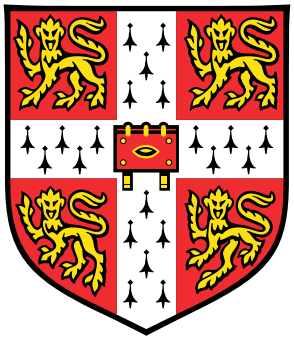
The Geometry of Statistical Machine Translation



Aurelien Waite

Department of Engineering

University of Cambridge

This dissertation is submitted for the degree of

Doctor of Philosophy

Girton College

2014

Abstract

Most modern statistical machine translation systems are based on the linear model. There are many reasons for the prevalence of the linear model: other component models can be incorporated as features, there are many methods for estimating their parameters, and the resulting model scores can be easily used in finite-state representations.

One popular method for estimating the parameters of a linear model is minimum error rate training (MERT). Galley and Quirk describe an optimal MERT algorithm that requires an exponential runtime for multiple sentences. We find that this form of MERT can be represented using convex geometry. Using this geometric representation we describe an optimal algorithm that runs in polynomial time with respect to the number of feasible solutions, and describe Projected MERT, a practical implementation of multidimensional MERT in low dimensions.

Using this geometric representation of MERT we investigate whether the optimisation of linear models is tractable in general. It has been believed that the number of feasible solutions of a linear model is exponential with respect to the number of training sentences, however we show that the exponential explosion is due to feature dimension. A result that has important ramifications because of the current trend of building statistical machine translation systems around a large number of sparse features.

We also show how these convex geometric descriptions of linear models can be neatly integrated into finite-state representations using tropical geometry. The resulting semirings provide a formulation for multidimensional MERT that can be applied to lattices and hypergraphs.

In contrast to this theoretical work, we also present practical descriptions of tools and techniques used for fast parameter estimation and filtering of hierarchical phrase-based translation models and N-gram language models. These techniques allow us to quickly build rich models over large datasets. The resulting models are used as the basis of a machine translation system that performs competitively at international evaluations.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Machine Translation (MT) is the translation of written text from a source language to a target language by purely mechanical means. Although concise, this definition fails to reflect the difficulty of the task. The underlying issue is that translation itself is a difficult concept to define. For example, the form of the translation varies depending on the context of its application. If the MT system is translating United Nations proceedings then it should be precise and accurate. If the system translates poetry or literature the output should be beautiful, pleasing, and reflect the mood of the original piece. Some argue that target languages may not even contain the words or phrases that convey concepts from the source text. Books have been written about the nebulous nature of translation [15].

To build an MT system, a model of language is needed. Early attempts were heavily influenced by the work of Chomsky [41] who popularised the context-free grammar (CFG). Famously, researchers claimed in the 1950s that using this model they would have a working system in five years. A decade later little progress had been made, a critical report was published [8], and funding for these programs was cut.

There are a variety of reasons for the failure of these early systems: for example unseen grammatical phenomena in the input sentence caused the translation to fail, and they did not capture context or semantics. Since these early efforts there have been many advances in building effective systems driven by a statistical approach to translation. Statistical machine translation (SMT) systems are robust, in that they can hypothesise about unseen or unusual grammatical structure and will always offer a translation. Statistical systems often result in surprisingly strong performance because they capture many aspects of language, such as context, which are implicit in the training data but are difficult to model in the pure rules-driven approach offered by a CFG.

Statistical approaches are possible because of the existence of parallel text. These texts

are a pairing of original source documents with translations in other languages that can be aligned with original text. Examples include Canadian Hansards [98] and European Parliamentary Proceedings [87]. The first statistical approach was based on the alignment of words between sentences [26] in parallel text. This word based method was improved by considering larger linguistic units called phrases [91]. Interestingly, further improvements have been found by incorporating the same CFGs that inspired the first wave of MT systems into a phrase-based framework [35, 36].

A consequence of using statistical based models is that systems do not produce a single translation, but a very large set of possible translations called hypotheses. These hypotheses can be compactly encoded into word lattices [146]. The lattices are then rescored with more sophisticated models [21, 93, 143]. Word lattices can be represented with Weighted Finite-State Transduces (WFST) [110]. The use of WFSTs provides a rich formalism that encompasses many disparate operations that can be applied to a lattice. An alternative approach to the word lattice encodes derivations of a CFG parse instead of strings [77].

In recent years large commercial SMT applications have been built. A very welcome consequence of these commercial developments has been the introduction of robust and efficient software to process large amounts of data. In particular the MapReduce [49] and BigTable [31] frameworks have facilitated the building of very large models, which in turn have improved the translation quality of SMT systems.

1.1 Minimum Error Rate Training

In the previous section we briefly described a statistical approach to translation. This approach provides a robust and powerful description of translation, but there is no reason to limit ourselves to this representation of translation. Instead this approach can be incorporated in another statistical model, and combined with other rich models and features.

One such additional model that can also be used as a feature is the statistical language model. This second model is a probability distribution over hypotheses based on data in a very large monolingual corpus. Both the translation and language models capture complementary properties of language, and much more monolingual data is available to train a language model than a statistical translation model. Combining both language and translation models into a single model allows us to exploit the strengths of both component models.

A commonly used method for combining models is the linear model, which provides a framework for combining many disparate features into a single model [88]. The language model can be applied alongside the translation model during translation using techniques

such as cube pruning [35] or WFST operations [81]. The next step after language model application is to augment the model with other features [114] to refine the scores of the individual hypotheses and to improve translation quality. Example features include a second model of the probability of the source sentence as a translation of the hypothesis, a length penalty, and lexical models.

SMT systems are assessed using automated error metrics such as the Bilingual Evaluation Understudy (BLEU) [117] and Translation Edit Rate (TER) [133]. These metrics work by comparing the output of the system with a set of references supplied by translators, which allows for consistent and fast measurement of translation quality. Considering that these metrics are used to judge the output of a system, it makes sense to use these metrics to also discriminatively train linear models. Examples of such methods include Minimum Error Rate Training (MERT) [62, 113], Margin Infused Relaxed Algorithm (MIRA) [151], and Pairwise Ranking Optimisation (PRO) [76].

Of all the methods of discriminative training, MERT is the most widely used. Among the reasons for its use is that the results are interpretable, it gives good performance, and is easy to implement. The MERT formulation is as follows: hypotheses with feature vectors that can never maximise the linear model given any parameter are discarded, the error for the remaining hypotheses is computed, and the parameters associated with the optimal hypothesis are selected. Interestingly, this discarding mechanism has been widely studied in the field of convex geometry [162], which is the formal study of convex polytopes. One of the contributions of this work is to describe MERT in terms of convex geometry.

1.2 Original Contributions

Our initial goal when investigating the field of convex geometry was to improve the MERT algorithm, and during this investigation we did discover useful algorithms and techniques. We provide a description of these algorithms, many of which have strong runtime and memory consumption guarantees.

The main outcome of the investigation was to discover that a linear model is severely limited when modelling high feature dimension. In this respect, our work is a negative result because we believe the performance of linear models degrades as the number of features increase. We do discuss speculative methods for transforming models with a high feature dimension to models of lower dimensions that may help with these limitations.

A summary of the contributions is as follows:

1. We build upon the work of Galley and Quirk [62] to give a description of single sen-

tence MERT using a convex polytope. We describe how parameter space is arranged, and a projection operation that allows us to perform MERT for any number of dimensions.

2. For MERT with many sentences we describe how MERT takes the form of a Minkowski sum of polytopes. We use convex geometry to describe a polynomial time algorithm for computing the Minkowski sum [59], and an upper bound on the number of feature vectors that cannot be discarded [72].

3. We give a formalism of MERT over WFSTs using a tropical polynomial semiring, which we call TGMERT. We also discuss how this semiring can be extended to any number of features using the previous two results.

4. We give a collection of software tricks for building a MapReduce pipeline to extract rules. Using these tricks gives an order of magnitude improvement in terms of both memory consumption and runtime.

1.3 Publications

The research described in this thesis has led to the following publications. We describe the original contributions made in these publications with respect to the items of the list in the previous section.

1. A. Waite, G. Blackwood, W. Byrne. Lattice-based Minimum Error Rate Training Using Weighted Finite-state Transducers with Tropical Polynomial Weights. In Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing. July 2012.

   This publication describes the formulation of the tropical polynomial semiring, corresponding to Item 3 in the previous section.

2. J. Pino, A. Waite, and W. Byrne. Simple and Efficient Model Filtering in Statistical Machine Translation. The Prague Bulletin of Mathematical Linguistics 98 (2012): 5-24.

   The original contribution in this publication is the description in Section 5 of fast N-gram filtering. This description is repeated in Section 4.3.1.

3. J. Pino, A. Waite, T. Xiao, A. de Gispert, F. Flego, and W. Byrne. The University of Cambridge Russian-English System at WMT13. In Proceedings of the Eighth Workshop on Statistical Machine Translation. August 2013.

   This publication is a system description of a Russian-to-English system. During the development of this system the MapReduce based rule extractor was rewritten to be faster and consume less memory. This work corresponds to Item 4 in the list of original contributions.

4. A.Waite and W. Byrne. The Geometry of Statistical Machine Translation. In Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.

   This work is a summary of items 1 and 2 from the list of original contributions.

1.4 Organisation of Thesis

The organisation of the thesis is as follows. Chapter 2 describes the approaches we use to build translation and language models, how to build an SMT decoder, and what is needed to build an effective SMT system with the exception of the training of parameters for a linear model. In Chapter 3 we discuss how to set the parameters of a linear model. We review MERT and all its variants, as well as other relevant methods for parameter tuning.

Chapter 4 describes a MapReduce pipeline for rule extraction. We then describe how to improve the MapReduce pipeline using implementation tricks and report on runtimes and memory usage.

Chapter 5 is the description of MERT for a single input sentence using convex geometry including a discussion of available algorithms. In this chapter we introduce projected MERT, a method of computing MERT for any number of features. We then move onto MERT for many input sentences in Chapter 6. The main insight is that MERT for a training set can be modelled as a Minkowski sum of single sentence polytopes. We discuss algorithms for computing this polytope and a discussion on the tractability of MERT for systems with many features.

In Chapter 7 we discuss TGMERT and the tropical polynomial semiring. In effect this semiring defines an algebra over convex polytopes. We describe how this semiring can be used to extend lattice-based MERT to a multidimensional search.

Chapter 2

Statistical Machine Translation Decoding

In this Chapter we describe how, given a source sentence $\mathbf{f}$, to generate a set of hypotheses $\{\mathbf{e}\}$ in the target language, and how these hypotheses are used to generate a set of feature vectors $\{\mathbf{h}(\mathbf{e},\mathbf{f})\}$. The main focus of our work is the linear model, and the description of machine translation in the context of this model.

The linear model is sometimes also referred to as the log-linear model [114]. The additional log qualifier is due to the model taking the form of a probability distribution. For D features the probability $p(\mathbf{e}|\mathbf{f})$ is the log-linear sum of a feature function $\mathbf{h}(\mathbf{e},\mathbf{f})$ weighted by the D-dimensional parameter vector $\mathbf{w}$. Following [162] we treat $\mathbf{w}$ as a row vector and write a decision rule that selects the most probable hypotheses $\hat{\mathbf{e}}$ as:

$$\begin{aligned}\hat{\mathbf{e}} &= \operatorname*{argmax}_{\mathbf{e}} p(\mathbf{e} \mid \mathbf{f}) \\ &= \operatorname*{argmax}_{\mathbf{e}} \frac{\exp \mathbf{w}\mathbf{h}(\mathbf{e},\mathbf{f})}{\sum_{\mathbf{e}'} \exp \mathbf{w}\mathbf{h}(\mathbf{e}',\mathbf{f})}\end{aligned} \tag{2.1}$$

The task of training linear and log-linear models is to find an optimal parameter $\hat{\mathbf{w}}$. For simplicity we assume a single reference $\mathbf{r}$, a single source sentence $\mathbf{f}$, and that it is possible to compute the feature vector $\mathbf{h}(\mathbf{r},\mathbf{f})$. For a log-linear model one method for finding optimal parameters $\hat{\mathbf{w}}$ is to maximise the log-likelihood [114].

$$\hat{\mathbf{w}} = \operatorname*{argmax}_{\mathbf{w}} \left\{ \mathbf{w}\mathbf{h}(\mathbf{r},\mathbf{f}) - \log \sum_{\mathbf{e}'} \exp \mathbf{w}\mathbf{h}(\mathbf{e}',\mathbf{f}) \right\} \tag{2.2}$$

The term resulting from the normalisation operation plays an important role. It acts as a penalty term that discourages the optimiser from rewarding other possible hypotheses. The resulting distribution should have a large amount of probability mass assigned to the

reference translation. However, computing this term is often problematic because it has to be computed over all possible hypotheses.

Now under the log probability, the decision rule in Eqn. (2.1) takes the following linear form:

$$\hat{\mathbf{e}} = \underset{\mathbf{e}}{\operatorname{argmax}}\, \mathbf{wh}(\mathbf{e}, \mathbf{f}) \tag{2.3}$$

Because the exponentiation and normalisation operations do not change the result of the argmax operation we can avoid computing the normalisation term when evaluating Eqn. (2.3).

If the normalisation term can be ignored when evaluating the decision rule, then it suggests the normalisation term may not be necessary for training. Let us consider the inner product $\mathbf{wh}(\mathbf{e}, \mathbf{f})$ as some model score that is correlated with the quality of the translation. In this method the optimiser maximises the function with respect to some oracle hypothesis $\hat{\mathbf{e}}$

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmax}} \{\mathbf{wh}(\hat{\mathbf{e}}, \mathbf{f})\} \tag{2.4}$$

Instead of incorporating other hypotheses in the objective function using the normalisation term, the optimisation is prevented from favouring other hypotheses by a set of constraints. We describe methods for constrained optimisation of Eqn. (2.4) in Chapter 3 and the set of constraints themselves are the focus of chapters 5 and 6. We prefer to use the term linear model for these methods, to distinguish between model scores and probabilities.

The distinction between the log-linear model and linear models is somewhat artificial. We note that the parameters found for a linear model can be substituted back into the fractional term Eqn. (2.1) to give a valid probability distribution, and the log-likelihoods can be treated as model scores. Essentially the difference is due to the training regime.

Applying the argmax operation in Eqn (2.3) is a process called decoding. This name is derived from the source-channel model originally proposed by Brown et al. [26]. Decoding is not a trivial operation to the size of the hypothesis set. To give an idea of scale of the problem, word lattices for a single translation can contain in the order of 10^{80} hypotheses [143], and the development of efficient decoding procedures is a major research focus.

Therefore before we can describe training of linear models, we first need to describe a method for computing the argmax operation and the feature vector $\mathbf{h}(\mathbf{e}, \mathbf{f})$. In the first section we describe the N-gram language model, in the second we describe hierarchal phrase-based translation, in the third we describe decoding, and finally in the fourth section we describe decoding using WFSTs.

2.1 N-gram Language Models

In this section we describe a statistical model applied to natural language. The N-gram language model (as summarised in Jelinek [82], Huang et al. [78], Jurafsky and Martin [83] and Koehn [88]) is used to assign a probability to a sequence of words. N-gram language models are widely used because they are robust, they capture both semantics and syntax at the surface level, and can be easily trained on large amounts of data. The log-probability of a hypothesis under a language model is one of the key features in the feature vector $\mathbf{h}(\mathbf{e},\mathbf{f})$.

The language model assigns a probability to a word sequence W_1^I. Long word sequences do not usually appear verbatim in training data so we apply the N-gram conditional independence assumption:

$$P(W_1^I) \approx \prod_{i=1}^{I} P(W_i \mid W_{i-n+1}^{i-1}) \tag{2.5}$$

The N-gram assumption is a type of Markov independence assumption where the probability of a word is conditioned only on the previous $n-1$ words. The value of n is referred to as the N-gram order. The distribution $P(W_i \mid W_{i-n+1}^{i-1})$ can be computed using maximum likelihood estimation where $f(W_{i-n+1}^{i})$ is a function that yields a count from a monolingual corpus:

$$P(W_i \mid W_{i-n+1}^{i-1}) = \frac{f(W_{i-n+1}^{i})}{f(W_{i-n+1}^{i-1})} \tag{2.6}$$

A problem with the N-gram model is that events may not appear in the monolingual corpus. An unseen N-gram results in $P(W_1^I) = 0$. This sparseness problem becomes more pronounced as high order N-grams are used.

2.1.1 Solutions for N-gram Sparseness

In the previous section we described the basic N-gram model and the sparseness problem. In this section we describe two approaches to solving data sparseness: discounting, and the interpolation or back-off to lower order N-grams. These approaches can be combined in language model smoothing.

Discounting

To create a valid probability distribution over N-grams we have to enforce the sum to 1 constraint for the number of the unique words in the vocabulary V:

$$\sum_{i=1}^{V} P(W_i \mid W_{i-n+1}^{i-1}) = 1 \tag{2.7}$$

If we have an unseen N-gram, this constraint does not allow any probability mass to be assigned to it if probabilities are calculated under Eqn. (2.6). We therefore have to discount the N-gram probability $P(W_i \mid W_{i-n+1}^{i-1})$ using a discount coefficient $d(\cdot)$. Eqn. (2.6) is rewritten to allow for the redistribution of probability mass:

$$P(W_i \mid W_{i-n+1}^{i-1}) = d(f(W_{i-n+1}^{i}))\frac{f(W_{i-n+1}^{i})}{f(W_{i-n+1}^{i-1})} \tag{2.8}$$

One form of discounting is Good-Turing discounting [68]. Defining n_r as the number of N-grams occurring r times, the discount coefficient is written as:

$$d(r) = \frac{(r+1)n_r + 1}{rn_r} \tag{2.9}$$

Note that $r > 0$ because the discounting is applied to seen N-grams only. The reserved probability mass is then distributed uniformly among the unseen N-grams in the test set.

Interpolation and Back-off

In the previous section we described discounting, which reserves probability mass in the N-gram probability distribution $P(W_i \mid W_{i-n+1}^{i-1})$ for unseen events. Instead of relying on a discount factor, both interpolation and back-off strategies use lower order N-gram models to guide the probability assigned to an unseen N-gram.

In an interpolated model the N-gram probabilities $P(W_i \mid W_{i-n+1}^{i-1})$ are computed as a linear interpolation of high and low order models. The interpolation makes the N-gram model more robust as there are far fewer unseen events associated with lower order N-gram models. The interpolated probability can be computed recursively as follows:

$$P_{\text{INTERP}}(W_i \mid W_{i-n+1}^{i-1}) = \lambda P(W_i \mid W_{i-n+1}^{i-1}) + (1-\lambda)P_{\text{INTERP}}(W_i \mid W_{i-n+2}^{i-1}) \tag{2.10}$$

The weights of the interpolated model can be optimised on a corpus of representative

held-out data using deleted interpolation [10]. The training data is split into blocks, with one block held-out (the deleted block). The values of λ are chosen to maximise the likelihood of the deleted block. The deleted block is then rotated and the process repeated.

The back-off model [85] differs in that a lower order N-gram model is only used if a higher order N-gram is unavailable. The recursive form of the equation that describes an N-gram probability is:

$$P(W_i \mid W_{i-n+1}^{i-1}) = \begin{cases} \rho(W_i \mid W_{i-n+1}^{i-1}) & \text{if } f(W_{i-n+1}^{i}) > 0 \\ \lambda_{W_{i-n+2}^{i}} P(W_i \mid W_{i-n+2}^{i}) & \text{otherwise} \end{cases} \tag{2.11}$$

where the count function $f(W_{i-n+1}^{i})$ is the number of times the N-gram was found in the training data and $\rho(W_i \mid W_{i-n+1}^{i-1})$ is an estimate of $P(W_i \mid W_{i-n+1}^{i-1})$. Usually $\rho(W_i \mid W_{i-n+1}^{i-1})$ is computed as a discount of the relative frequency estimate:

$$\rho(W_i \mid W_{i-n+1}^{i-1}) = d(f(W_{i-n+1}^{i})) \frac{f(W_i \mid W_{i-n+1}^{i-1})}{f(W_i \mid W_{i-n+2}^{i-1})} \tag{2.12}$$

Language Model Smoothing

We can combine discounting, interpolation and back-off to perform language model smoothing. A simple example is additive language model smoothing [61]. This method of smoothing adds a constant to each N-gram count and estimates probabilities using the relative frequency of the modified counts.

Another example is Kneser-Ney smoothing where the smoothed probability is defined as:

$$P_{KN}(W_i \mid W_{i-n+1}^{i-1}) = \begin{cases} \frac{\max\{f(W_{i-n+1}^{i}) - D, 0\}}{f(W_{i-n+1}^{i-1})} & \text{if } f(W_{i-n+1}^{i}) > 0 \\ \gamma(W_{i-n+1}^{i-1}) P_{KN}(W_i \mid W_{i-n+2}^{i-1}) & \text{otherwise} \end{cases} \tag{2.13}$$

where

$$P_{KN}(W_i \mid W_{i-n+2}^{i}) = \frac{\mathbb{C}(\bullet W_{i-n+2}^{i})}{\sum_{W_i} \mathbb{C}(\bullet W_{i-n+2}^{i})} \tag{2.14}$$

where $\mathbb{C}(\bullet W_{i-n+2}^{i}) = |\{W_{i-n+1} : f(W_{i-n+1}^{i}) > 0\}|$ is the number of unique words that precede the backed off N-gram W_{i-n+2}^{i}. The motivation behind this smoothing method is that some words naturally co-occur, such as *San Francisco* [33]. Therefore for any N-gram starting with *Francisco* the value of $\mathbb{C}(\bullet W_{i-n+2}^{i})$ would be low because the preceding word is almost always *San*.

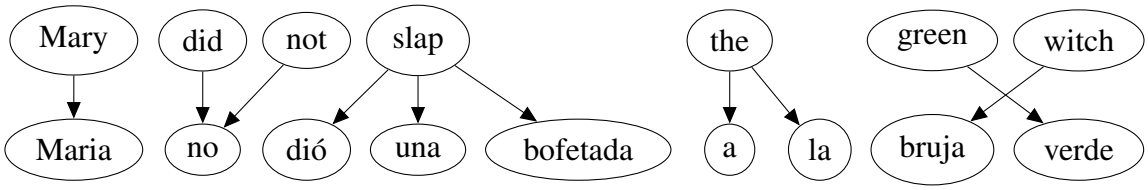


Fig. 2.1 Word alignment example showing the one-to-many links between source and target words for a Spanish → English sentence pair [83]

2.2 Hierarchical Phrase-Based Translation

In this section we describe an approach to statistical machine translation, called hierarchical phrase-based translation [35, 36]. This approach builds upon previous work such as word [26] and phrase translaton [91] models.

These approaches all rely on a collection of sentence pairs, called the parallel text, to build a model based on some latent variable. For example, in the word alignment approach it is assumed there is some hidden alignment link between words in the source and target sentences in a sentence pair, as in Figure 2.1. By analysing parallel sentences in the parallel text we can find occurrences of source and target words that consistently appear together across many sentences. Phrase-based and hierarchical phrase-based approaches build on this approach by recognising the appearance of more complex linguistic phenomena in parallel data.

We first describe word alignment approaches, followed by phrase-based approaches. We then describe how the rules of a synchronous context-free grammar can be extracted using phrase pairs.

2.2.1 Word Alignment

The foundation of rule extraction is the word alignment model of Brown et al. [26]. A word alignment is a latent linking between words of the source sentence and its translation. Links between words correspond to syntactic functions or semantic relationships shared by the words of a sentence and its translation. We provide an example of a parallel sentence with a set of word alignments in Figure 2.1.

We now briefly describe word alignment of parallel text. Brown et al. [26] introduce a series of five translation models of increasing sophistication known as IBM Model 1 to IBM Model 5. Lower numbered models are computationally more tractable, and higher

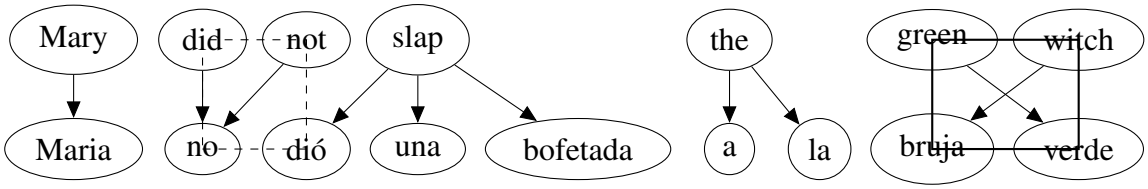


Fig. 2.2 A correctly and incorrectly aligned phrase for a Spanish → English sentence pair [83]. The incorrect alignment has a dashed border

numbered models attempt to capture more linguistic phenomena. In addition to the IBM models word, alignment models have also been proposed based on a hidden Markov model [51, 148].

Brown et al. [26] also provide algorithms for the unsupervised estimation of model parameters from the set of parallel sentences. In the models, word alignments are treated as hidden variables specifying the alignment from a word in the source sentence to a word in the target sentence. Expectation Maximisation (EM) [50] is used as the basis of an unsupervised algorithm for learning these alignments. In more sophisticated models it is possible for EM to find a poor local minimum, which can be prevented by initialising these models from simpler models in the series. For example, Model 2 can be initialised from a trained Model 1.

2.2.2 Phrase-Based Translation

Given a set word alignments, we use them to extract phrases [91]. A phrase is defined as any sequence of words. A phrase has no syntactic or semantic significance on its own, except that it is translatable. A heuristic is used to extract phrases from a parallel sentence $(\mathbf{f}, \mathbf{e})$ given a set of word alignments a: for example a phrase pair $(f_{j_1}^{j_2}, e_{i_1}^{i_2})$ is subject to the constraint that $\forall(j,i) \in a(j \in [j_1, j_2] \Leftrightarrow i \in [i_1, i_2])$ where $\Leftrightarrow$ denotes a word alignment. In figure 2.2 we show an example with two phrases. The phrase pair ("did not", "no dió") fails to satisfy the constraint as "dió" is aligned with "slap" which is outside the phrase pair. The phrase pair ("green witch", "bruja verde") has a crossing alignment but satisfies the constraint as it appears within the phrase pair.

For a given source sentence **f** and target sentence **e** from a parallel text let us assume that I phrase pairs have been found. Let us denote the probability of an ith target phrase $\bar{e}_i$ given a source phrase $\bar{f}_i$ as $\phi(\bar{e}_i \mid \bar{f}_i)$. Using an independence assumption between phrase pairs the

probability of the sentence can be written approximately as

$$\phi(\mathbf{e} \mid \mathbf{f}) = \prod_{i=1}^{I} \phi(\bar{e}_i \mid \bar{f}_i) d(\text{start}_i - \text{end}_{i-1} - 1) \tag{2.15}$$

where $d(\cdot)$ is a distance-based distortion function designed to penalise reordering, with start_i as the start position of the target phrase translated from the ith source phrase and end_{i-1} as the end position of target phrase translated from the preceding source phrase. This distance function can be estimated from parallel data or modelled as an exponential decay [91]. More complex phrase-based translation models account for repetition, deletion, permutation, insertion, and alternative segmentations.

The value of $\phi(\cdot)$ is computed using the relative frequency counts from the parallel data. The word alignments used to extract $\phi(\cdot)$ are not considered at this stage. The maximum likelihood estimate of the probability of translating phrase $\bar{e}_i$ given $\bar{f}_i$ is

$$\phi(\bar{e}_i \mid \bar{f}_i) = \frac{f(\bar{e}_i, \bar{f}_i)}{\sum_{\bar{e}'} f(\bar{e}', \bar{f}_j)} \tag{2.16}$$

2.2.3 Hierarchical Phrase-Based Translation

We have described a phrase-based approach to translation. This approach is successful at capturing local reordering within phrase pairs. One disadvantage of the approach was of the function $d(\cdot)$ to penalise reordering of phrases. This encourages phrase pairs to be appear in the same order in a hypothesis as in the source sentence. For similar languages, such as many of the European languages, this restriction is probably acceptable. For more diverse languages, such as English, Arabic, and Chinese, we may need to capture more complex structure between the languages.

Hierarchical Phrase Based Translation [35, 36] is based on the application of a Synchronous Context-Free Grammar (SCFG) [96]. The rewrite rules of an SCFG are defined with respect to two sets of symbols: A set of terminals T, which corresponds to the vocabularies of the source and target languages, and a set of non-terminals that allow for the hierarchical structure of the grammar to be formulated. Given a set of terminals the rules of the grammar are:

$$X \rightarrow < \gamma, \alpha, \sim >$$

where X is a non-terminal and, $\gamma, \alpha \in (\{X\} \cup T)^+$ are sequences of terminals in T and non-terminals in the source or target language. The relation $\sim$ defines a bijective mapping of non-terminals in the source and target sides of the rule.

Given an input sentence **f** and translation **e** there is a set of rules that can be applied in such way to produce a **f** and **e**. This set of rules, and the order they are applied, is called a derivation D. Two additional rules are specified:

$$S \rightarrow < X, X >$$
$$S \rightarrow < SX, SX >$$

these are called the "glue rules" as they complete the derivation of a sentence and allow monotonic concatenation of terminals or rules. In addition to these rules, a set of rules consisting of only of terminals are added.

Due to the context-free nature of the grammar it is necessary to make the assumption that the application of a rule is independent of the previous rules. The probability of a derivation can be written as

$$P(D) = \prod_{(X \rightarrow < \gamma, \alpha, \sim >) \in D} P(X \rightarrow < \gamma, \alpha, \sim >) \tag{2.17}$$

Compare this value to the probability of translation given phrases Eqn. (2.16), and note the deletion of the distance function. The ordering of target phrase relative to the source phrases can be modelled in the grammar directly.

Hierarchical phrase-based translation builds upon a phrase-based translation by extracting a synchronous context-free grammar from a set of phrase pairs. Many phrase pairs in this set contain smaller phrase pairs as constituents. By taking the larger phrase pairs and substituting the constituent phrase pairs with non-terminals a rule set is generated. This can lead to an explosion of rules so the rule set is filtered to discard infrequent and hopefully noisy rules [35, 80]. Once a set of hierarchical rules has been extracted the weights are estimated from a parallel corpus. This can be done by relative frequency [35, 36] or computed from the word alignment models [48].

Ultimately, the derivation is a latent variable that is not of much interest to us. Our goal is to describe a probability distribution over hypotheses, and not derivations. There is a complication in that many derivations, denoted as the set $\mathscr{D}_f$, can produce the same strings. The derivation should be marginalised out of the probability, but this can be difficult due to the large number of hypotheses that would require this operation. Therefore the optimal hypothesis $\hat{\mathbf{e}}$ is found using the max derivation approximation [23, 35, 36]

$$\hat{\mathbf{e}} \approx \mathbf{e}\left(\underset{D \in \mathscr{D}_f}{\operatorname{argmax}} P(D)\right) \tag{2.18}$$

where the notation $\mathbf{e}(D)$ refers to the hypothesis yielded by the derivation D. We stated in the introduction, that we are interested in model scores as opposed to probabilities. We can convert the probability to a score by taking the log of the derivation probability. Using the function $S(D)$ to denote the score of a derivation, the following can be written

$$S(D) = \mathbf{w} \left\{ \sum_{(X \rightarrow <\gamma,\alpha,\sim>) \in D} \mathbf{h}(X \rightarrow <\gamma,\alpha,\sim>) \right\} \tag{2.19}$$

We have altered the definition $\mathbf{h}$ so that it is evaluated with a rule as input. Most of the features used in SMT can be extracted from the rules directly. Examples include rule log-probabilities of both the source given the target and the target given the source, the length of the hypotheses, the lexical log-probabilities of a rule based on a IBM Model 1 word-based alignment model [26], the number of applications of the glue rule, and the total number of rules in the derivation. One important exception is the language model feature, which must be computed over N-grams. These N-grams may cross rule boundaries in the derivation, breaking the context-free property. We shall see in the next section that applying the language model is a non-trivial problem.

2.3 Statistical Machine Decoding

In Section 2.1 we described a statistical language model and in Section 2.2 we described the hierarchical phrase-based model. Both approaches can be used to ascribe a probability or score to a hypothesis $\mathbf{e}$, which can be incorporated as features in a linear model. Because the set of hypotheses is so large, it is intractable to enumerate through all possible hypotheses for scoring.

The solution to this problem is to build the hypotheses incrementally. A partial hypothesis is discarded if it has a very low score compared to the partial hypothesis with the best score. This process of building and discarding hypotheses is called decoding. We describe the decoding process for a hierarchical phrase-based translation [36] and then describe a hierarchical phrase-based formulation using WFSTs.

The hierarchical phrase-based decoder uses the CYK parsing algorithm [32] to parse the input. This algorithm is designed for parsing sentences using grammars in Chomsky Normal Form (CNF), which constrains the right hand side of a production rule to a maximum of two symbols. A derivation for a grammar is therefore binary branching. This binary branching structure allows the derivation to be represented as a two dimensional matrix called a chart.

The number of rows and columns in the chart equal the length of the input sentence, and each element correspond to a sub-span of the sentence. E.g. the element $[2,5]$ corresponds to a span from the 2nd to the 5th word.

Each chart cell contains the partial derivation which span the positions $[x,y]$. The chart is filled left to right and bottom to top. Each non-terminal maintains back-pointers to chart cells from which it was derived. Once the algorithm is complete the parse tree, which includes all possible derivations, can be generated by following the back-pointers from the start cell.

In Section 2.2.3 we stated that the language model cannot easily be computed in a rule-based feature function. The language model must be applied to a sequence of terminals, if there are any non-terminals in the partial derivation then the full language model cannot be applied. We therefore need to generate the full hypotheses associated with the derivations before language model application. This causes a problem as there are a vast number of derivations generated by the CYK algorithm. It is possible to prune the derivations, but the approach should not remove hypotheses with potentially high language model scores. Cube pruning [36] does this by considering the K derivations at each cell with the highest score whilst following the back-pointers. At each cell the derivation consists of a sequence of terminals and non-terminals. An approximation of the language model is applied that only considers the N-grams in the terminals of the derivation. The approximated language model scores are then used to rank the K-best derivations.

2.3.1 HiFST

We have described decoding for a hierarchical phrase-based model. This decoder is sufficient for producing the set of K feature vectors needed for Chapters 5 and 6.

One improvement would be to generate many more derivations during decoding. Because of the very large number of possible derivations, it is not possible to store each complete derivation. However, as a consequence of the CYK algorithm many of these derivations share sub-derivations. By exploiting this shared structure many more derivations, and therefore hypotheses, can be considered. One data structure that exploits these shared derivations to compactly encode a large number of derivations is the hypergraph [77].

There exists a data structure similar to the hypergraph for strings that is called a word lattice [92, 146]. This string-based data structure can be represented as a Weighted Finite-State Transducer (WFST)[110] giving us access to a rich formalism and software toolkits for processing the word lattice. To use the WFST representation we are forced to convert

the set of derivations to strings, and this allows an easier application of the string-based language model to the word lattice.

In this section we describe HiFST [47, 81], a hierarchical phrase-based decoder that uses a WFST. The HiFST decoder produces word lattices and applies the full language model using WFST operations. The WFST is also needed for the tropical polynomial semiring introduced in Chapter 7.

2.3.2 Semirings

WFST operations are defined with respect to an algebraic structure called a semiring. The semiring is a concept taken from modern algebra, and we now provide a brief summary of modern algebra based on Chapter 1 and Appendix A of Cox et al. [44].

An algebra is a set of values and corresponding operations. Examples of value sets include the set of real numbers $\mathbb{R}$, the set of complex numbers $\mathbb{C}$, and the set of integers $\mathbb{Z}$. Operations include addition, subtraction, multiplication, and division. A requirement of an algebra is that the application of operation must result in a value that is contained in the value set. This requirement precludes the division operator with the use of set of integers $\mathbb{Z}$, because the result may be a rational number.

Algebras are classed by the set of operations and their properties. The algebra that we are most familiar with is the *field*, which contains addition, subtraction, multiplication, and division. The definition of a field also requires that the multiplication and addition operators have commutative, associative, and distributive properties. For the definition of a WFST and the tropical algebra explored in Chapter 7 we need an algebra with a looser definition. This algebra is called a *ring*

Definition 2.1. *A commutative ring consists of a set $\mathbb{K}$ and two binary operations $\otimes$ and $\oplus$ defined on $\mathbb{K}$ for which the following conditions are satisfied:*

i $(a \oplus b) \oplus c = a \oplus (b \oplus c)$ *and* $(a \otimes b) \otimes c = a \otimes (b \otimes c)$ *for all* $a, b, c \in \mathbb{K}$ *(associative).*

ii $a \oplus b = b \oplus a$ *and* $a \otimes b = b \otimes a$ *for all* $a, b \in \mathbb{K}$ *(commutative).*

iii $a \otimes (b \oplus c) = a \otimes b \oplus a \otimes c$ *for all* $a, b, c \in \mathbb{K}$ *(distributive).*

iv *There are* $\bar{0}, \bar{1} \in \mathbb{K}$ *such that* $a \oplus \bar{0} = a \otimes \bar{1} = a$ *for all* $a \in \mathbb{K}$ *(identities).*

v *Given* $a \in \mathbb{K}$*, there is* $b \in \mathbb{K}$ *such that* $a + b = 0$ *(additive inverses).*

The classical addition $+$ and multiplication $\times$ arithmetic operations, and the set of real numbers satisfy the conditions of a ring. We can also define more abstract rings. For example a viable set of values is the set of all polynomials, which yields the polynomial ring.

The set of operations can also be redefined to yield new algebras. The tropical algebra uses the min function and classical $+$ operator as shown in Table 2.1. However, these operators do not satisfy condition v of the ring definition. In these cases, the algebras are called *semirings*.

2.3.3 Weighted Finite-State Transducers

Essentially a WFST is a type of graph. By convention vertices of the graph are called *states*, and edges are called *arcs*. More formally, a WFST [110] $T = (\mathscr{A}, \mathscr{B}, Q, I, F, E, \lambda, \rho)$ over weight set $\mathbb{K}$ is defined by an input alphabet $\mathscr{A}$, an output alphabet $\mathscr{B}$, a set of states Q, a set of initial states $I \subseteq Q$, a set of final states $F \subseteq Q$, a set of weighted arcs E, an initial state weight assignment $\lambda : F \to \mathbb{K}$ and a final weight state assignment $\rho : F \to \mathbb{K}$. The sets $\mathscr{A}, \mathscr{B}, Q, I, F, E$ are all of finite size. For each state $q \in Q$ let $E[q]$ denote the set of all arcs (i.e. edges) leaving state q. The weighted arcs of T form the set:

$$E \subseteq Q \times (\mathscr{A} \times \mathscr{B} \times \mathbb{K} \times Q)$$

where $\times$ is the Cartesian product of two sets.

Many algorithms and operations used with WFSTs are defined in terms of semirings [110]. A semiring $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ is an algebraic structure that is defined by a set of values $\mathbb{K}$, two binary operators $\oplus$ and $\otimes$ and designated values $\bar{0}$ and $\bar{1}$. Four commonly used semirings are shown in table 2.1.

For arc $e \in E$, let $p[e]$ denote its source state, $n[e]$ its target state, $i[e]$ its input label, $o[e]$ its output label, and $w[e]$ its weight. Let $\pi = e_1 ... e_K$ denote a complete path T from initial state $p[e_1]$ to final state $n[e_k]$, so that $n[e_{k-1}] = p[e_k]$ for $k = 2, ..., K$. The weight of the path π is the $\otimes$-product of the weights of the arcs.

In some instances we wish only to compute the weight of a string. The transduction ability of the WFST is not needed in these cases. Following a similar definition to the WFST, the weighted finite-state acceptor (WFSA) differs by using only a single alphabet $\mathscr{A}$ in its definition.

Using this formulation many operations can be described. These operations all have strong complexity guarantees for both runtime and memory usage. Examples of WFSA and WFST operations include:

Semiring	Weight Set	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$
boolean	$\{0,1\}$	$\vee$	$\wedge$	0	1
probability	$\mathbb{R}_+$	$+$	$\times$	0	1
log	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\oplus_{\log}$	$+$	$+\infty$	0
tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	min	$+$	$+\infty$	0

Table 2.1 Semirings often used for natural language processing [110]. The log semiring addition operator $\oplus$ is defined as $x \oplus_{\log} y = -\log(e^{-x} + e^{-y})$

Projection Convert a WFST to a WFSA by projecting onto either the input or output labels. This operation is denoted by $A = \Pi_1(T)$ for input projection and $A = \Pi_2(T)$ for output projection.

Shortest Path Find the path with the lowest weight in an WSFA or WFST.

Intersection Given two WFSA representations A_1 and A_2, the weight associated to the input string u by the intersection of two acceptors is $(A_1 \cap A_2)(u) = A_1(u) \otimes A_2(u)$.

Composition Let us assume we have two WSFT representations T_1 and T_2. There is a path in T_1 which maps the string u to v and another path in T_2 which maps string v to string w. The weight given to the path with the input string u and output string w in the composed WFST is defined as

$$(T_1 \circ T_2)(u,w) = \bigoplus_v \{T_1(u,v) \otimes T_2(v,w)\}$$

Determinisation In a determinised WFST each state has at most one arc with any given input. Note that although determinsiation may change the number of states and edges, the determinised WFST representation preserves the mapping between input and output strings, and their assigned weights. In a determinised WFST there exists only one path for any given input string, therefore matching the input string is faster.

Minimisation Minimisation attempts to remove states and edges from a determinised WFST. The minimisation operation also preserves the mapping between input and output strings, and their assigned weights.

2.3.4 Word Lattices

Formally, a word lattice [92, 146] is a weighted directed acyclic graph (DAG) [43]. The sequence of state arcs on each complete path from the initial state to a final state defines a translation hypothesis and its cost. The total cost of the hypothesis is obtained by aggregating the costs of the individual arcs that define the path.

A word lattice can be represented as a WFST. Using WFSTs provides a set of general purpose optimisation operations [110] to determinise and minimise the lattice for space efficiency. A WFST representation also allows manipulation of arc or path weights. Hypothesis scores can be converted to a normalised probability distribution or the weights redistributed optimally for second pass rescoring.

2.3.5 Hierarchical Phrase-Based Decoding with WFSTs

In this section we describe the HiFST decoder [47, 79, 81]. As stated in Section 2.3.1 the CYK algorithm produces a large number of derivations that share sub-derivations. By exploiting this shared structure many more derivations, and therefore hypotheses, can be considered.

One method of representing this shared structure is by using a recursive transition network (RTN) [107, 154]. An RTN is a family of many finite-state automata (FSA). Formally $R = (\mathbb{N}, \mathscr{A}, (T_v)_{v\in\mathbb{N}}, S)$ where $\mathbb{N}$ is a set of non-terminals, $(T_v)_{v\in\mathbb{N}}$ is a family of FSAs with input alphabet $\mathscr{A} \cup \mathbb{N}$, and $S \in \mathbb{N}$ is the root symbol with the FSA T_s as the root FSA.

The reason that RTNs provide a more compact representation of the derivations is because FSAs can be shared between other FSAs. If two FSAs share the same FSA only one copy of the shared FSA needs to be stored in memory. The two parent FSAs use a pointer to the single shared FSA.

Translation in HiFST is performed in two steps. The first step parses the source sentence according to a variant of the CYK algorithm [32]. Each cell in the CYK grid contains a non-terminal and a span over the foreign sentence. The span contains back pointers to other cells in the grid.

In the second step the derivations identified in the parse drives the generation of a target language word lattice containing all possible translations and derivations of the source sentence. Each span can be represented by a FSA in the RTN. HiFST starts at the top cell of the chart, which forms the root FSA T_s. The spans are enumerated and FSAs added to the RTN representation.

The RTN represents the set of derivations, which need to have the language model score

applied to them. The language model is represented as a WFSA [1]. The WFSA representing the language model is then intersected with the RTN to give hypotheses a language model score. This operation is referred to as a first pass language model. This first pass language model is chosen to balance complexity versus modelling. After intersection with the language model WFST the word lattice is pruned [110]. Pruning reduces the number of hypotheses to allow for efficient decoding.

Formally this procedure can be represented as a series of WFST operations. Given a string s which represents the source sentence, a SCFG G, and an N-gram language model M represented as an WFSA then HiFST proceeds as follows:

1. Apply the translation grammar: $\mathscr{T} = \Pi_2(s \circ G)$.

2. Apply the language model via intersection: $\mathscr{L} = \mathscr{T} \cap M$.

3. Apply the shortest path algorithm on $\mathscr{L}$ to find the highest scoring path.

In this formulation, the construction of the translation space $\mathscr{T}$ requires the use of the modified CYK algorithm that we have just described. The rest of the operations can be completed with standard WFST operations.

An RTN is not the only choice of representation for $\mathscr{T}$. For example, another alternative is a push down automata (PDA) [79]. A PDA representation can be used for a larger $\mathscr{T}$ at the cost of a smaller language model M.

