Reward Augmented Maximum Likelihood to Improve Neural Machine Translation Training

Proyag Pal (s1620871)

Master of Science
School of Informatics
University of Edinburgh
2017

Abstract

Neural Machine Translation in its current form suffers from some problems such as loss-evaluation mismatch and exposure bias. Reward Augmented Maximum Likehood is a technique that directly incorporates the task evaluation metrics such as BLEU score into the traditional maximum likelihood training framework. This is done by augmenting the training output targets with outputs that are sampled proportional to their exponentiated scaled rewards, on which cross-entropy is optimised. This is a more computationally efficient method than reinforcement learning-based methods and can be trained effectively from a cold start without bootstrapping with a cross-entropy trained model. This project implements Reward Augmented Maximum Likelihood in the Nematus neural machine translation framework, and observes significant improvements in BLEU score over a model trained to optimise perplexity.

Acknowledgements

I would like to convey my thanks to Dr. Kenneth Heafield, whose able supervision and guidance made this entire project possible, enjoyable, and hopefully successful. I would also like to thank Dr. Sharon Goldwater and Dr. Henry S. Thompson, whose wonderful course sparked my interest in the field of natural language processing, and Dr. Adam Lopez, for a fascinating course which left me with no doubt about wanting to work with machine translation.

I must also express my gratitude to my friends in Edinburgh as well as back home, and most importantly, my parents and my sister, for their support.

Declaration

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

(*Proyag Pal (s1620871)*)

Table of Contents

Chapter 1

Introduction

1.1 Before Neural Machine Translation

Machine translation, the automatic translation of one human language to another, has been a subject of research for decades. Most successful machine translation systems have been based on statistical models, learning to translate between languages by collecting statistics from a large parallel corpus of text in two languages (Koehn, 2010). The IBM models (Brown et al., 1993), based on the alignment of words between the source and target languages, were the first statistical machine translation models.

Phrase-based models (Och and Weber, 1998; Wang and Waibel, 1998; Och et al., 1999) use phrases as the units to be aligned instead of individual words. In this context, phrases are defined as contiguous sequences of words with no linguistic significance. Word-based and phrase-based machine translation systems are built out of a number of smaller components, which model latent structures like word/phrase alignment, sentence segmentation, and phrase reordering. For a long time, phrase-based systems constituted the state of the art; that is, until the advent of neural machine translation.

1.2 Neural Machine Translation

In recent times, neural machine translation (NMT) (Bahdanau et al., 2014; Cho et al., 2014; Sutskever et al., 2014) systems have emerged as a dominant force in the field of machine translation. They have surpassed phrase-based machine

translation as the best performing systems for a large number of language pairs (Bentivogli et al., 2016; Junczys-Dowmunt et al., 2016). NMT systems train one large neural network which takes a sentence in the source language as input, and produces its translation in the target language as output. The standard network architecture used in NMT systems is what is known as an encoder-decoder model, which is described in Section 2.1.1.

However, despite showing significant improvements in translation quality over phrase-based systems in many cases, NMT systems suffer from some problems, as listed below.

- **Loss-evaluation mismatch -** The loss function that is optimised to train the model is different from the metric used to evaluate the results.
- **Exposure bias -** The model only sees ground truths at training time, but words generated at test time may contain errors (Ranzato et al., 2015).
- **Label bias -** Low-entropy successors of incorrect histories appear equally probable as high-entropy successors of correct histories (Lafferty et al., 2001).

These problems are explained and discussed in greater detail in Section 3.1.

The aim of this project is to implement a technique that is designed to overcome these problems. There have been a number of papers published on the subject, attempting to solve the aforementioned problems in different ways. Of these, Reward Augmented Maximum Likelihood (Norouzi et al., 2016) has been chosen for this project. This technique has been implemented and the results compared against existing NMT baselines.

1.3 Outline of the Thesis

Chapter 2 contains a brief introduction to the neural machine translation paradigm. Section 2.1 describes the RNN encoder-decoder with attention model that is commonly used in NMT systems, detailing the neural network architecture, and the loss function that is optimised. Section 2.2 contains a brief overview of metrics for the evaluation of translation quality.

Chapter 3 discusses the problems with the existing systems (Section 3.1), followed by some techniques that have been suggested to improve them (Section 3.2).

Chapter 4 deals with Reward Augmented Maximum Likelihood and its implementation. Section 4.1 discusses the technique in detail. Section 4.2 describes some details of how the method will be implemented. Section 4.3 contains a description of the software framework, Nematus, that has been used for implementation.

Chapter 5 details the experiments that were run to test the implementation of Reward Augmented Maximum Likelihood and their results. Section 5.1 describes the data on which models are trained. Section 5.2 establishes the baseline models against which the new models will be compared. Section 5.3 describes how the experiments were designed to examine the effect of the new technique. Section 5.4 analyses the results of the experiments.

Finally, Chapter 6 summarises the work done in this project, and briefly explores some possible directions for future work in this area.

Chapter 2

Background

2.1 Neural Machine Translation (NMT)

This section (based on Section 2 of my IRP report) describes the general neural architecture used for machine translation, and how the networks are trained. Section 2.1.1 discusses the common encoder-decoder model, which is augmented by attention mechanisms (Section 2.1.2) in modern neural MT systems. Section 2.1.3 explains how these models are trained to maximise the likelihood of the parallel data.

Given a source sequence $\mathbf{x}$, the aim is to generate a translation $\mathbf{y}$ such that

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}}\, P(\mathbf{y}|\mathbf{x};\theta) \tag{2.1}$$

where θ represents the model parameters estimated from the data.

However, due to the large search space and the model being non-Markovian, it is not feasible to always find the exact sequence with the highest probability, and the search process has to be approximated by generating the target sequence one word at a time, and using a beam search to keep track of multiple good candidate translations.

For source sentence $\mathbf{x} = \mathbf{x}_1 \dots \mathbf{x}_m \dots \mathbf{x}_M$ and target sentence $\mathbf{y} = \mathbf{y}_1 \dots \mathbf{y}_n \dots \mathbf{y}_N$, a neural MT system models the translation probability as

$$P(\mathbf{y}|\mathbf{x};\theta) = \prod_{n=1}^{N} P(\mathbf{y}_n|\mathbf{y}_{<n}, \mathbf{x};\theta) \tag{2.2}$$

where $\mathbf{y}_{<n}$ represents the first $n-1$ words of the generated sentence.

The idea of encoder-decoder models has been around in the machine translation literature for a long time (Chrisman, 1991; Forcada and Ñeco, 1997). Modern end-to-end neural machine translation systems, based on the idea of one large neural network that is trained end-to-end, have evolved from architectures such as those used by Kalchbrenner and Blunsom (2013a), Cho et al. (2014), and Sutskever et al. (2014). Kalchbrenner and Blunsom (2013a) mapped the entire source sentence to a single fixed-length vector $\mathbf{c}$ using a convolutional neural network (Kalchbrenner and Blunsom, 2013b), and generated the target sentence one word at a time, with each word conditioned solely on the previously generated words and the source sentence encoding $\mathbf{c}$, in the form of a recurrent language model (Mikolov et al., 2010).

2.1.1 RNN Encoder-Decoder Architecture

Current neural MT systems differ from that of Kalchbrenner and Blunsom (2013a) mainly in the encoder mechanism. They are usually designed in the form of RNN encoder-decoder models (Sutskever et al., 2014; Cho et al., 2014). In this kind of model, both the encoder and decoder are modeled as some variety of a recurrent neural network (RNN). To facilitate explanation without loss of generality, this section assumes that Elman RNNs (Elman, 1990) are being used; more complicated flavours of RNN are touched upon towards the end of this subsection.

Sutskever et al. (2014) and Cho et al. (2014) use an RNN to map the input sequence to a fixed length vector $\mathbf{c}$, and then use $\mathbf{c}$ to initialise a recurrent neural network language model (Mikolov et al., 2010), which then generates the target sequence one word at a time, as illustrated by Figure 2.1. The entire encoder-decoder network is trained end-to-end.

