# Variable length word encodings for neural translation models

**Jiameng Gao**

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
*Master of Philosophy*

Peterhouse

August 11, 2016

# Declaration

I, Jiameng Gao of Peterhouse, being a candidate for the M.Phil in Machine Learning, Speech and Language Technology, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 11,708

**Signed:**

**Date:** 11/08/16

# Acknowledgements

Here is a group of people who, perhaps against their will, have had to put up with me over the year and unlike softmax layers in neural networks, I'll never tire to thank them for making this most challenging year the most rewarding year I've had in Cambridge.

I'd like to thank Prof. Bill Byrne for giving me the patience and time to think things through during our meetings, and guiding me through this year, accompanied by TensorFlow and FSTs, two things no one should ever live without.

I'd also like to thank Felix and Eva for helping with all my problems with TensorFlow and FSTs, without their help this disseration would not have been very long.

I'm incredibly thankful for everyone in the MPhil class, especially for their enthusiasm for things that weren't speech practicals. I'm lucky to have met so many wonderful people at the best MCR in Cambridge. I've absolutely loved 24 Parkside, everyone here had made it a fantastic place to live - the kitchen will be have more cupboard space soon!

Thanks go to Jenny and Hannah for always being there when I make too much food for dinner. Tom B and Tom B, thanks for putting up with my antics for so many years, no one else could do it. Dan, you're infinitely more knowledgeable than I could ever be. Josh and Hannah, I'll always be open to make more websites. Many thanks to Joanna and Alex for years of Union reunions. Amy, you've been my bestest friend, and thanks for teaching me that sometimes even law firms don't give people enough work to do - Jiamy4ever.

Finally, I can't thank my Mum and Dad enough for everything.

# Abstract

As described by Zipf's law, vast proportions of words in most natural languages occur very infrequently, meaning that in a statistical learning framework these words would be poorly modelled for any given corpus of practical, finite size. This is known as the *rare word problem.*

This is often worsened by the implementation of neural networks, since neural networks make use of softmax functions in the output layer. Evaluating these functions for each word in the vocabulary becomes computationally expensive in both training and run-time, as the normalisation constant is a summation of exponentials in the order $O(|\mathcal{V}|)$, where $\mathcal{V}$ is the output vocabulary. A simple method to counteract this is to label rare and infrequent words in the training corpus as an unknown word token, thereby limiting the size of the vocabulary.

Nevertheless, doing so only worsens the fact that, for a given translation corpus, the vocabulary may not include all words in the natural languages involved. This is an issue in agglutinative languages, where unseen, legitimate words can be formed from known constituent words, making them difficult to model. This is known as the *unknown word problem.*

We attempt to implement LSTM recurrent neural network language models based on Zaremba et al. (2014) for rescoring in the Syntactical-Guided Neural Machine Translation system by Stahlberg et al. (2016). Further, we compare methods to compress the vocabulary of a language through certain coding schemes, in order to reduce the impact of the computationally expensive softmax layer, and investigate there are better ways to represent natural language in neural network structures other than using tokens on the word level. Previous work have focused on word encodings for end-to-end translation systems, while we primarily focus on the more general application of language modelling.

We implement byte-pair encoding, Huffman coding and character decomposition, and compare their performance in a SMT setting to a truncated vocabulary that is standard for neural network language models. We find that BPE is better than character-level decomposition as it is more efficient, especially for the most frequent words, and it gives higher BLEU scores than Huffman coding and is able to decompose new words in the language. We also show that by using byte-pair encoding we can improve the BLEU score performance for SMT systems over that of a truncated vocabulary, investigating whether this is due to the efficient decomposition or the enlarged vocabulary.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

This dissertation concerns itself with the implementation of neural network language models in a statistical machine translation (SMT) setting, as well as the challenges posed by working with neural networks, namely the "unknown word problem" and the "rare word problem" (Barber and Botev (2016); Jean et al. (2015); Luong et al. (2014)).

We attempt to implement variants of LSTM recurrent neural network language models based on Zaremba et al. (2014) for rescoring in the Syntactical-Guided Neural Machine Translation system by Stahlberg et al. (2016). Further, we compare methods to compress the vocabulary of a language through certain coding schemes and investigate whether there are better ways to represent natural language in neural network structures other than using tokens on the word level. Previous work (Chitnis and DeNero (2015); Sennrich et al. (2015)) have focused on word encodings for end-to-end translation systems, while we primarily focus on the more general application of language modelling.

## 1.1 Statistical machine translation

Statistical machine translation (SMT) systems are designed to automatically translate one natural language to another without human intervention. They are designed to utilise statistical patterns in existing parallel training texts to predict the translation of new inputs to the system.

Chiang (2007) proposed one of the standards of modern machine translation in the form of hierarchical phrase-based models. These systems utilise synchronous probabilistic context free grammars to construct translation models from parallelised text. Often times, one could take the one-best outputs produced by these systems, or word lattices could be produced from them so that further feature scores could be used to rescore the lattice in order to produce better results.

Recent interest in neural networks has led to their application in statistical machine translation problems; many have used neural networks for tasks such as language modelling (Mikolov (2012); Vaswani et al. (2013); Devlin et al. (2014)), while more recently interests gathered around end-to-end translation systems that replace standard systems with neural networks (Sutskever et al. (2014); Bahdanau et al. (2015)).

Nonetheless, whereas neural networks structures have been successfully applied to problems such as speech recognition (Hinton et al. (2012); Dahl et al. (2012)) and language modelling in general (Bengio et al. (2006); Jozefowicz et al. (2016)), many of the problems associated with existing machine translation systems have not been directly addressed by the implementation of neural networks.

## 1.2 Rare word problem

As described by Zipf's law (Zipf (2016)), vast proportions of words in most natural languages occur very infrequently, meaning that in a statistical learning framework these words would be poorly modelled for any given corpus of practical, finite size. This is known as the *rare word problem*.

This is often worsened by the implementation of neural networks, since neural networks make use of softmax functions (eq. 1.1) in the output layer. Evaluating these functions for each word in the vocabulary becomes computationally expensive in both training and runtime, as the normalisation constant is a summation of exponentials in the order $O(|\mathcal{V}|)$, where $\mathcal{V}$ is the output vocabulary.

$$\alpha_i = \frac{\exp \langle \mathbf{w}_{l,i}, \mathbf{h}_{l-1} \rangle}{\sum_{i' \in \mathcal{I}} \exp \langle \mathbf{w}_{l,i'}, \mathbf{h}_{l-1} \rangle} \tag{1.1}$$

A simple method to counteract this is to label rare and infrequent words in the training corpus as an unknown word token (typically `<UNK>`), thereby limiting the size of the vocabulary and allow us to build more robust systems for the most frequent $K << |\mathcal{V}|$ words in the vocabulary. This also means we can significantly increase the speed of training and evaluation of neural networks by keeping the complexity to $O(K)$.

## 1.3 Unknown word problem

Nevertheless, doing so only worsens the fact that, for a given translation corpus, the vocabulary may not include all words in the natural languages involved. Lack of specific information for a given word in the source language could mean that significant information for the output is lost and reduce the robustness of the system. This is an issue in agglutinative languages or languages with large numbers of compounds, since previously unseen, legitimate words can be formed from known constituent words, impeding the performance of the translation system. This is known as the *unknown word problem*.

This is especially a problem for neural network end-to-end systems, where the vocabulary is constrained. However, if neural networks are only used for scoring in a language model setting in conjunction with syntactic translation system, this becomes less of an issue, as syntactic translation systems generally have much larger vocabulary capacities, since they are not tied to a output softmax function as neural networks are.

## 1.4 Contribution of the dissertation

The contribution of this dissertation is two-fold:

- We implement a target side RNN language model based on Zaremba et al. (2014) using TensorFlow (Abadi et al. (2016)) in the framework established by Stahlberg et al. (2016), and thereby allowing lattice rescoring in conjunction with Hiero scores.
- We perform word encodings such as those described by Sennrich et al. (2015) and Chitnis and DeNero (2015) in a language model setting, using these language models to rescore lattices and investigate the effect of these encodings on the BLEU score. The encoding can be performed either on-line during rescoring or through transforming word lattices into subword lattices.

## 1.5 Organisation of the thesis

In Chapter 2, we review the basics of neural networks and classical language modelling, and how the former can be used for language modelling. We also discuss the construction of statistical machine translation systems, where we look at the makeup of syntax-based machine translation, as well as how neural networks are used to undertake the task. In Chapter 3, we summarise existing methods for large vocabulary language modelling and SMT tasks through neural network structures and compare their respective performances. We detail and investigate the ways in which we perform vocabulary compression or word decompositions in Chapter 4. We then describe our SMT experiments and report their results in Chapter 5.

# Chapter 2

# Background

In this chapter, we provide the background for the technologies involved in statistical language modelling and machine translation systems. We describe classical approaches to both of these topics as well as how neural networks can be used to perform these tasks. Significant parts of language modelling are described in further detail in Jurafsky and Martin (2014), while Bishop (2006) provide detailed descriptions for the workings of neural networks.

## 2.1 Language modelling

Language modelling is the process of obtaining the overall probabilities of word sequences. These probabilities are generally approximated from an existing corpus of data during a training process, while in decoding the language model (LM) would estimate the probability of words.

To model the joint probability of a sentence $P(x_1, x_2, \ldots, x_S)$, we split this into:

$$P(x_1, x_2, \ldots, x_S) = \prod_{i=1}^{S} P(x_i | x_{i-1}, x_{i-2}, \ldots, x_1) \tag{2.1}$$

$$= \prod_{i=1}^{S} P(x_i | \mathbf{h}) \tag{2.2}$$

Thus we aim to find the posterior probability - $P(x_i|\mathbf{h})$ - where $x_i$ is the event of word $x$ occurring at position $i$ and $\mathbf{h}$ is a history vector of the information provided by the previous words. As such, we aim to maximise the joint probability of valid sentences in a natural language, and lower them if they're not.

Generally, language models are evaluated through the *perplexity* score (equation 2.3) for a test or evaluation dataset. The perplexity is essentially the distance between a predicted word distribution and the expected distribution measured by the cross-entropy between them. This means better, more generalising language models have lower perplexities (i.e. higher overall posterior probabilities) on evaluation and test datasets.

$$Perp = 2^H \tag{2.3}$$

$$\text{s.t.} \quad H = -\sum_i P(w_i|\mathbf{h}) \log_2 \hat{P}(w_i|\mathbf{h}) \tag{2.4}$$

Where $P(w_i|\mathbf{h})$ is the empirical distribution of the test set, and $\hat{P}(w_i|\mathbf{h})$ is the predicted model distribution.

### 2.1.1 N-gram language models

Kneser and Ney (1995) describes the back-off N-gram language models commonly used today. The model consists of two components: an N-gram language model, where the history vector is represented as the prior $N$ words before position $i$, and a back-off component, which retrieves probabilities from the $N-1$-gram probabilities. They are then balanced to a discounting factor such that $\sum_{i \in \mathcal{V}} P(x_i|\mathbf{h}) = 1$. The following equation represents the maximum-likelihood probability estimate for an N-gram language model, where $h \equiv x_1^{i-1}$, where $C$ is a frequency threshold below which the back-off mechanism takes effect.

$$P(x_i|\mathbf{h}) = P(x_i|x_{i-1}, x_{i-2}, \ldots, x_1) \tag{2.5}$$

$$\simeq P(x_i|x_{i-1}, x_{i-2}, \ldots, x_{i-N}) \tag{2.6}$$

$$\text{s.t.} \quad P(x_i|\mathbf{h}) = \begin{cases} d(f)\frac{f(x_i \cap x_{i-1} \cap x_{i-2} \cap \cdots \cap x_{i-N})}{f(x_{i-1} \cap x_{i-2} \cap \cdots \cap x_{i-N})} & \text{if } f \geq C \\ \alpha(x_i) P(x_i|x_{i-1}, x_{i-2}, \ldots, x_{i-N+1}) & \text{otherwise} \end{cases} \tag{2.7}$$

While it is preferable to use higher orders of $N$-grams to obtain more accurate estimates to exploit the history, the "curse of dimensionality" means that for a given vocabulary $\mathcal{V}$, the size of the language model increase in the order of $O(V^N)$ for complete sets of $N$-grams. Further, the training data also needs to increase in the same order of magnitude to make sure different $N$-grams occur frequently enough to obtain accurate estimations, and it also increases the computation load for retrieving these $N$-gram estimations. This is particularly a problem in the setting of large vocabulary translation tasks. Modern methods to work around the problem involve using distributed systems to store N-gram LMs, such as those proposed in Brants et al. (2007); Heafield et al. (2013).