2.3.6 Large Language Model Rescoring of Word Lattices

The first pass language model is an expensive operation to apply because of the number of derivations represented by the word lattice. Therefore typically a small 4-gram language model estimated over a subset of the monolingual training data is used. Once the word lattice has been generated by HiFST we apply a higher order second pass language model. This changes the rank of the hypotheses in the lattice. An example of this second pass rescoring is 5-gram language model rescoring [21].

A 5-gram language model can occupy a large amount of memory. We therefore extract all the 5-grams from the word lattice. We then build a lattice specific language model from these N-grams that is only suitable for rescoring the word lattice. A fast method for building these language models is described in Chapter 4.

2.4 Summary

Our goal in this chapter is to frame the discussion of linear models with respect to an active and interesting problem. We described many of the advances of the previous three decades that allow for large-scale and performant SMT systems. A system built around smoothed language models, hierarchical phrase-based models, and WFST based decoders is very effective. One example of such a performant system is the CUED Russian->English [125] entry to the Eighth Workshop on Machine Translation [24]. This system was amongst the top-scoring systems.

The final part of the description of an SMT system is the use of the linear model. We have deliberately left this description for the next chapter so that we can provide a thorough analysis. Even though the rest of this work treats the feature function $\mathbf{h}(\mathbf{e}, \mathbf{f})$ as an abstract input into an inner product, we note that the performance of a linear model is ultimately dependent on good quality, informative features and a large hypotheses space.

Chapter 3

Parameter Estimation for Linear Models

In the previous chapter we described the decision rule in Eqn. (2.3) used to select translations under the model scores. We also described the SMT decoder that can be used to evaluate the decision rule given the source sentence and a parameter vector $\mathbf{w}$. In this Chapter we describe methods for estimating $\mathbf{w}$ from data.

In the introduction to Chapter 1 we described two approaches to predicting the quality of a hypothesis. One was based on defining a probability distribution over hypotheses, the other used a model score. These two approaches are labelled by Bishop et al. [20] as discriminatively trained models, and discriminative functions respectively. In addition, Bishop et al. [20] defines a third type of model called the generative model.

In the generative model, the source sentence and target hypotheses are jointly modelled in some distribution $P(\mathbf{f}, \mathbf{e})$. An example of the generative model is the source-channel formulation of Brown et al. [26]. This formulation uses Bayes' rule to compute the probability of a hypothesis given the language and translation models. In practise, the source-channel formulation requires scaling factors that are learnt from data. These scaling factors perform a very similar role to the parameter vector $\mathbf{w}$ in the other approaches.

For the discriminatively trained model a conditional distribution $P(\mathbf{e} \mid \mathbf{f})$ is trained directly. An example of this model is the probabilistic model discussed in the introduction of Chapter 2 [114]. This training method results in a unconstrained continuous objective function.

In this Chapter we focus on the approaches that use discriminative functions. As stated in Chapter 2 these approaches yield model scores instead of probabilities. A function, called a discriminative function, is optimised. For SMT the discriminative function is derived from an automated quality metric. The performance of these models can be very strong because the same metric is used to assess the SMT system.

These approaches rely on each hypothesis being given a quality score by the automated quality metric. It is not possible to enumerate through the very large number of hypotheses that can be produced by the decoder and assign quality scores. Instead, we rely on an iterative scheme. An initial parameter $\mathbf{w}^{(0)}$ is chosen and used in the decoder. The K-best hypotheses are then selected according to their model score. These K are used in one of the training schemes described in this Chapter, and the new parameter $\mathbf{w}^{(1)}$ is used with the decoder to find another K-best hypotheses. The process continues until the decoder no longer finds hypotheses that improve the quality scores. At this point the training method is said to have converged. In Chapter 7 we describe a method to train over word lattices as opposed to K-best lists, which can reduce the number of iterations.

In this Chapter we first describe a selection of quality metrics that can be used for training. In the second section we describe minimum error training, a method for optimising parameters directly against the error metric. In the third section we describe large-margin based methods, and then in the fourth section we describe ranking approaches. Finally we end with a survey of the current methods used for parameter estimation.

3.1 Automated Quality Metrics

In Chapter 1 we discussed some of the problems in defining translation. If we cannot even define translation, then the task of assessing translation quality would seem equally hopeless. This conundrum is circumvented by requiring a set of reference translations to be supplied alongside the test and training sets of source sentences. Translation quality is therefore implicitly defined by the references, and by the translators who produced the references. An automated quality metric is some method of comparing the output of an SMT system with one or more references. There exist many metrics: examples include The BiLingual Evaluation Understudy (BLEU) [117], Translation Error Rate (TER) [133], NIST [53], and Metric for Evaluation of Translation with Explicit ORdering (METEOR) [11].

Quality metrics either report results as a score, where higher results are better, or as an error where lower results are better. By convention, the training methods we describe in this Chapter are described with respect to minimising errors. Scores can be easily converted to errors, for example we can use $1 - \text{BLEU}$ to convert the BLEU score to an error metric.

The use of a quality metric is justified by measuring its correlation with quality scores from a panel of human judges. Arguments are made that some metrics are superior to others because of closer correlations to human judgements. We report most results in BLEU, simply because it is one of the most commonly used metrics. It should be noted that the

training methods described in this chapter can also be used with any of these metrics.

3.1.1 The BiLingual Evaluation Understudy

The BLEU score is based on N-gram precision. The precision at order n is the proportion of N-grams that are matched in the references. The matched count of each N-gram $\mathbf{u}_n = W_1 \dots W_n$ is clipped by truncating it to the maximum number of times it occurs in any single reference. This count clipping prevents artificial inflation of precision by spurious repetition of high order N-grams.

Given S source sentences $\{\mathbf{f}_1, ..., \mathbf{f}_S\}$ a set of S candidate hypotheses $\{\mathbf{e}_1, ...\mathbf{e}_S\}$ is generated for scoring. For each source sentence $\mathbf{f}_s$, there is a set of I reference translations $\{\mathbf{r}_{s,1} ... \mathbf{r}_{s,I}\}$. The precision p_n is computed by summing the matched clipped N-gram counts over the whole set of reference m_n and dividing by the sum of all the N-grams in the candidate hypotheses c_n:

$$p_n = \frac{m_n}{c_n} = \frac{\sum_{s=1}^{S} \sum_{\mathbf{u}_n \in \mathbf{e}_s} \text{count}_{\text{clip}}(\mathbf{u}_n, \{\mathbf{r}_{s,1}, \dots, \mathbf{r}_{s,I}\})}{\sum_{i=1}^{S} \sum_{\mathbf{u}_n \in \mathbf{e}_s} \text{count}(\mathbf{u}_n, \mathbf{e}_s)}$$

High precisions could be obtained by producing very short output. Ideally the candidate translations should match the length of the reference translations. To punish systems that produce short output, the brevity penalty (BP) is used. For each candidate hypotheses $\mathbf{e}_s$ the reference with the closest length $\mathbf{r}_s^*$ is found. The brevity penalty is then a function with respect to the sum of hypothesis lengths C and sum of reference lengths R:

$$\text{BP} = \exp\left(\min\left(0, 1 - \frac{R}{C}\right)\right) = \exp\left(\min\left(0, 1 - \frac{\sum_{i=1}^{S} |\mathbf{r}_s^*|}{\sum_{i=1}^{S} |\mathbf{e}_s|}\right)\right)$$

Finally BLEU is defined as the geometric mean of N-gram modified precisions with respect to one or several translation references multiplied by the brevity penalty.

$$\text{BLEU} = \text{BP} \prod_{n=1}^{N} p_n^{\frac{1}{N}} \tag{3.1}$$

3.1.2 Translation Edit Rate

Translation Edit Rate (TER) [133] is the minimum number of edits required to modify a translation such that it matches one of the references. For multiple references TER is the smallest number of edits needed to transform a translation into any of the references. A

single edit is an insertion, deletion, substitution of a single word, or a shift of a contiguous word sequence to an alternative position in the translation. The total number of edits is then normalised by the average length of the references.

$$\text{TER} = \frac{\text{\# of edits}}{\text{average \# of reference words}} \tag{3.2}$$

Finding the minimum number of edits is an NP-complete problem. An approximation is provided by using a dynamic programming algorithm with a greedy search. Another solution to the edit problem is to use humans to edit the translation to a reference, which is known as human-targeted TER (HTER). This approach is expensive, and difficult to combine with a training method which may need to score many millions of hypotheses.

3.1.3 NIST

The NIST metric [53], named after the National Institute of Standards and Technology, is an N-gram based metric similar to BLEU. The difference to BLEU is that N-gram matches are weighted depending on their occurrences in the references. The weights are computed for each N-gram $\mathbf{u}_n = W_1 \dots W_n$ using the counts in reference translations

$$\text{Info}(\mathbf{u}_n) = \log_2 \frac{\text{count}(W_1 \dots W_n - 1)}{\text{count}(W_1 \dots W_n)} \tag{3.3}$$

These weights replace the clipped counts of the BLEU score. Let $\mathscr{N}_n$ denote set of N-grams in the translation output and $\mathscr{R}_n$ denote the set of N-grams in the references. The *NIST* score is defined as

$$\text{NIST} = \sum_{n=1}^{N} \left\{ \frac{\sum_{\mathbf{u}_n \in \mathscr{N}_n \cap \mathscr{R}_n} \text{Info}(\mathbf{u}_n)}{|\mathscr{N}_n|} \right\} \exp\left\{ \beta \log_2 \left[\min\left(\frac{c}{r}, 1 \right) \right] \right\} \tag{3.4}$$

where N is the maximum N-gram order, c is the candidate translation length, r is the average reference length, and β controls the impact of the brevity penalty. Note that the brevity penalty is different from the BLEU computation because of the average reference length and the use of the β factor.

3.1.4 Metric for Evaluation of Translation with Explicit ORdering

In information retrieval, the quality of a search result is judged using two metrics: precision and recall. Precision is the ratio of correct results in the output vs. the size of the output,

and recall is the ratio of correct results vs. all possible correct results. An automated quality metric has elements of these two metrics. For example, the N-gram count in BLEU is similar to recall, and the brevity penalty enforces precision.

The Metric for Evaluation of Translation with Explicit ORdering (METEOR) [11] computes precision and recall with respect to an explicit one-to-one word alignment of the system output with each reference. Word matching is performed incrementally, first exact matches are aligned, and then stemmed morphological variants and synonyms. Each word-to-word alignment is scored as the product of the weighted harmonic mean of unigram precision P and unigram recall R,

$$F_{\text{mean}} = \frac{PR}{\alpha P + (1-\alpha)R} \tag{3.5}$$

where the weight α controls the emphasis on recall. The output of the SMT system should not just have the same words as the references, but similar phrases too. A tuneable penalty ρ based on chunk fragmentation favours the grouping of adjacent words into longer spanning chunks. The final METEOR score is computed as

$$\text{METEOR} = (1-\rho)F_{\text{mean}} \tag{3.6}$$

3.2 Minimum Error-Rate Training

Minimum Error Rate Training (MERT) [113] is an iterative procedure for training a linear translation model. MERT optimizes model parameters directly against a criterion based on an automated translation quality metric, such as BLEU [117].

Following Och [113], we assume that we are given a tuning set of S parallel sentences $\{(\{\mathbf{r}_{1,1},\ldots,\mathbf{r}_{1,I}\},\mathbf{f}_1),...,(\{\mathbf{r}_{S,1},\ldots,\mathbf{r}_{S,I}\},\mathbf{f}_S)\}$, where $\{\mathbf{r}_{s,1},\ldots,\mathbf{r}_{s,I}\}$, are the reference translations of the source sentence $\mathbf{f}_s$ mentioned in the previous section. Recall that the decoder selects a hypothesis using the decision rule in Eqn. (2.3). Let us reformulate the decision rule as a function that yields a hypothesis given a source sentence $\mathbf{f}_s$ and a D-dimensional model parameter $\mathbf{w}$ vector

$$\hat{\mathbf{e}}(\mathbf{f}_s;\mathbf{w}) = \underset{\mathbf{e}}{\operatorname{argmax}}\, \mathbf{w}\mathbf{h}(\mathbf{e},\mathbf{f}_s) \tag{3.7}$$

For a training set of S source sentences we assume an error function of the form

$$E(\{\mathbf{e}_1,\ldots\mathbf{e}_S\},\{\{\mathbf{r}_{1,1}\ldots,\mathbf{r}_{1,I}\},\ldots,\{\mathbf{r}_{S,1},\ldots,\mathbf{r}_{S,I}\}\}) \tag{3.8}$$

Note that one of the strengths of MERT is that the error is computed over the whole training set. Methods that we describe in later sections, such as MIRA and PRO, are forced into using a sentence level approximation and summing sentence level errors. The final objective function is

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} E(\{\hat{\mathbf{e}}(\mathbf{f}_1;\mathbf{w}),\ldots,\hat{\mathbf{e}}(\mathbf{f}_S;\mathbf{w})\},\{\{\mathbf{r}_{1,1}\ldots,\mathbf{r}_{1,I}\},\ldots,\{\mathbf{r}_{S,1},\ldots,\mathbf{r}_{S,I}\}\}) \tag{3.9}$$

This objective function has two problems with respect to optimisation. The first problem is that the function is not continuous, which excludes an analytical solution. The second is the function is not convex, which excludes convex optimisation methods.

We describe three methods for optimising this objective function. The first, called expected BLEU, approximates the function to a continuous form that can be optimised using gradient descent methods. The next two methods, line optimisation and LP-MERT, optimise the discontinuous form directly.

3.2.1 Line Optimisation

Although the objective function in Eqn. (3.9) cannot be solved analytically, the line optimisation procedure of Och [113] can be used to find an approximation of the optimal model parameters. Rather than evaluating the decision rule in Eqn. (3.7) over all possible points in parameter space, the linear search procedure considers a subset of points defined by the line $\mathbf{w}^{(0)} + \gamma\mathbf{d}$, where $\mathbf{w}^{(0)}$ corresponds to an initial point in parameter space and $\mathbf{d}$ is the direction along which to optimize. Eqn. (3.7) can be rewritten as:

$$\begin{aligned}
\hat{\mathbf{e}}(\mathbf{f}_s;\gamma) &= \underset{\mathbf{e}}{\operatorname{argmax}} \left\{ (\mathbf{w}^{(0)} + \gamma\mathbf{d})\mathbf{h}(\mathbf{e},\mathbf{f}_s) \right\} \\
&= \underset{\mathbf{e}}{\operatorname{argmax}} \left\{ \underbrace{\mathbf{w}^{(0)}\mathbf{h}(\mathbf{e},\mathbf{f}_s)}_{a(\mathbf{e},\mathbf{f}_s)} + \gamma \underbrace{\mathbf{d}\mathbf{h}(\mathbf{e},\mathbf{f}_s)}_{b(\mathbf{e},\mathbf{f}_s)} \right\} \\
&= \underset{\mathbf{e}}{\operatorname{argmax}} \{ \underbrace{a(\mathbf{e},\mathbf{f}_s) + \gamma b(\mathbf{e},\mathbf{f}_s)}_{\ell_{\mathbf{e}}(\gamma)} \}
\end{aligned} \tag{3.10}$$

This decision rule shows that each hypothesis $\mathbf{e}$ is associated with a line defined by γ: $\ell_{\mathbf{e}}(\gamma) = a(\mathbf{e},\mathbf{f}_s) + \gamma b(\mathbf{e},\mathbf{f}_s)$, where $a(\mathbf{e},\mathbf{f}_s)$ is the y-intercept and $b(\mathbf{e},\mathbf{f}_s)$ is the gradient. The optimisation problem can be further simplified by defining the function [118]:

$$\mathrm{Env}(\mathbf{f}) = \max_{\mathbf{e}} \{ \underbrace{a(\mathbf{e},\mathbf{f}) + \gamma b(\mathbf{e},\mathbf{f})}_{\ell_{\mathbf{e}}(\gamma)} : \gamma \in \mathbb{R} \} \tag{3.11}$$

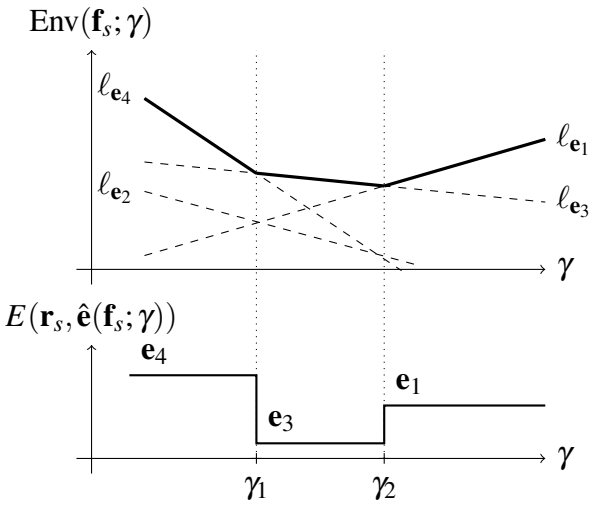


Fig. 3.1 An upper envelope and projected error. Note that the upper envelope is completely defined by hypotheses $\mathbf{e}_4$, $\mathbf{e}_3$, and $\mathbf{e}_1$, together with the intersection points γ_1 and γ_2 (after Macherey et al. [102], Fig. 1).

For any value of γ the linear functions $\ell_{\mathbf{e}}(\gamma)$ take up to K values, due to the K-best list restriction. The function in Eq. (3.11) defines the 'upper envelope' of these values over all γ. The upper envelope has the form of a continuous piecewise linear function in γ. The piecewise linear function can be compactly described by the linear functions which form line segments and the values of γ at which they intersect. The example in the upper part of Figure 3.1 shows how the upper envelope associated with a set of four hypotheses can be represented by three associated linear functions and two values of γ. The line search procedure computes this compact representation of the upper envelope.

The advantage of this procedure is that the error function need only be evaluated for the subset of hypotheses that contribute to the upper envelope. These errors are assigned into intervals of γ, as shown in the lower part of Figure 3.1, so that Eq. (3.9) can be readily solved. This piecewise constant function is called the error surface.

The Sweep Line Algorithm

To compute the upper envelope a method is needed to compute values of γ at which the piecewise function is discontinuous. Macherey et al. [102] propose computing the piecewise function for a single K-best list by using a geometric method called the sweep line algorithm [19].

The piecewise linear function that the upper envelope represents can be described as a convex polygon. This convex polygon is a convex hull of all the points enclosed by the

lines plotted from the functions in the upper envelope. The sweep line algorithm shown in Algorithm 3.1 builds the convex polygon from a set of lines. The convex polygon is stored as a sequence of line objects in an array A. In the algorithmic description the linear function ℓ is represented as a structure with three fields; a gradient m, y-intercept y, and the x-intercept with the adjacent left edge x. Once the algorithm has completed, the values of x contain the interval boundaries γ. An initial A is built from the K candidate hypotheses by building K ℓ structures with gradients and y-intercepts but empty x values.

The algorithm exploits a property of the convex polygon. If a ray is swept from $-\infty$ to $+\infty$ along the convex polygon the gradient of the edges always increases. The first step in computing the convex polygon is to sort the K line objects in A by their gradient m in ascending order. Iterating through the sorted array A searches for edges in the polygon from left to right along the γ axis.

At each iteration of the line objects in ℓ the value for x is computed between the current and previous line objects. If the value of x is less than the value of x of a previous line then it dominates the previous line. The previous line is deleted from A and replaced by the new line. Parallel lines need to be considered separately. There is a check at line 8 that removes any parallel lines with a lower y-intercept than the currently considered line.

The runtime of the algorithm is dominated by the initial sort of gradients. We can therefore compute the upper envelope in $O(K\log(K))$.

Line Optimisation for the Training Set

The objective function in Eqn. (3.9) is defined in terms of a training set of S source sentences, yet we have only described how to compute the upper envelope for a single source sentence. We now describe a method for constructing a training set error surface [102, 113].

The intersections found by the sweep line algorithm mark intervals where the error count is constant. Let $\gamma_1^s < \gamma_2^s < ... < \gamma_{N_s}^s$ denote a sequence of interval boundaries for the sentence $\mathbf{f}_s$. The quantity ΔE_n^s marks the change in error count if γ is moved from a point in the interval $[\gamma_{n-1}^s, \gamma_n^s)$ to a point in the interval $[\gamma_n^s, \gamma_{n+1}^s)$. The interval boundaries and associated error counts over all sentences in the development set are merged into one sequence of interval boundaries. From this merged sequence we can then identify the interval corresponding to the minimised error count along the entire line $\mathbf{w}^{(0)} + \gamma\mathbf{d}$.

At each interval in the merged sequence there are S hypotheses for which the error in Eqn. (3.8) can be computed. For most error metrics a faster computation can be achieved by computing changes in error statistics; the BLEU statistics are the N-gram precision counts (m_n, c_n) and length penalties (R, C). The statistics have the property that they can be summed

Algorithm 3.1 The SweepLine algorithm [19]

```
 1: sort(A : m)
 2: j ← 1
 3: K ← size(A)
 4: for i ← 1 to K do
 5:     ℓ ← A[i]
 6:     ℓ.x ← −∞
 7:     if 1 < j then
 8:         if a[j − 1].m = ℓ.m then
 9:             if ℓ.y <= A[j − 1].y then
10:                 continue
11:             end if
12:             j ← j − 1
13:         end if
14:         while 1 < j do
15:             ℓ.x = (ℓ.y − A[j − 1].y)/(A[j − 1].m − ℓ.m)
16:             if A[j − 1].x < ℓ.x then
17:                 break
18:             end if
19:             j ← j − 1
20:         end while
21:         if 1=j then
22:             ℓ.x ← −∞
23:         end if
24:         j ← j + 1
25:         A[j] ← ℓ
26:     else
27:         j ← j + 1
28:         A[j] ← ℓ
29:     end if
30: end for
```

over sentences, whereas the final error metric cannot.

3.2.2 Choosing Directions for Line Optimisation

The previous algorithm finds the upper envelope along a particular direction in parameter space over hypothesis sets. Because the error function is piecewise-constant and non-convex, choosing the direction to optimise over is difficult.

A common approach to MERT is to select the direction $\mathbf{d}$ using Powell's method [126]. A line optimisation is performed on each coordinate axis. The axis that gives the largest decrease in error is replaced with a vector between the initial parameters $\mathbf{w}^{(0)}$ and the optimised parameters. Powell's method halts when no further decreases in error are reported.

Instead of using Powell's method, the Downhill Simplex algorithm [126] can be used to explore the criterion in Eq. (3.9). This is done by defining a simplex in parameter space. Directions where the error count decreases can be identified by considering the change in error count at the points of the simplex. This has been applied to parameter searching over K-best lists [157].

Both Powell's method and the Downhill Simplex algorithms are approaches based on heuristics to select lines. There is no theoretical reason why they are a good predictor of local minima in the BLEU function. Therefore Cer et al. [29] choose the direction vectors $\mathbf{d}$ at random.

Instead of performing a single line optimisation, which may explore a region of high error, many line optimisations could be performed simultaneously. All these lines originate from the same initial vector $\mathbf{w}^{(0)}$. This is the approach taken by Macherey et al. [102], where the coordinate axes and a number of random directions are all explored and $\mathbf{w}^{(0)}$ is updated with the best result.

3.2.3 LP-MERT

More recently, Quirk and Galley [62] have introduced LP-MERT as an exact search algorithm that reaches the global optimum of the training criterion using linear programming. The algorithm first identifies the *extreme points* within the convex hull of feature vectors associated with a K-best list of K translations. Specifically, for a sentence $\mathbf{f}_s$, let $\mathbf{h}_{s,i} = \mathbf{h}(\mathbf{e}_{s,i}, \mathbf{f}_s)$ be the feature vector associated with the ith hypothesis $\mathbf{e}_{s,i}$ in the K-best list for the sth sentence. This hypotheses can be selected by the decoder only if the following holds

$$\mathbf{w}(\mathbf{h}_{s,j} - \mathbf{h}_{s,i}) \leq 0 \text{ for } 1 \leq j \leq K \tag{3.12}$$

for some parameter vector $\mathbf{w}$ ($\mathbf{w} \neq \mathbf{0}$). Any feature vector $\mathbf{h}_{s,i}$ that satisfies these constraints is denoted an extreme feature vector. This system of inequalities defines a convex region in parameter space, and for each parameter value in the region the decoder will select the associated hypothesis $\mathbf{e}_{s,i}$ from the K-best list.

The system of inequalities in (3.12) can be solved using a linear program of the following form

$$\begin{aligned} &\text{maximise} && \mathbf{w}(\mathbf{h}_{s,i} - \mathbf{h}_{\mu}) \\ &\text{subject to} && \mathbf{w}(\mathbf{h}_{s,j} - \mathbf{h}_{s,i}) \leq 0,\ 1 < j < K \end{aligned} \tag{3.13}$$

Where $\mathbf{h}_{\mu} = \frac{1}{K}\sum_{k=1}^{K} \mathbf{h}_k$ is the centroid of the feature vectors in the K-best list. If the feature vector $\mathbf{h}_{s,i}$ is not extreme, then the only parameter to satisfy these constraints is the zero vector $\mathbf{0}$. If the feature vector is extreme then $\mathbf{w}$ is set to some parameter that satisfies the system of inequalities in (3.12). Such a problem is called a linear programming feasibility problem, and if $\mathbf{h}_{s,i}$ is extreme then the system of inequalities in (3.12) is denoted as feasible.

Sets of feature vectors can also be denoted as separable, and this term is closely related to feasibility. In a separable problem the parameter $\mathbf{w}$ defines a decision boundary which separates $\mathbf{h}_{s,i}$ from the other $K-1$ feature vectors. The existence of such a decision boundary implies that a feasible set of inequalities exist.

The runtime complexity of a linear program is linear with respect to the number of constraints and polynomial with respect to the dimension of $\mathbf{w}$ [84], which results in $O(KD^{3.5})$ for the testing of a single feature vector [62]. The testing of an entire K-best list is therefore $O(K^2D^{3.5})$.

Each feasible parameter for the feature vector $\mathbf{h}_{s,i}$ is associated with a hypothesis $\mathbf{e}_i$, for which an error can be computed. Errors only need to be computed for feasible parameters.

A Note on Linear Programming

We now interrupt the description of LP-MERT to discuss a problem with the solution of linear programs: the parameter that maximises the linear objective for a feature vector will also yield an equal maximal model score for at least one other feature vector in the K-best list. The error function in Eqn. 3.9 is undefined for such a parameter because we now have more than one input hypothesis.

We note that the constraint set in (3.12) form a convex set, and that this convex set has a geometric representation. The constraint set thus has its own extreme points, formed where multiple constraints intersect. This geometric description of the constraints is further explored in chapters 5 and 6.

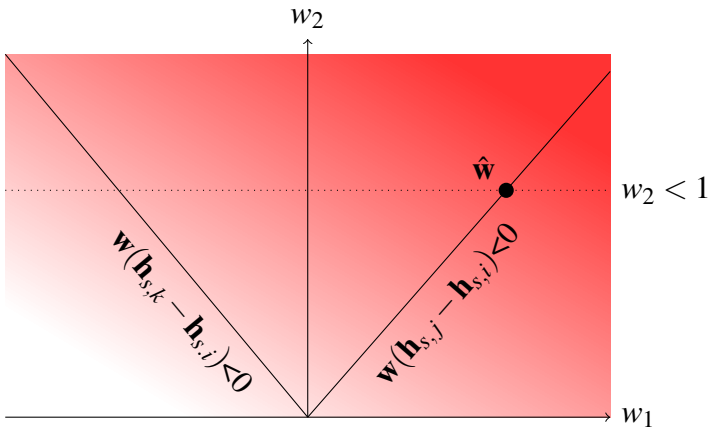


Fig. 3.2 An illustration of the fundamental theorem of linear programming. The gradient of the shaded region represents the linear function to be optimised. The two solid lines represent constraints from (3.12) and the dotted line is a bounding constraint to ensure a well-formed solution. The dot marked $\hat{\mathbf{w}}$ is the optimal parameter vector found at an extreme point of the constraint set.

Let us now invoke the the fundamental theorem of linear programming [101]

Theorem 3.1. *If this is a finite optimal solution to a linear programming problem, there is a finite optimal solution which is an extreme point of the constraint set.*

Proof. This geometric form of the theorem is shown in Corollary 2 in Chapter 2 of Luenberger and Ye [101]. □

We illustrate this problem with an example shown in Figure 3.2. In this example we have a two dimensional feature vector $\mathbf{h}_{s,i}$ being tested for extremity. There are two other feature vectors, $\mathbf{h}_{s,j}$ and $\mathbf{h}_{s,k}$, that constrain the region of parameter space where $\mathbf{h}_{s,i}$ is maximal. We have also applied the constraint $w_2 < 1$ to ensure a well-formed solution. The linear objective function is represented as the gradient of the shaded region. The darker the colour, the greater the value of the objective function. The optimum value $\mathbf{w}^*$ is found at the intersection of two constraints. The problem with this solution is that both $\mathbf{h}_{s,i}$ and $\mathbf{h}_{s,j}$ are extreme at $\hat{\mathbf{w}}$. The error function is undefined for $\hat{\mathbf{w}}$ because we now have two possible input hypotheses.

Galley and Quirk [62] note that there is an alternative formulation for finding extreme

points [60]. Let us examine this formulation

$$\begin{aligned} &\text{find} && y \in \mathbb{R} \text{ and } \mathbf{w} \in \mathbb{R}^D \\ &\text{satisfying} && \mathbf{w}\mathbf{h}_{s,j} \leq y,\ 1 \leq j \leq K, i \neq j \\ & && \mathbf{w}\mathbf{h}_{s,i} > y \\ & && y \geq 0 \end{aligned} \tag{3.14}$$

To be a linear program, this formulation needs to be expressed as the maximisation of a linear objective

$$\begin{aligned} \hat{f} = \text{maximise} \quad & \mathbf{w}\mathbf{h}_{s,i} - y \\ \text{subject to} \quad & \mathbf{w}\mathbf{h}_{s,j} - y \leq 0,\ 1 \leq j \leq K, i \neq j \\ & \mathbf{w}\mathbf{h}_{s,i} - y \leq 1 \end{aligned} \tag{3.15}$$

where $\hat{f}$ is the optimal value of the linear objective and the last inequality is added to ensure a bounded solution. If the solution is strictly positive ($\hat{f} > 0$) then the point $\mathbf{h}_{s,i}$ is extreme. Recall that this linear program performs two functions. The first is as a test for extremity of a feature vector. In this respect the formulation of Galley and Quirk [62] and the formulation in (3.15) identify the same extreme feature vectors.

The second function is to yield a parameter vector associated with the extreme point, and here the two linear programs yield different results. The inclusion of the y variable in the constraints encourages the optimiser to not place the optimal parameter on a constraint, which is a much more robust solution.

3.2.4 LP-MERT for the Training Set

Multiple K-best lists lead to more serious computational challenges for LP-MERT. Let $\mathbf{i}$ be an index vector that contains S elements. Each element is an index i_s to a hypothesis and a feature vector for the sth sentence. LP-MERT builds a set of N^S feature vectors associated with index vectors of the form $\mathbf{h}_\mathbf{i} = \mathbf{h}_{1,i_1} + \ldots + \mathbf{h}_{S,i_S}$ which are then substituted into the system of inequalities in (3.12). It is possible to reformulate the linear program to avoid N^S constraints [62], but there are still N^S such feature vectors.

A feature vector $\mathbf{h}_\mathbf{i}$ can be discarded if any one of the sentence level future vectors $\mathbf{h}_{i_s}$ is not extreme. We can therefore lazily enumerate the feature vectors [62]. For example if $\mathbf{h}_{1,1} + \mathbf{h}_{2,1}$ is not extreme, then neither will be $\mathbf{h}_{1,1} + \mathbf{h}_{2,1} + \mathbf{h}_{3,1}$ or any other sum which contains the terms $\mathbf{h}_{1,1}$ and $\mathbf{h}_{2,1}$.

The lazy enumeration of feature vectors is managed through a divide and conquer algorithm [62]. Many index vectors share smaller subsets of common indices. The algorithm

identifies these common subsets and tests them for extremity. If any subsets are not extreme, then whole sets of index vectors that share common subsets can be discarded.

This algorithm does greatly reduce the number of extremity tests, but Galley and Quirk [62] do not provide a complexity analysis for either runtime or memory consumption. Even with this scheme, they found that exhaustive testing is impractical and heuristics are used to avoid testing them all.

In Chapter 6 we describe the Minkowski sum algorithm of Fukuda [59], which also identifies feasible index vectors. This algorithm has a polynomial time upper bound for the runtime, a constant memory guarantee, and can be easily parallelized. The algorithm also predates the algorithm of Galley and Quirk [62] by many years.

3.3 Large-Margin Methods

Let us define a sentence-level error function of the form that computes the error for the sth sentence

$$E(\mathbf{e}_s, \{\mathbf{r}_{s,1} \ldots, \mathbf{r}_{s,I}\}) \tag{3.16}$$

Such a function can be implemented by computing BLEU for each individual sentence. Note that the sum of sentence-level BLEU is not equal to the corpus-level equivalent. Using the same index vector notation from the description of LP-MERT in Section 3.2.3, a set of S oracle hypotheses are indexed with the vector $\hat{\mathbf{i}}$.

$$\hat{i}_s = \operatorname*{argmin}_i E(\mathbf{e}_s, \{\mathbf{r}_{s,1} \ldots, \mathbf{r}_{s,I}\}) \text{ for } 1 \leq s \leq S \tag{3.17}$$

In this section we describe large-margin methods for SMT. In general, these methods select the oracle hypotheses indexed by $\hat{\mathbf{i}}$, and then relax a set of constraints until a feasible parameter vector is found. This is in contrast to MERT where infeasible hypothesis sets are rejected outright.

We start with a description of the structured Support Vector Machine (SVM) [141, 144, 145] which has been applied to SMT [34]. They are a coherent expression of the large-margin concept and provide a clear contrast to MERT.

We then move onto the Margin Infused Relaxed Algorithm (MIRA) [45], an online large-margin algorithm. MIRA has been applied, with different formulations, to SMT [34, 40, 151] and dependency parsing [105].

3.3.1 The Structured Support Vector Machine

Following Tsochantaridis et al. [145] the structured SVM maximises the following problem

$$\begin{aligned} &\text{maximise} && y \\ &\text{subject to} && \mathbf{w}(\mathbf{h}_{s,j} - \mathbf{h}_{s,\hat{i}_s}) + y \leq 0, \quad 1 \leq j \leq K, 1 \leq s \leq S, \hat{i}_s \neq j \\ & && \|\mathbf{w}\| = 1 \end{aligned} \tag{3.18}$$

This problem attempts to find a parameter that maximises a margin across all the constraints, hence the name large-margin problem. Note the resembelance between this formulation and the constraints defined for LP-MERT. The solution for this formulation of the structured SVM also satisfies the system of inequalities for LP-MERT in Eqn. (3.12). This form of the SVM will always yield a feasible parameter or fail.

With a small amount of algebraic manipulation - see Bishop et al. [20, Page 327] for a description - the problem can be reformulated as an equivalent quadratic programming problem

$$\begin{aligned} &\underset{\mathbf{w}}{\text{minimise}} && \frac{1}{2}\|\mathbf{w}\|^2 \\ &\text{subject to} && \mathbf{w}(\mathbf{h}_{s,j} - \mathbf{h}_{s,\hat{i}_s}) + 1 \leq 0, \quad 1 \leq j \leq K, 1 \leq s \leq S, \hat{i}_s \neq j \end{aligned} \tag{3.19}$$

This formulation is called the hard-margin SVM, because the system of inequalities in the constraints must be feasible. If the oracle index vector $\hat{\mathbf{i}}$ does not give a feasible system, then the constraints must be altered. The solution is to introduce a set of slack variables $\{\xi_1, \ldots, \xi_S\}$ that allow the constriants to be violated in proportion to the increase of error caused by the violation [140]

$$\begin{aligned} &\underset{\mathbf{w}}{\text{minimise}} && \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{S}\sum_{s=1}^{S} \xi_s \\ &\text{subject to} && \mathbf{w}(\mathbf{h}_{s,j} - \mathbf{h}_{s,\hat{i}_s}) + \Delta(\mathbf{e}_{s,\hat{i}_s}, \mathbf{e}_{s,j}) - \xi_s \leq 0, \quad 1 \leq j \leq K, 1 \leq s \leq S, \hat{i}_s \neq j \\ & && \xi_s \geq 0 \end{aligned} \tag{3.20}$$

where $\Delta(\mathbf{e}_{s,\hat{i}_s}, \mathbf{e}_{s,j}) = E(\mathbf{e}_{s,i_s}, \{\mathbf{r}_{s,1} \ldots, \mathbf{r}_{s,I}\}) - E(\mathbf{e}_{s,j}, \{\mathbf{r}_{s,1} \ldots, \mathbf{r}_{s,I}\})$ and $C > 0$ is some constant that controls the trade-off between error minimisation and margin maximisation.

This constrained problem is not solved directly, instead Lagrange multipliers are used to create a single Lagrangian function that incorporates the constraints into the objective function. Because of the large number of constraints, the Lagrangian cannot also be solved directly either. Instead, Tsochantaridis et al. [145] describe an algorithm which iteratively tightens relaxations of the original problem.

Each one of the S source sentences gives $K-1$ constraints. The algorithm iterates over

these sets and identifies, for the current $\mathbf{w}$, which of the $K-1$ constraints is most violated. This constraint is then added to a working set and $\mathbf{w}$ is recomputed. The algorithm continues until a solution is found that does not violate the constraints beyond some precision.

3.3.2 Margin Infused Relaxed Algorithm

We now move to online version of large-margin training called MIRA [45]. For a given s the objective of this algorithm is

$$\begin{aligned} \underset{\mathbf{w}^{(n+1)}}{\text{minimise}} \quad & \frac{1}{2}\|\mathbf{w}^{(n+1)} - \mathbf{w}^{(n)}\|^2 + C\sum_{j=1}^{K}\xi_j \\ \text{subject to} \quad & \mathbf{w}(\mathbf{h}_{s,j} - \mathbf{h}_{s,\hat{i}_s}) + \Delta(\mathbf{e}_{s,\hat{i}_s}, \mathbf{e}_{s,j}) - \xi_j \leq 0, \quad 1 \leq j \leq K, \hat{i}_s \neq j \\ & \xi_j \geq 0 \\ & \xi_{\hat{i}_s} = 0 \end{aligned} \tag{3.21}$$

In resemblance to the algorithm of Tsochantaridis et al. [145], MIRA approximates a solution by ignoring all except the most violated constraint. The hypothesis associated with this constraint is sometimes called the fear hypothesis [37]. As with the structured SVM, the Lagrangian is solved with a quadratic programming solver.

Because the algorithm is online, the K-best lists are not fixed. At each update of the parameter vector a new K-best list is generated from the decoder, allowing for a new fear hypothesis to be chosen.

Watanabe et al. [151] make use of the online nature of the algorithm to compute a more accurate sentence level error loss. The sth element of $\hat{\mathbf{i}}$ is replaced with the 'fear' hypotheses and the error computed using the error function in Eqn. (3.8). Chiang [37] changes the oracle hypothesis based on a linear combination of both BLEU and model score. This new hypotheses is called the *hope hypothesis*.

For fast computation MIRA requires a parallelised implementation with weight vectors computed on different CPU cores and averaged when computation is complete [40]. This implementation is difficult to build, so both Cherry and Foster [34], and Gimpel and Smith [67] describe simplified versions of MIRA where the hope and fear hypotheses are selected from reranked K-best lists instead of new hypotheses generated from the decoder. These methods are called batch-MIRA and RAMPION respectively.

3.4 Ranking Methods

We have previously described MERT and large-margin methods. Both these methods identify an index vector **i** of 1-best hypotheses that minimise the error across a training set. In this section we consider not just the 1-best, but all the hypotheses in the K-best lists. Our goal is to find a parameter vector $\mathbf{w}$ that ranks the hypotheses in the K-best lists with respect to the sentence level error in Eqn. (3.16).

Hopkins and May [76] note that for the sth source sentence, the parameter $\mathbf{w}$ that correctly ranks its K-best list must satisfy the following set of constraints

$$\mathbf{w}(\mathbf{h}_{s,j} - \mathbf{h}_{s,i}) \leq 0 \text{ for } 1 \leq i \leq K,\ 1 \leq j \leq K,\ \Delta(\mathbf{e}_{s,i}, \mathbf{e}_{s,j}) \geq 0 \tag{3.22}$$

where $\Delta(\mathbf{e}_{s,i}, \mathbf{e}_{s,j})$ is the difference in error between two hypotheses. Now consider the difference vectors $(\mathbf{h}_{s,j} - \mathbf{h}_{s,i})$ associated with each constraint. These difference vectors can be used as input vectors for a binary classification problem. The class label of the difference vector is assigned based on whether the difference in error $\Delta(\mathbf{e}_{s,i}, \mathbf{e}_{s,j})$ is positive or negative. Because there are SK^2 difference vectors across all source sentences, a subset is sampled. Hopkins and May [76] call this algorithm Pairwise Ranking Optimisation (PRO).

Transforming the problem into a binary classification problem allows for any binary classification algorithm to be used. There are many more binary classification algorithms than structured prediction algorithms, with much better guarantees of performance.

Variants of PRO have been proposed. Bazrafshan et al. [13] transform the binary classification problem into a regression problem, such that the $\mathbf{w}(\mathbf{h}_{s,j} - \mathbf{h}_{s,i})$ should be of a similar magnitude to $\Delta(\mathbf{e}_{s,i}, \mathbf{e}_{s,j})$. Watanabe [150] proposes an online algorithm large-margin algorithm subject to the constraints in (3.22).

3.5 Other Methods

In the previous three sections we described three families of methods to estimate parameters for SMT. We have based this division of methods on the objectives and constraints used in these methods. Our aim has to highlight what is common, namely the constraints that bound the objectives, rather than what differs between all these methods. This summary is far from exhaustive, and the task of parameter estimation has been approached from different perspectives.

One solution to optimising Eqn. (3.7) is to approximate it with a continuous function, which is the motivation behind expected BLEU [132]. The log of the BLEU function in

Eqn. (3.1) can be written as

$$\min\left(0, 1 - \frac{R}{C}\right) + \frac{1}{N}\sum_{n=1}^{N}(\log m_n - \log c_n) \tag{3.23}$$

where all the variables have the same denotations as the description of BLEU in Section 3.1.1. The expectation of this function is taken with respect to the log-linear model in Eqn. (2.1). This expectation is also difficult to solve analytically so a Taylor approximation is made. For example, the first order approximation is

$$\min\left(0, 1 - \frac{R}{\mathbb{E}[C]}\right) \sum_{n=1}^{N}(\log \mathbb{E}[m_n] - \log \mathbb{E}[c_n]) \tag{3.24}$$

This function can be differentiated for the computation of a gradient for an iterative solution. This gradient has also been used to set directions for MERT [63].

Instead of smoothing the loss function, it is possible to use stochastic subgradient descent methods [128] to optimise the discontinuous objective of the support vector machine. The result is an iterative algorithm that is similar to stochastic gradient descent.

Chiang [37] notes that updates in one dimension may have much more impact that others. For example, the language model is a very informative feature that should be tuned conservatively. This insight guides AROW, an algorithm similar to MIRA that can adapt the scale of its update along different dimensions.

The importance of adapting updates was also noted by Green et al. [71] who use AdaGrad with stochastic gradient descent. Because stochastic gradient descent requires a smooth function, a logistic form of PRO is used as the objective.

3.6 Survey of Recent Work

One avenue of SMT research has been the adding of as many features as possible to the linear model, especially in the form of sparse features. For example, Chiang et al. [39] describe 11,001 new features for machine translation and use MIRA to train the model. The assumption is that the addition of new features will improve translation performance.

We have described three approaches to parameter tuning for SMT: MERT, large-margin, and ranking approaches. The natural question to ask is which method has the best performance, especially when the problem has a large number of sparse features. Comparing different methods is difficult because small implementation details can greatly affect optimiser performance. Building a strong version of an optimiser involves much work in fixing implementation details and building two strong versions of two different optimisers is a significant engineering challenge. Recently, we have started to see comparisons of optimiser

performance where great care has been taken to build strong baselines.

Hopkins and May [76] created synthetic high dimensional training and test sets to demonstrate that MERT is unsuitable for systems with many features. For a system of tens of features they show parity in the performance between MERT, MIRA, and PRO. For a system with thousands of features, they claim the MERT does not converge and show parity between MIRA and PRO.

