UNIVERSITY OF AMSTERDAM

A thesis submitted in partial fulfilment for the degree of Master of Science
in the science of Artificial Intelligence

Tuning Methods in Statistical Machine Translation

Author:
Anne Gerard Schuth
aschuth@science.uva.nl
0418900

Supervisor:
Dr. Christof Monz
c.monz@uva.nl

September 1, 2010

Abstract

In a Statistical Machine Translation system many models, called features, complement each other in producing natural language translations. In how far we should rely on a certain feature is governed by parameters, or weights. Learning these weights is the sub field of SMT, called parameter tuning, that is addressed in this thesis. Three existing methods for learning such parameters are compared. We recast MERT, MIRA and Downhill Simplex in a uniform framework, to allow for easy and consistent comparison. Based on our findings and forthcoming opportunities for improvements, we introduce two new methods. A straightforward sampling approach, Local Unimodal Sampling (LUS), that uniformly samples from a decreasing area around a constantly updated peak in weight-vector space. And a ranking based approach, implementing SVM-Rank, that focusses on giving, besides the best translations, also its runner-ups a high score. We empirically compare our own methods to existing methods and find that LUS slightly, but significantly, outperforms the state-of-the-art MERT method in a realistic setting with 14 features. We claim that this progress, the simplicity of the radically different approach of the method obtaining this progress and the clear overview of existing work are contributions to the field. Our SVM-Rank showed no improvement over the-state-of-the-art within our experimental setup.

Acknowledgments

First and foremost, I would like to thank my supervisor dr. Christof Monz. His enthusiasm got me intrigued by Statistical Machine Learning and led to our early discussions on the subject of this thesis about a year ago. His ongoing support during the countless consultations were very valuable for me. Without his always justified critique on my research and writing style, this thesis would not have been what it is today.

I would like to thank dr.ir. Leo Dorst and dr. Khalil Sima'an for being a member of my graduation committee.

Nina Peters, Hylke Buisman and Wout Schuth are thanked for their much valued effort of going through early drafts of this thesis. If it became readable, it is largely due to them.

The ILPS-group is thanked for allowing me to occupy a desk and joining them during lunches. Simon Carter in particular helped me during various occasions when I got stuck in one of the many pitfalls that come with research in Statistical Machine Translation.

I also thank SARA Computing and Networking Services for their support in using the Lisa Compute Cluster. Without the access to this cluster my experiments would not have been as elaborate and conclusive as they are now.

Since this thesis marks the end of my studies, I would hereby like to thank my parents for their support —in the broadest sense of the word— throughout the long years.

Anne Gerard Schuth, *September 1, 2010*

Contents

Chapter 1

Introduction

With the growing abundance of information that is coming upon us since the rise of the Internet and globalisation, many of us have become used to a huge information intake. We are more and more capable of finding our way through fast quantities of data. We do not so much posses knowledge, we know where to look. Search engines come to our aid. One issue remains and surfaces more and more. Not everything we might want to know is available in our own language[1], we will need translations of what we can not read. Given the enormous amounts of text that needs translation we need this to be done automatically, by a machine; we need Machine Translation (Hutchins, 2003). The information explosion forces us to be aided by machines that produce at least rough translations. For the English speaking part of the world this need might not be as clear as it is for others; for others it can be a vital means of staying in contact with the rest of the world. Very well demonstrated by automated translations of international news-sources into Farsi during the rallies around the latest Iranian elections; translations were not perfect but allowed Iranians to follow public opinion abroad.

The process of translating in an automated way has been studied extensively for at least the past couple of decades and the research field made rapid progress. But still the state-of-the-art can not compete with a human, professional, translator. This work will not try to close that gap, but will study some aspects of the shortcomings of current approaches of others and propose two approaches that might not suffer from these shortcomings. But above all, the aim of this work is to provide insight into an important sub field of the statistical variant of machine translation. This work will study parameter optimisation methods, which is called tuning, in Statistical Machine Translation (SMT).

Current SMT systems, Phrase-Based SMT systems, use a log-linear model to estimate the probability of translations of a sentence. Such a system prefers, and returns, the most probable translation. This model is parametrized, and tuning is the process of finding the optimal set of parameters, given some metric, for this log-linear model. Some approaches exist, of which MERT is the most prominent. We compare three existing ways of tuning and introduce 2 new approaches. A straightforward sampling approach, Local Unimodal Sampling (LUS), that

[1]It is shown by Gerrand (2007) that the use of English on the world wide web has increased, but much less so than the use of other languages; more and more information appears in a variety of languages.

uniformly samples from a decreasing area around a constantly updated peak in weight-vector space. And a ranking based approach, implementing SVM-Rank, that focusses on giving, besides the best translations, also its runner ups a high score.

1.1 Reading Guide

This first chapter and Chapter 3 (p. 10) will provide a high level introduction into Machine Translation, and more specifically Statistical Machine Translation. We will quickly dive into the problems related to tuning such a SMT system. Anyone familiar with SMT can readily skip the following sections but should briefly read Chapter 3 (p. 10) to become acquainted with the notation used, and definitely section 3.4 (p. 19) stating the difficulties of the problem we address. The end of this chapter, section 1.3 (p. 4), will state the research questions that are to be answered in the remaining chapters. These research questions are followed directly by section 1.4 (p. 5) that lists the contributions of this work.

The next chapter, Chapter 2 (p. 7), stands on its own and describes a variety of approaches by others that tackle the same, or similar, issues we address in this work.

Chapter 4 (p. 20) will introduce and describe three existing approaches to Tuning in detail. The chapter is intended to be self-contained in the sense that it can be read without reading referenced work.

Chapter 5 (p. 32) compares the three approaches described in the preceding chapter on a theoretical basis. The findings of this comparison are used in Chapter 6 (p. 39) to motivate two, to our knowledge, new ways of performing the required optimisation. Together these chapters form heart of this thesis.

The before last Chapter 7 (p. 46) describes an empirical evaluation and comparison. We analyse where and why results agree or disagree with our theoretical comparison and motivation to introduce our new methods. Conclusions are drawn and described in Chapter 8 (p. 68).

1.2 Machine Translation

1.2.1 Rule Based Machine Translation

Early approaches to automatic translation took the route of transforming the source language into some semantic representation; an interlingua. This interlingua was, as the name suggests, completely language independent; it would describe real-world concepts in some logic representation. This interlingual form of the meaning of the source sentence could be transformed into any target language. All transformations mentioned here would be directed by rules. This approach has largely been abandoned since it turned out that it was impractical for anything but some small fixed toy domains. The availability of both data and faster computers led to the rise of a statistical approach instead.

1.2.2 Statistical Machine Translation

Already in the late forties of the previous century Weaver (1949) coined a statistical approach to Machine Translation. Back then the approach never really took off, due to both limiting factors in computing power and the lacking availability of example data. Only in the nineties researchers at the IBM Thomas J. Watson Research Center took up the old work (Brown et al., 1988, 1990, 1993). Since then the field of Statistical Machine Translation (SMT) became very active. In the early years of this century, new and leading approaches included Phrase-Based SMT (PBSMT) (Zens et al., 2002; Koehn et al., 2003; Och and Ney, 2003). This thesis should be placed in that line of work.[2]

SMT systems, in essence, are based on the Noisy Channel Model, by Shannon (1948), rewritten using the Bayes rule:

$$\hat{\mathbf{e}} = \operatorname*{argmax}_{\mathbf{e}} p(\mathbf{e}|\mathbf{f}) = \operatorname*{argmax}_{\mathbf{e}} p(\mathbf{f}|\mathbf{e})p(\mathbf{e}) \tag{1.1}$$

It states that the best translation $\hat{\mathbf{e}}$ (english) of a sentence $\mathbf{f}$ (foreign) is the sentence $\mathbf{e}$ that maximises $p(\mathbf{e}|\mathbf{f})$.

Loosely speaking and without going into details, this approach allows us to have a model that preserves meaning, $p(\mathbf{f}|\mathbf{e})$, separate from a model that controls the quality of the english sentence, $p(\mathbf{e})$. The first model is the *translation model* and in a PBSMT system it is implemented using a so called *phrase table* consisting of paired phrases and their probabilities estimated from a bilingual corpus[3]. The second model is called the *language model*, and is generally implemented using n-grams, whose probabilities are estimated using a monolingual corpus[4] of the target language.

A more general form of (1.1), that will allow us to have more than just the two models mentioned —the translation model and the language model—, can be derived as follows:

$$\begin{aligned}
\hat{\mathbf{e}} = \operatorname*{argmax}_{\mathbf{e}} p(\mathbf{e}|\mathbf{f}) &= \operatorname*{argmax}_{\mathbf{e}} p(\mathbf{f}|\mathbf{e})p(\mathbf{e}) && (1.2)\\
&= \operatorname*{argmax}_{\mathbf{e}} \prod_i h_i(\mathbf{e},\mathbf{f})^{\mathbf{w}_i} && (1.3)\\
&= \operatorname*{argmax}_{\mathbf{e}} \sum_i \log(h_i(\mathbf{e},\mathbf{f}))\cdot\mathbf{w}_i && (1.4)\\
&= \operatorname*{argmax}_{\mathbf{e}} \sum_i \Psi_i(\mathbf{e},\mathbf{f})\cdot\mathbf{w}_i && (1.5)\\
&= \operatorname*{argmax}_{\mathbf{e}} \Psi(\mathbf{e},\mathbf{f})\cdot\mathbf{w} && (1.6)
\end{aligned}$$

Here, h_i in (1.3) and (1.4) denotes a so-called *feature* with it's corresponding *weight* $\mathbf{w}_i$. So strictly speaking, $h_1(\mathbf{e},\mathbf{f})$ refers to $p(\mathbf{f}|\mathbf{e})$ and $h_2(\mathbf{e},\mathbf{f})$ to $p(\mathbf{e})$. This setup however allows for extensions by adding more models —now called features— that can each capture other aspects of translation. Each feature is

[2]In the remainder of our work, when we refer to SMT, we are actually referring to PBSMT.

[3]Bilingual corpora, also called parallel corpora or simply bi-texts, are large bodies of sentence by sentence translations.

[4]A monolingual corpus is usually an even larger corpus than the bi-text, and consists of a representative sample of the target language.

parametrized with weight $\mathbf{w}_i$ as an exponent, if a feature was simply multiplied by its weight, weights would be no longer specific to a feature due to commutativity of multiplication.

In existing implementations of so-called decoders —systems that perform the mentioned argmax search operation— such as Moses (Koehn et al., 2007), the translation model alone uses by default already five features. Add to that a feature for the language model and one for a reordering model (Zens and Ney, 2003), to name a few[5].

For reasons of convenience in later chapters, we rewrite (1.1) even more, into the log-linear form of (1.6)[6], where $\Psi(\mathbf{e}, \mathbf{f})$ denotes a vector consisting of log-scores for features and where $\mathbf{w}$ is now a weight-vector.

As mentioned, we will end up with many h_i's and equally many $\mathbf{w}_i$'s that need to be estimated, generally the number ranges from about 8 up to thousands and even millions. The process of estimating the weight-vector is called *parameter optimisation*, *parameter tuning* or just *tuning*, and is the most computationally expensive part of training a SMT system. It is also the problem that is addressed in this thesis.

The most frequently used method for doing this parameter tuning is Minimum Error Rate Training (MERT) (Och, 2003), but more recently the Margin Infused Relaxed Algorithm (MIRA) (Crammer and Singer, 2003) is used successfully and also the more general-purpose optimisation algorithm Downhill Simplex Method (Nelder and Mead, 1965) is used. These three methods will be described and discussed in depth in Chapter 4 (p. 20) on existing approaches.

1.3 Research Questions

Now that the setting of tuning has been introduced briefly —the stage has been set— we can turn to stating the research questions of this thesis. What follows are the three questions this thesis evolves around. The preceding text, obviously the remainder of this thesis and most explicitly the concluding Chapter 8 (p. 68) are all dedicated to answering these questions thoroughly.

1. **What factors of Tuning in Statistical Machine Translation make it a hard problem?** Is tuning, in the setting of SMT, not *just another optimisation problem*? Numerical optimisation has been studied extensively as a field in its own right. Is there really a need to reinvent the wheel and come up with methods that specialise in optimising a SMT system? The type of model that is optimised, a log-linear model, is not new and definitely not considered among the most difficult ones. That said, what makes our problem stand out from the others, why does it deserve special treatment? Or, and this would be a legitimate answer to this question; is our optimisation problem not harder than other problems per sé, and does it just allow for specialised algorithmic optimisations?

[5]See Chapter 7 (p. 46) on experiments and particularly sections 7.1.1 (p. 46) and 7.1.4 (p. 48) on the specific features we used.

[6]By abuse of notation, we leave out the transposition of the weight-vector in the dot product in equation (1.6). Formally, it should have read: $\Psi(\mathbf{e}, \mathbf{f}) \cdot \mathbf{w}^T$

2. **How do existing Tuning methods for Statistical Machine Translation compare?** A variety of existing methods addressing tuning in the SMT setting exist. Can they be fit into a single framework? If so, how different are they really? What are the fundamental differences, and do empirical results agree with these theoretical differences? Can we draw conclusions from what we observe in experimental results in relation to theoretical aspects of methods; which aspects cause a method to outperform others?

3. **What factors lead to improvements in new Tuning methods for Statistical Machine Translation?** Answering question 2 will supply us with aspects of existing methods that can be addressed to improve tuning. To give ourselves a head-start here, we mention a single aspect of existing methods that spiked our interest in investigating tuning in the first place. Most approaches seem to focus on getting a single best translation on top, is that enough? While it is desirable that a translation system only returns a single translation for any given source sentence —and not multiple— it might still be beneficial to, at the same time, try and optimise a system in such a way that also the runner-ups to the best sentence are good ones. This thought alone deserves investigation. However, other shortcomings of existing methods should also be tackled. And this should be done not just to outperform those methods, it should also serve as means to verify our understanding of the problem and that those methods where indeed lacking because of these shortcomings.

1.4 Contributions of This Work

We believe that it is helpful to explicitly state the contributions of this work to the field. Not to show off but rather to be crystal clear about where we claim to have made progress.

For one, this work will set out by introducing a framework that fits all tuning methods in this thesis. Furthermore, the author holds the strong believe that this framework, given its simplicity, would fit any tuning method within the SMT setting. Such a uniform way of describing a variety of approaches to the same problem brings insight in where such an approach is actually different and is therefore valuable.

We subject three existing approaches, that represent a broad range of possible approaches to tuning, to this strict framework. A theoretical comparison, facilitated by the uniformity of the framework, of these three existing methods follows. This discussion shows the strength of the framework, by pointing out weaknesses of the approaches.

Besides a clear description in words, we provide actual algorithms allowing a reader to easily reproduce our work. These algorithms came forth while re-implementing each method, and are therefore not just exercises. They have been tested and are the blueprint of implementations that underlie all results in this thesis.

A critique might be that what the above describes is only recasting work of others in new words. While the author sees no harm in that, and believes that collecting and providing an overview of the field is a concrete contribution, we also introduce, implement and compare two completely new approaches to tuning. These two approaches are very different in nature from existing work and shed new light on the field. Our sampling approach —Local Unimodal Sampling— performs significantly better than the state-of-the-art approach in at least one realistic setting with 14 features. If we gained no absolute improvement over other methods, merely the simplicity and generality of our method should make it preferable over any other more complicated special purpose algorithm that performs just as well.

Chapter 2

Related Work

This short chapter describes the work of others that touches our work but that we choose not to include beyond this description. We introduce overviews of SMT, and tuning in particular, by other authors. We also describe a variety of attempts on improving SMT performance with techniques related to tuning. The individual existing tuning approaches —MERT, MIRA and Simplex— we address in this work are treated in Chapter 4 (p. 20) because their in-depth description forms an integral part of this thesis.

In their work, dating only a few months back, Arun et al. (2010) describe a Minimum Bayes Risk (MBR) decoder that uses a Gibbs sampler. Their sampler samples the space of possible translation, and not weight-vector space as our sampler does. Although their experimental setup is not directly comparable, their highest BLEU[1] scores on Arabic to English MT05 are 0.45 where in this work we achieve 0.53. Also, their improvements over standard MERT are not big and they do not claim statistical significant improvements. They do point out the low standard deviation, and thus stability of their method. We, although calculated over fewer runs of the algorithm, report even lower standard deviations using our $\text{LUS}_{\alpha=1/100}$ method. Nonetheless, we do not wish to discard their work at all, quite the opposite; their approach is appealing since they use an objective function that can be differentiated, making it possible to use the gradient for optimisation.

In relatively recent work Hayashi et al. (2009) use Support Vector Machines, as we will, within their tuning setting. We did not include their work, but recognise it as being relevant. They show, on very different datasets and language pairs, slight improvements when MERT is extended to maximise the margin —the difference in model-score— between references and wrong translations.

A related, to tuning, approach to improving performance of SMT systems is Reranking. Shen et al. (2004) also have the decoder returning n-best lists. But instead of using these lists to find the best weight-vector, they train a reranker on their development set that stays in place. So, also when the system is deployed on the test set, n-best lists need to be returned and reranked. Such an approach obviously allows for more freedom in the model that is reranking the candidate translations. It can use features beyond those used by the decoder, and it

[1]See section 3.3.1 (p. 15), higher is better.

does not have to stick to a log-linear form. However, the downside of such an approach are the costs that come with returning n-best lists instead of single best translations and performing the required post processing after that. Moreover, Hasan and Ney (2009) show that incorporating models in the search process, and thus eventually in the tuning process, yields better results.

Also Duh and Kirchhoff (2008) worked on a reranking approach in their article on BoostedMERT. They keep a distribution over the n-best list and iteratively tune weight-vectors using MERT. The result is an assemble of MERT-rerankers —weight-vectors—, that obtains a BLEU score improvement between 0.7 and 2.5 points over MERT. Their main argument however is that the log-linear model is insufficient to model the training data —our problem— sufficiently. They claim that the training error for tuning is low, and that we are facing a modeling error instead, that can not be overcome by improving tuning methods.

Macherey et al. (2008) propose a method to perform MERT tuning on *all* translations in the search lattice as opposed to using an n-best list. This has several implications: *a*) the error surface will still be piece-wise constant, but the 'pieces' will be smaller; there are more candidate sentences that can top the list.[2] And *b*) the first iteration will much less over-fit any initial sample from the candidate translations and *c*) weight-vectors found will be much less of an approximation. The authors show much higher BLEU scores for their baseline than in this work, but use a very different experimental setup. Gains for their method are around +1 BLEU point over their baseline.

Foster and Kuhn (2009) investigate the stability of MERT. They report standard deviations, similar to what we report in section 7.2.1 (p. 51), and propose ways of overcoming these instabilities while improving performance. Although their results seem promising, there solutions are computationally expensive.

In his recent work, Zaidan (2009) proposes a re-implementation of MERT that also includes some minor improvements. For one, if a point of interest[3] was found for more than one sentence, the author takes that fact into consideration. The main advantage of his effort is the portability of his software regarding decoder and metric.

Moore and Quirk (2008) investigate the effect of random restarts in MERT. They experiment with varying in the number of restarts, whether to restart at all and with the number of iterations. In their conclusion they suggest only using 5 instead of 19 (or even 23) random restart, not waiting for convergence (cutting off after 7 iterations) and pruning the n-best list, even though they seem not to have performed exactly that experiment, and they do not test whether tuning these meta-parameters (the number of random restarts, the number of iterations) on a separate dataset is possible. A very interesting aspect of their work is the pruning of the n-best list; translation candidates that will never end up on top anyway are pruned away, considerably speeding up the tuning process.

Cer et al. (2008) also explore randomisation in their optimisation algorithm, which is a variation of MERT. As opposed to work in this thesis, they do not take completely random weight-vectors, but rather take the direction in which they are performing the MERT line minimisation randomly. Besides this, attempts are made to make MERT more stable, less susceptible to local optima, by

[2]For more of the piece-wise-constant nature of our objective function, see section 3.2 (p. 11).
[3]See section 4.1 (p. 20) for an explanation of what points-of-interest are.

generalisation —or smoothing— techniques.

Lopez (2007) and Koehn (2008) both provide an overview of the state-of-the-art in Phrase-Based SMT. Lopez (2007) does not focus on tuning much by spending 3 pages on MERT, where Koehn (2008) in his book dedicates 10 pages to the subject, clearly explaining the workings of both MERT and the Downhill Simplex Method.

Duh (2008) proposes a method for evaluation of multiple systems. Machine learning techniques are used to predict human preference of one system over another. Where we use Koehn (2004) to induce statistical significant improvement of one system over another based on BLEU scores, Duh (2008) directly calculates human preference by ranking the systems, skipping the BLEU score. While we acknowledge that such an approach will lead to more reliable judgments, the human annotated datasets make it prohibitively expensive.

Chapter 3

Considerations for a Uniform Framework for Tuning

In the literature many variations in notation and levels of abstraction are used to describe tuning methods. This chapter aims at introducing, or at least defining, a framework that will fit all methods described in this thesis. This framework can then serve as a solid base for an analysis and comparison of the methods. Care has been taken to not limit our framework to approaches described in this thesis. It is as simple and general as it can be.

We will model the decoding and tuning setting largely following Cer et al. (2008), making adjustments and additions where necessary.[1]

3.1 Decoding

Much details regarding the workings of the decoder, and thus much details of SMT, are out of scope of this thesis. We are only concerned with setting the parameters $\mathbf{w}$ of the log-linear model of the system, see (1.6). Beyond that, decoding can be considered a black-box that performs a beam-search through all possible translations given an input sentence $\mathbf{f}$ and these parameters $\mathbf{w}$. Details provided in the introduction in section 1.2.2 (p. 3) are sufficient for understanding the workings of tuning. Here, the workings of a decoder are quickly revisited to introduce some notation used in what follows, in section 3.2 (facing page), on tuning.

Within the tuning setting, we typically have a selection of representative foreign —source— sentences of which we have reference translations. Such a tuning- or development-set usually consists of about 1000 or more foreign sentences, each with 4 or 5 reference translation, created by 4 or 5 professional human translators. Only few such data-sets are available[2] since they are rather laborious and thus

[1] As opposed to Cer et al. (2008), we left out any reference to derivations as it would unnecessarily clutter the notation. Our work can be extended trivially to include them.

[2] We use the NIST MTeval sets in this work, see section 7.1.5 (p. 48).

expensive to produce.

Let us call the foreign side of a development set $\mathbf{F}$, a batch of foreign input sentences, $\mathbf{f}_1, \ldots, \mathbf{f}_k$. For each sentence $\mathbf{f}_i$, its translation into the target language, as produced by a decoder, is denoted by $\mathbf{e}_i$. The set of these translations, $\mathbf{E}$, is drawn from $\mathcal{E}$. With $\mathcal{E}$ representing all possible strings the decoder can produce. The tuples $(\mathbf{e}_i, \mathbf{f}_i)$ are represented by a vector Ψ, consisting of m feature values, defined as:

$$\Psi : \mathcal{E} \times \mathcal{F} \rightarrow \mathbb{R}^m \tag{3.1}$$

Where $\mathcal{F}$ represents all possible input sentences our decoder can translate.

The dot-product of the vector resulting from (3.1) with the weight-vector $\mathbf{w}$ gives the tuple $(\mathbf{e}_i, \mathbf{f}_i)$ a real value; the *model-score*. The task of the decoder is then to find for each $\mathbf{f}_i$ the translation that maximises the model-score:

$$\hat{\mathbf{e}}_i = \underset{\mathbf{e} \in \mathcal{E}}{\operatorname{argmax}} \, \Psi(\mathbf{e}, \mathbf{f}) \cdot \mathbf{w} \tag{3.2}$$

This formula is almost equivalent to (1.6), we just limited the translations to translations that can actually be produced by the decoder. Note here that the *scale* of the model-score and thus of the weight-vector will not change the result of the argmax operation; $\mathbf{w}$ is scale invariant.

For an entire set of input sentences $\mathbf{F}$, the sum of the individual model-scores is maximised by a decoder:

$$\mathbf{E}_{\mathbf{w}} = \underset{\mathbf{E} \in \mathcal{E}^k}{\operatorname{argmax}} \, \Psi(\mathbf{E}, \mathbf{F}) \cdot \mathbf{w} \tag{3.3}$$

Where the aggregate feature vector is defined as the sum of the feature vectors of the individual sentences in the batch:

$$\Psi(\mathbf{E}, \mathbf{F}) = \sum_{i=1}^{k} \Psi(\mathbf{e}_i, \mathbf{f}_i) \tag{3.4}$$

As mentioned before, for our purposes it is sufficient to view the decoder as a system that performs (3.3) and produces $\mathbf{E}_{\mathbf{w}}$ provided with input sentences $\mathbf{F}$ and weight-vector $\mathbf{w}$; $\mathbf{E}_{\mathbf{w}}$ are the translations of $\mathbf{F}$ given $\mathbf{w}$.