The Encoder

The encoder RNN takes one word of the source sequence $\mathbf{x}_t$ as input at each step, and updates its hidden state as follows

$$\mathbf{h}_t = u(\mathbf{x}_t, \mathbf{h}_{t-1}) \tag{2.3}$$

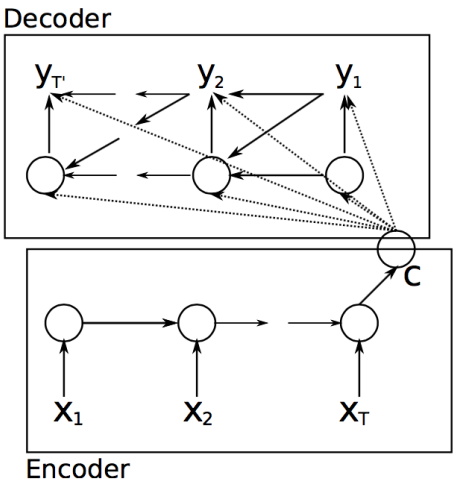

Figure 2.1: RNN encoder-decoder model as implemented by Cho et al. (2014). The encoder RNN encodes the source sequence $\mathbf{x} = \mathbf{x}_1 \dots \mathbf{x}_t \dots \mathbf{x}_T$ into a fixed length vector $\mathbf{c}$, and the decoder RNN generates the target sequence $\mathbf{y} = \mathbf{y}_1 \dots \mathbf{y}_t \dots \mathbf{y}_{T'}$ one word at a time, where each output step $\mathbf{y}_t$ is conditioned on $\mathbf{y}_{t-1}$ and also on $\mathbf{c}$.

where $\mathbf{h}_t$ is the state of the RNN at time step t. u is a non-linear *update* function, which for an Elman RNN (Elman, 1990) can be expanded as[1]

$$\mathbf{h}_t = \sigma(W_i \mathbf{x}_t + W_h \mathbf{h}_{t-1}) \tag{2.4}$$

where W_i and W_h are matrices that are parameters of the RNN that are included in the trainable model parameters θ.

The source encoding $\mathbf{c}$ is generated using the set of states of the encoder RNN. Exactly how the source encoding is formed from the states varies between implementations; for example, Cho et al. (2014) used simply the final RNN state $\mathbf{h}_M$. Bahdanau et al. (2014) proposed using a feed-forward neural network with a tanh non-linearity on the final encoder state. The mean of the state observed over all the time steps, or generally some other function of the set of states $f(\mathbf{h}_1, \dots, \mathbf{h}_M)$ could also be used. Intuitively, the vector $\mathbf{c}$ is expected to summarise all the information contained in the source sequence.

[1]Bias terms are omitted here and in other such equations in this thesis, for the sake of simplicity.

The Decoder

The decoder uses another similar RNN mechanism, which has its hidden state $\mathbf{s}$ initialised with $\mathbf{c}$ (which is the encoding of the source sentence obtained as described in the previous section), and is updated as follows

$$\mathbf{s}_t = u(\mathbf{y}_{t-1}, \mathbf{s}_{t-1}, \mathbf{c}) \tag{2.5}$$

The decoder thus generates probabilities over words conditioned on the previous words and the source representation as

$$p(\mathbf{y}_n|\mathbf{y}_{<n}, \mathbf{x}) = g(\mathbf{y}_{n-1}, \mathbf{s}_n, \mathbf{c}) \tag{2.6}$$

where g is usually formulated as a feed-forward neural network from the RNN state followed by a softmax output layer to produce probability distributions over the target vocabulary at each step. This can be expressed as

$$\mathbf{o}_n = W_o \mathbf{s}_n, \tag{2.7}$$

$$\mathbf{y_n} \sim \text{softmax}(\mathbf{o}_n) \tag{2.8}$$

The probability over the entire target sequence is given by

$$p(\mathbf{y}) = \prod_{n=1}^{N} p(\mathbf{y}_n|\mathbf{y}_{<n}, \mathbf{c}) \tag{2.9}$$

The entire encoder-decoder network is trained end-to-end to maximise these probabilities for the ground truth inputs, as elaborated in Section 2.1.3.

Note: Advanced RNN Architectures

It has been shown that using Elman RNNs does not work very well when the sentences are long because they face difficulties modelling long-term dependencies (Bengio et al., 1994). While an Elman RNN has been used to describe the encoder-decoder architecture above, other flavours of RNN such as LSTM and GRU are usually used in machine translation systems. The following paragraphs briefly touch upon these; however, the basic form of the RNN update shown in Equations 2.3 and 2.5 remains unchanged for any flavour of RNN. They only differ in the details of the update function u.

LSTM: Long Short-Term Memory (LSTM) networks, originally formulated by Hochreiter and Schmidhuber (1997), and later modified most notably by Gers et al. (2000) and Gers and Schmidhuber (2000), is a popular variant of RNNs. An LSTM maintains its internal state over time, and has three non-linear *gates* - the input, output, and forget gates - which control the flow of information in and out of the LSTM cell (Greff et al., 2016). LSTMs have been shown to be able to model long-term dependencies more effectively than vanilla RNNs, and are widely used in a variety of models.

GRU: Cho et al. (2014) proposed a simplified version of the LSTM called Gated Recurrent Unit (GRU), which combined the input and forget gates into an update gate, and also modified the mechanism of the output gate. Chung et al. (2014) found GRUs to outperform LSTMs on certain tasks, but neither was found to be universally better. Both these flavours of RNNs are used in various applications; in this project, a variant of GRU has been used for both the encoder and the decoder in the system.

Bidirectional RNN

While the encoders used by Cho et al. (2014) and Sutskever et al. (2014) use a single, possibly multi-layered, RNN to process the source sentence into its fixed length vector representation, the RNN used by Bahdanau et al. (2014) is a bidirectional RNN (Schuster and Paliwal, 1997). A bidirectional RNN has a forward RNN which reads the source sequence from start to finish in order, whose hidden states are $\overrightarrow{\mathbf{h}}_1 \dots \overrightarrow{\mathbf{h}}_M$, and a backward RNN which reads the sequence in reverse order and has hidden states $\overleftarrow{\mathbf{h}}_1 \dots \overleftarrow{\mathbf{h}}_M$. For a word $\mathbf{x}_m$ in the source sentence, the concatenation of $\overrightarrow{\mathbf{h}}_m$ and $\overleftarrow{\mathbf{h}}_m$ is known as its annotation, and these annotations are used during the decoding process to focus attention on different parts of the input, as explained in Section 2.1.2.

2.1.2 The Attention Mechanism

The problem with the basic formulation of the encoder-decoder model of neural machine translation is that the model is expected to encode sentences of any length into one vector of fixed length. If the vector is not long enough, then it becomes difficult to encode all the information in the sentence. On the other hand,

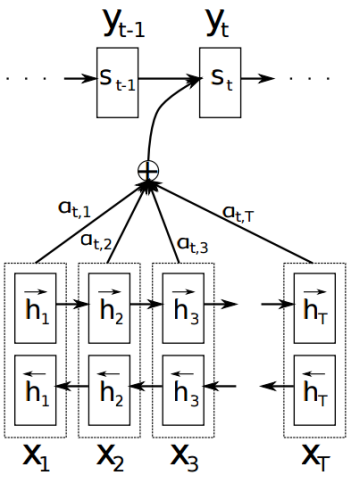

Figure 2.2: The RNN encoder-decoder with attention model (Bahdanau et al., 2014). The source sequence is represented as $\mathbf{x}_1 \dots \mathbf{x}_T$. The annotation for input word $\mathbf{x}_i$ is obtained by concatenating the forward and backward hidden states $\overrightarrow{\mathbf{h}_i}$ and $\overleftarrow{\mathbf{h}_i}$. The word generated at the t-th position is $\mathbf{y}_t$, and the hidden state of the decoder is $\mathbf{s}_t$, which is conditioned on $\mathbf{s}_{t-1}$ and a weighted average of the input annotations with the weights being $\alpha_{t,i}$.

if the vector is very long, the number of parameters in the network becomes very large, and it takes an infeasible amount of data to train the model. The solution to this problem is the attention mechanism, described here based on my IRP report.

Rather than encoding the entire source sequence into a fixed-length representation, the model keeps vectors for each word in the input sequence, referred to as annotations. The decoder RNN state is conditioned on a position-dependent context vector which is constructed by focusing on specific parts of the input sequence. The conditional word probabilities generated by the decoder are given by

$$p(\mathbf{y}_n|\mathbf{y}_{<n}, \mathbf{x}) = g(\mathbf{y}_{n-1}, \mathbf{s}_n, \mathbf{c}_n) \tag{2.10}$$

This only differs from Equation 2.6 in that $\mathbf{c}$, known as the context vector, varies depending upon the position in the decoder's generative process. $\mathbf{c}_n$ is derived from the source word annotations. Due to recency biases in RNN architectures, the annotation for a source word $\mathbf{x}_m$ contains information concentrated on words close to $\mathbf{x}_m$ before or after it. The context vectors based upon these

annotations allows an output word to softly align to a particular section of the source sequence where its meaning may be concentrated.

The context vector $\mathbf{c}_n$ is calculated as a weighted sum of the source sequence annotations as follows:

$$\mathbf{c}_n = \sum_{i=1}^{M} \alpha_{ni} \mathbf{h}_i \tag{2.11}$$

where the weights α_{ni} are calculated as

$$\alpha_{ni} = \frac{\exp(e_{ni})}{\sum_{k=1}^{M} \exp(e_{nk})} \tag{2.12}$$

where e_{ni} is a measure of the similarity between the encoder hidden state at position i with the decoder hidden state at position n. This measure can be computed in the form of a feed-forward neural network on the concatenation of the two states as done by Bahdanau et al. (2014) or with other similarity measures like dot product (Luong et al., 2015). This measure simulates the concept of alignment in phrase-based models and conceptually, it gives the decoder an idea of what part of the source sentence contains the most information relevant to the next word being generated.