Since the work of Hopkins and May [76], progress has been made on improving the line selection for MERT. Flanigan et al. [57] use a complex scheme where a gradient is computed for sparse features in a development set based on a smoothed version of RAMPION, and then this gradient is used in a MERT line search in a second development set. Gains over standard MERT are found, but Flanigan et al. [57] do not provide PRO or MIRA baselines.

Galley et al. [63] show that standard MERT can scale to many features if regularised and search directions based on the gradient of expected BLEU are chosen. Using a system with thousands of features they show a parity in performance with PRO. Both this result and the results of Flanigan et al. [57] support an argument that MERT can be used for sparse features if the directions are chosen with care.

Cherry and Foster [34] finds that batch-MIRA outperforms PRO and structured SVMs for a number of systems build across language pairs. The difference in performance was small, with improvements in the range of $0.1 - 0.3$ BLEU points. Similarly Gimpel and Smith [67] show a parity in performance between MERT, MIRA, and RAMPION for small feature sets, and a parity in performance between MIRA and RAMPION for large feature sets.

It is interesting to read the justification for many of these works as stated in their abstracts. For example Hopkins and May [76] state that:

> Moreover, unlike recent approaches ..., PRO is easy to implement. .... We establish PRO's scalability and effectiveness by comparing it to MERT and MIRA and demonstrate parity on both phrase-based and syntax-based systems in a variety of language pairs, using large scale data scenarios.

Cherry and Foster [34] state:

> Among other results, we find that a simple and efficient batch version of MIRA performs at least as well as training online.

Along similar lines Gimpel and Smith [67] state:

> [We] present a training algorithm that is easy to implement and that performs comparable to others.

In defence of MERT, Galley et al. [63] state:

> Experiments with up to 3600 features show that these extensions of MERT yield results comparable to PRO, a learner often used with large feature sets.

What is striking, is that none of these authors claim that their methods outperform other methods in terms of quality of output. Approaches are justified by being easier to implement, or to are less sensitive to initialisation of parameters. We also note that Green et al. [71] have cast doubt on the effectiveness of estimating the parameters for a large number of sparse features:

> Adaptation of discriminative learning methods for these types of features to statistical machine translation (MT) systems, which have historically used idiosyncratic learning techniques for a few dense features, has been an active research area for the past half-decade. However, despite some research successes, feature-rich models are rarely used in annual MT evaluations. For example, among all submissions to the WMT and IWSLT 2012 shared tasks, just one participant tuned more than 30 features [30]. Slow uptake of these methods may be due to implementation complexities, or to practical difficulties of configuring them for specific translation tasks (Gimpel and Smith [67], Simianer et al. [131], *inter alia*).

These quotes raise two questions. Why do all these methods give similar performance in research settings? And why do these research successes not translate to improvements in MT evaluations? We believe that the answers are due to feasibility. If the oracle index vector $\hat{\mathbf{i}}$ is feasible then a representative parameter can be found with a hard-margin method. The loosening of constraints is unnecessary, because the constraints are all feasible.

We note that the oracle index vector differs depending on the training method being used. The MERT oracle lowers the corpus level error, and the SVM and MIRA oracles differ because of the hope hypothesis. However, the oracle index vectors should still be similar, or index similar hypotheses with low error counts.

We believe that as the feature dimension increases, the chance of any given index vector being feasible also increases. In Chapter 6 we present theoretical evidence that supports this argument.

Chapter 4

Fast Model Parameter Estimation and Filtering for Large Datasets

In the previous two chapters we described models for SMT and how to estimate their parameters. In this Chapter we focus on the implementation of these models and parameter estimation methods. Most research for building practical SMT systems has focussed on building fast and efficient decoders, and there exist many mature and widely used open-source decoder implementations [81, 90, 97]. We instead focus on the many tasks that need to be completed before we can start decoding.

Building and processing SMT models is not a trivial procedure, and because larger models yield better results [123] we must build processing systems that can scale to large amounts of data. Developing software to process this data is an arduous task and requires proficiency with modern programming languages and computer systems as well as a deep understanding of the models being built.

A welcome development in SMT has been the introduction of software frameworks from industry [31, 49]. Using these frameworks cuts down development effort and helps researchers adopt industrial best-practice. The issue with general-purpose frameworks is that they may not have all the features necessary to support specific models, or may not be able to scale to the demands of large data. If this is case then the framework cannot be used and custom solutions have to built. In this chapter we describe a general approach based on MapReduce [49] and HFiles [5] that can be applied to many of the problems in SMT with comparable processing times and memory characteristics to custom solutions.

Our contribution is to show that fast and efficient implementations of SMT models can be built using general purpose industrial frameworks. We can match or even exceed the speed and memory consumption of custom built solutions. Our conclusion is that an im-

plementation of a model using an off-the-shelf tool is usually 'good enough' and requires much less time to develop than a custom solution.

Many of these software frameworks have open-source implementations provided by the Hadoop project [6]. We believe that framework is too general a term, and it is helpful to split the description of these frameworks into three components:

Programming Models A loose formalism of how software should be structured for certain tasks to achieve simple and fast programs. These models are also called design patterns [64].

Software Toolkits Sets of libraries and executables that capture reusable software components shared between many disparate tasks.

Distributed Computation Systems A method of running complex tasks across many different machines. They usually take the form of a set of processes that run on many machines and communicate via interprocess-communication. These processes are responsible for scheduling tasks, and storing and serving data.

The software frameworks are best described in the context of a practical problem. In this chapter we use the Stupid Backoff language model [25] as an example to base our description. The Stupid Backoff N-gram relaxes the constraint that probabilities be properly normalised. To reflect this the probability $P(\cdot)$, for the N-gram, is replaced with the pseudo-probability $S(\cdot)$. It has the form:

$$S(W_i \mid W_{i-n+1}^{i-1}) = \begin{cases} \frac{f(W_{i-n+1}^{i})}{f(W_{i-n+1}^{i-1})} & \text{if } f(W_{i-n+1}^{i}) > 0 \\ \alpha S(W_i \mid W_{i-n+2}^{i}) & \text{otherwise} \end{cases} \tag{4.1}$$

where the function $f(\cdot)$ describes the number of times the N-gram was observed in the language model training data and α is some constant independent of the N-gram history . This language model is simple, but when the N-grams are estimated from a large amount of data it has similar performance to more sophisticated models [25].

Building the Stupid Backoff model involves processing a very large monolingual corpus contained in a file on disk, and counting the N-grams contained in the file. Once we have the model, we only need a small fraction of the N-grams to compute a score for a hypothesis. This is a general pattern with SMT models. Once the model parameters have been estimated, the decoder only needs a fraction of the information contained in those models to be able to translate a specific input source sentence or a set of input source sentences. In this Chapter

we describe both how to build the models, and then how to filter them for an individual source sentence or test set.

In the first section we describe related work. The description covers highly customised solutions to other applications of software frameworks. In the next two sections we describe how to use the software frameworks to build a Stupid Backoff language model rescorer for translation lattices [21]. We start with a description of MapReduce [49], which is a batch processing framework suited to estimating model parameters. We then describe the HFile format, derived from components used in Bigtable [31], to quickly filter parameters for a test set. For the filtering experiments we compare the performance of the general purpose HFile against a custom built solution based on the suffix array, which is a specialised data structure designed to be used for text processing.

In the fourth Section we consider an existing system [122, 124] based on MapReduce and HFiles that is used to perform the translation rule extraction process described in Section 2.2.3. We demonstrate how a selection of "tricks" and refinements can be deployed to give an order of magnitude improvement in terms of memory and runtime. Finally we end with conclusions.

4.1 Related Work

In the introduction we stated that we had two tasks, estimating parameters and then filtering them. Let us consider the types of models seen in Chapter 2. Parameter estimation can take the form of expectation-maximisation statistics for word alignments, relative frequency counts for rule extraction, or a modified relative frequency count for language models. In all cases, some linguistic phenomena is extracted from text, counted, and then aggregated over the whole corpus.

These tasks are very suited to a batch processing framework such as MapReduce, and there has much preceding work applying MapReduce for parameter estimation. For language model parameter estimation, Brants et al. [25] describe how to compute different types of language models with MapReduce, including Kneser-Ney and Stupid Backoff. Word alignments and phrase-based conditional rule probabilities have been computed using MapReduce by Dyer et al. [55]. Software for computing hierarchical phrase-based models using MapReduce is also readily available [97, 147, 152].

Once the parameters have been estimated, we need to find some fast efficient way to filter the parameters for a test set. In general, we have a set of keys which could be N-grams, word alignments, or rules. Each key is associated with a value, such as an N-gram

count, or a translation rule probability. Given a source sentence we can extract a set of keys, which we call the query. For this query we need to find the set of associated values.

General key-value problems are a widely studied problem in computer science, with many data structures and algorithms available. Strategies include storing the model as a simple data structure in memory, in a plain text file, in more complicated data structures in memory, storing fractions of the entire model, storing the raw data as opposed to a precomputed model or storing models in a distributed fashion.

If small enough, it may be possible to fit the model into physical memory. In this case the model can be stored as a memory associative array, such as a hash table, for rapid query retrieval. In-memory storage has been used to store translation tables between iterations of expectation-maximisation for word alignment [55, 99].

For larger models, the set of key-value pairs can be stored as a table in a single text file on local disk. Values for keys in the query set are retrieved by scanning through the entire file. For each key in the file, its membership is tested in the query set. This is the approach adopted in the Joshua 3.0 decoder [152], which uses regular expressions or N-grams to test membership of translation rules. Venugopal and Zollmann [147] use MapReduce to scan a file of translation rules concurrently: a mapper is defined that tests if the vocabulary of a rule matches the vocabulary of a test set. The MapReduce framework then splits the file into subsections for the mappers to scan over in parallel.

The parameters can also be stored using a trie associative array [58]. A trie is a type of tree where each node represents a shared prefix of a set of keys represented by the child nodes. Each node only stores the prefix it represents. The keys are therefore compactly encoded in the structure of the trie itself. Querying the trie is a $O(\log(n))$ operation, where n is the number of keys in the dataset. The trie may also be small enough to fit in physical memory to further reduce querying time. Tries have been used for storing phrase tables [158] and hierarchical phrase-based grammars [65] as well as language models [74, 121].

One method of handling the large number of parameters associated with high order language models is to identify and remove parameters that have minimal impact on the predictive ability of the model. The smaller set of model parameters can then be stored in memory, which makes computing an N-gram probability fast. Count frequency cut-offs, probability quantisation and entropy based pruning [136] are methods used to reduce the size of language models. These methods are often used in the first pass of a two-pass SMT system. In the second pass the K-best lists or lattices are rescored with the larger language model, possibly filtered to cover the words in the K-best lists.

It is also possible to create a much smaller approximate version of the full model. Ran-

domised language models [137, 139] store parameters or counts associated with N-grams in a structure similar to a Bloom filter [22]. This structure is small in comparison to the original language model, although the reduction in size comes at the cost of randomly corrupting model parameters or assigning model parameters to unseen N-grams. Guthrie and Hepple [73] propose an extension which prevents the random corruption of model parameters but does not stop the random assignment of parameters to unseen N-grams.

Another way of building a smaller approximate version of a model is to retain items with high frequency counts from a stream of data [104]. This technique has been applied to language modelling [70] and translation rule extraction [127]. Levenberg and Osborne [95] combine randomised language models with stream-based language models.

Another solution for fast filtering of the full language model during decoding is to use a distributed computing approach based on a client-server paradigm. In this approach, decoder clients connect to one or more remote language model servers and request N-gram probabilities or counts. This approach allows for the language model parameters to be stored in memory on multiple machines.

Instead of pre-computing the dataset it is possible to compute the sufficient statistics at query time using the suffix array [103] data structure. A suffix array is a sequence of pointers to each suffix in a training corpus. The sequence is sorted with respect to the lexicographic order of the referenced suffixes. Suffix arrays have been used for computing statistics for language models [161], phrase-based systems [27, 160], and hierarchical phrase-based systems [100].

Finally, some approaches store language models in a distributed fashion. Brants et al. [25] describe a distributed, fast, low-latency infrastructure for storing very large language models. Zhang et al. [159] propose a distributed large language model backed by suffix arrays. HBase has also been used to build a distributed language infrastructure [155]. The method we propose to use is closely related to the latter but we use a more lightweight infrastructure than HBase and we apply it to two different tasks, demonstrating the flexibility of the infrastructure.

4.2 MapReduce

MapReduce is a framework for the batch processing of data concurrently on many machines in a compute cluster. It is both a programming model and a distributed computing system to support this programming model. First we describe MapReduce as a programming model.

MapReduce was inspired by functional programming [28]. It has been noted that func-

tional languages are well suited to concurrent execution [7]. There are two characteristics of functional languages that make them amenable for concurrent execution. The first is that data is immutable: a variable in a functional program never changes its value. The second characteristic is that functions are strictly defined. A function transforms some input data into an output value. The function is not allowed to alter the input data. A function is also forbidden from exhibiting side-effects, such as writing to the filesystem or altering some other variable unrelated to the output.

Because variables are immutable, the functional language runtime can share the variable between many execution contexts. Because functions are strictly defined they can be executed in any order, or concurrently, without affecting results or creating bugs due to out of sequence execution. Even though we may use an imperative language, such as C or Java, we can get the same benefits of a functional language by restricting ourselves to a similar style.

Functional languages provide two families of inbuilt functions that Dean and Ghemawat [49] classified as map and reduce functions. Let us describe these functions in the context of the Stupid Backoff model. First, consider the problem of counting the occurrences of all words in a file. Taking a two line text file:

$$(1, \textit{“the cat sat on the mat”})$$
$$(2, \textit{“the man with a telescope”})$$

Each line is stored with a key corresponding to the line number and a record holding the text value. The first step is to define a *map* function:

$$(1, (W_1, W_2, ...W_n)) \longrightarrow ((W_1, 1), (W_2, 1)...(W_n, 1))$$

The map function takes a single key-value as input. The key is the line number and in this example the key is equal to 1 so it is processing the first line. The value is a normal text string which represents a collection of words from W_1 to W_n. To the right-hand side of the arrow is the output of the function. The output is a collection of key-value records with the word as the key and an integer which represents the count for that individual token in the sentence. If applied on the first line in the file.

$$(1, \textit{“the cat sat on the mat”}) \longrightarrow$$
$$(\textit{“the”}, 1), (\textit{“cat”}, 1), (\textit{“sat”}, 1), (\textit{“the”}, 1), (\textit{“on”}, 1), (\textit{“mat”}, 1)$$

The data generated from a map function is written in an intermediate file stored on disk. Once every line has been processed the intermediate file will be full of key-value records with the words as keys and their counts as values. Before the reduce step is executed we need to group together all the keys with the same value. This grouping can be done efficiently with a sort on the intermediate file of the results from the map step, using the word keys to define the sort order. The final step is to define a reduce function. If w_i appears n times across several sentences the function would be:

$$((W_i,1),(W_i,1)...(W_i,1)) \longrightarrow (W_i,n) \tag{4.2}$$

or if this MapReduce was applied to the whole of the two line file then the reduce operation for the token "the" would be:

$$(\ \textit{"the"},1),(\ \textit{"the"},1),(\textit{"the"},1) \longrightarrow (\textit{"the"},3)$$

A more formal definition of MapReduce can be given.

map $\quad (k1,v1) \quad \longrightarrow list(k2,v2)$
reduce $\quad (k2,v2) \quad \longrightarrow list(k3,v3)$

An important aspect of this definition is *typing*. The output of the map function is allowed to transform the types $k1$ and $v1$ to $k2$ and $v2$. This can be seen in the word count example where $k1$ was an integer line number and was transformed into $k2$ which is string. The important constraint is the reduce function inputs has to use the same types as the map function output.

The principle behind MapReduce is that many problems suitable for parallisation can be expressed in these two steps. The key-value records can be processed concurrently during the map step, and the aggregated records can be processed concurrently during the reduce step. The only restriction in program flow is that the sort is executed sequentially. There is a contract between the programmer and MapReduce to respect the constraints imposed by the functional style of programming. In return MapReduce will be able to handle all the concurrent execution of the program.

Let us now return to the Stupid Backoff model. To compute pseudo-probabilities $S(\cdot)$ we need N-gram counts. To change our example to an N-gram counter, the map function would emit N-grams to be aggregated by the reduce function.

Brants et al. [25] describe a method for computing Kneser-Ney language models using MapReduce. This computation is a more involved process because of the requirement to observe N-gram contexts. Instead of a single map and reduce, Kneser-Ney requires three

successive stages of maps and reduces. Because only a single map and reduce is needed for the Stupid Backoff model, Brants et al. [25] were able to double the amount of training data used. In general, MapReduce is suited for batch processing where model parameters or features are extracted from a small data window and aggregated across a large dataset.

4.2.1 MapReduce Implementation

In the previous section the abstract map and reduce functions were defined. This section describes an implementation of a framework [49] that uses these abstract functions. When discussing a MapReduce implementation we refer to the cluster it runs on. A cluster is a set of machines running and managed by MapReduce software. We also denote a node as an individual machine inside the cluster.

Instead of using a small two line file we can look at how to process a much larger file, illustrated in Figure 4.1. The framework first splits the file into smaller segments. A map operation is applied to the records in each segment to produce an intermediate file composed of key-value records. As a map operation is independent of neighbouring records, map operations can be applied in any order. A MapReduce framework will send segments to separate nodes and each node will apply the map function to each record in the segment sequentially. In Figure 4.1 the keys of the intermediate files have been colour coded into four shades of grey corresponding to four distinct key values. The results of the map function are then partitioned into different segments. The distribution of keys among the partitions is decided by taking a hash of the key. As the map functions generate keys they are placed in the partitioned blocks in the background by a process known as *fetching*. Note that this is the only time in the process that data is exchanged between segments.

Everything left of the dotted line labelled 'barrier' can be performed in parallel. As a consequence of the *fetching* process crossing segments it is sensitive to the output of all the map operations. Therefore all map operations on all segments must be completed and the intermediate records placed in the correct partition before MapReduce can continue. Once all the segments have been fetched, the barrier is crossed and parallel computation can restart. The partitioned segments are sorted to group the records together by key. In figure 4.1 the sort order is indicated by the lightness of the grey colour. As each node has its own partitioned data the sort can be done in parallel. The sort algorithm is a merge sort which allows nodes to sort partitioned data sets larger than physical memory. Once sorted it is straightforward to collate a set of keys and input them into a reduce task. These tasks can be again be run concurrently for each set of keys. Finally the reduce tasks will output their

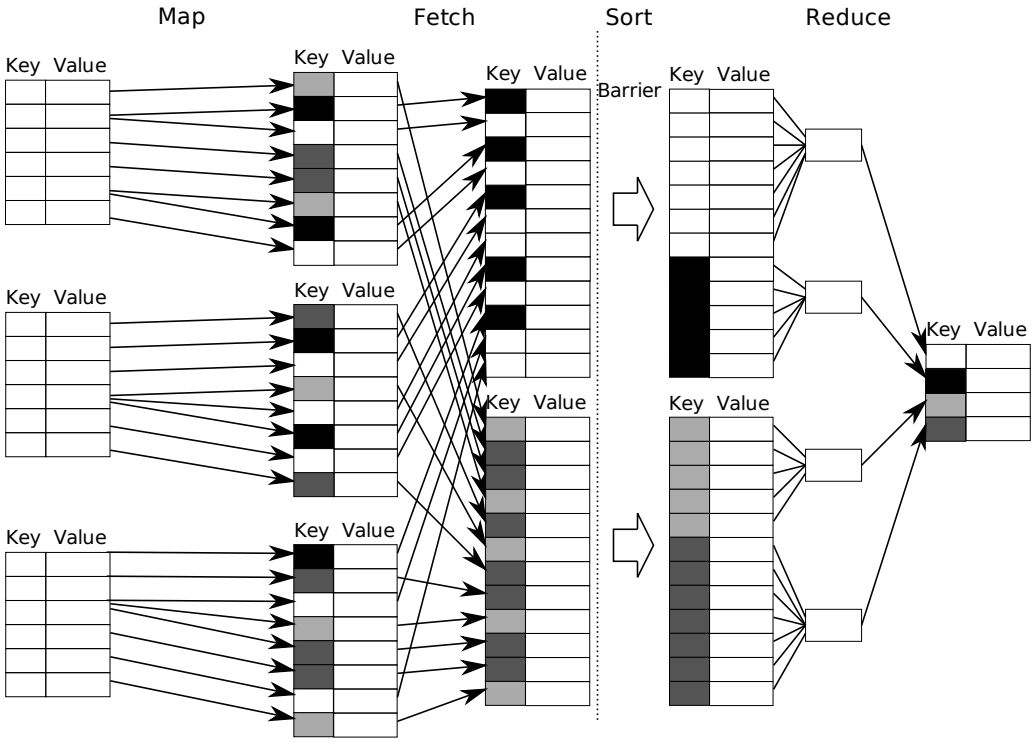


Fig. 4.1 Illustration of how MapReduce is applied to a large file. The execution moves from left to right in time. All the processing can be performed in parallel until the doted line labelled “barrier” is reached. The fetch stage is the only part of the process where data is moved between segments and hence forces the MapReduce to pause until all fetch functions have completed. After all jobs to the left of the barrier are finished parallel processing of the sort and reduce functions is allowed.

results to the final output file. The final output is not in sorted order and is sensitive to how the data is partitioned.

The MapReduce implementation is tightly coupled to a distributed filesystem [66] called the Hadoop Distributed Filesystem (HDFS). This filesystem takes the form of processes, called datanodes, running on nodes in a cluster. Each process serves data from local physical disk to a client MapReduce process. Using HDFS cuts network traffic in the cluster because the MapReduce process is aware of which datanode stores relevant data. The map and reduce functions are executed on the same machines where the data resides.

4.3 The HFile Format

MapReduce is useful for estimating parameters or extracting features of a model. In the introduction, we stated that the estimated parameters can be filtered because the decoder

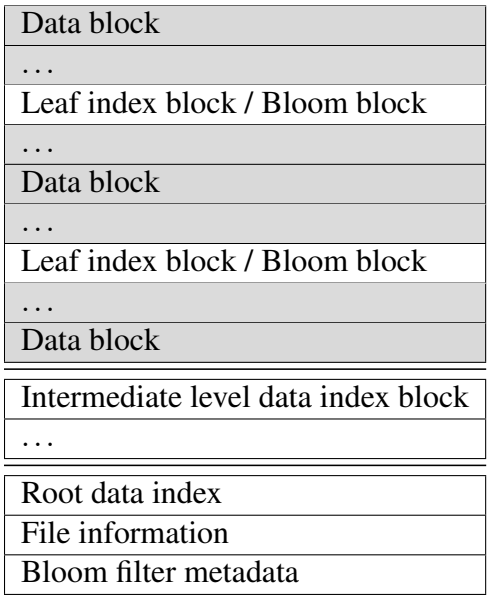


Fig. 4.2 The internal structure of an HFile, after Apache Software Foundation [5] (simplified)

only requires a small fraction of them. MapReduce is a poor choice for this task because each key-value has to be loaded from the file system, and we should try to minimise disk access to achieve fast filtering.

Before the introduction of Bigtable [31] the main choice of framework to solve this problem would have been a relational database [42], which combines a rich relational formalism with algorithms for querying data quickly. We can disregard the use of a relational database because it is too expressive for our problem. Our data can be adequately expressed as key-value pairs, and so the relational formalism is not useful. Also, Bigtable uses many of the same algorithms and techniques for fast querying. For example, the SSTable and HFile formats we describe in this section are directly inspired by the B-tree data structure found in database implementations [31]. Using a relational database would add additional complexity for no benefit.

The next logical consideration would be to use Bigtable [31] as a distributed key-value store for model parameters. HBase [5], which is the Hadoop implementation of Bigtable, has already been used for distributed language models [155] demonstrating that such an approach is feasible. The issue with using such a system is that it must be constantly maintained and available. Keeping a distributed system constantly running is beyond the resources of most researchers. We therefore propose using components of HBase, as opposed to the whole system.

The following description of the HFile format is jointly written with Pino et al. [124]. To store a model represented as key-value pairs, we use the HFile file format [5], which is a reimplementation of the SSTable file format [31]. The HFile is used at a lower level in the HBase infrastructure. In this work, we reuse the HFile format directly without having to install an HBase system. The HFile format is a lookup table with key and value columns. The entries are free to be an arbitrary string of bytes of any length. The table is sorted lexicographically by the key byte string for efficient record retrieval by key.

Internal structure

As can be seen in Figure 4.2, the data contained in an HFile is internally organised into data blocks. The block size is configurable, with a default size of 64KB. Note that HFile blocks are not to be confused with Hadoop Distributed File System (HDFS) blocks whose default size is 64MB. If an HFile is stored on HDFS, several HFile blocks will be contained in an HDFS block. A block index is constructed which maps the first key of an HFile block to the location of the block in the file. For large HFiles the block index can be very large. Therefore the block index is itself organised into blocks, which are called leaf index blocks. These leaf index blocks are interspersed with the data blocks in the HFile. In turn, the leaf index blocks are indexed by intermediate level data index blocks. The intermediate blocks are then indexed by a root data index. The root data index and optionally the Bloom filter metadata, described next, are stored at the end of the HFile. In order to distinguish block types (data block, index block, etc.), the first 8 bytes of a block will indicate the type of block being read. The HFile format allows for the blocks to be compressed. The choice of compression codec is selected when the file is created. We choose the GZip compression codec for all our experiments. Block compression is also used in other related software [121].

Record retrieval

When the HFile is opened for reading, the root data index is loaded into memory. To retrieve a value from the HFile given a key, the appropriate intermediate index block is located by a binary search through the root data index. Binary searches are conducted on the intermediate and leaf index blocks to identify the data block that contains the key. The data block is then loaded off the disk into memory and the key-value record is retrieved by scanning the data block sequentially.

Bloom filter optimization

It is possible to query for a key that is not contained in the HFile. This very frequently happens in translation because of language data sparsity. Querying the existence of a key is expensive as three blocks have to be loaded from disk and binary searched before it can be decided that a query fails. For fast existence check queries, the HFile format allows the inclusion of an optional Bloom filter [22]. A Bloom filter provides a probabilistic, memory efficient representation of the key set with an $O(1)$ membership test operation. The Bloom filter may provide a false positive, but never a false negative for existence of a key in the HFile. For a large HFile, the Bloom filter may also be very large. Therefore the Bloom filter is also organised into blocks called Bloom blocks. Each block contains a smaller Bloom filter that covers a range of keys in the HFile. Similar to the root data index, a Bloom filter metadata or Bloom index is constructed. To check for the existence of a key, a binary search is conducted on the Bloom index, the relevant Bloom block is loaded, and the membership test performed. Contrary to work on Bloom filter language model [138, 139], this filter only tests the existence of a key and does not return any statistics from the value. If a membership test is positive, the HFile data structure still requires to do a usual search. During the execution of a query, two keys may reference the same Bloom blocks. To prevent these blocks from being repeatedly loaded from disk, they are cached after reading.

Local disk optimization

The HFile format is designed to be used with HDFS. Large files are split into HDFS blocks that are stored on many nodes in a cluster. However, the HFile format can also be used completely independently of HDFS. If its size is smaller than disk space, the entire HFile can be stored on the local disk of one machine and accessed through the machine's local file system.

Query sorting optimization

Prior to HFile lookup, we sort keys in the query set lexicographically. If two keys in the set of queries are contained in the same block, then the block is only loaded once. In addition, the computer hardware and operating system allow further automatic improvements to the query execution. Examples of these automatic improvements include reduced disk seek time, the operating system caching data from disk [1], or CPU caching data from main memory [120].

[1] The Linux Documentation Project, The File System, http://tldp.org

4.3.1 Application to Stupid Backoff Models

To demonstrate that HFiles are fit to store model parameters, we build a Stupid Backoff model and measure retrieval times and memory usage. The HFile stores N-grams W_i^j as keys, and their counts $f(W_i^j)$ as values. Each word of the key is mapped to an integer so that the N-gram becomes a string of integers. Each integer is then converted into a binary representation with a three byte width, which is adequate for the vocabulary used by our collections. The count is stored using a four byte integer representation.

For a baseline, we use a suffix array based on the Suffix Array Language Model toolkit (SALM) toolkit [161]. The original SALM toolkit used a 32-bit integer representation for each element in the suffix array. This representation has been widened to 64-bits to allow a larger corpus to be indexed. SALM loads the suffix array and monolingual corpus into memory for fast computation of the counts. We did not report comparisons to the KenLM toolkit [74], which is designed for retrieving N-gram probabilities from an ARPA file as opposed to raw N-gram counts.

We note that we compare the HFile against frameworks designed specifically for SMT, Speech Recognition, and other tasks specific to Natural Language Processing. We are not comparing the HFile against other 'off the shelf' solutions due to the arguments made in Section 4.3. It is more than possible that a HBase cluster would be faster, but at the cost of more spending more time maintaining and managing the cluster.

We use the following setup:

Data We use a concatenation of the Gigaword Fifth Edition [119] with the English side of the NIST'12 parallel data for the constrained track. The SALM toolkit imposes a 256 word limit on sentence length in the corpus, therefore we truncated all sentences to 256 words. The corpus contains 5.4 billion words. From the monolingual corpus we extract 2.5 billion word sequences and counts. These are stored in an HFile with 8 KB data block size.

Translation lattices we replicate an experiment where a set of 2816 translation lattices are rescored using a 5-gram Stupid Backoff language model [21]. The N-gram keys required to build the set-specific language model are extracted from the lattices using modified counting transducers [109]. The queries take the form of 8.4 million keys, of which 7.3 million of the keys are unique.

HFile optimization we execute four HFile based queries based on whether the HFile contains a Bloom filter index, and whether the HFile is stored on local disk or a distributed

	Index Load	Query Processing	Query Retrieval	Total Time	Peak Memory
Suffix Array	8m39s	-	3m20s	11m59s	90.7G
HFile, HDFS	-	18s	3m54s	4m12s	3.1G
HFile, Bloom, HDFS	-	1m11s	2m52s	4m3s	5.8G
HFile, Local	-	18s	3m5s	3m23s	3.1G
HFile, Bloom, Local	-	25s	1m56s	2m21s	5.8G

Table 4.1 Comparison of times and memory usage for N-gram count queries using various HFile implementations and the suffix array baseline. Rows labelled HDFS or Local indicate whether the HFile is stored in a distributed filesystem or local disk. The rows labelled with Bloom indicated that the query used a Bloom filter.

file system.

Time measurement phases we split the query execution into distinct phases. For SALM we record the time taken to load the suffix array and monolingual corpus into memory, which we label index load time. We then enumerate through the unsorted keys in the query and compute the count associated with the key. Note that for any duplicate key in the query a duplicate count is computed. We call this phase query retrieval. For the HFile based infrastructure, the query has to be sorted. A Bloom filter may also be applied after the sort. We call this phase query processing. We then look up the HFile to locate the query keys. The look up phase is also labelled query retrieval.

Hardware configuration the machine used for the query has 94GB of memory and an Intel Xeon X5650 CPU. The distributed file system is hosted on the querying machine and other machines with the same specification, which are used to generate the HFile.

File system caching Modern linux based operating systems buffer the results of local disk access in RAM [142]. To achieve consistent and repeatable results, the query is executed twice for both SALM and the HFile based system. The time for the second query execution is recorded.

During these experiments the only processes running on the machine are the HFile query processes and background processes required by the operating system. The results are shown in Table 4.1 from which we can draw the following observations:

Speed column 4 shows that the HFile infrastructure provides a competitive query speed with respect to SALM.

Memory column 5 shows that the memory overhead of the HFile infrastructure is much lower than SALM. We could reduce the suffix array memory usage by doing an on-disk binary search but this would increase the query processing time.

HFile optimizations an interesting result is the effect that the Bloom filter has on the query processing time for the distributed query. The time spent loading the blocks that comprise the Bloom filter offsets the time saved retrieving the counts. However, when using local disk the Bloom filter has only a small impact on the query processing time.

In addition, although disk usage is not an issue, it is worth mentioning that the English monolingual data together with the suffix array represent 90G of uncompressed data and the HFile size is 11G without Bloom filter and 14G with Bloom filter. We store the English monolingual data in a decompressed file for more efficient loading into a suffix array. On the other hand, only HFile blocks potentially containing a key are uncompressed during an HFile query.

4.4 Improvements to Hierarchical Rule Extraction

In this Section we move onto a more complicated problem than the Stupid Backoff language model. We describe a system [122, 124] for computing the hierarchical rules in Section 2.2.3. We then discuss a series of "tricks" to improve the runtime and memory performance of the system.

In Section 4.1 we noted that several translation rule extraction systems based on MapReduce already exist. However, taking the general principles of MapReduce and building a fast and memory efficient implementation is not a trivial operation. We are not aware of a detailed implementation guide for building parameter estimation processing systems using MapReduce.

Let us assume that we are given parallel text with word alignment links, similar to the example in Figure 2.1. Rules are extracted following the scheme described in Section 2.2.3. These rules take the form a pair, source $\mathbf{u}$ and target $\mathbf{v}$ strings of terminals and non-terminals. For each rule $(\mathbf{u}, \mathbf{v})$ a set of features is computed for the rule.

Most features, such as the number of non-terminals in-each rule, can be computed directly from the strings. Two sets of features require a more complex procedure. The first set is the rule conditional probability as computed by relatively frequency counts

$$\phi(\mathbf{u} \mid \mathbf{v}) = \frac{f(\mathbf{u}, \mathbf{v})}{\sum_{u'} f(\mathbf{u}', \mathbf{v})} \tag{4.3}$$

	⋯	white	⋯	house	⋯
⋮					
maison		$P_{M1}(\text{white} \mid \text{maison})$		$P_{M1}(\text{house} \mid \text{maison})$	
⋮					
blanche		$P_{M1}(\text{white} \mid \text{blanche})$		$P_{M1}(\text{house} \mid \text{blanche})$	
⋮					

Table 4.2 An extract of a translation table with English as the target and French as the source

where $f(\mathbf{s},\mathbf{t})$ is the count of rules extracted from parallel data. These features are computed in both directions, $\mathbf{s} \mid \mathbf{t}$ and $\mathbf{t} \mid \mathbf{s}$.

The second set of features are called lexical features and are based on the IBM word alignment model 1 [26]. Let us denote $\tilde{\mathbf{u}}$ as a subsequence of the terminals in $\mathbf{u}$ and similarly $\tilde{\mathbf{v}}$ as a subsequence of the terminals in $\mathbf{v}$. The lexical features are computed using the following method [89]:

$$s(\tilde{\mathbf{u}},\tilde{\mathbf{v}}) = \frac{1}{(V+1)^U} \prod_{u=1}^{U} \sum_{v=0}^{V} P_{M1}(\tilde{u}_u \mid \tilde{v}_v) \tag{4.4}$$

where U and V are the respective number of terminals, and $P_{M1}(\tilde{u}_u \mid \tilde{v}_v)$ is the model 1 probability.

The probabilities $P_{M1}(\tilde{u}_u \mid \tilde{v}_v)$ are stored in large translation table containing the probability of a terminal in $\mathbf{u}$ given a terminal in $\mathbf{v}$. The rows of this table are indexed by words on the source side, and the columns by the target side. An example of a translation table with the probabilities between two words is shown in Table 4.2. The translation table is very large because it contains a mapping between every word in the source vocabulary to the target vocabulary, although many entries will take a zero or a default value. Because word alignment models are not symmetrical using two translation tables to compute $\mathbf{u} \mid \mathbf{v}$ and $\mathbf{v} \mid \mathbf{u}$ versions of this feature gives additional performance.

Both sets of features are estimated over the entire set of parallel data, and subsets based on the provenance of the parallel data [38]. For example, some of the parallel data may be from a high quality source such as translated EU parliamentary proceedings, and some may be scraped from the world-wide-web using some automated tool. We can extract features based on each provenance. The parallel data is also combined to create a global provenance. This leads to accessing a fairly large (approximately a dozen) collection of translation tables.

4.4.1 Baseline Rule Extraction System Design

One methodology for designing software design is the iterative development cycle [14]. In this methodology an initial system is built quickly, and defects are fixed over time. The proponents of this methodology argue that many defects only become apparent once the system is running, and that software development is better modelled as a continuous iterative process that fixes these defects and adds small improvements. This is in marked contrast to the 'big bang' or 'waterfall' development model where the system is delivered in full, given a rigid set of specifications.

In this section we describe the describe the system of Pino [122] for extracting rules and computing their features. The system uses MapReduce to extract rules and compute feature values, and then stores the results in an HFile for fast retrieval. We consider this to be our initial system. In the next section we describe a second iteration of the iteration of the system, incorporating many of the lessons learnt from this initial system. The system proceeds over the following five stages:

Rule Extraction

In this stage, a MapReduce job extracts rules from the parallel data. The map function extracts the rules using rule alignments as described in Section 2.2.3.

The decision was made not to aggregate the rule counts after extraction. This can be achieved by the use of identity reducer function, which does not transform the input. The resulting key-value records take the form:

$$((\mathbf{u}, \mathbf{v}), 1)$$

As a result of this decision, all later stages are required to aggregate counts separately before processing can begin.

Feature Computation

Once the rules have been extracted, a series of MapReduce jobs is executed that compute the conditional rule probabilities and lexical features. The input into the jobs is the output of the previous rule extraction job. Each feature is computed with a single MapReduce job, such that two jobs are needed for each provenance.

For the rule conditional probability the MapReduce job is based on method 3 in Dyer et al. [55]. First the pair of source and target strings is transformed using a map function

such that the key is the target side of the rule and the source side string is moved into the value.

$$((\mathbf{u},\mathbf{v}),1) \longrightarrow (\mathbf{v},(\mathbf{u},1))$$

The MapReduce framework then aggregates all the rules with the same target side. For the reduce function, the input key is the target side string and value is a list of the source side strings that share the same target side. From this list the occurrence counts can be computed.

The output of the reducer is a key-value record where the key is the target side strings, and the value is a list of source side strings and their probabilities. If there are n rules that share the same target side $\mathbf{v}$ then the output is

$$(\mathbf{v},((\mathbf{u}_1,P(\mathbf{v} \mid \mathbf{u}_1)),\ldots,(\mathbf{u},P(\mathbf{v} \mid \mathbf{u}_n))))$$

For computing $P(\mathbf{u} \mid \mathbf{v})$ a similar job is executed, but the with the $\mathbf{u},\mathbf{v}$ strings swapped.

For the lexical features the map function does nothing, modelled by an identity map function. The MapReduce framework aggregates the rules such that there is only a unique key-value record for each rule. The reducer loads then loads the appropriate translation table into memory. For each rule the lexical feature is then computed via Eqn. (4.4). Note if many reducers are running in parallel on the same node then the translation table will be loaded multiple times.

Merging

The feature computation stage requires a separate MapReduce job for each feature, direction, and provenance. So for example, if we compute rule probabilities and lexical features from two provenances then we will have 12 MapReduce outputs from the previous stage:

- two sets of features in the source-to-target direction for the global provence
- two sets of features in the target-to-source direction for the global provence
- two sets of features in the source-to-target direction for the first provence
- two sets of features in the target-to-source direction for the first provence
- two sets of features in the source-to-target direction for the second provence
- two sets of features in the target-to-source direction for the second provence

On disk these results are stored as 12 separate files of key-value pairs. The format of these files is also varied because of the transformation required by the rule probability computation. For example source-to-target features have the target string as the key, and the target-to-source use the source string as the key. These job outputs have to be merged into a consistent set of records, such that for a single rule key, all the features are stored in a single value.

The map function transforms the results of the feature computation jobs into a list of key-value records where the key is the source side string $\mathbf{u}$ and value a list of triples of target side string $\mathbf{v}$, feature vector index d, and feature value $h_d(\mathbf{u},\mathbf{v})$.

The MapReduce framework aggregates the results such that all the rules with the same source side strings are passed into the reduce function. The reducer then loads all feature values into memory, and does a secondary sort of the triples in memory by target side $\mathbf{v}$ such that the records with the same target side strings are contiguous. Note that this second sort is done outside of the MapReduce framework and must occur in the memory of the process executing the reduce function. The reduce function can then create sparse representations of the feature vectors $\mathbf{h}(\mathbf{u},\mathbf{v})$. The final output is a key-value record where the source string $\mathbf{u}$ is the key and the value is a list of pairs of targets strings and sparse feature vectors. If there are m rules that share the same source side string then the final output is

$$(\mathbf{u},((\mathbf{v}_1,\mathbf{h}(\mathbf{u},\mathbf{v}_1)),\ldots,(\mathbf{v}_m,\mathbf{h}(\mathbf{u},\mathbf{v}_m))))$$

HFile Generation

Once the merge has been completed the Hfile is created. The Hfile is essentially the output of the merge job converted to the Hfile format. The output of the merge must be sorted with respect to the source side string keys so that the Hfile can be generated. Ideally the MapReduce framework should be used to sort the records but the framework partitions keys with respect to the hash of the key. Because of this partitioning the keys are not guaranteed to be in sorted order across all partitions if multiple reducers are used. One solution is to tell the framework how to partition keys using a TotalOrderPartitioner, which requires a sample of the key distribution before the job is run. This distribution can be difficult to obtain because it requires a previous output of the extraction job.

Rule Retrieval

To retrieve the subset of rules necessary to translate a sentence, all the possible source-sides string are generated from the input sentences. These query strings are sorted to form a query. The HFile is then queried to find matching source side strings, and the list of target side strings and associated feature vectors.

The query is stored as a large associative array in memory, with rules as keys as feature vectors as values. Once the query is complete the array is dumped to disk to become the set-specific rule file.

4.4.2 Improvements to Rule Extraction

We now describe a set of improvements to the system of Pino [122] that result in improvements in runtime and memory consumption. The fundamental design principle of our approach is to exploit the different types of data storage. There are roughly three different types of data storage [120]: disk, RAM, and CPU cache. The disk has largest capacity but with the slowest access, RAM has smaller capacity than disk but has faster access, and the CPU cache has smallest capacity but with fastest access. The differences between these types of storage vary by orders of magnitude. Therefore to improve runtimes we must minimise requests to disk, and if possible also minimise requests to RAM versus CPU cache access. Similar design constraints are explored with respect to language modelling in Heafield et al. [75].

Another design principle is the reduction of complexity. There are many jobs and data transformations in the system that can be refined and simplified. Simplifying the design allows us to identify bottlenecks and remove unnecessary computation. We first identify improvements in each of the five stages of the pipeline, and then describe low-level changes that squeeze maximum performance from the design.

Rule Extraction

The main issue to fix with rule extraction is that the rule counts are not aggregated in the reduce function. This forces the MapReduce framework to repeatedly perform aggregation during the feature computation. Aggregating the rule counts in this stage means that it is slower, but time is saved in later stages.

Feature Computation

In Section 4.2 we described MapReduce in the context of the Stupid Backoff language model. For such a simple model the programming model offered by MapReduce is a good fit, yielding a concise and performant system. For the more complicated hierarchical phrase-based model, we can see problems trying to compute the model using MapReduce.

One problem that MapReduce struggles to model is the computation of lexical features. This feature requires a large static set of data to be loaded from disk. This static set of data is loaded by each reducer causing a large RAM requirement for the system. We believe that such a feature is unsuitable for MapReduce style programming model. Instead, we push this computation to the rule retrieval stage. Because the key-value records no longer contain lexical features the MapReduce computation is faster because less data is being processed. This one change gives us a large reduction in peak RAM consumption, because we no longer have to load the translation table into memory multiple times.

The computation of rule conditional probabilities is a better fit for the MapReduce programming model, but there are still problems. The representation of the rules, a pair of strings $(\mathbf{u}, \mathbf{v})$, has to be repeatedly transformed by the mapper such that either the source or target string is used as the key for sorting and aggregation. These transformations cause an increase in disk activity and added complexity to the system.

The Hadoop implementation of MapReduce recognises that the programming model can be restrictive. It allows the user to take control of how the key is treated to allow for refinements in data processing. To simplify computing conditional rule probabilities we treat the rule differently depending on the context. Consider the computation of the probability $P(\mathbf{v} \mid \mathbf{u})$ and assume that we have extracted a set of rules where the key is of the form $(\mathbf{u}, \mathbf{v})$.

During the fetching phase we hash only on target side string $\mathbf{v}$, which ensures all rules with the same target are in the same the same partition. During sorting we perform a lexicographic sort considering the $\mathbf{v}$ part of the key first, and then the $\mathbf{u}$ part. Because the rules are now arranged in contiguous blocks with respect to the source string the reducer implementation can stream through the results and compute the conditional probabilities. Throughout the whole job we can keep the key intact.

The probability $P(\mathbf{u} \mid \mathbf{v})$ can then be computed on the same data by hashing on the $\mathbf{u}$ part of the key and reversing the lexicographic sort. This improvement is intended to simplify the system, but keeping a single representation of the key will be useful when considering low-level tricks to improve the runtime of the feature computation.