It should be clear by now that $\mathbf{w}$ is not varied for different source sentences, $\mathbf{w}$ is a parameter of the system and should do well on average.

3.2 Tuning

The task of tuning then, is to find that $\mathbf{w}$ that gives the *best* translations. Thus we need some way to asses the quality of translations. And we will need an automated means of doing so given the large amount of different $\mathbf{w}$'s, and thus translations, we need to evaluate; an automated evaluation metric is a requirement to even start thinking about tuning. See section 3.3 (p. 15) on some metrics. We will use the Bilingual Evaluation Understudy metric (BLEU) (Papineni et al., 2001) as our metric for reasons described there. Here it is enough to assume that BLEU is a function that takes a set of candidate translation, produced by our decoder, and returns a real value in $[0, 1]$, denoting how good

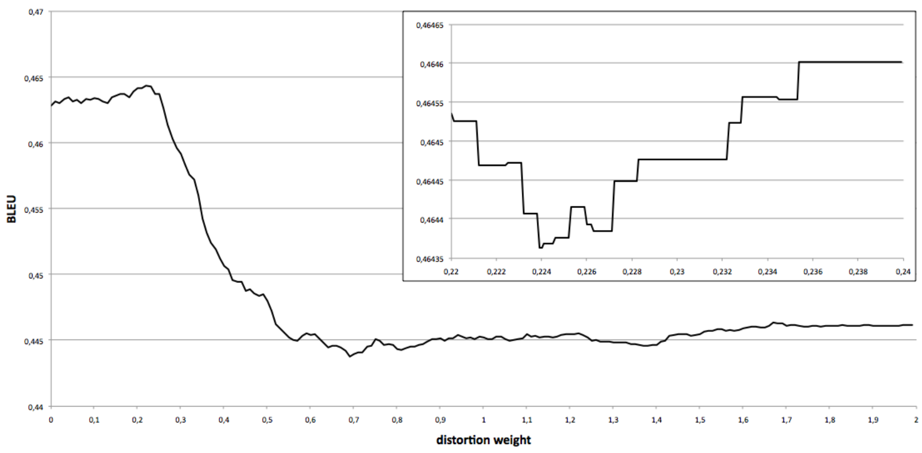


Figure 3.1: The optimisation surface —BLEU— for a SMT system. The distortion weight is being varied while all other weights are kept constant. The BLEU score is measured on our development set, MT04. We follow Koehn (2008) by zooming in on the area around the peak. Both graphs show a rugged surface. Also note, in the lay-in, the piece-wise-constant nature of our objective function; small perturbations in the distortion weight often do not lead to changes in BLEU. See also Appendix A (p. 76) for an overview of this type of graphs.

the translations are; a score of 0 is as bad as it can get, 1 is generally unreachable but good[3].

$$\textsc{Bleu} : \mathcal{E}^n \rightarrow [0, 1] \tag{3.5}$$

With the introduction of BLEU we are given an objective function on which we can optimise. Tuning can thus be defined as finding the optimal weight-vector $\mathbf{w}^*$; the weight-vector that leads to the highest BLEU score.

$$\boxed{\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmax}}\, \textsc{Bleu}(\mathbf{E}_\mathbf{w})} \tag{3.6}$$

This formula, and specifically the argmax operation is central in this thesis.

Tuning seems like a straight forward task. A simple approach would be to apply hill-climbing techniques; vary $\mathbf{w}$ such that BLEU increases with every change. However, if we examine the surface we are optimising it becomes clear that if we do so, we might end up in local optimum, see Figure 3.1.

Moreover, the problem is even worse. Computing the derivative of the function we are optimising, to examine the gradient, is impossible. We are dealing with so called black-box optimisation (Pedersen, 2010, p. 4); given a candidate solution —a weight-vector— our black box produces some measure of fitness: a BLEU score. And that is all we have for guiding us to the optimal solution.

Not only that, the optimisation surface is piece-wise-constant: minor perturbations in $\mathbf{w}$ will likely not cause a shift in the best translations, and thus not in the BLEU score assigned to that $\mathbf{w}$. See Figure 3.1, and in particular the lay-in, for an illustration of the piece-wise-constant characteristics of our objective function.

[3]For the unfamiliar reader, as a rule of thumb, it very much depends on the language though, BLEU scores of 0.3 and above are assigned to sets of translations that are readable in the broadest sense of the word *readable*.

It seems that what is left is a method of trial and error: choose a $\mathbf{w}$, run the decoder to obtain $\mathbf{E_w}$, asses this using (3.5) and try the next $\mathbf{w}$ to run the decoder with, etc. Such an approach is called Direct Search. However, this approach is infeasible for many reasons. For one, running a decoder is computationally expensive. Moreover, Moses (the decoder of our choice) requires a restart whenever weights are changed due to pruning applied to the phrase-table on basis of these weights. But also, the evaluation our fitness function —BLEU— which is holistic —defined on the whole corpus—, is expensive. And on top of that it would be impossible to evaluate all values in weight-vector space, it being a real valued multi-dimensional space.

Despite all these difficulties, it will be necessary to have a reasonable fast algorithm to perform tuning. Tuning is nowadays in integral part of a SMT system, and tuning is performed by researchers and developers to test progress of small adjustment to any part of the pipeline. Therefore, a fast turn-around is required; it is unacceptable to have the whole cycle running for more than a day, preferably much less.

A solution, that avoids reruns of the decoder for every $\mathbf{w}$ is to let the decoder return n-best[4] lists instead of just the single best translation. These n-best translations $\mathbf{e}_i^1, \ldots, \mathbf{e}_i^n$ should be accompanied by $\Psi(\mathbf{e}_i^j, \mathbf{f}_i)$ for each $\mathbf{e}_i^j$. A simplified example of such an n-best list is shown in Table 3.1 (next page).[5]

Using such a list it becomes feasible to perform an approximation of (3.2) and thus (3.3) without running the decoder. For each $\mathbf{w}$ that needs to be evaluated, the model-score for each $\mathbf{e}_i^j$ is calculated, the sentences are re-ordered according to this score and the new top translations for all $\mathbf{f}_i$ are used to calculate the new BLEU-score. The result of this are *approximations* of (3.2) because n-best lists typically can not enlist the full search space $\mathcal{E}^k$. Therefore, any set of *best* translations, found by changing the value of $\mathbf{w}$, that is obtained from these n-best lists is an *approximate* set of best translations, $\tilde{\mathbf{E}}_\mathbf{w}$. It is *approximate* in the sense that the decoder itself might have returned a different set of best translations were it run given the same $\mathbf{w}$.

It is now possible to do a search for the (approximate) optimal weight-vector $\mathbf{w}^*$ without running the decoder:

$$\mathbf{w}^* = \operatorname*{argmax}_{\mathbf{w}} \textsc{Bleu}(\tilde{\mathbf{E}}_\mathbf{w}) \tag{3.7}$$

Note that this is similar to, but an approximation of (3.6); we optimise $\mathbf{w}$ against a BLEU-score on $\tilde{\mathbf{E}}_\mathbf{w}$ rather than $\mathbf{E_w}$.

As mentioned, an issue still remains. The n-best lists are obtained using an (initial) weight-vector $\mathbf{w}$, and this parameter actually determines what is in that list. So we are not only optimising a landscape with many local optima, we are also not able to fully trust the objective function, it is limited by whatever value of $\mathbf{w}$ was used to obtain the n-best list. In fact, a value for $\mathbf{w}$ might be judged with a low BLEU score simply because the one perfect candidate translation that would have been on top with that weight-vector was beyond the nth translation when the n-best list was retrieved, or vice versa. The objective

[4]The word *best* might be a source of confusion. It refers to what the *decoder* 'thinks' is best: the top n sentences according to the model-score, so not necessarily and usually the highest BLEU score.

[5]As can be read in our experiments Chapter 7 (p. 46), we will usually set $n = 500$.

	Candidate Translations	$\Psi(\mathbf{e}_i^j, \mathbf{f}_i)$				$\Psi(\mathbf{e}_i^j, \mathbf{f}_i) \cdot \mathbf{w}$
		d	l	w	t-1	
$\mathbf{e}_i^1$	dutchman jaap de hoop scheffer assumed his duties of nato	0.0	-57.79	-10.0	-17.42	-85.21
$\mathbf{e}_i^2$	the dutch jaap de hoop scheffer , assumed his duties at the head of nato	0.0	-68.1	-15.0	-9.94	-93.04
$\mathbf{e}_i^3$	dutchman jaap de hoop scheffer tops nato assumed his functions	-8.0	-66.19	-10.0	-9.02	-93.21
$\mathbf{e}_i^4$	dutchman jaap de hoop scheffer , assumed his post at the head of the north atlantic treaty organization	0.0	-60.93	-18.0	-14.7	-93.63
$\mathbf{e}_i^5$	dutch jaap de hoop scheffer , assumed his functions on top of nato	0.0	-69.88	-13.0	-12.3	-95.18
$\mathbf{e}_i^6$	dutch jaap de hoop scheffer assumed his functions on top of nato	0.0	-70.08	-12.0	-13.8	-95.88
$\mathbf{e}_i^7$	dutchman jaap de hoop scheffer on ras nato undertakes function	-8.0	-73.1	-10.0	-11.92	-103.02
$\mathbf{e}_i^8$	the dutch jaap de hoop scheffer will undertake its tasks on the nato	0.0	-73.44	-13.0	-18.19	-104.63
$\mathbf{f}_i$	dutchman jaap de hoop scheffer takes over post at top of atlantic organization					

Table 3.1: Example of an n-best translation list (with $n = 8$) for the arabic source phrase shown below, as generated by the decoder. The model-score is calculated using $\mathbf{w} = [1.0\ 1.0\ 1.0\ 1.0]$. With these weights, sentence $\mathbf{e}_i^1$ would have been returned. Note that in our default setting 4 additional translation model weights are present, see section 7.1.1 (p. 46).

الهولندي ياب دي هوب شيفر يتولي مهامه على راس الحلف الاطلسي

function —BLEU— becomes unreliable because of our decision to use n-best list instead of re-running the decoder each time we change $\mathbf{w}$. It is therefore common practice to iteratively run the decoder with new found $\mathbf{w}$'s —found with our optimisation algorithm— and perform (3.7) until the n-best list or $\mathbf{w}^*$ does not change significantly anymore, as illustrated in Figure 3.2 (next page). Every next n-best list is merged with the previous in order to get a sample of the search space $\mathcal{E}^k$ that is as accurate as possible.

We can now, in the next sections, turn to the implementation of the argmax operator from equation (3.7). This is not as simple as it might seem, we cannot exhaustively search the space of possible $\mathbf{w}$, it is an m-dimensional space over real numbers.

In general, a tuning algorithm is of the form listed in Algorithm 1 (p. 16).

The function n-Best$(\mathbf{w}, \mathbf{F})$ is a call to the decoder, where the decoder is expected to return a set of n-best translation given a batch of foreign sentences $\mathbf{F}$ and a weight-vector $\mathbf{w}$.

Optimise() can be replaced by a specific implementation of a tuning algorithm, such as OptimiseMERT() in Algorithm 2 (p. 24), OptimiseMIRA() in Algorithm 3 (p. 28), OptimiseSimplex() in Algorithm 4 (p. 31), OptimiseLus$_\alpha$() in Algorithm 5 (p. 42) or OptimiseRank() in Algorithm 6 (p. 45).

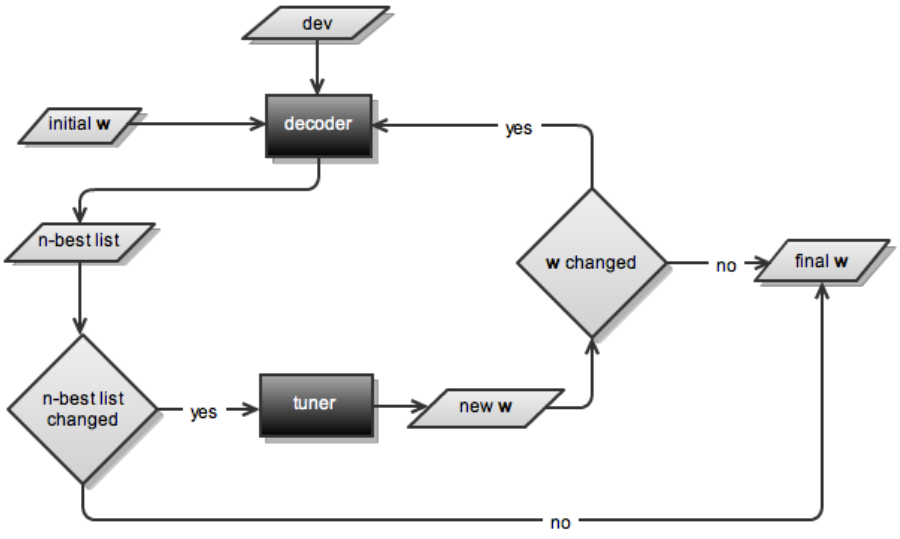


Figure 3.2: Iterative parameter tuning, using n-best lists, following Koehn (2008). Convergence is reached when either $\mathbf{w}$ or the n-best does not change (significantly) anymore.

3.3 Evaluation Metrics

Over the years many metrics have been devised to measure translation quality. Since an automated metric, one that does not involve human assessment, is essential to tuning we describe a few in this section. Our selection includes BLEU, NIST, METEOR, WER, PER and TER. We will however follow suit in our decision of which one to choose and stick with the BLEU score. We will therefore discuss this metric first and in much more depth.

3.3.1 BLEU

The Bilingual Evaluation Understudy (BLEU), (Papineni et al., 2001) is the most often used metric for evaluation Machine Translation performance since it was found to correlate well with human judgment of how good translations are. Recently, Cer et al. (2010) reconfirmed this.

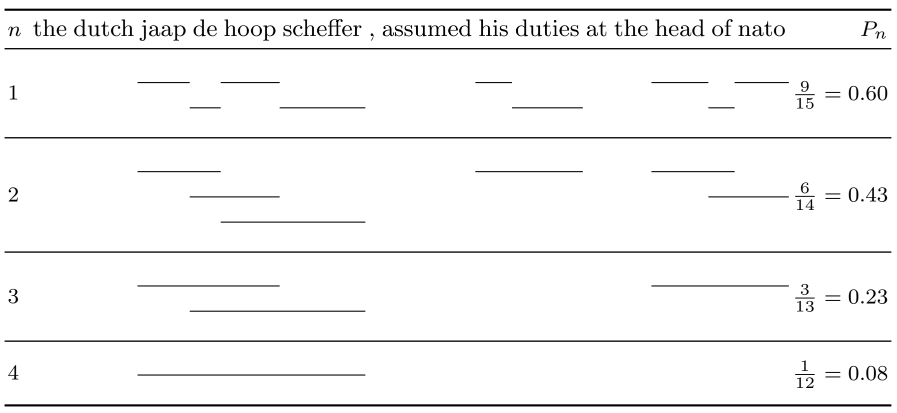
The BLEU metric measures the correspondence between a translation of the system, a candidate translation, and multiple human reference translations of the same sentence. In this work, there will always be 4 or 5 reference translations per sentence.[6]

This correspondence between candidate and references is defined upon the precision of n-grams in the candidate translation with respect to the references. For an example see Table 3.2 (next page). For $n = 1 \ldots 4$ the number of n-grams with a match in one of the references are counted and divided by the total number of n-grams to get the n-gram precision P_n.

[6]More on the specific datasets that were used can be found in section 7.1.5 (p. 48).

Algorithm 1 TUNING($\mathbf{F}, \mathbf{R}, \mathbf{w}$)

```
Require: $\mathbf{F}$, a set of source sentences, $\mathbf{f}_1, \ldots, \mathbf{f}_k$
Require: $\mathbf{R}$, a set of corresponding target reference sentences
Require: An initial weight-vector $\mathbf{w}$
1: $\mathbf{E}^n \leftarrow \{\}$
2: repeat
3:     $\mathbf{w}' \leftarrow \mathbf{w}$
4:     $\mathbf{E}^{n'} \leftarrow \mathbf{E}^n$
5:     $\mathbf{E}^n \leftarrow \mathbf{E}^n \cup n\text{-BEST}(\mathbf{w}, \mathbf{F})$
6:     $\mathbf{w} \leftarrow \text{OPTIMISE}(\mathbf{w}, \mathbf{E}^n, \mathbf{R})$
7: until $\|\mathbf{w} - \mathbf{w}'\| < \epsilon$ or $\mathbf{E}^n \approx \mathbf{E}^{n'}$
8: return $\mathbf{w}$
```



n	the dutch jaap de hoop scheffer , assumed his duties at the head of nato	P_n
1		$\frac{9}{15} = 0.60$
2		$\frac{6}{14} = 0.43$
3		$\frac{3}{13} = 0.23$
4		$\frac{1}{12} = 0.08$

Table 3.2: Precision calculation for BLEU with a single reference:
dutchman jaap de hoop scheffer takes over his duties as head of nato.
Matches for this candidate with the reference sentence are shown as lines. The number of matches, lines, is divided by the total number n-grams for each n to arrive at P_n.

There is a problem though, with these counts. It is best illustrated with a candidate sentence like: *his his his his his his his.* When using the same reference sentence, from Table 3.2, this sentence will get a very high precision score for $n = 1$: $P_1 = \frac{7}{7} = 1$. Obviously this is not a desired property of the measure. Which is why the notion of *clipped counts* was introduced: counts for an n-gram are clipped with the maximum number of times the n-gram occurs in any reference sentence. The precision would be corrected to $P_1 = \frac{1}{7} = 0.14$ because the 1-gram *his* only occurred once in the reference translation.

To calculate BLEU, the product of these clipped n-gram precisions is taken:

$$\text{BLEU}_n = BP \cdot \prod_{i=1}^{n} P_n \tag{3.8}$$

And penalised with a brevity penalty, BP, where lengths are calculated in number of words:

$$BP = \min\left(1, \frac{\text{output-lenght}}{\text{reference-length}}\right) \tag{3.9}$$

This brevity penalty is necessary because the measure is a precision measure; otherwise a system could perform relatively well by always returning small or even single word sentences, consisting of a very common word; a 'sentence' like *his*. Such a small sentence would in the example render a brevity penalty of $BP = \min(1, \frac{1}{15}) \approx 0.067$.

In the case of multiple references, which is the common case, some variations in which length to take exist. More recent implementations of the BLEU score use the length of the reference sentence that is nearest in length to the candidate translation, and in case of a tie, the shortest. Since we use the standard NIST `mteval-v11b.pl` script in this work for final evaluation of each system, we match what they do there in our objective function. Our brevity penalty is calculated using the length of the *shortest* reference translation.

Using the before mentioned example, the result would be as follows for some n.

$$\begin{aligned}
\textsc{Bleu}_1 &= \min\left(1, \frac{15}{13}\right) \cdot 0.6 = 0.6 \\
\textsc{Bleu}_2 &= \min\left(1, \frac{15}{13}\right) \cdot 0.6 \cdot 0.43 = 0.26 \\
\textsc{Bleu}_3 &= \min\left(1, \frac{15}{13}\right) \cdot 0.6 \cdot 0.43 \cdot 0.23 = 0.06 \\
\textsc{Bleu}_4 &= \min\left(1, \frac{15}{13}\right) \cdot 0.6 \cdot 0.43 \cdot 0.23 \cdot 0.08 = 0.005 \\
\textsc{Bleu}_5 &= \min\left(1, \frac{15}{13}\right) \cdot 0.6 \cdot 0.43 \cdot 0.23 \cdot 0.08 \cdot 0 = 0
\end{aligned}$$

It is clear from these examples that for larger n the scores drop. And with a single reference this metric is quite unstable; as soon as for a certain n not a single n-gram appears in the reference, the whole measure goes to 0.[7] Papineni et al. (2001) showed that $\textsc{Bleu}_4$ gave the most satisfactory results, so when we refer to BLEU, we mean $\textsc{Bleu}_4$.

The above, the instability, is a reason why BLEU is usually calculated on a whole corpus instead of just on a single sentence. And op top of that using multiple references per sentence, where the references should ideally be as varied as possible. BLEU can only be expected to correlate with human judgment when averaged over many sentences and if the goal is comparing systems that are treated the same, with the same test sets (Papineni et al., 2001, p. 313). This holistic property of BLEU is also one of its downsides; it means that we can not reliably calculate the BLEU score of an individual sentence, it is always done in the context of a corpus. Some variations on BLEU, that aim at overcoming this limitation, exist and are discussed in 5.3.1 (p. 36).

[7]Note that mentioning $\textsc{Bleu}_5$ here only serves as an illustration of this point, it will not be used anywhere else in this thesis.

3.3.2 Other Evaluation Metrics

NIST

A metric of the U.S. National Institute of Standards and Technology, the metric is called NIST after the institutes name, is very similar to BLEU (Doddington, 2002). The main difference is that n-grams are weighted with their inverse document frequency, which seems a reasonable idea. Although one could argue that getting the function words right, those with low inverse document frequency, is also a rather important part of SMT. And maybe even more difficult than n-grams with high inverse document frequencies that will probably occur only in very specific contexts, leaving less room for error anyway.

Just as BLEU, NIST performs well according to human judges (Cer et al., 2010). The reason we prefer BLEU over NIST is that BLEU is used more frequently, computationally cheaper to use in evaluation and slightly easier to implement.

METEOR

The Metric for Evaluation of Translation with Explicit ORdering (METEOR) (Banerjee and Lavie, 2005) does some word alignment by itself. It aligns word in a candidate translation with words in reference sentences and bases an F-measure on the recall and precision of aligned words. This seems a rather expensive way of performing evaluation. The method is not adopted widely.

WER

Word Error Rate (WER) (Niessen et al., 2000) is a metric that was introduced to evaluate Automatic Speech Recognition systems, but has been used to evaluate SMT systems as well. The metric is based on a Levenshtein distance between words in the candidate translation and a reference. In fact, BLEU was designed with this metric in mind (Papineni et al., 2001, p. 311).

PER

Position-independent word Error Rate (PER) was introduced by Och (2002, p. 22) to overcome the strict word-order requirement of WER. In PER, words without a match are counted as substitution errors. Shorter translations are punished with deletion errors, long translations with insertion errors.

TER

The Translation Edit Rate (TER) (Snover et al., 2006) is a metric that uses the number of edits of the candidate translation needed to obtain a reference translation. The edits are deletion, insertion, and substitution of words, like WER. And besides that, swaps of consecutive sequences of words, of n-grams, are allowed.

3.4 Summary of Difficulties in Tuning

In the above sections 3.2 (p. 11) and 3.3 (p. 15) we have detailed many reasons for deeming our optimisation problem a difficult one. Here we reiterate our considerations in a concise manner.

Non differential objective function
Optimisation techniques cannot follow a gradient.

Piece-wise constant objective
Even evaluating two nearby points in weight-vector space may not provide us with a gradient.

Infinite number of possible parameters
If evaluation of a weight-vector were cheap, it would not be possible to try all points in weight-vector space, given the infinite size of a real valued multi-dimensional space.

Expensive evaluation of holistic BLEU
We have to restrict the total number of evaluations. The BLEU metric is defined on a corpus instead of sentences and thus computationally expensive.

Expensive to run a decoder
Ideal evaluation of $\mathbf{w}$, using a decoder, needs to be approximated by reordering an n-best list since running the decoder for each candidate $\mathbf{w}$ is prohibitively expensive.

Unreliable evaluation since we use n-best list approximations
This approximation renders our objective function unreliable. And forces us to repeat the whole process.

Run-time is an issue
As tuning is part of the SMT pipeline. For research and development purposes, on any part of SMT, it is a necessity to have a fast turn-around time.

All these difficulties are to be overcome by a successful approach to tuning. In the next chapter we will reformulate and discuss existing approaches to gain insight in how this can be done and where opportunities for improvement can be found.

Chapter 4

Casting Existing Tuning Approaches in a Uniform Framework

This chapter gives an overview of existing approaches to tuning within the setting of SMT. It also strives to place this work within the framework from Chapter 3 (p. 10). By standardising a setting for existing methods, a comparison comes forth naturally. The placement of existing methods within this framework should be considered a major contribution of this work to the field, as such a comparison does not yet exist.

