and practice*. SUNY press.

Miao, Y. and Blunsom, P. (2016). Language as a latent variable: Discrete generative models for sentence compression. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 319–328, Austin, Texas. Association for Computational Linguistics.

Mikolov, T. (2012). *Statistical Language Models based on Neural Networks*. PhD thesis, Brno University of Technology.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. In *Proceedings of the 2013 International Conference on Learning Representations*, Scottsdale, Arizona, USA.

Mikolov, T., Deoras, A., Povey, D., Burget, L., and Cernocky, J. (2011a). Strategies for Training Large Scale Neural Network Language Models. In *Proceedings of ASRU 2011*, pages 196–201, Hilton Waikoloa Village, Big Island, Hawaii, US.

Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. (2010). Recurrent Neural Network based Language Model. In *Proceedings of INTERSPEECH*, pages 1045–1048, Makuhari, Japan.

Mikolov, T., Kombrink, S., Burget, L., Cernocky, J., and Khudanpur, S. (2011b). Extensions of Recurrent Neural Network Language Model. In *Proceedings of the 2011 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5528–5531, Prague, Czech Republic.

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

Mikolov, T. and Zweig, G. (2012). Context Dependent Recurrent Neural Network Language Model. In *Proceedings of 2012 IEEE Workshop on Spoken Language Technology*, pages 234–239, Miami, Florida.

Mirowski, P. and Vlachos, A. (2015). Dependency recurrent neural language models for sentence completion. In *ACL*, pages 511–517, Beijing, China. Association for Computational Linguistics.

Mitchell, J. and Lapata, M. (2010). Composition in Distributional Models of Semantics. *Cognitive Science*, 34(8):1388–1439.

Mnih, A. and Hinton, G. (2007). Three new graphical models for statistical language modelling. In *Proceedings of the 24th International Conference on Machine Learning*, pages 641–648.

Mnih, A. and Kavukcuoglu, K. (2013). Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems 26*, pages 2265–2273.

Mnih, A. and Teh, Y. W. (2012). A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1751–1758, Edinburgh, Scotland.

Narasimhan, K., Yala, A., and Barzilay, R. (2016). Improving information extraction by acquiring external evidence with reinforcement learning. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2355–2365, Austin, Texas. Association for Computational Linguistics.

Narayan, S. and Gardent, C. (2014). Hybrid simplification using deep semantics and machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 435–445, Baltimore, Maryland. Association for Computational Linguistics.

Netzer, Y., Gabay, D., Goldberg, Y., and Elhadad, M. (2009). Gaiku: Generating Haiku with Word Associations Norms. In *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, pages 32–39, Boulder, Colorado.

Nivre, J. (2003). An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 149–160, Nancy, France.

Nivre, J. (2004). Incrementality in deterministic dependency parsing. In Keller, F., Clark, S., Crocker, M., and Steedman, M., editors, *Proceedings of the ACL Work-*

shop Incremental Parsing: Bringing Engineering and Cognition Together, pages 50–57, Barcelona, Spain.

Nivre, J. (2008). Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Nivre, J., Hall, J., and Nilsson, J. (2006a). Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of LREC*, volume 6, Genoa, Italy.

Nivre, J., Hall, J., Nilsson, J., Eryiğit, G., and Marinov, S. (2006b). Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 221–225, New York City.

Och, F. J. (2003). Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 160–167, Sapporo, Japan. Association for Computational Linguistics.

Oliveira, H. G. (2012). PoeTryMe: a Versatile Platform for Poetry Generation. *Computational Creativity, Concept Invention, and General Intelligence*, 1:21.

Palangi, H., Deng, L., Shen, Y., Gao, J., He, X., Chen, J., Song, X., and Ward, R. (2016). Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 24(4):694–707.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318.

Pei, W., Ge, T., and Chang, B. (2015). An effective neural network model for graph-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 313–322, Beijing, China. Association for Computational Linguistics.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–43.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, 1–2(46):77–105.

Ranzato, M., Chopra, S., Auli, M., and Zaremba, W. (2016). Sequence level training with recurrent neural networks. *ICLR, San Juan, Puerto Rico*.

Reiter, E. and Dale, R. (1997). Building applied natural language generation systems. *Natural Language Engineering*, 3(01):57–87.

Reiter, E. and Dale, R. (2000). *Building Natural Language Generation Systems*. Cambridge University Press, New York, NY, USA.

Reiter, E., Mellish, C., and Levine, J. (1995). Automatic generation of technical documentation. *Applied Artificial Intelligence an International Journal*, 9(3):259–287.

Rello, L., Bayarri, C., Górriz, A., Baeza-Yates, R., Gupta, S., Kanvinde, G., Saggion, H., Bott, S., Carlini, R., and Topac, V. (2013). Dyswebxia 2.0!: More accessible text for people with dyslexia. In *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility*, pages –, Brazil.