There are other improvements we can make to system that are unrelated to the program-

ming model. We can compute all provenances in one pass, meaning that we only need two jobs to compute all features for each $\mathbf{u} \mid \mathbf{v}$ and $\mathbf{v} \mid \mathbf{u}$ directions. A separate job for each feature is expensive because we have to read the entire set of rules from disk for each feature computation.

Merging and HFile Generation

The merge job is now much simpler because all the data is stored with a source-target (u, v) pair as the key, as opposed to some records using target-source (v, u). We can also use a similar hashing trick for computing conditional probabilities, so that all keys with the same source are sent to the same partition. We also reuse the lexicographic sort, meaning that records with the same rule keys are contiguous when entering the reducer. The reducer now no longer needs to do the custom sort, which speeds up merging.

Another bottleneck is the HFile generation job. Using a TotalOrderPartitioner requires a prior distribution of keys and usually the partitions are not evenly distributed. Instead we use the default hashing partitioner. We also combine the merge and generation jobs into one, so that the output of a merge job is a HFile. This results in many HFiles with keys distributed by their hash function.

Rule Retrieval

The HFile generation now produces many HFiles with each HFile containing keys that have the same hash. We therefore adjust the rule retrieval step to partition the query by hash function and query the respective HFiles. We also take advantage of having separate HFiles by querying each of them in a separate thread, allowing us to maximise the disk throughput.

As stated before the lexical features are now computed in this stage. We use a client-server model to compute these features. A server process loads the translation table into RAM, and a client process requests the parameters. The translation table is loaded into memory of a single machine, and can be shared across a network among clients running on other machines. The feature is computed during rule retrieval, after the rules are retrieved from the HFile. Computing features in this stage does slow down retrieval, but this is compensated by faster HFile querying due to the smaller size of the HFile.

We also do not store the results in an associative array before dumping the results. Instead we use a streaming model, where a rule and its features are written as soon as they are retrieved. This change cuts down the memory consumption of the querying tool.

Low-level Changes

We now describe a set of low-level improvements that do not change the overall design of the system.

Combiner A combiner is an a function that aggregate the results of a map before a fetch. For example, we can sum all rule counts that have been seen in the same mapper process. The output of the mapper will be smaller and results in less network traffic and disk access. Combiners are recommended by Dyer et al. [55] for the computation of phrase-based models and we use them in the extraction stage.

Byte String Comparator Hadoop is implemented in the object-orientated language Java. Keys are represented as objects, with a protocol to convert them to byte Strings representations for storage. During the sort stage of Hadoop, chunks of intermediate data are loaded from disk and the keys in these chunks are compared. To compare two keys, two byte strings representations must be converted to objects. Such a process is expensive because the Java runtime has to allocate RAM, create the objects, and then invoke the garbage collector at a later time to clear up the objects. Although these operations are fast with respect to disk access, we will be performing them many, many times. It also means that the more data has to transferred between RAM and the CPU cache.

We stated in the feature computation modification that the rule representation used as a key is never transformed. Because this consistent representation is used for all feature computation and merging, we can focus on improving the speed for comparison operations on this representation and get performance improvements across the whole system.

Hadoop allows the user to define a byte-level comparator. This is a function that takes two byte strings and informs the framework of their relative order. Because the rule representation is a pairs of strings, a byte string comparator can be used to compare two keys. The framework loads bytes strings into buffers, which are likely to be hosted in the CPU cache, and the byte string comparator works directly from the buffers.

Custom Collections Hadoop supplies a library of collection classes, such as sets and lists, that can easily be converted into byte strings. These collections have a problem due to a limitation of the Java language. In Java, the type of objects held in a collection is erased by the compiler. At run time, Java does not know whether a collection contains

a collection of strings, integers, floats, etc. Therefore to convert a list of objects to a byte string, the Hadoop libraries must prefix the type in front of the value. For example, for a list of n 4-byte integers, Hadoop will prefix "integer" n times before each integer. This dramatically increases the amount of data being stored, processed, passed around the network, and stored in HFiles.

Hadoop provides an object-orientated mechanism to avoid such prefixing by creating custom collections. The user inherits these collection classes and overrides methods describing the types of the collection. Using these collections allows us to reduce the size of records. This should result in smaller HFile sizes, and faster processing because less data has to be loaded from disk.

4.4.3 Impact of Improvements

We now perform an experiment to measure the impact of these changes. We use the baseline system of Pino [122] and compare it to a system incorporating all the improvements. For the basis of the experiment we use CUED Russian->English [125] entry to Eighth Workshop on Machine Translation [24]. As noted in Section 2.4, this system was amongst the top-scoring systems. The grammar computed is for the CoreNLP segmentation. The output of the two systems is identical.

We run the systems sequentially on a single node cluster. The node is a machine containing 24 Intel Xeon cores with a clock speed of 2.67GHZ and has 96 gigabytes of RAM. The machine is dedicated to experiments with no other processes running. We measure the system in two phases. The first phase is a MapReduce phase which includes rule extraction, feature computation, merging and HFile generation jobs. During the running of these jobs we measure the peak resident memory of all Hadoop process. Memory is sampled by using the Linux ps command executed every 15 minutes.

Because of the large-memory consumption of the baseline feature computation jobs, due to loading of translation tables for lexical features, we are unable to run them concurrently on this machine. We have to limit the maximum number of reducers that can run on the machine to 7 so that the translation tables are only loaded a maximum of 7 times. To get a fairer comparison we use a second configuration of baseline system that does not compute lexical features. This second system is allowed to use more than 7 reducers. The improved system also does not computed lexical features, because they are computed in the second phase. The results are in Table 4.3.

The second phase is the retrieval phase which measures the time to retrieve all the rules

	Baseline (Lex)		Baseline (No Lex)		Improved (No Lex)	
	Time	**Mem (GB)**	**Time**	**Mem (GB)**	**Time**	**Mem (GB)**
Extraction	53m	4.1	54m	4.7	1h 38m	20.2
Feature Computation	17h 48m	81.7	4h 15m	9.6	59m	42.2
Merging	28h 59m	72.5	5h 41m	91.8	57m	32.2
HFile Generation	5h 8m	2.3	2h 33m	2.0	-	-

Table 4.3 Comparison of the runtime and peak memory usage of the baseline and improved systems for MapReduce based tasks. The lex and no lex qualifiers indicate whether systems computed lexical features.

	Baseline (Lex)		Baseline (No Lex)		Improved (Lex)	
	Time	**Mem (GB)**	**Time**	**Mem (GB)**	**Time**	**Mem (GB)**
Lexical Server Startup	-	-	-	-	7m 57s	60.3
HFile Retrieval	17m 22s	37.9	12m	36	5m 47s	15.3

Table 4.4 Comparison of the runtime and peak memory usage of the baseline and improved systems for retrieval tasks. The lex and no lex qualifiers indicate whether systems computed or retrieved lexical features.

needed for the newstest2012.test test set and the peak memory usage of the retrieval process. Memory is sampled by executing the linux ps command every 30 seconds. For the retrieval results we execute the query twice and record the second measurement to account for file system caching. The results are in Table 4.4. We also report the size of the resulting HFiles in 4.5.

From the results we can see that computing the lexical features with MapReduce is not a good fit. The memory requirements are very high, and the impact on runtimes is also high. Even with the lexical features removed, the improved system still shows large improvements in runtime and memory consumption. We also note that the memory consumption of the merge job in the baseline system without lexical features is very large. This is due to the secondary sort performed in the reducer. This is strong evidence that a lexicographic sort in the MapReduce framework is a better approach.

In terms of retrieval time, we can see that it is faster to wait for the lexical servers to start

	Size (GB)
Baseline (Lex)	51
Baseline (No Lex)	11
Improved (No Lex)	11

Table 4.5 A comparison of HFile size between the baseline system with lexical features, and the improved and baseline systems without lexical features

up and compute the lexical features on the fly than to store the feature values in the HFile. We note that the lexical servers only needed to be started once and can be left running in the background. Many rule retrieval tasks can be executed using the same lexical server processes.

In terms of retrieval memory consumption, the streaming approach taken by the improved retriever cuts the memory consumption by half. Admittedly, the lexical servers do consume a lot of memory. We note that the 60GB consumption is occupied by two 30GB lexical servers, each holding source-to-target and target-to-source versions of the translation tables. These retrieval process could be distributed among three machines with 32GB of RAM each.

4.5 Conclusion

This chapter we formulated the parameter estimation and filtering problems. We described two software frameworks, MapReduce and Bigtable, than can be used to address these problems for SMT. We then moved onto the detailed description of a real system for rule extraction and how to employ these frameworks. Our results show that with care and consideration, these frameworks can be the foundation of a system that efficiently extracts rules from large amounts of parallel data and filters them quickly for decoding.

In an interesting counterpoint to chapters 5, 6, and 7 where we strongly argue for formalism, we believe that our results show that the MapReduce programming model is not suitable for certain tasks in SMT. It seems that the programming model is not rich enough to represent the many disparate operations involved in parameter estimation. We are faced with the dilemma of trying to'shoehorn' the problem into the formalism or disregarding the formalism altogether. We believe that most of the mistakes made in the baseline system design described in Section 4.4.1 were due to forcing the problem to fit into the programming model. Our improved design took the opposite route of adapting the programming model to

fit the problem.

The inflexibility of MapReduce seems to be understood by the makers of the Hadoop framework who are extending it to use a more general programming model [4]. One potential framework of interest that has been developed by distributed systems researchers is the Spark framework [156], which also extends Hadoop. This framework has a programming model that augments the map and reduce functions with a set of new functions. It also has facilities for distributed caching data in memory of many machines. As an example where the Spark framework could be used where MapReduce would struggle is word alignment, where previous iterations of Expectation-Maximisation need to be accessed quickly.

The issues with the MapReduce programming model should not tarnish the many other good ideas found in the framework. These include, but are not limited to, the tight integration of the compute framework with the distributed filesystem, the importance of fast aggregation and sorting, and functional programming influenced designs. These ideas are at the heart of the successor frameworks such as Spark.

Chapter 5

A Description of Minimum Error Rate Training Using Convex Polytopes

5.1 Introduction

In this chapter we continue the study of Minimum Error Rate Training (MERT) from Chapter 3. We discuss both the line optimisation procedure of Och [113], and the linear programming form introduced by Galley and Quirk [62]. We note that both these approaches exploit the geometric properties of convex sets. Interestingly, there exists a branch of mathematics called convex geometry where the geometry of convex sets is studied in detail.

Applying geometric methods for optimisation has precedent. The linear models we discuss in this section are closely related to binary linear classifiers, such as the perceptron Bishop et al. [20, Page 192] or the support vector machine (SVM) Bishop et al. [20, Page 325]. The perceptron is often optimised using a geometric method [17]. Typically the SVM is transformed into a Lagrangian dual form for optimisation. However, the primary form of the SVM is often given in an intuitive geometric form [18], and in some circumstances it is preferable to solve the primary problem [18]. These classifiers are applied to a binary class problem with reasonably large amounts of labelled training data for both classes. The models we study in this chapter differ in that we have many many classes with typically only a single sample per class. Also recall that linear programming can be described geometrically, as noted in Section 3.2.3.

This chapter provides a formal description of MERT for a single sentence using convex geometry. The more complete case is considered later in Chapter 6. Because we only consider the single sentence case, we simplify our notation to ignore the source sentence index. Given the source language sentence $\mathbf{f}$, let $\mathbf{h}_i = \mathbf{h}(\mathbf{e}_i, \mathbf{f})$ be the feature vector associated

with hypothesis $\mathbf{e}_i$.

We describe two mathematical constructions in great detail to develop an understanding of MERT. The first is a form of the convex hull called the convex polytope [162]. Convex hulls are typically used for MERT [62, 102, 113] but the convex polytope provides a much richer and more formal description than has appeared before in the field of statistical machine translation. The second construction is the dual to the convex polytope: the normal fan [162]. The normal fan is a geometric object in parameter space constructed from a convex polytope in feature space.

Recall that the linear model of Statistical Machine Translation (SMT) introduced by Och and Ney [114] casts translation as a search for translation hypotheses under a linear combination of weighted features: the source language sentence $\mathbf{f}$ is translated as

$$\hat{\mathbf{e}}(\mathbf{f};\mathbf{w}) = \underset{\mathbf{e}}{\operatorname{argmax}}\{\mathbf{w}\mathbf{h}(\mathbf{e},\mathbf{f})\} \tag{5.1}$$

Translation scores are a linear combination of features $\mathbf{h}(\mathbf{e},\mathbf{f})$ under a set of model parameters $\mathbf{w}$. The key characteristic of MERT is that model parameters $\mathbf{w}$ are optimised such that the hypothesis with the greatest model score should have the lowest error as judged by an error metric.

In Section 3.2 we saw two geometric approaches to MERT. The first by Och [113] defines an error function defined over K-best lists of translations, the optimisation criterion becomes a piece-wise constant function over the model parameters. MERT proceeds as a series of line optimisations that efficiently detect boundary points along the line at which the error function is discontinuous. These boundary points define line segments in parameter space over which the error function is constant, so that parameters can be chosen from these segments so as to minimise the overall error objective. We refer to this operation as line optimisation.

The second approach was LP-MERT [62], an exact search algorithm that reaches the global optimum of the training criterion using linear programming. The algorithm first identifies the *extreme points* within the convex hull of feature vectors associated with an K-best list of hypotheses. Specfically, the hypotheses $\mathbf{e}_i$ can be selected by the decoder only if the following K inequalities are satisfied

$$\mathbf{w}(\mathbf{h}_j - \mathbf{h}_i) \leq 0 \;\text{ for }\; 1 \leq j \leq K \tag{5.2}$$

for some parameter vector $\mathbf{w}$ ($\mathbf{w} \neq \mathbf{0}$). Any feature vector $\mathbf{h}_i$ that satisfies these constraints is denoted an extreme feature vector. Other feature vectors are *redundant* and not part of the

convex hull.

This system of inequalities defines a convex region in parameter space, and for each parameter value in the region the decoder will select the associated hypothesis $\mathbf{e}_i$ from the K-best list. The system of inequalities can be solved using a linear program and if the set is feasible then the linear program yields a parameter vector from the region. LP-MERT requires K linear programs to yield a set of feasible parameters.

Recall that for a single input sentence, LP-MERT has a polynomial run time with respect to the dimensionality of the feature vectors. However LP-MERT faces more serious computational challenges in combining the results over many K-best lists. For a training set of input sentences the number of systems of equations increases exponentially with respect to the number of input sentences. In Chapter 6 we describe how only a subset of these systems are feasible, and that an algorithm exists to enumerate these feasible systems in polynomial time [59]. Many of the key concepts used in the algorithm are first described in this chapter, for the simpler case of the individual sentence.

We first provide geometric descriptions of MERT with respect to two geometric objects: the convex polytope and the normal fan. We then describe how these objects have a dual representation, which enables us to describe a number of useful algorithms. In the fourth section we describe how Och's line optimisation can be generalised to many dimensions using a projected convex polytope. We then discuss recent work on the regularisation of MERT [63] and analyse the regularisation scheme using the normal fan. Finally we decribe a geometric representation of ranking methods.

5.2 The Normal Fan

Most methods of training an SMT system involve finding some single, optimal parameter $\mathbf{w}^*$ that optimises a constrained function over the training data. One good example of this form of optimisation is the MIRA algorithm described in Section 3.3.2.

MERT is unusual in that any parameter that satisfies a set of constraints is acceptable. Instead of a single optimal parameter, there is usually a set of possible parameters to choose from that could maximise the model score for a hypothesis. The goal of MERT is to find the parameter set that maximises the model score of a set of oracle hypotheses, and from this set an acceptable parameter is drawn.

LP-MERT [62] casts MERT as the computation of the extreme feature vectors of a convex hull. A point is labelled extreme if a single parameter $\mathbf{w}$ from a set of parameters can be identified which maximises the associated feature vector. The aim of this section

is to describe a method of identifying the whole parameter set associated with an extreme feature vector, instead of a single parameter.

The normal fan divides the entire set of possible parameters into subsets which maximises one or many feature vectors. The description of the normal fan involves an amount of new terminology. We endeavour to keep the presentation tight and as relevant as possible to the task of statistical machine translation optimisation. We believe that the effort expended on learning this new formalism is worthwhile because it allows us refer to many theorems and algorithms that are new to SMT. Examples of these include the fast polynomial time algorithm for multi-sentence MERT, projected parameter spaces to allow for MERT optimisations in any number of dimensions, and upper bound theorems on the number of feasible feature vectors.

We move quickly through the material. Definitions are highlighted in bold. In between the definitions is explanatory text and theorems. Any mathematical results of note are from Ziegler [162]. Any result which we do not attribute was deemed too trivial to be presented by Ziegler [162], but we include to make the presentation more accessible.

This section is split into five subsections. The first is a primer where basic concepts are introduced. In the second subsection the concept of an extreme point is extended into the idea of a face of a polytope. In the third subsection we discuss the relationship between faces and parameter sets, which leads to description of the normal cone. The fourth subsection describes how normal cones fit together to form the normal fan. Finally, in the fifth subsection we present an example of a normal fan.

5.2.1 Convex Geometry Basics

In this subsection we present introductory material that will be useful for later sections. Although some of this material may be familiar to many readers, the concepts in this section lay a firm foundation for the later sections and are worth reviewing.

Vector Space The real valued vector space $\mathbb{R}^D$ represents the space of all D finite dimensional feature vectors. Each element of $\mathbb{R}^D$ represents a $D \times 1$ feature vector $\mathbf{h}$. We reserve the letter $\mathbf{h}$ for feature vectors. Occasionally we use other letters, such as $\mathbf{x}$ and $\mathbf{t}$, for vectors in this space but we try to only use $\mathbf{h}$ where possible.

Dual Vector Space The dual vector space $(\mathbb{R}^D)^*$ is the real vector space of linear functions $\mathbb{R}^D \to \mathbb{R}$. The linear functions are given by a $1 \times D$ dimensional row vector $\mathbf{w}$. We reserve the letter $\mathbf{w}$ exclusively for parameter vectors, with no other letter representing vectors from this space.

Because the vector space consists of column vectors and the dual space consists of row vectors we omit the transpose when writing the linear function $\mathbf{wh}, \mathbf{w} \in (\mathbb{R}^D)^*, \mathbf{h} \in \mathbb{R}^D$.

Affine Subspace A linear subspace is a vector subspace of $\mathbb{R}^D$ that contains the origin $\mathbf{0} \in \mathbb{R}^D$. An affine subspace is a translation of a linear subspace. The dimension of an affine subspace is the dimension of the associated linear subspace. Examples of affine subspaces include points, lines, planes, and hyperplanes.

We illustrate this definition with an example. Let us consider a vector space $\mathbb{R}^3$. We can define a $\mathbb{R}^2$ linear subspace as:

$$\{\lambda_1 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \lambda_2 \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}\} \tag{5.3}$$

This subspace takes the form of a plane in $\mathbb{R}^3$ that contains the origin. An affine subspace can be created by a translation in the third dimension:

$$\{\lambda_1 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \lambda_2 \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\} \tag{5.4}$$

The affine subspace is now a plane in $\mathbb{R}^3$ that does not contain the origin. In this example many unique affine subspaces can be defined depending on the length of the third $[0,0,1]^T$ vector. All these subspaces are two dimensional.

Affine Hull Given $P < D$ let a P-dimensional affine subspace of $\mathbb{R}^D$ be denoted A. Consider a finite set of feature vectors $\{\mathbf{h}_1, \ldots, \mathbf{h}_n\} \subseteq A$. Using the definition of an affine subspace we can decompose the feature vectors into a term $\mathbf{x}_i$ that is contained in the P-dimensional linear subspace and a translation term $\mathbf{t}$. We rewrite each feature vector in terms of this decomposition $\mathbf{h}_i = \mathbf{x}_i + \mathbf{t}$ where $1 \leq i \leq n$. The linear combination of a finite set of feature vectors is:

$$\begin{aligned} \mathbf{h} &= \lambda_1 \mathbf{h}_1 + \ldots + \lambda_n \mathbf{h}_n \\ &= \lambda_1 (\mathbf{x}_1 + \mathbf{t}) + \ldots + \lambda_n (\mathbf{x}_n + \mathbf{t}) \\ &= \lambda_1 \mathbf{x}_1 + \ldots + \lambda_n \mathbf{x}_n + \mathbf{t} \sum_{i=1}^{n} \lambda_i \end{aligned} \tag{5.5}$$

If $\mathbf{h} \in A$ then $\mathbf{h} = \mathbf{x} + \mathbf{t}$. The value of $\mathbf{t}$ is fixed, therefore for $\mathbf{h}$ to be in the affine subspace A the coefficients λ_i must sum to one. The affine subspace can be written as

$$A = \{\lambda_1 \mathbf{h}_1 + \ldots + \lambda_n \mathbf{h}_n : \sum_{i=1}^{n} \lambda_i = 1\} \tag{5.6}$$

An affine subspace defined in terms of a finite set of vectors of this form is called the affine hull. The notation $\text{aff}(\{\mathbf{h}_1, \ldots, \mathbf{h}_n\})$ denotes the affine hull of a given finite set of feature vectors.

Convex Set A set of feature vectors $K \subseteq \mathbb{R}^D$ is convex if for any two vectors $\mathbf{h}_i, \mathbf{h}_j \in K$ also contains the straight line segment $[\mathbf{h}_i, \mathbf{h}_j] = \{\lambda \mathbf{h}_i + 1(1-\lambda)\mathbf{h}_j : 0 \leq \lambda \leq 1\}$ between them.

Convex Hull For the set of feature vectors $K \subseteq R^D$ the convex hull is defined as the intersection of all convex sets that contain K

$$\text{conv}(K) := \cap\{K' \subseteq \mathbb{R}^D : K \subseteq K', K' \text{ convex}\} \tag{5.7}$$

It can be shown [162] that if $K = \{\mathbf{h}_1, \ldots, \mathbf{h}_n\}$ is finite, then the convex hull can be constructed as:

$$\text{conv}(K) = \{\lambda_1 \mathbf{h}_1 + \ldots + \lambda_n \mathbf{h}_n : \lambda_i \geq 0, \sum_{i=1}^{n} \lambda_i = 1\} \tag{5.8}$$

Note that the convex hull is an affine hull with an additional constraint. A convex hull is embedded inside an affine hull. The dimensionality $(\dim)$ of a convex hull is the dimensionality of the affine subspace defined by the affine hull of its finite point set

$$\dim(\text{conv}(K)) = \dim(\text{aff}(K))$$

.

Convex Polytope The convex hull of a set of finite feature vectors. Because we do not consider non-convex polytopes we follow the convention of Ziegler [162] and drop the convex adjective from all following discussions.

Conical Hull For a specified collection of parameter vectors $C = \{\mathbf{w}_1, \ldots, \mathbf{w}_n\}$ the conical

hull is a finitely generated combination of parameter vectors in $(\mathbb{R}^D)^*$

$$\text{cone}(C) := \{t_1\mathbf{w}_1 + \ldots + t_n\mathbf{w}_n : t_i \geq 0\}$$

Unlike a convex hull the conical hull is unbounded, and always includes the origin $\mathbf{0}$. Because the cone always contains the origin, it is embedded in some linear subspace. The dimension of the subspace that contains $\text{cone}(C)$, and thus $\dim(\text{cone}(C))$ is the rank of the matrix formed from the vectors in C.

It seems that we have made an arbitrary decision to define polytopes in $\mathbb{R}^D$ and conical hulls in $(\mathbb{R}^D)^*$. As stated in the introduction to this section, we wish to restrict the description of convex geometry to concepts that are directly applicable to SMT. For the purposes of describing MERT we can restrict these geometric objects to these spaces.

5.2.2 Faces in Feature Space

Recall from the introduction that a feature vector is labelled extreme if the system of inequalities in (5.2) can be satisfied. With an appropriate parameter an extreme feature vector gives the maximal model score over all other feature vectors in a K-best list. An extreme feature vector is an example of a special type of face of a polytope, called a vertex. Following Ziegler [162] we provide a formal definition of a face and describe some its properties.

H The polytope $H \in \mathbb{R}^D$ is the convex hull of the finite set of feature vectors $\{\mathbf{h}_1, \ldots, \mathbf{h}_K\}$ associated with the N hypotheses $\{\mathbf{e}_1, \ldots, \mathbf{e}_K\}$.

Valid Inequality A parameter $\mathbf{w} \in (\mathbb{R}^D)^*$ and a value $y \in \mathbb{R}$ define a *valid inequality* over H if

$$\mathbf{wh} \leq y \;\; \forall \mathbf{h} \in H.$$

Faces in $\mathbb{R}^D$ Suppose $\mathbf{w} \in (\mathbb{R}^D)^*$ and $y \in \mathbb{R}$ form a valid inequality. A *face* is defined as

$$F = \{\mathbf{h} \in H : \mathbf{wh} = y\}. \tag{5.9}$$

From the definitions of a face and the valid inequalities, it also follows that

$$F = \{\mathbf{h} \in H : \mathbf{wh} = \max_{\mathbf{h}' \in H} \mathbf{wh}'\}$$

In this way, $\mathbf{w}$ defines a face F.

Vertex A face consisting of a single point is called a *vertex*. The concept of a vertex is very similar to the definition of an extreme point. Note that Galley and Quirk [62] disqualify feature vectors that lie on line segments between two extreme points from extremity, which is consistent with the definition of a vertex.

The main difference between the two concepts is rather minor: a vertex is a set that includes a single feature vector, whereas an extreme point is a feature vector itself. For clarity we exclusively use the term vertex from this point on.

Vertex Set The vertices of a polytope H are denoted $\text{vert}(H)$.

From these definitions it is possible to make a number of important statements about faces. The following Proposition concerns the relationship between a polytope and its vertices.

Proposition 5.1. *Let $H \subseteq \mathbb{R}^D$ be a polytope.*

(i) Every polytope is a convex hull of its vertices: $H = \text{conv}(\text{vert}(H))$.

(ii) If a polytope can be written as the convex hull of a finite set of feature vectors $H = \text{conv}(\text{vert}(V))$, *then the set contains all the vertices of the polytope:* $H = \text{conv}(V)$ *implies that* $\text{vert}(H) \subseteq V$.

Proof. See Ziegler [162, proposition 2.2] □

Let us now consider all faces in the polytope, other than vertices.

Proposition 5.2. *Let $H \subseteq \mathbb{R}^D$ be a polytope, and $V = \text{vert}(H)$. Let F be a face of H.*

(i) The face F is a polytope, with $\text{vert}(F) = F \cap V$.

(ii) Every intersection of faces of H is a face of H.

(iii) The faces of F are exactly the faces of H that are contained in F.

(iv) $F = H \cap \text{aff}(F)$.

Proof. See Ziegler [162, proposition 2.3] □

Let us now consider the structure of a polytope implied by these propositions. A polytope contains faces, which are polytopes themselves. These polytopes in turn also contain faces, which are again polytopes. The nesting of polytopes is continued until the vertices are reached. From these two propositions we can make the following corollary

Corollary 5.3. *Let $H \subseteq \mathbb{R}^D$ be a polytope*

(i) H has a finite number of faces

(ii) A face F of H has dimension $\dim(F) = \dim(\text{aff}(F))$

Proof. For Part (i) we note that $\text{vert}(H)$ is finite. Since Proposition 5.2 states that for a face $F \subseteq H$ the vertices of the face obey $\text{vert}(F) \subseteq \text{vert}(H)$, there are only a finite number of sets of vertices that could define a face.

Part (ii) follows immediately from Proposition 5.2 Part (iv) and definition of the affine hull in Section 5.2.1 □

There are two classes of faces with interesting properties:

Edge A face F is an edge if $\dim(F) = 1$. An edge takes the form of a line segment between two vertices $\mathbf{h}_i$ and $\mathbf{h}_j$ in the polytope H. The edge can be written as $[\mathbf{h}_i, \mathbf{h}_j] = \text{conv}(\mathbf{h}_i, \mathbf{h}_j)$.

Corollary 5.4. *Let $H \in \mathbb{R}^D$ be a polytope. Two vertices $\mathbf{h}_i$ and $\mathbf{h}_j$ in the polytope H form an edge if and only if there is a linear decision boundary in $(\mathbb{R}^D)^*$ between the parameters that maximise $\mathbf{h}_i$ and those that maximise $\mathbf{h}_j$.*

Proof. The proof follows directly from the definition of an edge. For the edge to exist the following system of constraints must be feasible:

$$\begin{aligned} &\mathbf{w}(\mathbf{h}_j - \mathbf{h}_i) = 0 \\ &\mathbf{w}(\mathbf{h}_k - \mathbf{h}_i) < 0,\ 1 \leq k \leq K, k \neq i, k \neq j \\ &\mathbf{w}(\mathbf{h}_l - \mathbf{h}_j) < 0,\ 1 \leq l \leq K, l \neq i, l \neq j \end{aligned} \tag{5.10}$$

The set of parameters that can satisfy these constraints is a hyperplane of dimension $D-1$ in $(\mathbb{R}^D)^*$, which forms the decision boundary. □

Facet A face F is a facet if $\dim(F) = D-1$. Note that because the affine hull of a facet is a $D-1$ hyperplane, the facet is uniquely defined by the facet normal $\mathbf{w}_F$.

Corollary 5.5. *Let $H \in \mathbb{R}^D$ be a polytope, then every face of H is contained in a facet of H*

Proof. Ziegler [162, Corollary 2.14] describes how for any polytope H a polar polytope H^Δ can be constructed. The faces in H are translated to H^Δ such that the face F of $\dim(F) = P$ becomes a face F^Δ of $\dim(F^\Delta) = D-P-1$ and inclusion is reversed such that if $F' \subset F$ then

$F'^{\Delta} \supset F^{\Delta}$. Because facets are translated to vertices and following directly from Proposition 5.2 Part (i) the Corollary must be true otherwise the polar polytope cannot be constructed. See Ziegler [162] for the detailed presentation. □

5.2.3 The Normal Cone

We now switch to describing geometric objects in the dual vector space $(\mathbb{R}^D)^*$. It should be noted that $(\mathbb{R}^D)^*$ is a linear space like the primary feature space $\mathbb{R}^D$ and it is equally valid to define geometric objects here. Because MERT is concerned with assigning error to parameters, the geometric objects in this space are much more relevant to SMT as it is the optimum value of parameters that are sought by training procedures.

In the previous subsection we used a single parameter vector to define a set of feature vectors called a face. In this subsection we reverse this process and consider the set of parameter vectors that could be used to give a single face.

Normal Cone Following from the constraints in (5.2) and the definition of a face in Section 5.2.2, for the face F in polytope H the normal cone N_F takes the form

$$N_F = \{\mathbf{w} : \mathbf{w}(\mathbf{h}_j - \mathbf{h}_i) \leq 0, \forall \mathbf{h}_i \in \text{vert}(F), \forall \mathbf{h}_j \in \text{vert}(H)\} \tag{5.11}$$

When the face is a vertex $F = \{\mathbf{h}_i\}$ then the normal cone is identical to the set of parameters that satisfy the constraints in (5.2).

Recall the definition of an edge from the previous section. This definition allows us to make the following statement

Lemma 5.6. *Given the polytope H the set of edges contained in the polytope is denoted E. The normal cone N_F of the face F can be described by the subset of constraints that also form edges*

$$N_F = \{\mathbf{w} : \mathbf{w}(\mathbf{h}_j - \mathbf{h}_i) \leq 0, \forall \mathbf{h}_i \in \text{vert}(F), \forall \mathbf{h}_j \in \text{vert}(H), [\mathbf{h}_i, \mathbf{h}_j] \in E\} \tag{5.12}$$

Proof. Let us consider the vertex $\mathbf{h}_m$ that does *not* form an edge with any vertex $\mathbf{h}_i \in F$. Substituting $\mathbf{h}_m$ for $\mathbf{h}_j$ into the constraints in (5.10) results in an infeasible system. The implication being that there is no decision boundary for these two vertices and that $\mathbf{wh}_i > \mathbf{wh}_m, \forall \mathbf{h}_i \in F, \forall \mathbf{w} \in N_F \setminus \mathbf{0}$. □

The constraints that are not essential for a normal cone are another example of *redundancy*. It is the redundancy in constraints that is exploited for the multi-sentence MERT

algorithm in Chapter 6.

The name of this set, the normal cone, may seem a little mysterious at this point. We leave the full definition of a cone to Section 5.3.2, for now we can assume that a cone is the finite intersection of a set of closed linear half spaces that include the origin. Let us consider the following statement about the dimensionality of N_F, which explains what makes a cone 'normal'.

Lemma 5.7. *Let H be a polytope, F be a face in H, and N_F the normal cone of F. The dimensionality of N_F is related to F by*

$$\dim(N_F) = D - \dim(F)$$

Proof. Consider the matrix A formed by the M vertices of F. The rank of A is equal to the dimensionality of the affine hull containing F. Recall from Section 5.2.1 that because F is contained in affine hull that any $\mathbf{h}_i \in A$ can be written as $\mathbf{h}_i = \mathbf{t} + \mathbf{x}_i$ where $\mathbf{t}$ is a constant translation.

Now due to Eqn. (5.9) for any parameter $\mathbf{w} \in N_F$ the model score will be some constant y such that $\mathbf{w}\mathbf{h}_i = y$ for all $\mathbf{h}_i \in A$. Let $\mathbf{y}$ be a M-dimensional row vector with the model scores $\mathbf{y} = [y, \ldots, y]$, T be the matrix of M columns containing the constant translation $T = [\mathbf{t}, \ldots, \mathbf{t}]$ and X the matrix of the feature vectors components in a linear subspace $X = [\mathbf{x}_i, \ldots, \mathbf{x}_M]$ we can write

$$\mathbf{w}(T + X) = \mathbf{y}$$

which implies that $\mathbf{w}X = 0$, the condition for $\mathbf{w}$ to be in null space of A. The normal cone N_F is therefore contained in the null space of A which is of dimension $D - \dim(F)$. □

The set of vectors in N_F are all normal to the affine subspace containing the face F, hence the use of the adjective 'normal' in the term 'normal cone'. We have seen examples of these orthogonal spaces, the hyperplane in $(\mathbb{R}^D)^*$ associated with an edge is the null space of the line that passes through both vertices in the edge. Another example is a facet, which is a $D-1$ dimensional face associated with a 1-dimensional facet normal.

Face of a Cone A vector $\mathbf{e} \in \mathbb{R}^D$ defines a *valid inequality* for cone $N \in (\mathbb{R}^D)^*$ if

$$\mathbf{w}\mathbf{e} \leq 0 \; \forall \mathbf{w} \in N$$

Suppose $\mathbf{e} \in \mathbb{R}^D$ defines a valid inequality, it follows that a *Face of a Cone* is

$$(F)^* = \{\mathbf{w} \in N : \mathbf{w}\mathbf{e} = 0\}$$

The fact that both polytopes and cones have faces may lead to some confusion. In the definition of the normal fan we see that there is a relationship between faces of both types of objects. To minimise confusion, if we refer to a face then we mean a face of a polytope in $\mathbb{R}^D$. We always qualify references to faces in $(\mathbb{R}^D)^*$ by calling them a face of a cone or a face in $(\mathbb{R}^D)^*$. Similarly any reference to F or F' refers to faces of a polytope in $\mathbb{R}^D$ and $(F)^*$ refers to a face of a cone in $(\mathbb{R}^D)^*$.

5.2.4 The Normal Fan

In the previous subsection the normal cone was introduced. This set of all normal cones is called the normal fan. Let us start with the description of a fan [162].

Fan The set of cones $\mathscr{N}$ is a fan if it has the following two properties

1. Every nonempty face of a cone in $\mathscr{N}$ is also a cone in $\mathscr{N}$.
2. The intersection of any two cones in $\mathscr{N}$ is a face of both of cones.

Let us consider characteristics of normal cones.

Lemma 5.8. *For the polytope H and the face $F \subset H$, a face of the normal cone $(F)^* \subset N_F$ is the normal cone for some other face F' in H such that $(F)^* = N_{F'}$.*

Proof. Let F' be a face in the polytope with the inclusion $F \subset F'$. Now consider the following vector

$$\mathbf{e} = \sum(\mathbf{h}_j - \mathbf{h}_i), \text{ for all } \mathbf{h}_j \in \text{vert}(F'), \mathbf{h}_i \in \text{vert}(F) \tag{5.13}$$

For any parameter $\mathbf{w}_F \in N_F$ the following condition is true

$$\mathbf{w}_F\mathbf{h}_i \geq \mathbf{w}_F\mathbf{h}_j, \text{ for all } \mathbf{h}_j \in \text{vert}(F'), \mathbf{h}_i \in \text{vert}(F) \tag{5.14}$$

Thus, the condition $\mathbf{w}_F\mathbf{e} \leq 0$ for $\mathbf{e}$ to form a valid inequality in $(\mathbb{R}^D)^*$ is satisfied. Now let us consider the normal cone of F'. Any parameter $\mathbf{w}_{F'} \in N_{F'}$ satisfies the condition

$$\mathbf{w}_{F'}\mathbf{h}_i = \mathbf{w}_{F'}\mathbf{h}_j, \text{ for all } \mathbf{h}_j \in \text{vert}(F'), \mathbf{h}_i \in \text{vert}(F) \tag{5.15}$$

Implying that $\mathbf{w}_{F'}\mathbf{e} = 0$, which is the condition for $N_{F'}$ to be a face in the normal cone N_F. For any two faces with the inclusion $F \subset F'$, we can always find some $\mathbf{e}$ that yields $N_{F'}$ as a face of N_F.

To complete the proof we need to show that is impossible to find a face of the cone $(F)^* \subset N_F$, such that any parameter in $\mathbf{w}_{(F)^*} \in (F)^*$ solely maximises the feature vectors in F and not another face $F' \in H$. For this we note that for $(F)^*$ to be a face of the cone N_F, then $\mathbf{w}_{(F)^*}$ must be on some constraint of the normal cone.

Lemma 5.6 states the all constraints are derived from edges, such that at least one of the two vertices in the edge is contained in F. Let us consider the edge $[\mathbf{h}_i, \mathbf{h}_j]$ with $\mathbf{h}_i, \mathbf{h}_j \in F$. Now, for all $\mathbf{w} \in N_F$ we can state $\mathbf{w}\mathbf{h}_i = \mathbf{w}\mathbf{h}_j$ implying the normal cone of F is fully contained in the normal cone of the edge $N_F \subset N_{[\mathbf{h}_i,\mathbf{h}_j]}$, and $N_{[\mathbf{h}_i,\mathbf{h}_j]}$ cannot be a face of of N_F.

Let us change the edge such that $\mathbf{h}_i \in F$ and $\mathbf{h}_j \notin F$. If $w_{(F)^*}$ is contained within this constraint, then it must also maximise $\mathbf{h}_j$. Therefore $w_{(F)^*}$ is the maximiser for some other face $F' \neq F$, where $F \subset F'$ and $\mathbf{h}_j \in F'$.

□

Lemma 5.9. *For the polytope H with two faces $F \subset H$ and $F' \subset H$, the intersection of the two normal cones $N_F \cap N_{F'}$ is a face in both normal cones.*

Proof. If a parameter $\mathbf{w}$ lies in the intersection of the two cones $N_F \cap N_{F'}$ then it maximises all feature vectors in $F \cup F'$, which implies that $F \cup F'$ is a face itself and that $N_F \cap N_{F'} = N_{F \cup F'}$. Using a similar argument to Lemma 5.8, it can be shown that $N_{F \cup F'}$ is face in both N_F and $N_{F'}$. □

Corollary 5.10. *For a polytope H the normal cones associated with all faces in H form a fan.*

Proof. Condition 1 is satisfied by Lemma 5.8 and Condition 2 is satisfied by Lemma 5.9.

□

This result leads us directly to our desired goal.

Normal Fan For a polytope H the normal fan $\mathcal{N}(H)$ is the fan constructed from the normal cones of all faces of H.

In Section 5.2.2 we remarked on the nesting structure of the faces in a polytope. The normal fan reflects this structure except with inclusion reversed, such that for two faces F and F' in the polytope H with $F \supset F'$ the normal cones have the opposite relationship $N_{F'} \supset N_F$.

Note that every normal fan includes the zero vector $\mathbf{0}$, which is the normal cone for the entire polytope H. This 0-dimensional normal cone is a face of all normal cones in the normal fan.

	$h_{LM} : \log(P_{LM}(\mathbf{e}))$	$h_{TM} : \log(P_{TM}(\mathbf{f}\|\mathbf{e}))$
$\mathbf{e}_1$	-0.1	-1.2
$\mathbf{e}_2$	-1.2	-0.2
$\mathbf{e}_3$	-0.9	-1.6
$\mathbf{e}_4$	-0.9	-0.1
$\mathbf{e}_5$	-0.8	-0.9

Table 5.1 An example set of two dimensional feature vectors (after Cer et al. [29], Table 1) with language model (h_{LM}) and translation model (h_{TM}) components. A fifth feature vector has been added to illustrate redundancy.

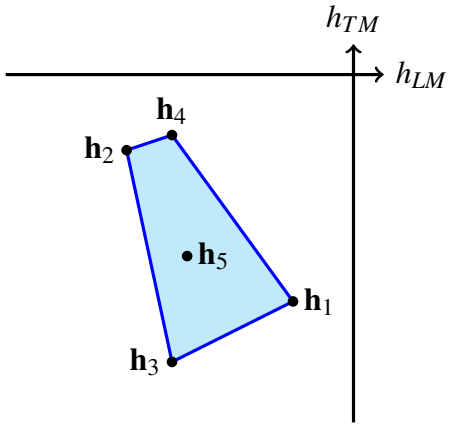


Fig. 5.1 The polytope constructed from the feature vectors in Table 5.1. The shaded area is the convex hull, and the thick blue lines are the polytope's edges.

Let us now return to the task at hand and consider what objects in the normal fan are important for MERT.

Normal cones for vertices The set of parameters that satisfy 5.2. To perform MERT we need to enumerate through all these cones.

Normal cones for edges These are cones embedded in $D-1$ hyperplanes in $(\mathbb{R}^D)^*$ They are the boundaries between the normal cones of vertices. They are also used to define the concept of adjacency used in Chapter 6 for the polynomial-time multi-sentence MERT algorithm.

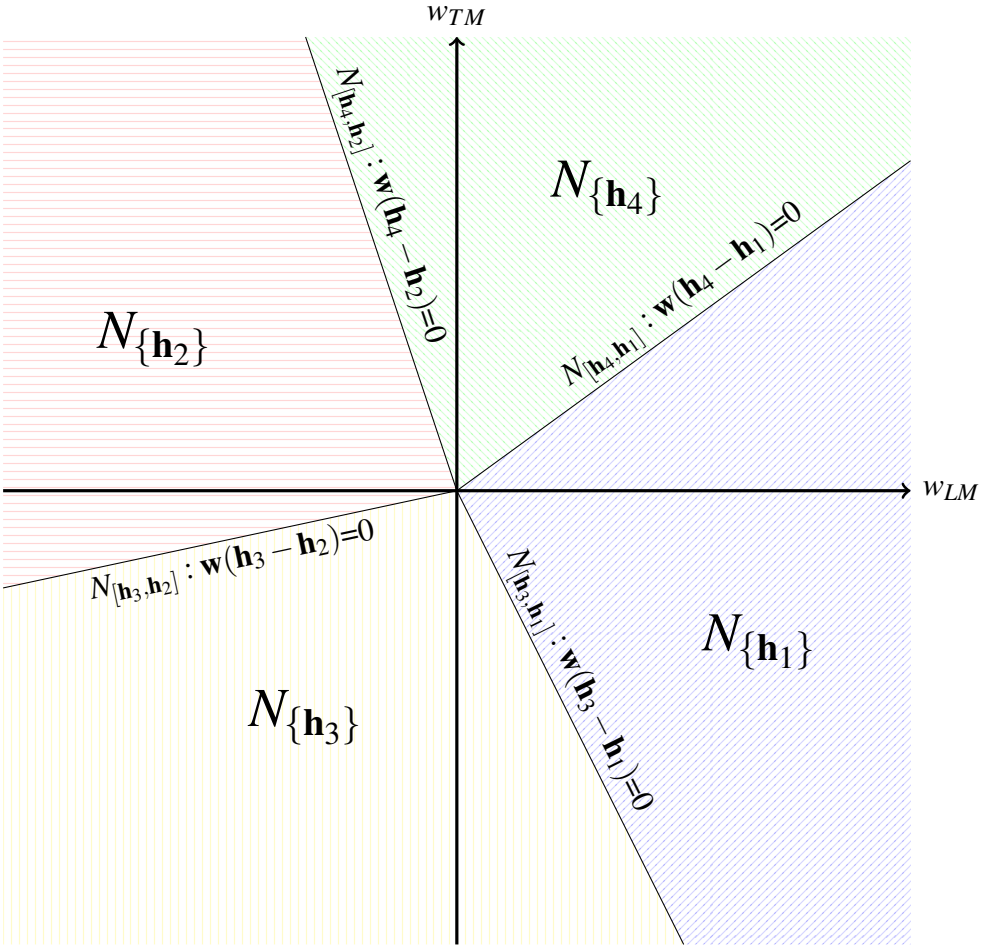


Fig. 5.2 Reproduction of Figure 1 from Cer et al. [29] showing parameter space for the feature vectors in Table 5.1 and the polytope in Figure 5.1. This figure is an example of a normal fan. There are nine normal cones shown, the four associated with non-redundant hypotheses, the four associated with edges, and the **0** vector.