We have chosen to discuss three existing methods: Minimum Error Rate Training (MERT) in section 4.1, Margin Infused Relaxed Algorithm (MIRA) in section 4.2 (p. 23) and Simplex in section 4.3 (p. 27). MERT is by far the most commonly used of these three, and comes distributed in various implementations with commonly used decoders like Moses and Joshua. MIRA has recently also been used. Simplex has not been heard of as much within the SMT literature until recently (Koehn, 2008; Macherey et al., 2008; Hasan and Ney, 2009; Zhao and Chen, 2009).

4.1 Minimum Error Rate Training

What follows is a description of MERT, mainly inspired by Och (2003); Cer et al. (2008); Koehn (2008). In this section, MERT is reformulated to fit into the framework and described in algorithmic form.

As any tuning algorithm in this thesis, MERT is initialised as follows: select $\mathbf{F}$, a batch of foreign input sentences, $\mathbf{f}_1, \ldots, \mathbf{f}_k$, choose an initial weight-vector $\mathbf{w}$ and obtain an n-best list $\mathbf{e}_i^1, \ldots, \mathbf{e}_i^n$ from the decoder using $\mathbf{w}$.

Then the iterative part of the algorithm starts, where in the first run the starting point is the initial weight-vector $\mathbf{w}$ and in later iterations this will be the best weight-vector from the previous iteration. After each iteration, the decoder is run again to obtain new n-best lists that are merged with existing ones. Besides this single starting point, MERT typically uses a number of additional random points in vector space to avoid poor local optima (Lopez, 2007). These random points are not completely random, but within fixed ranges[1]. The iterations stop if there are no changes in the weight-vector, or if there are no new translations in the n-best list.

Technically, MERT is minimising an error, the error metric in our case is 1 minus BLEU; $\ell = 1 - \textsc{Bleu}$. So the search that MERT is performing is:

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} \ell(\tilde{\mathbf{E}}_{\mathbf{w}}) \tag{4.1}$$

Now each of the starting points —each random weight-vector— mentioned before is in turn taken as a starting point for this search.

As mentioned in the previous section, the error function that is being optimised is piecewise-constant; that is $\ell(\tilde{\mathbf{E}}_{\mathbf{w}})$ mostly does not change with changes in $\mathbf{w}$, simply because a small change in $\mathbf{w}$ likely does not influence the model-score enough to prefer another candidate (in the n-best list) as the single best translation.

In parameter optimisation this is often considered a problem; it makes finding a gradient very difficult. However, MERT exploits this fact using Powell's method (Powell, 1964) by first finding the points of interest; the points where the loss *does* change. These points are the points where the single best translations changes and thus the BLEU score, and the loss, changes. This greatly reduces the number of points in weight-vector space that we have to consider; from an infinite number of points to typically a few thousand points.

Powell's method starts with a single point in weight-vector space and then performs an optimisation along each individual dimension of this vector. In the extremely simplified case of a two dimensional space this search has (Koehn, 2008) and can be described as finding the highest point in a hilly city with a grid of streets, like San Francisco. We start along a certain street. Find its highest point and continue along the cross-street. Also in this cross-street we find the highest point. Obviously, we are not guaranteed to actually find the highest point, and whether we do so will also depend on our starting position. But despite these simplifications and limitations, examining a single 'street' —a single dimension— in weight-vector space is still a hard task since the same problems of an infinite number of points along this dimension and the piece-wise character still exist. We will need to find the points of interest along each dimension.

Recall Equation 1.5 (p. 3) that models our way of finding a best translation using features Ψ_i and weights $\mathbf{w}_i$

$$\hat{\mathbf{e}} = \operatorname*{argmax}_{\mathbf{e}} \sum_i \Psi_i(\mathbf{e}, \mathbf{f}) \cdot \mathbf{w}_i \tag{4.2}$$

Now we are interested in finding the points along a single dimension of $\mathbf{w}$ where our best translation changes. Let us call this specific dimension var, its weight is then $\mathbf{w}_{var}$ and its feature value is Ψ_{var}.

We will keep all other dimensions, $i \neq var$, fixed while we are looking at var, so

[1]We use 23 additional weight-vectors, to make optimal use of the 8-core architecture of the machines used. See section 7.1.1 (p. 46) for the ranges we used.

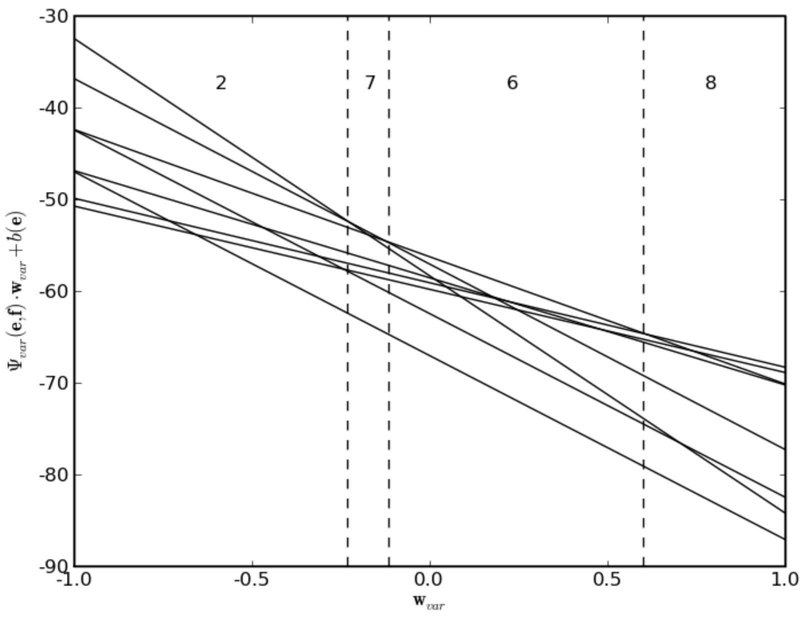


Figure 4.1: An illustration of our method to find points of interest. For an n-best list with $n = 8$ we get 3 points of interest along the **t-1** dimension: -0.23, -0.115 and 0.6. The used sentences and the feature values are in Table 4.1 (next page)

we can define a portion of the model-score as fixed for a certain **e**

$$b(\mathbf{e}) = \sum_{i \neq var} \Psi_i(\mathbf{e},\mathbf{f}) \cdot \mathbf{w}_i \tag{4.3}$$

This reduces (4.2) to:

$$\hat{\mathbf{e}} = \underset{\mathbf{e}}{\operatorname{argmax}}\, \Psi_{var}(\mathbf{e},\mathbf{f}) \cdot \mathbf{w}_{var} + b(\mathbf{e}) \tag{4.4}$$

Each candidate translation **e** in our n-best list is now represented by a line: $\Psi_{var}(\mathbf{e},\mathbf{f}) \cdot \mathbf{w}_{var} + b(\mathbf{e})$. We can plot these lines for all candidates in an n-best list, as done in Figure 4.1. If we do so, we see that at any given point along the $\mathbf{w}_{var}$ axis a single translation has the highest model-score. We can analytically work out where these *best* translations switch, it happens at a specific number of intersections of the before mentioned lines. In the figure, the vertical dotted lines denote the points of interest for this specific sentence along the **t-1** dimension.

For each dimension the union of all points of interest over all sentences is taken. Each of these points is evaluated using BLEU. In general, the point with the highest BLEU score is taken as the optimal point for that dimension. However, some variations exist, Cer et al. (2008) take also the neighbouring points along each dimension of interest into account when optimising: either the average or the lowest BLEU score of the point in question and its two neighbours is taken into consideration.

	Candidate Translations	d	l	w	**t-1**	t-2	t-3	t-4	t-5
1	dutchman jaap de hoop scheffer , will take office at the head of the north atlantic treaty organization	0,00	-55,88	-18,00	-20,05	-33,39	-22,54	-41,44	8,00
2	dutchman jaap de hoop scheffer will take office on the nato	0,00	-54,30	-11,00	-25,87	-30,87	-15,55	-19,06	8,00
3	dutchman jaap de hoop scheffer , will take office at the head of nato 's	0,00	-56,38	-15,00	-20,02	-30,21	-21,08	-29,99	7,00
4	dutchman jaap de hoop scheffer , assumed his duties at the head of the nato	0,00	-64,88	-15,00	-9,07	-26,39	-18,67	-28,91	8,00
5	dutchman jaap de hoop scheffer assumed his post at the head of the nato	0,00	-63,55	-14,00	-11,68	-27,50	-15,62	-27,05	8,00
6	dutchman jaap de hoop scheffer , assumed duties at the head of nato	0,00	-62,02	-13,00	-13,87	-25,81	-15,59	-26,27	8,00
7	dutchman jaap de hoop scheffer will function at the head of nato	0,00	-57,66	-12,00	-20,22	-28,60	-14,89	-23,75	8,00
8	dutchman jaap de hoop scheffer take over his duties at the head of nato	0,00	-64,05	-14,00	-9,21	-28,91	-13,50	-28,33	6,00

Table 4.1: A small sample from an n-best list, these sentences are used in the example shown in Figure 4.1 (facing page).

For MERT in a more algorithmic for see Algorithm 2 (next page). The algorithm is a function, that can be called by Algorithm 1 (p. 16). It expects, as any of our tuning algorithms, an initial weight-vector —which, in later iterations, can be the best of a previous iteration—, a merged n-best list —the union of n-best lists of all previous iterations— and a set of reference translations, which is merely used to calculate a BLEU score. A few notes on the used notation follow. On line 3, the call $\leftarrow zeros(m)$ returns a vector of length m consisting of just zeros. The next line sets one element in that vector to 1. Any bold printed variable in the algorithm denotes either an indexable array or a vector. The first is always indexed with super- or sub-script the latter is not indexed.

4.2 Margin Infused Relaxed Algorithm

The more recent optimisation algorithm Margin Infused Relaxed Algorithm (MIRA) (Crammer and Singer, 2003; Crammer et al., 2006), can also be applied to tuning in SMT (Chiang et al., 2008, 2009; Watanabe et al., 2007a,b). MIRA is an on-line learning method; it updates the weight-vector on the fly as training examples, candidate translation pairs in this case, come in one by one. The way the updates are conducted is crucial and is the heart of the algorithm. Earlier steps just serve as a means of making the algorithm computationally feasible. It is attractive to use MIRA because its updates to the weight-vector are in proportion to the loss incurred by misclassifying a pair of candidate translations. In other words, MIRA adapts a weight-vector based on how far off we are

Algorithm 2 OptimiseMERT($\mathbf{w}, \mathbf{E}^n, \mathbf{R}$)

```
Require: $\mathbf{w}$ a starting point in weight-vector space
Require: $\mathbf{E}^n$ an $n$-best list $\mathbf{e}_1^1, \dots, \mathbf{e}_1^n, \dots, \mathbf{e}_k^1, \dots, \mathbf{e}_k^n$
Require: $\mathbf{R}$ corresponding reference translations
 1: $m \leftarrow |\mathbf{w}|$
 2: for $l \leftarrow 1$ to $m$ do
 3:    $\mathbf{d} \leftarrow zeros(m)$
 4:    $\mathbf{d}_l \leftarrow 1$
 5:    $T \leftarrow \{\}$
 6:    for $i \leftarrow 1$ to $k$ do
 7:       for $j \leftarrow 1$ to $n$ do
 8:          $\mathbf{m}_i^j \leftarrow \Psi(\mathbf{e}_i^j, \mathbf{f}_i) \cdot \mathbf{d}$
 9:          $\mathbf{b}_i^j \leftarrow \Psi(\mathbf{e}_i^j, \mathbf{f}_i) \cdot \mathbf{w}$
10:       $t \leftarrow 0$
11:       $j_t^* \leftarrow \mathrm{argmax}_{1 \le j < n}\, \mathbf{m}_i^j$
12:       repeat
13:          $j_{t+1}^* \leftarrow \mathrm{argmin}_{1 \le j < n} \max\left(0, \frac{\mathbf{b}_i^{j_t^*} - \mathbf{b}_i^j}{\mathbf{m}_i^j - \mathbf{m}_i^{j_t^*}}\right)$
14:          $intercept \leftarrow \max\left(0, \frac{\mathbf{b}_i^{j_t^*} - \mathbf{b}_i^{j_{t+1}^*}}{\mathbf{m}_i^{j_{t+1}^*} - \mathbf{m}_i^{j_t^*}}\right)$
15:          $t \leftarrow t + 1$
16:          if $intercept > 0$ then
17:             $T \leftarrow T \cup \{intercept\}$
18:       until $intercept = 0$
19:    $bleumax \leftarrow 0$
20:    $t^* \leftarrow \epsilon$
21:    for $t \in$ sort(T) do
22:       $\tilde{\mathbf{w}} \leftarrow \mathbf{w} + (t - \epsilon) \cdot \mathbf{d}$
23:       if Bleu($\tilde{\mathbf{E}}_{\tilde{\mathbf{w}}}, \mathbf{R}$) $> bleumax$ then
24:          $t^* \leftarrow t$
25:          $bleumax \leftarrow$ Bleu($\tilde{\mathbf{E}}_{\tilde{\mathbf{w}}}, \mathbf{R}$)
26:    $\mathbf{w} \leftarrow \mathbf{w} + (t^* - \epsilon) \cdot \mathbf{d}$
27: return $\mathbf{w}$
```

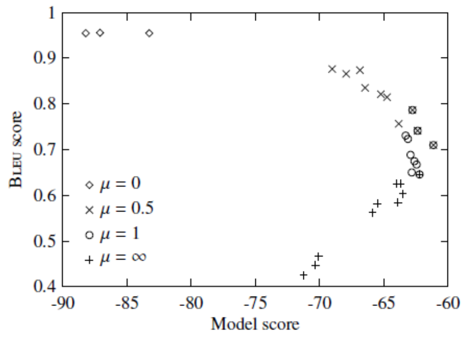


Figure 4.2: Scatter plot of 10-best unique translations of a single sentence obtained by using various values of μ in equation (4.5) (taken from Chiang et al. (2008))

—expressed in feature-vector difference— for a pair of translations and how much this costs —expressed in sentence BLEU difference— to be that far off.

Again as with MERT and any algorithm in this thesis, we need $\mathbf{F}$, a batch of foreign input sentences, $\mathbf{f}_1, \ldots, \mathbf{f}_k$. Also, we have to choose an initial weight-vector $\mathbf{w}$. We then iteratively do the following:

1. To limit the number of training examples used in Step 2 we iterate through the development set, so for each $\mathbf{f}_i$ do the following.

   (a) Translate $\mathbf{f}_i$ to obtain an n-best lists $\mathbf{e}_i^1, \ldots, \mathbf{e}_i^n$ from the decoder using $\mathbf{w}$ and merge these with existing, if any, n-best lists for this $\mathbf{f}_i$.

   (b) Keep from this merged list only those translations $\mathbf{e}_i^1, \ldots, \mathbf{e}_i^l$ that are the 10-best (per μ) according to:

   $$\mathbf{e}_i = \underset{\mathbf{e}_i^j}{\operatorname{argmax}}\, \textsc{Bleu}(\mathbf{e}_i^j) - \mu(\textsc{Bleu}(\mathbf{e}_i^j) - \Psi(\mathbf{e}_i^j, \mathbf{f}_i) \cdot \mathbf{w}) \qquad (4.5)$$

   For $\mu = 0.5$, $\mu = 1$ and $\mu = \infty$, see Figure 4.2.

   That is, for each source sentence we only keep a representative sample, consisting of maximum 30 translations, of the n-best list. Some are chosen because of their BLEU score, some due to their model-score, Ψ, and some on the basis of a combination of the BLEU and model-score.

   (c) Select a single oracle translation $\mathbf{e}_i^*$ by taking the single best translation according to (4.5) with $\mu = 0.5$. The oracle translation is, or is supposed to be, the *best* translation the decoder could produce using this $\mathbf{w}$. Other ways of choosing this oracle translation exist, including methods of forcing the decoder to generate one of the reference translations. Here we follow Chiang et al. (2008).

   (d) For each $\mathbf{e}_i^j$ in our sample, we pre-compute the difference in feature vector to our oracle

   $$\Delta\Psi_i^j = \Psi(\mathbf{e}_i^*, \mathbf{f}_i) - \Psi(\mathbf{e}_i^j, \mathbf{f}_i) \qquad (4.6)$$

   The dot-product of $\Delta\Psi_i^j$ with a weight-vector $\mathbf{w}$ is called the margin, the namesake of this algorithm.

(e) We also pre-compute the *loss*, ℓ, for each $\mathbf{e}_i^j$ in our sample:

$$\ell_i^j = \textsc{Bleu}(\mathbf{e}_i^*) - \textsc{Bleu}(\mathbf{e}_i^j) \tag{4.7}$$

The loss signifies what harm would be incurred by preferring a candidate translation $\mathbf{e}_i^j$ over the oracle $\mathbf{e}_i^*$.

There is a problem with the use of BLEU. BLEU was defined on a corpus level, not on sentence level, see section 3.3.1 (p. 15). Again, we follow Chiang et al. (2008) and define BLEU on a pseudo-document: a moving average over previously translated sentences. This pseudo-document serves as a corpus on which BLEU is defined.

2. Step 1 left us with a limited number of translation candidates; a maximum of $30i$. We can now start updating $\mathbf{w}$. Just like Chiang et al. (2008), we take the objective —in (4.8)— as is done by (Crammer and Singer, 2003) in their original article. So, our objective is different from what Watanabe et al. (2007a,b) do.

$$\mathbf{w} = \operatorname*{argmin}_{\mathbf{w}'} \underbrace{\frac{1}{2}\|\mathbf{w}' - \mathbf{w}\|^2}_{\mathbf{w}' \text{ near } \mathbf{w}} + C \sum_{i=1}^{k} \max_{1 \le j \le l} \underbrace{(\ell_i^j - \underbrace{\Delta\Psi_i^j \cdot \mathbf{w}')}_{\text{margin}}}_{\text{Hinge loss}} \tag{4.8}$$

The update rule forces $\mathbf{w}'$ to stay near $\mathbf{w}$, to keep a new weight-vector close to the previous one and thereby retain information learned in previous iterations. On the other hand, we prefer a $\mathbf{w}'$ that scores our oracle $\mathbf{e}_i^*$ higher than each candidate translation $\mathbf{e}_i^j$ with a margin at least as large as the loss. In other words: we prefer a $\mathbf{w}'$ that minimises the maximum Hinge loss it incurs while staying close to $\mathbf{w}$.

The variable C defines the size of the weight-vector update, it is the learning rate. We set $C = 0.01$ following Chiang et al. (2008, 2009), and verified this setting on a separate dataset[2].

Obviously, we can still not simply enumerate all $\mathbf{w}'$ to find the one that minimises (4.8), just as in section 3.2 (p. 11). Instead, a variant of Sequential Minimal Optimisation by Platt (1998), that solves our constrained objective, is used. So, instead of attempting to perform (4.8) directly, we approximate it by the following, online, procedure:

(a) For each i and each j with $1 \le j \le l$, initialise a Lagrange multiplier a_i^j as follows:

$$a_i^j = \begin{cases} C & \text{if } \mathbf{e}_i^j = \mathbf{e}_i^* \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

Only those Lagrange multipliers associated with a oracle get a value other then 0, and only those will play a role at first.

(b) Make a copy of current weight-vector, so we can see whether we made progress: $\mathbf{w}' \leftarrow \mathbf{w}$

[2]This was done on MT02, LDC2003T18.

(c) Then, repeatedly until $\mathbf{w}'$ does not change anymore, do the following: for each i and each pair of j and j', with $1 \leq j \leq l$:

$$\begin{aligned} \mathbf{w}' &\leftarrow \mathbf{w}' + \delta \cdot (\Delta\Psi_i^j - \Delta\Psi_i^{j'}) && (4.10) \\ a_i^j &\leftarrow a_i^j + \delta && (4.11) \\ a_i^{j'} &\leftarrow a_i^{j'} - \delta && (4.12) \end{aligned}$$

with

$$\delta = \max\left(-a_i^j, \min\left(a_i^{j'}, \frac{(\ell_i^j - \ell_i^{j'}) - (\Delta\Psi_i^j - \Delta i\Psi_i^{j'}) \cdot \mathbf{w}'}{\|\Delta\Psi_i^j - \Delta\Psi_i^{j'}\|^2}\right)\right) \quad (4.13)$$

At first, $\mathbf{w}'$ only changes when an oracle translation is involved in the pair, in the case that $\mathbf{e}_i^{j'} = \mathbf{e}_i^*$. In following iterations also other pairs start influencing the weight-vector.

3. The algorithm terminates, Step 1 and Step 2 are not performed anymore, on the following condition: $\|\mathbf{w}' - \mathbf{w}\| < \epsilon$. If the weight-vector did not change significantly, we do not expect any new translations to come up in a run of our decoder. And thus a new run of the algorithm will not result in any progress.

4. If, on the other hand, there was a change in the weight-vector, we continue with the new one, thus: $\mathbf{w} \leftarrow \mathbf{w}'$

In even more algorithmic form MIRA looks like Algorithm 3 (following page). Notation related issues are as described briefly in section 4.1 (p. 20).

4.3 Downhill Simplex Method

The Downhill Simplex Method (Nelder and Mead, 1965) —also called Nelder-Mead method, amoeba method or in this work usually just Simplex— is a general and often used optimisation algorithm, applied to problems where the gradient is unknown, like ours. What follows below is following Koehn (2008); Press et al. (2007), and adjusted to fit into the framework presented in Chapter 3 (p. 10).

The method is called after the geometrical figure *simplex*. A simplex is a generalisation of a triangle into more dimensions: in m dimensions, a simplex consists of $m + 1$ points and all connecting vertex.

The Downhill Simplex Method consist of transforming a simplex in weight-vector space. These transformations, or reflections, are designed to allow the initial simplex to find its way down the error surface. The error surface is again $1 -$ Bleu, corpus BLEU that is. Maybe the most appealing thing about Simplex is the fact that it can be described in geometrical terms.

The reasons for applying the simplex algorithm to tuning are *a*) the problem description it was developed for fits that of tuning; *b*) it is simple, almost trivial, to implement; and *c*) simplex can work with a holistic metric, like BLEU.

1. The initial simplex starts with $m + 1$ random points in weight-vector space at the first iteration, or with m random points and the best point from the previous iteration in later iterations.

Algorithm 3 OptimiseMIRA($\mathbf{w}, \mathbf{E}^n, \mathbf{R}$)

```
Require: $\mathbf{w}$ a starting point in weight-vector space
Require: $\mathbf{E}^n$ an $n$-best list $\mathbf{e}_1^1, \ldots, \mathbf{e}_1^n, \ldots, \mathbf{e}_k^1, \ldots, \mathbf{e}_k^n$
Require: $\mathbf{R}$ corresponding reference translations
 1: for $i \leftarrow 1$ to $k$ do
 2:   $O_i \leftarrow \text{arg10max}_{\mathbf{e}_i^j} \Psi(\mathbf{e}_i^j, \mathbf{f}_i) \cdot \mathbf{w}$
 3:   $O_i \leftarrow O \cup \text{arg10max}_{\mathbf{e}_i^j} \textsc{Bleu}(\mathbf{e}_i^j, \mathbf{R}) + \Psi(\mathbf{e}_i^j, \mathbf{f}_i) \cdot \mathbf{w}$
 4:   $O_i \leftarrow O \cup \text{arg10max}_{\mathbf{e}_i^j} -\textsc{Bleu}(\mathbf{e}_i^j, \mathbf{R}) + \Psi(\mathbf{e}_i^j, \mathbf{f}_i) \cdot \mathbf{w}$
 5:   $\mathbf{e}_i^* \leftarrow \text{argmax}^1_{\mathbf{e}_i^j} \textsc{Bleu}(\mathbf{e}_i^j, \mathbf{R}) + \Psi(\mathbf{e}_i^j, \mathbf{f}_i) \cdot \mathbf{w}$
 6:   for each $\mathbf{e}_i^j \in O_i$ do
 7:     $\Delta\Psi_i^j \leftarrow \Psi(\mathbf{e}_i^*, \mathbf{f}_i) - \Psi(\mathbf{e}_i^j, \mathbf{f}_i)$
 8:     $\ell_i^j \leftarrow \textsc{Bleu}(\mathbf{e}_i^*, \mathbf{R}) - \textsc{Bleu}(\mathbf{e}_i^j, \mathbf{R})$
 9:     $a_i^j \leftarrow \begin{cases} C & \textbf{if } \mathbf{e}_i^j = \mathbf{e}_i^* \\ 0 & \textbf{else} \end{cases}$
10: $\mathbf{w}' \leftarrow \mathbf{w}$
11: repeat
12:   $\mathbf{w} \leftarrow \mathbf{w}'$
13:   for $i \leftarrow 1$ to $k$ do
14:     for each $\mathbf{e}_i^j \in O_i$ do
15:       for each $\mathbf{e}_i^{j'} \in O_i$ do
16:         $\delta = \max\left(-a_i^j, \min\left(a_i^{j'}, \frac{(\ell_i^j - \ell_i^{j'}) - (\Delta\Psi_i^j - \Delta i\Psi_i^{j'}) \cdot \mathbf{w}'}{\|\Delta\Psi_i^j - \Delta\Psi_i^{j'}\|^2}\right)\right)$
17:         $\mathbf{w}' \leftarrow \mathbf{w}' + \delta \cdot (\Delta\Psi_i^j - \Delta\Psi_i^{j'})$
18:         $a_i^j \leftarrow a_i^j + \delta$
19:         $a_i^{j'} \leftarrow a_i^{j'} - \delta$
20: until $\|\mathbf{w}' - \mathbf{w}\| < \epsilon$
21: return $\mathbf{w}'$
```

2. Then, the simplex is being transformed until convergence. Convergence here means that the simplex becomes very small; that all points spanning the simplex are near each other.