### 2.1.2 Neural network language models

Artificial neural networks are a family of discriminative, non-linear models that use error back-propagation as training procedure to approximate functions. The most common models used today are feed-forward multilayer perceptrons (figure 2.1).

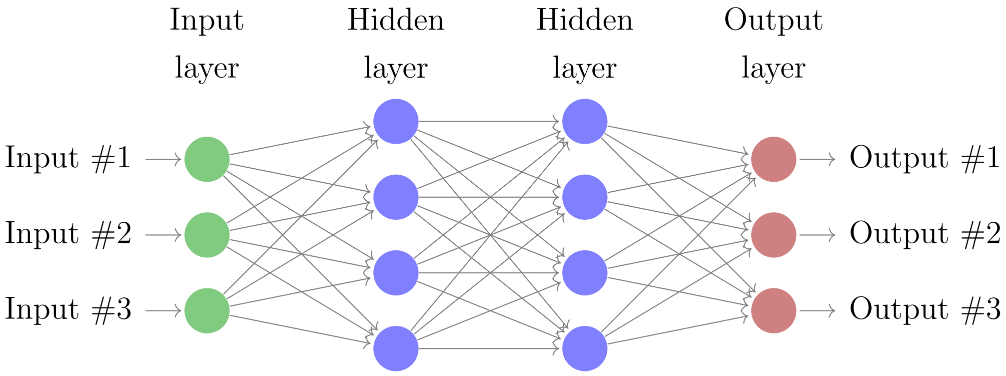


Figure 2.1: A neural network with two hidden layers

Generally a multilayer neural network consists of a set of inputs, several layers of hidden nodes and an output layer. For transitions from one layer to the next, each arc is weighted according to a weight matrix $\mathbf{W^{(k)}}$. The sink nodes then gather their weighted inputs and apply some non-linear function and outputs to the next layer. In this way, the set of inputs $\mathbf{x}$ of hidden layer $k$, out of $K$ layers in total, are calculated:

$$\mathbf{x}^{(k)} = f(\mathbf{z}^{(k)}) \tag{2.8}$$

$$\text{s.t. } \mathbf{z}^{(k)} = \mathbf{W}^{(k)}\mathbf{x}^{(k-1)} + \mathbf{b}^{(k)} \tag{2.9}$$

Where $f()$ is the activation function (typically tanh, ReLU, sigmoids or softmax) and $\mathbf{b}^{(k)}$ is a set of bias offsets for layer $k$.

Feed-forward neural networks are generally trained for their function approximations using error back-propagation (Rumelhart et al. (1986)). This is done by finding the error $E$ in the output layers through some measure, and calculating the error gradient with respect to each of the layers. This process is called back propagation because the error gradient at a specific layer $k$ is dependent on layer $k+1$. Equation 2.10 shows the error calculated as the cross-entropy (i.e. the log-perplexity in language modelling), for training inputs of $\mathbf{x}_1, \ldots, \mathbf{x}_N$, outputs $y_1, \ldots, y_N$ and targets $t_1, \ldots, t_N$, and an output layer of $|\mathbf{V}|$ classes. A description of error back-propagation can be found in Appendix A.1

$$E = -\sum_{i \in \mathcal{V}} \sum_{n=1}^{N} (t_{n,i} \log(y_i(\mathbf{x}_n)) + (1 - t_{n,i}) \log(1 - y_i(\mathbf{x}_n))) \tag{2.10}$$

### Feed-forward neural network language models

Bengio et al. (2006) describes a competitive example of using neural networks as probabilistic language models. A standard feed-forward neural network language model is set up similar to fig. 2.1, where the output layer activation function is that of the softmax (eq. 1.1), such that the $i$th output node represents the posterior probability of the $i$th word $P(w_i|\mathbf{h})$. This means our distribution for target word $j$ would be in the form of one-hot encoding:

$$P_{t_j}(w_i|\mathbf{h}) = \begin{cases} 1 & i = j \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

Bengio et al. (2006) proposes that, instead of feeding in the word context using one-hot encoding, directly into the first hidden layer, there would be an intermediate layer, where each word in the input vocabulary $\mathcal{V}$ would be represented by a distributed vector $\mathbf{c} \in \mathbb{R}^k$, where $k$ is the fixed dimensionality of the vector. The entire mapping of the vocabulary can be represented as a matrix $\mathbf{C} \in \mathbb{R}^{|\mathcal{V}|,k}$, and the vector representations can be learnt through the error back-propagation like most neural network layers.

Nevertheless, as we have described in section 1.2, since we use the one-hot encoding in the output layer, this means the complexity of the output layer softmax function scales with the size of the vocabulary. While we can often compare the "logits" (the inner product $\langle \mathbf{w_{l,i}}, \mathbf{h_{l-1}} \rangle$) of the softmax function during decoding, thereby sparing exponential and reduced sum operation. This cannot be avoided during training, since the calculation of the error gradient requires all of the posterior probabilities in the output layer.

Devlin et al. (2014) also describe a feed-forward neural network "joint model", which is a language model taking in the source language into its context. The model takes, for the $i$th target word, in the input a context formed of the $i$th word in the source sentence, the $\pm m$ words and the prior $n$ words in the target language, all of which are encoded in some trained vector representation. As such, the system resembles that of a $(n+m)$-gram language model. This is particularly beneficial in a neural network settings, since if we take values of $n$ and $m$ to be 4 and 11 respectively, a 15-gram language model would be far too sparse to be trained through Kneser-Ney models, whereas neural network based models can be more easily trained to do this.

### Recurrent neural network language models

Recurrent neural networks (fig. 2.2) add the concept of recursiveness and dynamism to the neural network architecture by using recurrent hidden layers. The simplest method to do this is to feed the output of a recurrent hidden layer from the previous time step back into its input in the current step (Rumelhart et al. (1986)).

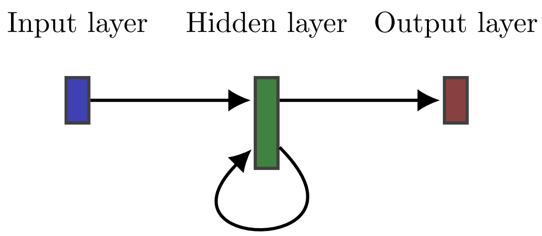


Figure 2.2: Simple recurrent neural network

This recursive structure intends to utilise information from previous inputs for the output of the current step in the shape of the history vector. Such a structure is particularly useful since context is no longer needed in the input, as the history vector should be able to be trained to preserve the necessary, useful information from the previous inputs.

However, the recursiveness means the recurrent hidden layers can not be trained using simple error back-propagation, since the history vector would go back into itself. Because of this, recurrent neural networks are generally trained using error back-propagation through time (Robinson (1989); Werbos (1990)), where the recurrent neural network is modeled as an unfolded version of itself and is trained in a similar way to a feed-forward neural network using back-propagation. The unfolded structure is shown in Fig. 2.3. Here the weight adjustments would be done as an average of the error gradients of the same weights. As such, the further the unfolded length, the longer the neural network would be trained to remember.

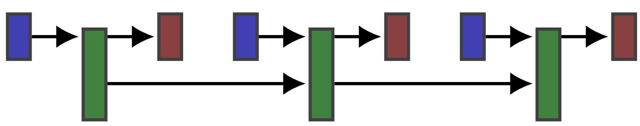

Figure 2.3: Unfolded RNN structure for training with back-propagation through time

RNNs are essentially dynamic probabilistic models and Mikolov et al. (2010) provides a description of RNNs as language models, in a structure similar to the feed-forward neural network language model, except we no longer need to have contexts in the input. The discrete sequence in language suits the discrete time steps that recurrent neural networks model, and Mikolov (2012) has shown that they perform better than existing Kneser-Ney language models.

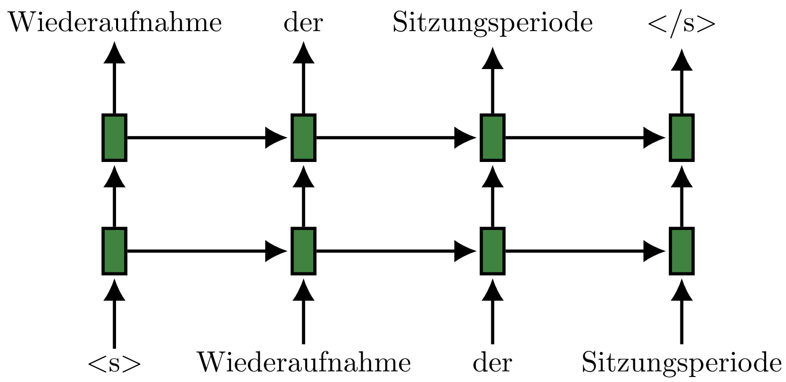


Figure 2.4: A RNN Language Model

### LSTM language models

While the recursiveness of RNNs suggest that the network could store information for an arbitrary period, Bengio et al. (1994) proves that the basic RNN structure fundamentally suffers from the "vanishing gradient", and its counterpart "exploding gradient", problem. The problem comes down to the issue that for a standard RNN, the weight matrix and the non-linear activation function can be viewed as a simple mapping function. This means that if this mapping function, in the case of neural networks the weight matrix and the activation function, which we shall call $\mathcal{W}$, is such that $|\mathcal{W}| > 1$, the history vector would grow exponentially as unit time increases, destroying signals by amplifying noise within the system. On the contrary, if a stable mapper is developed, where the norm of mapping function is less than unity $|\mathcal{W}| < 1$, the gradient decreases exponentially with unit time, thereby eliminating past components of data along with noise. While, arguably, the mapping may lie on the boundary such that $|\mathcal{W}| = 1$, this condition is very rarely satisfied given that the system behaves in discrete time while noise is prevalent. It also suggests the further input must either be in the null-space of the mapping function, meaning that no new information could be stored in the RNN, or move carefully along the surface of $|\mathcal{W}| = 1$.

Typically the "exploding gradient" problem is avoided by limiting the maximum of the norm of gradient updates in back-propagation-through-time training, but the problem of the "vanishing gradient" is non-trivial to solve. As a way to combat this, a mechanism called Long Short-Term Memory (LSTM) was devised (Hochreiter and Schmidhuber (1997)), where gating mechanisms in the LSTM cell can decide whether to store and utilise past data. Graves et al. (2013) popularised LSTM structures by using them to tackle phone-level speech recognition tasks.

Sundermeyer et al. (2012) describes the use of LSTM cells instead of standard RNN cells in language models; as shown in figure 2.5, LSTM cells contain an input gate, output gate and a forget gate, where each $b_i = tanh(a_i)$. Meanwhile, Zaremba et al. (2014) applies dropout as a method of regularisation for LSTM cells during training. This is done by randomly deactivating specific arcs in 2.4 according to some probability.

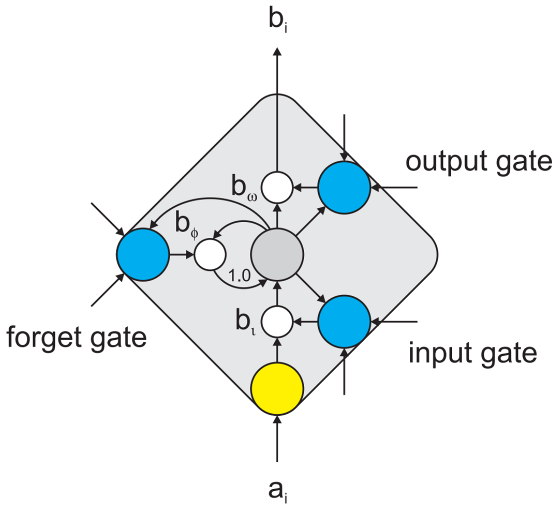


Figure 2.5: LSTM structure. Taken from Sundermeyer et al. (2012)

## 2.2 Statistical machine translation

Statistical machine translation is based on the concept of a source-channel model, where we would like to find the target sentence $\mathbf{e}$ in a known language, given the foreign source sentence $\mathbf{f}$, such that:

$$\hat{\mathbf{e}} = \arg\max_{\mathbf{e}} P(\mathbf{e}|\mathbf{f}) \tag{2.12}$$

$$= \arg\max_{\mathbf{e}} \frac{P(\mathbf{f}|\mathbf{e})P(\mathbf{e})}{P(\mathbf{f})} \tag{2.13}$$

$$= \arg\max_{\mathbf{e}} P(\mathbf{f}|\mathbf{e})P(\mathbf{e}) \tag{2.14}$$

Here, $P(\mathbf{f}|\mathbf{e})$ is generally known as the translation model, while $P(\mathbf{e})$ is the language model.