The attention-based encoder-decoder model described so far, as designed by Bahdanau et al. (2014) and illustrated in Figure 2.2, is the standard network architecture used in modern neural machine translation systems. It is also the model used in this project, with a few modifications in the implementational details that have been discussed in Section 4.3.

2.1.3 Training RNN Encoder-Decoder Models

In the previous two sections, the structure of the neural encoder-decoder model has been described. This section discusses how these networks are trained, including the cross-entropy loss function and how it is optimised.

2.1.3.1 Cross-entropy and Perplexity

The cross-entropy of a sequence is the negative log likelihood per word of the sequence, given by

$$H(p) = -\frac{1}{N} \log_2 p(\mathbf{y}_1 \dots \mathbf{y}_N | \mathbf{x}) \tag{2.13}$$

Perplexity is defined by a simple transformation of cross-entropy (Koehn, 2010); it is given by

$$PP = 2^{H(p)} \tag{2.14}$$

The perplexity of a model is a measure of how uncertain the model is about generating a new word at each step. Therefore, the lower the perplexity of the model is, the more certain it is when predicting a word in the target sequence conditioned upon the previous words and the source sequence.

The cross-entropy loss used in machine translation is the negative log likelihood of the target sequence given the source sequence. The loss function is given by

$$\mathcal{L}_{CE} = -\log_2 p(\mathbf{y}|\mathbf{x}) \tag{2.15}$$

$$= -\log_2 \prod_{n=1}^{N} p(\mathbf{y}_n|\mathbf{y}_{<n}, \mathbf{x}) \tag{2.16}$$

The model is trained end-to-end to minimise cross-entropy loss over the entire parallel corpus, or in other words, to estimate parameters to maximise the likelihood of the data. It is easy to see that cross-entropy is a strictly word-level loss function, while the evaluation metrics in Section 2.2 are sentence-level measures. This is one of the chief motivations for this project, as discussed in Section 3.1.

2.1.3.2 Optimisation

Maximum Likelihood The standard method of training neural machine translation models is known as maximum likelihood training. Given a training set of sentence pairs $D = \{\langle \mathbf{x}^{(s)}, \mathbf{y}^{(s)} \rangle\}_{s=1}^{S}$, the model tries to find a set of parameters $\hat{\theta}$ such that

$$\hat{\theta}_{\text{MLE}} = \underset{\theta}{\text{argmax}} \{\mathcal{L}(\theta)\} \tag{2.17}$$

where $\mathcal{L}(\theta)$ is the log likelihood of the data given a set of parameters θ, calculated as

$$\mathcal{L}(\theta) = \sum_{s=1}^{S} \log_2 p_\theta(\mathbf{y}^{(s)}|\mathbf{x}^{(s)}) \tag{2.18}$$

The term $p_\theta(\mathbf{y}^{(s)}|\mathbf{x}^{(s)})$ is computed as in Equation 2.2. This is equivalent to minimising the cross-entropy loss of the target sequences over the entire corpus.

Stochastic Gradient Descent For non-convex problems such as the neural network models used for machine translation, there does not exist a closed form solution to find the parameters that maximise the likelihood. Instead, an iterative optimisation procedure such as stochastic gradient descent, or one of its variants like Adam (Kingma and Ba, 2014) or Adadelta (Zeiler, 2012), is used to find parameters that maximise the likelihood. Due to the non-convex nature of the problem, there is no guarantee that the globally optimum combination of parameters will be found. Stochastic gradient descent starts from a random initialisation of parameters, and iteratively moves the parameters in a direction which reduces the cross-entropy error, with step sizes proportional to the gradient of the error function with respect to the parameters. Through this process, it finds a local optimum, and the parameters at this local optimum define the trained model which is used for translation.

The neural encoder-decoder model is trained end-to-end to minimise cross-entropy loss using back-propagation, back-propagation through time (BPTT) (Williams et al., 1986) on the RNNs, and stochastic gradient descent (or one of its variants) for optimisation.

2.2 Evaluation

This section discusses the common metrics used for evaluation of machine translation quality, focussing on BLEU. As will be evident from this section, these evaluation metrics measure translation quality at the sentence level, in contrast to the word-level cross-entropy loss function that the model is trained to optimise.

2.2.1 BLEU: A Bilingual Evaluation Understudy

The most popular metric used for the automatic evaluation of translation quality is BLEU (Papineni et al., 2002). It works by matching n-grams between reference (ground truth) and translation hypotheses. It is calculated as

$$\text{BLEU-}n = BP * \exp \sum_{i=1}^{n} \lambda_i \log(precision_i) \tag{2.19}$$

where $precision_i$ is the precision at the i-gram level, n is the maximum order for which n-grams are matched, and λ_i are weights for the precisions for different

values of i. BP is the brevity penalty which penalises hypotheses that are shorter than the reference translation, computed as

$$BP = \min\left(1, \exp\left(1 - \frac{reference\_length}{hypothesis\_length}\right)\right) \tag{2.20}$$

The most commonly used highest n-gram order for calculating BLEU-n is n=4. λ_i is set to $1/n$ for all values of i. This reduces the BLEU measure to

$$\text{BLEU} = \min\left(1, \exp\left(1 - \frac{reference\_length}{hypothesis\_length}\right)\right)\left(\prod_{i=1}^{4} precision_i\right)^{1/4} \tag{2.21}$$

The n-gram precision of any order being 0 causes the BLEU score as calculated above to be 0 overall, irrespective of any other matches. Since n-gram precisions, especially for higher order n-grams, may be zero in a given sentence, BLEU scores are typically computed over the entire test corpus instead of at the sentence level (Koehn, 2010, Section 8.2.3).

If sentence-level BLEU scores are needed, smoothing techniques may be applied on the BLEU calculation. There are many possible smoothing techniques (Chen and Cherry, 2014). One possible technique, which is used in this project, adds 1 to the matched n-gram count (the numerator in the calculation of precision), and also adds 1 to the total n-gram count (the denominator in the calculation of precision) for $2 \leq i \leq n$.

Evaluation of all techniques in this project uses a document-level BLEU-4 score, and for places where a sentence-level metric is needed, the smoothed BLEU score with n=4 is used. BLEU scores range from 0 to 1. Scores are reported as $100 * \text{BLEU-4}$, with 0% being the worst and 100% the best possible. However, an absolute BLEU score is not interpretable, since the score depends on many factors like the language pair, tokenisation methods, etc. The effectiveness of any machine translation system can only be evaluated by comparing against a reasonable baseline.

2.2.2 Other Metrics

NIST A common criticism of the BLEU metric is that it treats all words as equal, where the presence of more informative words like names in the translation should be given more importance than trivial words like determiners and

punctuation (Koehn, 2010, Section 8.2.5). The NIST metric (Doddington, 2002) slightly modifies the BLEU measure to weight n-grams higher if they are rarer, resulting in more informative n-grams being given more importance in the overall score. The calculation of the brevity penalty is also different.

METEOR Another common criticism of the BLEU metric is that it scores near misses in translation (for example, *different* instead of *difference*) the same as completely wrong translations. The METEOR metric (Denkowski and Lavie, 2014) attempts to overcome this by considering morphologically and semantically similar words while scoring a translation. However, this results in the computation being much more complicated and expensive than BLEU. In addition, compared to BLEU, METEOR places a greater emphasis on recall.

Hamming and Edit Distance Hamming distance is the number of positions where the tokens are not identical for two sentences of equal length. Edit distance, or Levenshtein distance, is the minimum number of insertion, deletion, and substitution operations required to transform one sentence to the other. These are not used for evaluation of translation quality in practice. However, they are a measure of difference between a hypothesis and a reference translation, and are mentioned in this section because they have been used in the implementation of this project as a reward function (see Section 4.2).

Chapter 3

Advanced Training Methods

This chapter deals with some of the problems that exist in neural machine translation (Section 3.1), followed by some of the techniques that have been proposed to mitigate these problems (Section 3.2).

3.1 Problems with NMT Training

The neural architecture described in Section 2.1 has been used successfully enough to surpass traditional phrase-based systems in terms of translation quality for many languages. However, there remain three notable drawbacks that these systems suffer from. These are explained (based on Section 1 of my IRP report) in this section.

3.1.1 Loss-Evaluation Mismatch

As observed by Ranzato et al. (2015), one fundamental problem in the neural machine translation systems described so far is the fact that the metric for evaluation of translation quality is not being optimised directly. As discussed previously in Section 2.2, the metrics used for evaluation of machine translation are sentence-level metrics such as BLEU or METEOR. These are discrete, non-differentiable measures, and are therefore not amenable to gradient-based optimisation methods. Cross-entropy (see Section 2.1.3), in contrast, is a differentiable function. NMT systems are thus designed to optimise cross-entropy loss. While minimising perplexity is a reasonable choice to train these systems, it is not a very good metric for evaluation. It is assumed that perplexity is a good substi-

tute for optimisation, but there is no guarantee that this optimisation correlates well with the maximisation of the evaluation metrics of translation quality.