   Letters and names refer to points in Figure 4.3 (next page), that illustrates the transformations for the two dimensional case. Each of the points spanning the simplex is evaluated using the error metric. The one with the highest error —the lowest BLEU— will be called *worst*. The one with the second highest error is (confusingly) called *good*. And naturally, *best* is the point with the lowest error.

   Let us define M as the midpoint between *best* and *good*, it will be used to define the other points in Figure 4.3 (following page):

   $$M = \frac{1}{2}(\mathit{best} + \mathit{good}) \tag{4.14}$$

   These transformation are considered in the order they are presented

   (a) E: reflection and expansion
   If a reflection and expansion of the *worst* point into E leads to a lower error score, then replace *worst* with E.

   $$E = M + 2 \cdot (M - \mathit{worst}) \tag{4.15}$$

   (b) R: reflection
   In case the expansion is not, but just the reflection into point R is an improvement over *worst*, then replace *worst* with R.

   $$R = M + (M - \mathit{worst}) \tag{4.16}$$

   (c) *C1*: contraction
   If *C1* is an improvement over both *worst* and *C2*, then replace it for *worst* and thus contract the simplex.

   $$\mathit{C1} = M + \frac{1}{2} \cdot (M - \mathit{worst}) \tag{4.17}$$

   (d) *C2*: contraction and reflection
   And the same for the reflection of *C1*. If *C2* is an improvement over both *worst* and *C1*, then replace it for *worst* and thus contract and reflect the simplex.

   $$\mathit{C1} = M + \frac{3}{2} \cdot (M - \mathit{worst}) \tag{4.18}$$

   (e) S and M: contraction in all dimensions
   If nothing of the above gave an improvement, than shrink all points halfway towards *best*. In the case of 2 dimensions, that means replacing *worst* with S and *good* with M.

   $$S = \frac{1}{2} \cdot (\mathit{best} + \mathit{worst}) \tag{4.19}$$

For Simplex in an algorithmic form see Algorithm 4 (p. 31). Again, notation related issues are as described briefly in section 4.1 (p. 20).

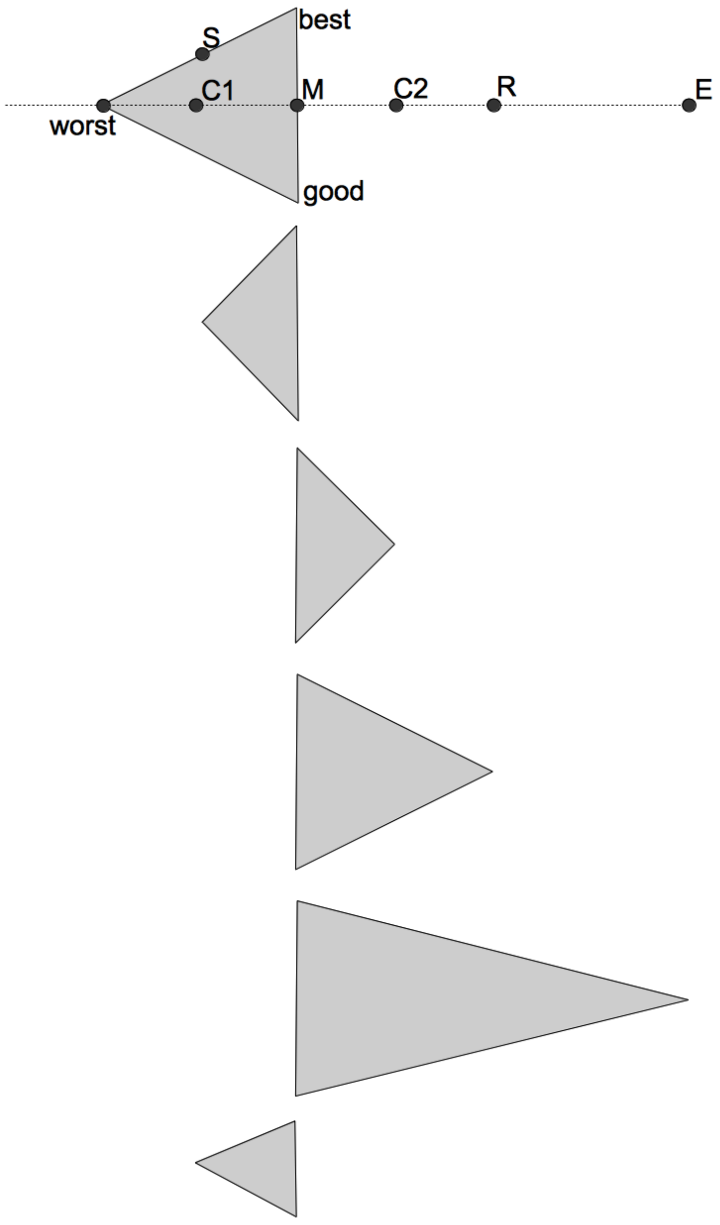


Figure 4.3: Illustration of the transformations of Simplex for a two dimensional (and thus simplified) problem. The points *best*, *good* and *worst* refer to the points with the lowest, second highest and highest error scores respectively.

Algorithm 4 $\textsc{OptimiseSimplex}(\mathbf{w}, \mathbf{E}^n, \mathbf{R})$

```
Require: $\mathbf{w}$ a starting point in weight-vector space
Require: $\mathbf{E}^n$ an $n$-best list $\mathbf{e}_1^1, \ldots, \mathbf{e}_1^n, \ldots, \mathbf{e}_k^1, \ldots, \mathbf{e}_k^n$
Require: $\mathbf{R}$ corresponding reference translations
 1: $\mathbf{W}_1 \leftarrow \mathbf{w}$
 2: $m \leftarrow |\mathbf{w}|$
 3: for $l \leftarrow 2$ to $m + 1$ do
 4:     $\mathbf{W}_l \sim U(\mathbf{w}_{min}, \mathbf{w}_{max})$
 5: repeat
 6:     order $\mathbf{W}$ such that:
 7:     $\textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}^{\text{worst}}}) < \textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}^{\text{good}}}) < \cdots < \textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}^{\text{best}}})$
 8:     $\mathbf{w}^{\text{m}} \leftarrow \frac{1}{2}(\mathbf{w}^{\text{worst}} + \mathbf{w}^{\text{best}})$
 9:     $\mathbf{w}^{\text{r}} \leftarrow \mathbf{w}^{\text{m}} + (\mathbf{w}^{\text{m}} - \mathbf{w}^{\text{worst}})$
10:     $\mathbf{w}^{\text{e}} \leftarrow \mathbf{w}^{\text{m}} + 2(\mathbf{w}^{\text{m}} - \mathbf{w}^{\text{worst}})$
11:     if $\textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}^{\text{e}}}) > \textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}^{\text{r}}}) > \textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}^{\text{worst}}})$ then
12:         $\mathbf{w}^{\text{worst}} \leftarrow \mathbf{w}^{\text{e}}$
13:     else if $\textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}^{\text{r}}}) > \textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}^{\text{worst}}})$ then
14:         $\mathbf{w}^{\text{worst}} \leftarrow \mathbf{w}^{\text{r}}$
15:     else
16:         $\mathbf{w}^{\text{c1}} \leftarrow \mathbf{w}^{\text{m}} + \frac{1}{2}(\mathbf{w}^{\text{m}} - \mathbf{w}^{\text{worst}})$
17:         $\mathbf{w}^{\text{c2}} \leftarrow \mathbf{w}^{\text{m}} + \frac{3}{2}(\mathbf{w}^{\text{m}} - \mathbf{w}^{\text{worst}})$
18:         if $\textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}^{\text{c1}}}) > \textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}^{\text{worst}}})$ and $\textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}^{\text{c1}}}) > \textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}^{\text{c2}}})$ then
19:             $\mathbf{w}^{\text{worst}} \leftarrow \mathbf{w}^{\text{c1}}$
20:         else if $\textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}^{\text{c2}}}) > \textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}^{\text{worst}}})$ and $\textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}^{\text{c2}}}) > \textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}^{\text{c1}}})$ then
21:             $\mathbf{w}^{\text{worst}} \leftarrow \mathbf{w}^{\text{c2}}$
22:         else
23:             $\mathbf{w}^{\text{worst}} \leftarrow \frac{1}{2}(\mathbf{w}^{\text{best}} + \mathbf{w}^{\text{worst}})$
24:             $\mathbf{w}^{\text{good}} \leftarrow \mathbf{w}^{\text{m}}$
25: until $\|\mathbf{w}^{\text{worst}} - \mathbf{w}^{\text{best}}\| < \epsilon$
26: return $\mathbf{w}^{\text{best}}$
```

Chapter 5

Theoretical Comparison of Existing Approaches to Tuning

In this chapter the three methods described in Chapter 4 (p. 20) are compared on the basis of some general properties that apply to Machine Learning techniques. We discuss how each method behaves with respect to this property and how they compare. We conclude this chapter with some thoughts and directions on where we see opportunities for improvements in section 5.4 (p. 37).

5.1 General Machine Learning Characteristics

Tuning, or black-box parameter optimisation in general, can be considered a subfield of Machine Learning. We are concerned with learning from example data and we aim at learning something that extends beyond those samples. Our goal is learning to generalise. Here we mention some characteristics associated with Machine Learning techniques and discuss our approaches in their light.

5.1.1 Exploration versus Exploitation

In Machine Learning there always is a trade off between exploration and exploitation. An exploring action would be one where there is no evidence that it would lead to any improvement; going into a direction in the weight-vector space while there is no indication that it would lead to a higher BLEU score. While an exploiting action would be one that exploits current knowledge of the nature of the search space. Such an action is aimed at fast success; on direct BLEU-score improvements.

The three methods we described all seem to focus more on exploitation —on climbing the hill— than they do on exploitation —on looking for other hills—. MERT explicitly focuses on finding optima, optimising dimension by dimension. Also Simplex is obviously only interested in progress with respect to the objective function, it keeps replacing its *worst* point by improvements. Both methods do however consider more points than just those that lead to improvements; they have to, considering the nature of the problem, see section 3.4 (p. 19). MIRA is a slightly different case, as it is an on-line algorithm. MIRA might actually update its weight-vector in a direction that is globally —according to corpus BLEU— not an improvement. So, MIRA does not perform explicit exploration actions, but might do so when it is following a training example that is an outlier.

All three methods at least partly rely on random restarts, or partial random initialisations in Simplex, for exploring behaviour.

5.1.2 Robustness

It is usually a desired property of a Machine Learning algorithm to be robust, meaning for example that its performance should not degrade quickly due to minor changes in the training data. Also, an algorithm might rely on random effects for exploration of the search space, which might render the algorithm unstable. Multiple runs of the same algorithm might then result in different weight-vectors and thus different BLEU scores.

Foster and Kuhn (2009); Arun et al. (2010) analyse the stability of MERT and report standard deviations over several runs of the algorithm. In general, MERT shows a standard deviation of 0.1 up to 0.3 BLEU points, which in SMT terms is big. No such studies of Simplex and MIRA within the setting of SMT are known, but will be performed in this work, see Chapter 7 (p. 46).

5.1.3 Convergence

Convergence criteria that decide whether a new iteration —including running the decoder and obtaining and merging an n-best lists— is required are the same for all three methods. This criteria is part of the unified framework presented in this work. These criteria are based on changes in the weight-vector, and the size of expansion of the merged n-best after the decoder was run. If neither shows a big change, the tuning is deemed converged.

On top of that MERT, MIRA and Simplex each have their own stopping criteria within the algorithm. Each of these methods is called with a merged n-best list and an initial weight-vector. Then they start their own internal iterations and have a stopping criteria to decide when to stop these iterations.

Simplex is stopped whenever a simplex became sufficiently small $\|\mathbf{w}^{\text{worst}} - \mathbf{w}^{\text{best}}\| < \epsilon$, with $\epsilon = 0.0001$.

MERT simply explores all dimensions and all points of interest per direction. No extra iterations are performed afterward, even though it might actually be worth exploring a dimension again after other dimensions have been changed.

MIRA is stopped from an infinite loop whenever the weight-vector is not changing much. In our particular implementation: if $\|\mathbf{w} - \mathbf{w}'\| < \epsilon$, again with $\epsilon = 0.0001$.

5.1.4 Over-fitting

Over-fitting is the process of modelling the peculiarities of a development set better than required. A model that is over-fitted performs worse when deployed on a test-set. The property can not always be ascribed to an optimisation method. It can also be a property of the data-sets. If a test set is very different from a development set, we can hardly expect a model to generalise well towards the test set. However, since we use all methods on the same data, in the same setting, we can compare our methods in how far we expect them to over-fit the development data. Also, sometimes the problem of over-fitting is defined as the model having too many degrees of freedom for the data it has to model, in such a case the model would replicate instead of summarise the data. This last way of over-fitting does not apply to our setting. All approaches to tuning are limited in that they are to produce a weight-vector of defined dimensions. These weight-vectors are generally not sufficiently high dimensional to summarise the training data.

As has been mentioned in this comparison, MIRA deploys an *averaging* strategy after all parallel runs of the algorithm to avoid over-fitting. It is easy to imagine how a single, highly optimised weight-vector —one that performs very well on a development set— would perform worse on a test-set; it probably sits on a high, narrow peak in the BLEU objective function. By averaging over a couple of good performing weight-vectors we arrive at a vector that is representative of an good area —not just a narrow peak— in weight-vector space. Such an approach leads to less over-fitting behaviour of MIRA.

Simplex, on the other hand, has such behaviour build in to the algorithm. It only replaces its *worst* points with higher ones while trying to expand and converges when all points that span the simplex are near each other; when all points have climbed up a peak that is big enough to house the simplex.

Cer et al. (2008) focus their work on preventing MERT from over-fitting, by introducing a regularisation method. Instead of choosing the maximum point, with respect to BLEU, per dimension they choose either a minimum BLEU score of a neighbourhood or the average of that neighbourhood. We did experiment with implementing these additions but did not observe any improvements, we also did not come across work of others that confirmed their findings.

On a side note, it is questionable whether over-fitting is possible at all with our log-linear model. Duh and Kirchhoff (2008) claim that this model is not expressive enough to fit the data, let alone over-fit.

5.2 Practical Issues

Many specialised optimisation methods showcase some properties that are there just because they gain improvements when applied to a specific problem. The *No Free Lunch Theorem for Optimisation* (Wolpert and Macready, 1997) explains why. The authors develop a theory that states that an optimisation technique that performs well on a certain problem has to pay with performance loss for other problems. Or, in other words, there is no such thing as a *general purpose optimisation technique*. One could read this as an excuse to start fine-tuning methods to better fit a problem, or to even over-fit a problem. Indeed, the authors observe that practitioners of optimisation deploy heuristic methods to match the right optimisation algorithm to a problem. While some adaptions might be theoretically justified, others seem peculiar and and are merely present because they have shown to give improvements in practice.

5.2.1 Initialisation

The initialisation of particularly MERT and MIRA involves some prior knowledge about what a good weight-vector should look like. However, since there are claims for at least MERT that random restarts help (Moore and Quirk, 2008), it is questionable in how far initialisation is important. Still, one could argue that it should not hurt performance when a method is initialised with a good starting point.

We choose to also initialise Simplex with the initial weight-vector we use for MERT and MIRA. In Simplex we simply use it as one of the initial points spanning the simplex.

5.2.2 Randomness

All our methods use random points in one way or another. In Simplex it is inherent to the algorithm, in MERT and MIRA it serves as a way to run extra instances of the algorithm.

MIRA and MERT both expect a initialised weight-vector $\mathbf{w}$. Simplex does strictly not need this and simply chooses $|\mathbf{w}| + 1$ random weight-vectors to start with. We do however feed Simplex with an initial weight-vector at the start of each iteration.

MERT takes 19 additional random weight-vectors as starting points (next to the initial vector), and does so in every iteration. Also, the order in which the weights in the weight-vector are being updated in MERT is a source of randomness, we do not consider the trial line searches to determine the order of updating.

In MIRA, since it is an on-line algorithm, the order in which examples (candidate translation pairs) are presented to the algorithm matters and can be a source of randomness. In fact, some approaches shuffle them in every iteration, we did not do this.

5.2.3 Peculiarities

In MERT the ranges of the 19 random additional points are biased, as is the choice for the number 19. Nowhere in the original literature there seems to be much of a justification for these numbers. Only recently, as described in our related work Chapter 2 (p. 7), Moore and Quirk (2008) investigate the effect of random restarts in MERT.

In MIRA the setting $C = 0.01$ (Chiang et al., 2008, 2009) seems arbitrary, even though we verified it on a held-out dataset, the authors do not give a satisfactory explanation.

Simplex uses heuristics to find a replacement for the *worst* point, their location and the order in which they are tried is heuristically defined. Although, when described in words for the two dimensional case the transformation seem intuitively sound, this does not guarantee that in higher dimensions this is also the case.

5.3 Other Properties

5.3.1 Objective Function

Tuning became only possible with the rise of automatic evaluation metrics. Many metrics have been developed, and so far BLEU seems to have won, see also section 3.3.1 (p. 15). MERT has been tried with many different metrics, variations of BLEU and completely different objective functions; those listed in section 3.3.2 (p. 18). Both MIRA and MERT have been seen using BleuS (Lin and Och, 2004), also called sBleu , which is a sentence based instead of corpus based metric. Liang et al. (2006) also apply a variant called *smoothed* BLEU in MERT and Watanabe et al. (2007b) apply a variant called *approximated* BLEU in MIRA.

Like MERT, Simplex can do with the normal, corpus based BLEU score. Since BLEU is the objective we will finally be judged by, it is also the metric we use for both these methods.

For MIRA, we choose to not use either BleuS or *approximated* BLEU and choose to follow Chiang et al. (2008) instead. Their method uses a moving average over previously translated sentences to form a pseudo document that can be used as a corpus to evaluate on using normal corpus based BLEU. Of course, this is still an approximation of BLEU.

We note here that it is definitely a huge pre for a method to be using the metric that will be the final judge as the objective function for optimisation directly (Cer et al., 2010).

Besides these issues, the number of times an objective function has to be evaluated is crucial if this is a computationally expensive computation, calculation corpus BLEU is rather expensive. Simplex needs much less such evaluations when compared to MERT (Koehn, 2008). For MIRA, of course, the story is different since BLEU is not calculated over the whole corpus and thus much cheaper per evaluation. MIRA does need an evaluation for each of the candidates in the oracle set though.

5.3.2 Parallelisation

All three methods seem on a first glance what is called *embarrassingly parallel* (Fox et al., 1994, p. 257). Meaning that it is almost trivial to speed up execution of the algorithm at no extra costs. This is mainly the case because all three methods can use random restarts —additional random points next to the initial starting point— in order to improve performance. Such restart are independent and can be run in parallel on separate processors.

For MERT, each of the 20 weight-vectors (19 random plus 1 initial)[1] can be distributed among processors. After each of these restarts reached convergence, we simply select the weight-vector among these 20 weight-vectors that resulted in the highest BLEU score.

MIRA could also simply have several independent parallel runs and average in the end. It was shown by McDonald et al. (2005) that averaging instead of taking the weight-vector for the maximum BLEU scores lead to less over-fitting. We follow Chiang et al. (2008) in doing so. Another option that was shown to be more effective for MIRA, would be to share information between processors. We did however not implement this variation. Note, however, the set of oracles —O in Algorithm 3 (p. 28)— is filled independent of weight-vectors, and will therefor be the same for each parallel run.

Also Simplex could be run independently in parallel on several processors and share the overall *best* point after every update. We have opted however to run several instantiations independently in parallel and take the weight-vector belonging to the run with the maximum BLEU score after each iteration, just like MERT.

All three methods would benefit in the same way from the speedup that parallelisation can offer.

5.3.3 Dimensionality

For MERT, the number of features that can be used successfully, the maximum dimensionality of the weight-vector, is rather low; around 15 to 30. MIRA can handle many more; up to millions, according to the authors that introduced the method (Chiang et al., 2009).

This property can be ascribed to the fact that MERT is updating and evaluating individual dimensions of the weight-vector. While MIRA updates the complete vector. Because of this, in higher dimensions, MERT will be much more likely to get stuck in local optima, where the one weight it was optimising has its peak. For some problems it might be necessary to vary in more than one dimension at a time to find an optimum. If the number of dimensions goes up, this scenario becomes more likely and thus methods that vary more dimension at a time can be expected to perform better.

But then, these claims would also hold for Simplex, which is also optimising all dimensions at once. However, Simplex is limited in its movements, and might not be able to explore a space that grows exponentially with each added dimension sufficiently enough.

5.4 Summary of Opportunities for Improved Tuning Approaches

Based on our considerations above, we can identify some properties that might be desirable for a tuning method, or at least worth an investigation.

[1]Note that the original MERT (Och, 2003) used 19 + 1 starting points, we use 23 + 1 starting points, given our 8 core hardware.

For one, none of the methods has an explicit exploration policy, given the ruggedness of our objective function, see Figure 3.1 (p. 12), we could expect a policy that is actively exploring the search space might perform better.

Also, one reason why a method might be unreliable —not so robust— is that it relies on random samples, and usually just a few. If we would increase the number of random samples, stability would go up. Of course, this could not be readily applied to MERT, MIRA or Simplex, because it would lead to just as much extra run-time, and trading run-time for performance is generally not desirable, see 3.2 (p. 11).

Current convergence criteria for MIRA and Simplex do not guarantee convergence at all. MERT does have a strict finite number of iterations, but their number is not known before run-time. In other words, it is unpredictable for how long each algorithm would run. We could think of fixing a maximum time, or a maximum number of iterations after which an algorithm should return. Although such a limitation might be convenient, we can not expect that it leads to improvements, except maybe for an early stopping effect that avoids over-fitting (Chiang et al., 2009).

It has been stated that a good starting point —a good initial weight-vector— is not a necessity. The reason has been ascribed to random restarts. Random restarts lead to improvements. Moreover, Simplex takes the majority of its starting points randomly. We might, as mentioned two paragraphs back, consider ways of taking many more random points.

Even though a local —a sentence based— evaluation function is cheaper, and can therefore be applied more often, corpus based BLEU should be preferred as the objective function during optimisation simply because this metric will also be the final judge.

Given the given recent progress in hardware (Sutter, 2005), showing a shift towards multiple processors per machine, we should definitely prefer methods that are *embarrassingly parallel*. And if possible even in a more fine-grained sense. This would allow an optimisation algorithm to distribute work to many processors in parallel.

Over-fitting issues for out-of-domain test sets —where the development set and test set are far apart in nature— can be addressed by not only considering top ranked sentences. All approaches so far are only dealing with either the 1-best sentence or with all other candidate translations with respect to the 1-best sentence. MERT and Simplex use the corpus BLEU score, only looking at the top sentence, while MIRA uses sentence BLEU score differences between some candidates and a *single* oracle. It might very well be beneficial to also start considering where runners up to the best sentence end up.

In the next chapter we will introduce two new approaches to tuning that will address the above opportunities.

Chapter 6

New Approaches to Tuning

This chapter will introduce two new approaches to tuning within the SMT setting based on our findings in Chapter 5 (p. 32) and in particular section 5.4 (p. 37).