### Evaluation

One of the difficulties of machine translation tasks is the nebulous definition of a "good translation", which itself is the subject of discussion for many books and essays (such as Hofstadter (1997); Tolkien (1940)). In general, the task of translation itself is often one where there is no single right answer, as the same sentence could be translated differently due to its context, the linguistic ability and idiosyncrasies of the translator, or simply situations where a double meaning in the sentence cannot be perfectly mapped to the target language. This is generally problematic for SMT tasks as we often only have one reference sentence per source sentence for evaluation, meaning that even if a translated sentence is semantically and syntactically correct, the system will still be penalised if it is not translated in the same way as the reference. Nonetheless, we try to avoid these ambiguity issues by choosing news commentary and reporting for our datasets, as it is a domain where ambiguities are generally few between languages.

Cer et al. (2010) investigate a wide range of statistical evaluation metrics for SMT tasks. These include the Translation Error Rate (where error types include insertions, deletions, substitutions and adjacent swaps) and its variants; METEOR, a metric that combines the unigram harmonic mean ($F_\alpha$) and a fragmentation penalty$P_{\beta,\gamma}$:

$$P_{\beta,\gamma} = 1 - \gamma(\frac{fragments}{unigrammatches})^\beta \tag{2.15}$$

$$F_\alpha = \frac{PR}{\alpha P + (1-\alpha)R} \tag{2.16}$$

$$\text{s.t. } \mathrm{METEOR}_{\alpha,\beta,\gamma} = F_\alpha P_{\beta,\gamma} \tag{2.17}$$

Where $P$ is the fraction of unigram matches compared to the translated sentence length, and $R$ is the same fraction compared to the reference sentence length, and $\alpha, \beta, \gamma$ are parameters tuned to be close to human judgement.

Cer et al. (2010) also investigate BLEU (Papineni et al. (2002)), which is a measure of N-gram similarity between the translation and the reference sentences. BLEU is defined as the geometric mean of the number of $n$-gram matches between the translation and reference for up to $N$:

$$\mathrm{BLEU}(T,R) = \gamma(T,R)\exp\left(\frac{1}{N}\sum_{n=1}^{N}\log p_n(T,R)\right) \tag{2.18}$$

$$= (\prod_{n}^{N} p_n)^{\frac{1}{N}} \tag{2.19}$$

$$\text{s.t.} \quad \gamma(T,R) = \begin{cases} 1 & c > r \\ \exp(1-\frac{r}{c}) & c \leq r \end{cases} \tag{2.20}$$

$$\text{where.} \quad c = \sum_i |T^i| \tag{2.21}$$

$$\& \quad r = \sum_i \min\{R^i \in \mathcal{S}\} \tag{2.22}$$

$$\& \quad p_n = \frac{\sum_i \vec{c}_n^i}{\sum_i c_n^i} \tag{2.23}$$

where $\mathcal{S}$ is the set of references for the specific sentence; such that $\vec{c}_n^i$ is the number of correct n-grams of length $n$ for translated sentence $i$, while $c_n^i$ is the number of $n$-grams for translated sentence $i$, for a maximum n-gram length $N$. The brevity penalty $\gamma(T, R)$ penalises against translations that are shorter than the shortest of the references, since if it is too short then it is easy to cheat the system with a few correct n-grams. Typically $N$ is chosen as 4, and we use this as our evaluation metric for our translation systems. This is because BLEU is widely used and is therefore easy to compare with other literature in the field.

Both Snow et al. (2008) and Callison-Burch (2009) investigate using human evaluation for natural language processing tasks made available by crowd-sourcing platforms such as Amazon Mechanical Turk. However this often leads to issues such as unreliable evaluators or evaluations of poor quality, which necessitates weighting or filtering poor performing evaluators.

### 2.2.1 Syntax-based translation

Modern syntax-based machine translation systems are primarily build upon the Hiero system described by Chiang (2007). Hiero systems models language as probabilistic synchronous context-free grammars (CFG), such that it is a collections of rules in the form:

$$\chi \to \langle \gamma, \alpha, \sim \rangle \tag{2.24}$$

Where $\chi$ is a non-terminal symbol, while $\gamma, \alpha \in (X \cup \mathcal{V})^+$ are a string of terminals and non-terminals in the source and target languages respectively, where $\mathcal{V}$ is the set of terminals in the target language (the vocabulary), while $\sim$ is the one-to-one alignment of the non-terminals in $\gamma$ and $\alpha$. This means that there is a great amount of freedom allowing for long distance reordering of phrases. As such, rules that do not produce any non-terminals are called *phrased-based rules*, while rules that produce non-terminals are called *hierarchical rules*.

There are also two *glue* rules:

$$S \to \langle \chi, \chi \rangle \tag{2.25}$$

$$S \to \langle S\chi, S\chi \rangle \tag{2.26}$$

Where $S$ is the start symbol. These rules allow for the derivation to start, as well as for the concatenation of hierarchical and phrase-based rules.

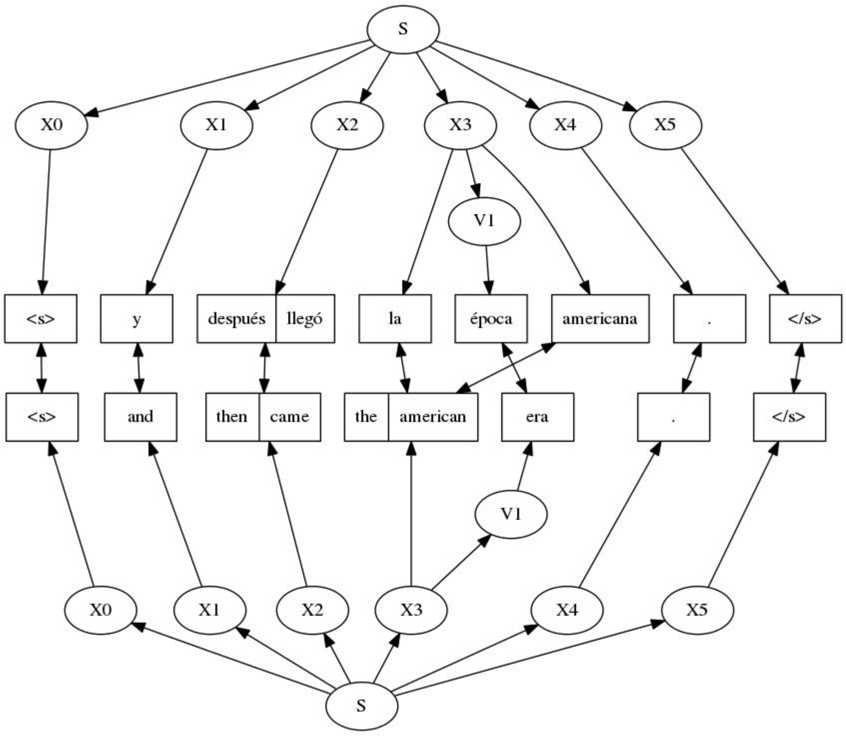


Figure 2.6: Example of SCFG derivations of a pair of sentences in Spanish and English

To generate parallel sentences from such a model, the SCFG rules are continuously applied until there are no more non-terminals. This means for a given input sentence **f** and a translation **e**, we can obtain a set of phrase-based rules which, when applied, would simultaneously give us **f** and **e**, and we name the set of rules used to produce the sentences the derivation $D$. An example of a derivation of a pair of parallel sentences in Spanish and English are shown in figure 2.6. Since one of the assumptions of context-free grammars is that rules are independent of each other, we can calculate the probability of derivation by multiplying the individual probabilities of the rules used:

$$P(D) = \prod_{(\chi \to \langle \gamma, \alpha, \sim \rangle) \in D} P(\chi \to \langle \gamma, \alpha, \sim \rangle) \tag{2.27}$$

For such a system, we have a set of parameters to tune, such as the rule probabilities, source-to-target and target-to-source translation probabilities, word and rule insertion penalties, and more. To do this, we optimise these parameters under a log-linear model using minimum error rate training (MERT, Och (2003)) with respect to the BLEU score.

In this case a log-linear model is where:

$$\hat{\mathbf{e}} = \arg\max_{\mathbf{e}} P(\mathbf{e}|\mathbf{f}) \tag{2.28}$$

$$= \arg\max_{\mathbf{e}} \frac{\exp(\sum_m \lambda_m h_m(\mathbf{f}, \mathbf{e}))}{\sum_{e'} \exp(\sum_m \lambda_m h_m(\mathbf{f}, \mathbf{e}))} \tag{2.29}$$

$$= \arg\max_{\mathbf{e}} \exp(\sum_m \lambda_m h_m(\mathbf{f}, \mathbf{e})) \tag{2.30}$$

$$= \arg\max_{\mathbf{e}} \sum_m \lambda_m h_m(\mathbf{f}, \mathbf{e}) \tag{2.31}$$

Such that $\lambda_1^m$ is the set of feature/parameter weights to be optimised under MERT, and $h_m(\mathbf{f}, \mathbf{e})$ are feature functions, such as language model or translation model scores like those in Devlin et al. (2014).

Nonetheless, instead of taking $argmax_{\mathbf{e}}$, we generate word lattices from the Hiero system. This gives us a large search space scored by the Hiero system, which we can then rescore using other models such as language models, or other neural machine translation models and more (Stahlberg et al. (2016)). Figure 2.7 shows an example of a word lattice for German, where alternate paths can be taken, representing different translations of an input sentence.

### 2.2.2 End-to-end translation

Sutskever et al. (2014) first proposed a neural network encoder-decoder structure, very much in the shape of an auto-encoder, composed of a LSTM encoder that takes in the input words individually, through a vector representation, and accumulates the history to produce a vector representation of the source sentence. A decoder LSTM is then used to decipher the vector representation of the input and output a word sequence in the target language, with the output layer of the decoder LSTMs being one-hot encodings of the target vocabulary. Figure 2.8 shows the structure of the encoder-decoder end-to-end system. To decode using such a system, one can take the one-best output from the decoder, or utilise search algorithms such as beam search to obtain the best path or lattice for the target language.

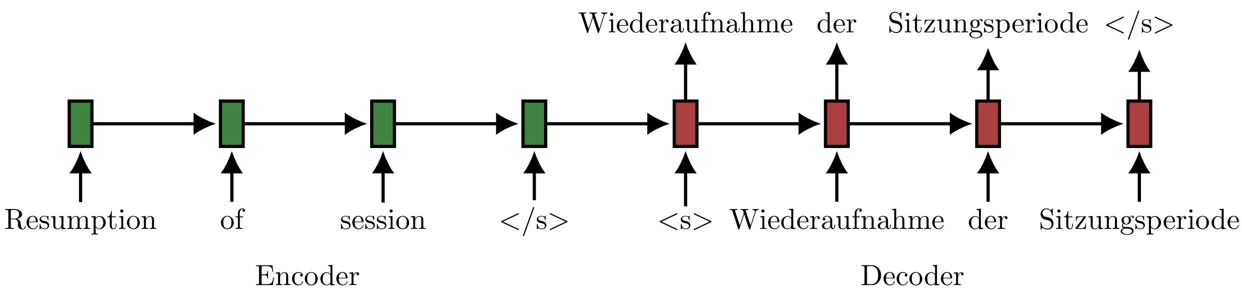


Figure 2.8: The encoder-decoder translation system from Sutskever et al. (2014)

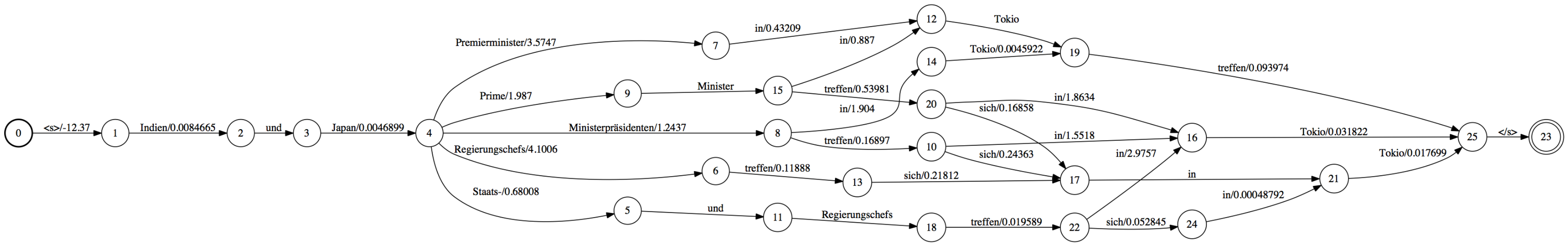


Figure 2.7: Example of a German word lattice

This largely forms the basic structure for sequence-to-sequence neural network design, and could be used for almost any other sequence-to-sequence task such as video description generation (Venugopalan et al. (2014)), text summarisation (Rush et al. (2015)), grapheme-to-phoneme generation (Yao and Zweig (2015)) and more.