5.2.5 An Example of the Normal Fan

Throughout this section we have described a formulation of sentence-level MERT using convex geometry. We now illustrate these concepts with a worked example taken from Cer et al. [29]. Our goal is to expand upon the description of Cer et al. [29] and give the reader a familiar entry point into convex geometry.

Cer et al. [29] start with a singe source sentence $\mathbf{f}$ and an SMT system based on two features: the translation model ($P_{TM}(\mathbf{f}|\mathbf{e})$) and the language model ($P_{LM}(\mathbf{e})$). The system produces a set of four hypotheses $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4\}$. These hypotheses yield four 1×2 feature vectors of log-probabilities $\{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{h}_4\}$ shown in Table 5.1. To this set of four hypotheses, we add an fifth hypotheses $\mathbf{e}_5$ and feature vector $\mathbf{h}_5$ to illustrate redundancy of feature vectors.

In Figure 5.1 we have drawn the polytope from the feature vectors in Table 5.1. In this example the first four feature vectors are vertices, and $\mathbf{h}_5$ is redundant. We also note that there are four edges in the polytope $[\mathbf{h}_4, \mathbf{h}_2], [\mathbf{h}_4, \mathbf{h}_1], [\mathbf{h}_3, \mathbf{h}_2]$, and $[\mathbf{h}_3, \mathbf{h}_1]$.

Cer et al. [29] actually show an example of a normal fan, albeit they do not name it as such. The diagram in Figure 5.2 is a reproduction of Figure 1 from Cer et al. [29]. It shows a normal fan for the polytope in Figure 5.1. Note how the decision boundaries in the fan correspond to the edges in the polytope.

There are nine normal cones in the diagram. Four regions are associated with vertices, for example the region associated with the first hypotheses is maximal is labelled $N_{\{\mathbf{h}_1\}}$. Additional four normal cones are the decision boundaries associated with edges, e.g. in this example the normal cone for the edge $[\mathbf{h}_1, \mathbf{h}_2]$ is labelled as $N_{[\mathbf{h}_1, \mathbf{h}_2]}$. The final normal cone is the $\mathbf{0}$ vector, which is the normal cone for the entire polytope.

5.3 Dual Representations of Polytopes and Cones

In the previous section we described the normal fan as a set of normal cones, which are themselves defined by a set of decision boundaries given by edges. This description is sufficient for understanding the rest of this chapter, and following chapters.

In this section, we describe how polytopes and cones have equivalent, dual representations. We mainly provide this description for completeness: for example, it explains why we use the two terms ‘polytope’ and ‘convex hull’ to refer to the same geometric object. Additionally, the dual description is an integral part of many algorithms and proofs used in convex geometry.

Let us start with a theorem that Ziegler [162] refers to as the 'main theorem for polytopes':

Theorem 5.11. *A subset $H \subseteq \mathbb{R}^D$ is the convex hull of K feature vectors (a $\mathscr{V}$-polytope)*

$$H = \text{conv}(V) \quad \textit{for some } V \in \mathbb{R}^{D \times K}$$

if and only if it is a bounded intersection of M half spaces (an $\mathscr{H}$-polytope)

$$H = \{\mathbf{h} \in \mathbb{R}^D : A\mathbf{h} \leq \mathbf{z}\} \quad \textit{for some } A \in \mathbb{R}^{M \times D}, \mathbf{z} \in \mathbb{R}^M$$

Proof. See Ziegler [162, Theorem 1.1] □

It should be clear now why there is a distinction made between polytopes and convex hulls. A polytope is a set of points in $\mathbb{R}^D$ given in either representation, and the convex hull is the $\mathscr{V}$-polytope representation. The set of half spaces defined by A and $\mathbf{z}$ must at least contain the half spaces given by the affine hulls of the facets of H, but the set may also contain redundant half spaces.

The full proof of Theorem 5.11 occupies the majority of Chapter 1 of Ziegler [162], but we can provide a short sketch of the proof. In the direction of a $\mathscr{V}$-polytope to an $\mathscr{H}$-polytope a $\mathscr{V}$-polytope can be written as

$$\{\mathbf{h} \in \mathbb{R}^D : \exists \boldsymbol{\lambda} \in \mathbb{R}^K : \mathbf{h} = V\boldsymbol{\lambda}, \boldsymbol{\lambda} \geq \mathbf{0}, \mathbf{1}\boldsymbol{\lambda} = 1\} \tag{5.16}$$

where $\mathbf{0}$ is a $1 \times K$ vector of zeros and $\mathbf{1}$ is a $1 \times K$ vector of ones. This representation can be rewritten as

$$\{(\mathbf{h}, \boldsymbol{\lambda}) \in \mathbb{R}^{D+K} : \mathbf{h} = V\boldsymbol{\lambda}, \boldsymbol{\lambda} \geq \mathbf{0}, \mathbf{1}\boldsymbol{\lambda} = 1\} \tag{5.17}$$

We note that a set of K equalities in the form $\mathbf{h} = V\boldsymbol{\lambda}$ can written as a set of $2K$ inequalities of the form $\mathbf{h} \geq V\boldsymbol{\lambda}, \mathbf{h} \leq V\boldsymbol{\lambda}$. The set in Eqn. (5.17) is therefore a $\mathscr{H}$-polytope of dimension $D+K$.

The key to the full proof is an operation called Fourier-Motzkin elimination. This operation applies a transformation, which is called a projection, to the $\mathscr{H}$-polytope. The result of Fourier-Motzkin elimination is a new $\mathscr{H}$-polytope such that kth feature in the features vectors is deleted

$$\text{proj}_k(H) = \{\mathbf{h} - h_k\mathbf{e}_k : \mathbf{h} \in H\} \tag{5.18}$$

where $\mathbf{e}_k$ is a unit vector along the kth axis. After K applications of Fourier-Motzkin elimination the dual representation is found.

In the other direction, a $D+M$ dimensional $\mathscr{V}$-polytope is constructed from an $\mathscr{H}$-polytope. It can be shown that the intersection of a $\mathscr{V}$-polytope with a hyperplane results in a new $\mathscr{V}$-polytope. The D-dimensional $\mathscr{V}$-polytope is constructed through M hyperplane intersections.

5.3.1 Algorithms for Computing Dual Representations

We have described Theorem 5.11 that allows for two representations of polytopes. One of the reasons that the dual representation is interesting to SMT is that the algorithms for moving between representations also identify redundancies in the primary representation. For example, moving from a $\mathscr{V}$-polytope to a $\mathscr{H}$-polytope identifies the vertices in the set of feature vectors that form the $\mathscr{V}$-polytope.

There exist two types of algorithm to compute a dual representation: one type enumerates vertices of polytopes given a set of half spaces, the other enumerates facets given a set of feature vectors [162]. The facet enumeration problem is often called the convex hull problem [162]. The facet enumeration problems and vertex enumeration problems can be transformed into each other by the use of the polar polytope referred to in the proof to Corollary 5.5. To create the polar polytope, a feature vector interior to the polytope is needed. This interior feature vector can be found using a linear program similar to the one in (3.15).

An interesting characteristic of the proof to Theorem 5.11 is that it describes algorithms based upon Fourier-Motzkin elimination and hyperplane intersections of polytopes. An issue with Fourier-Motzkin elimination is that it creates an exponential number of inequalities [162] to create the projected representation. This combinatorial explosion can be ameliorated by keeping track of both descriptions of the polytope and using heuristics based on this dual description to remove redundant inequalities. This is the idea behind the double description algorithm [112].

An alternative to the double description algorithm is the QuickHull algorithm [12] for facet enumeration. Similar to the double description algorithm, this algorithm considers both representations simultaneously. A set of facets is computed for a minimal subset of feature vectors. The rest of the feature vectors are then incrementally tested against facets that divide feature vectors from the minimal polytope. If the tested feature vector is a vertex then the facets which include the vertex are recomputed.

Another algorithm for vertex enumeration is the reverse search algorithm [9]. We describe a variant of the reverse search algorithm for computing the Minkowski sum of polytopes in Chapter 6.

These algorithms do not have the strong polynomial time guarantees of linear programming. In practise, they can be very effective for certain problems. For example, Galley and Quirk [62] show that QuickHull performs an order of magnitude faster than linear programming for problems of a low dimension.

5.3.2 The Dual Representation of a Normal Cone

Let us now move to definition of a cone. In Section 5.2.3 the normal cone was described as the intersection of a set of half spaces. A cone also has a dual representation and there exists a 'main theorem for cones':

Theorem 5.12. *A cone $N \subseteq (\mathbb{R}^D)^*$ is a finitely generated conical hull of L vectors*

$$N = \text{cone}(\{\mathbf{w}_1, \ldots, \mathbf{w}_L\})$$

if and only if it is a finite intersection of M closed linear half spaces

$$N = \{\mathbf{w} \in (\mathbb{R}^D)^* : E\mathbf{w}^T \leq \mathbf{0}\} \text{ for some } E \in \mathbb{R}^{M \times D}$$

Proof. See Ziegler [162, Theorem 1.3] □

We now investigate the conical hull representation of the cone.

Ray of a Cone A face of a cone with the property $\dim((F)^*) = 1$ is denoted a ray of the cone. It can be defined by a single vector $\mathbf{w}_r$ such that the ray takes the form $\{\mathbf{w} : t\mathbf{w}_r, t > 0\}$. The rays serve a similar function for cones that vertices serve for polytopes.

Ziegler [162] notes at the start of Chapter 2 that a version of Proposition 5.1 exists for cones and rays. Similarly to how a polytope can be defined as the convex hull of its vertices, a cone can be defined as the conical hull of its ray set. Now let us examine the relationship between facets and normal cones

Lemma 5.13. *Let F be a face in the polytope $H \subseteq \mathbb{R}^D$ and let $N_F \subseteq (\mathbb{R}^D)^*$ be the normal cone of F. The set $\{P_1, \ldots, P_L\}$ is the set of facets that contain F. Each facet is the intersection of the polytope with a $D-1$ hyperplane given by a facet normal $\mathbf{w}_{P_l}$ with $1 \leq l \leq L$. The normal cone is the conical hull of the facet normals.*

$$N_F = \text{cone}(\{\mathbf{w}_{P_1}, \ldots, \mathbf{w}_{P_L}\})$$

Proof. First we show that a facet normal is always a ray in the normal cone for F. For the facet P such that $F \subset P$ let us consider the inequality defined by the vector

$$\mathbf{e}_P = \sum(\mathbf{h}_j - \mathbf{h}_i), \text{ for all } \mathbf{h}_j \in \text{vert}(P), \mathbf{h}_i \in \text{vert}(F)$$

For any $\mathbf{w} \in N_F$ the model score obeys $\mathbf{w}\mathbf{e}_P \leq 0$ satisfying the definition of a valid inequality. Only a facet normal $\mathbf{w}_P$ satisfies $\mathbf{w}_P\mathbf{e}_P = 0$, thereby satisfying the definition of a ray.

Next we show that no other vector $\mathbf{w}'$ contained in N_F could form a ray. To be a ray there must be some vector $\mathbf{e}'$ that defines a valid inequality and $\mathbf{w}'$ must be unique in the sense that only scaled versions of the parameter satisfy $t\mathbf{w}'\mathbf{e}' = 0$ where $t > 0$. To satisfy this condition $\mathbf{w}'$ must be a maximiser for a face $F' \subseteq F$. Because of Corollary 5.5 the face F' is contained in a facet P, meaning that $\mathbf{w}_P\mathbf{e}' = 0$ which invalidates the uniqueness of $\mathbf{w}'$. □

This Lemma demonstrates an additional benefit of using a facet enumeration algorithm to identify vertices in a polytope H. The algorithm computes the set of all facet normals, giving us full representations of normal cones as a conical hull. Recall from Section 5.2.1 that the conical hull definition is

$$\text{cone}(C) := \{t_1\mathbf{w}_1 + \ldots + t_L\mathbf{w}_L : t_i \geq 0, 1 \leq i \leq L\}$$

We can retrieve any representative parameter for a cone by setting these variables to any value greater than zero, or we can set them all to one to retrieve the centroid of the cone. This property of facet enumeration algorithms has been noted in the two dimensional case by Dyer [54] who referred to it as 'point-line duality'.

5.4 Projected MERT

In the preceding sections we have described how sentence level MERT can be modelled using a convex polytope and a normal fan. We illustrated this technique with a reworking of the example from Cer et al. [29] in Section 5.2.5. Up until this point, our new formulation has not yielded any improvements or novelty over LP-MERT or Och's line optimisation.

We can now start applying convex geometry to solve problems that are difficult to model with previous formulations. In this section we describe how Och's line optimisation can be formulated using a projected polytope. We describe an alternative formulation of 1-dimensional optimisation and then we describe how Och's line optimisation could be generalised to searching over M feature dimensions simultaneously, where $1 \leq M \leq D$. We call

this procedure projected MERT. At the end of this section we extend the example from Section 5.2.5 to illustrate how a projected normal fan corresponds to Och's line optimisation.

5.4.1 Affine Projection

This section introduces a form of affine transformation called a projection. We have already encountered a form of the projection operation in Eqn. (5.18). The projection operation that follows is more general because it is not restricted to feature directions. Consider the polytope $H \subseteq \mathbb{R}^P$, the affine transformation $\pi : \mathbb{R}^p \to \mathbb{R}^D$ is any map of the form [162]

$$\pi(\mathbf{h}) = A\mathbf{h} - \mathbf{z} \tag{5.19}$$

with $A \in \mathbb{R}^{(K \times D)}$ and $\mathbf{z} \in \mathbb{R}^K$. If an affine transformation is surjective, then it is called an affine projection [162].

5.4.2 Och's Line Optimisation as a Projection Operation

In this section we reformulate Och's Line Optimisation in terms of an affine projection. Let us consider a single line optimisation. The goal is to select the optimal parameter from a set of parameters $\mathscr{W} \subset (\mathbb{R}^D)^*$. This set $\mathscr{W}$ takes the form of a line in parameter space $(\mathbb{R}^D)^*$. The line is defined by two given vectors: a starting parameter $\mathbf{w}^{(0)} \in (\mathbb{R}^D)^*$ and a direction vector $\mathbf{d} \in (\mathbb{R}^D)^*$. Using the free variable γ the set can be written as:

$$\mathscr{W} = \{\mathbf{w}^{(0)} + \gamma\mathbf{d} : \gamma \in \mathbb{R}\} \tag{5.20}$$

This line in $(\mathbb{R}^D)^*$ can be used to define a projection operation in $\mathbb{R}^D$. We define an affine projection in terms of the given vectors

$$\pi(\mathbf{h}_i) = A_{2,D}\mathbf{h}_i \tag{5.21}$$

where

$$A_{2,D} = \begin{bmatrix} d_1 & \cdots & d_D \\ w_1^{(0)} & \cdots & w_D^{(0)} \end{bmatrix} \tag{5.22}$$

Applying this transformation to vector $\mathbf{h}_i$ results in a transformed feature vector $\tilde{\mathbf{h}}_i \in \mathbb{R}^2$

$$\tilde{\mathbf{h}}_i = \pi(\mathbf{h}_i) = \begin{bmatrix} \tilde{h}_{i,1} \\ \tilde{h}_{i,2} \end{bmatrix} = \begin{bmatrix} \mathbf{d}\mathbf{h}_i \\ \mathbf{w}^{(0)}\mathbf{h}_i \end{bmatrix} \tag{5.23}$$

Now consider a parameter vector of the form

$$\tilde{\mathbf{w}}^{\gamma} = \begin{bmatrix} \gamma & 1 \end{bmatrix} \tag{5.24}$$

The matrix $A_{2,D}$ provides an injective map from $\tilde{\mathbf{w}}^{\gamma}$ to a parameter $\mathbf{w}^{\gamma} \in \mathscr{W}$

$$\tilde{\mathbf{w}}^{\gamma} A_{2,D} = \begin{bmatrix} \gamma & 1 \end{bmatrix} \begin{bmatrix} d_1 & \cdots & d_D \\ w_1^{(0)} & \cdots & w_D^{(0)} \end{bmatrix} = \mathbf{w}^{(0)} + \gamma \mathbf{d} = \mathbf{w}^{\gamma} \tag{5.25}$$

We can state the following relationship between the model scores of the full feature space $\mathbb{R}^D$ and the projected feature space $\mathbb{R}^2$

$$\tilde{\mathbf{w}}^{\gamma} \tilde{\mathbf{h}}_i = \tilde{\mathbf{w}}^{\gamma} A_{2,D} \mathbf{h}_i = \mathbf{w}^{\gamma} \mathbf{h}_i = \gamma \mathbf{d} \mathbf{h}_i + \mathbf{w}^{(0)} \mathbf{h}_i \tag{5.26}$$

Because the model scores are consistent between the full and projected polytopes we can now write a form of constraints that agree with the constraints in (5.2).

$$\begin{aligned} \tilde{\mathbf{w}}^{\gamma} (\tilde{\mathbf{h}_j} - \tilde{\mathbf{h}_i}) \leq 0,\ 1 < j < K \\ \tilde{w}_2^{\gamma} = 1 \end{aligned} \tag{5.27}$$

If a feature vector is extreme in the projected space with $\tilde{\mathbf{w}}^{\gamma}$ it will also be extreme in the full space for the corresponding $\mathbf{w}^{\gamma} \in \mathscr{W}$.

The constraint $\tilde{w}_2^{\gamma} = 1$ can be loosened. If $\tilde{\mathbf{h}}_i$ is extreme then there may exist a parameter $\tilde{\mathbf{w}} \in \tilde{N}_{\tilde{\mathbf{h}}_i} \subset (\mathbb{R}^2)^*$ where $\tilde{w}_2 \neq 1$. From the definition of a cone we can state the following

$$\{ t\tilde{\mathbf{w}} : t > 0 \} \subseteq \tilde{N}_{\tilde{\mathbf{h}}_i} \tag{5.28}$$

Setting $t = \frac{1}{\tilde{w}_2}$ allows the recovery of a parameter that can be mapped to $(\mathbb{R}^D)^*$

$$\tilde{\mathbf{w}}^{\gamma} = \begin{bmatrix} \frac{\tilde{w}_1}{\tilde{w}_2} \\ 1 \end{bmatrix}, \quad \tilde{w}_2 > 0 \tag{5.29}$$

We can therefore rewrite the constraints that need to be satisfied as:

$$\begin{aligned} \tilde{\mathbf{w}} (\tilde{\mathbf{h}_j} - \tilde{\mathbf{h}_i}) \leq 0,\ 1 < j < K \\ \tilde{w}_2 > 0 \end{aligned} \tag{5.30}$$

So that any solution can be mapped to the desired form by Eqn. (5.29). We have now reformulated Och's line optimisation as a feasibility problem. As a consequence of this reformulation we are freed from the requirement of using the SweepLine algorithm and could use other techniques, such as linear programming, that function in multiple dimensions.

5.4.3 Optimal Search over Many Directions using a Projected Polytope

The reason for representing line optimisation as a projection is that we can generalise the method to M directions $\{\mathbf{d}_1 \dots \mathbf{d}_M\}$ where $1 \leq M \leq D$. We redefine $\mathscr{W}$ to be any affine subspace within $(\mathbb{R}^D)^*$ with respect to a vector of M free variables $\boldsymbol{\gamma} = [\gamma_1 \cdots \gamma_M]$

$$\mathscr{W} = \{\mathbf{w}^{(0)} + \boldsymbol{\gamma}\Delta\} \tag{5.31}$$

where Δ is a $D \times M$ matrix of direction vectors

$$\Delta = \begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_M \end{bmatrix} \tag{5.32}$$

The projection matrix $A_{M+1,D}$ is now:

$$A_{M+1,D} = \begin{bmatrix} \Delta \\ \mathbf{w}^{(0)} \end{bmatrix} \tag{5.33}$$

The projection of a feature vector is performed with the operation $\tilde{\mathbf{h}} = A_{M+1,D}\mathbf{h}$. The constraints necessary for both $\mathbf{h}_i$ and $\tilde{\mathbf{h}}_i$ to be satisfied for extremity can also be rewritten in terms of M

$$\begin{aligned} \tilde{\mathbf{w}}(\tilde{\mathbf{h}_j} - \tilde{\mathbf{h}_i}) \leq 0,\ 1 < j < K \\ \tilde{w}_{M+1} > 0 \end{aligned} \tag{5.34}$$

The major advantage of this formulation is that, for any dimension M, we can now use a linear programming or a vertex enumeration algorithm to find the normal cone associated with the lowest error. The mechanics of this method are similar to unprojected LP-MERT case, except we reject all cones where the representative parameter vectors have a negative $M+1$ component.

Although we can test for vertices in projected parameter space, we note that some prop-

erties of the resulting normal cones may be distorted. One of these distorted properties is Euclidian distance, which follows the relationship

$$\begin{aligned} ||\tilde{\mathbf{w}}|| &= \sqrt{(\gamma_1 \ \dots \ \gamma_M \ 1)(\gamma_1 \ \dots \ \gamma_M \ 1)^T} \\ &= \sqrt{\gamma_1^2 + \ldots + \gamma_M^2 + 1} \end{aligned} \tag{5.35}$$

We can see that this is different to the norm of the underlying parameter because it does not take into account $\mathbf{w}^{(0)}$ or the magnitude of the direction vectors $\{\mathbf{d}_1, \ldots, \mathbf{d}_M\}$

$$\begin{aligned} ||\mathbf{w}|| &= \sqrt{(\mathbf{w}^{(0)} + \gamma_1 \mathbf{d}_1 + \ldots + \gamma_M \mathbf{d}_M)(\mathbf{w}^{(0)} + \gamma_1 \mathbf{d}_1 + \ldots + \gamma_M \mathbf{d}_M)^T} \\ &= \sqrt{(w_1^{(0)} + \gamma_1 d_{1,1} + \ldots + \gamma_M d_{M,1})^2 + \ldots + (w_D^{(0)} + \gamma_1 d_{1,D} + \ldots + \gamma_M d_{M,D})^2} \end{aligned} \tag{5.36}$$

Note that projected MERT never uses parameters from the projected space in the final linear model. The parameters are always transformed back into higher dimensional parameters. The only utility of the projected space is for the identification of redundant feature vectors.

Algorithm 5.1 The projected MERT algorithm

Require: S polytopes $H[s]$, initial parameter $\mathbf{w}^{(0)}$, Direction generator generate(), Error function $E(\mathbf{w})$, Convergence criterion ε.

$n \leftarrow 1$ ▷ Iteration counter
repeat
  $\mathbf{w}^{(n)} = \mathbf{w}^{(n-1)}$
  $\Delta =$ generate()
  $A_{M+1,D} = \left[\begin{smallmatrix} \Delta \\ \mathbf{w}^{(n)} \end{smallmatrix}\right]$
  for $s = 1$ to S **do**
    $\tilde{H}[s] = A_{M+1,D} H[s]$
  end for
  $M = \mathcal{N}(\tilde{H}[1] + \cdots + \tilde{H}[S])$ ▷ Normal fan of the Minkowski sum
  for all feasible **i do**
    $\tilde{\mathbf{w}} = M[\mathbf{i}]$
    $\mathbf{w} = \tilde{\mathbf{w}} A_{M+1,D}$
    if $\tilde{w}_{M+1} > 0$ **and** $E(\mathbf{w}) > E(\mathbf{w}^{(n)})$ **then**
      $\mathbf{w}^{(n)} = \mathbf{w}$
    end if
  end for
  $n \leftarrow n + 1$
until $E(\mathbf{w}^{(n)}) - E(\mathbf{w}^{(n-1)}) < \varepsilon$
print $\mathbf{w}^{(n)}$

We describe the full projected MERT algorithm in Algorithm 5.1. This algorithm is described for the full multi-sentence case, where we have S polytopes. There are a number of support functions needed for the algorithm. The error function $E(\mathbf{w})$ is the same error function used in Eqn. (3.9). The generate() function generates M directions, which could be a subset of the feature axes, random directions, gradient based schemes, or a combination of all three. See Section 3.2.2 for a description of possible schemes to select directions.

Because the algorithm computes the multi-sentence case we reintroduce the index vector $\mathbf{i}$ notation from Section 3.2.4. We note that Line 9 refers to as yet unspecified Minkowski sum operation. The result of the Minkowski sum is an associative array, indexed by $\mathbf{i}$, of a representative parameter vector for each feasible index vector. I.e., for a particular $\mathbf{i}$ the parameter $\mathbf{w} = M[\mathbf{i}]$ satisfies the constraints in (3.12) for $\mathbf{i}$. Note that the Minkowski sum operation also implicitly identifies the set of index vectors that are feasible. Chapter 6 defines the Minkowski sum, describes an algorithm for computing it, and gives a thorough complexity analysis. Section 6.4 describes the computation of a single iteration of the projected MERT algorithm for a Russion-to-English SMT system.

5.4.4 An Example of Och's Line Optimisation using a Projected Polytope

We now illustrate an example of polytope projection using the example from Cer et al. [29] described in Section 5.2.5. Cer et al. [29] describe an application of the SweepLine algorithm on their two dimensional set of feature vectors using an initial point of $\mathbf{w}^{(0)} = [0.5 \quad 1]$ and a direction $\mathbf{d} = [1 \quad 0]$. Our goal is to demonstrate how the interval boundaries of the upper envelope can be computed without using the SweepLine algorithm described in Section 3.2.1.

First, we must create projected feature vectors using the given initial point and direction vectors. These can be computed using the matrix:

$$A_{2,D} = \begin{bmatrix} 1 & 0 \\ 0.5 & 1 \end{bmatrix} \tag{5.37}$$

For the purposes of this example we are projecting a 2-dimensional space to another 2-dimensional space. In practice, we would wish to project from spaces of a higher dimension to spaces of a lower dimension. The projected feature vectors for the first feature vector is

	$h_{LM} : \log(P_{LM}(\mathbf{e}))$	$h_{TM} : \log(P_{TM}(\mathbf{f}\|\mathbf{e}))$	$\tilde{h}_1 : b(\mathbf{e},\mathbf{f})$	$\tilde{h}_2 : a(\mathbf{e},\mathbf{f})$
$\mathbf{e}_1$	-0.1	-1.2	-0.1	-1.25
$\mathbf{e}_2$	-1.2	-0.2	-1.2	-0.8
$\mathbf{e}_3$	-0.9	-1.6	-0.9	-2.05
$\mathbf{e}_4$	-0.9	-0.1	-0.9	-0.55

Table 5.2 The set of hypotheses from Table 5.1 transformed by direction $\mathbf{d} = [1 \quad 0]$ and initial parameter $\mathbf{w}^{(0)} = [0.5 \quad 1]$ (after Cer et al. [29], Table 2). The first pair of columns are the original features from Table 5.1, and the second pair are the projected features.

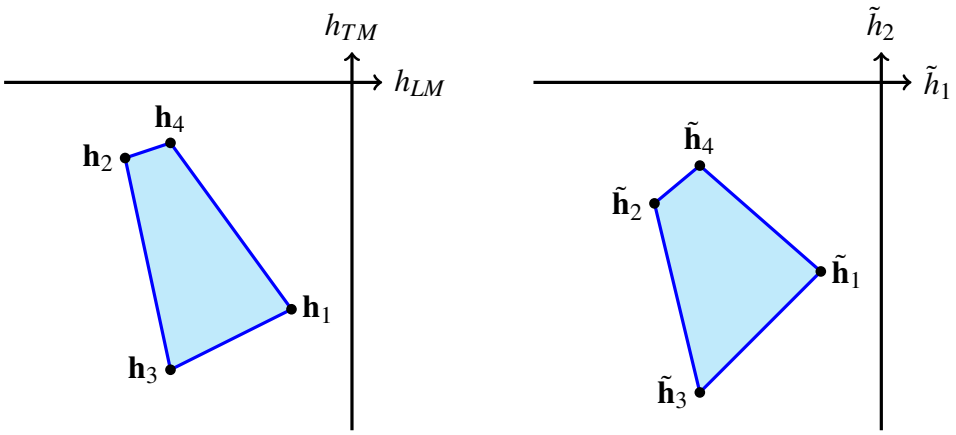


Fig. 5.3 The polytope in the left part is a replication of Figure 5.1, and the polytope in the right part is constructed from the projected feature vectors in Table 5.2.

computed as follows:

$$\tilde{\mathbf{h}}_1 = \begin{bmatrix} b(\mathbf{e}_1,\mathbf{f}) \\ a(\mathbf{e}_1,\mathbf{f}) \end{bmatrix} = A_{2,D}\mathbf{h}_1 = \begin{bmatrix} 1 & 0 \\ 0.5 & 1 \end{bmatrix} \begin{bmatrix} -0.1 \\ -1.2 \end{bmatrix} = \begin{bmatrix} -0.1 \\ -1.25 \end{bmatrix} \tag{5.38}$$

where the functions $b(\mathbf{e},\mathbf{f})$ and $a(\mathbf{e},\mathbf{f})$ are the gradient and y-intercepts of the linear function $\ell_{\mathbf{e}}$ as described in Section 3.2.1. The projected feature functions are shown in Table 5.2 and the associated polytope is shown in Figure 5.3. Note that we do not show the redundant feature vector $\mathbf{h}_5$ from the previous example, because it is also redundant in projected space.

The normal fan of the projected feature functions $\mathcal{N}(\tilde{H})$ is shown in the lower part of Figure 5.4. This projected fan resembles the normal fan in Figure 5.2 but with the decision boundaries demarking different regions of parameter space. Now consider the line drawn at $w_2 = 1$, the intersection of respective normal cones with this line is the set of projected

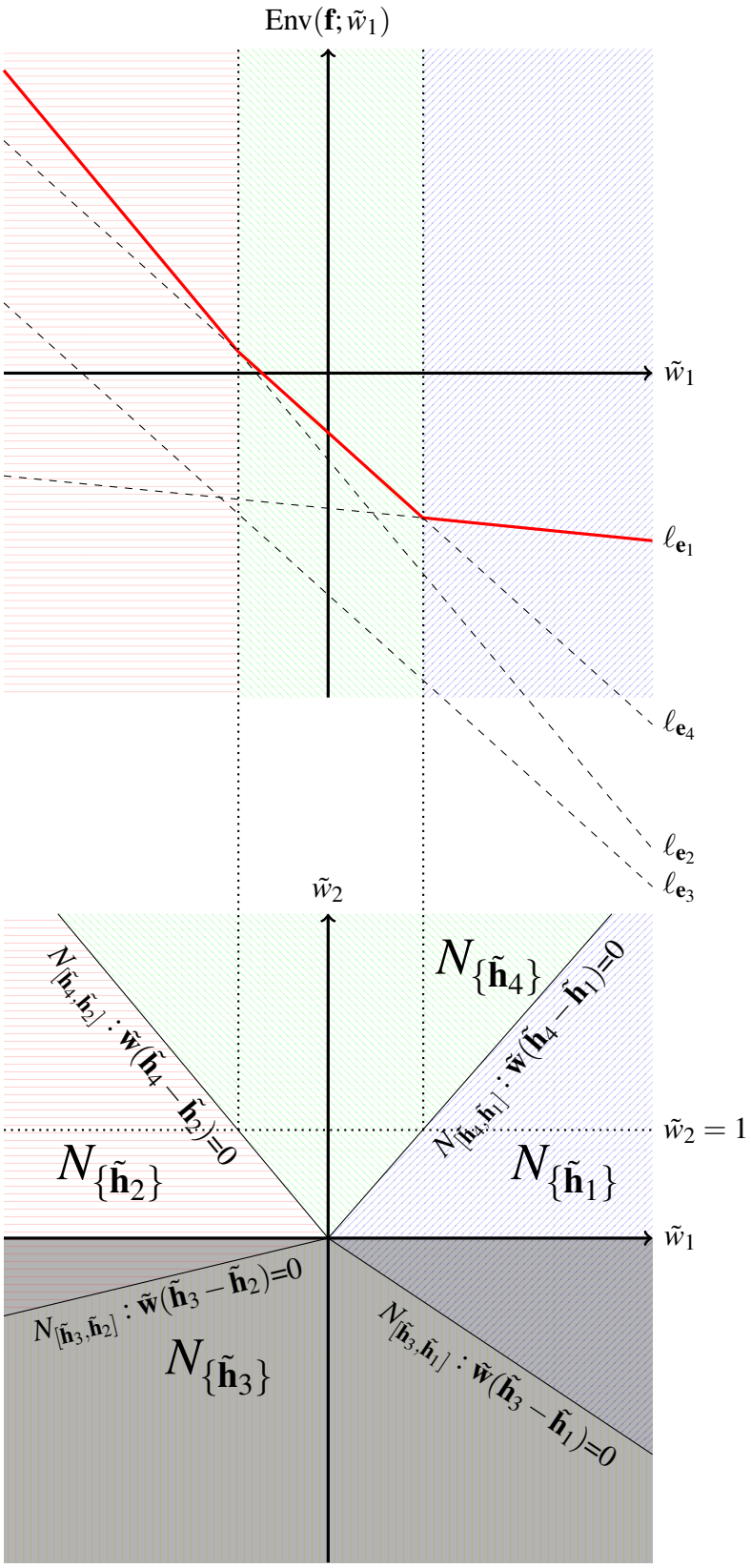


Fig. 5.4 The upper part of the Figure shows the application of Och's Line Optimisation using the same initial parameter and direction used in Table 5.2 (after Cer et al. [29], Figure 2). The lower part of the Figure shows the equivalent normal fan for the projected polytope in Figure 5.3. The darkened region represents infeasible parameters. This diagram illustrates that the line through the normal fan at $\tilde{w}_2 = 1$ yields the solution found by Och's line optimisation.

parameters that satisfy the constraints in (5.27). In the upper part of the figure we have reproduced Figure 2 from Cer et al. [29] that shows the application of Och's line optimisation with the upper envelope highlighted in the thick red line. We have drawn lines between the intersections of the decision boundaries of the normal fan with the interval boundaries in the upper envelope, to show their equivalence.

Recall from the constraints in (5.30) that any parameter in the projected normal fan with a negative $\tilde{w}_2$ component cannot be mapped back to a parameter in the original space, and are infeasible. We represent this region in the lower part of Figure 5.4 by shading below the line $\tilde{w}_2 = 0$. Note how all the parameters that maximise $\tilde{\mathbf{h}}_3$ are subject to $\tilde{w}_2 \leq 0$ and are entirely in this region. Thus there is no line segment is the upper envelope for the hypothesis $\mathbf{e}_3$.

5.5 Regularisation and the Normal Fan

A common problem that is encountered when training parametric statistical models is that of *overfitting*, where the training data is modelled exactly but we see very poor performance on a test set. Usually this is due to noise or outliers in the training data, and a failure of the model's training scheme to recognise this noisy data. We wish to use models and training schemes that *generalise*, which means that they are not affected by noisy data.

In Section 6.3.3 we discuss how a large feature dimension is a possible cause of overfitting, and that using models with lower feature dimensions is one method of guaranteeing generalisation. The problem with models of a lower feature dimension is that they may not be able to model the training data effectively, which also results in lower test set performance. One technique for preventing overfitting without reducing feature dimension is regularisation, where a penalty is applied to the objective function to discourage certain values of $\mathbf{w}$.

In this section we concern ourselves specifically with ℓ_2 regularisation, in which the penalty takes the form $\|\mathbf{w}\|^2$. One good example of ℓ_2 regularisation is the regularisation applied to polynomial curve fitting, see Page 10 of Bishop et al. [20] for details.

Unfortunately, this form of regularisation cannot be directly applied to MERT. Let us assume we have found an optimal parameter $\hat{\mathbf{w}}$ that satisfies the constraints in (5.2). If we apply a scale factor $t > 0$, then all parameters of the form $t\hat{\mathbf{w}}$ also satisfy the constraints in (5.2). The optimiser can therefore reduce the penalty by driving t down towards 0, yielding a very small parameter vector.

Galley et al. [63] describe methods of regularising Och's line optimisation. Several

methods of regularisation are discussed but we focus on their adapted form of ℓ_2 regularisation because it provides the highest gain in their experiments. Our aim is to interpret this form of regularisation with respect to the normal fan. The first step is to define a function $\mathbf{r}(\gamma)$ that can produce a vector suitable for regularisation. Let us define the objective function to be minimised as (after Galley et al. [63] Eqn. 5)

$$\hat{\gamma} = \underset{\gamma}{\operatorname{argmin}} \left\{ E(\gamma) + \frac{\|\mathbf{r}(\gamma)\|^2}{2\sigma^2} \right\} \tag{5.39}$$

where $E(\gamma)$ is the form of error function in Eqn. (3.9) used in line optimisation as described in Section 3.2.1. We focus on a single form of $\mathbf{r}(\gamma)$, again justified by the experimental results of Galley et al. [63]

$$\mathbf{r}(\gamma) = \gamma \mathbf{d} \tag{5.40}$$

where $\mathbf{d}$ is the direction vector used is Och's line optimisation. Essentially, this form of regularisation penalises parameters vectors that are far from the initial parameter $\mathbf{w}^{(0)}$. We can constrain the function, with no loss of generality, such that $\|\mathbf{d}\| = 1$. The objective in Eqn. (5.39) can now be written as

$$\hat{\gamma} = \underset{\gamma}{\operatorname{argmin}} \left\{ E(\gamma) + \frac{\|\gamma\|^2}{2\sigma^2} \right\} \tag{5.41}$$

Note that the regularisation term reaches its minimum when $\gamma = 0$ yielding $\mathbf{w} = \mathbf{w}^{(0)}$. This regularisation encourages $\hat{\mathbf{w}}$ to be placed on the decision boundary closest to $\mathbf{w}^{(0)}$, or if the decision region includes $\mathbf{w}^{(0)}$ to select $\mathbf{w}^{(0)}$ itself. To break ties, an ε is added to $\hat{\mathbf{w}}$ to keep it away from the decision boundary.

Now, let us interpret this form of regularisation with respect to the projected normal fan. We again use the example of Cer et al. [29] to illustrate our description. First, we note that the initial parameter can be expressed in terms of the projection matrix as $\mathbf{w}^{(0)} = [0\ 1]A_{2,D}$. Let us redraw the lower part of Figure 5.4 in Figure 5.5 with the feature vector $\tilde{\mathbf{w}}^{(0)} = [0\ 1]$ marked.

We also mark two further points $\tilde{\mathbf{w}}^{(1)} = [\hat{\gamma}_1\ 1]$ and $\tilde{\mathbf{w}}^{(2)} = [\hat{\gamma}_2\ 1]$ where $\hat{\gamma}_1$ is the optimal value for the objective in Eqn. (5.41) if $\mathbf{e}_1$ has the lowest error, and $\hat{\gamma}_2$ is optimal value if $\mathbf{e}_2$ has the lowest error. Note that both of these points are of a similar distance to $\tilde{\mathbf{w}}^{(0)}$. Thus, the optimiser would choose the feature vector associated with the lowest error. If both hypotheses have similar errors, then the optimiser would struggle to find the best parameter.

Let us now consider what happens in the unprojected space. We redraw Figure 5.2 in

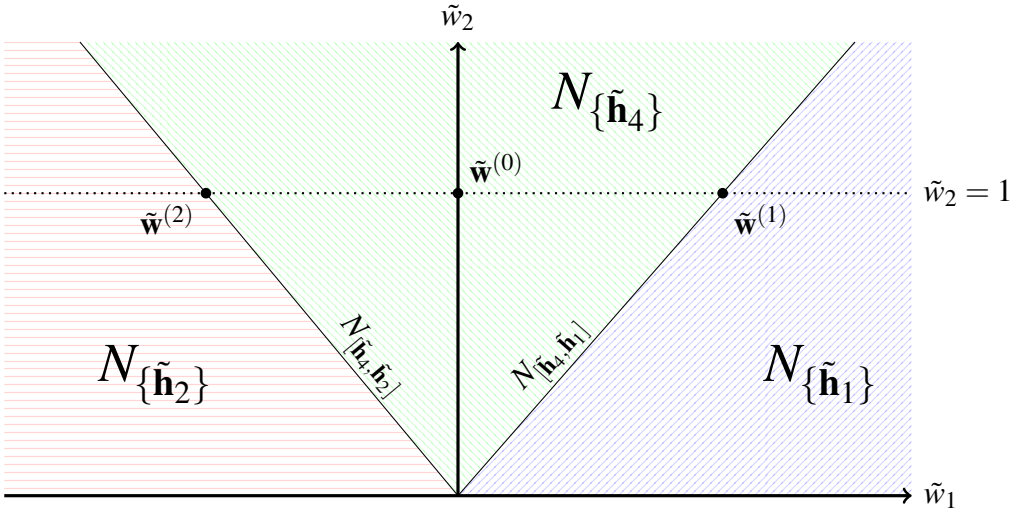


Fig. 5.5 We redraw Figure 5.4 with potential optimal parameters under the ℓ_2 regularisation scheme of Galley et al. [63] marked. The parameter $\tilde{\mathbf{w}}^{(0)}$ would be picked if $\mathbf{e}_4$ has the lowest error, and $\tilde{\mathbf{w}}^{(1)}$ and $\tilde{\mathbf{w}}^{(2)}$ represent the optimal parameters if $\mathbf{e}_1$ or $\mathbf{e}_2$ have the lowest error respectively

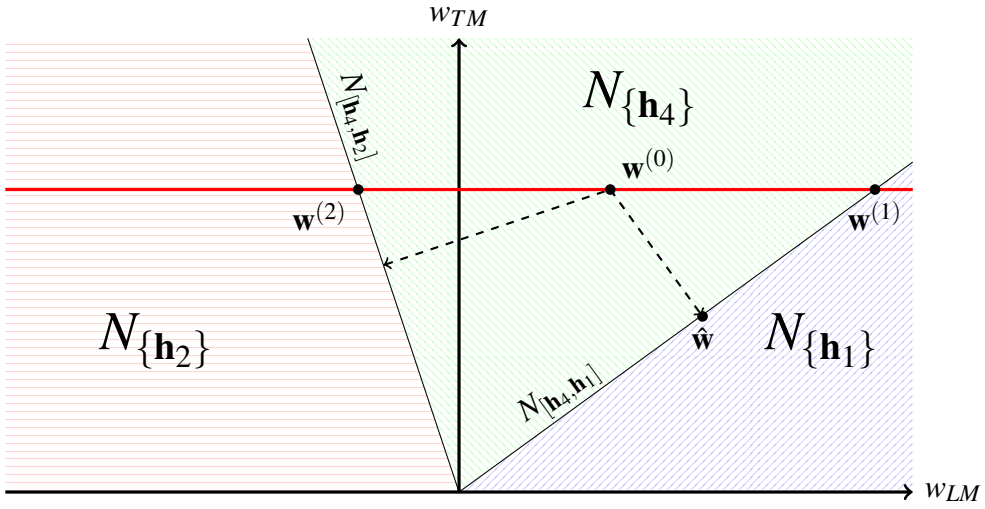


Fig. 5.6 We redraw Figure 5.2 with potential optimal parameters under the ℓ_2 regularisation scheme of Galley et al. [63] marked. The thick red line is the subspace of $(\mathbb{R}^2)^*$ optimised. The parameter $\mathbf{w}^{(0)}$ would be picked if $\mathbf{e}_4$ has the lowest error, and $\mathbf{w}^{(1)}$ and $\mathbf{w}^{(2)}$ represent the optimal parameters if $\mathbf{e}_1$ or $\mathbf{e}_2$ have the lowest error respectively. The dashed lines mark the distances between the decision boundaries and $\mathbf{w}^{(0)}$. The parameter $\hat{\mathbf{w}}$ is the optimal point under our proposed modification of the regulation scheme

Figure 5.6 and draw in thick red the line used in the line optimisation. We reverse the projection operation such that $\mathbf{w}^{(1)} = \tilde{\mathbf{w}}^{(1)} A_{2,D}$, and $\mathbf{w}^{(2)} = \tilde{\mathbf{w}}^{(2)} A_{2,D}$. Again, we can see in this diagram that both these points are equidistant to the $\mathbf{w}^{(0)}$. The results are unchanged from the projected space in Figure 5.5.