Our initial motivation to start this research in the first place is the intuition that performing tuning in order to get the best translation on top could be improved by also aiming at getting the second best second on the second place, the third best third, etcetera. Such an approach would learn a model that should also do reasonably well in places where a source sentence does not match the training data, the data that we build our phrase table and language model on, as good as the development data did. In other words, it would do well where the best sentences are simply not available and we have to all back on second or even third best sentences. We would expect such a method that takes a ranking into account to perform better on so called *out of domain* test data.

But before diving into that investigation we were curious whether the often stated assumption that our search space is to big to explore in ordinary fashion, actually holds. What happens if we try a rigorously different approach of just repeatedly sampling our search space? We are encouraged with what is listed in section 5.4 (p. 37). An investigation follows next.

6.1 A Sampling Approach to Tuning

As has been mentioned in for example section 3.2 (p. 11) on tuning, iterating over all possible values for $\mathbf{w}$ to perform Equation 3.6 (p. 12) is not feasible since we are dealing with a real-valued space that suffers from the *curse of dimensionality* (Bellman, 1957, p. IX). We can however take samples from the weight-vector space and evaluate each of these samples. This seems a rather naive and impractical approach, but given the simplicity and the appeal that comes with simplicity, it should be tried. Also, our findings in the comparison of existing methods in the previous chapter encourages us to see where such an approach brings us. See section 5.4 (p. 37).

Many ways of performing sampling exist. It is far beyond the scope of this work to provide an overview. In his PhD thesis, Pedersen (2010) describes, proposes and fine-tunes some black-box optimisation techniques. Where he defines a black-box optimisation technique as one that does not use any gradient information but a fitness function instead to guide the search for optimal parameters. We have a problem that needs such an optimisation technique. One of the algorithms described by Pedersen (2010) is a straightforward algorithm called Local Unimodal Sampling (LUS). It is called that way because it was designed to, and does, perform well on unimodal problems.[1] Our problem seems distributed more or less unimodal, see Figure 3.1 (p. 12) and Appendix A (p. 76), at least when each dimension is considered individually. We do not claim this to hold in the combined high dimensional space of all features. For what it is worth, if these figures can show us anything, it is that our space is *not* distributed *non*-unimodal.

We proceed with a description of our sampling method, largely following Pedersen (2010, p. 14) and adapting his work to our setting where necessary.

1. The LUS method starts with an initial point in weight-vector space. This point is either the initial weight-vector, the best weight-vector from the previous iteration or chosen randomly, with a uniform distribution within limits (as was done for MERT, MIRA and Simplex).

   $$\mathbf{w} \sim U(\mathbf{w}_{min} - \mathbf{w}_{max}) \tag{6.1}$$

   Vector $\mathbf{w}_{min}$ denotes the lower bound of weight-vectors, $\mathbf{w}_{max}$ the upper bound.[2]

2. Set the initial sampling range $\mathbf{d}$ to be of the size of the initial weight-vector space.

   $$\mathbf{d} = \mathbf{w}_{max} - \mathbf{w}_{min} \tag{6.2}$$

3. Until termination, a fixed number of iterations i, repeat the following updates to either $\mathbf{w}$ or $\mathbf{d}$:

   (a) Pick a random direction vector $\mathbf{a}$ that will be used to move $\mathbf{w}$ to $\mathbf{w}'$:

   $$\mathbf{a} \sim U(-\mathbf{d}, \mathbf{d}) \tag{6.3}$$

   (b) So, add $\mathbf{a}$ to the current weight-vector $\mathbf{w}$ to create a new candidate weight-vector $\mathbf{w}'$:

   $$\mathbf{w}' = \mathbf{w} + \mathbf{a} \tag{6.4}$$

   (c) If the candidate performs better, if $\textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}'}) > \textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}})$, then move to the new position by setting

   $$\mathbf{w} = \mathbf{w}' \tag{6.5}$$

   otherwise decrease the sampling-range by multiplying with factor q, see (6.7):

   $$\mathbf{d} = q \cdot \mathbf{d} \tag{6.6}$$

4. Now $\mathbf{w}$ holds the best weight-vector we could find in the fixed number of iterations; after sampling i times.

[1] And apparently also because *lus* is Danish for *louse*.

[2] See section 7.1.1 (p. 46) for the ranges that define these bounds.

We are left with an undefined q, and we follow Pedersen (2010) when he describes the decreasing sampling range. In general, and at first, we sample from a space defined by our lower and upper bound for weight-vectors. In fact, we allow for our samples to go beyond those boundaries, nothing guarantees that we will end up between them.

But as soon as a sample does not improve the BLEU score; as soon as the candidate $\mathbf{w}'$ is worse than $\mathbf{w}$, we decrease the area in which we sample; we zoom in on $\mathbf{w}$. Pedersen (2010) defined q as depending on the dimension of the weight-vector space, $m = |\mathbf{w}|$, and α:

$$q \leftarrow 2^{-\frac{\alpha}{m}} \tag{6.7}$$

For a set of experiments we set $\alpha = 0$, to see the effect of sampling alone, without any zooming in. We also estimated α on a separate dataset,[3] and found that $\alpha = \frac{1}{100}$ gave us the most satisfying results, more so than the $\alpha = \frac{1}{3}$ suggested by Pedersen (2010). Making the learning rate dependent on the number of dimensions ensures that we do not zoom in too quickly for higher dimensions; for larger spaces. If we would zoom in rather quickly, we risk missing a better local optimum. On the other hand, if we would not zoom in at all, we would leave the sampling space to big, and our samples will turn out to be too sparse, especially for higher dimensions. So, we find the classical trade-off in Machine Learning between exploration and exploitation here.

We have our experiments in Chapter 7 (p. 46) using both $\text{LUS}_{\alpha=0}$ and $\text{LUS}_{\alpha=1/100}$.

As with other methods, it is trivial to parallelise this algorithm, it is so called *embarrassingly parallel*. We run our algorithm with a weight-vector from the previous iteration —or the initial weight-vector— on one core, while we take a random weight-vector, within limited ranges[4], for an additional 7 cores. Running each instance with $i = 50,000$, see Algorithm 5 (following page), gave us good results on a reserved portion of the separate dataset,[5] while staying within decent time limits. In total, each iteration involved 400,000 BLEU evaluations, which can be considered a lot and thus very expensive. On the other hand, our sampling approach involves hardly any other calculations.

For a more algorithmic listing of LUS, see Algorithm 5 (following page).

6.2 A Ranking Approach to Tuning

All existing methods, as mentioned in the previous chapter as well as our sampling approach, are focusing on getting a single sentence in an n-best list on top, see section 5.4 (p. 37). Generally this is the sentence with the highest BLEU score, or in the case of MIRA it is a combination of BLEU and model-score. It might very well be in such a method that the second best sentence —no matter how *best* is defined— is nowhere near the top. A weight-vector learned that way is likely to do well on the training/tuning sentences; it will be good at getting those

[3]Again, as with setting C in MIRA and SVM-Rank, this was done on MT02, LDC2003T18. We reserved a portion for estimating the sample size though.

[4]See section 7.1.1 (p. 46) for the specific ranges.

[5]We used the reserved portion of MT02, LDC2003T18. See also Figure 7.7 (p. 62) and its discussion.

Algorithm 5 OptimiseLus$_\alpha(\mathbf{w}, \mathbf{E}^n, \mathbf{R})$

```
Require: $\mathbf{w}$ a starting point in weight-vector space
Require: $\mathbf{E}^n$ an $n$-best list $\mathbf{e}_1^1, \ldots, \mathbf{e}_1^n, \ldots, \mathbf{e}_k^1, \ldots, \mathbf{e}_k^n$
Require: $\mathbf{R}$ corresponding reference translations
 1: $m \leftarrow |\mathbf{w}|$
 2: $q \leftarrow 2^{-\frac{\alpha}{m}}$
 3: $bleumax \leftarrow \textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}}, \mathbf{R})$
 4: $\mathbf{d} \leftarrow \mathbf{w}_{max} - \mathbf{w}_{min}$
 5: repeat
 6:   $\mathbf{a} \sim U(-\mathbf{d}, \mathbf{d})$
 7:   $\mathbf{w}' \leftarrow \mathbf{w} + \mathbf{a}$
 8:   if $\textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}'}, \mathbf{R}) > bleumax$ then
 9:     $bleumax \leftarrow \textsc{Bleu}(\tilde{\mathbf{E}}_{\mathbf{w}'}, \mathbf{R})$
10:     $\mathbf{w} \leftarrow \mathbf{w}'$
11:   else
12:     $\mathbf{d} \leftarrow q \cdot \mathbf{d}$
13: until repeated $i$ times
14: return $\mathbf{w}$
```

sentences on top. When tested on a different set, it might very well perform much worse.

To put it differently, a weight-vector that is learned using a method that is solely looking at single best translations will perform well in cases where such a good translation is constructible from the phrase table. When deployed in a setting where phrases are harder to construct, and where we can not even expect *good*, let alone *best*, translations, this vector might not perform good at all.

On the other hand, if we where to prefer not only the best but also some inferior translations to even lesser ones, we might end up with a weight-vector that performs good in the before mentioned situation.

This is the intuition behind our Ranking approach: we are not only concerned with getting the best translation on top, we rather aim at learning a weight-vector that sorts translations from best to worst; we are interested in the best *ranking*.

6.2.1 A Variety of Possible Ranking Approaches to Tuning

In the Information Retrieval field, many approaches to solving Ranking problems exist and many can be applied to our problem. Here we are presenting a range of ideas. Note that we will choose the pair-wise approach, and merely present the other approaches for reasons of completeness.

Rank Regression

One could simply apply Linear Regression techniques to predict a metric using the feature values on a candidate by candidate basis. And instead of the metric one could think of predicting the rank. A downside of such a method is that we will not be able to use previous or a priori knowledge; we can not incorporate a weight-vector from the previous iteration.

Another problem is that linear regression in general does not perform well in higher dimensional space, an issue we could address by using dimensionality reduction techniques as done in Principal Component Regression by (Hawkins, 1973).

Ranking Metric

A very different approach to obtaining a good ranking is to adjust the objective function to incorporate the quality of the ranking. In the Information Retrieval field such metrics are common. We could think of Precision at n (P@n) or Discounted Cumulative Gain (DCG). Such methods rely on a precision or relevance score for which we can use BLEU. At each rank we will need such a score. We simply take for rank 1 the normal BLEU score. For rank 2 we consider all candidates in all n-best lists that came second according to the model-score calculate BLEU on them as if they where the best, and so forth.

This new objective function could then be used together with any optimisation algorithm that uses a holistic metric, like MERT or Simplex.

Exploratory investigation into this method learned that the necessary BLEU evaluations to obtain a ranking metric were prohibitively expensive.

Pair-wise Ranking

Again a different approach would be the pair-wise approach where pairs of sentences are taken and the preference of one over the other is learned. When enough pairs are taken into consideration a ranking is learned implicitly. This is the route we take, and the approach is explained in detail below.

6.2.2 Our Ranking Approach to Tuning

An algorithm for performing this ranking is SVM-Rank introduced by Joachims (2002), and based on his older work in Joachims (1999). Many more pair-wise ranking algorithm exist, see Liu (2009, p. 258) for some examples, we favoured SVM-Rank for its good generalisation ability.

We can rephrase the words of Joachims (2002, p. 135), for the SMT tuning setting.

For a foreign sentence $\mathbf{f}_i$ and an n-best list $\mathbf{E}_i^n$ consisting of candidate translations into the target language, the optimal tuning system should return a ranking r^* that orders the translations in $\mathbf{E}_i^n$ according to their BLEU score.

Our system will not exactly return a ranking r^* but rather a weight-vector $\mathbf{w}^*$ that orders an n-best list when the dot-product with its features is taken according to r^*.

What follows is a reformulation of the SVM-Rank method to fit it into the tuning setting.

For each source sentence $\mathbf{f}_i \in \mathbf{F}$ we can construct an optimal ranking r_i^*, so that the translation candidates $\mathbf{e}_i^j \in \mathbf{E}_i^n$ are ordered according to a sentence based BLEU score. These BLEU scores are, of course, an approximation of the real BLEU score. We use the same pseudo-document variation as MIRA does.

We define our optimal ranking as follows:

$$r_i^* \subset \mathbf{E}_i^n \times \mathbf{E}_i^n \tag{6.8}$$

So, r_i^* is a subset of all possible translation pairs, with the following restriction:

$$\forall(\mathbf{e}_i^j, \mathbf{e}_i^l) \in r_i^* \Leftrightarrow \textsc{Bleu}(\mathbf{e}_i^j) > \textsc{Bleu}(\mathbf{e}_i^l) \tag{6.9}$$

We can now perform tuning by finding the weight-vector $\mathbf{w}$ that maximises the number of the following inequalities that is fulfilled:

$$\begin{gathered}\forall(\mathbf{e}_1^j, \mathbf{e}_1^l) \in r_1^* : \mathbf{w} \cdot \Psi(\mathbf{e}_1^j, \mathbf{f}_1) > \mathbf{w} \cdot \Psi(\mathbf{e}_1^l, \mathbf{f}_1) \\ \vdots \\ \forall(\mathbf{e}_i^j, \mathbf{e}_i^l) \in r_i^* : \mathbf{w} \cdot \Psi(\mathbf{e}_i^j, \mathbf{f}_i) > \mathbf{w} \cdot \Psi(\mathbf{e}_i^l, \mathbf{f}_i)\end{gathered} \tag{6.10}$$

That is, we are trying to find that $\mathbf{w}$ that gives us model-scores that obey the optimal rankings.

However, Hoffgen et al. (1995) proved this problem to be NP-hard. But not if a non-negative slack variable ξ for each translation pair is introduced, we can approximate the solution (Cortes and Vapnik, 1995) as follows:

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} \frac{1}{2}\mathbf{w} \cdot \mathbf{w} + C \sum \xi_{j,l,i} \tag{6.11}$$

subject to the constraints

$$\begin{gathered}\forall(\mathbf{e}_1^j, \mathbf{e}_1^l) \in r_1^* : \mathbf{w} \cdot \Psi(\mathbf{e}_1^j, \mathbf{f}_1) \geq \mathbf{w} \cdot \Psi(\mathbf{e}_1^l, \mathbf{f}_1) + 1 - \xi_{j,l,1} \\ \vdots \\ \forall(\mathbf{e}_i^j, \mathbf{e}_i^l) \in r_i^* : \mathbf{w} \cdot \Psi(\mathbf{e}_i^j, \mathbf{f}_i) \geq \mathbf{w} \cdot \Psi(\mathbf{e}_i^l, \mathbf{f}_i) + 1 - \xi_{j,l,i}\end{gathered} \tag{6.12}$$

and

$$\forall j \forall l \forall i : \xi_{j,l,i} \geq 0 \tag{6.13}$$

We can re-arrange (6.12) into the following constraint:

$$\forall j \forall l \forall i : \mathbf{w} \cdot (\Psi(\mathbf{e}_i^j, \mathbf{f}_i) - \Psi(\mathbf{e}_i^l, \mathbf{f}_i)) \geq 1 - \xi_{j,l,i} \tag{6.14}$$

Making our optimisation problem an ordinary classification problem on pair-wise difference vectors; difference vectors of the feature vectors of translation pairs: $\Psi(\mathbf{e}_i^j, \mathbf{f}_i) - \Psi(\mathbf{e}_i^l, \mathbf{f}_i)$. Solutions to this type of problems exist, in the form of SVM classifiers.[6] Such a classifier learns a model of which we can extract the weight-vector $\mathbf{w}$.

For a compact implementation, see Algorithm 6 (facing page). To make our solution computationally feasible, we had to set some limitations on the number of pairs we consider. Line 5 in our actual implementation is replaced by $j \leftarrow 1$ **to** 3 and Line 6 by $l \leftarrow j + 1$ **to** $j + 31$. Giving us 90 pairs per source sentence. More than three times as much as considered by MIRA. As in Algorithm 3

[6]We use SVM^{light} and refer to Joachims (1999) for implementation details.

Algorithm 6 OptimiseRank($\mathbf{w}, \mathbf{E}^n, \mathbf{R}$)

```
Require: $\mathbf{w}$ a starting point in weight-vector space
Require: $\mathbf{E}^n$ an $n$-best list $\mathbf{e}_1^1, \ldots, \mathbf{e}_1^n, \ldots, \mathbf{e}_k^1, \ldots, \mathbf{e}_k^n$
Require: $\mathbf{R}$ corresponding reference translations
 1: $t \leftarrow 1$
 2: for $i \leftarrow 1$ to $k$ do
 3:   order $\mathbf{E}_i^n$ such that:
 4:   $\textsc{Bleu}(\mathbf{e}_i^1) > \textsc{Bleu}(\mathbf{e}_i^2) > \cdots > \textsc{Bleu}(\mathbf{e}_i^n)$
 5:   for $j \leftarrow 1$ to $n-1$ do
 6:     for $l \leftarrow j+1$ to $n$ do
 7:       $\mathbf{d}^t \leftarrow \Psi(\mathbf{e}_i^j, \mathbf{f}_i) - \Psi(\mathbf{e}_i^l, \mathbf{f}_i)$
 8:       $\mathbf{c}^t \leftarrow 1$
 9:       $\mathbf{d}^{t+1} \leftarrow \Psi(\mathbf{e}_i^l, \mathbf{f}_i) - \Psi(\mathbf{e}_i^j, \mathbf{f}_i)$
10:       $\mathbf{c}^{t+1} \leftarrow -1$
11:       $t \leftarrow t+2$
12: $svm \leftarrow$SVMClassifier$(\mathbf{d}, \mathbf{c}, C)$
13: return $model(svm)$
```

(p. 28) we use C, and set it the same way as we do for MIRA, to $C = 0.04$ in this case.

In a sense, the SVM-Ranking approach is very similar to the approach of MIRA in section 4.2 (p. 23). The main difference is that MIRA only considers pairs with the oracle. It could be said that in this approach multiple oracles are used instead of just one. MIRA also takes the loss, the difference in Bleu scores, into account, in our setting a pair is simply ordered correctly or not, it is binary.

6.3 Summary of Advantages of Our New Approaches to Tuning

The generality and simplicity of the LUS method is appealing; it makes it difficult to introduce errors in actual implementations and easy to understand. Its exploratory nature makes the method fit for rugged surfaces as the one we are dealing with. The fact that the algorithm itself is computationally very cheap allows us to evaluate many points in weight-vector space. Also, the fact that the weight-vectors we consider are chosen relatively freely can bee seen as an advantage. We are not limited to changing one dimension at a time nor by a fixed set of transformations. The opportunities for parallelisation this algorithm provides us with are not exploited fully by us, but are definitely an asset.

The main selling point of our SVM-Rank algorithm has to be its consideration of runner ups to the best candidate translation, and the appeal that comes with this intuitively attractive approach.

In the next chapter we will verify whether we really made progress and thus whether the claimed advantages will indeed lead to improved performance. We will see that LUS$_{\alpha=1/100}$ will outperform any other method while SVM-Rank does not seem to be an improvement.

Chapter 7

Experiments

In this section we describe how our systems are evaluated empirically. The systems we will evaluate are our own implementations of existing algorithms MERT, MIRA, Simplex, and our own methods SVM-Rank and two variations of the LUS method: $\text{LUS}_{\alpha=1/00}$ and $\text{LUS}_{\alpha=0}$.

We (re)implemented all methods for our own understanding. Diving into the code forces one to care about the details. And often the details matter. Also, using our own implementation ensures us that methods are behaving the way they do because of properties of the method and not due to some side effect of wrapper scripts or libraries used.[1]

First we will detail our experimental setup in section 7.1 and then continue with the results of our methods deployed in this setup in section 7.2 (p. 50). In section 7.3 (p. 66) we discuss if and why results did not match our expectations.

7.1 Experimental Setup

For our experiments, we only consider Arabic to English translation. This section describes both the tools and the data we use for our experiments. We detail the settings used and the preprocessing we applied. Finally, in section 7.1.5 (p. 48) we describe our evaluation method and how we decide on statistical significance.

7.1.1 Decoder

All experiments are run with Moses (Koehn et al., 2007) as the decoder, using the default 8 features, listed in Table 7.1 (next page), (word-penalty, distortion-penalty, language model and 5 times translation model) and its default settings (*distortion limit* 6, *table limit* 20). We did not bother fiddling with settings too much since we are mostly interested in relative improvements and not so much in achieving the highest score possible. Although we acknowledge that improving on a low baseline is easier and less an achievement than doing so on a high one, we believe that our baseline is sufficiently competitive. All our n-best list were retrieved with $n = 500$, except when explicitly stated otherwise.

[1]An interested reader can contact the author to obtain a copy of the code.

Feature	Description	Initial Weight	Weight Range
l	$p(\mathbf{e})$ language model	0.5	(0,2)
w	$w(\mathbf{e}) = e^{\lvert\mathbf{e}\rvert}$ word penalty	-1	(-1,1)
d	$d(\mathbf{f}, \mathbf{e})$ distortion model	0.3	(0,2)
t-1	$p(\mathbf{f}\lvert\mathbf{e})$ phrase translation probability	0.2	(0,0.5)
t-2	$lex(\mathbf{f}\lvert\mathbf{e})$ lexical weighting	0.2	(0,0.5)
t-3	$p(\mathbf{e}\lvert\mathbf{f})$ phrase translation probability	0.2	(0,0.5)
t-4	$lex(\mathbf{e}\lvert\mathbf{f})$ lexical weighting	0.2	(0,0.5)
t-5	$e^1 = 2.718$ phrase penalty, fixed per phrase	0.2	(0,0.5)

Table 7.1: The 8 features with their short name and description, used for our default setting. The initial weights are Moses' defaults and denote how weights for each feature are initialised to obtain the first n-best list. Random weights, if they are used, are taken within the given range.

We also used the default ranges[2] in which weights for the features can be randomised, the so called *lambda* settings. See Table 7.1 for a listing. Note that weight-vectors are not restricted to stay within these ranges; only initial random weight-vector have to obey these limits.

7.1.2 Language Model

The language model is build using SRILM (Stolcke, 2002) toolkit. We applied Kneser-Ney smoothing (Kneser and Ney, 1995) and interpolation to build a 4-gram language model.[3] We did not apply any filtering of our language model. Although, in order to obtain smaller models while tuning on a fixed set of sentences, we did filter our phrase table, as described below. We then based on that filtered phrase table our language model in such a way that no n-gram included a word that did not occur in either target side or source side of our phrases table. This should, and did not, have any effect on performance except for a speedup.

n	#ngrams
1-gram	1383469
2-gram	25235571
3-gram	47567930
4-gram	85556331

Table 7.2: n-gram counts for the Language Model used in every experiment in this work.

The model was estimated on the English Gigaword corpus (Graff et al., 2007, LDC2007T07) and the non-overlapping parts of the English side of the parallel data, see section 7.1.3 (next page). The total corpus consists of about 33M lines, or about 800M words. Table 7.2 lists the size of the resulting language model.

[2]Default in the MERT implementation that comes with Moses.

[3]Our exact parameters are: `-order 4 -interpolate -kndiscount`.

LDC-catalog	Name	#lines	#words
LDC2004T18	Arabic-English Parallel News Part 1	88K	3.3M
LDC2006E25	GALE Y1 Arabic-English Parallel News	65K	2.4M
LDC2005E46	Treebank Arabic-English Translation	25K	0.7M
LDC2004E72	eTIRR Arabic-English News	4K	92K
LDC2007T08	ISI Arabic-English Automatically Extracted	160K	4.5M
Total		343K	11.1M

Table 7.3: Datasets used to build the Phrase Table. The counts are only on the english —target— side of the parallel text. The texts on the target side, except for LDC2007T08, were added to the training data for the language model.

7.1.3 Phrase table

For training the phrase table, the standard tools that come with Moses are used. So Giza++ (Och and Ney, 2000) is used for alignments with the default `grow-diag-final` heuristic to refine word alignments. Phrases up to 7 words are extracted and scored.

As bi-texts we used LDC data, see Table 7.3. The resulting Phrase table consist of a total of 7.6M phrases.

When deployed in the tuning setting, the sentences that are to be translated are fixed. We can therefore get a speedup for free —we don't pay with worse translations— if we filter our phrase table to only include phrases that will, or could, actually be used. Any source phrases that do not occur in the dataset we are to translate can be omitted from the phrase table.

7.1.4 Lexicalised Reordering

Since 8 features is rather small compared to some other work in the field, we also trained a Lexicalised Reordering model (Tillmann, 2004; Kumar and Byrne, 2005; Koehn et al., 2005), adding another 6 features and thus weights. We used the default *msd-bidirectional-fe* configuration, see Table 7.4 (facing page). Where it is noted explicitly, we added these additional features, mainly to test how tuning methods perform with a larger number of weights.