Bahdanau et al. (2015) builds on this work and proposes a more complex encoder-decoder structure, where the encoder is in the form of a bidirectional LSTM-based neural network that extracts vector representations for each of the source words in the input, while a feed-forward neural network is used to assign weights from each of the source words for each of the words in the target output, similar to the concept of an attention mechanism. By combining the individual outputs from the bidirectional encoder units using these weights, a vector is obtained and a LSTM decoder is then used to output the target word in a sequence.

Figure 2.9 shows a diagram of the encoder-decoder system with the attention mechanism. As such, the attention mechanism learns to align (by weighting) words in the source and target language. The advantage of using such a structure is in the better performance for sentences of longer lengths, as the vector representation in Sutskever et al. (2014) is a history vector, whose quality would degrade as noise is introduced through the decoding process. By using the attention mechanism, along with bidirectional LSTMs in the encoder, we can train for situations where the word order between the source and target languages are drastically different.

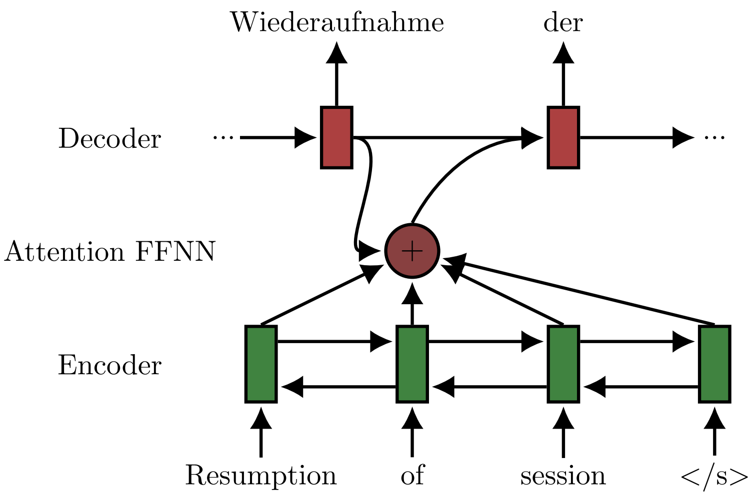


Figure 2.9: The encoder-decoder translation system from Bahdanau et al. (2015)

Similarly, such a system has just as many uses as the one described by Sutskever et al. (2014), but further, Luong et al. (2015) describes a system that is similar to Bahdanau et al. (2015), except it has multiple attention mechanisms and decoder LSTMs, with one of each for a specifically trained task.

Nevertheless, the neural network nature of end-to-end systems means the vocabularies of such systems would be much more limited compared to syntactical translation systems such as Hiero, which use SCFG rules instead of a neural network architecture to model translations.

# Chapter 3

# Related Work

We discuss here a collection of related work on neural network based methods for language modelling as well as neural machine translation. First, we investigate existing literature on sampling methods for neural network training, speeding up training time enough to make large vocabulary tasks feasible; many of these methods are also summarised in Barber and Botev (2016). Afterwards, we investigate the range of methods that have been used for neural machine translation systems to increase their respective vocabulary capacities.

## 3.1 Sampling and approximation methods

One method to combat the rare and unknown word problems was suggested by Jean et al. (2015), where an importance sampling method was used to decorrelate the computation expense with the size of the vocabulary. This was done through a neural network structure proposed by Bahdanau et al. (2015). The method was implemented by dividing the target vocabulary into subsets $V_i \in V$, such that during the training of the output sigmoid layer in the decoder neural network, we only perform gradient updates in the decoder with respect to the correct word and the subset of the vocabulary $V_i$. Nonetheless, the sampling method involves further partitioning the subset such that each partition $V_i' \in V_i$ contains a number of words roughly inversely proportional to their frequency, where an approximating distribution $Q_i(y_t) = \frac{1}{|V_i'|} \quad y_t \in V_i'$ approximates the likelihood $P(y_t|y_{t'<t}, x)$. This nonetheless biases the updates against the true distribution. In English to French translation, their method could reach a better level of performance compared to existing limited vocabulary systems with BLEU scores from 30.0 to 32.7. However, for English to German translation the performance with and without the rare-words through importance sampling remained more-or-less similar with a BLEU score of around 17.

Similarly, Ji et al. (2015) proposes BlackOut, which is an approximation method applied to RNN language models. In BlackOut, the log-likelihood objective function is approximated as shown in equation 3.2 and back-propagation is performed accordingly.

$$J = \log(P_\theta(w_i|s)) \tag{3.1}$$
$$\approx \log(\tilde{P}(w_i|s)) + \sum_{j \in S_K} (\log(1 - \tilde{P}(w_i|s))) \tag{3.2}$$

Such that $S_K$ is a small subset of the vocabulary $\mathcal{V}$, where $w_i$ and $s$ are the $i$th row of the final weight matrix to the output layer and the output from the penultimate layer, respectively. Further:

$$\tilde{P}(w_i|s) = \frac{q_i \exp(\langle W_i, s \rangle)}{q_i \exp(\langle W_i, s \rangle) + \sum_{j \in S_K} \exp(\langle W_j, s \rangle)} \tag{3.3}$$

Where the importance weights $q_j := \frac{1}{Q(w_j)}$ according to the proposal distribution $Q(w)$. Ji et al. (2015) proposes for this distribution to be $Q_\alpha(w) \propto p^\alpha_{uni}(w)$, such that $\alpha$ is a tunable parameter that shape $Q(w)$ to be the uniform distribution when $\alpha = 0$ and the unigram probabilities when $\alpha = 1$. They showed comparative perplexity results to larger neural networks with a much longer computation time.

The method of using Noise Contrastive Estimation (NCE) for language modelling was described in Mnih and Teh (2012); Chen et al. (2015) as a more stable alternative to importance sampling methods. NCE assumes that data samples, for a given history context vector $h$, are generated from a mixture of the data distribution $P_d(w)$ and a $k$-times more frequent noise distribution $P_n(w)$ (which is taken as the unigram distribution), such that:

$$P(w|h) = \frac{1}{k+1} P_d(w|h) + \frac{k}{k+1} P_n(w|h) \tag{3.4}$$
$$\text{s.t. } P(D = 1|w, h) = \frac{P_d(w|h)}{P_d(w|h) + kP_n(w|h)} \tag{3.5}$$

Where $P(D = 1|w, h)$ is the posterior probability that the data was drawn from the data distribution. As such, an NCE objective function can be defined as:

$$J(\theta) = -\frac{1}{N_w} \sum_{i=1}^{N_w} \big( \log P(D = 1|w_i, h_i) + \sum_{j=1}^{k} \log P(D = 1|\tilde{w}_{i,j}, h_i) \big) \tag{3.6}$$

Where $\theta$ represents parameters in the neural network, $N_w = \mathcal{V}$ and $\tilde{w}_{i,j}$ represents the $j$ noise sample drawn for $w_i$. The advantage of NCE is that the normalisation constant $Z$ for $P_d(w|h) = \frac{\exp(\langle \mathbf{W_1}, \mathbf{h_{l-1}} \rangle)}{Z}$ does not have to be calculated, and can be freely adjusted to suit training speed or performance.

Devlin et al. (2014) points out that in a neural network translation system, we can train the output layer of the neural network to automatically approximate softmax function.

This would be done by augmenting the objective function for training to include a term for the normalisation constant in the softmax function, using a Lagrange multiplier, such that we enforce:

$$\log(Z(x)) = 0 \tag{3.7}$$

$$Z(x) = \sum_{i \in |\mathcal{V}|} \exp(U_i(x)) \tag{3.8}$$

Where $U_i(x)$ are the logits of word $i$, (or $\langle \mathbf{w}_{l,i}, \mathbf{h}_{l-1} \rangle$ in equation 1.1). This means that the logits $U_i(x)$ become approximations of the log-posterior. They use this method for an end-to-end feed-forward neural network translation system, and this approximation increases the lookup time by a factor of $\sim$15 during decoding.

## 3.2 Word encoding

Luong et al. (2014) attempted to address the problem by marking the location of these OOV words and its alignment to the source language. They then translate the words from the source language using a more general purpose translation dictionary, or simply the original source words if no translation is found. They build a translation system as per Sutskever et al. (2014). The alignment of unknown words were obtained through explicitly stating alignments in the training data, with the decoder outputting a pointer to the word in the input. In order to train for the correct alignments, and account for the fact that the source vocabulary may be larger than the target vocabulary, the most notable method for doing so was to have each unknown word in the target language marked by an alignment to the source language (though only within $\pm$7 words). They are then translated from the source word through a dictionary, with back-off to an identity translation. This method improves the BLEU score from 31.5 to 33.1 for English to French translations, similar to that of Jean et al. (2015), with larger gains in BLEU scores for smaller vocabularies and training data, from 29.5 to 32.7.

Similarly, Gulcehre et al. (2016) tries to resolve this problem by focusing on a feature from Luong et al. (2014), training a specialised NMT system that decides when to point to words in the source sentence, which may then be translated using a dictionary, or are loanwords and proper nouns that do not need translation. With Bahdanau et al. (2015) as a baseline, the pointer mechanism is added to the decoder, which then decides to translate or point to a word from the source language. This approach brings a BLEU increase from 20.2 to 23.8 for English to French translations, which is similar to the other English to French results obtained by others, except for a significantly lower baseline.

An interesting method was devised by Chitnis and DeNero (2015), where they employed the equivalent of Huffman coding to the rare words, thereby decomposing and replacing them with a sequence of commonly shared symbols (or "words"), and used this configuration to train the neural translation system. The system used also follows Bahdanau et al. (2015), where we once again focus on the decoder neural network. The Huffman code is a uniquely decodable coding method that efficiently expresses a large vocabulary $\mathcal{V}$ by representing words as codes using a smaller alphabet $\mathcal{V}'$, such that $|\mathcal{V}'| \leq |\mathcal{V}|$. In normal applications of the Huffman code, the length of the codewords are inversely proportional to the log-probability/log-frequency of the word. However, in the case of Chitnis and DeNero (2015), the frequency of the rare words are similar, so codewords are assigned at random. By applying this technique to rare words in the target language, we can significantly reduce the number of rare words. This method necessarily elongates the length of the target strings, but nonetheless improves on English to French from 25.8 to 27.5, the same kind of improvement seen in Jean et al. (2015); Luong et al. (2014), though with a lower baseline. Test set precision results show that, using the best method of encoding, only 28.0% of the rare words in the target vocabulary were accurate, compared to 65.8% for common words.

A question arises from Chitnis and DeNero (2015), which is whether the individual symbols in the compact vocabulary can, in themselves, hold some semantic or syntactical utility in relation to the real words it represents. This leads to the question of whether we can break down rare words to smaller units. For example, de Gispert et al. (2009) utilised morphological decomposition systems to combine with word based translation systems. This proved to be useful, providing an BLEU increase from 27.9 to 28.9, for Finnish, a agglutinative and morphologically rich language where it is possible to form infinitive length compound nouns. While the aim of de Gispert et al. (2009) was not to compact the vocabulary, perhaps the rare word or unknown word problem could be tackled through the morphological decomposition of rare words in place of the more arbitrary units designated by Chitnis and DeNero (2015).

In this vein, Sennrich et al. (2015) proposes the use of Byte Pair Encoding (BPE) to automatically discover subword units. This is done by building up subword units starting from alphabetic letters alone and merging the most common sequence pairs of these units until the limit on the number of merge operations is reached. This is done through the same structure proposed by Bahdanau et al. (2015). The vocabularies for both source and target languages are encoded using byte pair tokens, with some of the tokens being words themselves, the alignment of the subword tokens across the languages is done through crude automatics. Two methods of encoding were investigated - obtain separate byte pairs independently in the source and target language, or jointly obtain pairs from both languages at the same time. While independently learning the byte pairs increases the likelihood that the composed word will be in-vocabulary, the jointly learnt vocabulary generally performs similar, if not better, as it has an overall smaller vocabulary. For example, with 60,000 tokens for the source and target languages separately (120,000 in total) for English to German translation, the BPE system performs similar, if not worse, than a jointly trained BPE vocabulary with 90,000 tokens shared between both languages, with BLEU scores of 22.8 and 21.5 for the joint and separate vocabularies, respectively, versus 20.6 for the baseline. Sennrich et al. (2015) had shown that for English to German translation they could obtain a respectable rare word $F_1$ accuracies of 41.8% for English to German translations, compared to 58.5% for common words, though this was not replicated for English to Russian translations (29.7% and 55.8% for rare and common words, respectively). Presumably this is partially because there is not as much of a benefit to using the joint vocabulary, as English and Russian have distinctively different scripts and unicode assignments for its characters.