3.1.2 Exposure Bias

The model is trained on corpora of gold standard translations, and thus, during training, the model is only exposed to the ground truths. At test time, the model generates translations one word at a time, based on the source sentence and its own previously predicted words. Errors may occur in these translated words, and then the rest of the words will be conditioned on these erroneous outputs, which the model has never been exposed to while training. When that happens, the errors start accumulating, and the model is unable to recover to generate a good translation. In other words, the distribution on which the model is trained and the distribution from which it draws words for generation of the target sentence are not the same. The model is therefore not very robust. This is referred to as exposure bias by Ranzato et al. (2015).

3.1.3 Label Bias

Wiseman and Rush (2016) pointed out another problem with these models. These models are locally normalised, i.e., the output word probabilities at each step are normalised. As a result, if the successors of an erroneous output word have low entropy, the sentence will be as probable as with high-entropy successors of a correct output word. Therefore, if the model generates an erroneous word with low-entropy successors, the sentence appears to be as good as a correct translation in terms of its overall probability. As a result, the model is unable to recover from the error, and generates a bad translation. This problem is known as label bias (Lafferty et al., 2001).

3.2 Improved Training Methods

Faced with the flaws in the neural machine translation technique given in the previous section, there have been a number of efforts to mitigate these issues. Attempts to incorporate the task reward into the optimisation process and/or to expose the model to its own predictions during training have shown improvements over the systems trained to optimise perplexity. Some of these techniques are

briefly explained in this section (taken almost verbatim from my IRP report) to give a sense of the great variety of approaches being explored. The technique touched upon in Section 3.2.5 has been chosen for implementation in this project, and is explored in much greater detail in Chapter 4.

3.2.1 Minimum Risk Training

Minimum risk training (Shen et al., 2016) enables the direct use of sentence-level metrics for the parameter optimisation process. It optimises the expected loss on the training data, where the loss function is a sentence-level metric like BLEU (Papineni et al., 2002) or METEOR (Denkowski and Lavie, 2014). This expected loss is called the "risk", and is calculated by Shen et al. (2016) as

$$\mathcal{R}(\theta) = \sum_{s=1}^{S} \mathbb{E}_{\mathbf{y}|\mathbf{x}^{(s)};\theta}[\Delta(\mathbf{y}, \mathbf{y}^{(s)})] \tag{3.1}$$

$$= \sum_{s=1}^{S} \sum_{y \in \mathcal{Y}(\mathbf{x}^{(s)})} p(\mathbf{y}|\mathbf{x}^{(s)}; \theta)\Delta(\mathbf{y}, \mathbf{y}^{(s)}) \tag{3.2}$$

where $\mathcal{Y}(\mathbf{x}^{(s)})$ is the set of all possible translations for a given input sentence $\mathbf{x}^{(s)}$, and $\Delta(\mathbf{y}, \mathbf{y}^{(s)})$ is the reward function. Since calculating this for all possible translations is intractable, the search space is sampled to approximate the expectation of loss.

By directly optimising towards the evaluation metric, minimum risk training eliminates the discrepancy between training objective and evaluation criteria. Also, since the sampling process exposes the model to its own predictions during training, the problem of exposure bias is also mitigated.

3.2.2 MIXER

One way to directly optimise the model towards discrete evaluation criteria is to formulate it as a reinforcement learning problem. The REINFORCE algorithm (Williams, 1992) can be used in this context by thinking of the generative RNN model as an agent. The agent learns by observing the rewards at the end of each sequence it generates, where the reward can be the BLEU or METEOR score or any other evaluation metric with respect to the ground truths of the training examples. The training process optimises the expected reward for generated

sequences. The loss function can thus be written as

$$\mathcal{L}(\theta) = -\mathbb{E}_{\mathbf{y}\sim p_\theta}[r(\mathbf{y})] \tag{3.3}$$

where the right hand side is approximated by the reward r observed from a single sequence $\mathbf{y}$ drawn from the action space of the RNN.

It is very difficult to successfully train the REINFORCE algorithm starting from a random policy when the state action space is very large, like in generating sentences where each word can be chosen from the entire target vocabulary. The Mixed Incremental Cross-Entropy Reinforce (MIXER) algorithm (Ranzato et al., 2015) overcomes this drawback by bootstrapping with a model trained on cross-entropy error. It makes another modification, which is that it gradually exposes the model to its own predictions at training time. The training process starts with only ground truth inputs, and gradually introduces the model's own predictions for later sequence steps based on an annealing schedule, and eventually uses only its own predictions for the final epochs of training.

Due to this incorporation of model predictions into the training process, the model is able to avoid exposure bias, and the evaluation metrics being used directly as rewards eliminates the loss-evaluation mismatch.

3.2.3 Globally Normalised Transition-Based Neural Networks

Andor et al. (2016) have shown that globally normalised transition-based neural networks perform better than locally normalised ones. As a reminder, in the context of generative RNN models as described so far, the output of each step is a locally normalised probability distribution over the vocabulary of the target language given by[1]

$$p(\mathbf{y}_n|\mathbf{y}_{<n};\theta) = \frac{\exp(f(\mathbf{y}_n, \mathbf{y}_{n-1}, \mathbf{s}_n;\theta))}{Z_L(\mathbf{y}_{n-1}, \mathbf{s}_n;\theta)} \tag{3.4}$$

where f is a scoring function (such as a feed-forward neural network from the RNN hidden state) similar to g as used in Equation 2.10, but without the final

[1]The notation in Equation 2.10 did not show $\mathbf{y}_n$ on the right hand side; the equation instead expressed a probability distribution obtained after a softmax. However, the function f on the right hand side here does not include the softmax, and shows the probability calculation for $\mathbf{y}_n$ making the normalisation step explicit.

softmax layer, and thus the expression in Equation 3.4 shows how g is formulated. $Z_L(\mathbf{y}_{n-1}, \mathbf{s}_n; \theta)$ is the normalisation term

$$Z_L(\mathbf{y}_{n-1}, \mathbf{s}_n; \theta) = \sum_{\mathbf{y}'_n \in \mathcal{V}} \exp(f(\mathbf{y}'_n, \mathbf{y}_{n-1}, \mathbf{h}_n; \theta)) \tag{3.5}$$

The probability of the full sequence is thus given by

$$\mathbf{y} = \prod_{n=1}^{N} p(\mathbf{y}_n | \mathbf{y}_{<n}; \theta) \tag{3.6}$$

$$= \frac{\exp \sum_{n=1}^{N} f(\mathbf{y}_n, \mathbf{y}_{n-1}, \mathbf{s}_n; \theta)}{\prod_{n=1}^{N} Z_L(\mathbf{y}_{n-1}, \mathbf{s}_n; \theta)} \tag{3.7}$$

In contrast, for globally normalised models, instead of having the local normalisation term Z_L at each step, the unnormalised scores calculated by f are used, and a global normalisation term is used for computing the probability of the whole sequence as

$$p(\mathbf{y}) = \frac{\exp \sum_{n=1}^{N} f(\mathbf{y}_n, \mathbf{y}_{n-1}, \mathbf{s}_n; \theta)}{Z_G(\theta)} \tag{3.8}$$

where

$$Z_G(\theta) = \sum_{\mathbf{y}' \in \mathcal{Y}_n} \exp \sum_{n=1}^{N} f(\mathbf{y}'_n, \mathbf{y}_{n-1}, \mathbf{s}_n; \theta) \tag{3.9}$$

where $\mathcal{Y}_n$ is the set of all possible sequences of length n. Since actually using $\mathcal{Y}_n$ is intractable, beam search is used.

Global normalisation eliminates the problem of label bias in RNN models. The beam search method also ensures that the model is exposed to its own predictions while training.

3.2.4 Beam Search Optimisation

Wiseman and Rush (2016) proposed another modification of the model training method where the output at each step is an unnormalised score, instead of normalised probabilities. The function g in Equation 2.10 consists of a feed-forward neural network which has an output with same number of dimensions as the size of the target vocabulary, followed by a softmax layer which transforms that output into a discrete probability distribution over the vocabulary. In this method, a function f is used, which is identical to g, except it does not have the

final softmax layer. As a result, the output at each step consists of unnormalised activations instead of a locally normalised probability distribution. This helps to avoid the label bias problem.