7.1.5 Evaluation

As has been the convention for some years now, we use BLEU (Papineni et al., 2001) as the metric for evaluating our experiments. For an explanation of how BLEU works and a more theoretical discussion of BLEU see section 3.3.1 (p. 15). The standard NIST `mteval-v11b.pl` script is used to score the output of the decoder against the reference sentences. We report the cumulative 4-gram BLEU-score and do not consider any casing (we do not perform re-casing or true-casing) since that would unnecessarily complicate our pipeline and would not be of any help in judging the difference between tuning methods.

For tuning the MT04 dataset is used, while the resulting parameters are tested

Feature	Description
d-1	$p(monotone_{previous}\|\mathbf{f},\mathbf{e})$, the probability this phrase is monotone with the previous phrase.
d-2	$p(swap_{previous}\|\mathbf{f},\mathbf{e})$, the probability this phrase is swapped with the previous phrase.
d-3	$p(discontinuous_{previous}\|\mathbf{f},\mathbf{e})$, the probability this phrase is discontinuous with the previous phrase.
d-4	$p(monotone_{next}\|\mathbf{f},\mathbf{e})$, the probability this phrase is monotone with the next phrase.
d-5	$p(swap_{next}\|\mathbf{f},\mathbf{e})$, the probability this phrase is swapped with the next phrase.
d-6	$p(discontinuous_{next}\|\mathbf{f},\mathbf{e})$, the probability this phrase is discontinuous with the next phrase.

Table 7.4: The additional 6 features with their short name and description. Note that each probability is conditioned on both source and target phrase; on both $\mathbf{e}$ and $\mathbf{f}$. Theses features are always used in conjunction with the features in Table 7.1 (p. 47). Weights for these features are initialised uniform, with $\mathbf{w}_i = 0.3$ and are randomised in the range $(0.0, 2.0)$

	LDC-catalog	Name	#lines	#references
Dev	LDC2006E44	MT04	1353	5
Test	LDC2006E39	MT05	1056	5
	LDC2009T05	MT06	1797	4
		MT08	1360	4
Combined			5566	4

Table 7.5: Evaluation datasets. All experiments are trained on MT04 and tested on MT05, MT06 and MT08. In the combination of the four sets, we have only considered the first 4 references for each sentence.

on MT05, MT06 and MT08. The volumes and LDC-catalog numbers are listed in Table 7.5. Throughout this thesis we will use the short names (MT04, MT05, etc.) to refer to these datasets.

MT04 and MT05 have 5 references for each sentence possibly causing higher BLEU-scores compared to MT06 and MT08, which have only 4 references per sentence.

Both MT04 and MT05 mainly consist of news-wire, while MT06 and MT08 are mostly web-data. When tuning on MT04, it can be expected that BLEU scores for MT05 are much higher than BLEU scores on MT06 and MT08.

Our experiments are run 4 times, unless stated otherwise, and the reported BLEU-scores are averaged over these runs. Also the standard deviation between BLEU-scores of these runs is reported and should be taken as an indication of the stability of the methods, as was done recently by Arun et al. (2010).

Statistical Significance

BLEU scores alone are generally not enough evidence to prefer a certain weight-vector over another. To compare two tuning approaches, it is enough to compare the weight-vectors that resulted from deploying these approaches to the same setting. So, when we talk about comparing two systems that is equivalent to saying that we compare two weight-vectors, two methods or two approaches.

Koehn (2004) proposes a method for deciding on which of two system, and thus weight-vectors, is significantly better using Paired Bootstrap Resampling.

The proposed procedure, and the procedure we use is the following.

1. Choose two systems for comparison.
2. Use the combination of all datasets mentioned in Table 7.5 (page before).
3. Repeatedly, 100 times, do the following.
   (a) Take a random sample with replacements of 300 or 3000 sentences.
   (b) This sample is again 1000 times re-sampled with replacement.
   (c) Each of these 1000 sets is translated using both systems and evaluated using BLEU[4].
   (d) We note the number of times each system has the higher BLEU score.
4. The percentage of times, out of 100, that one system outperforms the other with at least 0.95 percent is reported.

Koehn (2004) showed that if the sample size grows, that there are progressively less errors in the decisions made by this procedure. We have chosen not to take all sample sizes used by Koehn, but to only consider samples of sizes 300 and 3000. In fact, he showed that a sample size of 300 was sufficient for judging one system significantly better than another for his setting. In that case the 100 repetitions would also not be necessary. However, given that we are dealing with a different setting, we decided to partly repeat his experiments and perform the repeated sampling with different sampling sizes.

7.2 Results

This section lists, describes and interprets result obtained from running each of the 5 methods[5] described in Chapter 4 (p. 20) and Chapter 6 (p. 39) using the setup described above.

In section 7.2.2 (p. 54) we investigate the performance of each method individually. In section 7.2.3 (p. 63) we compare all systems to each other to see which system outperforms the others, and whether a gain in performance is significant. Then we will list and discuss some weight-vectors that result from training with our methods in section 7.2.4 (p. 65). However, first we will look at some absolute BLEU scores.

[4]We did not actually translate the sentences for each sample, we precomputed our translations at the start of the procedure.

[5]We run and report result for LUS with α set to $\alpha = 1/100$ and some experiments with $\alpha = 0$.

Dataset		Existing methods			New methods		
		MERT	MIRA	Simplex	LUS$_{\alpha=1/100}$	LUS$_{\alpha=0}$	SVM-Rank
Dev	MT04	0.4802 *0.0020*	0.4550 *0.0000*	0.4386 *0.0198*	**0.4849** *0.0006*	0.4545 *0.0085*	0.4402 *0.0000*
Test	MT05	**0.5335** *0.0030*	0.4883 *0.0000*	0.5170 *0.0068*	0.5284 *0.0006*	0.5064 *0.0243*	0.5319 *0.0000*
	MT06	0.3852 *0.0031*	0.3643 *0.0000*	0.3512 *0.0170*	**0.3874** *0.0018*	0.3656 *0.0050*	0.3515 *0.0000*
	MT08	0.3736 *0.0017*	0.3492 *0.0000*	0.3469 *0.0151*	**0.3801** *0.0009*	0.3601 *0.0077*	0.3505 *0.0000*

Table 7.6: BLEU scores are obtained by running the decoder with weight-vectors that resulted when each algorithm was optimised on MT04. BLEU scores are averaged over 4 runs, the highest scores per dataset are printed **bold**, the standard deviation over the four runs is printed in *italic*.

7.2.1 Comparing Machine Learning Characteristics

BLEU scores for all our methods, and two variations of the LUS method, are listed in Table 7.6 for the default 8 feature case and in Table 7.7 (following page) for our extended 14 feature scenario.

Both tables list BLEU scores obtained after running each of the 6 algorithms until convergence. Then, the resulting weight-vectors are used to translate each of the datasets, and each set of translations is compared to the relevant reference translations. Per algorithm, this procedure is repeated 4 times and BLEU scores are averaged over 4 runs. Also the standard deviation over these 4 runs is reported. The first table shows scores for the default 8 features as described in section 7.1.1 (p. 46). The second shows results for when we added the 6 additional described in section 7.1.4 (p. 48).

Let us discuss the results listed in both Tables 7.6 and 7.7 (following page) together. The following sections denote the empirical characteristics —the behaviour— of our methods that we can extract from these results. We do so by discussing the relevant Machine Learning characteristics, as mentioned in section 5.1 (p. 32).

Over-fitting and Robustness

One of the most striking result when looking at Table 7.6, is the lack of generalising ability of MIRA, when compared to Simplex and SVM-Rank, or any other method for that matter. Or, put the other way around, the incredible generalisation abilities of Simplex and SVM-Rank, when compared to MIRA. MIRA does considerable better than Simplex on de development set (+1.65 BLEU points[6]), but on the most similar test set, MT05, MIRA does worse than Simplex (-2.86 points). On the other test sets MIRA does outperform Simplex,

[6]We follow the convention in SMT literature to call an increase from a BLEU score of 0.4385 to 0.4550 an increase of 1.65 BLEU points.

Dataset		Existing methods			New methods		
		MERT	MIRA	Simplex	LUS$_{\alpha=1/100}$	LUS$_{\alpha=0}$	SVM-Rank
Dev	MT04	0.4889	0.4575	0.4497	**0.4930**	0.4487	0.4477
		0.0000	*0.0000*	*0.0065*	*0.0018*	*0.0202*	*0.0000*
Test	MT05	**0.5327**	0.5045	0.5009	0.5324	0.5065	0.5263
		0.0000	*0.0000*	*0.0100*	*0.0021*	*0.0246*	*0.0000*
	MT06	0.3919	0.3683	0.3635	**0.3952**	0.3662	0.3582
		0.0000	*0.0000*	*0.0048*	*0.0018*	*0.0200*	*0.0000*
	MT08	0.3823	0.3589	0.3557	**0.3867**	0.3573	0.3573
		0.0000	*0.0000*	*0.0050*	*0.0019*	*0.0192*	*0.0000*

Table 7.7: A table similar to Table 7.6 (preceding page), using 6 extra lexicalized reordering features, see section 7.1.4 (p. 48). BLEU scores are obtained by running the decoder with weight-vectors that resulted when each algorithm was optimised on MT04. BLEU scores are averaged over 4 runs, the highest scores per dataset are printed **bold**, the standard deviation over the four runs is printed in *italic*.

but not as much as would be expected given the gain on the development set. We might be able to at least partly blame this on the instability of Simplex that has a standard deviation of 1.47 BLEU points on average, making it an unreliable method to compare another method to. In that sense, MERT for example is much more stable with an average standard deviation of 0.24 BLEU points, and thus better suited to serve as a basis for comparison. But also compared against MERT, MIRA shows an absence of generalisation on MT05; MIRA does 2.52 points worse than MERT on the development set, and with 4.52 points, much worse on MT05. Again, on the other two test sets, MIRA does seem to generalise slightly.

This over-fitting behaviour of MIRA is definitely most present in the results of Table 7.6 (page before), but the trend is also visible in Table 7.7. While we just stated that Simplex seems to generalise well, this does not show in Figure 7.1 (next page). That graph shows the objective function of our five methods, calculated on the n-best lists. We see that Simplex is greedily climbing the function while both MERT and LUS stabilise after only 4 iterations. MIRA and SVM-Rank are both not explicitly optimising this objective corpus BLEU function, and this visible; both show that the corpus BLEU is dropping almost every iteration.

When turning to our new methods we have to mention our surprise at the performance of LUS$_{\alpha=1/100}$. This sampling based method is outperforming any other method both on our development set and on 2 out of 3 test sets. Not just in BLEU scores, which is achievement, but also in stability. This method is able to score higher than any other method while doing so more reliable. This is not just the case, as one might expect, on our 8 feature test case, also with more features our zooming sampling method outperforms all other methods most of the time, see Table 7.7. It is still MERT that performs best on MT05, but LUS$_{\alpha=1/100}$ comes sufficiently close (only 0.03 BLEU points behind), making up by performing better on the development set as well as transferring the learned

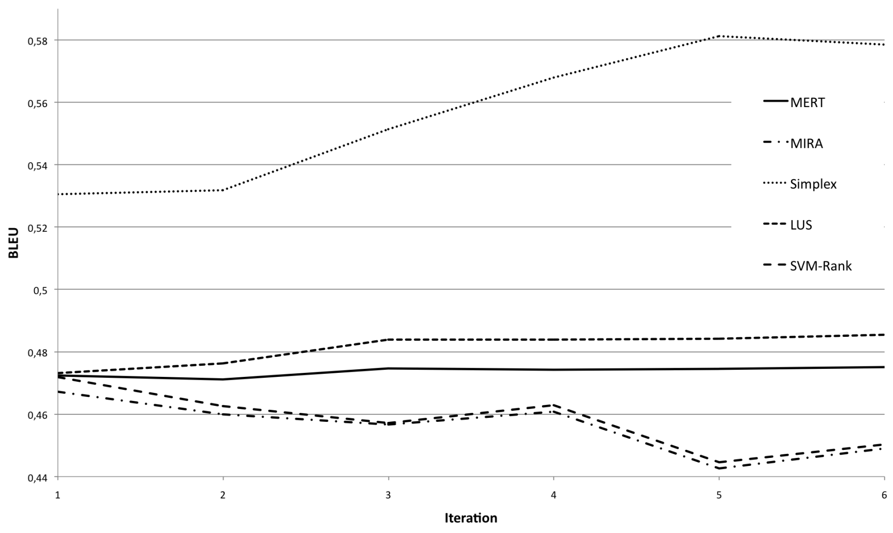


Figure 7.1: Internal objective function for each of the methods over 6 iterations, measured while tuning on the development dataset MT04. Since both MIRA and SVM-Rank do not explicitly optimise corpus BLEU, it is calculated separatly for the resulting weight-vector on the n-best list after every iteration. LUS here refers to $\mathrm{LUS}_{\alpha=1/100}$.

knowledge to the test sets.

Exploration versus Exploitation

Another eye-catcher is the variation in standard deviations. Some are mentioned already. We have two extremes here. On the one hand MIRA and SVM-Rank with no variation at all. Every run these methods behaves just the same. While Simplex very much seem to depend on initial and additional random points and $\mathrm{LUS}_{\alpha=0}$ is also very depended on the specific random weight-vectors selected in a run. Both properties are undesirable: the stiffness of MIRA and SVM-Rank keep it from exploring much of the weight-vector space, while the flexibility of Simplex and $\mathrm{LUS}_{\alpha=0}$ make it unpredictable methods. MIRA and SVM-Rank exploit too much, Simplex and $\mathrm{LUS}_{\alpha=0}$ do so too little and vice versa. The other methods, MERT and $\mathrm{LUS}_{\alpha=1/100}$ seem to have found a better balance between the two.

There has to be a limit to what one can do using a sampling method. It will suffer from the *curse of dimensionality*: as the dimensionality goes up, the size of the weight-vector space grows exponentially. Since we will still be bounded by a certain amount of maximum time we can spent on tuning, we can not allow the number of samples to grow exponential as well. The effect will be that our samples of the space will become sparser. This might lead to the expectation that performance will drop if the dimensionality, the number of features, goes up. In fact, when we look at the performance of $\mathrm{LUS}_{\alpha=0}$, that is exactly what happens. Using the default 8 features, this method performs better than when using de additional 6 lexicalised reordering features (+0.58, -0.01, +0.06 and

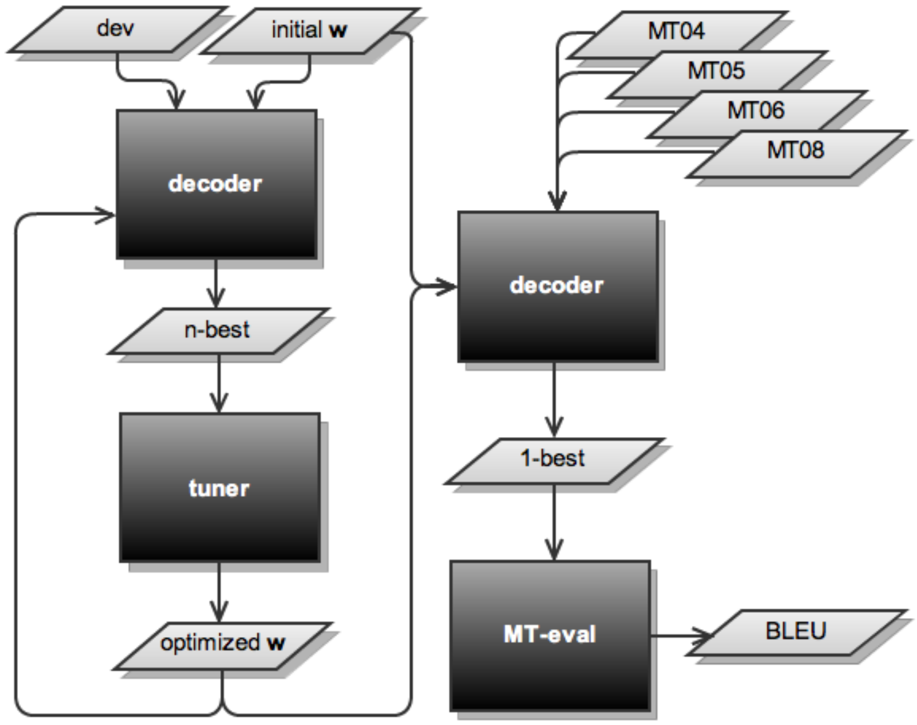


Figure 7.2: The process that generates graphs like Figure 7.3 (p. 56). The decoder is re-run for each dataset with each optimised weight-vector. This whole process is repeated 4 times for each method.

+0.28 BLEU points). This performance drop can definitely be contributed to the *curse of dimensionality*; none of our other methods performs worse when the extra features are added.

However, that is what happens if α is kept to 0, and therefore the learning rate parameter q is kept to 1. In other words, if the we are not zooming in on the current optimal solution. On the other hand, if we set α to a fraction, and to $\frac{1}{100}$ in particular[7], then q goes to $2^{-\frac{\alpha}{m}}$ where m is the dimensionality. So, for $m = 14$, learning rate $q \approx 0.999505$ and for $m = 8$, $q \approx 0.999134$. This causes the space $\mathbf{d}$, around the current optimal point, in which we sample to shrink slowly. We zoom in towards the currently found optimum, and in higher dimensions we do so on a slower pace, since our samples are more sparse and we are thus more likely to have missed a higher local optimum we should zoom in on.

It turns out, this strategy works in practice. The $\text{LUS}_{\alpha=1/100}$ always outperforms $\text{LUS}_{\alpha=0}$ with 2 to 3.5 BLEU points, which in SMT terms is a huge improvement. We will discuss these matters further in section 7.2.2 (p. 60).

7.2.2 Behaviour of Individual Approaches

In this section we discuss the results for each method individually, in the sense that we do not focus on comparing the approaches. An explanation of the graphs

[7]See also section 6.1 (p. 39).

and tables in this section is in place.

We show graphs as in Figure 7.3 (next page). What we see in such a graph is the progress over iterations of a method —MERT in this case— on all four the datasets. Each line denotes the progress, in BLEU score, on its respective dataset. These scores are obtained by running the decoder on the respective dataset after they where tuned on MT04, our development set. We did that for our development set itself as well. This explains the drop in performance on the development set, after the first iteration. The objective function that the tuner is optimising might point at higher BLEU scores as in Figure 7.1 (p. 53), however these scores are computed on a re-scored n-best list, not by re-running the decoder. On the left vertical axis the initial BLEU score is depicted, this is the BLEU score obtained using the initial weight-vector, see section 7.1.1 (p. 46), these initial scores are the same for all graphs of this kind. The BLEU score on the right vertical axis denotes final performance of the method on the respective dataset. This process is depicted in Figure 7.2 (facing page), and is repeated 4 times to obtain 4 BLEU points per dataset per iteration. The standard deviation is depicted in the graphs by a gray area around the mean.

Besides this graph, each method is discussed on basis of a table like Table 7.8 (next page). Such a table illustrates how 8 candidate translations are ordered by the respective method. Below each iteration number, the numbers denote how each candidate translation is ranked by the system after that iteration converged. The candidate translations are sorted in the table by their final ranking, the last column before the BLEU score. Ideally the BLEU score should show a descending order. However, keep in mind that BLEU scores are generally calculated on the basis of a corpus, not a single sentence. This means that while tuning, the performance on one sentence might go down simply because it goes up on average. Also, the BLEU score we denote in these tables is Bleu_2, for reason described in section 3.3.1 (p. 15). Note that sometimes the table mentions less iterations than the graph. In those cases we cut the table off if the order of the translations did not change any more. Since we used the same candidate translations for each method, to allow for easier comparison, sometimes a translation is missing if it was not present in the merged n-best list. We used the same sentence as in Table 3.1 (p. 14) which originates from the MT04 dataset.

Performance and Behaviour of MERT

See the beginning of section 7.2.2 (facing page) for an explanation of how graphs like Figure 7.3 (next page) are generated. What we observe is that the BLEU score for each dataset shows a dip, right after the first iteration, most likely due to an n-best list that is too small and not representative enough. However, MERT recovers quickly, at least as far as the performance on both MT04 and MT05 is concerned. On those two, relatively similar, datasets the evaluation metric quickly shows improvement and stability, with a standard deviation that becomes smaller. MT06 and MT08 however need more iterations with training on MT04 before they show improvement above initial scores. This can probably be blamed on the different nature of those two datasets, different from MT04. MERT needs to generalise more in order to perform well on them, and needs more iterations and presumably a larger n-best list to do so.

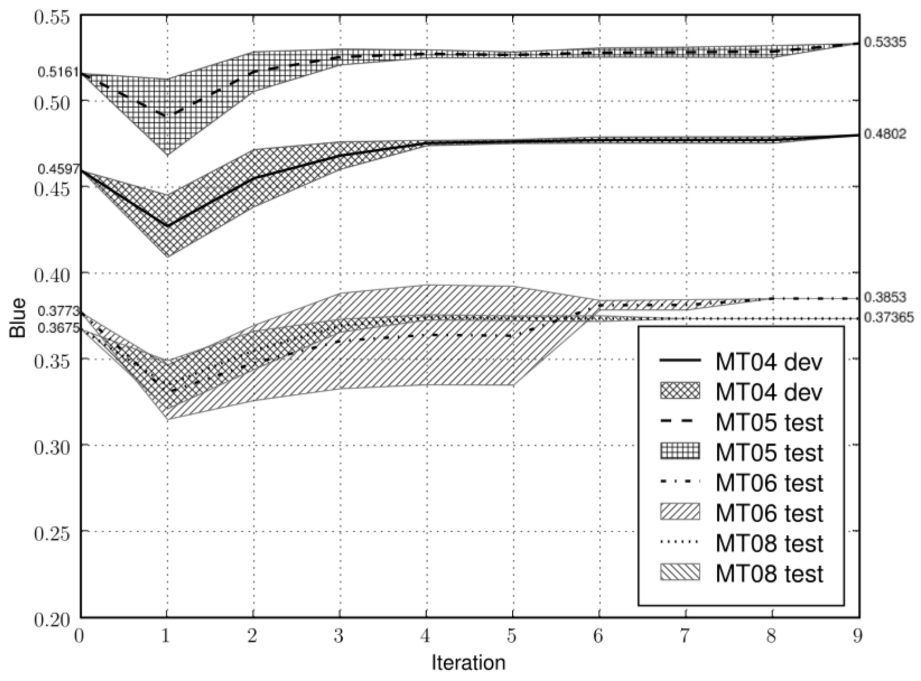


Figure 7.3: Result obtained over 4 runs of the MERT tuner. Lines denote the mean, the gray area around each line the standard deviation.

Candidate Translations	Iterations					BLEU
	1	2	3	4	5	
dutch jaap de hoop scheffer assumed his functions on top of nato	1	1	1	3	1	0.6396
dutchman jaap de hoop scheffer assumed his duties at top of nato	5	3	2	2	2	0.8165
dutchman jaap de hoop scheffer holds its function to head nato	6	2	3	1	3	0.5045
jaap de hoop scheffer dutch assumed his functions on top of nato	2	4	4	5	4	0.5839
dutch jaap de hoop scheffer on top of nato assumed its tasks	3	6	6	8	5	0.6030
dutchman jaap de hoop scheffer undertakes its tasks ras on nato	4	5	5	7	6	0.4671
dutchman jaap de hoop scheffer , will take office at the head of nato	7	7	7	4	7	0.5447
dutchman jaap de hoop scheffer , will take office at the head of the north atlantic treaty organization	8	8	8	6	8	0.4042
dutchman jaap de hoop scheffer takes over post at top of atlantic organization						

Table 7.8: Ranking of an example n-best list is changing over 6 iterations of MERT. In the perfect case, the BLEU scores would be ordered descending.