At the same time, Luong and Manning (2016) suggest viewing translation as an even lower level problem. They work around the rare and unknown word problem by creating a "hybrid" NMT system using a standard backbone and a word encoder and a separate decoder. The backbone NMT system is built similar to Bahdanau et al. (2015), in a deep LSTM encoder-decoder setup with an attention mechanism to learn an alignment vector and the normal convention of assigning rare words to be out of vocabulary. The word-level encoder (which is separate from the encoder in the backbone system) is a deep LSTM recurrent neural network for feeding in OOV source words from a character level to output a word-level state for the backbone system. Meanwhile, the target language decoder is a further deep LSTM recurrent neural network that is trained to output the target language characters from the word-level state provided by the backbone NMT decoder. In English to Czech translation, they were able to increase the BLEU score over a standard NMT system from 12.6 to 17.5, though this is only marginally better than the gains obtained using techniques described in Luong et al. (2014). Further, for the rare words, they could show that not only does the word embeddings from the hybrid system outperform embeddings from word-based translation systems, in terms of the Spearman's correlation between similar words, the embeddings also outperform embeddings trained through RNNs and morphological analysers Luong et al. (2013).

Nonetheless, this leads to the logical conclusion of character level machine translation. Chung et al. (2016) approaches the problem from a neural network design perspective; they devise a "bi-scale recurrent neural network" that uses two Gated Recurrent Units that capture dependencies at a fast and a slow timescale. This would allow the recurrent unit to take into account the local (intra-word level) as well as the more global (inter-word level) contexts at the same time. Nonetheless, they do this through encoding the source language using Byte Pair Encoding, as per Sennrich et al. (2015), and use the bi-scale neural network in place of the decoder from Bahdanau et al. (2015). Beam-search of width 15 is then used for the character level decoding. Overall, this improves upon a baseline built using the BPE method from Sennrich et al. (2015) by $\sim 2$ BLEU scores, and they show that the concept of the bi-scale unit is robust as the improvement in BLEU scores remain steady regardless of the overall sentence length, while the typical decoder unit falters at higher sentence lengths.

Meanwhile Costa-Jussà and Fonollosa (2016) devise convolutional filters to extract affix units in order to ensure some morphological features are captured from the source language. They also use the model from Bahdanau et al. (2015) as a baseline and base their character-level neural network architecture on Kim et al. (2015). The setup is such that the affixes of a word are filtered through a convolutional filter, while other parts of the word are fed into a highway network, which are neural networks with hidden layers that partially pass on their inputs without transformation through the activation function. While this does not directly improve the rare word problem or the unknown word problem for the target language, this character based method nonetheless improves English to German translations by 3 BLEU points from 16.5 to 19.5, a slightly larger gain than Sennrich et al. (2015), but with lower baselines. However it remains to be seen, as different language pairings were investigated, whether this is an improvement compared to de Gispert et al. (2009), which has more sophisticated morphological modelling but is also different as a phrase-based system.

Finally, table 3.1 is a summary of the improvements in BLEU points made by these encodings discussed with respect to their language pairings. Generally, these methods show similar increases in the BLEU score compared to a truncated vocabulary (Unk), when the discrepancies between the baselines are taken into account, since it is easier to improve on poor performing systems.

| **English to French** | Unk Baseline | Improved Score | Delta |
|---|---|---|---|
| Jean et al. (2015) | 30.0 | 32.7 | +2.7 |
| Luong et al. (2014) | 31.5 | 33.1 | +1.6 |
| Chitnis and DeNero (2015) | 25.8 | 27.5 | +1.6 |
| Gulcehre et al. (2016) | 20.2 | 23.8 | +3.6 |
| **English to German** | Unk Baseline | Improved Score | Delta |
| Jean et al. (2015) | 16.5 | 17.0 | +0.5 |
| Sennrich et al. (2015) | 20.6 | 22.8 | +2.2 |
| Chung et al. (2016) | 21.7* | 23.5 | +1.8 |
| Costa-Jussà and Fonollosa (2016) | 16.5 | 19.5 | +1.6 |
| **English to Czech** | Unk Baseline | Improved Score | Delta |
| Chung et al. (2016) | 14.6* | 17.0 | +1.4 |
| Luong and Manning (2016) | 12.6 | 17.5 | +4.9 |
| **English to Russian** | Unk Baseline | Improved Score | Delta |
| Sennrich et al. (2015) | 18.8 | 20.4 | +1.6 |
| Chung et al. (2016) | 19.7* | 21.1 | +1.4 |
| **English to Finnish** | Unk Baseline | Improved Score | Delta |
| Chung et al. (2016) | 9.0* | 10.9 | +1.9 |

Table 3.1: BLEU scores for SMT papers discussed and their respective improvements, with Unk baselines being systems with truncated vocabularies unless specified otherwise. *Baseline from Sennrich et al. (2015).

# Chapter 4

# Subword encodings

In this chapter we describe a number of approaches we have taken to decompose words or compress the vocabulary $\mathcal{V}$. The aim is to investigate methods for finding an encoding $\mathcal{M} : \mathcal{V} \to \mathcal{V}'$, where the size of the encoding vocabulary $|\mathcal{V}'|$ is small enough to be feasibly trained with a neural network, at the expense of more tokens in sentences.

## 4.1 Truncated vocabulary

In standard neural network based language modelling, we sort the vocabulary of the training data by their frequencies, and preserve only the 30,000 most frequent words. This means all other words in the vocabulary would be assigned to the unknown symbol `<UNK>`. While limiting, this nonetheless allows us to rescore all words in the word lattices, as we have LM probabilities for unknown word tokens.

## 4.2 Huffman encoding

Huffman coding (Huffman et al. (1952)) is originally a close-to-entropy, uniquely decodable compression coding scheme for compressing fixed length codes in vocabulary $\mathcal{V}$ into variable length codes in vocabulary $\mathcal{V}'$. It works on the basis of allocating shorter codes to the most probable binary sequences or characters, while longer codes are allocated to the least probable characters. This also makes the assumption that each unit in the sequence is independent of each other. This is done by recursively joining the lowest frequency alphabets together and summing their frequencies until no more nodes can be joined, then traceback down the graph structure, giving each branch an alphabet in the vocabulary, generating the Huffman code.

For example, for a 2-bit binary alphabet $\{00, 01, 10, 11\} \in \mathcal{V}$ to be mapped onto vocabulary $\{0, 1\} \in \mathcal{V}'$, such that:

$$P(00) = 1/8 \tag{4.1}$$
$$P(01) = 1/8 \tag{4.2}$$
$$P(10) = 1/4 \tag{4.3}$$
$$P(11) = 1/2 \tag{4.4}$$

We obtain the mapping shown in figure 4.1 using the Huffman encoding algorithm.

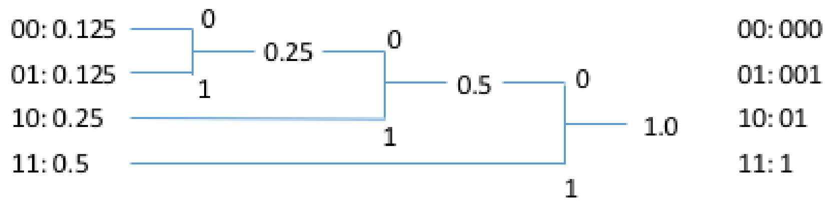


Figure 4.1: Example of the Huffman coding scheme

Nonetheless, this method can be trivially extended for larger, non-binary original vocabularies as well as non-binary Huffman codes. Figure 4.2 shows an example of Huffman encoding applied to vocabularies, as described by Chitnis and DeNero (2015). As such, words can be broken down into subword units in the form of `s_0`, etc.

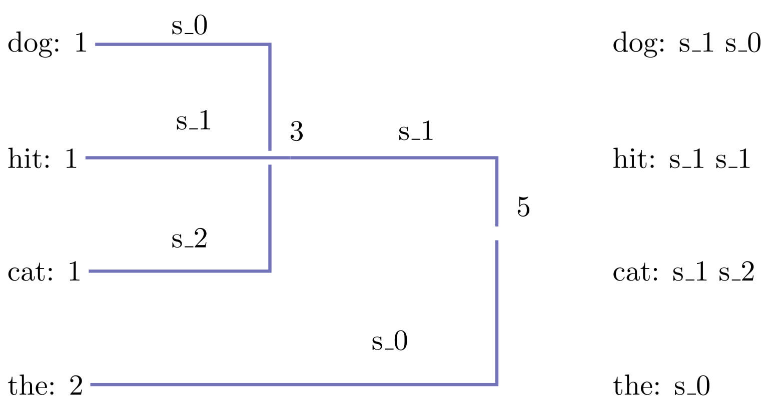


Figure 4.2: Example of the Huffman coding scheme

In our experiment, we follow the "Repeat Symbol" approach described in Chitnis and DeNero (2015), where we preserve the most frequent $K = 30,000$ words, and use a subword vocabulary of $|\mathcal{V}'| = 500$ to represent the rare words in the original vocabulary. As such, we do not have probabilities assigned to unknown words since we have representations for all words in our training corpus.

Nonetheless, one can see that the disadvantage of such a coding scheme is the deletion of information presiding in the character sequences of individual words. It is possible that for a language such as German, where many rare words could be compound nouns, we essentially eliminate the information its constituent words may contain and treat the word as an individual word, which in turn is transformed into a sequence of subword units. This would perhaps unnecessarily complicate the language modelling problem for our neural network.

Further, Huffman coding assumes that the training set contains the set of all possible words in the target language, which is not the case. Since it has no representation of unknown words, this poses a problem for decoding and scoring new words not seen in training.

## 4.3 Byte-pair encoding

In contrast, byte-pair encoding (BPE) described in Gage (1994) is a compression method that attempts to replace the most frequent sequences of symbols with new but shorter symbols. This technique is applied in Sennrich et al. (2015) for end-to-end translation systems to construct subword units by looking at the most frequent character pairs in the data. In this way we discover subword units by building up pairs from the character level, where we also define the end of a word as a separate character.

The scheme finds the most frequent consecutive pairs of symbols and replaces them with a single symbol, updating the list of pairs to include the new pairs made possible through merging, and repeats the process until a pre-defined limit is reach or if no new pairs have frequencies greater than 1. This could prove to be more useful than Huffman encoding as it explicitly takes advantage of character sequences within words.

One method to approach this is by finding the most frequent pair and updating words in the vocabulary containing the pair, then calculating the new pairs generated from the merged pair. This has the benefit of obtaining a complete mapping of the vocabulary to its subword units at the end of the process. Pseudo-code for BPE is shown in the Appendix A.2.

In this way, this becomes a computationally intensive task since every merging operation must be applied iteratively to every word in the vocabulary, and since the number of rules correlate with the "size" of the new vocabulary, the computation complexity scales with $O(|\mathcal{V}|N)$, where $N$ is the number of merging operations performed.

As an example, if only the words *low*, *lower*, *lowest* and *slower* were in the corpus, occuring once, then the merge operations are the sequence below in order of the most frequent pairs, also showing the state of the vocabulary:

```
(l o w </w>) (l o w e r </w>) (s l o w e r </w>)  (l o --> lo)
(lo w </w>) (lo w e r </w>) (s lo w e r </w>)     (lo w --> low)
```

```
(low </w>) (low e r </w>) (s low e r </w>)        (e r --> er)
(low </w>) (low er </w>) (s low er </w>)          (er </w> --> er</w>)
(low </w>) (low er</w>) (s low er</w>)            (low er</w> --> lower</w>)
(low </w>) (lower</w>) (s lower</w>)
```

By keeping track of the changes to the words in the vocabulary, we can also obtain the mappings for each of the words to their respective subword units:

```
low --> [low, </w>]
lower --> [lower</w>]
slower --> [s, lower</w>]
```

The number of merge rules does not necessarily correspond to the vocabulary; in the example shown, we can see that in our corpus of three words, we obtain 4 distinct subword units in our new vocabulary from 5 merge operations. Nevertheless, it is a limit on the size of the new vocabulary, such that $|\mathcal{V}'| \leq N + |\mathcal{C}|$, where $\mathcal{C}$ is the set of basic characters from which byte pairs are found (i.e. the alphabet plus punctuation).

For any new word in the input source language, for example *locker*, we can break down each of the words and apply each of the rules in succession to encode its characters into pairs, until no rules can be applied further:

```
l o c k e r </w>  (l o --> lo)
lo c k e r </w>   (e r --> er)
lo c k er </w>    (er </w> --> er</w>)
lo c k er</w>
```

For our experiments, we decompose the entire vocabulary $\mathcal{V}$ using BPE, using 30,000 merge operations. As with Huffman encoding, this means we would not have unknown word tokens to train for from the training corpus, as every latin character sequence (such as words) in the vocabulary would be able to be represented by a subword sequence.