We now break out of the affine subspace indicated by the thick red line, and mark the distances between $\mathbf{w}^{(0)}$ and the decision boundary with dashed lines. From inspection of the diagram we can see that the decision boundary between $\mathbf{e}_4$ and $\mathbf{e}_1$ is significantly closer than the decision boundary between $\mathbf{e}_4$ and $\mathbf{e}_2$. If $\mathbf{e}_2$ and $\mathbf{e}_1$ have similar error counts, then we would wish to pick the optimal parameter $\hat{\mathbf{w}}$ from $N_{\{\mathbf{h}_1\}}$ shown in Figure 5.6.

We present this example as an argument that regularisation should only be performed in the original unprojected space. Using a distance γ from Och's line optimisation is misleading because it is restricted to a projected affine subspace of the original unprojected space. We need to be careful about distinguishing which properties of the normal fan are invariant under the projection implied by Och's line optimisation and which are not. If a feature vector is found to be a vertex, then this is a property that is invariant. The feature vector is also a vertex in the original space, hence the use of Och's line search as redundancy test. Distances, however, are not invariant under a projection and should not be used for regularisation.

The application of regularisation for MERT and ℓ_2 regularisation seems similar. Both schemes attempt to minimise Euclidian distances, but the affect of projection on distances, as shown in Eqn. (5.35) and Eqn. (5.36), means that the results of regularisation with MERT are not always well defined. We therefore propose a modification to regularisation scheme of Galley et al. [63] that reflects the distances in the original space.

1. Compute the line search as normal, with the result of a set of line segments given as a sequence of γ values of the form $(\gamma_1, \ldots, \gamma_n, \ldots, \gamma_N)$. Each γ is associated with a feature vector $\mathbf{h}_n$.

2. For each γ_n compute the distance between the initial parameter $\mathbf{w}^{(0)}$ and the decision boundary $\delta_n = \mathbf{w}^{(0)}(\mathbf{h}_n - \mathbf{h}_{n-1})$.

3. For each interval, add the regularisation term $\frac{\|\delta_n\|^2}{2\sigma}$ to the error $E(\gamma_n)$.

4. If the optimal interval contains $\mathbf{w}^{(0)}$ then $\hat{\mathbf{w}} = \mathbf{w}^{(0)}$, otherwise set $\hat{\mathbf{w}}$ to the feature vector in $N_{\{\mathbf{h}_n\}}$ closest to $\mathbf{w}^{(0)}$ with the addition of an ε to break ties.

The extra computations of steps 2 and 4 can be performed in linear time with respect to the dimension D.

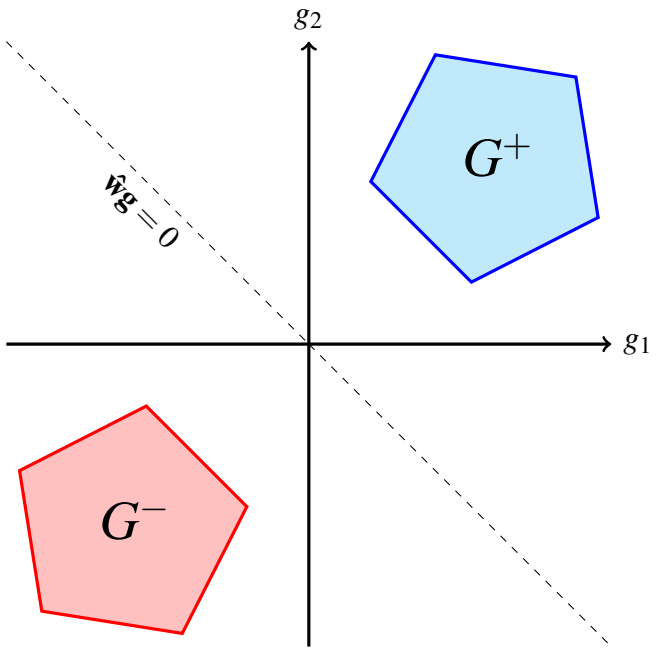


Fig. 5.7 A geometric representation of PRO to illustrate the symmetry in the ranking problem. The G^+ and G^- polytopes include the set of positively labelled and negatively labelled $\mathbf{g}$ vectors. The dashed line represents the decision boundary given by the optimal parameter vector $\hat{\mathbf{w}} \in (\mathbb{R}^D)^*$.

5.6 A Geometric Description of Ranking Methods

Recall from Section 3.4 that in PRO the ranking of a K-best list can by described by the pairwise comparison of feature vectors. These pairwise comparisons form the set of constraints in 3.22. Each of these constraints correspond to an edge in the polytope H.

Now recall from Section 5.2.2 that the edge of two vertices $\mathbf{h}_i$ and $\mathbf{h}_j$ in the polytope H form is a line segment $[\mathbf{h}_i, \mathbf{h}_j]$. Note that this line segment is undirected. By Corollary 5.4 we know the edge defines a decision boundary in $(\mathbb{R}^D)^*$ such that any parameter on the boundary yields equal model scores for any feature vector $\mathbf{h} \in [\mathbf{h}_i, \mathbf{h}_j]$.

The decision boundary is embedded in a hyperplane in $(\mathbb{R}^D)^*$ given by the equation $\mathbf{gw} = 0$ where $\mathbf{g} \in \mathbb{R}^D$. We have two choices for the value of the vector $\mathbf{g}$: either $\mathbf{h}_j - \mathbf{h}_i$ or $\mathbf{h}_i - \mathbf{h}_j$. Both of these vectors are parallel to line segment $[\mathbf{h}_i, \mathbf{h}_j]$ but point in opposite directions. Let us denote both vectors as $\mathbf{g}_{j,i} = \mathbf{h}_j - \mathbf{h}_i$ and $\mathbf{g}_{i,j} = \mathbf{h}_i - \mathbf{h}_j$.

The choice of vector determines which hypothesis has a higher model score in each half space. For example, all parameters that satisfy $\mathbf{wg}_{j,i} < 0$ imply that the model scores obey $\mathbf{wh}_i > \mathbf{wh}_j$. Following Hopkins and May [76] the vectors are labelled with two classes

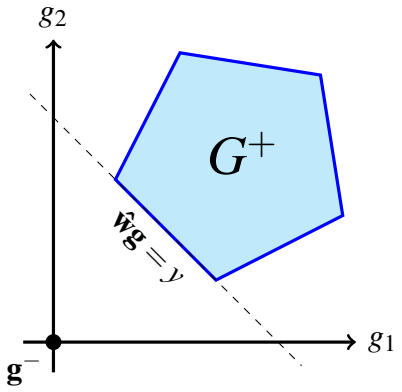


Fig. 5.8 An illustration of how the method of Schölkopf et al. [130] can be used to solve the problem in Figure 5.7. The dashed line represents the decision boundary given by $\hat{\mathbf{w}}$ and the offset y. This decision boundary defines a face of the polytope G^+. The origin is labelled as the $\mathbf{g}^-$ vector, and is used to ensure the symmetry of the solution

depending on the relative error rate of the hypotheses. We denote these two classes as positive $+$ and negative $-$. If the error rate of the hypothesis $\mathbf{e}_i$ is lower than the hypothesis $\mathbf{e}_j$, then then the positively labelled vector is $\mathbf{g}_{j,i}$. We can use these two labels to generate two polytopes: a G^+ polytope of $\mathbf{g}$ vectors with positive labels, and a G^- polytope of negatively labelled $\mathbf{g}$ vectors.

There is an implicit symmetry in this geometric representation. For each vector $\mathbf{g} \in G^+$ there exists a second vector $-\mathbf{g} \in G^-$. We illustrate this symmetry in Figure 5.7 with an example showing two symmetric polytopes. In between the polytopes we have drawn a dashed decision boundary, which is normal to an optimal parameter vector $\hat{\mathbf{w}}$. From inspection of this example we can see that the decision boundary satisfies both the constraints

$$\hat{\mathbf{w}}\mathbf{g} < 0 \;\text{ for all } \mathbf{g} \in G^+ \tag{5.42}$$

and the following

$$\hat{\mathbf{w}}\mathbf{g} > 0 \;\text{ for all } \mathbf{g} \in G^- \tag{5.43}$$

Not only are both these constraints satisfied but the magnitude of the distances of symmetric vectors from the decision boundary are equal, such that $\hat{\mathbf{w}}\mathbf{g} = -\hat{\mathbf{w}}(-\mathbf{g})$, where $\mathbf{g} \in G^+$ and $-\mathbf{g} \in G^-$. This relationship is true for any parameter vector $\mathbf{w} \in (\mathbb{R}^D)^*$ because they all result in a decision boundary that includes the origin. This symmetry is noted by Hopkins and May [76] who use the complement of a $\mathbf{g}$ vector when building their sample set to 'ensure balance' in their binary class problem.

Because of this symmetry in the ranking problem Green et al. [71] have observed that pairwise ranking can be modelled as a unary class separation problem. One form of parameter estimation in a unary class problem is the method of Schölkopf et al. [130], which elegantly estimates a parameter vector $\hat{\mathbf{w}}$ for a binary class SVM using only the G^+ polytope. Let us constrain the solution such that $\hat{\mathbf{w}}$, along with some $y \in \mathbb{R}$, yield a face in G^+ as defined by Eqn. 5.9. The $\hat{\mathbf{w}}$ parameter and y offset are found using a binary class SVM under this constraint and with the polytope G^+ as the set of positive examples. This trick behind this method is to also include the origin as a negatively labelled $\mathbf{g}^-$ vector, which then ensures the required symmetry.

We illustrate the method of Schölkopf et al. [130] in Figure 5.8. The resulting hyperplane satisfies the constraint that is encloses a face of G^+ and also maximises the margin for all $\mathbf{g} \in G^+$ and $\mathbf{g}^-$. After $\hat{\mathbf{w}}$ and y have been estimated, we can discard y. The resulting $\hat{\mathbf{w}}$ is identical to the $\hat{\mathbf{w}}$ used in Figure 5.7.

Chapter 6

Training Set Geometry

In the previous chapter we built upon the formulation of LP-MERT [62] by using the polytope and the normal fan to describe MERT for a single sentence. Using this description we were able to provide a novel generalisation of Och's Line Optimisation [113] to multiple dimensions by using a projected polytope. In this chapter we continue to make novel contributions to SMT using convex geometry.

The main work of this chapter is to extend the use of polytopes to a training set of many sentences. To find a parameter vector that will generalise we need to consider the error across a large training set. Our goal is to find a parameter set that lowers the error across a training set of S sentences.

Because we now have multiple source sentences we return to the notation used in Section 3.2.4. Recall that we denote the S-dimensional vector $\mathbf{i}$ as the index vector. Each element is an index i to a hypothesis in the K-best list for the sth sentence. Let us consider the set of $K \times S$ constraints necessary for a parameter $\mathbf{w}$ to yield the hypotheses as selected by the index vector $\mathbf{i}$

$$\mathbf{w}(\mathbf{h}_{s,j} - \mathbf{h}_{s,i_s}) \leq 0 \text{ for } 1 \leq j \leq K, 1 \leq s \leq S \tag{6.1}$$

Consider the polytope H_s constructed from the feature vectors of the K-best list of the sth sentence. From our previous discussion of the normal fan, we know that the set of parameters that satisfy the constraints for each individual sentence take the form of a normal cone $N_{\{\mathbf{h}_{s,i_s}\}} \in \mathscr{N}(H_s)$. The set of parameters that agree with all the constraints in (6.1) are

$$\bigcap_{s=1}^{S} N_{\{\mathbf{h}_{s,i_s}\}} \tag{6.2}$$

If this parameter set is equal to the zero-vector set $\{\mathbf{0}\}$ then we label the index vector as

infeasible and discard it. For a single sentence training set, if a feature vector is found to be infeasible then we continue testing feature vectors in the K-best. The next feature vector is chosen in the order of the error associated with its hypothesis. However, for the training set of many sentences we encounter problems with this approach. Because there are K^S possible values of $\mathbf{i}$ we cannot even compute the error associated with every index vector, let alone test them for feasibility.

Recall from Section 3.2.4 that Galley and Quirk [62] do not explicitly search for a feasible set of constraints. Instead they create a linear combination of all features associated with hypotheses identified by $\mathbf{i}$. With the index vector notation we can write this sum in terms of $\mathbf{i}$

$$\mathbf{h}_\mathbf{i} = \sum_{s=1}^{S} \mathbf{h}_{s,i_s} \tag{6.3}$$

This operation gives a set of K^S summed feature vectors $\{\mathbf{h}_\mathbf{i} : \mathbf{i} \in I\}$. This finite set of vectors can be used to define a polytope $\text{conv}(\{\mathbf{h}_\mathbf{i} : \mathbf{i} \in I\})$. Galley and Quirk [62] do not make the connection that the polytope $\text{conv}(\{\mathbf{h}_\mathbf{i} : \mathbf{i} \in I\})$ is the Minkowski sum of all the sentence level polytopes. Making this connection is valuable because convex geometry describes the relation between the Minkowksi sum and the parameters defined in Eqn. (6.2).

This chapter applies convex geometry to the problem of finding a feasible parameter that yields a low error hypothesis set for the training data. We start by considering the intersection of normal fans, an operation called the common refinement. We follow a proof of Gritzmann and Sturmfels [72] to show that the normal fan of the Minkowski sum of polytopes is equivalent to the common refinement of the normal fans of the polytopes. This result formalises the relationship described by Galley and Quirk [62].

The exhaustive version of the algorithm of Galley and Quirk [62] for computing the complete Minkowski sum has a complexity of $O(K^S)$. Remarkably, there already exists an algorithm by Fukuda [59] that can compute the Minkowski sum in polynomial-time with respect to the number of *feasible* index vectors. The reason this algorithm is so much faster is that it exploits the geometry of the normal fan. Testing the feasibility of a normal cone only requires the constraints implied by edges in the polytope. The full fan does not need to be known. We give a full description of this algorithm in the second section.

The reader may note that we have given the complexity in terms of an unknown quantity: the number of feasible index vectors. In the third section we consider the study of Minkowski sums by Gritzmann and Sturmfels [72] that explores the size of this set. In short the number of feasible index vectors has the upper bound $O(S^{D-1}K^{2(D-1)})$. So, instead of K^S, we have $S^{D-1}K^{2(D-1)}$, indicating that the exponential complexity is in the dimension-

ality of the feature vector rather than the size of the training data.

In the fourth section we use projected MERT and the Minkowski sum algorithm to compute the 3-dimensional error surface for an SMT system. We use an implementation of the Minkowski sum algorithm [153] and discuss how it can be improved for SMT tasks.

Finally we continue discussion of linear models in Section 3.6, in which we discussed the suitability of linear models for high dimensional problems. We present the upper bounds theorems as an argument for why linear models are of limited use for systems of many features.

6.1 The Minkowski Sum

Consider the sth and the tth sentences of the training set. From these sentences two K-best lists are generated. The hypotheses in each of the K-best lists are used to define two sets of feature vectors. Recall from Chapter 5 that these two polytopes H_s and H_t can be defined by taking the convex hulls of their feature vectors.

For the two polytopes H_s and H_t, the Minkowski sum is defined as [72, 162]

$$H_s + H_t := \{\mathbf{h} + \mathbf{h}' : \mathbf{h} \in H_s, \mathbf{h}' \in H_t\} \tag{6.4}$$

H_s and H_t are called summands of $H_s + H_t$. The Minkowski sum is commutative and associative and generalises naturally to more than two polytopes [72].

Recall from Chapter 5 that the set of all normal cones of a polytope is called the normal fan. To understand why the Minkowski sum is relevant a second operation called the common refinement has to be defined. The common refinement of two normal fans $\mathcal{N}(H_s)$ and $\mathcal{N}(H_s)$ is [162]

$$\mathcal{N}(H_s) \wedge \mathcal{N}(H_t) := \{N \cap N' : N \in \mathcal{N}(H_s), N' \in \mathcal{N}(H_t)\} \tag{6.5}$$

Each normal cone in the normal fan is the set of parameters that define a face in the associated polytope. The common refinement defines a new set of cones. Each cone is associated with two faces, one from H_s and one from H_t. The parameters in the cone will yield both those faces when substituted into Eqn. 5.9 with the respective polytopes.

The common refinement contains cones that are equal to the sets in (6.2). For a training set of S sentences with S normal fans, the common refinement contains a cone for each feasible index vector $\mathbf{i}$. Each parameter vector in a cone satisfies the constraints in (6.1) for the cone's associated index vector. The common refinement is thus the desired geometric

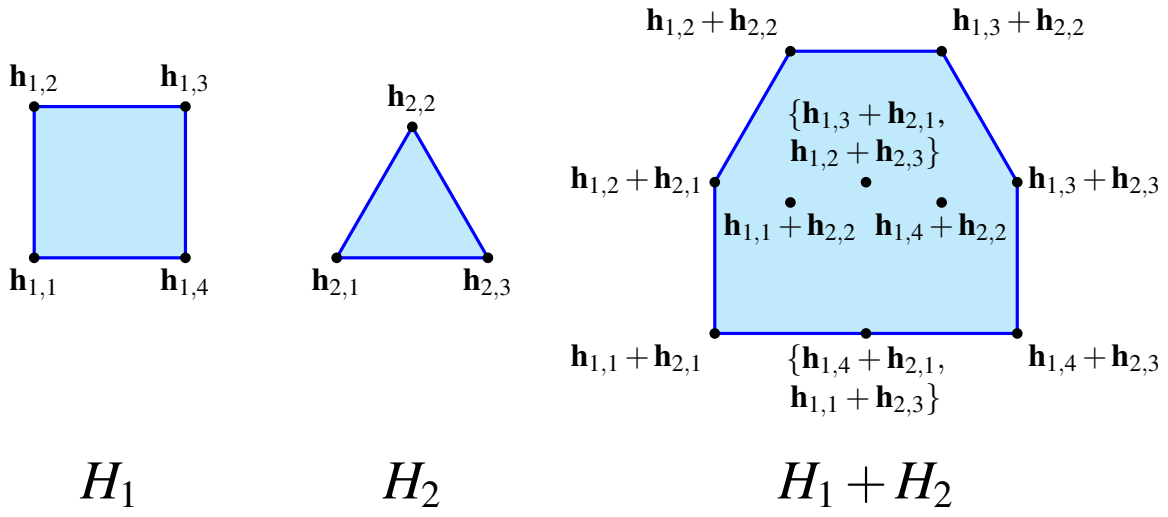


Fig. 6.1 An illustration of the Minkowski sum for two input polytopes. H_1 is the square polytope on the left, H_2 is the triangular polytope in the middle, and the sum polytope $H_1 + H_2$ is the on the right. There are 12 resultant feature vectors but only 10 dots because some feature vectors in the sum polytope are not unique, for example $\mathbf{h}_{1,3}+\mathbf{h}_{2,1} = \mathbf{h}_{1,2}+\mathbf{h}_{2,3}$.

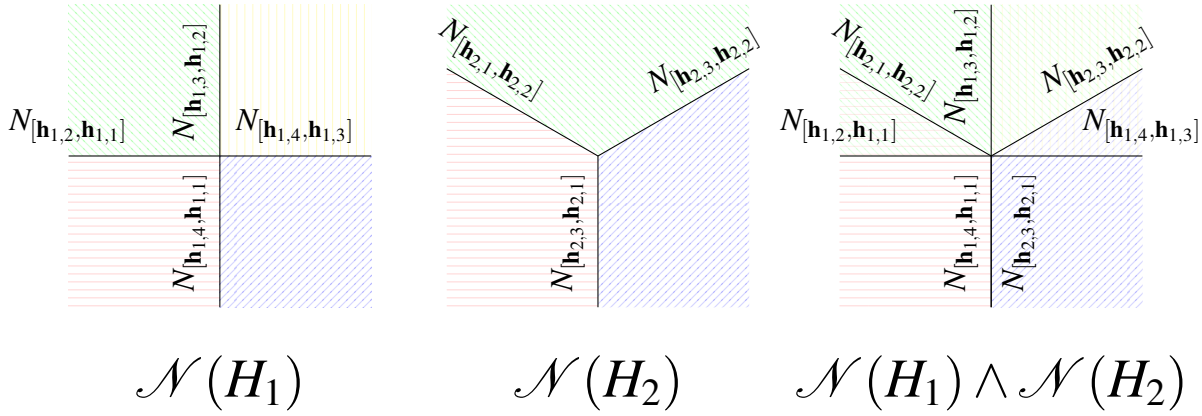


Fig. 6.2 The equivalent common refinement to the Minkowski sum in Figure 6.1. The labels $N_{[\mathbf{h}_{1,4},\mathbf{h}_{1,1}]}$ and $N_{[\mathbf{h}_{2,3},\mathbf{h}_{2,1}]}$ refer to the same decision boundary because the edges $[\mathbf{h}_{1,4},\mathbf{h}_{1,1}]$ and $[\mathbf{h}_{2,3},\mathbf{h}_{2,1}]$ are parallel.

object for multi-sentence MERT, and an optimal MERT algorithm should yield it.

If the common refinement is the required operation, then it may seem odd that the focus of this chapter is the Minkowski sum. The reason for this focus is because the normal fan of the Minkowski sum is equal to the common refinement of the normal fans of the summands, a relationship proved in Section 6.1.1

Proposition 6.1. $\mathcal{N}(H_s + H_t) = \mathcal{N}(H_s) \wedge \mathcal{N}(H_t)$

Before we start describing the characteterstics of these operations in detail, let us illustrate their application with an example. In Figure 6.1 we show an example of the Minkowski sum. There are two summands: H_1 is a square and H_2 a triangle. To compute the Minkowski sum polytope, we take the vertices of the summand polytopes and follow the definition in (6.4). We compute a total of $4 \times 3 = 12$ summed feature vectors, yet only half of these are vertices in the sum polytope $H_1 + H_2$.

Now consider the equivalent common refinement shown in Figure 6.2. In this example very little has to be computed. The common refinement appears as if one normal fan was superimposed on the other. We can see there are six decision boundaries associated with the six edges of the Minkowski sum. Even in this simple example, we can see that the common refinement is an easier quantity to compute than the Minkowski sum.

The convex geometry literature treats the Minkowski sum, the left hand side of Proposition 6.1, as the primary problem to solve. Because of this treatment the descriptions in this chapter are made with respect to the Minkowski sum. In terms of computation it is the common refinement, on the right hand side of Proposition 6.1, that is often easier to solve. In fact, the Minkowski sum algorithm of Fukuda [59] described in Section 6.2 actually computes the common refinement, and extracts the resulting Minkowski sum.

A major difference between our work and that of Galley and Quirk [62] is that they compute the Minkowski sum directly and we compute the common refinement. The advantage of their approach is that the same techniques for finding vertices in the single sentence case can be redeployed in the multi-sentence case. In our approach we have to introduce new terminology and concepts, such as the normal fan. However, once we are comfortable with these concepts the computation becomes much simpler. The result is also closer to the problem domain we wish to solve in SMT, namely identifying a good parameter to maximise a set of feature vectors.

6.1.1 Equivalence of the Minkowski Sum and the Common Refinement

We now describe a proof of Proposition 6.1 as given by Gritzmann and Sturmfels [72]. Recall that for a given $\mathbf{w}$ a face is defined as:

$$F = \{\mathbf{h} \in H : \mathbf{wh} = \max_{\mathbf{h}' \in H} \mathbf{wh}'\} \tag{6.6}$$

Much of the work to be done in this section considers a single face that can be determined by many parameters. To make the these relationships explicit a function $P(H;\mathbf{w})$ is introduced which determines a face given a parameter

$$P(H;\mathbf{w}) = \{\mathbf{h} \in H : \mathbf{wh} = \max_{\mathbf{h}' \in H} \mathbf{wh}'\} \tag{6.7}$$

so that $P(H;\mathbf{w}) = F$. Now consider polytopes associated with two sentences H_s and H_t for the sth and tth input sentences. First of all, we note that the linear nature of the model score using $\mathbf{w}$ allows us to make the following statement

$$\begin{aligned} \mathbf{wh}_s + \mathbf{wh}_t &= \mathbf{w}(\mathbf{h}_s + \mathbf{h}_t), \quad \forall \mathbf{h}_s \in P(H_s;\mathbf{w}) \text{ and } \forall \mathbf{h}_t \in P(H_t;\mathbf{w}) && (6.8) \\ &> \mathbf{w}(\mathbf{h}'_s + \mathbf{h}'_t), \quad \forall \mathbf{h}'_s \notin P(H_s;\mathbf{w}) \text{ and } \forall \mathbf{h}'_t \notin P(H_t;\mathbf{w}) && (6.9) \end{aligned}$$

This implies that the Minkowski sum of faces in the two summands is a face in the Minkwoski sum of the polytope

$$P(H_s + H_t;\mathbf{w}) = P(H_s;\mathbf{w}) + P(H_t;\mathbf{w}) \tag{6.10}$$

Let us consider S polytopes in $\mathbb{R}^D$. Because of the associativity and commutativity of the addition operations the result can be extended across S polytopes such that

$$P(H_1 + H_1 + \ldots + H_S;\mathbf{w}) = P(H_1;\mathbf{w}) + P(H_2;\mathbf{w}) + \ldots + P(H_S;\mathbf{w}) \tag{6.11}$$

Let the polytope resultant from the Minkowski sum be denoted as H such that $H = H_1 + H_2 + \ldots + H_S$. Also let us denote the face F_s as a face in the sth polytope H_s. From Eqn. (6.11) we know that the face $F \subseteq H$ is formed as the result of a Minkowksi sum $F = F_1 + F_2 \ldots + F_S$. The set of summands $\{F_1, F_2, \ldots, F_S\}$ for F is called the Minkowski decomposition of F [59]. We can state the following about the Minkowski decomposition [59]

Lemma 6.2. *Given the parameters* $\mathbf{w}$ *and* $\mathbf{w}'$ *such that*

$$F = P(H_1;\mathbf{w}) + P(H_2;\mathbf{w}) + \ldots + P(H_S;\mathbf{w}) = P(H_1;\mathbf{w}') + P(H_2;\mathbf{w}') + \ldots + P(H_S;\mathbf{w}')$$

then the Minkowski decomposition is unique, i.e. $P(H_s;\mathbf{w}) = P(H_s;\mathbf{w}')$ *for all s.*

Proof. The uniqueness of decomposition is proved by contradiction. Consider the Minkowski sum of all S faces such that all S summands are equal except for the sth sentence. Let be $\mathbf{h}^*$ a vertex in a reduced Minkowski sum that excludes the sth face $P(H_1;\mathbf{w}) + \ldots + P(H_{s-1};\mathbf{w}) + P(H_{s+1};\mathbf{w}) + \ldots + P(H_S;\mathbf{w})$. Let $\mathbf{h}'$ satisfy $\mathbf{h}' \in P(H_s;\mathbf{w}')$ and $\mathbf{h}' \notin P(H_s;\mathbf{w})$. The face F must contain $\mathbf{h}^* + \mathbf{h}'_s$, yet this point is not in the face F under $\mathbf{w}$, a contradiction. We can show the other direction by substituting $\mathbf{w}$ with $\mathbf{w}'$, which gives $P(H_s;\mathbf{w}) = P(H_s;\mathbf{w}')$. □

It is now possible to prove Proposition 6.1 [72].

Proof. It is sufficient to show that a cone in the normal fan of the Minkowski sum is equal to a cone in the common refinement.

$$\begin{aligned}
N_{P(H;\mathbf{w})} &= \{\mathbf{w}' \in (\mathbb{R}^D)^* : P(H_1 + \ldots + H_S;\mathbf{w}') = P(H_1 + \ldots + H_S;\mathbf{w})\} \\
&= \{\mathbf{w}' \in (\mathbb{R}^D)^* : P(H_1;\mathbf{w}) + \ldots + P(H_S;\mathbf{w}) = P(H_1;\mathbf{w}') + \ldots + P(H_S;\mathbf{w}')\} \qquad (6.12) \\
&= \{\mathbf{w}' \in (\mathbb{R}^D)^* : P(H_1;\mathbf{w}) = P(H_1;\mathbf{w}'), \ldots, P(H_S;\mathbf{w}) = P(H_S;\mathbf{w}')\} \qquad (6.13) \\
&= N_{P(H_1;\mathbf{w})} \cap \ldots \cap N_{P(H_S;\mathbf{w})}
\end{aligned}$$

The step from line (6.12) to line (6.13) is justified by the uniqueness of the Minkowski decomposition. Each normal cone of a face in the Minkowski sum is the intersection of the normal cones of faces in the summands. □

The dimension of F is at least as large as the dimension of each F_i in the decomposition. Thus following can be stated [59]

Corollary 6.3. *Let* $H_1, H_2, \ldots, H_S$ *be polytopes in* $\mathbb{R}^D$ *and let* $H = H_1 + H_2 + \ldots + H_S$. *A feature vector* $\mathbf{h} \in H$ *is a vertex of* H *if and only if there exists some* $\mathbf{w} \in (\mathbb{R})^*$ *such that each element of the decomposition is a vertex, i.e.* $\{\mathbf{h}_s\} = P(H_s;\mathbf{w})$ *for all s.*

This corollary is important because it means that the Minkowski decomposition of a vertex in the polytope H maps bijectively to an index vector $\mathbf{i}$. For example, for the vertex $\mathbf{h}_\mathbf{i} \in H$ we can write $\mathbf{h}_\mathbf{i} = \mathbf{h}_{1,i_1} + \mathbf{h}_{2,i_2} + \ldots + \mathbf{h}_{S,i_S}$.

6.2 A Polynomial Time Minkowski Sum Algorithm

In this section we describe the algorithm of Fukuda [59], which computes the Minkowski sum in polynomial time. For this description we define polynomial time as an algorithm that runs in time bounded by a polynomial function of both the input size and output size. The full complexity analysis is withheld until the end of the section, because it requires an understanding of the algorithm's mechanics.

The idea is to exploit the properties of edges as described in Section 5.2.2. If two vertices share an edge, then they are denoted as *adjacent*. Recall from Lemma 5.6 that a normal cone can be defined by a subset of the decision boundaries associated with adjacent vertices. In brief, the algorithm identifies an initial normal cone of the common refinement, removes redundant decision boundaries, and then enumerates through the remaining adjacent vertices of the Minkowski sum.

The full algorithm is an example of a reverse search [9]. The reverse search is a general technique that can be applied to many problems. For example, as noted in Section 5.3.1, a reverse search can be used to compute vertices of a single polytope. A reverse search is suited to the enumeration of a graph where we do not know the full structure of a graph, but do have information about adjacent vertices.

The reverse search specifies two functions, the local search function and the adjacency oracle, but does not provide an implementation for them. Different implementations of these functions can be 'plugged in' to the algorithm to solve different problems. We must therefore describe the reverse search first, because it provides a specification of these two supporting functions. We can then provide a thorough description of these functions.

6.2.1 Enumerating Vertices with a Reverse Search

We first describe reverse search with respect to the general problem of enumerating the vertices of any polytope H. After the description of the reverse search technique, we can then describe the supporting functions that are specific to the Minkowski sum enumeration problem.

Recall from Section 5.2.2 that the vertex set of H is defined as $V = \text{vert}(H)$. An undirected graph $G(H) = (V, E)$ can be constructed from the polytope where E is an edge set. Each edge E corresponds to an edge face in the polytope. We also denote the degree of a vertex in the graph as the number of edges incident to a vertex, and the maximum degree of a graph is the maximum degree of its vertices.

What makes a reverse search algorithm interesting is that the full graph $G(H)$ does

not need to be given explicitly. Instead the graph can be built up incrementally through a function called the adjacency oracle. A second function called the local search function defines the enumeration order of vertices.

The Adjacency Oracle

Not all graphs can have their vertices enumerated with a reverse search. If the graph $G(H)$ is suitable for the application of the search, then it is said to be 'given' by an adjacency oracle (adj). Specifically, if the graph is given by an adjacency oracle then the following conditions have to be satisfied:

(A1) The vertices are represented by nonzero integers

To avoid the proliferation of index variables we use $\mathbf{h} \in V$ to refer to a vertex, noting that while it is not a nonzero integer it can be readily mapped to an integer index to satisfy condition A1.

(A2) An integer δ is explicitly given which is an upper bound on the maximum degree of G

(A3) The adjacency oracle (adj) satisfying the following conditions is given:

1. for each vertex $\mathbf{h}$ and each number k with $1 \leq k \leq \delta$ the oracle returns $\text{adj}(\mathbf{h},k)$, a vertex adjacent to $\mathbf{h}$ or extraneous 0 (zero),
2. if $\text{adj}(\mathbf{h},k) = \text{adj}(\mathbf{h},k') \neq 0$ for some $\mathbf{h} \in V, k$ and k', then $k = k'$,
3. for each vertex $\mathbf{h}, \{\text{adj}(\mathbf{h},k) : \text{adj}(\mathbf{h},k) \neq 0, 1 \leq k \leq \delta\}$ is exactly the set of vertices adjacent to $\mathbf{h}$.

It will be shown in Section 6.2.2 that δ can be established from the summand polytopes of the Minkowski sum. The conditions in A3 imply that adj returns each adjacent vertex to $\mathbf{h}$ exactly once in a consistent order during the δ inquiries $\text{adj}(\mathbf{h},k), 1 \leq k \leq \delta$ for each vertex $\mathbf{h}$.

The Local Search Function

It would be possible to enumerate through the vertices of the graph using the adjacency oracle, but we to need to decide upon an enumeration order for vertices. Unless we store every vertex visited in memory, then the algorithm may end up cycling around previously

enumerated vertices. The algorithm would no longer be compact and the memory size would be proportional to the output.

The key insight behind a reverse search is that enumeration is much easier if the graph is transformed into a tree. We do not know the structure of the graph before the reverse search, which means that we have to perform this transformation incrementally during enumeration. This incremental transformation is guided by the local search search function.

Let us denote some vertex in the the graph as $\mathbf{h}^{(0)}$. For the purposes of reverse search this vertex is considered to be some 'optimal' vertex in the graph. Note that this does not imply that $\mathbf{h}^{(0)}$ has any special significance outside of the reverse search, it could be any vertex in the graph. Using the optimal vertex is it now possible to specify a local search function.

Consider the undirected graph $G(H) = (V,E)$, where V and E are the vertex and edge sets respectively. A triple $(G,\mathbf{h}^{(0)},f)$ is a called a local search if $\mathbf{h}^{(0)} \in V$ and f is a mapping $V \setminus \mathbf{h}^{(0)} \to V$ that satisfies

(L1) $\{\mathbf{h}, f(\mathbf{h})\} \in E$ for each $\mathbf{h} \in V \setminus \mathbf{h}^{(0)}$.

A local search is considered finite if it satisfies the following

(L2) for each $\mathbf{h} \in V \setminus \mathbf{h}^{(0)}$, there exists a positive integer k such that $f^k(\mathbf{h}) = \mathbf{h}^{(0)}$.

If the graph $G(H)$ is given by an adjacency oracle, then the local search is also said to be given by the adjacency oracle. Starting from any vertex in the graph, following the output of a finite local search function gives a path that ends in optimal vertex $\mathbf{h}^{(0)}$. The union of all these paths is called the trace T of the local search. The trace is a directed subgraph of $G(H)$ taking the form $T(H) = (V,E(f))$ where V is the same vertex set used in $G(H)$ and $E(f) = \{(\mathbf{h}, f(\mathbf{h})) : \mathbf{h} \in V \setminus \mathbf{h}^{(0)}\}$

Property 6.4. *If $(G(H),\mathbf{h}^{(0)},f)$ is a finite local search then its trace $T(H)$ is a spanning directed tree of $G(H)$ with edges pointing to $\mathbf{h}^{(0)}$ as the unique sink node of the tree.*

The Search

We can now describe the enumeration order of a reverse search. Starting at the optimal vertex $\mathbf{h}^{(\mathbf{0})}$ we then proceed to do a depth first search of the trace $T(H)$, moving away from the optimal vertex. The technique is called a reverse search due to this characteristic of searching in the opposite direction to the optimal vertex.

Because the search is a depth first search, once a leaf vertex has been enumerated the search must return back towards $\mathbf{h}^{(\mathbf{0})}$ to find branches that have not yet been enumerated.

The trace is thus traversed in two directions: a reverse traversal descends to the leaves of the tree, and a forward traversal ascends back towards the root. The full algorithm is stated in Algorithm 6.1.

Algorithm 6.1 Reverse search algorithm. After Avis and Fukuda [9]

Require: adj, $\delta, \mathbf{h}^{(0)}, f$
1: $\mathbf{h} \leftarrow \mathbf{h}^{(0)}$
2: $j \leftarrow 0$ ▷ neighbour counter
3: **repeat**
4: **while** $j < \delta$ **do**
5: $j \leftarrow j+1$
6: $next \leftarrow \text{adj}(\mathbf{h}, j)$
7: **if** $next \neq 0$ **then**
8: **if** $f(next) = \mathbf{h}$ **then** ▷ reverse traverse
9: **print h** ▷ output an enumerated vertex
10: $\mathbf{h} \leftarrow next$
11: $j \leftarrow 0$
12: **end if**
13: **end if**
14: **end while**
15: **if** $\mathbf{h} \neq \mathbf{h}^{(0)}$ **then** ▷ forward traverse
16: $\mathbf{h}' \leftarrow \mathbf{h}$
17: $\mathbf{h} \leftarrow f(\mathbf{h})$
18: $j \leftarrow 0$
19: **repeat** ▷ restore j
20: $j \leftarrow j+1$
21: **until** $\text{adj}(\mathbf{h}, j) = \mathbf{h}'$
22: **end if**
23: **until** $\mathbf{h} = \mathbf{h}^{(0)}$ and $j = \delta$

Note that each vertex $\mathbf{h} \in V \setminus \mathbf{h}^{(0)}$ is associated with a single forward traversal. Let us denote $t(f)$ and $t(\text{adj})$ as the time taken to evaluate the local search function and adjacency oracle. The complexity of the reverse search can now be given [9].

Theorem 6.5. *Suppose that a local search $(G, \mathbf{h}^{(0)}, f)$ is given by an adjacency oracle. Then the complexity of the reverse search algorithm is $O(\delta t(\text{adj})|V| + t(f)|E|)$*

Proof. The time complexity is determined by lines 6, 8 for reverse traversal and lines 17, 21 for forward traversal. Line 6 is executed at most δ times for each vertex, resulting in total time of $O(\delta t(\text{adj})|V|)$. Line 8 is executed for each edge incident with the vertex $\mathbf{h}$ for a total time of $O(\delta t(f)|E|)$. The two lines in the forward traversal are executed for each

vertex $\mathbf{h} \in V \setminus \mathbf{h}^{(0)}$ and hence the total total time for line 17 is $O(t(f)(|V|-1))$. Similarly, the total time for line 21 is $O(\delta t(\text{adj})(|V|-1)$. Since $|V|-1 \leq |E|$, by adding up the four complexities we have the claimed result. □

Corollary 6.6. *Suppose that a local search $(G, \mathbf{h}^{(0)}, f)$ is given by an adjacency oracle. Then the time complexity of the reverse search algorithm is $O(\delta|V|(t(\text{adj}) + t(f)))$. In particular if $\delta, t(f)$, and $t(\text{adj})$ are independent of the number of vertices $|V|$ in G then the complexity is linear in the output size $|V|$.*

Proof. The claim immediately follows from Theorem 6.5 and the fact that $2|E| \leq \delta V$ □

Now that the algorithm has been formally described we make some observations about the implementation details of reverse search. Because the reverse search is a depth-first search of a tree, Avis and Fukuda [9] note that a parallel implementation of reverse search is very easy to build. Branches can be enumerated independently of each other, with very little communication needed between different execution contexts.

Another potential improvement in line 19 is that during forward traversal the adjacency oracle is executed several times to restore the neighbour counter j. Instead the counter could be cached after each reverse traverse, which would reduce the compute time at the cost of increasing memory. Weibel [153] makes this change for his implementation of the Minkowski sum algorithm and reports only a modest increase in memory usage.

Example

The preceding description was a formal definition of the reverse search. To complement this description we illustrate the pseudo-code listing in Algorithm 6.1 with an example. For this example, we assume we have a polytope H and we show the full undirected cyclic graph associated with this polytope in Figure 6.3. Note that the complete graph is not a required input into the algorithm, but we provide it for clarity.

We are given a local search. Recall from Property 6.4 that a local search implies a trace of the graph. The trace of the graph in Figure 6.3 is shown in Figure 6.4. Note that the optimal vertex $\mathbf{h}^{(0)}$ can be reached from every other vertex in the trace.

The two previous figures show all the vertices in the graph, yet the reverse search algorithm is designed to enumerate the vertices without the complete graph. In Figure 6.5 we show in a subgraph the first two consecutive reverse traversals as bent blue directed edges. The edges are labelled with the order they are completed by the algorithm. The first reverse traversal sets the $\mathbf{h}$ variable to vertex $\mathbf{h}_1$. At this point the algorithm consults the adjacency-oracle for the next edge and finds the edge between $\mathbf{h}_1$ and $\mathbf{h}_6$. This edge is shown with a

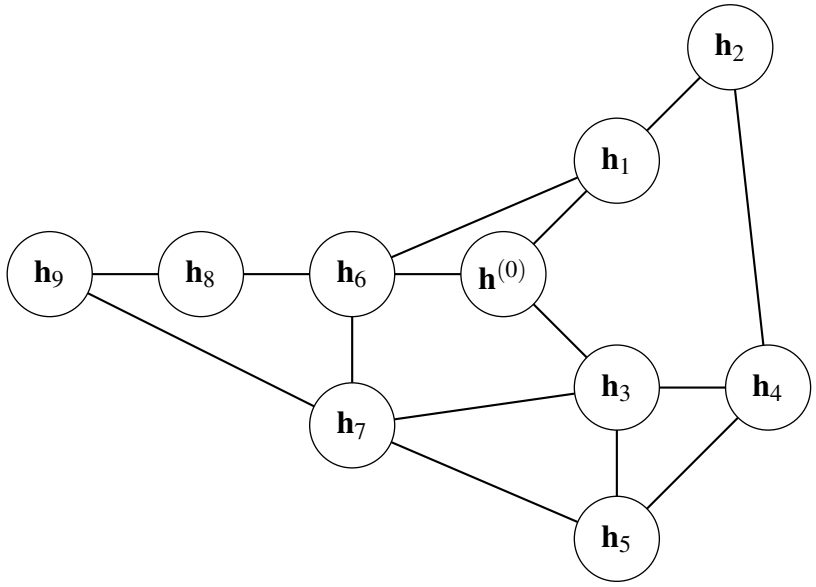


Fig. 6.3 An example undirected graph that represents the adjacency in a polytope

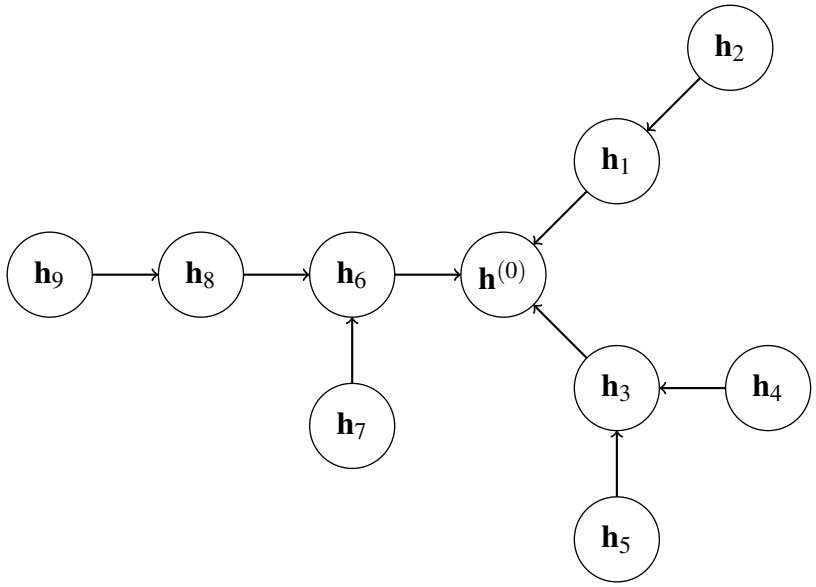


Fig. 6.4 An example trace of the graph in Figure 6.3 given a local search.

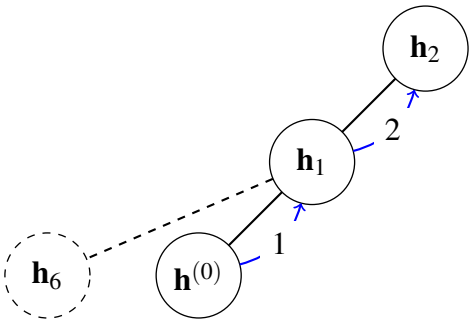


Fig. 6.5 Incremental enumeration for a subgraph of the graph in Figure 6.3. The undirected edges represent the original edges in the graph $G(H)$ that will form part of the trace. The bent blue directed lines represent iterations of reversal traversal. The dashed line is an edge in $G(H)$ that is rejected by the local search and does not form part of the trace.

dashed line in the figure. The algorithm then consults the local search function to find that it is not part of the trace $\mathbf{h}_1 \neq f(\mathbf{h}_6)$. The edge is discarded and the algorithm moves to the edge between $\mathbf{h}_1$ and $\mathbf{h}_2$. This edge satisfies the local search function $\mathbf{h}_1 = f(\mathbf{h}_2)$ and the second reverse traversal is executed.