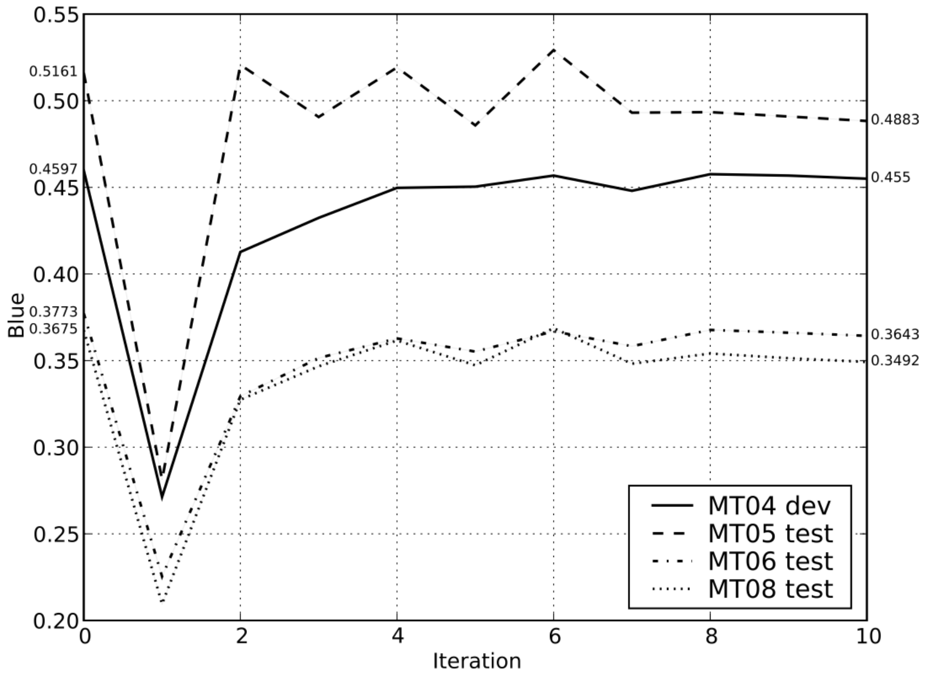


Figure 7.4: Result obtained over 4 runs of the MIRA tuner. Lines denote the mean, given that MIRA has a standard deviation of 0, it can not be seen.

In Table 7.8 (facing page), of which the workings are also introduced in the beginning of section 7.2.2 (p. 54), we see that the order in which the candidate translations are ranked does not change anymore after just 5 iterations. Looking back at the line of MT04 in Figure 7.3 (facing page) this seems an expected result; the figure shows a flat line. Overall, MERT performs well on this particular example; translations with high BLEU scores are near the top, those with lower scores near the bottom.

Performance and Behaviour of MIRA

The performance MIRA is rather different from that of MERT. MIRA is in fact performing disappointingly bad, BLEU scores after convergence are below initial scores, see Figure 7.4. Even on the development dataset. However, if we would have stopped earlier, after 6 iterations, we would have observed slight progress, especially on MT05, showing MIRA's apparent generalisation capabilities. It would, however, have been difficult to stop earlier because MIRA was still making improvements on the development set; the optimisation was not converged yet. A reason for this behaviour might be that MIRA is not exploring enough; the method is stuck in a local optimum. As MIRA is gathering more and more evidence —in the form of n-best lists— of the shape of the area in weight-vector space around this hill, it finds that the hill is not as high as it expected, but it does not have the exploratory characteristics to do anything about it and find the bigger hill. This performance can be blamed on the fact the MIRA is relying on the sentence based BLEU. It has no sense of the bigger picture.

Candidate Translations	Iterations										BLEU
	1	2	3	4	5	6	7	8	9	10	
dutchman jaap de hoop scheffer , assumed office at the head of nato	1	1	1	1	1	1	1	1	1	1	0.5883
dutchman jaap de hoop scheffer , assumed his duties at the head of the north atlantic treaty organization	5	5	5	5	5	4	4	4	4	2	0.4851
the dutch jaap de hoop scheffer , assumed his post at the head of nato	2	2	2	2	2	2	2	2	2	3	0.6055
dutchman jaap de hoop scheffer , will take office at the head of nato	3	3	3	3	3	3	3	3	3	4	0.5447
dutchman jaap de hoop scheffer holds his post at the head of the north atlantic treaty organization	7	7	7	7	7	6	6	6	6	5	0.5145
dutchman jaap de hoop scheffer , will take office on the top of the north atlantic treaty organization	8	8	8	8	8	8	7	7	8	6	0.3835
dutchman jaap de hoop scheffer assume office at the head of nato	4	4	4	4	4	5	5	5	5	7	0.6396
dutchman jaap de hoop scheffer take over his duties at the head of nato	6	6	6	6	6	7	8	8	7	8	0.7263
dutchman jaap de hoop scheffer takes over post at top of atlantic organization											

Table 7.9: Ranking of an example n-best list is changing over 10 iterations of MIRA. In the perfect case, the BLEU scores would be ordered descending.

Another feature of MIRA worth mentioning is standard deviation of 0; all runs perform exactly the same. A desirable but unexpected effect given the random restarts. This is partly due to the fact that MIRA bases its optimisation on the set O, which is chosen independent of a specific $\mathbf{w}$. In other words, MIRA always uses the same set of sentence pairs to update a weight-vector $\mathbf{w}$ on, and apparently the update rule completely overwhelms the initial value of the weight-vector it was optimising.

Table 7.9 shows how MIRA is putting the better translation candidates at the bottom of the list instead of at the top. Of course this could be a peculiarity of this single sentence, but given the lack of BLEU score improvements this can hardly be expected.

Looking back at Tables 7.6 (p. 51) and 7.7 (p. 52), we see that MIRA does not show much improvement when more features are added. MIRA was developed with a large number of features in mind (Chiang et al., 2009) while our result shows that MIRA is the one approach that gaining much from the added features. One explanation might be the very different nature of features we added when compared to the work of Chiang et al. (2009). Most features in their work are binary or at least nominal, where the features we add are probabilities and thus of a continuous nature.

Performance and Behaviour of Simplex

The enormous drop in BLEU score of the Simplex method after one iteration is characteristic for this method, see Figure 7.5 (next page). None of our other methods shows a drop this deep. It seems that whatever knowledge was fed to the method using an initial weight-vector values is lost right at the start. In a

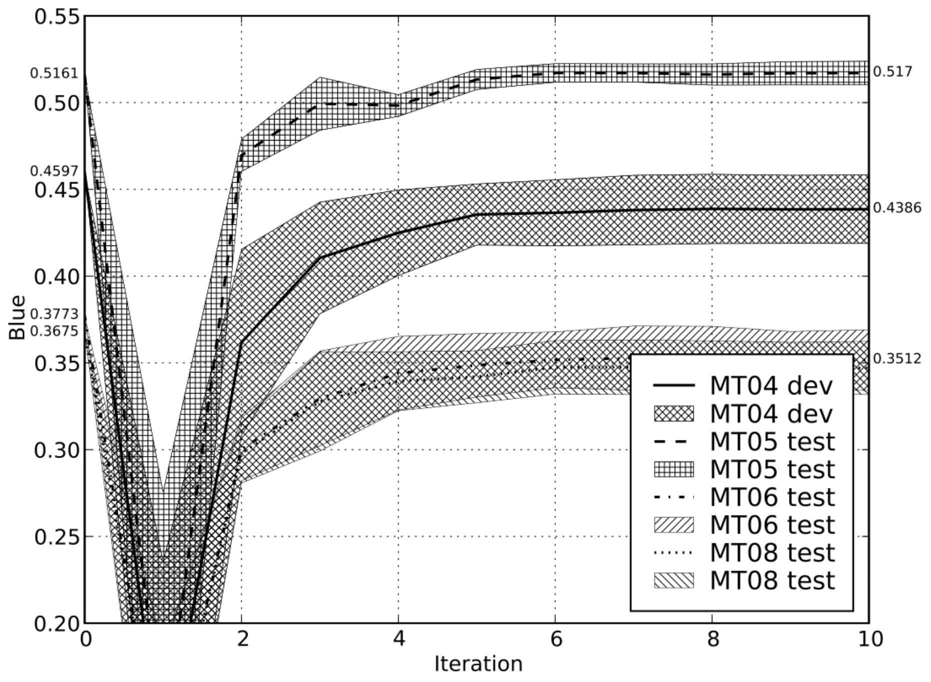


Figure 7.5: Result obtained over 4 runs of the Simplex tuner. Lines denote the mean, the gray area around each line the standard deviation. The final results for MT08 is left out of the figure, but is 0.3469.

Candidate Translations	Iterations					BLEU
	1	2	3	4	5	
dutchman jaap de hoop scheffer assumed his duties of nato	3	1	1	1	1	0.7746
the dutch jaap de hoop scheffer assume his duties at the head of nato	2	5	5	3	2	0.6504
dutchman jaap de hoop scheffer take office at the head of nato	5	4	2	2	3	0.6396
the dutch jaap de hoop scheffer assumed office to head nato	1	3	4	4	4	0.5045
dutchman jaap de hoop scheffer will function to head nato	4	2	3	5	5	0.5578
dutchman jaap de hoop scheffer will take office at the head of nato	6	6	6	6	6	0.5883
dutchman jaap de hoop scheffer , will take office at the head of the north atlantic treaty organization	7	7	7	7	7	0.4042
dutchman jaap de hoop scheffer takes over post at top of atlantic organization						

Table 7.10: Ranking of an example n-best list is changing over 5 iterations of Simplex . In the perfect case, the BLEU scores would be ordered descending.

Candidate Translations	Iterations					BLEU
	1	2	3	4	5	
the dutch jaap de hoop scheffer assume his duties at the head of nato	2	3	1	1	1	0.6504
dutchman jaap de hoop scheffer take office at the head of nato	5	1	2	2	2	0.6396
the dutch jaap de hoop scheffer assume the duties of nato	1	5	3	3	3	0.6030
the dutch jaap de hoop scheffer assumed office to head nato	3	6	4	4	4	0.5045
dutchman jaap de hoop scheffer assumed his duties of nato	4	2	5	5	5	0.7746
dutchman jaap de hoop scheffer will function to head nato	6	4	6	6	6	0.5578
dutchman jaap de hoop scheffer will take office at the head of nato	7	7	7	7	7	0.5883
dutchman jaap de hoop scheffer , will take office at the head of the north atlantic treaty organization	8	8	8	8	8	0.4042
dutchman jaap de hoop scheffer takes over post at top of atlantic organization						

Table 7.11: Ranking of an example n-best list is changing over 4 iterations of $\text{LUS}_{\alpha=1/100}$. In the perfect case, the BLEU scores would be ordered descending.

sense this seems like a waste: we did not feed the initial weight-vector without a reason, it holds a direction in weight-vector space that has been proved to work reasonable. On the other hand, it also means that Simplex is not dependent on a specific initialisation. When we also take the graph in Figure 7.1 (p. 53) into account, we see the reason for the performance drop. The algorithm is rather greedy, as we have mentioned before. Simplex rapidly finds the most optimal point for an n-best list, but this weight-vector turns out to be far from optimal when used to rerun the decoder; BLEU scores drop to almost zero. Also note the wide variation in BLEU scores on even the development dataset after convergence.

On our example translation in Table 7.10 (page before), Simplex does well. The worst candidate translation is at the bottom, the best at the top.

Performance and Behaviour of $\text{LUS}_{\alpha=1/100}$

As with most tuning approaches we discuss, LUS showcases the steep drop in performance right after the first iteration, see Figure 7.6 (facing page). Not as steep as with Simplex, but the BLEU score drops below the level of MERT, MIRA and SVM-Rank. On top of that, the standard deviation at that point is enormous; much larger than with any other method. It takes $\text{LUS}_{\alpha=1/100}$ two more iterations to recover from these two blows to its performance and to perform above initial level. After that, performance shows a stable but small increase up the sixth iteration where $\text{LUS}_{\alpha=1/100}$ converges.

Striking is the fact that the shape of both the mean and the standard deviation for all datasets is similar. MT06 and MT08 are lower, MT05 higher than the development set, but their shape is almost the same. This points to great generalisation capabilities. While the small standard deviation after just 4 iterations points at stability.

Even though $\text{LUS}_{\alpha=1/100}$ seems to be our best performing approach to tuning, it is not extraordinary good at getting the ranking of our candidate translations right. See Table 7.11.

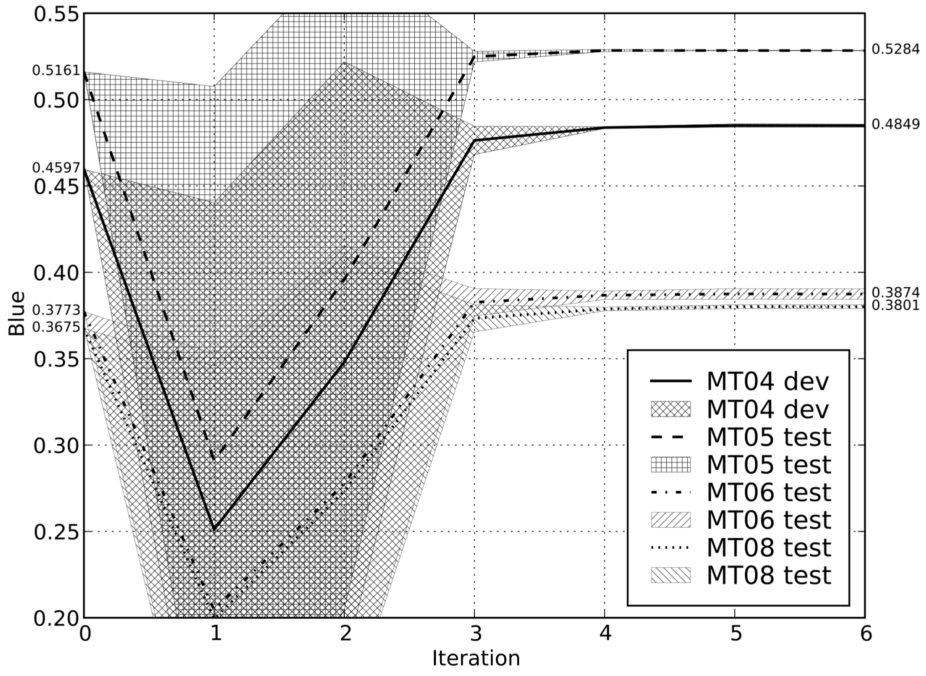


Figure 7.6: Result obtained over 4 runs of the $\mathrm{LUS}_{\alpha=1/100}$ tuner. Lines denote the mean, the gray area the standard deviation.

Using a separate dataset, we estimated that a sample size of around 50,000 samples per iteration would give the best result. We only used our default 8 feature setting to perform that estimation, and were interested to see what would happen if we varied the samples size for both the 8 feature case but also for the extended 14 feature case. We run this experiment only on our development set, MT04, and results are listed in Figure 7.7 (following page).

For the 8 feature case, the result agree with our estimation; 50,000 seems to be the right number of samples. Surprisingly though, for the 14 feature setting the peak in the graph lies around only 1,000 samples. While BLEU scores for the higher dimensional setting are generally higher, this is mainly the case the early peak up to 20,000 samples. For 100,000 samples both settings perform on par with respect to the BLEU score. This result is counter intuitive; one would expect that a higher dimensional space would benefit more from a larger sample since the space grows exponential with the number of dimensions and the samples would grow sparser if their number would not increase at the same rate.

One could argue that basing the above discussion only on experiments performed on the development set is not conclusive. Since, even though runs with larger samples perform worse on MT04 —our development set— this might be a feature of the algorithm that avoids over-fitting. In other words, it would be a strength of the algorithm if performance on the development dataset went down while it went up on test sets. However, we tested 4 of the resulting weight-vectors from this experiment on MT05, MT06 and MT08 as well, observing the same trends as in Figure 7.7 (next page).

Beyond the discussion of which number of samples is the right one, we might want to look at other convergence criteria for LUS$_{\alpha=1/100}$. One explanation of the effect observed in Figure 7.7 and discussed above might be the fact that the learning rate q in the LUS algorithm is not dependent on the number of samples. When a large number of samples are expected, one might prefer a smaller learning rate, to zoom in at a slower pace. Since this learning rate —this zooming in— is only applied when a new sample is worse than the best point so far, zooming in will happen exponentially more often if the search space is larger due to added dimensions. In higher dimensional spaces the probability for a new sample to be worse than the best point is much larger. A solution to this problem is either setting a different convergence criteria based on how far we zoomed in; the size of **d**. Or making q dependent on the number of samples.

Performance and Behaviour of SVM-Rank

As with MIRA, SVM-Rank harms performance. On three out of four datasets the BLEU score drops when comparing initial results with the final results. Surprisingly it is not the development dataset but MT05 that sees an improvement of 1.58 BLEU points. Performance on the other three sets drops with 1.6 unto 2.58 BLEU points. Reasons behind this behaviour should be sought in the same direction as with MIRA. Again, this method is not optimising directly on the corpus BLEU score, but on a local —sentence based— version of it. So, we have more evidence that such a metric is not helpful when trying to optimise the holistic variant. For none of the methods early stopping would have helped out in this case, even though they all perform best after just three iterations.

Also, as with MIRA, the standard deviation is 0 despite the random restarts. This should, however, not come as a surprise here as our OptimiseRank algorithm, see Algorithm 6 (p. 45) is not using the initial weight-vector at all. The algorithm only uses the n-best list for optimisation and this n-best list is the same for each restart.

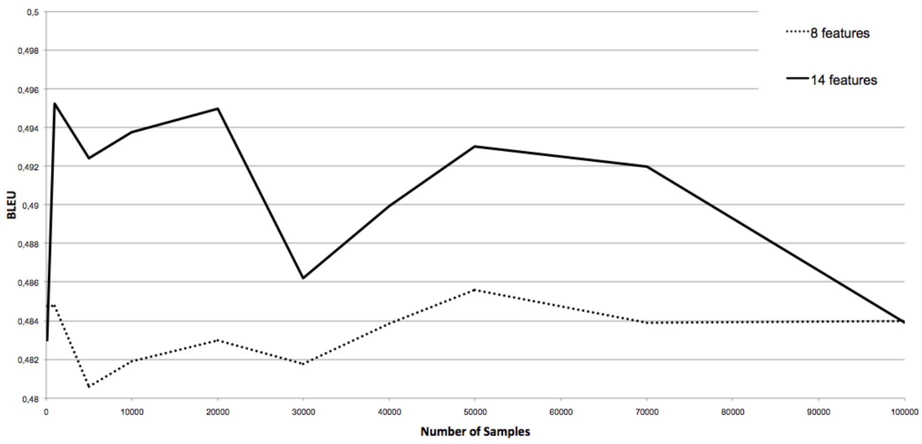


Figure 7.7: BLEU performance of LUS$_{\alpha=1/100}$ on MT04 for both our default 8 feature setting, and the 14 feature setting when the sample size is varied. Scores are obtained by rerunning the decoder, not by reranking an n-best list. The sample size of 50,000 was chosen for the 8 feature setting, albeit on a different dataset, and does indeed perform well for this default setting.

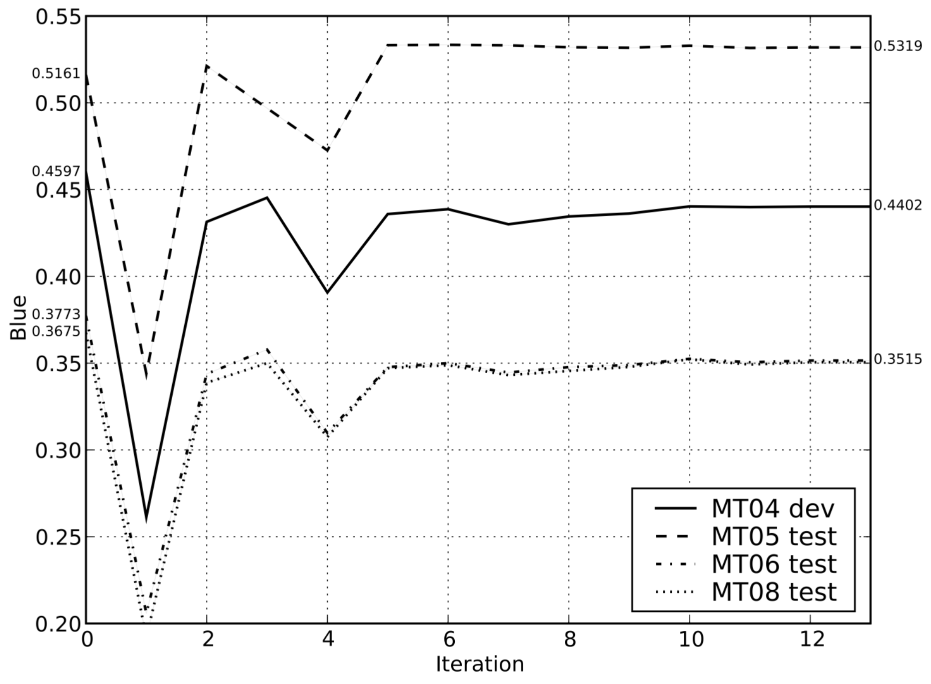


Figure 7.8: Result obtained over 4 runs of the SVM-Rank tuner. Lines denote the mean, however, no deviation was observed, so also no standard deviation is depicted.

7.2.3 Statistical Comparison of Approaches

As was mentioned before, in 7.1.5 (p. 50) and by Koehn (2004), it is impossible to base preference for one system over another just on BLEU score differences. In the sense that it is hard to tell how much of a difference in BLEU score is sufficient to speak of a *significant* difference in the statistical sense. Koehn introduced a method, using Paired Bootstrap Resampling, as described in section 7.1.5 (p. 50), that allows for drawing valid and strong conclusions.

We compared each of our systems pairwise with each other system by using the same weight-vectors as in section 7.2.1 (p. 51) but now deployed on the combination of all datasets, see section 7.1.5 (p. 48). The difference in BLEU score is noted, but more importantly the fractions of times one system outperforms the other for a samples size of 300 and 3000. Table 7.12 (following page) list the results.

From this table, we can conclude that MERT outperforms any other method[8] although it is a very close call with $\text{LUS}_{\alpha=1/100}$. Together, these two methods perform significantly better, no matter the sample size, than any other method.

Given the results in Table 7.12 (next page), we can order our systems as follows, from best to worst:

[8]Note that we are using our own implementation of MERT but that it was compared, using the same method, to the standard implementation of MERT that is distributed with MOSES. Our version of MERT did slightly but not significantly better: 0.05/0.06 on a sample size of 3000.

System comparison (conclusion)	**BLEU difference**	**Sample Size**	
		300	3000
MERT better than Simplex	0.0373	1.0	1.0
MERT better than $\text{LUS}_{\alpha=0}$	0.0296	1.0	1.0
MERT better than SVM-Rank	0.0241	1.0	1.0
MERT better than MIRA	0.0124	0.75	1.0
MERT better than $\text{LUS}_{\alpha=1/100}$	0.0013	0.15/0.01	0.37
$\text{LUS}_{\alpha=1/100}$ better than Simplex	0.0360	1.0	1.0
$\text{LUS}_{\alpha=1/100}$ better than $\text{LUS}_{\alpha=0}$	0.0283	1.0	1.0
$\text{LUS}_{\alpha=1/100}$ better than SVM-Rank	0.0228	0.98	1.0
$\text{LUS}_{\alpha=1/100}$ better than MIRA	0.0200	0.99	1.0
MIRA better than Simplex	0.0144	0.77	1.0
MIRA better than $\text{LUS}_{\alpha=0}$	0.0112	0.51	1.0
MIRA better than SVM-Rank	0.0057	0.23	0.73
SVM-Rank better than Simplex	0.0132	1.0	1.0
SVM-Rank better than $\text{LUS}_{\alpha=0}$	0.0055	0.26	0.86
$\text{LUS}_{\alpha=0}$ better than Simplex	0.0077	0.24	0.89

Table 7.12: We show, following Koehn (2004, Table 3) as described in section 7.1.5 (p. 50), how often a conclusion with $p \leq 0.05$ is made for different system comparisons and sample size 300 and 3000. 0.15/0.01 means 15% correct and 1% wrong conclusions. 5566 test sentences from Table 7.5 (p. 49) are resampled with replacement 100 times into samples of size 300 and 3000.

$$\text{MERT} \geq \text{LUS}_{\alpha=1/100} > \text{MIRA} \geq \text{SVM-Rank} \geq \text{LUS}_{\alpha=0} \geq \text{Simplex}^{9}$$

Statistical Comparison with More Features

The above results were obtained using our default 8 features. Below we investigate what happens if we add our additional 6 feature of the lexicalised reordering model, see section 7.1.4 (p. 48). We performed the exact same experiment as above and obtained Table 7.13 (next page).

Things have changed: $\text{LUS}_{\alpha=1/100}$ and Simplex moved up at the cost of MERT and $\text{LUS}_{\alpha=0}$ respectively. And MIRA moved down in the hierarchy making place for SVM-Rank, $\text{LUS}_{\alpha=0}$ and Simplex.