Nonetheless, the above example also points out some potential flaws of BPE, namely that even though the alphabets make up the basic subword units, which guarantees that every valid word can be broken down, in practice most combinations of them will be merged and there will be very few singletons in the training data. For a given new word such as *locker*, it is possible that it will contain characters or sequences of characters that the training data does not have examples of, such as er</w>.

In decoding, we obtain outputs in the form of these subword units, and we recognise the end of a word by the end of word </w> symbol. Nevertheless, this allows us to have a theoretically unlimited vocabulary, as the output could generate any sequence of characters before ending with the end of word symbol. However, in practice we imagine that most of the words would be constrained to versions previously seen words.

## 4.4 Character-level language modelling

Character level decomposition is a special case of BPE is when no merge operations are used. The only subword units we have would be the characters in the Latin alphabet, the word end symbol `</w>` and punctuations contained in the training corpus. Similar methods have been described in Kim et al. (2015). However, we differ from them by forgoing the convolutional layer and the highway network set up in order to maintain a fair comparison of the various encoding schemes, as opposed to neural network design. Another difference is that we also predict on a character level in order to compress the size of the output layer.

## 4.5 Experiments

We perform decompositions and encodings for each of the methods described in this section over a dataset consisting of the *Europarl 7*, *Common Crawl* and *News Commentary 2007* data provided for the WMT15 competition. The resulting decompositions are compared to the *news-test15* test data. The data is normalised to lowercase ASCII characters, with ASCII punctuations except hyphens separated and preserved, while Arabic numerals are normalised to a number token (represented by *0*).

We limit the vocabulary size of the truncated vocabulary model (Unk) to 30,000; for Huffman coding, we keep the most common 30,000 words as is, and encode the rest of the vocabulary using a set of 500 pseudo-words; for byte-pair encoding, we perform 30,000 merge operations, resulting in 30,021 subword units. Additionally, we train a BPE-10k model which is the BPE decomposition with only 10,000 merge operations.

Figure 4.3 shows the variation in the character-lengths of the subword units generated using the BPE algorithm, with a large proportion of the subword units between 3 and 12 character lengths.

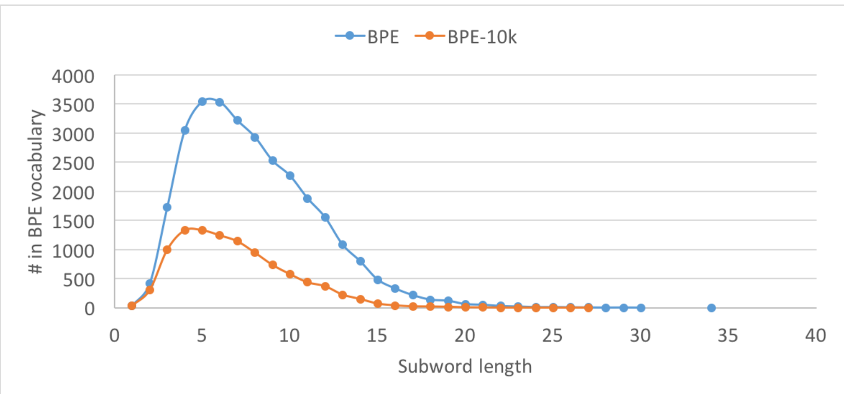


Figure 4.3: The distribution of BPE subwords with respect to their length for the 30,000 merge operations and 10,000 operations.

Table 4.1 shows the increases in the number of word tokens for each of the encoding methods. This shows that for encodings such as BPE or Huffman coding, we gain much better coverage over the test set vocabulary without significantly increasing the number of tokens.

| Encoding | Vocab size | Training set tokens (million) | Test vocab coverage | New word |
|---|---|---|---|---|
| Training | 1,440,888 | 115 | 90.7% | - |
| Unk | 30,001 | 115 | 61.7% | Unk |
| Char | 54 | 726 | 100% | Decompose |
| BPE-10k | 10,029 | 146 | 100% | Decompose |
| BPE | 30,021 | 130 | 100% | Decompose |
| Huffman | 30,500 | 127 | 90.7% | None |

Table 4.1: Speed differences in lattice rescoring methods.

Of these methods, only the truncated vocabulary setup is able to represent unknown words, though this is at the sacrifice of much lower test set vocabulary coverage compared to other methods. Meanwhile, both character decompositions and byte-pair encodings can significantly increase the test set word coverage, BPE uses far fewer subword units to represent the same words.

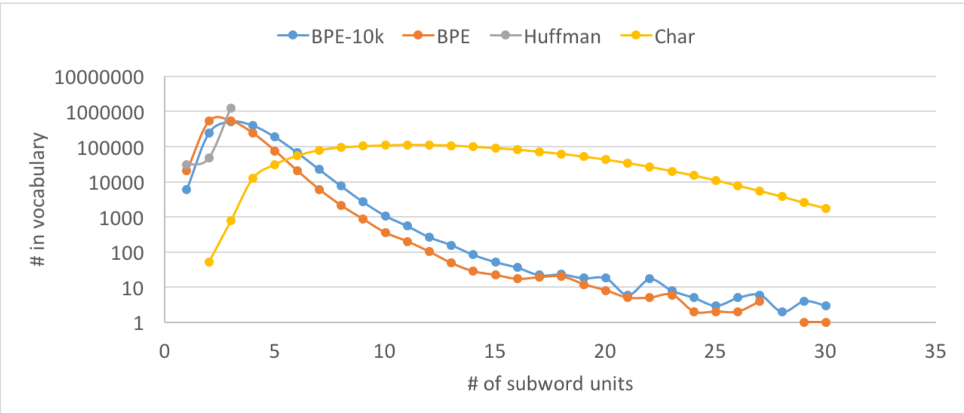


Figure 4.4: The distribution of the subword units by length, with respect to their length

Figure 4.4 shows the number of subword units used to represent words in the original vocabulary $\mathcal{V}$. We can see that for BPE, most words are split into 2 to 5 subword units, while for Huffman encoding it is 3. For character encoding, most words are split into its constituent characters and the distribution of the number of units or tokens to represent words is spread more evenly from 4 to 25. However if we weight this by the frequencies of the words in the training corpus (fig. 4.5), we can see that for BPE and Huffman coding (where the most common words are not decomposed), the majority of the words in the corpus are not decomposed at all, while for character encoding most of the words are thinly distributed across a range of lengths, many between 3 and 11.

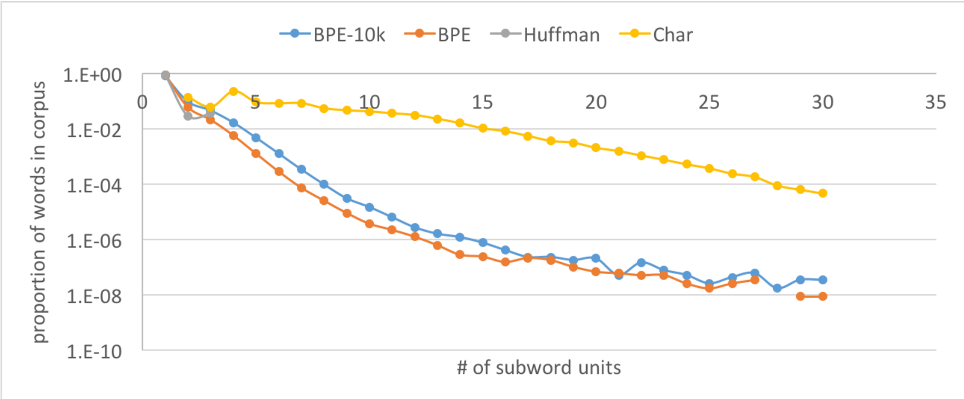


Figure 4.5: The distribution of the subword units by length, with respect to their length, in the training corpus

From these results, we can see that byte-pair encoding is the more efficient method of decomposition. It has the ability to decompose previously unknown words, which is something that Huffman coding lacks, and at the same time it is far more efficient at using a minimum number of subword units, compared to character decomposition. We can also see that there is a trade-off between reducing the number of merge operations, which decrease the size of the subword vocabulary, and the representation efficiency, i.e. the number of subwords each word decomposes to.

# Chapter 5

# SMT with word encodings

In this section we describe the set up of the neural network language model used as well as the tools used to create our experiments. We investigate the methods discussed in Section 4 and compare their respective performance to that of the standard truncated vocabulary neural network language model.

We do not compare language perplexities between different encoding schemes, as the perplexity score for different vocabulary set ups are not mathematically equivalent, and therefore they cannot be fairly compared without non-trivial adjustments. While existing literature quotes both single and ensemble system performances for neural network models, all results reported here would be from single systems due to limitations in resources.

## 5.1 Implementation

In this project we investigate the application of various decomposition and word encoding techniques described in the previous chapter on language modelling. These language models are then applied to syntactical translation systems described in chapter 2.2.1. We use the phrase-based hiero system described in Stahlberg et al. (2016) and its FST-based lattices as our translation system, with the 1-best paths through the lattices as the baseline. The lattice scores already contain a combination of grammar scores and scores from a Kneser-Ney language model.

We use *news-test14* as our development dataset, while testing and report results on the *news-test15* dataset. Both of these datasets are news commentary texts and belong to the same domain. For training, we use *Europarl 7*, *Common Crawl* and *News Commentary 2007* as our training set for English to German translation. For the lattices, the data was normalised to TrueCase and also contains umlauts as well as punctuation characters, while keeping Arabic numerals.

Meanwhile, for the language modelling, in the German target language we train the neural network on normalised lowercase ASCII characters, with all punctuation excluding hyphens, while Arabic numerals are normalised to a number token (represented by *0*). While this harsher normalisation means our decompositions could be irreversible, we can nonetheless maintain the casing in the original lattice and simply score lattices or N-best lists non-destructively.

The language model is trained with over 110 million German words, totaling approximately 800MB. Compared to the test data, the selection of the three training datasets means we are ensured to contain some data that are in domain. We train the language model on a random 95% of the entire training dataset of *Europarl 7*, *Common Crawl* and *News Commentary 2007*, while we hold out the other 5% of the data as the validation set. Table 5.1 summarises the datasets used.

| | Words | Sentences |
|---|---|---|
| Training set | 115M | 4.7M |
| Dev set | 59.3K | 2.7K |
| Test set | 44.8K | 2.2K |

Table 5.1: Corpus sizes of datasets used

In cases where there are unknown words in the source text in the test set, the Hiero system copies the unknown words from the source sentence into the target sentence, akin to Luong et al. (2014). This means for certain encodings such as Huffman coding, there would be certain unknown words in the target language word lattice for which there are no decompositions. In these cases, we assume that the word was copied from the source sentence, and would therefore be present in all hypotheses, and assign a log-perplexity of 0 for models such as Huffman coding that do not handle unknown words.

To implement neural network language models, we use the TensorFlow API library to enable GPU-accelerated computing (Abadi et al. (2016)). The use of TensorFlow has the benefit of creating programs that are easily scalable to a wide range of computation capabilities. Further, TensorFlow provides existing LSTM as well as other RNN objects and optimisers, with optimised matrix and other mathematical operations using CUDA. This means little work needs to be done to create complicated structures like LSTMs from first principles, and at the same time ensures fast neural network training and decoding through GPU optimisations via CUDA.

We base our neural network model on Zaremba et al. (2014); LSTMs are trained with dropout to ensure regularisation in the neural network. We deactivate a random proportion of the hidden layer nodes throughout each RNN training sequence with probability 0.5.

The parameters of the LSTM LMs are configured close to what is described in Jozefowicz et al. (2016); we set the learning rate at 0.2 in order to prevent the training procedure from converging too early to a local optimum, as there is a relatively large amount of data used; we set a forget gate bias of 1.0; we also unfold the LSTM for 20 steps during error back-propagation training to ensure that longer sentence contexts are learnt and re-normalise the $L_2$ norms of the gradients at 1.0 in order to prevent exploding gradients. We train the language model stochastically using AdaGrad (Duchi et al. (2011)).

We use 3 LSTM hidden layers of 1024 units each and train them in batches of 256. While Jozefowicz et al. (2016) shows that two LSTM layers are enough to saturate the performance of neural network LMs, 3 layers were used as we are using relatively complex encodings, with the language model learning on the subword level, and therefore an additional layer is used to encourage and ensure that the neural network could exploit the additional layer of information.

For all of our experiments, we limit our vocabulary to be approximately 30,000 as described in Section 4.5. This is in comparison to 1.48 million words in the vocabulary of the phrase based system. Typically, the neural network language models take 2 hours to train per epoch on a single NVidia GTX Titan X (2015) GPU, achieving a speed of approximately 8,900 words per second.