Moreover, a version of beam search is used during the training process. At time step t, the generative RNN maintains a set of the K best sequences of length t, where sequences are ranked using the scoring function f. The model uses a search-based loss function which is designed to incur loss only when the ground truth sequence falls off the beam. The search-based loss is defined as

$$\mathcal{L}(f) = \sum_{t=1}^{T} \Delta(\hat{\mathbf{y}}_{1:t}^{(K)}) \max(0, 1 - f(\mathbf{y}_t, \mathbf{h}_{t-1}, \mathbf{x}) + f(\hat{\mathbf{y}}_t^{(K)}, \hat{\mathbf{h}}_{t-1}^{(K)}, \mathbf{x})) \tag{3.10}$$

where $\mathbf{y}_t$ is the ground truth at time t and $\hat{\mathbf{y}}_t^{(K)}$ is the K-th ranked candidate sequence of length t on the beam. This loss function returns a value of 0 when the difference in score between the ground truth prefix $\mathrm{y}_{1:t}$ and $\hat{\mathrm{y}}_{1:t}^{(K)}$ does not violated a specified margin (assumed in Equation 3.10 to be 1), and otherwise returns a positive number which is weighted by the term $\Delta(\hat{\mathbf{y}}_{1:t}^{(K)})$. This term can incorporate any sequence-level measures, for example, $\Delta(\hat{\mathbf{y}}_{1:t}^{(K)}) = 1 - SentenceBLEU(\hat{\mathbf{y}}_{1:t}^{(K)}, \mathbf{y}_{1:t})$ can be used ($SentenceBLEU$ is the sentence-level BLEU score which lies between 0 and 1).

Optimising this search-based loss function during the training process enables the model to avoid label bias due to the absence of local normalisation, to be exposed to its own predictions, and to directly optimise sequence-level evaluation metrics.

3.2.5 Reward Augmented Maximum Likelihood

There are two broad categories of approaches to the sequence generation training process. One class of methods takes the supervised learning approach, i.e., using the ground truth sequence labels to maximise likelihood of the training data. The other class of methods uses reinforcement learning objectives, ignoring the target labels and using only the task reward. Techniques falling purely in the first category suffer from the problem of loss-evaluation mismatch, and also from exposure bias. Those purely in the second class of reinforcement learning objectives are impossible to train from scratch due to the reward being sparse in

the huge output space, leading to very slow learning. The process also does not consider the ground truth labels, which is inefficient. These methods usually have to be bootstrapped with a model trained to optimise perplexity. Most effective methods, including the ones discussed above, combine these two approaches in some way.

Reward-Augmented Maximum Likelihood (Norouzi et al., 2016) is one such hybrid method which incorporates the concept of reinforcement learning rewards, which can be sentence-level evaluation metrics, into the gradient descent-based parameter optimisation framework. The details of this technique are discussed in Section 4.1.

This project aims to implement the Reward-Augmented Maximum Likelihood method in an effort to improve the standard maximum likelihood training process. The next chapter explains the method, some details required for the implementation, and the software framework in which it will be implemented.

Chapter 4

Method and Implementation

This chapter discusses the technique of Reward Augmented Maximum Likelihood, the method chosen for implementation, in detail. Section 4.2 provides some extra details about how to sample auxiliary outputs that are required for the technique. Finally, Section 4.3 briefly describes the neural machine translation codebase Nematus, in which RAML has been introduced by this project.

4.1 Reward Augmented Maximum Likelihood (RAML)

Maximum likelihood training, as given in Equation 2.18, minimises[1] the following loss over a training data set $D = \{\langle \mathbf{x}^{(s)}, \mathbf{y}^{(s)} \rangle\}_{s=1}^{S}$

$$\mathcal{L}_{\mathrm{ML}}(\theta) = -\sum_{s=1}^{S} \log p_\theta(\mathbf{y}^{(s)}|\mathbf{x}^{(s)}) \tag{4.1}$$

Reward-Augmented Maximum Likelihood (Norouzi et al., 2016) modifies the loss function as

$$\mathcal{L}_{\mathrm{RAML}}(\theta; \tau) = \sum_{s=1}^{S} \left\{ -\sum_{y \in \mathcal{Y}} q(\mathbf{y}|\mathbf{y}^{(s)}; \tau) \log p_\theta(\mathbf{y}|\mathbf{x}^{(s)}) \right\} \tag{4.2}$$

where $\mathcal{Y}$ is the set of all possible translations. q is known as the *exponentiated payoff distribution* defined as

$$q(\mathbf{y}|\mathbf{y}^{(s)}; \tau) = \frac{\exp(r(\mathbf{y}, \mathbf{y}^{(s)})/\tau)}{\sum_{\mathbf{y}' \in \mathcal{Y}} \exp(r(\mathbf{y}', \mathbf{y}^{(s)})/\tau)} \tag{4.3}$$

[1] This was described as maximising the log likelihood in Equations 2.17 and 2.18, but here, the sign is changed to represent it as a negative log likelihood and show it as a cost function to be minimised.

with $r(\mathbf{y}, \mathbf{y}^{(s)})$ being a reward function or *payoff*, for which a sentence-level evaluation metric such as edit distance or BLEU is used, and τ being a temperature parameter of the distribution which controls the sharpness of the distribution around the ground truth $\mathbf{y}^{(s)}$.

To optimise the RAML objective, the gradient of the loss with respect to the model parameters is calculated as

$$\nabla_\theta \mathcal{L}_{\text{RAML}}(\theta; \tau) = \mathbb{E}_{q(\mathbf{y}|\mathbf{y}^{(s)};\tau)}[-\nabla_\theta \log p_\theta(\mathbf{y}|\mathbf{x}^{(s)})] \tag{4.4}$$

which is estimated by sampling $\mathbf{y}$. It is to be noted that for τ=0, this reduces to the standard maximum likelihood training.

To compute the gradient in such reward-based approaches, outputs have to be sampled, which introduces an additional computational cost to the training process. For a purely reinforcement learning based approach, output samples have to be drawn from the model distribution $p_\theta(\mathbf{y}|\mathbf{x})$. Since the sampling has to be done at the current state of the model, and the model distribution is evolving during the training process, it cannot be done in parallel with the training process. Moreover, sampling from the model naturally involves a forward pass through the encoder-decoder network to generate each sample. As a result, sampling from the model slows down the gradient computation. In contrast, for the RAML method, the sampling has to be done from the exponential payoff distribution. This distribution is stationary, and as a result, the sampling process does not have to refer to the model; instead, the sampling can be interpreted as a form of input processing and can be done in parallel to the training process.

RAML training therefore just adds a sampling step on top of the standard training process. For a particular source sentence in the training step, instead of optimising the negative log likelihood for the ground truth target sequence, sample outputs are drawn from the exponentiated payoff distribution centred around the ground truth, proportional to the exponentiated scaled reward. The mean log likelihood is optimised on these sampled outputs. Thus, RAML is able to combine the advantages of the relative computational efficiency of maximum likelihood training with the ability of reinforcement learning algorithms to directly optimise the evaluation metric.

4.2 Sampling from the Exponentiated Payoff Distribution

As mentioned in Section 4.1, the RAML model needs to sample output sequences from the exponentiated payoff distribution $q(\mathbf{y}|\mathbf{y}^{(s)};\tau)$. Since it is a stationary distribution, it is more efficient to sample from the exponentiated payoff distribution than to sample from the model, which is a non-stationary distribution.

Ideally, to eliminate the problem of loss-evaluation mismatch, the samples should be drawn from the distribution based on the very same metric that is used for final evaluation. However, it is not always tractable to directly draw samples from the distribution. If the reward function used is chosen to be Hamming distance or edit distance, it becomes possible to draw samples directly from the q distribution, as described in Sections 4.2.1 and 4.2.2. To use other reward value functions like BLEU, techniques like importance sampling can be used (Section 4.2.3).

4.2.1 Hamming Distance

Hamming distance is the number of positions where the tokens are not the same for two sentences. It is a simpler special case of edit distance, which is described in the next subsection.

Norouzi et al. (2016) used negative Hamming distance as the reward value to draw samples from the q distribution for their machine translation experiments. This is because using Hamming distance makes it relatively easy to draw exact samples from $q(\mathbf{y}|\mathbf{y}^{(s)};\tau)$.

To define the distribution from which the distance is sampled, the number of sentences with a certain Hamming distance value e is counted, the counts are reweighted by the exponentiated payoff $\exp(-e/\tau)$, and then the reweighted counts are normalised. For Hamming distance, it is not hard to see that the number of sentences at a distance of e from the unperturbed target of length n

is given by

$$c(e, n) = \binom{n}{e}(v-1)^e \tag{4.5}$$

The first combinatorial term is the number of ways e positions can be chosen to perturb from the sequence of length n. Each chosen position can then be substituted by any one of $v-1$ words, where v is the size of the target vocabulary. The distribution is thus defined by

$$p(e) = \begin{cases} \frac{\binom{n}{e}(v-1)^e \cdot \exp(-e/\tau)}{\sum_{e'=1}^{n}\binom{n}{e'}(v-1)^{e'} \cdot \exp(-e'/\tau)}, & e \in \{0, 1, \ldots, n\} \\ 0, & otherwise \end{cases}$$

Samples can be drawn from the exponentiated payoff distribution by stratified sampling. First, a Hamming distance reward value, say e, is drawn from the distribution, and then an output sequence with that Hamming distance with respect to $\mathbf{y}^{(s)}$ is obtained by substituting e words in the ground truth sequence with random words from the target vocabulary.