We can thus order our systems when applied to a setting with more features as follows:

$$\text{LUS}_{\alpha=1/100} \geq \text{MERT} \geq \text{SVM-Rank} > \text{Simplex} > \text{LUS}_{\alpha=0} > \text{MIRA}$$

Here, the effect of the learning rate α on LUS shows strongly: $\text{LUS}_{\alpha=1/100}$ is performing best of all methods while $\text{LUS}_{\alpha=0}$ is far down the ladder. Note also how our new method SVM-Rank is outperforming existing methods Simplex

[9] We use the notation "sys1 > sys2" to denote that sys1 is significantly better than sys2, and "sys1 ≥ sys2" to denote that sys1 would show significantly better than sys2 have we had a larger sample size, see Koehn (2004).

System comparison (conclusion)	**BLEU difference**	**Sample Size**	
		300	3000
$\text{LUS}_{\alpha=1/100}$ better than MIRA	0.1219	1.0	1.0
$\text{LUS}_{\alpha=1/100}$ better than $\text{LUS}_{\alpha=0}$	0.0460	1.0	1.0
$\text{LUS}_{\alpha=1/100}$ better than Simplex	0.0204	1.0	1.0
$\text{LUS}_{\alpha=1/100}$ better than SVM-Rank	0.0056	0.3	0.9
$\text{LUS}_{\alpha=1/100}$ better than MERT	0.0018	0.2/0.01	0.42
MERT better than MIRA	0.1201	1.0	1.0
MERT better than $\text{LUS}_{\alpha=0}$	0.0442	1.0	1.0
MERT better than Simplex	0.0186	1.0	1.0
MERT better than SVM-Rank	0.0038	0.26	0.64
SVM-Rank better than MIRA	0.1163	1.0	1.0
SVM-Rank better than $\text{LUS}_{\alpha=0}$	0.0404	1.0	1.0
SVM-Rank better than Simplex	0.0148	0.84	1.0
Simplex better than MIRA	0.1015	1.0	1.0
Simplex better than $\text{LUS}_{\alpha=0}$	0.0256	1.0	1.0
$\text{LUS}_{\alpha=0}$ better than MIRA	0.0759	1.0	1.0

Table 7.13: Just like Table 7.12 (facing page), but using 6 extra lexicalised reordering features, see section 7.1.4 (p. 48). We show, following Koehn (2004, Table 3) as described in section 7.1.5 (p. 50), how often a conclusion with $p \leq 0.05$ is made for different system comparisons and sample size 300 and 3000. 0.2/0.01 means 20% correct and 1% wrong conclusions. 5566 test sentences from Table 7.5 (p. 49) are resampled with replacement 100 times into samples of size 300 and 3000.

and MIRA. Perhaps most noteworthy, although covert in section 7.2.2 (p. 57), is the failure of MIRA to cope with more features.

7.2.4 Resulting Weight-Vectors

Of course, the work in this thesis is more about methods of finding the optimal weight-vector, and not so much about the resulting weight-vectors themselves. It is, however, still instructive to see and discuss the result of these methods, once they are converged. The final weight-vectors, that are the result of applying each of the methods in this thesis, are listed in Table 7.14 (p. 67).

Looking at the weight-vector, we can see some trends. *a*) The word-penalty (w) is always negative, while *b*) the phrase-penalty (t-5) is negative about half the time. *c*) All other weights are almost always positive, with one exception; and *d*) translation table weights (t-1 up to t-4) are mostly within range (0.2, 0.5).

However, what is probably most striking is that the two best performing systems —MERT and $\text{LUS}_{\alpha=1/100}$— produce very different weight-vectors, hinting at the ruggedness of the optimisation surface.

Also note that the weight-vectors always have the absolute magnitude of maximum 1, this is because we rescale them back to this magnitude. We can to that without harming performance, see section 3.1 (p. 10). Partly due to this scaling, and partly despite the scaling many weights did not stay within the range they where sampled in. This is, however, not surprising. We did not require nor did we build in any mechanism to keep the weight-vectors within these limits. We perform the scaling to make weight-vectors comparable, and to prevent that dot products with a weight-vector that has a high magnitude would cause an overflow.

7.3 Error Analysis

This short section aims at describing where and why experimental results did not live up to our theoretical expectations.

For one, MIRA does worse when more features are added, see section 7.2.3 (p. 64). While the algorithm was designed with many more features in mind than we added. In fact, that might already be the reason. MIRA excels in very high dimensional spaces, where the features outnumber the training examples. Also, the type of features we added might not be of the right kind. Chiang et al. (2009) add nominal and even binary features. In a sense we might not have subjected MIRA to a fair comparison by only considering the situations MERT was designed for.

Also SVM-Rank, as it is a similar algorithm to MIRA, might suffer from the same drawbacks. For this algorithm though, we also had to limit the number of candidate translation pairs in order to make our algorithm feasible. These restrictions dictated by time limits, definitely harmed performance.

The discussion in section 7.2.2 (p. 60), on the effect of the sample size of our $\text{LUS}_{\alpha=1/100}$ method, deserves a place here. The fact that a larger sample size in a higher dimensional space hurts performance is the opposite from what we expected when we developed our algorithm. What happens is, that in higher dimensional spaces less weight-vectors leading to an improvement are found, so that the algorithm zooms in too quickly on an inferior optimum in weight-vector space. When the algorithms zooms in too far —by sampling too often— it over-fits the n-best list on an inferior optimum leading to worse performance, even on the development set.

Other areas of expected but not observed results, are more difficult to qualify, because our experiments where not aimed at them. The fact that regularisation in MERT, as proposed by Cer et al. (2008), did not lead to improvements is noteworthy but not explainable by our experiments.

System	#Iteration	Time	Features													
			d-1	d-2	d-3	d-4	d-5	d-6	d	l	w	t-1	t-2	t-3	t-4	t-5
			(0, 2)	(0, 2)	(0, 2)	(0, 2)	(0, 2)	(0, 2)	(0, 2)	(0, 2)	(-1, 1)	(0, .5)	(0, .5)	(0, .5)	(0, .5)	(0, .5)
MERT	4	4:09							0.541	0.610	-0.863	0.444	0.705	0.327	0.430	1.000
	3	3:11	0.486	0.513	0.403	0.555	0.413	0.482	0.454	0.672	-1.000	0.392	0.814	0.547	0.273	0.275
MIRA	14	28:16							0.982	1.000	-0.002	0.327	0.278	0.266	0.267	0.273
	13	29:21	0.739	0.653	0.970	1.000	0.764	0.860	0.630	0.900	-0.007	0.200	0.219	0.230	0.183	-0.019
Simplex	6	2:30							1.000	0.449	-0.521	0.314	0.287	0.296	0.294	-0.331
	5	1:57	0.343	0.657	0.802	1.000	0.410	0.269	0.760	0.410	-0.383	0.326	0.266	0.078	0.263	0.143
SVM-Rank	12	23:30							0.314	0.596	-1.000	0.021	0.577	0.458	0.394	-0.110
	12	28:10	0.209	0.429	-0.041	0.329	0.365	0.303	0.630	0.417	-0.729	0.193	0.095	0.265	0.252	1.000
LUS$_{\alpha=1/100}$	5	4:06							0.351	0.408	-1.000	0.157	0.436	0.309	0.292	-0.385
	6	9:32	0.105	0.090	0.022	1.000	0.298	0.411	0.146	0.291	-0.761	0.171	0.237	0.233	0.274	0.232
LUS$_{\alpha=0}$	6	2:36							0.616	0.657	-1.000	0.373	0.622	0.465	0.486	-0.839
	5	2:01	0.628	0.683	0.457	0.591	0.543	0.155	1.000	0.427	-0.260	0.277	0.137	0.247	0.053	-0.403

Table 7.14: Some examples of weight-vectors produced by the systems discussed in this thesis. These weight-vectors are the result after the listed number of iterations and running for the listed amount of time. Feature names refer to those in Table 7.1 (p. 47) and Table 7.4 (p. 49). Ranges for random weights are reprinted right below the feature names.

Chapter 8

Conclusions

8.1 Summary

In this thesis we set out on exploring all aspects of tuning within the Statistical Machine Translation (SMT) setting. Our investigation ranged from introducing a uniform framework for tuning —optimising the weights of the log-linear model— to recasting three existing methods in that framework. Comparing these three methods led us to insights on which we based two new methods: Local Unimodal Sampling (LUS) and SVM-Rank. Finally we compared all five approaches to tuning empirically by running and evaluating experiments.

We were guided by the following three research questions. See section 1.3 (p. 4) for an elaboration.

1. What factors of Tuning in SMT make it a hard problem?
2. How do existing Tuning methods for SMT compare?
3. What factors lead to improvements in new Tuning methods for SMT?

Question 1 has largely been answered in Chapter 3 (p. 10), which is summarised in section 3.4 (p. 19). We can conclude that tuning in SMT is difficult because it is a black-box optimisation problem with some peculiarities. Computing the gradient of the objective function is impossible. Our objective function —BLUE— is holistic, making evaluation computationally expensive. Besides, the piece-wise constant nature of our problem prohibits indirect inference of a gradient. Furthermore, running a decoder —which is necessary for perfect evaluation— is prohibitively expensive, forcing us to perform approximated evaluations instead.

The second question is discussed and answered in Chapter 5 (p. 32) describing a comparison and in Chapter 7 (p. 46) describing our experiments. We conclude that MERT outperforms both MIRA and Simplex significantly. While MIRA performs significantly better than Simplex on our default 8 feature setting and vice versa on our extended 14 feature setting. Both MERT and Simplex use the holistic BLEU function as their optimisation objective, giving them an advantage over MIRA which relies on a sentence based approximation. The reason why Simplex is not performing on par with MERT is that Simplex is too aggressively climbing the first hill it encounters. One of the keys to the success of MERT are its random restarts. MIRA's standard deviation is 0, it does nothing with a random restart.

Our last question is answered throughout chapter 5, 6 and 7. The comparison in Chapter 5 (p. 32) led up to a discussion of opportunities for improvements in its summarising section 5.4 (p. 37). Of which some were realised in our two new approaches to tuning, as described in Chapter 6 (p. 39). An empirical comparison in Chapter 7 (p. 46) tells us that our LUS approach indeed is an improvement over the state-of-the-art. While our other new approach —SVM-Rank— does not give us the expected progress. We attribute the success of LUS to its exploratory qualities which are well in balance with the exploitation of directions in weight-vector space that proved fruitful. The failure of our SVM-Rank approach can be attributed to its use of a local instead of holistic objective function and to the fact that the method is not incorporating the initial weight-vector nor random restarts. To reiterate, the factors that led to improvements are a good balance between exploration and exploitation, the use of initial starting points and random restarts and the use of the holistic BLEU function instead of an approximation.

In short, we claim the following contributions to the field of Statistical Machine Translation and tuning the parameters of the log-linear model within Phrase-Based SMT in particular. We made significant improvements, with our $\text{LUS}_{\alpha=1/100}$ approach, over the state-of-the-art tuning method MERT, within a realistic setting with 14 features. Moreover, this thesis provides an elaborate overview and comparison of existing work in the field by recasting three existing and two new methods in the same terminology and wording.

8.2 Open Issues

Some open issues remain. First of all, the experiments could be extended into several directions. To name a few: *a*) SVM-Rank might perform relatively good on out-of-domain data, we still expect our algorithm to generalise relatively well towards data very different from our development set. This should be tested. *b*) We performed some investigation into the sample size most suited for LUS. We do not claim our results to be conclusive and think further investigation is justified and promising. *c*) As we mentioned in our conclusions, methods like MIRA and SVM-Rank might benefit strongly from adding nominal or binary features. Extending the dimensionality far beyond our maximum of 14 might give them the opportunity to show their strength. At the same time we would be interested to see the behaviour of LUS in such circumstances.

Regarding our new approaches, some possible improvements can be made. As mentioned in section 7.2.2 (p. 60) more work on defining a convergence criteria and a learning rate for LUS will be worth the effort. For SVM-Rank it might be beneficial to find a way to incorporate initial weight-vector. Also, taking the first 30 sentences according to BLEU from the n-best list might not be the ideal way. Selecting these as in MIRA or using a sampling approach might be beneficial.

Bibliography

Arun, A., Haddow, B., and Koehn, P. (2010). A unified approach to minimum risk training and decoding. In *Proceedings of the Joint 5th Workshop on Statistical Machine Translation and MetricsMATR*, pages 365–374. (cited on p. 7, 33, and 49)

Banerjee, S. and Lavie, A. (2005). METEOR: an automatic metric for MT evaluation with improved correlation with human judgments. In *Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, page 65. (cited on p. 18)

Bellman, R. (1957). *Dynamic Programming.* Princeton University Press. (cited on p. 39)

Brown, P. F., Cocke, J., Pietra, S. A. D., Pietra, V. J. D., Jelinek, F., Lafferty, J. D., Mercer, R. L., and Roossin, P. S. (1990). A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85. (cited on p. 3)

Brown, P. F., Cocke, J., Pietra, S. A. D., Pietra, V. J. D., Jelinek, F., Mercer, R. L., and Roossin, P. S. (1988). A statistical approach to language translation. In *Proceedings of the 12th conference on Computational linguistics - Volume 1*, pages 71–76. (cited on p. 3)

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311. (cited on p. 3)

Cer, D., Jurafsky, D., and Manning, C. D. (2008). Regularization and search for minimum error rate training. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 26–34. (cited on p. 8, 10, 20, 22, 34, and 66)

Cer, D., Manning, C. D., and Jurafsky, D. (2010). The best lexical metric for Phrase-Based statistical MT system optimization. In *Proceedings of Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 555–563. (cited on p. 15, 18, and 36)

Chiang, D., Knight, K., and Wang, W. (2009). 11,001 new features for statistical machine translation. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 218–226. (cited on p. 23, 26, 35, 37, 38, 58, and 66)

Chiang, D., Marton, Y., and Resnik, P. (2008). Online large-margin training of syntactic and structural translation features. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 224–233. (cited on p. 23, 25, 26, 35, 36, and 37)

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297. (cited on p. 44)

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585. (cited on p. 23)

Crammer, K. and Singer, Y. (2003). Ultraconservative online algorithms for multiclass problems. *The Journal of Machine Learning Research*, 3:951–991. (cited on p. 4, 23, and 26)

Doddington, G. (2002). Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*, pages 138–145, San Diego, California. Morgan Kaufmann Publishers Inc. (cited on p. 18)

Duh, K. (2008). Ranking vs. regression in machine translation evaluation. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 191–194, Columbus, Ohio. Association for Computational Linguistics. (cited on p. 9)

Duh, K. and Kirchhoff, K. (2008). Beyond log-linear models: boosted minimum error rate training for n-best re-ranking. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, pages 37–40. (cited on p. 8 and 34)

Foster, G. and Kuhn, R. (2009). Stabilizing minimum error rate training. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 242–249, Athens, Greece. Association for Computational Linguistics. (cited on p. 8 and 33)

Fox, G. C., Williams, R. D., and Messina, P. C. (1994). *Parallel computing works!* Morgan Kaufmann Pub. (cited on p. 36)

Gerrand, P. (2007). Estimating linguistic diversity on the internet: A taxonomy to avoid pitfalls and paradoxes. *Journal of Computer-Mediated Communication*, 12(4):1298–1321. (cited on p. 1)

Graff, D., Kong, J., Chen, K., and Maeda, K. (2007). English gigaword third edition. *Linguistic Data Consortium.* (cited on p. 47)

Hasan, S. and Ney, H. (2009). Comparison of extended lexicon models in search and rescoring for SMT. 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 17–20, Boulder, Colorado. Association for Computational Linguistics. (cited on p. 8 and 20)

Hawkins, D. M. (1973). On the investigation of alternative regressions by principal component analysis. *Applied Statistics*, 22(3):275–286. (cited on p. 43)

Hayashi, K., Watanabe, T., Tsukada, H., and Isozaki, H. (2009). Structural support vector machines for Log-Linear approach in statistical machine translation. In *Proceedings of International Workshop on Spoken Language Translation*, pages 144–151. (cited on p. 7)

Hoffgen, K. U., Simon, H. U., and Vanhorn, K. S. (1995). Robust trainability of single neurons. *Journal of Computer and System Sciences*, 50(1):114–125. (cited on p. 44)

Hutchins, W. J. (2003). Machine translation. In *Machine translation*, pages 1059–1066. (cited on p. 1)

Joachims, T. (1999). Making large-scale support vector machine learning practical. In Schölkopf, B., Burges, C., and Smola, A., editors, *Advances in kernel methods: support vector learning*, pages 169–184. MIT Press. (cited on p. 43 and 44)

Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. (cited on p. 43)

Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, volume 1, page 181–184. (cited on p. 47)

Koehn, P. (2004). Statistical significance tests for machine translation evaluation. In *Proceedings of Empirical Methods on Natural Language Processing*, volume 4, pages 388–395. (cited on p. 9, 50, 63, 64, and 65)

Koehn, P. (2008). Parameter tuning. In *Statistical Machine Translation*, pages 301–310. (cited on p. 9, 12, 15, 20, 21, 27, and 36)

Koehn, P., Axelrod, A., Mayne, A. B., Callison-Burch, C., Osborne, M., and Talbot, D. (2005). Edinburgh system description for the 2005 IWSLT speech translation evaluation. In *Proceedings of the International Workshop on Spoken Language Translation*. (cited on p. 48)

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. (2007). Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180. (cited on p. 4 and 46)

Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase-based translation. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, volume 1, pages 48–54. (cited on p. 3)

Kumar, S. and Byrne, W. (2005). Local phrase reordering models for statistical machine translation. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 161–168, Vancouver, British Columbia, Canada. Association for Computational Linguistics. (cited on p. 48)

Liang, P., Bouchard-Cote, A., Klein, D., and Taskar, B. (2006). An end-to-end discriminative approach to machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, volume 44, pages 761–768. (cited on p. 36)

Lin, C. and Och, F. J. (2004). Orange: a method for evaluating automatic evaluation metrics for machine translation. In *Proceedings of The 20th International Conference on Computational Linguistics*, pages 501–507. (cited on p. 36)

Liu, T. (2009). *Learning to rank for information retrieval.* Now Pub. (cited on p. 43)

Lopez, A. (2007). A survey of statistical machine translation. Technical report. (cited on p. 9 and 21)

Macherey, W., Och, F. J., Thayer, I., and Uszkoreit, J. (2008). Lattice-based minimum error rate training for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 725–734, Honolulu, Hawaii. Association for Computational Linguistics. (cited on p. 8 and 20)

McDonald, R., Crammer, K., and Pereira, F. (2005). Online large-margin training of dependency parsers. *Ann Arbor*, 100. (cited on p. 37)

Moore, R. C. and Quirk, C. (2008). Random restarts in minimum error rate training for statistical machine translation. In *Proceedings of the 22nd International Conference on Computational Linguistics - Volume 1*, pages 585–592, Manchester, United Kingdom. Association for Computational Linguistics. (cited on p. 8 and 35)

Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The computer journal*, 7(4):308–313. (cited on p. 4 and 27)

Niessen, S., Och, F. J., Leusch, G., and Ney, H. (2000). An evaluation tool for machine translation: Fast evaluation for MT research. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation*, pages 39–45. (cited on p. 18)

Och, F. J. (2002). *Statistical machine translation: From single-word models to alignment templates.* PhD thesis, RWTH Aachen. (cited on p. 18)

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. (cited on p. 4, 20, and 37)

Och, F. J. and Ney, H. (2000). Giza++: Training of statistical translation models. (cited on p. 48)

Och, F. J. and Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51. (cited on p. 3)

Papineni, K., Roukos, S., Ward, T., and Zhu, W. (2001). BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 311–318. (cited on p. 11, 15, 17, 18, and 48)

Pedersen, M. E. H. (2010). *Tuning & Simplifying Heuristical Optimization*. PhD thesis, University of Southampton. (cited on p. 12, 39, 40, and 41)

Platt, J. C. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines. (cited on p. 26)

Powell, M. J. D. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162. (cited on p. 21)

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). Downhill simplex method in multidimensions. In *Numerical Recipes*. (cited on p. 27)

Shannon, C. E. (1948). A mathematical theory of communication. *Reprinted for the Bell System Technical Journal with corrections*, 5(1):3–55. (cited on p. 3)

Shen, L., Sarkar, A., and Och, F. J. (2004). Discriminative reranking for machine translation. In *Proceedings of the Human Language Technologies and North American Association for Computational Linguistics annual meeting*, pages 177–184. (cited on p. 7)

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Weischedel, R. (2006). A study of translation error rate with targeted human annotation. In *In proceedings of the Association for Machine Translation in the Americas*. (cited on p. 18)

Stolcke, A. (2002). SRILM-an extensible language modeling toolkit. In *Proceedings of the Seventh International Conference on Spoken Language Processing*, volume 3, pages 901–904. (cited on p. 47)

Sutter, H. (2005). The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3):16–20. (cited on p. 38)

Tillmann, C. (2004). A unigram orientation model for statistical machine translation. In *Proceedings of North American Chapter of the Association for Computational Linguistics - Human Language Technologies*, pages 101–104, Boston, Massachusetts. Association for Computational Linguistics. (cited on p. 48)

Watanabe, T., Suzuki, J., Sudoh, K., Tsukada, H., and Isozaki, H. (2007a). Larger feature set approach for machine translation in IWSLT 2007. In *Proceedings of the International Workshop on Spoken Language Translation*. (cited on p. 23 and 26)

Watanabe, T., Suzuki, J., Tsukada, H., and Isozaki, H. (2007b). Online large-margin training for statistical machine translation. In *Proceedings of Conference on Empirical Methods in Natural Language Processing and Conference on Computational Natural Language Learning*, pages 764–773. (cited on p. 23, 26, and 36)

Weaver, W. (1949). Translation. 12:15–23. (cited on p. 3)

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82. (cited on p. 34)

Zaidan, O. F. (2009). Z-MERT: a fully configurable open source tool for minimum error rate training of machine translation systems. In *The Prague Bulletin of Mathematical Linguistics*, volume 91, pages 79–88. (cited on p. 8)

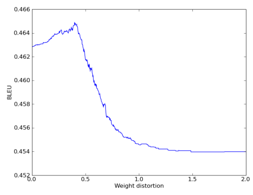
Zens, R. and Ney, H. (2003). A comparative study on reordering constraints in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 144–151. (cited on p. 4)

Zens, R., Och, F. J., and Ney, H. (2002). Phrase-based statistical machine translation. *Lecture Notes in Computer Science*, 2479/2002:35–56. (cited on p. 3)

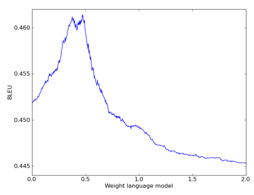
Zhao, B. and Chen, S. (2009). A simplex armijo downhill algorithm for optimizing statistical machine translation decoding parameters. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 21–24. (cited on p. 20)

Appendix A

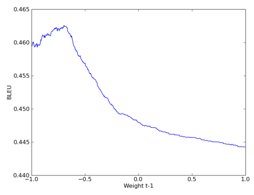
Distribution of BLEU



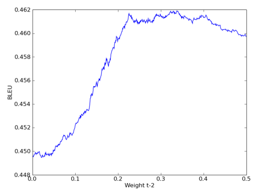
(a) $d(\mathbf{f}, \mathbf{e})$



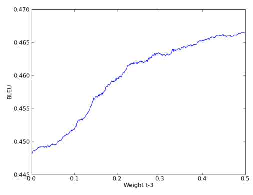
(b) $p(\mathbf{e})$



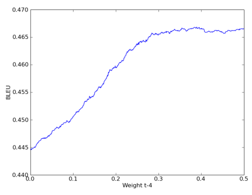
(c) $p(\mathbf{f}|\mathbf{e})$



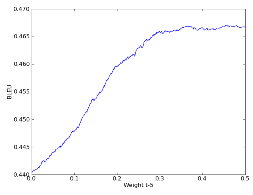
(d) $lex(\mathbf{f}|\mathbf{e})$



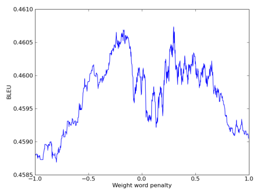
(e) $p(\mathbf{e}|\mathbf{f})$



(f) $lex(\mathbf{e}|\mathbf{f})$



(g) e^1



(h) $w(\mathbf{e})$

Figure A.1: Distribution of BLEU score per dimension of the weight-vector. These graphs are obtained by retrieving 1000-best lists from the decoder with an initial weight-vector and then varying the weight-vector in each dimension at at a time while keeping all other dimensions constant. As far as these graphs can show it, all but the word penalty graph have a unimodal shape. See also section 7.1.1 (p. 46).