## 5.2 N-best rescoring

One way to use a language model is to rescore N-best hypotheses produced from the word lattice. There are two methods for doing this:

- We use FST tools to output the N-best hypotheses list, where each of the words are then mapped to the encoded vocabulary using a dictionary mapping. These hypotheses are then rescored using the LSTM language models.
- Create a vocabulary transducer $\mathcal{M} : \mathcal{V} \to \mathcal{V}'$ that maps the vocabulary $\mathcal{V}$ from the word lattice to the encoding vocabulary $\mathcal{V}'$. We can then extract the N-best list from such a lattice.

It is checked that the N-best hypotheses produced by both of these methods are the same. These mapped hypotheses are then rescored by the neural language models for their log-perplexity values for the test set.

We implement a batch rescoring procedure, such that we group sentences for the N-best list for each sentence into batches of size $K$. In many cases the lengths of the sentences in a batch are different, in which case sentences are filled with symbols so that they are the length of the longest sentence within the batch. We then record the perplexity costs of the sentences when the sentence end symbol is observed. In cases where the list size

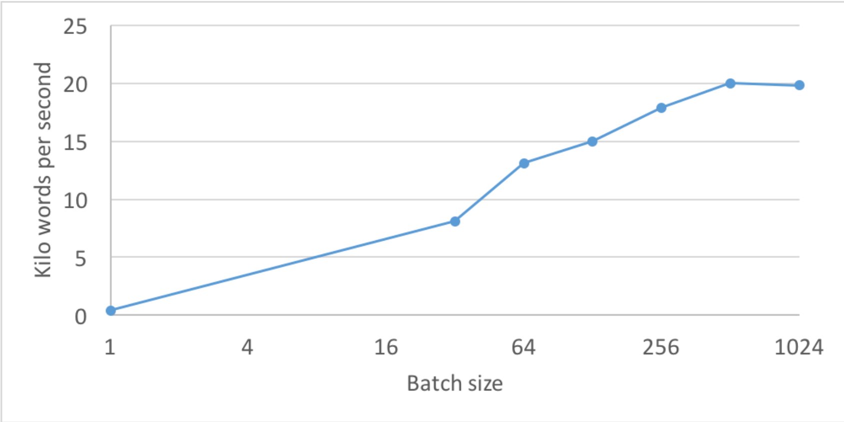


Figure 5.1: The effect of batch size in language model N-best list rescoring using TensorFlow.

$N$ does not divide by $K$, we add $K - N\text{mod}(K)$ sentences to the last batch, such that the shape of the matrices used are not changed. Nonetheless, this could be modified to be faster by joining different sentences together and thereby removing the need to fill the input matrix with mock sentences.

Figure 5.1 shows the speed-up by using such a procedure during decoding. Here we can see a clear logarithmic speed-up to up to 50× when using larger batch sizes using a Nvidia Titan X (2015) GPU. This shows the ability in TensorFlow to calculate large matrix operations faster than performing the same task sequentially, and shows how GPU-accelerated, parallel operation is more preferable than sequential operations in a neural network setting. We can also see that the decoding speed remains more or less constant after a batch size of 512, which may be the point where the programming becomes bottleneck, given that the GPU is not at 100% utilisation.

To obtain a final score, we interpolate the existing lattice scores and the language model log-perplexity scores:

$$S_{\text{Overall}} = \lambda_{\text{hiero}} \times S_{\text{hiero}} + \lambda_{\text{NLM}} \times S_{\text{NLM}} \tag{5.1}$$

$$\text{s.t.} \sum_i \lambda_i = 1 \tag{5.2}$$

Here we use a grid search with coarsity 0.01 to find the optimal $\lambda$ for the *news-test14* development set and this value is then used for the test set to obtain the final BLEU score.

## SMT Results

Table 5.2 shows the BLEU scores for N-best rescoring in using neural language models with various encoding vocabularies. The 1-best result corresponds to the baseline lattice-only performance from the *hiero* system. We can see here that in more constrained search spaces with 100-best list, the Huffman encoding performs marginally better than that with simple unknown word tokens. Nonetheless, this small advantage diminishes when a larger 1000-best search space is used.

| N-best | Vocabulary Encoding | LM Interpolation Ratio | BLEU | Brevity Penalty |
|---|---|---|---|---|
| 1 | - | - | 20.63 | |
| 100 | Unk | 0.24 | 21.95 | 1.0000 |
| | Char | 0.24 | 21.71 | 1.0000 |
| | BPE | 0.27 | **22.14** | 1.0000 |
| | Huffman | 0.22 | 21.84 | 1.0000 |
| 1000 | Unk | 0.24 | 22.00 | 1.0000 |
| | Char | 0.20 | 21.79 | 1.0000 |
| | BPE | 0.21 | **22.11** | 1.0000 |
| | Huffman | 0.22 | 21.89 | 1.0000 |

Table 5.2: N-best rescoring using various RNN LMs on the *news-test15*.

Nonetheless, BPE decoding performs marginally better than the rest of the methods. Compared to the character based decomposition and Huffman coding, it is marginally better, however this improvement is smaller when compared to the standard truncated vocabulary language model (Unk). This may be because, given that the BLEU score is evaluated on the word level, the Unk model is better at modelling the 30,000 words within its vocabulary, even if all other words are modeled as unknown words. In contrast, both Huffman coding and the character level models require a further level of understanding for the language model, and therefore perform slightly worse.

Meanwhile, figure 5.2 shows the evolution of the BLEU score for N-best rescoring as the number of merge operations vary, where the character-level language modelling represents BPE with no merge operations. We can see that BPE with only 10,000 merge operations produce results marginally worse than that of BPE with a larger number of merge operations, and that the difference between the two in a larger search space (1000-best) is much smaller than a smaller space such as 100-best.

The use of BPE shows a small increase in performance over the Unk model for $0.1 - 0.2$ BLEU points. This could mean two things - that the larger vocabulary encapsulated by encoding in BPE has led to a marginal improvement, or that the use of the subword units meant the language model was representing the German language in a more useful format.

In order to test these hypotheses, we ran a further experiment, in which we limited our vocabulary to the 30,000 most frequent words, and decompose the rest using the BPE

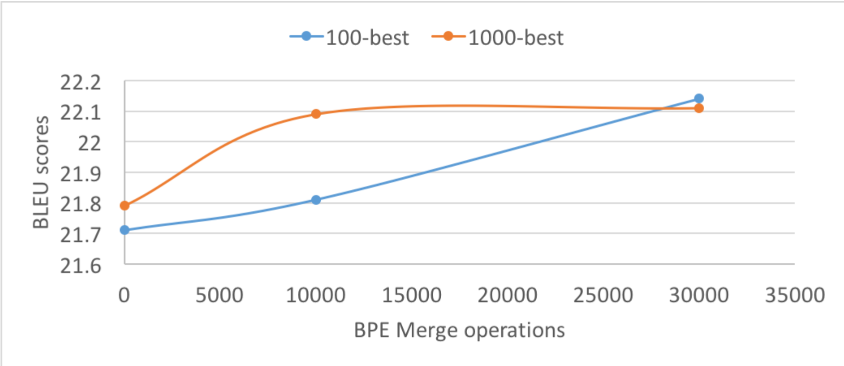


Figure 5.2: The effect of the number of merge operations on the N-best BLEU rescore.

rules obtained before. With this new model, which we shall call *BPE_Unk*, we reproduced similar experiments as before and the results are shown in table 5.3.

| N | Vocabulary Encoding | LM Interpolation Ratio | BLEU | Brevity Penalty |
|---|---|---|---|---|
| 100 | Unk | 0.24 | 21.95 | 1.0000 |
| | BPE_Unk | 0.24 | 21.94 | 1.0000 |
| | BPE | 0.27 | *22.14* | 1.0000 |
| 1000 | Unk | 0.24 | 22.00 | 1.0000 |
| | BPE_Unk | 0.20 | 22.00 | 1.0000 |
| | BPE | 0.21 | *22.11* | 1.0000 |

Table 5.3: N-best rescore comparing BPE, the truncated vocabulary model Unk and an intermediate model BPE_Unk while varying the interpolation ratios

The interpretation of the results depends on the use of the interpolation ratios. For the optimal interpolation ratio found through optimising over the development set, we find that BPE_Unk performs similar to that of the truncated vocabulary LM (Unk). This means the BLEU improvements come from the larger vocabulary offered in the BPE encoding.

## 5.3 Lattice rescoring

In order to implement lattice rescoring, we use the lattice decoder system described in Stahlberg et al. (2016), where we employ a lattice-based beam search technique that is an extension upon the standard beam search algorithm. Lattice manipulations are done through OpenFST through PyFst (Allauzen et al. (2007)). It is compiled in C and provides an API that allows for very fast WFST operations.

We propose two methods for doing so. Firstly (*Online encoding*), to implement our neural network language models, at each node, the language model takes the list of outgoing arcs and map each of the labels into the encoding vocabulary. This allows us to rescore each of the arcs using the TensorFlow-based language model. The scored negative perplexities are then returned back to the decoder to be combined with the FST weights on the lattices themselves. Similar to N-best rescoring, we combine the language model scores with the lattice scores using an interpolation ratio $\lambda$.

Able to obtain the combined scores for each of the arcs, going from left to right, we can then utilise beam search to explore paths with the highest combined probabilities. The width of the beam search is set at 12.

A second method (*FST Transform*) is to compose the word lattice using a vocabulary mapping transducer $\mathcal{M} : \mathcal{V} \leftarrow \mathcal{V}'$, such that we obtain a subword lattice $H = \mathcal{M}\dot{L}$, where $L$ is the original word lattice. We can then rescore the subword lattices and, using the respective coding schemes, recompose the original words from the subwords.

The benefit of this approach is in the reduction of the rescoring speed, since the decoder no longer needs to transform each of the outgoing arcs and rescore them sequentially, and can simply calculate the output layer from the language at the given node and match indices of the output layer to the corresponding ID in the arcs. Table 5.4 shows the significant speed difference between the two methods.

| Method | Rescore time for *news-test15* (Seconds) | Speed-up |
|---|---|---|
| *Online encoding* | 7,884 | 1.0X |
| *FST transform* | 3,775 | 2.1X |

Table 5.4: Speed differences in lattice rescoring methods.

Given the destructive text normalisation that was performed, we do not report lattice rescoring results from this second method in our results. Nevertheless, given the uniquely decodable nature of the encodings used, if a non-destructive text normalisation was used, one could simply rescore using *FST Transform* to find the best sentence and map back to the original vocabulary $\mathcal{V}$.

## SMT Results

We find that the interpolation ratio optimised for the development *news-test14* produces worse than baseline results for the test set, so we find the best interpolation ratios empirically and report here the BLEU scores obtained for these values.

| Search space | Vocabulary Encoding | LM Interpolation Ratio | BLEU | Brevity Penalty |
|---|---|---|---|---|
| 1 | - | - | 20.63 | |
| 1000 best | Unk | 0.24 | 22.00 | 1.0000 |
| | Char | 0.20 | 21.79 | 1.0000 |
| | BPE | 0.21 | **22.11** | 1.0000 |
| | Huffman | 0.22 | 21.89 | 1.0000 |
| Lattice | Unk | 0.2 | 21.37 | 1.0000 |
| | Char | 0.01 | 20.71 | 1.0000 |
| | BPE | 0.085 | 21.17 | 1.0000 |
| | Huffman | 0.1 | 21.17 | 1.0000 |

Table 5.5: Rescoring using various RNN LMs on the *news-test15*.

The lattice rescoring does not perform nearly as well as N-best list rescoring, contrary to those suggested in Stahlberg et al. (2016). This could be the result of a problem with the decoding algorithm, or perhaps an implementation issue with respect to the TensorFlow language model. We can see here that both of the encoding methods that performed well for N-best rescoring - BPE and Huffman coding - are now underperforming compared to the truncated vocabulary language model.

Nonetheless, we report results for news-test14 using news-test15 as a development set. Here we can see that both BPE and Huffman coding perform better than that of the truncated vocabulary, though none of them showed the kind of increase from the baseline 1-best output from the lattice that we have seen from N-best rescoring for the *news-test15* dataset.

| Search space | Vocabulary Encoding | LM Interpolation Ratio | BLEU |
|---|---|---|---|
| 1 | - | - | 21.17 |
| Lattice | Unk | 0.2 | 21.35 |
| | Char | 0.01 | 21.32 |
| | BPE | 0.085 | 21.68 |
| | Huffman | 0.1 | 21.70 |

Table 5.6: Rescoring using various RNN LMs on the *news-test14*.