4.2.2 Edit Distance

Sampling can be done according to edit distance instead of Hamming distance in a similar way. We calculate the counts of sentences, reweight the counts by $\exp(-e/\tau)$, and normalise to get the distribution over reward values. However, calculating the counts in this case is somewhat more complicated than in the case of Hamming distance.

When $e = 1$, there are $n+1$ possible positions for insertions, for each of which there are v word options (v is the size of the vocabulary). There are n positions (n is the length of the target sequence) for substitution, with $v-1$ options for each of these. However, deletion can be thought of as a substitution with an empty token, so there are v options for substitution as well.

When $e > 1$, let the number of substitutions be s, so the number of insertions is $e-s$. Tokens lose their significance once they are substituted, so insertions before and after substituted tokens are merged. Thus, there are n possible positions for substitutions, and $m-s$ positions for insertions. The counts can be calculated

by enumerating over the number of substitutions, and are given by Norouzi et al. (2016) as the following

$$c(e,n) = \begin{cases} \sum_{s=0}^{n} \binom{n}{s} \binom{n+e-2s}{e-s} v^e, & e > 1 \\ (2n+1)v, & e = 1 \end{cases}$$

It is to be noted that these counts are approximate, since edge cases like word repetitions are ignored. However, these counts are good enough for sampling. While edit distance can be arbitrarily high, the implementation only considers sentences with a maximum edit distance of n.

4.2.3 BLEU Score

It is not tractable to sample directly from the exponentiated payoff distribution based on BLEU scores or any other arbitrary reward function. Importance sampling can be used to sample according to BLEU score, using Hamming distance or edit distance as the proposal distribution. After samples have been drawn according to Hamming or edit distance, an importance correction is performed by reweighting the samples according to their BLEU scores. The samples are reweighted by $\exp(BLEU(\mathbf{y}, \mathbf{y}^{(s)})/\tau)/exp(-e/\tau)$.

These are the three reward functions that have been used for sampling in this project. Any other arbitrary evaluation can also be used by importance sampling.

4.3 Nematus

Nematus[2] (Sennrich et al., 2017) is an open-source Python Theano-based neural machine translation toolkit. It uses an attention-based encoder-decoder model based on Bahdanau et al. (2014), which is the same as described in Sections 2.1.1 and 2.1.2, with a few differences in the implementation details. The details of these differences are described in Sennrich et al. (2017). Some of the important ones are as follows (based on my IRP report).

- While Bahdanau et al. (2014) initialised the decoder hidden state with the final backward encoder state, i.e. $\overleftarrow{\mathbf{h}}_1$. In contrast, Nematus initialises the

[2] `https://github.com/EdinburghNLP/nematus`

decoder with the mean of all the source annotations, i.e., $\frac{\sum_{m=1}^{M} \mathbf{h}_m}{M}$.

- Bahdanau et al. (2014) used a feed-forward neural network from the RNN state with a tanh non-linearity followed by a softmax to generate outputs. Nematus uses the previous word and the context vector along with the RNN state as inputs to the network to form an intermediate representation before applying the softmax.

- The decoder uses a novel variation of RNN known as "conditional GRU with attention" or cGRU_{att}. In addition to its own hidden state and the previous step output, a cGRU_{att} also uses all the source annotations to update its state (see Sennrich et al., 2017, for details). This basically incorporates the attention mechanism into the internal update of the decoder. The cGRU_{att} has two transition blocks with the attention mechanism in between. The first transition generates an intermediate representation $\mathbf{s}'_j$ from the previous word and the previous hidden state. The attention mechanism uses the entire set of source annotations along with $\mathbf{s}'_j$ to generate the context vector. Finally, the second transition block generates $\mathbf{s}_j$ from $\mathbf{s}'_j$ and the context vector.

The details and update equations are discussed in detail by Sennrich et al. (2017) and some of these have also been summarised in Appendix A for reference.

Nematus also has an option to use standard cross-entropy minimisation training, or to use minimum risk training (Shen et al., 2016). RAML training has been added[3] into the framework as an option for the training objective by this project, so that all three types of training can be compared while keeping all other implementation details constant (see Section 5.4).

[3] `https://github.com/Proyag/nematus`

Chapter 5

Experiments and Results

This chapter contains descriptions of the experiments that were designed and run to test the effectiveness of the RAML technique against some baseline systems trained by existing methods. Section 5.1 describes the data on which all models will be trained. Section 5.2 contains the details of the two baseline models that results will be compared against. Section 5.3 contains a description of the set of experiments that were run along with the rationale behind choosing these experiments. Section 5.4 shows the results obtained from these experiments, and how they compare with the baselines.

5.1 The Data

5.1.1 Datasets

The data used for training the systems was from the WMT17 news translation task[1]. The systems were trained to translate German→English. The training set for the German→English task consists of data from the Europarl v7 corpus (Koehn, 2005), the Common Crawl corpus, the News Commentary v12 corpus, and the Rapid corpus of EU press releases. The validation set[2] consists of the test sets from the news translation tasks of WMT14, WMT15, and WMT16, and the test set[3] is from the same task for WMT17.

[1] `http://data.statmt.org/wmt17/translation-task/`
[2] `http://data.statmt.org/wmt17/translation-task/dev.tgz`
[3] `http://data.statmt.org/wmt17/translation-task/test.tgz`

5.1.2 Preprocessing and Postprocessing

Preprocessing

There is a preprocessed version of the data available on the website for the WMT17 data[4]. In this dataset, the following preprocessing steps have already been applied (using scripts from the Moses machine translation toolkit (Koehn et al., 2007)) on the data from the corpora mentioned in the previous section.

- Punctuation is normalised[5] and the text is tokenised[6].
- The data is cleaned[7] to remove empty lines, remove redundant spaces, and to drop sentence pairs that are empty, too short, or too long.
- Truecasing is applied[8]. Instead of lowercasing all text, truecasing retains the natural casing of the text, and only converts words at the beginning of sentences to their most common form.

In addition to the above, another preprocessing step has been applied for experiments in this project, which is the byte pair encoding technique to split rare and unknown words into sequences of subword units (Sennrich et al., 2015)[9]. 89500 BPE merge operations were used.

Postprocessing

Generated translation are postprocessed by the following steps

- Subword tokens created by the BPE technique used for rare and unknown words are concatenated using the simple Bash command `sed 's/\@\@//g'`.
- Truecasing is undone[10] to capitalise first words of sentences.

[4] `http://data.statmt.org/wmt17/translation-task/preprocessed`
[5] `https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/normalize-punctuation.perl`
[6] `https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/tokenizer.perl`
[7] `https://github.com/moses-smt/mosesdecoder/blob/master/scripts/training/clean-corpus-n-ratio.perl`
[8] `https://github.com/moses-smt/mosesdecoder/blob/master/scripts/recaser/train-recaser.perl` and `https://github.com/moses-smt/mosesdecoder/blob/master/scripts/recaser/truecase.perl`
[9] `https://github.com/rsennrich/subword-nmt`
[10] `https://github.com/moses-smt/mosesdecoder/blob/master/scripts/recaser/detruecase.perl`

- Final translations are detokenised[11]. However, evaluation is done in the tokenised form.

5.2 Baselines

5.2.1 Cross-entropy Training Baseline

Since the aim of this project is to explore a technique that attempts to overcome flaws in cross-entropy training, the natural baseline to compare against would be a model trained to optimise cross-entropy. The exact settings used to train the model in Nematus are listed in Appendix B. The model was trained till the BLEU score on the held-out validation set stopped improving, which took between 2-3 weeks on one M60 GPU.

CE Baseline The BLEU score obtained on the test set is 28.92. This is the baseline BLEU score that the experiments on RAML seek to improve upon.

5.2.2 Minimum Risk Training Baseline

Minimum Risk Training (Shen et al., 2016), as described in Section 3.2.1, is another technique that uses a different approach to mitigate the problems in cross-entropy training. This training objective was already present as an option in Nematus, and this is our second baseline system. All the settings in Nematus were the same as for the cross-entropy baselines, except those related specifically to MRT. The hyperparameters of MRT are the number of output samples drawn per input sequence (`mrt_samples`), which is set to 70, and the `mrt_alpha` parameter, set to 0.005. 10 independent samples are drawn (`mrt_samples_meanloss`) to calculate mean loss, which is subtracted from the loss for each sentence. The loss used for MRT (`mrt_loss`) is SentenceBLEU with n=4.

MRT Baseline The test set BLEU score obtained by MRT is 29.26.

[11] `https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/detokenizer.perl`

5.3 Experiment Design

Experiments have been designed in order to compare the effectiveness of RAML against cross-entropy training and MRT. In order to do this, all parameters have been kept unchanged except the hyperparameters pertaining specifically to RAML. These hyperparameters are the following

- `raml_tau`: The τ parameter of the exponentiated payoff distribution
- `raml_samples`: The number of auxiliary samples drawn from the exponentiated payoff distribution per output sequence.
- `raml_reward`: The reward value used for RAML - Hamming distance, edit distance, or BLEU score.