Finally in Figure 6.6 we show the full enumeration order of the graph in terms of forward and reverse traversals. The traversals are shown as directed edges, labelled by the order of execution. Note that each vertex, except for $\mathbf{h}^{(0)}$, has exactly one forward traversal leaving the vertex.

6.2.2 Implementation of Reverse Search Functions

It is now possible to describe the full Minkowski sum vertex enumeration algorithm. The final step is to provide implementations of the adjacency oracle and the local search function. Both these implementations are based on linear programming. Recall from Section 3.2.3 that the complexity of a linear program can be reasonably approximated to $O(D^{3.5}M)$ where D is the number of variables and M is the number of constraints. Once these descriptions are complete, it will be possible to prove the following result [59].

Theorem 6.7. *There is a polynomial algorithm for the Minkowski addition of S polytopes that runs in time $O(\delta(D^{3.5}\delta)|V|)$ with $O(1)$ memory consumption.*

The Adjacency Oracle

Let us assume that we are given a vertex in the Minkowski sum polytope, associated with a index vector $\mathbf{i}$. Because of Proposition 6.1, the normal cone of this vertex is defined by the

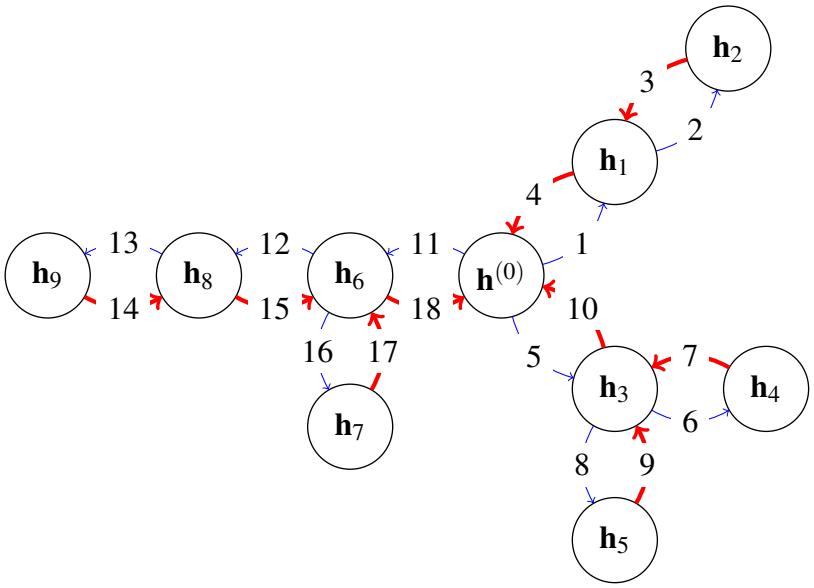


Fig. 6.6 Full execution of the reverse search of the graph in Figure 6.3. Reverse traversal is marked by thin blue edges, forward traversal in thick red edges

the set of constraints in (6.1).

Fukuda [59] defines an adjacency oracle based on these set of constraints. Recall from Corollary 5.4 that each non-redundant constraint corresponds to an edge in the polytope. Thus if there exists a constraint in (6.1) that is essential to the definition of the normal cone then it implies the existence of an adjacent vertex in the Minkowski sum.

Given two adjacent vertices in the sth polytope, $\mathbf{h}_{s,j}$ and $\mathbf{h}_{s,i_s}$, we can test whether a constraint associated with these two vertices is redundant in the normal come of $\mathbf{h_i}$ using the following feasibility problem

$$\begin{aligned} &\mathbf{w}(\mathbf{h}_{s,j} - \mathbf{h}_{s,i_s}) < 0 \\ &\mathbf{w}(\mathbf{h}_{t,k} - \mathbf{h}_{t,i_t}) \geq 0 \quad \text{for all } 1 \leq t \leq S, 1 \leq j \leq K \end{aligned} \tag{6.14}$$

This linear program to test this feasibility problem would have to be executed $K \times S$ times for each constraint of the cone, and it contains approximately $K \times S$ constraints. Even though this is not an exponential quantity and would be acceptable for the algorithm, we can improve upon it.

A important property of adjacency in the Minkowski sum polytope, is that it is derived from the adjacency of the summand polytopes. The formal relationship between the adjacency of vertices in the sum and summands is defined by the following proposition from

Fukuda [59].

Proposition 6.8. *Let $H_1, H_2, \ldots, H_S$ be polytopes in $(\mathbb{R})^D$ and let $H = H_1 + H_2 + \ldots + H_S$. Let $\mathbf{h_i}$ and $\mathbf{h_j}$ be two adjacent vertices in H with the Minkowski decompositions $\mathbf{h_i} = \mathbf{h}_{1,i_1} + \mathbf{h}_{2,i_2} + \ldots + \mathbf{h}_{S,i_S}$ and $\mathbf{h_j} = \mathbf{h}_{1,j_1} + \mathbf{h}_{2,j_2} + \ldots + \mathbf{h}_{S,j_S}$. Then $\mathbf{h}_{s,i_s}$ and $\mathbf{h}_{s,j_s}$ are either equal or adjacent in H_s for all s.*

Proof. It is sufficient to show that the Minkowski decomposition for an edge in the Minkowski sum H is formed of edges from the summands

$$[\mathbf{h_i}, \mathbf{h_j}] = [\mathbf{h}_{1,i_1}, \mathbf{h}_{1,j_1}] + [\mathbf{h}_{2,i_2}, \mathbf{h}_{2,j_2}] + \ldots + [\mathbf{h}_{S,i_S}, \mathbf{h}_{S,j_S}] \tag{6.15}$$

where each $[\mathbf{h}_{s,i_s}, \mathbf{h}_{s,j_s}]$ is a face of H_s. The statement is proved in two steps. First, for a given weight $\mathbf{w}$ such that $[\mathbf{h_i}, \mathbf{h_j}] = P(H; \mathbf{w})$ the Minkowski decomposition is $[\mathbf{h_i}, \mathbf{h_j}] = P(H_1; \mathbf{w}) + P(H_2; \mathbf{w}) + \ldots + P(H_S; \mathbf{w})$. If the edge $[\mathbf{h_i}, \mathbf{h_j}]$ contains both the vertices $\mathbf{h_i}$ and $\mathbf{h_j}$ then each element of the vertices' decomposition must be contained in the edge's decomposition

$$[\mathbf{h}_{s,i_s}, \mathbf{h}_{s,j_s}] \subseteq P(H_s; \mathbf{w}), \quad \text{for all } s. \tag{6.16}$$

For the second step, note the that edge in H must be contained in the Minkowski sum of the S edges.

$$[\mathbf{h_i}, \mathbf{h_j}] \subseteq [\mathbf{h}_{1,i_1}, \mathbf{h}_{1,j_1}] + [\mathbf{h}_{2,i_2}, \mathbf{h}_{2,j_2}] + \ldots + [\mathbf{h}_{S,i_S}, \mathbf{h}_{S,j_S}] \tag{6.17}$$

Both Eqn. (6.16) and Eqn. (6.17) can only be true if Eqn. (6.15) is true. □

Because of this proposition, we only need to consider the subset of constraints in (6.1) that are associated with adjacent vertices in the sentence-level polytopes. The other constraints can be removed, as they are redundant. Let us now characterise the adjacency of the vertices in the graph $G(H_s)$ with respect to an index vector $\mathbf{i}$. We define

$$\Delta(\mathbf{i}) = \{(s, j) : s = 1, \ldots, S \text{ and } j = 1, \ldots, \delta_s\} \tag{6.18}$$

where δ_s is the maximum degree of $G(H_s)$ and each element (s, j) of $\Delta(\mathbf{i})$ is called a pair. Because δ_s is the maximum degree of $G(H_s)$, for some index vectors $\mathbf{i}$ and polytopes (H_s) there will be pairs where j is greater than the number of edges incident to the vertex $\mathbf{h}_{i_s} \in H_s$. The subset of pairs that do correspond to edges are called valid pairs.

To be able to reference a specific adjacent vertex in H_s we need a function $a(s, j)$ that maps a pair to an index in an K-best list of the sth sentence. For brevity we omit the first

argument where it is obvious which sentence we are referring to, for example the feature vector $\mathbf{h}_{s,a(j)}$ is associated with the sth sentence only.

We are almost in a position to rewrite the feasibility problem in (6.14) with only the subset of constraints associated with adjacent vertices. However, there is a detail that we need to consider due to the following corollary [59]:

Corollary 6.9. *Let $H_1, H_2, \ldots, H_S$ be polytopes in $\mathbb{R}^D$ and let $H = H_1 + H_2 + \ldots + H_S$. For some parameter $\mathbf{w} \in (\mathbb{R})^*$ a subset E of H is an edge of H if and only if $E = P(H_1;\mathbf{w}) + P(H_2;\mathbf{w}) + \ldots + P(H_S;\mathbf{w})$ such that $\dim(P(H_s;\mathbf{w})) = 0$ or 1 and all faces of dimension 1 are parallel.*

As a brief but important aside, this corollary shows that all edges of H incident with $\mathbf{h_i}$ are parallel to at least one edge in an input polytope. This implies that $\delta = \delta_1 + \delta_2 + \ldots + \delta_S$ is an upper bound of the maximum degree in H.

The occurrence of parallel edges cause a problem, because they yield identical constraints in (6.14). In practise, the decomposition of an edge in the Minkowski sum polytope will almost always contain a single edge and $S-1$ vertices. However to be correct, we must cover the rare case where we have two or more parallel edges in the summand polytopes. For each pair (s, j) let us group together the parallel edges as

$$\Delta(\mathbf{i}, s, j) = \{(t,k) \in \Delta(\mathbf{i}) : \mathbf{h}_{t,a(k)} - \mathbf{h}_{t,i_t} \parallel \mathbf{h}_{s,a(j)} - \mathbf{h}_{s,i_s}, (s,j) \text{ and } (t,k) \text{ are valid}\} \quad (6.19)$$

Now we can finally rewrite the feasibility problem is (6.14) as a problem with a maximum of δ constraints

$$\begin{aligned} &\mathbf{w}(\mathbf{h}_{s,a(j)} - \mathbf{h}_{s,i_s}) < 0 \\ &\mathbf{w}(\mathbf{h}_{t,a(k)} - \mathbf{h}_{t,i_k}) \geq 0 \quad \text{For all valid } (t,k) \notin \Delta(\mathbf{i}, s, j) \end{aligned} \quad (6.20)$$

This feasibility problem is an improvement upon the problem in (6.14) because $\delta \leq K \times S$

If the problem in (6.20) is feasible, then there is edge incident to $\mathbf{h_i}$ parallel to $(\mathbf{h}_{s,j} - \mathbf{h}_{s,i_s})$. By Proposition 6.8 the adjacent index vector $\hat{\mathbf{i}}$ is given by

$$\hat{\mathbf{i}} = [\hat{i}_1, \hat{i}_2, \ldots, \hat{i}_S]^T \quad (6.21)$$

$$\hat{i}_s = \begin{cases} a(j) & \text{if there exists j such that } (s,j) \in \Delta(\mathbf{i}, s, j) \\ i_s & \text{otherwise} \end{cases} \quad (6.22)$$

The last step is to give the adjacency oracle in a form that satisfies the conditions A1, A2,

and A3 in Section 6.2.1. According to these conditions the adjacency oracle accepts a non-zero integer k, with $1 \leq k \leq \delta$, as an argument to indicate adjacency. In an effort to reduce the number of index variables we note that $|\Delta| = \delta$ and that pairs are countable, allowing the adjacency oracle to be written with respect to a pair and vertex of the Minkowski sum. The specification of the adjacency oracle is therefore [59]

$$\text{adj}(\mathbf{h}_{\mathbf{i}}, (s,j)) = \begin{cases} \mathbf{h}_{\hat{\mathbf{i}}} & \text{if } (s,j) \in \Delta(\mathbf{i}) \text{ and a valid pair} \\ 0 & \text{otherwise} \end{cases} \tag{6.23}$$

Lemma 6.10. *The adjacency oracle* $\text{adj}(\mathbf{h}, (s,j))$ *can be evaluated in time* $O(D^{3.5}\delta)$.

Proof. The adjacency oracle is dominated by solving the system in (6.20). The feasibility problem is solved with a linear program with D variables and at most δ inequalities and thus the claim follows. □

The Local Search Function

The final piece of the full algorithm is the local search function, and we now describe a local search function defined by Fukuda [59]. An arbitrary parameter $\mathbf{w}^{(0)}$ is selected. This parameter has no special significance in terms of the algorithm, but for relevance to SMT let us use the starting parameter for an iteration of MERT. Using this parameter we rank the feature vectors in K-best lists with respect to model score. The feature vector $\mathbf{h}^{(0)}$ is the sum of the 1-best feature vectors.

Recall that from Section 6.2.1 that the local search is a triple with the undefined element f, which is a mapping $V \setminus \mathbf{h}^{(0)} \to V$. For the Minkowski sum algorithm, the function f identifies the adjacent vertex with the highest model score under $\mathbf{w}^{(0)}$.

This implementation of f starts by noting a geometric property of normal fans [59].

Proposition 6.11. *Let* $\mathbf{h}$ *and* $\mathbf{h}^{(0)}$ *be two distinct extreme points of H, and let* $\mathbf{w} \in N_{\{\mathbf{h}\}}$ *and* $\mathbf{w}^{(0)} \in N_{\{\mathbf{h}^{(0)}\}}$. *Then there exists a vertex* $\mathbf{h}'$ *adjacent to* $\mathbf{h}$ *such that* $N_{\{\mathbf{h}'\}}$ *contains a parameter in* $(\mathbb{R}^D)^*$ *of the form* $(1-\theta)\mathbf{w} + \theta\mathbf{w}^{(0)}$ *for some* $0 \leq \theta \leq 1$.

Proof. Since $\mathbf{h}^{(0)} \neq \mathbf{h}$, their normal cones are two distinct full dimensional cones in the normal fan $\mathscr{N}(H)$. This means the parametrised point $t(\theta) := \mathbf{w} + \theta(\mathbf{w}^{(0)} - \mathbf{w})(0 \leq \theta \leq 1)$ in the line segment $[\mathbf{w}^{(0)}, \mathbf{w}]$ must leave at least one of the decision boundaries of the first cone $N_{\{\mathbf{h}\}}$ as θ increases from 0 to 1. Since the decision boundaries are in one-to-one correspondence with the edges of G incident with $\mathbf{h}$, any one of the half-spaces violated first corresponds to a vertex $\mathbf{h}'$ adjacent to $\mathbf{h}$ claimed by the proposition. □

The next step is to define a method for selecting a single representative parameter for a cone, which is called the canonical parameter. We demand the parameter must lie in the interior in the cone, and not some shared face. This canonical parameter is found using a linear program similar to the program for finding interior points in (3.15). Given a vertex $\mathbf{h_i} \in H$ a canonical parameter for $N_{\{\mathbf{h_i}\}}$ is found by

$$\begin{array}{lll} \text{maximise} & w_0 & \\ \text{subject to} & \mathbf{w}(\mathbf{h}_{s,a(j)} - \mathbf{h}_{s,i_s}) + w_0 \leq 0 & \text{for all valid } (s,j) \in \Delta(\mathbf{i}) \\ & w_0 \leq C & \end{array} \tag{6.24}$$

where C is any positive constant to bound the solution. The solution set to this problem is not unique and different linear programming solvers will yield different results. The assumption is made that the same solver is used throughout the algorithm to ensure a unique solution.

It is now possible to provide an implementation for f. First we assume that $\mathbf{w}^{(0)}$ is interior to the cone $N_{\{\mathbf{h}^{(0)}\}}$ and compute the canonical parameter $\mathbf{w}$ for $N_{\{\mathbf{h_i}\}}$. These two parameters define the oriented line $t(\theta) := \mathbf{w} + \theta(\mathbf{w}^{(0)} - \mathbf{w})$. The function then iterates through δ possible edges, solving for θ to find the intersection between the line $t(\theta)$ and the bounding hyperplane associated with the edge. The edge associated with the smallest value of θ determines the vertex returned by the function.

This implementation of f may introduce a degeneracy if the line hits the intersection of two or more decision boundaries, which causes a tie. The function would select multiple edges, making reverse search untenable. Ties are broken by a symbolic perturbation of $\mathbf{w}$ as $\mathbf{w} + (\varepsilon^1, \varepsilon^2, \ldots, \varepsilon^D)$ for sufficiently small ε.

Before the function can be used it must be shown to yield a valid trace $T(f)$ that satisfies Property 6.4. The property is recast as a proposition to be proved [59].

Proposition 6.12. *The directed graph of $T(f) = (V, E(f))$ is a spanning tree of $G(H)$ (an undirected graph) and $\mathbf{h}^{(0)}$ is a unique sink node of $T(f)$.*

Proof. By the construction, $\mathbf{h}^{(0)}$ is a unique sink node of $T(f)$. It is sufficient to show that $T(f)$ has no directed cycles. For this, take any edge $(\mathbf{h}, \mathbf{h}' = f(\mathbf{h})) \in E(f)$. Let $\mathbf{w}$ and $\mathbf{w}^{(0)}$ be the canonical vector for $\mathbf{h}$ and $\mathbf{h}^{(0)}$ respectively. Since $\mathbf{w}$ is an interior point of $N_{\{\mathbf{h}\}}$

$$\mathbf{w}(\mathbf{h}' - \mathbf{h}) < 0 \tag{6.25}$$

Because the canonical points are selected as interior points of the associated normal cones, there exists $0 < \theta < 1$ such that $\mathbf{w}' := (1-\theta)\mathbf{w} + \theta\mathbf{w}^{(0)} \in N_{\mathbf{h}'}$. Again, because $\mathbf{w}'$ is interior

to the normal cone $N_{\mathbf{h}'}$ the following derivation can be made

$$\begin{aligned} 0 &\leq ((1-\theta)\mathbf{w} + \theta\mathbf{w}^{(0)})(\mathbf{h}' - \mathbf{h}) \\ &= (1-\theta)\mathbf{w}(\mathbf{h}' - \mathbf{h}) + \theta\mathbf{w}^{(0)}(\mathbf{h}' - \mathbf{h}) \\ &< \theta\mathbf{w}^{(0)}(\mathbf{h}' - \mathbf{h}) \quad \text{by (6.25)} \end{aligned}$$

This implies that the vertex $\mathbf{h}'$ attains a strictly higher model score than $\mathbf{h}$ under $\mathbf{w}^{(0)}$. Therefore, there is no directed cycle in $T(f)$. □

The last part of the description of the local search function is to provide a complexity analysis of the time taken to execute the function [59].

Lemma 6.13. *There is an implementation of the local search function $f(\mathbf{h})$ with evaluation time $O(\delta(D^{3.5}\delta))$, for each $\mathbf{h} \in V \setminus \mathbf{h}^0$ with the Minkowski decomposition $\mathbf{h} = \mathbf{h}_1 + \mathbf{h}_2 + \ldots + \mathbf{h}_S$.*

Proof. The local search function implementation can be split into two stages. The first is the computation of the canonical parameter. The second is to determine the first bounding hyperplane of $N_{\{\mathbf{h}\}}$ that hits the line segment $t(\theta)$. The first stage is computed by the linear program in (6.24) with $D+1$ variables and at most $\delta+1$ inequalities, giving a complexity of $O(D^{3.5}\delta)$. The second stage consists of solving at most δ one-variable equations. Therefore the computation is dominated by the first stage and the total complexity is $O(D^{3.5}\delta)$. □

Complexity of the Full Algorithm

One of the main claims of this chapter is that it is possible to define a compact polynomial time algorithm to compute the Minkowski sum. We are now in a position to take the various complexity results from the reverse search and support functions to finalise this claim.

An aspect of the algorithm we have not described is how to compute the edges in the summand polytopes. One method would be to infer the edges from a set of facets computed by a facet enumeration algorithm. Another method would be the use of a linear program to test all pairs in a K-best list, which would result is SK^2 linear programs. This computation is dwarfed by the time spent enumerating vertices and can be ignored.

Let us now combine the complexity results of the reverse search algorithm together with the complexity statements of the support functions to prove Theorem 6.7 [59].

Proof. Recall from Corollary 6.6 that the complexity of the reverse search is $O(\delta(t(\text{adj}) + t(f)|V|)$. By Lemmas 6.10 and 6.13 both $t(\text{adj})$ and $t(f)$ can be replaced by $O(D^{3.5}\delta)$ and

the claimed time complexity follows. The space complexity is dominated by those of the functions f and adj which are linear in the input size. □

This is a significant result. Galley and Quirk [62] describe the Minkowski sum as a problem with N^S vertices to test for feasibility. They do give a divide-and-conquer algorithm to reduce the number of tests, but they do not give time or memory complexities for this algorithm. It is also not clear how to adapt their algorithm for parallelisation, when the algorithm of Fukuda [59] can easily be adapted to parallel execution.

We should be clear that this result is heavily dependent on the output size $|V|$. This quantity is the number of feasible index vectors, which correspond to sets of feasible constraints in (6.1).

6.3 Upper Bound of the Minkowski Sum

We have described the Minkowski sum of many polytopes and a method for computing it in polynomial time with respect to the number of vertices $|V|$ in the Minkowski sum. Thus, the number of vertices is critical to the computability of MERT. If $|V|$ does not increase exponentially with respect to S, K, or D then the algorithm will identify a global minimum in polynomial time.

This section investigates the upper bounds on $|V|$. We first describe a collection of upper bound theorems that state that $|V|$ does increase exponentially with respect to D, we then discuss what impact these upper bounds have on SMT systems before moving on to the implications of the upper bound theorems on linear models in general.

6.3.1 Upper Bound Theorems

In Section 7.2.1 we describe how MERT can be applied to lattices and hypergraphs. In this description, there are certain operations concerning Och's line optimisation that are formulated using a 2-dimensional Minkowski sum [54, 94]. This use of the Minkowski sum is said to be tractable because of the following theorem

Theorem 6.14. *Let $H_1, \ldots, H_S$ be polytopes in $\mathbb{R}^2$ with at most K vertices each. Then the number of vertices of $H_1 + \ldots + H_S$ is $O(SK)$.*

Proof. de Berg et al. [46, Theorem 13.5] states that the same upper bound exists for edges in a Minkowski sum. We note that the number of edges in a 2-dimensional polytope is equal to the number of vertices. □

For 2-dimensional polytopes the Minkowski sum is a very computationally efficient operation. Recall from Section 5.4.2 that Och's line optimisation can be formulated as the projection of a polytope into a 2-dimensional space. This result confirms that Och's line optimisation is very cheap to compute for many source sentences. Let us now move onto the case where $D > 2$

Theorem 6.15. *Let $H_1, \ldots, H_S$ be polytopes in $\mathbb{R}^D$ with at most K vertices each. Then the upper bound on the number of vertices of $H_1 + \ldots + H_S$ is $O(S^{D-1}K^{2(D-1)})$.*

Proof. See Gritzmann and Sturmfels [72, Corollary 2.1.11] □

We cannot understate the significance of this result. Essentially, this result states that the upper bound on the number of feasible index vectors increases exponentially with dimension. We also note that it is not just the number of feasible index vectors that explodes with an increase in dimension. Consider the following upper bound theorem for edges:

Theorem 6.16. *If H is a D-dimensional polytope with |V| vertices, then for $D \geq 3$ the following is an upper bound on the number of edges $|E|$*

$$|E| \leq \binom{|V|}{2} \tag{6.26}$$

Proof. This is a special case of the upper bound theorem. See Ziegler [162, Theorem 8.23]. □

Because Theorem 6.15 states that the upper bound on |V| grows exponentially with dimension, then the possible number of edges also increases. Recall that a vertex corresponds to a normal cone in the normal fan and edge to a feasible decision boundary between the normal cones. Thus not only does the possible number of normal cones increase exponentially as dimension increases, the possible number of decision boundaries increases, and the upper bound on the max degree δ of the associated graph also increases.

6.3.2 The Impact of Upper Bounds on SMT systems

We have described theoretical results based around upper bound theorems. The next step is to investigate what role these upper bounds have when estimating the parameters for a typical SMT systems.

Let us conduct this investigation by plugging in some hard numbers into the theorems. Consider the CUED Russian-to-English [125] entry to the Eighth Workshop on Machine

Translation (WMT'13) [24]. This entry used MERT to estimate parameters for a linear model on a training set of 1502 parallel sentences and 1000-best lists. The maximum possible number of index vectors is $K^S = 1000^{1502}$. The upper bound in Theorem 6.15 reaches this value at $D = 492$.

An argument might be made that this is an unreasonable calculation because we have not taken into account the redundancy of feature vectors in the K-best lists. Let us then set K=10 to model a 99% redundency in feature vectors. The maximum possible number of index vectors is 10^{1502} and this value is reached at $D = 291$. Both of these values of D are well below what is used in the SMT systems with sparse features surveyed in Section 3.6.

Instead of varying D, we can keep it fixed to illustrate how the size of the training set would need to increase to hit the upper bound in Theorem 6.15. Let us assume that feature dimension is $D = 1000$ and the K-best list size is fixed at $K = 1000$ then we would need the training set size to be $S = 3164$ for us to have enough vertices in the Minkowski sum to hit the upper bound. If the feature dimension is increased to $D = 2000$ then the training set would have to increase to $S = 6540$. In the CUED Russian-to-English entry to WMT'13 we used a training set of $S = 1502$, and we note there were only 3003 sentences available in the Russian-to-English track to cover both the training and test data.

These numbers show that due to high feature dimension and limited amounts of training data the upper bound on feasible index vectors for an SMT system greatly exceeds the number of possible index vectors. We believe that it is therefore plausible for every index vector in an SMT system to be feasible.

6.3.3 Linear Models and the Upper Bound Theorems

We have described a set of upper bound theorems. We have then established that these upper bounds could be hit in an SMT system with a relatively low feature dimension D. It is our belief that this upper bound needs to be taken into account when estimating the parameters of a linear model. We now discuss the implications of these upper bounds with respect to MERT, large-margin methods, and ranking methods.

We note when we have a low feature dimension then due to Theorem 6.15 it is impossible for all index vectors in a training set to be feasible. We believe that limiting the number of feasible index vectors aids generalisation. This limit constrains the ability of the optimiser to boost the model score of a hypothesis without affecting the model score of other hypotheses. If the optimiser boosts the model score of one hypothesis in a K-best list, it may penalise the model score of another hypothesis in a second K-best list.

For example, the CUED Russian-to-English entry to WMT'13 uses 12 features which means that a maximum of $1502^{11}1000^{22}$ index vectors are feasible. Although this is a large number, it is a minuscule fraction of the 1000^{1502} total possible index vectors. The rest of the possible index vectors are discarded, and we believe that all parameter estimation methods for linear models benefit from this discarding mechanism.

Next we consider what happens as feature dimension increases. This discarding mechanism affects fewer hypotheses, until eventually the discarding mechanism ceases. At this point, it will be possible for each index vector to be feasible. Let us assume we have some oracle index vector $\hat{\mathbf{i}}$ with the lowest possible error. If all index vectors are feasible, then it follows that $\hat{\mathbf{i}}$ is feasible.

It is possible to argue that ranking methods avoid this problem due to the large number of additional constraints in (3.22). It is more difficult to satisfy all these additional constraints, and thus the number of feasible solutions is limited. In response to this argument we note that each constraint is associated with an edge, and by Theorem 6.16 it will become possible to satisfy more of these constraints.

This feasibility of $\hat{\mathbf{i}}$ raises two questions. Is it possible to find a parameter $\hat{\mathbf{w}}$ that yields $\hat{\mathbf{i}}$? If so, then will $\hat{\mathbf{w}}$ generalise to a test set? Let us assume that the answer to first question is yes, then we have to decide whether $\hat{\mathbf{w}}$ will generalise. One recent result that seems to answer this question is that of Green et al. [71] where gains of over 13 BLEU points are reported in the training set with the addition of many features, yet only 2 to 3 BLEU points are found in the test set.

If overtraining is already happening, then the question as to whether we can find $\hat{\mathbf{w}}$ seems irrelevant. The systems surveyed in Section 3.6 exhibit lower training set gains, but test set gains found by adding sparse features tends to be more modest within the range of $\frac{1}{2}$ to 1 BLEU points. Although we cannot categorically say that overtraining is inherit in linear models with a large feature dimension the evidence, both theoretical and empirical in the literature, does seem to suggest that this is not a fruitful avenue to pursue without good means to ensure generalisation.

6.4 The Minkowski Sum with Projected MERT

By Theorem 6.15, MERT is impractical for a large D-dimensional SMT system, but it can still be applied to problems of a lower dimension using projected MERT. In this section we demonstrate that the Minkowski sum could be computable for low values of D.

We are fortunate that there is an open-source implementation of the Minkowski sum

algorithm described in Section 6.2 [153]. Using the CUED Russian-to-English [125] entry to WMT'13 [24] we build a tune set of 1502 sentences. The system uses 12 features which we initially tune with lattice MERT [102] to get a parameter $\mathbf{w}^{(0)}$. Using this parameter we generate 1000-best lists. We then project the feature functions in the 1000-best lists to a 3-dimensional representation that includes the source-to-target phrase probability (UtoV), the word insertion penalty (WIP), and the model score due to $\mathbf{w}^{(0)}$. In our experimental setup these two features are the 2nd and 4th features respectively, yielding the following projection matrix

$$A_{3,12} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ w_1^{(0)} & w_2^{(0)} & w_3^{(0)} & w_4^{(0)} & w_5^{(0)} & w_6^{(0)} & w_7^{(0)} & w_8^{(0)} & w_9^{(0)} & w_{10}^{(0)} & w_{11}^{(0)} & w_{12}^{(0)} \end{bmatrix}$$

We use the Minkowski sum algorithm to compute BLEU as a Δ is applied to the parameters from $\mathbf{w}^{(0)}$. By Theorem 6.15 the upper bound on the number of vertices when $D = 3$ is $O(S^2K^4) = O(1502^2 1000^4)$.

The implementation was executed was on a machine with 12 CPU cores for several weeks and terminated it before completion. Because the final computation did not terminate we only have partially enumerated the vertices of the Minkowski sum.

Figure 6.7 is the plot of the error surface as computed by the Minkowski sum algorithm. It is the 3-dimensional equivalent to the 2-dimensional diagram in the lower part of Figure 3.1. Because we used a parallel implementation, the algorithm has enumerated distinct search regions. Another point to note is because of the nature of the depth-first search, the algorithm chooses to explore regions away from $\mathbf{w}^{(0)}$.

Even with only a subset of the vertices, we can draw some conclusions. The two parameters seem to be closely correlated, in that increasing them both equally leaves the BLEU score unchanged. Increasing one parameter at the expense of another results in a drop in performance, and decreasing the UtoV parameter seems to be catastrophic.

Weibel [153] designed and built this algorithm as a general tool for mathematicians. In its current form it is not suited for SMT, but it does show that the Minkowski sum could be a computable quantity in lower dimensions with some changes to the implementation.

The first change would be to switch to some form of breadth-first search, at the expense of higher memory consumption. We could also compute the error as vertices are enumerated, which could be used to guide the algorithm to enumerate regions of low error. The Minkowski sum algorithm is very suited to parallelisation over many machines because very little communication and coordination is needed to enumerate the branches of

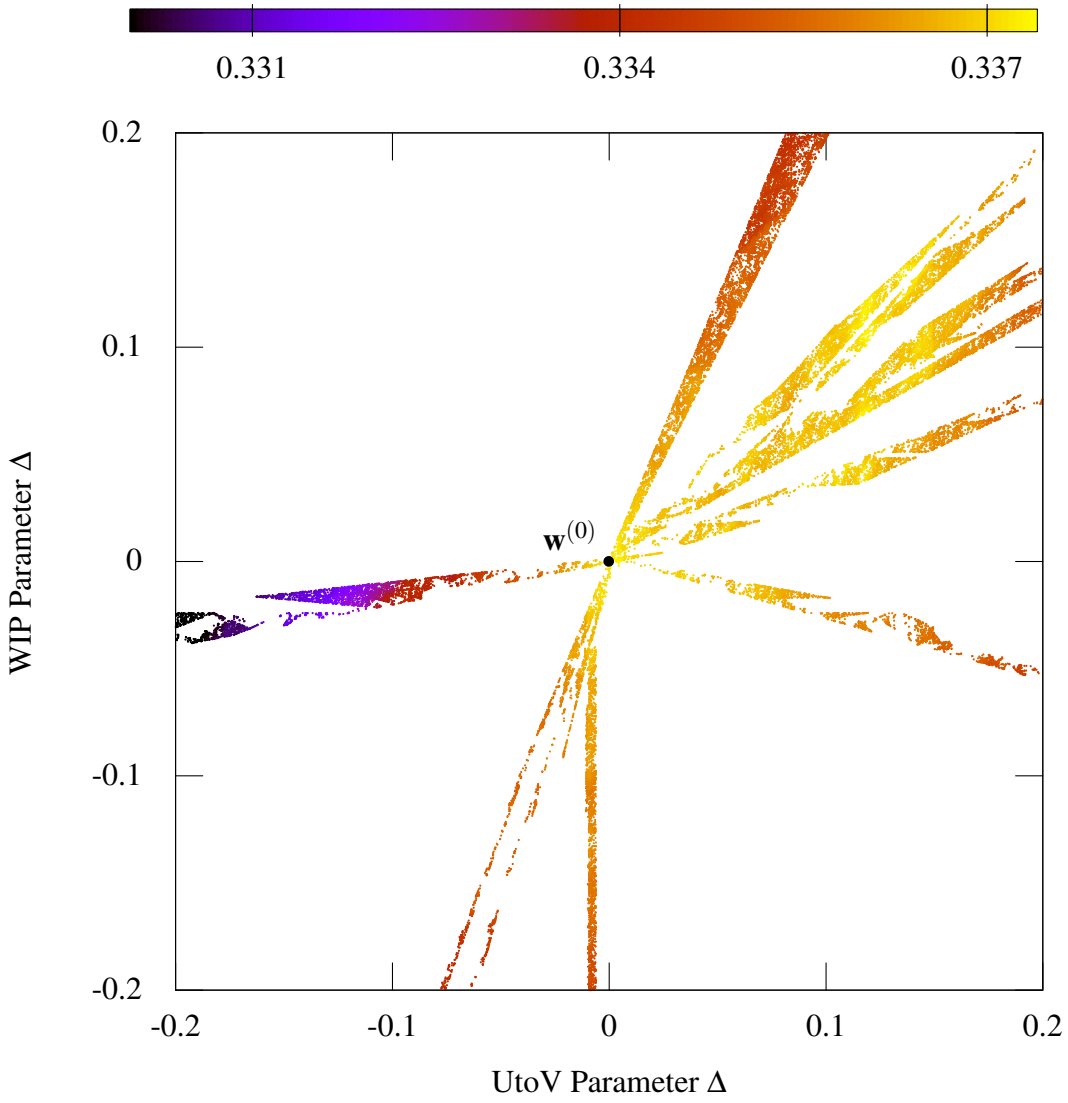


Fig. 6.7 The BLEU score over a 1502 sentence tune set for the CUED Russian-to-English system plotted over the change Δ in two parameters from $\mathbf{w}^{(0)}$. Due to the long runtime of the algorithm only a subset of the vertices of the Minkowski sum were enumerated, with enumerated vertices shown in the shaded regions.

the trace. We could also explore using the double-description or quick-hull algorithms in the adjacency oracle or local search functions.

The directions chosen in this experiment were selected at random to illustrate the application of the Minkowski sum algorithm. Another issue with projected MERT is that we do not have a good method to select directions. There is no guarantee that the projected space would contain parameters associated with low errors.

6.5 Summary

We now summarise the points made in this chapter. Section 6.1 describes the equivalence of the common refinement and the normal fan of the Minkowski sum. We reiterate that our method of computing the common refinement, based on the algorithm of Fukuda [59], is more computationally efficient than the of method Galley and Quirk [62] which computes the Minkowski sum. We believe the investment made in learning new terminology and techniques is worthwhile because we can build the required normal fan in parameter space directly. This normal fan is a much more useful representation of MERT because our final goal is to identify parameter vectors that maximise a set of feature vectors.

Section 6.3.3 discusses the some of the fundamental problems involved for estimating parameters for linear models. In summary, we believe that the combinatoric explosion in the number of feasible index vectors due to a large feature dimension results in overtraining. One possible to solution to this problem is to project the feature vectors to a lower dimension and perform error-based training in this projected space.

Chapter 7

A Description of Lattice-based MERT Using Tropical Geometry

7.1 Introduction

In Chapter 2 we described how, given a source sentence, we can generate a very large number of hypotheses. Section 3.2 described MERT, a method for finding a parameter vector that assigns the highest model score to the lowest error hypotheses from a set of K hypotheses. Typically $K = 1000$, which means that MERT only considers a small number of the billions of possible hypotheses.

Section 2.3.4 describes how hypotheses can be compactly encoded in a word lattice. This lattice representation allows for a much larger representation of hypotheses space. Macherey et al. [102] describe a method for performing MERT using the word lattice as input. Because the word lattice contains many more hypotheses, a fewer number of iterations are needed until convergence. This treatment was expanded to hypergraphs by Kumar et al. [94].

It has been noted that the operations described by Macherey et al. [102] can be formulated as a semiring for WFSTs and hypergraphs. The first reference was in the CDEC system description [56] to a 'MERT semiring', although no formulation of this semiring was given. Sokolov and Yvon [134] provided a formulation for a MERT semiring, based on sets of lines and the SweepLine algorithm. Finally, Dyer [54] provide a description based on 'point-line duality', which as noted in Section 5.3.2 is an instance of the well-known dual representation of polytopes.

Our approach is to a MERT formalism for WFSTs is based on *tropical geometry* [135]. Essentially, tropical geometry is an algebra based around operations on polytopes and which

neatly ties together WFSTs with convex geometry. Our approach also immediately extends to multiple dimensions because polynomials can be expressed in terms of many variables. We published an early version of this work in Waite et al. [149], independently of Sokolov and Yvon [134] and preceding Dyer [54]. In Section 7.6 we discuss the advantages of our formulation relative to these others.

We first describe the method of Macherey et al. [102] for lattice MERT, and describe the WFST shortest distance algorithm. The next section is a description of the *tropical polynomial* [135], which forms the set of values in our semiring. Using an example we illustrate how the tropical polynomial provides a concise and expressive description of MERT. We conduct experiments to show that the tropical polynomial formalism yields identical results to lattice MERT. Finally, we describe an extension of our work to lattice-based MERT over multiple dimensions and then end with a discussion.

7.2 Lattice-based MERT

We now describe lattice-based MERT for statistical machine translation. The original description by Macherey et al. [102] was given as two operations preformed over a directed acyclic graph and so we present it in this form. We then describe the WFST shortest distance and how it can be used to model lattice-based MERT.

7.2.1 Lattice Line Optimisation

Recall from Section 3.2.1 that Och's line optimisation for a source sentence $\mathbf{f}$ considers the points in $(\mathbb{R}^D)^*$ along a line $\mathbf{w}^{(0)} + \gamma\mathbf{d}$. The application of this line to a feature function $\mathbf{h}(\mathbf{e},\mathbf{f})$ create a linear function $\ell_{\mathbf{e}}$. Also recall from Section 2.3.4 that a word lattice is a weighted directed acyclic graph that encodes a very large set of hypotheses.

Macherey et al. [102] use methods from computational geometry to develop a procedure for conducting line search directly over a word lattice. Each lattice edge is labelled with a word e from a hypothesis $\mathbf{e}$ and has a weight defined by the vector of word specific feature function values $\mathbf{h}(e,\mathbf{f})$ so that the weight of a path in the lattice is found by summing over the word specific feature function values on that path. Given a line through parameter space, the goal is to extract from a lattice its upper envelope and the associated hypotheses.

We summarise here the Macherey et al. [102] formulation. The procedure proceeds node by node through the lattice. Suppose that for a lattice state q the upper envelope is known for all the partial hypotheses on all paths leading to q. The upper envelope defines a set of

functions $\{\ell_{\tilde{\mathbf{e}}_1}(\gamma), ..., \ell_{\tilde{\mathbf{e}}_N}(\gamma)\}$ over the partial hypotheses $\tilde{\mathbf{e}}_n$. Two operations propagate the upper envelope to other lattice nodes.

We refer to the first operation as the 'extend' operation. Consider a single edge from state q to state q'. This edge defines a linear function associated with a single word $\ell_e(\gamma)$. A path following this edge transforms all the partial hypotheses leading to q by concatenating the word e. The upper envelope associated with the edge from q to q' is changed by adding $\ell_e(\gamma)$ to the set of linear functions. The intersection points are not changed by this operation.

The second operation is a union. Suppose q' has another incoming edge from a state q'' where $q \neq q''$. There are now two upper envelopes representing two sets of linear functions. The first upper envelope is associated with the paths from the initial state to state q' via the state q. Similarly the second upper envelope is associated with paths from the initial state to state q' via the state q''. The upper envelope that is associated with all paths from the initial state to state q' via both q and q'' is the union of the two sets of linear functions. This union is no longer a compact representation of the upper envelope as there may be functions which never achieve a maximum for any value of γ. The SweepLine algorithm [19] is applied to the union to discard redundant linear functions and their associated hypotheses [102].

The union and extend operations are applied to states in topological order until the final state is reached. The upper envelope computed at the final state compactly encodes all the hypotheses that maximise Eqn. (5.1) along the line.

7.2.2 Line Search using WFSTs

We now describe how the computation of the upper envelope of a lattice can be formulated as a shortest distance problem in a weighted finite-state transducer (WFST). We use the definition of a WFST used in Section 2.3.3. Recall that the weight of a path π is

$$w[\pi] = \bigotimes_{k=1}^{K} w[e_k] = w[e_1] \otimes \cdots \otimes w[e_K] \tag{7.1}$$

If $\mathscr{P}(q)$ denotes the set of all paths in T starting from an initial state in I and ending in state q, then the shortest distance $d[q]$ is defined as the generalised sum $\oplus$ of the weights of all paths leading to q [108]:

$$d[q] = \oplus_{\pi \in \mathscr{P}(q)} w[\pi] \tag{7.2}$$

For some semirings, such as the tropical semiring, the shortest distance is the weight of the shortest path. For other semirings, the shortest distance is associated with multiple paths through a WFST [108]; for these semirings there are shortest distances but in general no

shortest paths. That will be the case in what follows.

The crux of the formulation of Sokolov and Yvon [134] is that the operations in Section 7.2.1 can be expressed as a semiring. The generalised times $\otimes$ is the extend operation, and the generalised plus $\oplus$ is the union operation. Under this semiring, the shortest distance corresponds to the upper envelope. Note that this formulation does not describe how to keep track of the partial hypotheses, and the hypotheses must be tracked or extracted outside of the WFST formulation. We describe how to extract these hypotheses using a WFST in Section 7.3.5.

7.3 Tropical Geometry

Tropical geometry [135] is a relatively new branch of algebraic geometry [44] concerned with algebraic representations of piecewise linear functions. It has been used for machine learning and bioinformatics [115, 116]. The Bank of England has also applied tropical geometry for modelling auctions during the recent financial crises [86].

We first describe tropical polynomials [135] and demonstrate that they form a suitable semiring for the line optimisation problem. We then describe how to compute the upper envelope over translation lattices by mapping edge weights to tropical polynomials and computing shortest distances under the tropical polynomial semiring. We pull these results together into an algorithm we call tropical geometry MERT (TGMERT). Finally we work through an example of TGMERT.

7.3.1 Tropical Polynomials

A polynomial is a linear combination of a finite number of non-zero monomials. A monomial consists of a real valued coefficient multiplied by a variable, and this variable may have an exponent that is a non-negative integer. A polynomial function is defined by evaluating a polynomial:

$$f(\gamma) = a_n\gamma^n + a_{n-1}\gamma^{n-1} + \cdots + a_2\gamma^2 + a_1\gamma + a_0$$

A useful property of polynomials is that they form a ring[1] [44] and therefore are candidates for use as weights in WFSTs.