Roark, B. (2001). Probabilistic top-down parsing and language modeling. *Computational linguistics*, 27(2):249–276.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

Ryang, S. and Abekawa, T. (2012). Framework of automatic text summarization using reinforcement learning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 256–265, Jeju Island, Korea. Association for Computational Linguistics.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Semeniuta, S., Severyn, A., and Barth, E. (2016). Recurrent dropout without memory loss. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1757–1766, Osaka, Japan.

Sennrich, R. (2015). Modelling and optimizing on syntactic n-grams for statistical machine translation. *Transactions of the Association for Computational Linguistics*, 3:169–182.

Shardlow, M. (2014). A survey of automated text simplification. *International Journal of Advanced Computer Science and Applications*, pages 581–701. Special Issue on Natural Language Processing.

Shen, L., Xu, J., and Weischedel, R. (2008). A new string-to-dependency machine translation algorithm with a target dependency language model. In *Proceedings of ACL-08: HLT*, pages 577–585, Columbus, Ohio, USA.

Siddharthan, A. (2004). Syntactic simplification and text cohesion. *Research on Language and Computation*, 4(1):77–109.

Siddharthan, A. (2014). A survey of research on text simplification. *International Journal of Applied Linguistics*, 165(2):259–298.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *ICLR*.

Smith, D. A. (2010). *Efficient inference for trees and alignments: modeling monolingual and bilingual syntax with hard and soft constraints and latent variables*. Johns Hopkins University.

Smith, D. A. and Eisner, J. (2006). Quasi-synchronous grammars: Alignment by soft projection of syntactic dependencies. In *Proceedings of the Workshop on Statistical Machine Translation*, pages 23–30. Association for Computational Linguistics.

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Makhoul, J. (2006). A study of translation edit rate with targeted human annotation. In *Proceedings of association for machine translation in the Americas*, volume 200.

Snow, R., Jurafsky, D., and Ng, A. Y. (2005). Learning syntactic patterns for automatic hypernym discovery. In *Advances in Neural Information Processing Systems 17*, pages 1297–1304, Vancouver, British Columbia.

Socher, R., Huang, E. H., Pennington, J., Manning, C. D., and Ng, A. (2011a). Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems*, pages 801–809.

Socher, R., Huval, B., Manning, C. D., and Ng, A. Y. (2012). Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1201–1211, Jeju Island, Korea. Association for Computational Linguistics.

Socher, R., Pennington, J., Huang, E. H., Ng, A. Y., and Manning, C. D. (2011b). Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 151–161, Edinburgh, Scotland, UK.

Specia, L., Jauhar, S. K., and Mihalcea, R. (2012). Semeval-2012 task 1: English lexical simplification. In *In Proceedings of *SEM 2012*, pages 347–355, Montréal, Canada.

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

Sukhbaatar, S., Weston, J., Fergus, R., et al. (2015). End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448.

Sundermeyer, M., Schlüter, R., and Ney, H. (2012). LSTM neural networks for language modeling. In *Interspeech*, pages 194–197.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9.

Tai, K. S., Socher, R., and Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long*

Papers), pages 1556–1566, Beijing, China. Association for Computational Linguistics.

Titov, I. and Henderson, J. (2007). Constituent parsing with incremental sigmoid belief networks. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 632–639, Prague, Czech Republic.

Tosa, N., Obara, H., and Minoh, M. (2008). Hitch Haiku: An Interactive Supporting System for Composing Haiku Poem How I Learned to Love the Bomb: Defcon and the Ethics of Computer Games. In *Proceedings of the 7th International Conference on Entertainment Computing*, pages 209–216, Pittsburgh, PA.

Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics.

Tran, K., Bisazza, A., and Monz, C. (2016). Recurrent memory networks for language modeling. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 321–331. Association for Computational Linguistics.

Tseng, H., Chang, P., Andrew, G., Jurafsky, D., and Manning, C. (2005). A conditional random field word segmenter for Sighan bakeoff 2005. In *Proceedings of the 4th SIGHAN workshop on Chinese language Processing*, pages 168–171, Jeju Island, Korea.

Vania, C., Zhang, X., and Lopez, A. (2017). Uparse: the edinburgh system for the conll 2017 ud shared task. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 100–110.

Vaswani, A., Zhao, Y., Fossum, V., and Chiang, D. (2013). Decoding with large-scale neural language models improves translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1387–1392, Seattle, Washington, USA.

Venugopalan, S., Xu, H., Donahue, J., Rohrbach, M., Mooney, R., and Saenko, K. (2015). Translating videos to natural language using deep recurrent neural networks. In *Proceedings of the 2015 Conference of the North American Chapter of the*

Association for Computational Linguistics: Human Language Technologies, pages 1494–1504, Denver, Colorado. Association for Computational Linguistics.

Vickrey, D. and Koller, D. (2008). Sentence simplification for semantic role labeling. In *Proceedings of ACL-08: HLT*, pages 344–352, Columbus, OH.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2017). Show and tell: Lessons learned from the 2015 mscoco image captioning challenge. *IEEE transactions on pattern analysis and machine intelligence*, 39(4):652–663.