There are three main sets of experiments for the three reward functions. For each reward function, a range of values of τ are tested, and the number of samples drawn per output sequence is usually restricted to 1, 2, or 3, since drawing a large number of samples for the same sequence slows down learning. It is to be noted that this is not because the sampling process is computationally expensive, it is only due to the fact that the training process has to go through more examples to observe all the original training pairs. The full set of experiments, along with the results observed, and a comparison of the RAML results with the baselines (Table 5.4) is shown in Section 5.4.

5.4 Results

When the reward value being used is Hamming distance, most of the experiments draw a single sample, as this is the setting used by Norouzi et al. (2016) in their experiments. Since good results are observed around $\tau = 0.85$ and $\tau = 0.90$ (refer to Table 5.1), an experiment with two samples per output sequence with $\tau = 0.85$ was also run for comparison. Table 5.1 shows the BLEU scores obtained for these experiments.

When sampling according to SentenceBLEU, since samples are actually drawn using Hamming distance and importance correction weights are applied, drawing a single sample would result in similar samples as with Hamming distance, only

with different sample weights. Therefore, for BLEU, only experiments with more than 1 sample per output sequence are conducted.

Tables 5.2 and 5.3 respectively show the BLEU scores observed when edit distance and SentenceBLEU are used as the reward function. Table 5.4 shows an overview of the results, comparing the best observed results obtained by RAML training with the baseline BLEU scores.

Value of τ	Number of samples	Best BLEU score
0.70	1	29.32
0.75		28.93
0.80		28.90
0.85		29.44
0.90		29.70
0.95		29.99
1.0		29.38
0.85	2	29.58

Table 5.1: BLEU scores for experiments using Hamming distance as `raml_reward`

Value of τ	Number of samples	Best BLEU score
0.75	1	29.23
0.80		29.37
0.85		29.48
0.90		29.41
0.75	2	29.66
0.80		29.51
0.85		29.51
0.90		28.80

Table 5.2: BLEU scores for experiments using edit distance as `raml_reward`

Value of τ	Number of samples	Best BLEU score
0.75	2	29.05
0.80		29.65
0.85		29.52
0.90		29.63
0.75	3	29.06
0.80		29.27
0.85		29.51
0.90		28.05

Table 5.3: BLEU scores for experiments using SentenceBLEU as `raml_reward`

Training Objective	Best BLEU score
Cross-entropy baseline	28.92
MRT	29.26
RAML	29.99

Table 5.4: Comparison of best RAML results with the baselines

Since our primary consideration was to eliminate the discrepancy between the loss function and the evaluation metric, it might seem a little surprising to see better results when Hamming distance is used as a reward value. However, this is due to the fact that it is not possible to directly sample the auxiliary outputs according to BLEU scores. The sampling, as explained in Section 4.2, is done according to Hamming distance, and importance weight corrections are applied on the samples. Since a small number of samples are being drawn, it is possible that this is not a very accurate approximation of an exponentiated payoff distribution actually constructed according to BLEU reward values. A larger number of samples would help to better approximate the true distribution, possibly leading to higher scores.

RAML is thus observed to achieve higher BLEU scores on the test set compared to the baselines. It is seen that just a single sample drawn per output sequence during RAML optimisation is enough to improve translation quality. Among the reward functions, Hamming distance yields the best results, which is the same

reward function used by Norouzi et al. (2016) in their experiments. At best, when Hamming distance is used for reward values with $\tau = 0.95$, an improvement of 1.07 BLEU points over the perplexity-trained model is obtained, which is quite significant. The authors of Norouzi et al. (2016) reported an increase of about 0.4 BLEU points in their experiments, so the improvement in results is better than expected.

Chapter 6

Conclusion

This project has successfully implemented and explored a recent technique to improve neural machine translation training, namely, Reward Augmented Maximum Likelihood (RAML). This is one of a number of methods that have been proposed to incorporate reinforcement learning-like task rewards directly into the optimisation process. This technique is computationally efficient, does not require bootstrapping with a cross-entropy trained model, and effectively only requires the systematic augmentation of the output targets used for training. Once the outputs are augmented, the training can be performed within the traditional cross-entropy training framework.

The RAML training objective has been implemented in the Nematus neural machine translation framework, where it has been compared against both cross-entropy training and minimum risk training. While minimum risk training also uses a different method to directly optimise the evaluation metric, experiments performed with MRT have yielded a smaller improvement over the cross-entropy baseline. The implementation of RAML has shown an improvement of 1.07 BLEU points over the cross-entropy baseline, which is quite significant.

The RAML technique can also be applied to other probabilistic learning models with arbitrary task rewards. For machine translation, many of the other improved training techniques described in Section 3.2 still remain to be comprehensively explored.

Appendix A

Nematus: Details

A.1 Initialising the Decoder

As previously touched upon in Sections 2.1.1 and 4.3, Bahdanau et al. (2014) initialised the decoder RNN state with a feed-forward neural network from the final backward hidden state of the bidirectional RNN encoder, i.e.,

$$\mathbf{s}_0 = \tanh\left(W_{init}\overleftarrow{\mathbf{h}}_1\right) \tag{A.1}$$

In Nematus, in contrast, Sennrich et al. (2017) initialise the decoder hidden state with the mean of all the encoder annotations, i.e.,

$$\mathbf{s}_0 = \tanh\left(W_{init}\frac{\sum_{m=1}^{M}\mathbf{h}_m}{M}\right) \tag{A.2}$$

A.2 The Conditional GRU with Attention (cGRU_{att})

The "conditional GRU with attention" or the cGRU_{att}, as mentioned before in Section 4.3, consists of two state transition blocks with an attention mechanism in between. All the equations below are supplied for reference from Sennrich et al. (2017).

The first transition block GRU_1, uses the previous word $\mathbf{y}_{j-1}$ and the previous hidden state $\mathbf{s}_{j-1}$ to generate an intermediate representation $\mathbf{s}'_j$

$$\mathbf{s}'_j = \text{GRU}_1(\mathbf{y}_{j-1}, \mathbf{s}_{j-1}) \tag{A.3}$$

$$= (1 - \mathbf{z}'_j) \odot \underline{\mathbf{s}}'_j + \mathbf{z}'_j \odot \mathbf{s}_{j-1} \tag{A.4}$$

where

$$\underline{\mathbf{s}}'_j = \tanh\left(W'E[\mathbf{y}_{j-1}] + \mathbf{r}'_j \odot (U'\mathbf{s}_{j-1})\right) \tag{A.5}$$
$$\mathbf{r}'_j = \sigma(W'_r E[\mathbf{y}_{j-1}] + U'_r \mathbf{s}_{j-1}) \tag{A.6}$$
$$\mathbf{z}'_j = \sigma(W'_z E[\mathbf{y}_{j-1}] + U'_z \mathbf{s}_{j-1}) \tag{A.7}$$

where E is an embedding matrix (so $E[\mathbf{y}_i]$ gives the embedding of the word $\mathbf{y}_i$), $\mathbf{r}'_j$ and $\mathbf{z}'_j$ are the activations for the reset and update gates, and all the W and U matrices are trainable parameters of the model.

The attention mechanism component of the cGRU_{att} then uses the intermediate representation $\mathbf{s}'_j$ along with the entire set of source annotations $\{\mathbf{h}_1, \ldots, \mathbf{h}_M\}$ to construct the context vector $\mathbf{c}_j$.

$$\mathbf{c}_j = \sum_{i=1}^{M} \alpha_{ij}\mathbf{h}_i \tag{A.8}$$
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{M}\exp(e_{kj})} \tag{A.9}$$
$$e_{ij} = \mathbf{v}'_a \tanh(U_a\mathbf{s}'_j + W_a\mathbf{h}_i) \tag{A.10}$$

where $\mathbf{v}'_a$, U_a, and W_a are trainable parameters.

Finally, the second transition block uses the context vector $\mathbf{c}_j$ along with the intermediate representation $\mathbf{s}'_j$ to generate the new hidden state $\mathbf{s}_j$ as

$$\mathbf{s}_j = \text{GRU}_2(\mathbf{s}'_j, \mathbf{c}_j) \tag{A.11}$$
$$= (1 - \mathbf{z}_j) \odot \underline{\mathbf{s}}_j + \mathbf{z}_j \odot \mathbf{s}'_j \tag{A.12}$$

where

$$\underline{\mathbf{s}}_j = \tanh\left(W\mathbf{c}_j + \mathbf{r}_j \odot (U\mathbf{s}'_j)\right) \tag{A.13}$$
$$\mathbf{r}_j = \sigma(W_r\mathbf{c}_j + U_r\mathbf{s}'_j) \tag{A.14}$$
$$\mathbf{z}_j = \sigma(W_z\mathbf{c}_j + U_z\mathbf{s}'_j) \tag{A.15}$$

where, similar to the GRU_1, $\mathbf{r}_j$ and $\mathbf{z}_j$ are the reset and update gate activations, and all the W and U matrices are trainable model parameters.