# Chapter 6

# Conclusion

In summary, we have achieved the two aims set out for this project. First of all, we investigated several decomposition methods, namely byte-pair encoding, Huffman coding and character decomposition, and have found byte-pair encoding to be the more effective method of decomposition. This is true in two ways: in Section 4, we concluded that BPE is better than character-level decomposition as it produces fewer subword units, especially for the most frequent words, and it is better than Huffman coding as it is able to decompose new words in the language; secondly, in Section 5, we have shown that by using byte-pair encoding we can improve the BLEU score performance for SMT systems over that of a truncated vocabulary, having also looked at whether this is due to the efficient decomposition or the enlarged vocabulary.

Secondly, we were able to implement TensorFlow based language models to the system described in Stahlberg et al. (2016). We also investigated methods to efficiently rescore N-best lists, as well as methods for rescoring word lattices with subword language models, either via online decomposition or through FST operations.

## 6.1 Future work

For further investigations, there are several ways to extend the work presented here:

- For N-best rescoring, we have only looked at using perplexity scores from left-to-right language models as feature scores. Instead, we could combine this score with that from right-to-left language models trained using the same encodings.
- In Huffman coding, the only information used for each encoded word is its frequency in the corpus. Instead of encoding words based on their frequency in the training corpus, perhaps one could encode words based on their distributional representation. Such a system would work as follows:

- Train a vector/distributional representation of the vocabulary $\mathcal{V}$, such as those described in Turney et al. (2010) or otherwise, such that for each word $i$, we have a vector $C_i \in \mathbb{R}^k$, $\mathbf{C} \in \mathbb{R}^{k,|\mathcal{V}|}$.
- For a given subword vocabulary $\mathcal{V}'$, where $N = |\mathcal{V}'|$, we cluster the embeddings using, for example, K-means to obtain $N$ clusters, assigning a label $S_i, i \in [1, N]$ to each
- Repeat the process for each cluster until the labels indicate a single word
- Traceback to obtain the overall encoding for each word
- Alternatively, instead of taking the top down approach, one could cluster together words of the closest distances until there are $N$ words in the cluster, then repeat the process for each cluster, finally tracing back to obtain the encoding. This method would be uniquely decodable as it is an analogy of Huffman coding.

• In Byte-pair encoding, we have looked at joining the most frequent character pairs in a maximum likelihood approach. However, we can take a Bayesian approach to the problem, by recognising that perhaps shorter character sequences should be preferred over long ones. This means assigning a prior distribution over the length of the byte-pairs, i.e. a Poisson distribution with a mean of $\lambda = 3$ or a geometric distribution. Such a method would encourage shorter byte-pairs and it would be interesting to see the effect this would have.

• If we ignore uniquely decodable encodings and simply view the language model over $|\mathcal{V}'|$ as a rescoring tool to produce feature scores, we can use more non-recoverable methods of encoding. An example of this would be to decompose words into their respective phonemes and training a language model to score the phonemes. We could obtain these phone decompositions by training grapheme to phonemes systems such as those in Bisani and Ney (2008) or Yao and Zweig (2015).

# Bibliography


Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., and Mohri, M. (2007). Openfst: A general and efficient weighted finite-state transducer library. In *International Conference on Implementation and Application of Automata*, pages 11–23. Springer.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Barber, D. and Botev, A. (2016). Dealing with a large number of classes–likelihood, discrimination or ranking? *arXiv preprint arXiv:1606.06959*.

Bengio, Y., Schwenk, H., Senécal, J.-S., Morin, F., and Gauvain, J.-L. (2006). Neural probabilistic language models. In *Innovations in Machine Learning*, pages 137–186. Springer.

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

Bisani, M. and Ney, H. (2008). Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5):434–451.

Bishop, C. M. (2006). *Pattern recognition and Machine Learning*.

Brants, T., Popat, A. C., Xu, P., Och, F. J., and Dean, J. (2007). Large language models in machine translation. In *In Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Citeseer.

Callison-Burch, C. (2009). Fast, cheap, and creative: evaluating translation quality using amazon's mechanical turk. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 286–295. Association for Computational Linguistics.

Cer, D., Manning, C. D., and Jurafsky, D. (2010). The best lexical metric for phrase-based statistical mt system optimization. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 555–563. Association for Computational Linguistics.

Chen, X., Liu, X., Gales, M. J., and Woodland, P. C. (2015). Recurrent neural network language model training with noise contrastive estimation for speech recognition. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5411–5415. IEEE.

Chiang, D. (2007). A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 263–270. Association for Computational Linguistics.

Chitnis, R. and DeNero, J. (2015). Variable-length word encodings for neural translation models. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Chung, J., Cho, K., and Bengio, Y. (2016). A character-level decoder without explicit segmentation for neural machine translation. *arXiv preprint arXiv:1603.06147*.

Costa-Jussà, M. R. and Fonollosa, J. A. (2016). Character-based neural machine translation. *arXiv preprint arXiv:1603.00810*.

Dahl, G. E., Yu, D., Deng, L., and Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30–42.

de Gispert, A., Virpioja, S., Kurimo, M., and Byrne, W. (2009). Minimum bayes risk combination of translation hypotheses from alternative morphological decompositions. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 73–76. Association for Computational Linguistics.

Devlin, J., Zbib, R., Huang, Z., Lamar, T., Schwartz, R. M., and Makhoul, J. (2014). Fast and robust neural network joint models for statistical machine translation. In *ACL (1)*, pages 1370–1380. Citeseer.

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

Gage, P. (1994). A new algorithm for data compression. *The C Users Journal*, 12(2):23–38.

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.

Gulcehre, C., Ahn, S., Nallapati, R., Zhou, B., and Bengio, Y. (2016). Pointing the unknown words. *arXiv preprint arXiv:1603.08148*.

Heafield, K., Pouzyrevsky, I., Clark, J. H., and Koehn, P. (2013). Scalable modified kneser-ney language model estimation. In *ACL (2)*, pages 690–696.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., r. Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., and Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97.

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

Hofstadter, D. R. (1997). *Le ton beau de Marot: In praise of the music of language*. Basic Books New York.

Huffman, D. A. et al. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101.

Jean, S., Kyunghyun, C., Memisevic, R., and Bengio, Y. (2015). On using very large target vocabulary for neural machine translation. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*.

Ji, S., Vishwanathan, S., Satish, N., Anderson, M. J., and Dubey, P. (2015). Blackout: Speeding up recurrent neural network language models with very large vocabularies. *arXiv preprint arXiv:1511.06909*.

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. (2016). Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*.

Jurafsky, D. and Martin, J. H. (2014). *Speech and language processing*. Pearson.

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2015). Character-aware neural language models. *arXiv preprint arXiv:1508.06615*.

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.

Luong, M.-T., Le, Q. V., Sutskever, I., Vinyals, O., and Kaiser, L. (2015). Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*.

Luong, M.-T. and Manning, C. D. (2016). Achieving open vocabulary neural machine translation with hybrid word-character models. *arXiv preprint arXiv:1604.00788*.

Luong, T., Socher, R., and Manning, C. D. (2013). Better word representations with recursive neural networks for morphology. In *CoNLL*, pages 104–113. Citeseer.

Luong, T., Sutskever, I., Le, Q. V., Vinyals, O., and Zaremba, W. (2014). Addressing the rare word problem in neural machine translation. *CoRR*, abs/1410.8206.

Mikolov, T. (2012). Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*.

Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *INTERSPEECH*, volume 2, page 3.

Mnih, A. and Teh, Y. W. (2012). A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*.

Och, F. J. (2003). Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 160–167. Association for Computational Linguistics.

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.

Robinson, A. J. (1989). *Dynamic error propagation networks*. PhD thesis, University of Cambridge.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Rush, A. M., Chopra, S., and Weston, J. (2015). A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*.

Sennrich, R., Haddow, B., and Birch, A. (2015). Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Snow, R., O'Connor, B., Jurafsky, D., and Ng, A. Y. (2008). Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In *Proceedings of the conference on empirical methods in natural language processing*, pages 254–263. Association for Computational Linguistics.

Stahlberg, F., Hasler, E., Waite, A., and Byrne, B. (2016). Syntactically guided neural machine translation. *arXiv preprint arXiv:1605.04569*.

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 Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Tolkien, J. R. R. (1940). *On Translating Beowulf.*

Turney, P. D., Pantel, P., et al. (2010). From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37(1):141–188.

Vaswani, A., Zhao, Y., Fossum, V., and Chiang, D. (2013). Decoding with large-scale neural language models improves translation. In *EMNLP*, pages 1387–1392. Citeseer.

Venugopalan, S., Xu, H., Donahue, J., Rohrbach, M., Mooney, R., and Saenko, K. (2014). Translating videos to natural language using deep recurrent neural networks. *arXiv preprint arXiv:1412.4729.*

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

Yao, K. and Zweig, G. (2015). Sequence-to-sequence neural net models for grapheme-to-phoneme conversion. *arXiv preprint arXiv:1506.00196.*

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

Zipf, G. K. (2016). *Human behavior and the principle of least effort: An introduction to human ecology.* Ravenio Books.

# Appendix A

# Appendices

## A.1 Error back-propagation

We describe here the error back-propagation commonly used for training feed-forward neural networks. Here we use the cross-entropy between the output distribution and the target distribution as our error function.

$$E = -\sum_{i \in \mathcal{V}} \sum_{n=1}^{N} (t_{n,i} \log(y_i(\mathbf{x}_n)) + (1 - t_{n,i}) \log(1 - y_i(\mathbf{x}_n))) \tag{A.1}$$

Since we can relatively easily calculate the error gradient in the output layer $\frac{\partial E}{\partial y}$, we can then calculate the error gradient for the weights going from the penultimate layer to the output layer through the chain rule:

$$\frac{\partial E}{\partial \mathbf{W}^{(K-1)}} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial \mathbf{z}^{(K-1)}} \frac{\partial \mathbf{z}^{(K-1)}}{\partial \mathbf{W}^{(K-1)}} \tag{A.2}$$

$$= \Delta^{(K-1)} \mathbf{x}^{(K-1)} \tag{A.3}$$

$$\text{where } \Delta^{(K-1)} = \frac{\partial E}{\partial \mathbf{z}^{(k)}} \tag{A.4}$$

Similarly, since we know the activation functions for the rest of the hidden layers, we can then apply the chain rule again to calculate their respective error gradients.

$$\frac{\partial E}{\partial \mathbf{W}^{(k)}} = \Delta^{(k)} \mathbf{x}^{(k)} \tag{A.5}$$

$$\Delta^{(k)} = \frac{\partial E}{\partial \mathbf{z}^{(k)}} \tag{A.6}$$

$$= \frac{\partial E}{\partial \mathbf{z}^{(k+1)}} \frac{\partial \mathbf{z}^{(k+1)}}{\partial \mathbf{z}^{(k)}} \tag{A.7}$$

$$= \Delta^{(k+1)} \frac{\partial \mathbf{z}^{(k+1)}}{\partial \mathbf{z}^{(k)}} \tag{A.8}$$

$$= \Delta^{(k+1)} \frac{\partial \mathbf{z}^{(k+1)}}{\partial \mathbf{y}^{(k)}} \frac{\partial \mathbf{y}^{(k)}}{\partial \mathbf{z}^{(k)}} \tag{A.9}$$

$$= \Delta^{(k+1)} \mathbf{W}^{(\mathbf{k+1})} \frac{\partial \mathbf{y}^{(k)}}{\partial \mathbf{z}^{(k)}} \tag{A.10}$$

Where $\frac{\partial \mathbf{y}^{(k)}}{\partial \mathbf{z}^{(k)}}$ can be calculated, as it is simply the derivative of the activation function.

As such, with these back-propagating error corrections, the neural network is trained to slowly approximate a function which would produce the target outputs for the given input.

## A.2 Byte-pair encoding

We can implement Byte Pair Encoding using the following pseudo-code:

```
# Count the frequency of words in the corpus
counts = dict()
for line in text:
        words = tokenize(line)
        for word in words:
                counts[word] += 1

# Find a mapping of the vocabulary to its constituent pairs
vocab = dict()
for word in counts.keys:
        vocab[word] = list(word)+"</w>"

# Find the initial counts of all pairs of joined
pairs = dict()
for word in vocab.keys:
        chars = vocab[word]
        for i = 1, ..., len(chars)-1:
                pair = merge(chars[i],chars[i+1])
                pairs[pair] += counts[word]

while iterations < Threshold:
        # Find the most frequent pairs
        max_pair = argmax(pairs)
        output(max_pair)

        # Update the vocabulary by merging the pairs in the vocabulary,
           keeping track of which pairs have been changed
        changed = []
        for word in vocab.keys:
```

```
            chars = vocab[word]
            if max_pair in chars:
                vocab[word] = chars.sub(max_pair, join(max_pair)
                    )
                changed += word
        pairs[max_pair] = 0

        # Update the list of pairs to include new pairs made possible by
            the merged pairs
        for word in changed:
            pairs = update_pairs(pairs, word, max_pair)

        if max(pairs) < 2:
            break
```