Wang, L. (2002). *A Summary of Rhyming Constraints of Chinese Poems (Shi Ci Ge Lv Gai Yao)*. Beijing Press, 2002.

Wang, Q., Luo, T., Wang, D., and Xing, C. (2016). Chinese song iambics generation with neural attention-based model. *arXiv preprint arXiv:1604.06274*.

Watanabe, W. M., Junior, A. C., de Uzêda, V. R., de Mattos Fortes, R. P., Pardo, T. A. S., and Aluísio, S. M. (2009). Facilita: reading assistance for low-literacy readers. In *Proceedings of the 27th ACM International Conference on Design of Communication*, Bloomington, IN.

Weiss, D., Alberti, C., Collins, M., and Petrov, S. (2015). Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 323–333, Beijing, China.

Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4):339–356.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Woodsend, K. and Lapata, M. (2011). Learning to simplify sentences with quasi-synchronous grammar and integer programming. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 409–420, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Woodsend, K. and Lapata, M. (2014). Text rewriting improves semantic role labeling. *Journal of Artificial Intelligence Research*, 51:133–164.

Wu, X., Tosa, N., and Nakatsu, R. (2009). New Hitch Haiku: An Interactive Renku Poem Composition Supporting Tool Applied for Sightseeing Navigation System. In *Proceedings of the 8th International Conference on Entertainment Computing*, pages 191–196, Paris, France.

Wubben, S., Van Den Bosch, A., and Krahmer, E. (2012). Sentence simplification by monolingual machine translation. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 1015–1024. Association for Computational Linguistics.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A. C., Salakhutdinov, R., Zemel, R. S., and Bengio, Y. (2015a). Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, volume 14, pages 77–81.

Xu, W., Callison-Burch, C., and Napoles, C. (2015b). Problems in current text simplification research: New data can help. *Transactions of the Association for Computational Linguistics*, 3:283–297.

Xu, W., Napoles, C., Pavlick, E., Chen, Q., and Callison-Burch, C. (2016). Optimizing statistical machine translation for text simplification. *Transactions of the Association for Computational Linguistics*, 4:401–415.

Yamada, H. and Matsumoto, Y. (2003). Statistical dependency analysis with support vector machines. In *Proceedings of the 8th Workshop on Parsing Technologies*, pages 195–206, Nancy, France.

Yamada, K. and Knight, K. (2001). A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 523–530. Association for Computational Linguistics.

Yan, R. (2016). i, poet: Automatic poetry composition through recurrent neural networks with iterative polishing schema. In *IJCAI*, pages 2238–2244.

Yan, R., Jiang, H., Lapata, M., Lin, S.-D., Lv, X., and Li, X. (2013). I, Poet: Automatic Chinese Poetry Composition Through a Generative Summarization Framework Under Constrained Optimization. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pages 2197–2203, Beijing, China.

Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Zhang, H. and McDonald, R. (2012). Generalized higher-order dependency parsing with cube pruning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 320–331, Jeju Island, Korea.

Zhang, H. and McDonald, R. (2014). Enforcing structural diversity in cube-pruned dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 656–661, Baltimore, Maryland.

Zhang, X., Cheng, J., and Lapata, M. (2017). Dependency parsing as head selection. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 665–676, Valencia, Spain. Association for Computational Linguistics.

Zhang, X. and Lapata, M. (2014). Chinese poetry generation with recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 670–680, Doha, Qatar.

Zhang, X. and Lapata, M. (2017). Sentence simplification with deep reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Copenhagen, Denmark.

Zhang, X., Lu, L., and Lapata, M. (2016). Top-down tree long short-term memory networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 310–320, San Diego, California.

Zhang, Y. (2009). *Structured language models for statistical machine translation*. PhD thesis, Johns Hopkins University.

Zhang, Y. and Clark, S. (2008). A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571, Honolulu, Hawaii.

Zhang, Y. and Nivre, J. (2011). Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, Oregon, USA.

Zhou, C.-L., You, W., and Ding, X. (2010). Genetic Algorithm and its Implementation of Automatic Generation of Chinese SongCi. *Journal of Software*, pages 427–437.

Zhu, Z., Bernhard, D., and Gurevych, I. (2010). A monolingual tree-based translation model for sentence simplification. In *Proceedings of the 23rd international conference on computational linguistics*, pages 1353–1361. Association for Computational Linguistics.

Zweig, G. and Burges, C. J. (2012). A challenge set for advancing language modeling. In *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, pages 29–36, Montréal, Canada.

Zweig, G. and Makarychev, K. (2013). Speed Regularization and Optimality in Word Classing. In *Proceedings of the 2014 IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 8237–8241, Florence, Italy.

Zweig, G., Platt, J. C., Meek, C., Burges, C. J., Yessenalina, A., and Liu, Q. (2012). Computational approaches to sentence completion. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 601–610. Association for Computational Linguistics.