A.3 The Output Layers

As said in Section 4.3, the decoder uses $\mathbf{s}_j$, $\mathbf{y}_{j-1}$, and $\mathbf{c}_j$ to generate the output probability distribution. This is done as follows:

$$p(\mathbf{y}_j|\mathbf{s}_j, \mathbf{y}_{j-1}, \mathbf{c}_j) = \text{softmax}(\mathbf{t}_j W_o) \tag{A.16}$$

$$\mathbf{t}_j = \tanh(\mathbf{s}_j W_{t1} + E[\mathbf{y}_{j-1}] W_{t2} + \mathbf{c}_j W_{t3}) \tag{A.17}$$

where W_{t1}, W_{t2}, W_{t3}, and W_o are trainable parameters of the model.

Appendix B

Training Settings for Experiments

Table B.1 contains the parameters given to the `nmt.train()` function of Nematus to train the models used for the project. Only the `objective` parameter needs to be changed to select cross-entropy (CE), minimum risk training (MRT) or RAML. Apart from that, the hyperparameters specific to RAML have been tweaked for different experiments on RAML training. Many parameters which were left as the default values have not been shown here. Only the parameters which are particularly important to understand the network architecture or the ones which have been changed from their defaults are shown.

Parameter	Value	Description
dim_word	500	Size of word embeddings
dim	1024	Number of LSTM units
enc_depth	1	Number of encoder layers
dec_depth	1	Number of decoder layers
encoder	gru	GRU used in encoder
decoder	gru_cond	cGRU_{att} used in decoder
clip_c	-1	Gradient clipping threshold
optimizer	adam	The Adam optimiser is used
lrate	0.0001	Learning rate
n_words_src	90000	Source vocabulary size
n_words	90000	Target vocabulary size
maxlen	50	Maximum sentence length
batch_size	80	Batch size
valid_batch_size	80	Batch size for validation

Parameter	Value	Description
shuffle_each_epoch	True	Shuffle data for every training epoch
objective	CE	Training objective
mrt_alpha	0.005	α parameter for MRT
mrt_samples	70	Number of samples per input
mrt_reference	False	Add ground truth to set of samples
mrt_loss	SENTENCEBLEU n=4	Loss function for MRT
raml_tau	0.95	$\tau \in (0, 1]$, $\tau = 0$ is same as CE
raml_samples	1	Number of samples per output
raml_reward	hamming_distance	Reward function for RAML

Table B.1: Nematus training parameters

Bibliography

P. Koehn, *Statistical Machine Translation*, 1st ed. Cambridge University Press, 2010.

P. F. Brown, V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer, "The mathematics of statistical machine translation: Parameter estimation," *Computational linguistics*, vol. 19, no. 2, pp. 263–311, 1993.

F. J. Och and H. Weber, "Improving statistical natural language translation with categories and rules," in *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 2*. Association for Computational Linguistics, 1998, pp. 985–989.

F. J. Och, C. Tillmann, H. Ney *et al.*, "Improved alignment models for statistical machine translation," in *Proc. of the Joint SIGDAT Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999, pp. 20–28.

Y.-Y. Wang and A. Waibel, "Modeling with structures in statistical machine translation," in *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 2*. Association for Computational Linguistics, 1998, pp. 1357–1363.

D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014. [Online]. Available: https://arxiv.org/pdf/1409.0473.pdf

K. Cho, B. van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN

encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. [Online]. Available: http://www.aclweb.org/anthology/D14-1179

I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

L. Bentivogli, A. Bisazza, M. Cettolo, and M. Federico, "Neural versus phrase-based machine translation quality: A case study," *CoRR*, vol. abs/1608.04631, 2016. [Online]. Available: http://arxiv.org/pdf/1608.04631.pdf

M. Junczys-Dowmunt, T. Dwojak, and H. Hoang, "Is neural machine translation ready for deployment? A case study on 30 translation directions," *arXiv preprint arXiv:1610.01108*, 2016. [Online]. Available: http://arxiv.org/pdf/1610.01108.pdf

M. Ranzato, S. Chopra, M. Auli, and W. Zaremba, "Sequence level training with recurrent neural networks," *arXiv preprint arXiv:1511.06732*, 2015. [Online]. Available: https://arxiv.org/pdf/1511.06732.pdf

J. Lafferty, A. McCallum, F. Pereira *et al.*, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proceedings of the eighteenth international conference on machine learning, ICML*, vol. 1, 2001, pp. 282–289.

M. Norouzi, S. Bengio, N. Jaitly, M. Schuster, Y. Wu, D. Schuurmans *et al.*, "Reward augmented maximum likelihood for neural structured prediction," in *Advances In Neural Information Processing Systems*, 2016, pp. 1723–1731.

L. Chrisman, "Learning recursive distributed representations for holistic computation," *Connection Science*, vol. 3, no. 4, pp. 345–366, 1991.

M. L. Forcada and R. P. Ñeco, "Recursive hetero-associative memories for translation," *Biological And Artificial Computation: From Neuroscience To Technology*, vol. 1240, pp. 453–462, 1997.

N. Kalchbrenner and P. Blunsom, "Recurrent continuous translation models." in *EMNLP*, vol. 3, no. 39, 2013, p. 413.

N. Kalchbrenner and P. Blunsom, "Recurrent convolutional neural networks for discourse compositionality," *Proceedings of the 2013 Workshop on Continuous Vector Space Models and their Compositionality*, 2013.

T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *Interspeech*, vol. 2, 2010, p. 3.

J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, March 1990.

Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Trans. Neur. Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994. [Online]. Available: http://dx.doi.org/10.1109/72.279181

S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

F. A. Gers, J. A. Schmidhuber, and F. A. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Comput.*, vol. 12, no. 10, pp. 2451–2471, Oct. 2000. [Online]. Available: http://dx.doi.org/10.1162/089976600300015015

F. A. Gers and J. Schmidhuber, "Recurrent nets that time and count," in *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, vol. 3. IEEE, 2000, pp. 189–194.

K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE transactions on neural networks and learning systems*, 2016.

J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014. [Online]. Available: https://arxiv.org/pdf/1412.3555.pdf

M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015. [Online]. Available: https://arxiv.org/pdf/1508.04025.pdf

D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. [Online]. Available: https://arxiv.org/pdf/1412.6980.pdf

M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012. [Online]. Available: https://arxiv.org/pdf/1212.5701.pdf

R. Williams, D. Rumelhart, and G. Hinton, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–538, 1986.

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

B. Chen and C. Cherry, "A systematic comparison of smoothing techniques for sentence-level BLEU." in *WMT@ ACL*, 2014, pp. 362–367.

G. Doddington, "Automatic evaluation of machine translation quality using n-gram co-occurrence statistics," in *Proceedings of the Second International Conference on Human Language Technology Research*, ser. HLT '02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 138–145. [Online]. Available: http://dl.acm.org/citation.cfm?id=1289189.1289273

M. Denkowski and A. Lavie, "Meteor universal: Language specific translation evaluation for any target language," in *Proceedings of the EACL 2014 Workshop on Statistical Machine Translation*, 2014.

S. Wiseman and A. M. Rush, "Sequence-to-sequence learning as beam-search optimization," *arXiv preprint arXiv:1606.02960*, 2016. [Online]. Available: https://arxiv.org/pdf/1606.02960.pdf

S. Shen, Y. Cheng, Z. He, W. He, H. Wu, M. Sun, and Y. Liu, "Minimum risk training for neural machine translation," in *Proceedings of the 54th Annual*

Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, 2016, pp. 1683–1692. [Online]. Available: http://aclweb.org/anthology/P16-1159

R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

D. Andor, C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and M. Collins, "Globally normalized transition-based neural networks," *arXiv preprint arXiv:1603.06042*, 2016. [Online]. Available: https://arxiv.org/pdf/1603.06042.pdf

R. Sennrich, O. Firat, K. Cho, A. Birch, B. Haddow, J. Hitschler, M. Junczys-Dowmunt, S. Läubli, A. V. Miceli Barone, J. Mokry, and M. Nadejde, "Nematus: a toolkit for neural machine translation," in *Proceedings of the Software Demonstrations of the 15th Conference of the European Chapter of the Association for Computational Linguistics*. Valencia, Spain: Association for Computational Linguistics, April 2017, pp. 65–68. [Online]. Available: http://aclweb.org/anthology/E17-3017

P. Koehn, "Europarl: A parallel corpus for statistical machine translation," in *MT summit*, vol. 5, 2005, pp. 79–86.

P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens *et al.*, "Moses: Open source toolkit for statistical machine translation," in *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*. Association for Computational Linguistics, 2007, pp. 177–180.

R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *arXiv preprint arXiv:1508.07909*, 2015. [Online]. Available: https://arxiv.org/pdf/1508.07909.pdf