Speyer and Sturmfels [135] apply the definition of a classical polynomial to the formulation of a tropical polynomial. The tropical semiring uses summation for the generalised product $\otimes$ and a min operation for the generalised sum $\oplus$. In this form, let γ be a variable

[1] A ring is a semiring that includes negation.

that represents an element in the tropical semiring weight set $\mathbb{K} \cup \{-\infty, +\infty\}$. We can write a monomial of γ raised to an integer exponent as

$$\gamma^i = \underbrace{\gamma \otimes \cdots \otimes \gamma}_{i},$$

where i is a non-negative integer and $\otimes$ is classical addition. The monomial can also have a constant coefficient: $a \otimes \gamma^i, a \in \mathbb{R}$. We can define a function that evaluates a tropical monomial for a particular value of γ. For example, the tropical monomial $a \otimes \gamma^i$ is evaluated as:

$$f(\gamma) = a \otimes \gamma^i = a + i\gamma$$

This shows that a tropical monomial is a linear function with the coefficient a as its y-intercept and the integer exponent i as its gradient. A tropical polynomial is the generalised sum of tropical monomials where the generalised sum is evaluated using the min operation. For example:

$$f(\gamma) = (a \otimes \gamma^i) \oplus (b \otimes \gamma^j) = \min(a + i\gamma, b + j\gamma)$$

Evaluating tropical polynomials in classical arithmetic gives the minimum of a finite collection of linear functions.

Tropical polynomials can also be multiplied by a monomial to form another tropical polynomial. For example:

$$\begin{aligned} f(\gamma) &= [(a \otimes \gamma^i) \oplus (b \otimes \gamma^j)] \otimes (c \otimes \gamma^k) \\ &= [(a+c) \otimes \gamma^{i+k}] \oplus [(b+c) \otimes \gamma^{j+k}] \\ &= \min((a+c) + (i+k)\gamma, (b+c) + (j+k)\gamma) \end{aligned}$$

The product of two tropical polynomials is a Minkowski sum, as defined in Section 6.1. Suppose A, B, C, and D are tropical monomials in γ, so that $A = a \otimes \gamma^i$ and the equivalent linear function of A is $\ell_A = a + i\gamma$. An example of a product of two tropical polynomials is:

$$\begin{aligned} f(\gamma) &= (A \oplus B) \otimes (C \oplus D) \\ &= (A \otimes C) \oplus (A \otimes D) \oplus (B \otimes C) \oplus (B \otimes D) \\ &= \min(\ell_A + \ell_B, \ell_A + \ell_D, \ell_B + \ell_C, \ell_B + \ell_D) \end{aligned}$$

The tropical semiring is distributive over the generalised product. Evaluating the product of two tropical polynomials in classical arithmetic gives the minimum of the Minkowski sum

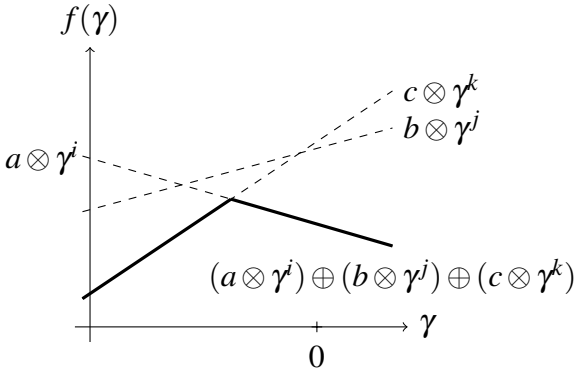


Fig. 7.1 Redundant terms in a tropical polynomial and their associated lines. In this case $(a \otimes \gamma^i) \oplus (b \otimes \gamma^j) \oplus (c \otimes \gamma^k) = (a \otimes \gamma^i) \oplus (c \otimes \gamma^k)$.

of two sets of linear functions [135].

A tropical polynomial in one variable represents a continuous piecewise linear function $\mathbb{R} \to \mathbb{R}$ with a finite number of segments [135]. The function $f(\gamma)$ defines the lower envelope. If we change Eqn. (3.11) to use an argmin instead of an argmax and negate the feature functions we can now see a parallel between the upper envelope and a tropical polynomial. Inverting the upper envelope in this fashion does not change the values of γ at which the intersection points occur.

7.3.2 Canonical Form of a Tropical Polynomial

We noted in Section 3.2.1 that linear functions induced by some hypotheses do not contribute to the upper envelope and can be discarded. Terms in a tropical polynomial can have similar behaviour. Figure 7.1 plots the lines associated with the three terms of the example polynomial function $f(\gamma) = (a \otimes \gamma^i) \oplus (b \otimes \gamma^j) \oplus (c \otimes \gamma^k)$. We note that the piecewise linear function can also be described with the polynomial $f(\gamma) = (a \otimes \gamma^i) \oplus (c \otimes \gamma^k)$. The latter representation is simpler but equivalent.

Having multiple representations of the same polynomial causes problems when implementing the shortest distance algorithm defined by Mohri [108]. This algorithm performs an equality test between values in the semiring used to weight the WFST. The behaviour of the equality test is ambiguous when there are multiple polynomial representations of the same piecewise linear function. We therefore require a canonical form of a tropical polynomial so that a single polynomial represents a single function. We define the canonical form of a tropical polynomial to be the tropical polynomial that contains only the monomial terms necessary to describe the piecewise linear function it represents.

We remove redundant terms from a tropical polynomial after computing the generalised sum. Each term corresponds to a linear function; linear functions that do not contribute to the upper envelope are discarded. Only monomials which correspond to the remaining linear functions are kept in the canonical form. The canonical form of a tropical polynomial thus corresponds to a unique and minimal representation of the upper envelope.

7.3.3 Integer Approximations for Tropical Monomials

In Section 7.3.1 we noted that the function defined by the upper envelope in Eqn. (3.11) is similar to the function represented by a tropical polynomial. A significant difference is that the formal definition of a polynomial only allows integer exponents, whereas the gradients in Eqn. (3.11) are real numbers. The upper envelope can therefore encode a larger set of model parameters than can the tropical polynomial.

To create an equivalence between the upper envelope and tropical polynomials we can approximate the linear functions $\{\ell_{\mathbf{e}}(\gamma) = a(\mathbf{e},\mathbf{f}_s) + \gamma \cdot b(\mathbf{e},\mathbf{f}_s)\}$ that compose segments of the upper envelope. We define $\tilde{a}(\mathbf{e},\mathbf{f}_s) = [a(\mathbf{e},\mathbf{f}_s) \cdot 10^n]_{\text{int}}$ and $\tilde{b}(\mathbf{e},\mathbf{f}_s) = [b(\mathbf{e},\mathbf{f}_s) \cdot 10^n]_{\text{int}}$ where $[x]_{\text{int}}$ denotes the integer part of x. The approximation to $\ell_{\mathbf{e}}(\gamma)$ is:

$$\begin{aligned} \ell_{\mathbf{e}}(\gamma) \approx \tilde{\ell}_{\mathbf{e}}(\gamma) &= \frac{\tilde{a}(\mathbf{e},\mathbf{f}_s)}{10^n} + \gamma \cdot \frac{\tilde{b}(\mathbf{e},\mathbf{f}_s)}{10^n} \\ &= \frac{[a(\mathbf{e},\mathbf{f}_s) \cdot 10^n]_{\text{int}}}{10^n} + \gamma \cdot \frac{[b(\mathbf{e},\mathbf{f}_s) \cdot 10^n]_{\text{int}}}{10^n} \end{aligned} \tag{7.3}$$

The result of this operation is to approximate the y-intercept and gradient of $\ell_{\mathbf{e}}(\gamma)$ to n decimal places. We can now represent the linear function $\tilde{\ell}_{\mathbf{e}}(\gamma)$ as the tropical monomial $-\tilde{a}(\mathbf{e},\mathbf{f}_s) \otimes \gamma^{-\tilde{b}(\mathbf{e},\mathbf{f}_s)}$. Note that $\tilde{a}(\mathbf{e},\mathbf{f}_s)$ and $\tilde{b}(\mathbf{e},\mathbf{f}_s)$ are negated since tropical polynomials define the lower envelope as opposed to the upper envelope defined by Eqn. (3.11).

The linear function represented by the tropical monomial is a scaled version of $\ell_{\mathbf{e}}(\gamma)$, but the upper envelope is unchanged (to the accuracy allowed by n). If for a particular value of γ, $\ell_{\mathbf{e}_i}(\gamma) > \ell_{\mathbf{e}_j}(\gamma)$, then $\tilde{\ell}_{\mathbf{e}_i}(\gamma) > \tilde{\ell}_{\mathbf{e}_j}(\gamma)$. Similarly, the boundary points are unchanged: if $\ell_{\mathbf{e}_i}(\gamma) = \ell_{\mathbf{e}_j}(\gamma)$, then $\tilde{\ell}_{\mathbf{e}_i}(\gamma) = \tilde{\ell}_{\mathbf{e}_j}(\gamma)$.

Using a scaled version of $\ell_{\mathbf{e}}(\gamma)$ as the basis for a tropical monomial may cause negative exponents to be created. Following Speyer and Sturmfels [135], we widen the definition of a tropical polynomial to allow for these negative exponents.

7.3.4 Computing the Upper Envelope using the Shortest Distance Algorithm

In lattice line optimisation a word specific linear function $\ell_e(\gamma)$ is associated with each edge. Each path from an initial state in I to a state $q \in Q$ corresponds to a partial translation hypothesis so that the path weight can be interpreted as a linear function where the y-intercept and gradient are the arithmetic sum of the word specific linear functions. The upper envelope at state q is the maximum of the lines associated with the paths from the initial state to q.

Analogously, we create tropical monomials to represent the word specific linear functions $\ell_e(\gamma)$ using the integer approximation described in Section 7.3.3. We transform the word lattice so that each edge weight is a tropical monomial. The weight of each path from the initial state to a state q is defined as the generalised product $\otimes$ of the word specific monomials; this path weight is itself a monomial. The shortest distance $d[q]$ from the initial state to state q is the generalised sum $\oplus$ of the weights of paths to q defined by Eqn. (7.2), which forms a canonical tropical polynomial. This tropical polynomial is equivalent to the upper envelope at state q. The upper envelope of the full lattice is given by the generalised sum $\oplus$ of tropical polynomials in the lattice final states F.

This suggests an alternate explanation for what the shortest distance computed using the tropical polynomial semiring represents. Conceptually, there is a continuum of lattices which have identical edges and vertices but with varying, real-valued edge weights determined by values of $\gamma \in \mathbb{R}$, so that each lattice in the continuum is indexed by γ. The tropical polynomial computed by the shortest distance algorithm agrees with the shortest distance through each lattice in the continuum.

A Note on Hypergraphs

We do not discuss the application of the tropical polynomial semiring to hypergraphs. However, one interesting aspect of the hypergraph application is that the operands of the generalised times $\otimes$ might be full polynomials, i.e. contain more than one monomial term. In this case, the generalised times takes the form of a Minkowski sum. It has been noted elsewhere that the generalised times for the other MERT semiring formulations is also a Minkowski sum [54, 94]. This is the other use of the Minkowski sum for MERT that was alluded to in Section 6.3.

We only mention this aspect of MERT for hypergraphs because of the upper bound theorems described in Section 6.3. The implication being that the number of feasible derivations

after a generalised times increase exponentially with dimension. Consequently, given a high feature dimensionality the number of feasible hypotheses encoded in the hypergraph could be very large. This effect is not seen in a lattice because edges can only be weighted with monomial terms.

7.3.5 Extracting the Error Surface

Tropical monomial weights can be transformed into regular tropical weights by evaluating the tropical monomial for a specific value of γ. For example, a tropical polynomial evaluated at $\gamma = 1$ corresponds to the tropical weight:

$$f(1) = -\tilde{a}(e,\mathbf{f}_s) \otimes 1^{-\tilde{b}(e,\mathbf{f}_s)} = -\tilde{a}(e,\mathbf{f}_s) - \tilde{b}(e,\mathbf{f}_s)$$

Each monomial term in the tropical polynomial shortest distance represents a linear function. The intersection points of these linear functions define intervals of γ (as in figures 3.1 and 7.1). To compute the error surface we need to find the lowest cost hypothesis for each interval. Let us assume we have $n+1$ intervals separated by n interval boundaries. We use the midpoint at of each interval to transform the lattice of tropical monomial weights into a lattice of tropical weights. The sequence of words that label the shortest path through the transformed lattice is the lowest cost hypothesis for the interval. The shortest path can be extracted using the WFST shortest path algorithm [111]. As a technical matter, the midpoints of the first interval $[-\infty, \gamma_1)$ and last interval $[\gamma_n, \infty)$ are not defined. We therefore evaluate the tropical polynomial at $\gamma = \gamma_1 - 1$ and $\gamma = \gamma_n + 1$ to find the lowest cost hypothesis in the first and last intervals, respectively.

7.3.6 The Tropical Geometry MERT Algorithm

We now describe an alternative algorithm to Lattice MERT that is formulated using the tropical polynomial shortest distance. As input to this procedure we use a word lattice weighted with word specific feature functions $\mathbf{h}(e,\mathbf{f})$, a starting point $\mathbf{w}^{(0)}$, and a direction $\mathbf{d}$ in parameter space.

1. Convert the word specific feature functions $\mathbf{h}(e,\mathbf{f})$ to a linear function $\ell_e(\gamma)$ using $\mathbf{w}^{(0)}$ and $\mathbf{d}$, as in Eqn. (3.10).

2. Convert $\ell_e(\gamma)$ to $\tilde{\ell}_e(\gamma)$ by approximating y-intercepts and gradients to n decimal places, as in Eqn. (7.3).

3. Convert $\tilde{\ell}_e(\gamma)$ in Eqn. (7.3) to the tropical monomial $-\tilde{a}(e,\mathbf{f}_s) \otimes \gamma^{-\tilde{b}(e,\mathbf{f}_s)}$.

4. Compute the WFST shortest distance to the exit states [108] with generalised sum $\oplus$ and generalised product $\otimes$ defined by the tropical polynomial semiring. The resulting tropical polynomial represents the upper envelope of the lattice.

5. Compute the intersection points of the linear functions corresponding to the monomial terms of the tropical polynomial shortest distance. These intersection points define intervals of γ in which the lowest cost hypothesis does not change.

6. Using the midpoint of each interval convert the tropical monomial $-\tilde{a}(e,\mathbf{f}_s) \otimes \gamma^{-\tilde{b}(e,\mathbf{f}_s)}$ to a regular tropical weight. Find the lowest cost hypothesis for this interval by extracting the shortest path using the WFST shortest path algorithm [111].

TGMERT is implemented using the OpenFst Toolkit [2]. A weight class is added for tropical polynomials which maintains them in canonical form. The $\otimes$ and $\oplus$ operations are implemented for piecewise linear functions, with the SweepLine algorithm used to remove redundant monomials.

7.3.7 TGMERT Worked Example

This section presents a worked example showing how we can use the TGMERT algorithm to compute the upper envelope of a lattice. We start with a three state lattice with a two dimensional feature vector shown in the upper part of Figure 7.2.

We want to optimize the parameters along a line in two-dimensional feature space. Suppose the initial parameters are $\lambda_1^2 = [0.7, 0.4]$ and the direction is $d_1^2 = [0.3, 0.5]$. Step 1 of the TGMERT algorithm (Section 7.3.6) maps each edge weight to a word specific linear function. For example, the weight of the edge labelled "x" between states 0 and 1 is transformed as follows:

$$\begin{aligned}
\ell_e(\gamma) &= \underbrace{\sum_{m=1}^{2} \lambda_m h_1^M(e,\mathbf{f})}_{a(e,\mathbf{f})} + \gamma \underbrace{\sum_{m=1}^{2} d_m h_1^M(e,\mathbf{f_s})}_{b(e,\mathbf{f})} \\
&= \underbrace{0.7 \cdot -1.4 + 0.4 \cdot 0.3}_{a(e,\mathbf{f})} + \gamma \cdot \underbrace{0.3 \cdot -1.4 + 0.5 \cdot 0.3}_{b(e,\mathbf{f})} \\
&= -0.86 - 0.27\gamma
\end{aligned}$$

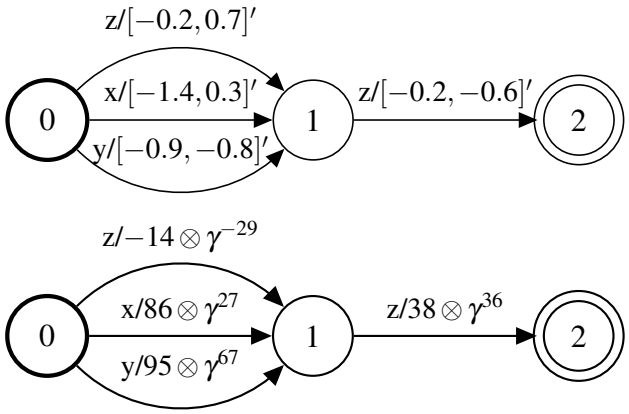


Fig. 7.2 The upper part is a translation lattice with 2-dimensional log feature vector weights $h_1^M(e,\mathbf{f})$ where $M = 2$. The lower part is the lattice from the upper part with weights transformed into tropical monomials.

Step 2 of the TGMERT algorithm converts the word specific linear functions into tropical monomial weights. Since all y-intercepts and gradients have a precision of two decimal places, we scale the linear functions $\ell_{\mathbf{e}}(\gamma)$ by 10^2 and negate them to create tropical monomials (Step 3). The edge labelled "x" now has the monomial weight of $86 \otimes \gamma^{27}$. The transformed lattice with weights mapped to the tropical polynomial semiring is shown in the lower part of Figure 7.2.

We can now compute the shortest distance [108] from the transformed example lattice with tropical monomial weights. There are three unique paths through the lattice corresponding to three distinct hypotheses. The weights associated with these hypotheses are:

$$\begin{aligned} -14 \otimes \gamma^{-29} \otimes 38 \otimes \gamma^{36} &= 24 \otimes \gamma^{7} && \text{z z} \\ 86 \otimes \gamma^{27} \otimes 38 \otimes \gamma^{36} &= 122 \otimes \gamma^{63} && \text{x z} \\ 95 \otimes \gamma^{67} \otimes 38 \otimes \gamma^{36} &= 133 \otimes \gamma^{103} && \text{y z} \end{aligned}$$

The shortest distance from initial to final state is the generalised sum of the path weights: $(24 \otimes \gamma^7) \oplus (133 \otimes \gamma^{103})$. The monomial term $122 \otimes \gamma^{63}$ corresponding to "x z" can be dropped because it is not part of the canonical form of the polynomial (Section 7.3.2). The shortest distance to the exit state can be represented as the minimum of two linear functions: $\min(24 + 7\gamma, 133 + 103\gamma)$.

We now wish to find the hypotheses that define the error surface by performing Steps 5 and 6 of the TGMERT algorithm. These two linear functions define two intervals of γ. The

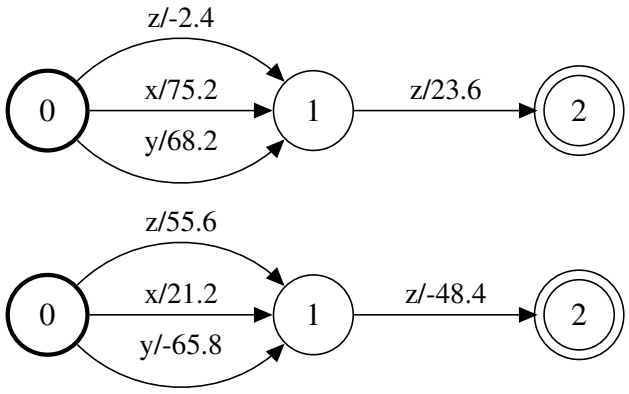


Fig. 7.3 The lattice in the lower part of Figure 7.2 transformed to regular tropical weights: $\gamma = -0.4$ (top) and $\gamma = -1.4$ (bottom).

linear functions intersect at $\gamma \approx -1.4$; at this value of γ the lowest cost hypothesis changes. Two lattices with regular tropical weights are created using $\gamma = -0.4$ and $\gamma = -2.4$. These are shown in Figure 7.3. For the lattice shown in the upper part the value for the edge labelled "x" is computed as $86 \otimes -0.4^{27} = 86 + 0.4 \cdot 27 = 75.2$.

When $\gamma = -0.4$ the lattice in the upper part in Figure 7.3 shows that the shortest path is associated with the hypothesis "z z", which is the lowest cost hypothesis for the range $\gamma < 1.4$. The lattice in the lower part of Figure 7.3 shows that when $\gamma = -2.4$ the shortest path is associated with the hypothesis "y z", which is the lowest cost hypothesis when $\gamma > 1.4$.

7.3.8 Tropical Polynomial Edge Pruning Algorithm

An inefficiency of Step 6 in the TGMERT algorithm (Section 7.3.6) is the repeated use of the shortest path algorithm on lattices with regular tropical weights to extract lowest cost hypotheses. There are many paths in these lattices that do not correspond to a lowest cost hypothesis for any value of γ. We describe a prune algorithm that deletes edges from the tropical monomial weighted lattice that do not belong to a shortest path of any of the associated regular tropical weighted lattices. This algorithm is applied to reduce the size of the tropical monomial lattice produced by TGMERT.

If $\mathscr{R}(q)$ denotes the set of all paths from state q to a final state in F, then the reverse distance $d'[q]$ is defined as the generalised sum $\oplus$ of the weights of the paths in $\mathscr{R}(q)$: $d'[q] = \oplus_{\pi \in \mathscr{R}(q)} w[\pi]$. At state q, the edge $e_i \in E[q]$ can be deleted if its weight does not contribute to the tropical polynomial shortest distance. The edge e_i does not contribute and

	Arabic-to-English					
	MERT		LMERT		TGMERT	
	Tune	Test	Tune	Test	Tune	Test
1	36.2 42.1	40.9	36.2 44.5	40.3	36.2 44.5	37.7
2	42.0 45.1	43.2	41.4 45.7	44.2	38.7 45.7	44.0
3	44.5 45.5	44.1	45.8 45.8	44.2	45.7 45.8	44.2
4	45.6 45.7	44.0				

	Chinese-to-English					
	MERT		LMERT		TGMERT	
	Tune	Test	Tune	Test	Tune	Test
1	19.5 25.3	16.7	19.5 29.3	21.4	19.6 29.3	21.4
2	16.4 18.9	23.9	21.2 30.1	31.2	21.2 30.1	31.3
3	23.6 28.2	29.1	30.9 32.3	32.2	30.9 32.3	32.4
4	29.2 31.3	31.5	32.3 32.3	32.2	32.4 32.4	32.4
5	31.3 31.8	32.1				
6	32.1 32.4	32.3				
7	32.4 32.4	32.3				

Table 7.1 GALE AR→EN and ZH→EN BLEU scores by MERT iteration. BLEU scores at the initial and final points of each iteration are shown for the Tune sets.

can therefore be deleted if the following equality is satisfied:

$$\bigoplus_{\{e_j \in E[q]: e_j \neq e_i\}} (w[e_j] \otimes d'[n[e_j]]) = d'[q] \tag{7.4}$$

7.4 Experiments

We compare feature weight optimization using K-best MERT [113], lattice MERT [102], and tropical geometry MERT. We refer to these as MERT, LMERT, and TGMERT, resp.

We investigate MERT performance in the context of the Arabic-to-English GALE P4

and Chinese-to-English GALE P3 evaluations[2]. For Arabic-to-English translation, word alignments are generated over around 9M sentences of GALE P4 parallel text. Following de Gispert et al. [48], word alignments for Chinese-to-English translation are trained from a subset of 2M sentences of GALE P3 parallel text. Hierarchical rules are extracted from alignments using the constraints described in [36] with additional count and pattern filters [80]. We use a hierarchical phrase-based decoder [81] that supports direct generation of word lattices that encode large numbers of alternative hypotheses. For K-best MERT optimisation, we extract the top 1000-best hypotheses.

MERT optimizes the weights of the following 13 features: target language model, source-to-target and target-to-source translation models, word and rule penalties, number of usages of the glue rule, word deletion scale factor, source-to-target and target-to-source lexical models, and three count-based features that track the frequency of rules in the parallel data [16]. In both Arabic-to-English and Chinese-to-English experiments all MERT implementations start from a flat feature weight initialization. At each iteration new lattices and K-best lists are generated from the best parameters at the previous iteration, and each subsequent iteration includes 100 hypotheses from the previous iteration. For Arabic-to-English we consider an additional twenty random starting parameters at every iteration. All translation scores are reported for the IBM implementation of BLEU using case-insensitive matching. We report BLEU scores for the Tune set at the start and end of each iteration.

The results for Arabic-to-English and Chinese-to-English are shown in Table 7.1. Both TGMERT and LMERT converge to a comparable performance level in fewer iterations than MERT, consistent with previous reports [102]. The differences observed between LMERT and TGMERT are mainly due to the accumulation of numerical differences.

7.4.1 Effect of Tropical Polynomial Pruning

We now investigate the number of edges that can be removed from a lattice using a tropical polynomial prune (Section 7.3.8). Using lattices generated at the start of iteration 2 for the Chinese-to-English experiments we perform line searches over each coordinate axis. Before each line search we prune the lattice, recording the number of edges in the lattice before and after pruning. Figure 7.4 shows the effect of pruning on two features. We observe that the number of pruned edges differs substantially, suggesting varying diversity of lowest cost hypotheses as a function of the two different features.

[2]See http://projects.ldc.upenn.edu/gale/data/catalog.html

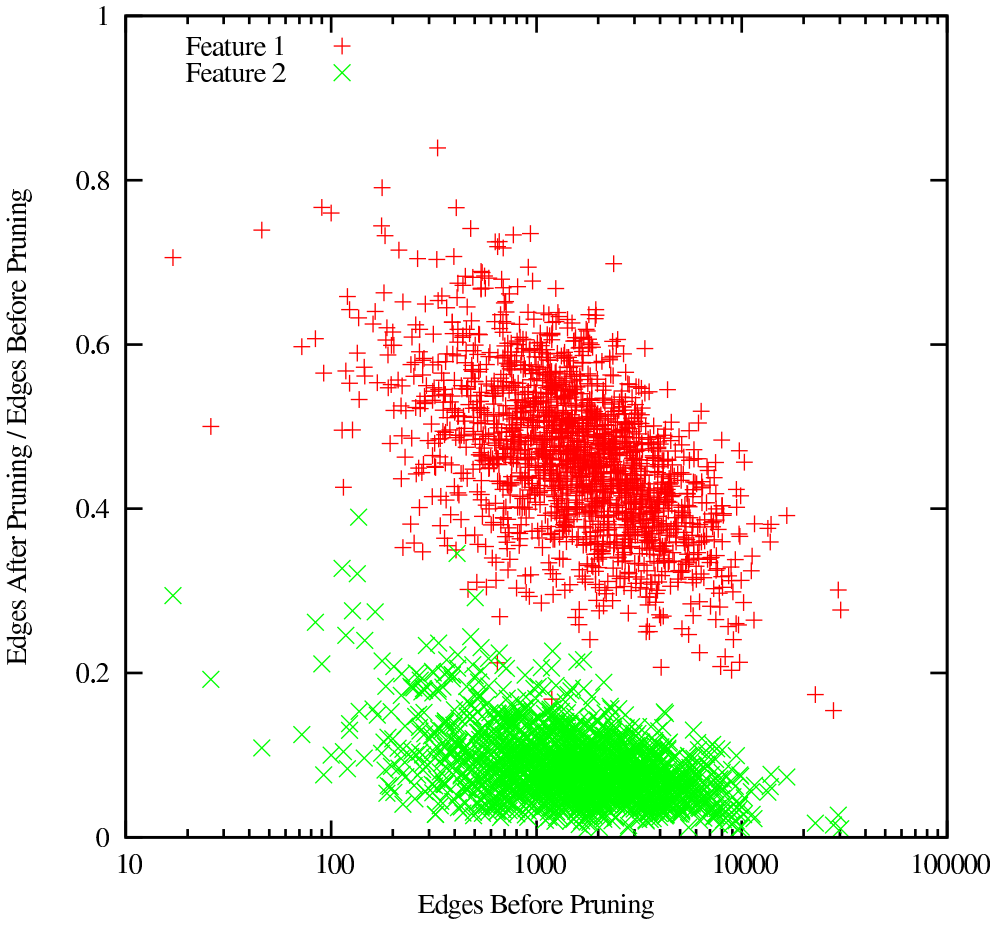


Fig. 7.4 The proportion of edges pruned from each lattice on a sentence-by-sentence basis.

7.5 Multi-direction Lattice-based MERT

Section 5.4 describes a version of MERT for multiple directions over K-best lists called projected MERT. This algorithm is presented with respect to the affine projection of a polytope. As tropical geometry is an algebra over polytopes, we can extend the formalism to represent projected polytopes for lattices.

In the description of TGMERT we used tropical polynomials expressed in terms of a single variable γ, which represents a piecewise linear function and an associated set of intervals along a line. Following the steps of the algorithm described in Section 7.3.6 it is possible to define a tropical polynomial in two variables where each monomial takes the form $a \otimes \gamma_1^i \otimes \gamma_2^j$. These monomial terms can be used to represent the projected polytope resultant from applying the projection matrix $A_{3,D}$ as described in Section 5.4.2.

Richter-Gebert et al. [129] describe tropical polynomials in multiple variables and an algorithm called the tropical curve drawing algorithm. The result of this algorithm is a set of values of γ_1, γ_2 where two or more monomial terms have the same cost. This set is called the tropical hypersurface $\mathscr{H}$.

The hypersurface is derived from the normal fan for a projected polytope, which was described in Section 5.4.2. In this section we present a brief description of the tropical curve drawing with respect to a projected polytope. The motivation for this presentation is to show that tropical polynomials have a multiple variable formulation, rather than for any deep insight into tropical hypersurfaces. We refer the reader to Speyer and Sturmfels [135] and Richter-Gebert et al. [129] for detailed descriptions with examples of polynomials and their associated hypersurfaces.

The tropical curve drawing algorithm is as follows

1. Transform each monomial term $a \otimes \gamma_1^i \otimes \gamma_2^j$ into a point (i, j, a) in $\mathbb{R}^3$.

2. Compute the convex hull of these points.

3. Project the lower envelope of the convex hull onto the plane $\mathbb{R}^3 \mapsto \mathbb{R}^2$ by deleting the last coordinate such that $(i, j, a) \mapsto (i, j)$. The result is a graph embedded in the plane formed from the vertices and edges of the convex hull.

4. Compute the dual graph, this is the tropical hypersurface $\mathscr{H}$.

This algorithm seems a little mysterious at first, but let us work through each item in turn to show the connection to a projected polytope. The first step converts a tropical polynomial

to set of points that correspond to the projected feature vectors in Eqn. (5.23). The second item computes the set of faces for the polytope of the feature vectors.

The reason we only require the lower envelope in item 3 is due to the constraints in (5.30). We only require parameter vectors that have a negative final component - note the switch in sign because of the tropical representation - to describe the lower envelope. Recall from Section 5.3.2 that the normal cones can be defined with respect to the facet normals. Therefore we only need the facets that form the lower envelope, because they are the facets that have normals with a negative third component. The resultant primary graph exists in $\mathbb{R}^2$. Note that the primary graph yields the tropical polynomial representation of the shortest distance. Each vertex in the primary graph corresponds a monomial term in the tropical polynomial.

Recall from Lemma 5.7 that the normal cone of the edge is orthogonal to the edge. To represent the decision boundary between two hypotheses on the plane, we can draw a line normal to the edge. The set of these orthogonal lines form the dual graph in $(\mathbb{R}^2)^*$ and yield the hypersurface. Hence item 4.

Each line in the hypersurface corresponds to a decision boundary, where two hypotheses have equal low costs. These decision boundaries are associated with the the essential constraints in (5.27). Thus, the hypersurface can be characterised as the intersection of the decision boundaries for all normal cones in $(\mathbb{R}^3)^*$ and the plane at $\tilde{\mathbf{w}}_3 = 1$.

The resultant hypersurface corresponds to the lattice for a single input sentence. To perform MERT we need to compute a Minkowski sum of the projected feature vectors, as represented by the tropical polynomial shortest distances, for the entire training set. We can use the information in the primary graph in step 3 to identify the vertices and adjacencies necessary for the Minkowski sum algorithm of [59] as described in Section 6.2.

We can then compute the error surface from the resulting Minkowski sum, allowing us to perform MERT. This relates the presentation in this chapter to the projected MERT algorithm described in Algorithm 5.1 and the description of the Minkowski sum in Chapter 6.

7.6 Discussion

We have described a lattice-based line search algorithm which can be incorporated into MERT for parameter tuning of SMT systems and systems based on linear models. The approach recasts the problem in terms of tropical geometry, and once the problem is formulated, implementation is relatively straightforward using available WFST algorithms.

As noted in the introduction, there are other formulations of the MERT semiring. We believe our formulation to have many advantages over these other formulations. The first, and main advantage, is that it ties SMT research with existing research from algebraic and tropical geometry. We note that due to the work of researchers in tropical geometry, only the tropical polynomial formulation allows for search over multiple dimensions.

Another advantage is that the formulation makes the links between the existing tropical and MERT semirings somewhat obvious. For example, in Section 7.3.5 we were able to represent a WFST with tropical polynomial weights as a continuum of WFSTs with tropical weights.

We also note how concise the notation is, a tropical polynomial can be written and manipulated much like a classical polynomial. The other formulations require expressions of the shortest distance in terms of geometric objects, which results in difficult and clumsy notation.

One criticism of our formulation is that the geometric representations of semirings are somehow more 'intuitive' than the algebraic notation. To answer this criticism we note that the whole field of algebraic geometry is dedicated to finding common characteristics of algebra and geometry. Some problems which appear hard and obtuse in one domain can suddenly become simple and clear in the other. For example, it is not immediately obvious that the generalised times operation for a hypergraph should result in a Minkowski sum, but the multiplication of two polynomials immediately gives the desired quantity.

Chapter 8

Conclusion

Our goal throughout this thesis has been to take a fresh perspective on a familiar topic. From the review in Chapter 3 we can see that linear models are widely used in the statistical machine translation community. Yet despite their prevalence and their simplicity, we can still find many new interesting theorems and algorithms.

We now discuss the various results and ideas developed in this thesis. First we review the original contributions, the next section describes the publications and presentations resulting from the research, and finally we discuss extensions and future work.

8.1 Review of Work

This thesis developed a geometric formalism of minimum error rate training. Using this formalism, we were able to able to describe new algorithms, and investigate the properties of these algorithms. Finally, through the lens of this geometric formalism we were able to make important deductions about linear models in high dimensional spaces.

In contrast to the very theoretical nature of the work on linear models, we also presented a set of practical tips, tricks, and design heuristics for building an efficient and fast system that has strong performance in evaluations.

Much of our work is spent considering the benefits and disadvantages of formalism. In our work in linear models we believe that a rich formalism is needed to understand their properties. However, with MapReduce we argued that is important not to be a slave to a formalism that is not rich enough to capture the intricacies of the task at hand. Ultimately we believe that we should always start by understanding problems, and that solutions should be justified with respect to solving these problems.

8.1.1 Fast Model Parameter Estimation and Filtering

Chapter 4 separates model building into two separate tasks: estimation and filtering. We find that the HFile provides a solution that is 'good enough' for most parameter filtering tasks for SMT systems. We would recommend that when building new systems researchers consider a simple HFile based system and only reject HFiles if they do not meet specific requirements. For example in real time translation, where the whole filtering and decoding pipeline should execute in the space of a few seconds, the HFile may not be fast enough.

For parameter estimation we would still recommend using a MapReduce based solution, but with a caveat. That caveat is that not all problems fit neatly into the MapReduce formalism, and care should be taken that extra steps or inefficiencies are not being put into the system just to conform to this formalism. The key is to have a good understanding of the underlying model and the framework to craft a fast and efficient solution.

8.1.2 A Convex Geometric Description of MERT

In Section 3.6 we observed that the forms of linear models such as MERT, Structured SVMs, MIRA, and Pro yield similar levels of performance. Motivated by this survey, and the difficulty in achieving performance gains, we studied linear models under lens of the convex geometric description of MERT.

Using convex geometry we built a novel formalism of MERT. We described a geometric object called the normal fan, which provides a mapping of regions of parameter space to maximal feature vectors. Using the normal fan we were able to provide new insights into single sentence MERT, such as representing Och's line optimisation as a projection operation.

We were able to formalise multi-sentence MERT as the Minkowski sum of many polytopes and show that the vertices of the Minkowski sum can be enumerated in polynomial time. Using the upper bound theorem we found that the exponential explosion of vertices of the Minkowski sum is due to feature dimension, and not the number of sentences in the training set.

8.1.3 Projected MERT

Due to the exponential relationship between vertices in the Minkowski sum and feature dimension we should consider methods that reduce the feature dimension during optimisation. One method is Och's line optimisation, where we reduce the feature dimension to a two di-

mensional projected space. We extended this approach so that we can project to an arbitrary number of dimensions.

We have found that we can use an existing implementation of the Minkowski sum algorithm to computer the error surface in two dimensions. In its current state, this implementation is too slow for practical use. It does demonstrate that the algorithm is tractable and, with engineering effort, we could create a usable implementation.

8.1.4 Tropical Geometry MERT

We found that tropical geometry can be used to form a semiring for WFST operations. The tropical polynomial semiring can be used with the shortest distance algorithm to compute the upper envelope for MERT. We also saw that the tropical polynomial can be readily extended to many dimensions.

8.2 Future Work

We divide our discussion of future work in two. First, we discuss further improvements that can be made to model parameter estimation and filtering. Then we discuss the direction of research suggested by convex geometry.

For parameter estimation, we have seen that the MapReduce programming model struggles to represent many of the estimation techniques used in SMT. Ideally we want a framework that can abstract much of the engineering work through a rich programming model, allowing us to focus on the task of estimating parameters. We should investigate frameworks with richer programming models and believe that the Spark framework would be a good first step. It would be interesting to test whether this framework would allow the easy implementation of word alignment models, which require expectation-maximisation statistics to be shared between nodes, or lexical feature computation.

For model parameter filtering, we found the HFile to be a versatile and efficient method of storing model parameters. However, because industry continues to move quickly, we should continue to investigate new developments. For example, in Section 4.3 the motivation for using an HFile was that maintaining a dedicated HBase cluster was beyond the resources of most researchers. Recently industry has introduced cloud key-value stores based on either HBase [3, 106], or BigTable [69]. These key-value stores are constantly online and fully maintained, potentially making parameter filtering even easier.

We now move to a discussion of convex geometry. The scientific cliché found often in a

conclusion section is that more research is needed. Our analysis of linear models suggests the opposite, we believe that linear models have reached the limit of their effectiveness. Because the hypothesis space used for estimating linear model parameters is already so huge, it is unlikely that expanding it would result in a performance increase. The only other avenue for an improvement is to increase feature dimension, which as we have seen has many problems.

If linear models suffer from such severe limitations, then it is strange that they are so widely used in SMT. To understand their prevalence, we need to frame the discussion within the context of finite-state methods. We note that both hierarchical phrase-based models and language models have finite-state representations: the hypergraph and the WFSA. These representations are possible due to the independence constraints in the models, which means that the resulting log-probabilities are part of the tropical semiring. These finite-state methods allows for very large hypothesis spaces to be considered, which contribute greatly to strong modern systems. We note that MERT can be represented as a tropical polynomial semiring, which is generalisation of the tropical semiring. This suggests that linear models are popular because they can either be represented by finite-state methods or be easily converted to a finite-state representation by the means of an inner product.

The natural reaction to our work is to ask: what could replace linear models? One option would be to pursue a non-linear representation such as a neural network or a random decision forrest. We must be careful when selecting our non-linear representation. For example, we could use the kernel trick to introduce non-linearities into a support vector machine. However, these kernels result in an increase in feature dimension. The result is that more feature vectors are feasible.

Unfortunately, due to the restrictions enforced by the use of finite-state methods non-linear models, such as neural networks, are difficult to apply. One way forward may be to accept the limitations of linear models and include these non-linear models as features. For example, Devlin et al. [52] found impressive BLEU gains by using a neural network as a joint language model, which then formed a feature in an expected BLEU based system.

One issue blocking the adoption of projected MERT is that we have not solved the underlying issue with MERT. Namely, the problem of picking directions. We have recast the problem from picking directions to picking projections, which may allow for new solutions. For example, a dimensionality reduction scheme such as PCA could be used to select interesting affine subspaces.

Another avenue would be to consider many parameters when decoding. We note that given a parameter $\mathbf{w}^{(0)}$ and a set of directions, projected MERT can compute an error surface

for a projected parameter space. We should be able to refer to the entire error surface when decoding, as opposed to relying on a single parameter vector. Polytopes could be generated for test sentences, and their projected normal fans could be intersected with the error surface to find normal cones that contain consistently low errors. We admit that this method is highly speculative and would require the investment in building a good Minkowski sum implementation, but it should be tractable.

References

[1] Allauzen, C., Mohri, M., and Roark, B. (2003). Generalized algorithms for constructing statistical language models. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 40–47, Sapporo, Japan. Association for Computational Linguistics.

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

[3] Amazon.com, Inc. (2014). Store data with HBase - Amazon elastic mapreduce. `http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-hbase.html`.

[4] Apache Software Foundation (2014a). Apache Hadoop 2.4.1 - YARN. `http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html`.

[5] Apache Software Foundation (2014b). The Apache HBase reference guide. `http://hbase.apache.org/book/book.html`.

[6] Apache Software Foundation (2014c). Welcome to Apache Hadoop. `http://hadoop.apache.org`.

[7] Armstrong, J. L. and Virdin, S. R. (1990). Erlang - an experimental telephony programming language. In *In XIII International Switching Symposium*, pages 2–7.

[8] Automatic Language Processing Advisory Committee (1966). *Language and Machines: Computers in Translation and Linguistics*. The National Academies Press.

[9] Avis, D. and Fukuda, K. (1993). Reverse search for enumeration. *Discrete Applied Mathematics*, 65:21–46.

[10] Bahl, L. R., Jelinek, F., and Mercer, R. (1983). A maximum likelihood approach to continuous speech recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, pages 179–190.

[11] Banerjee, S. and Lavie, A. (2005). METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

[12] Barber, C. B., Dobkin, D. P., and Huhdanpaa, H. (1996). The Quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483.

[13] Bazrafshan, M., Chung, T., and Gildea, D. (2012). Tuning as linear regression. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 543–547, Montréal, Canada. Association for Computational Linguistics.

[14] Beck, K. and Andres, C. (2004). *Extreme Programming Explained: Embrace Change (2Nd Edition)*. Addison-Wesley Professional.

[15] Bellos, D. (2011). *Is That a Fish in Your Ear?: Translation and the Meaning of Everything*. Penguin Books Limited.

[16] Bender, O., Matusov, E., Hahn, S., Hasan, S., Khadivi, S., and Ney, H. (2007). The RWTH Arabic-to-English spoken language translation system. In *Automatic Speech Recognition Understanding*, pages 396 –401.

[17] Bennett, K. P. and Bredensteiner, E. J. (1997). Geometry in learning. In *In Geometry at Work*.

[18] Bennett, K. P. and Bredensteiner, E. J. (2000). Duality and geometry in SVM classifiers. In *ICML*, pages 57–64.

[19] Bentley, J. L. and Ottmann, T. A. (1979). Algorithms for reporting and counting geometric intersections. *Computers, IEEE Transactions on*, 100(9):643–647.

[20] Bishop, C. M. et al. (2006). *Pattern recognition and machine learning*, volume 1. springer New York.

[21] Blackwood, G. (2010). *Lattice Rescoring Methods for Statistical Machine Translation*. PhD thesis, University of Cambridge, Cambridge, United Kingdom.

[22] Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426.

[23] Blunsom, P., Cohn, T., and Osborne, M. (2008). A discriminative latent variable model for statistical machine translation. In *Proceedings of ACL-08: HLT*, pages 200–208, Columbus, Ohio. Association for Computational Linguistics.

[24] Bojar, O., Buck, C., Callison-Burch, C., Federmann, C., Haddow, B., Koehn, P., Monz, C., Post, M., Soricut, R., and Specia, L. (2013). Findings of the 2013 Workshop on Statistical Machine Translation. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 1–44, Sofia, Bulgaria. Association for Computational Linguistics.

[25] Brants, T., Popat, A. C., Xu, P., Och, F. J., and Dean, J. (2007). Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 858–867, Prague, Czech Republic. Association for Computational Linguistics.

[26] Brown, P. F., Pietra, V. J., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19:263–311.

[27] Callison-Burch, C., Bannard, C., and Schroeder, J. (2005). Scaling phrase-based statistical machine translation to larger corpora and longer phrases. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 255–262, Ann Arbor, Michigan. Association for Computational Linguistics.

[28] C.Clack, C.Myers, E. (1994). *Programming with Miranda*. Prentice Hall International.

[29] 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, Columbus, Ohio. Association for Computational Linguistics.

[30] Cettolo, M. F. M., Bentivogli, L., Paul, M., and Stüker, S. (2012). Overview of the iwslt 2012 evaluation campaign. *Proceedings IWSLT 2012*.

[31] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. (2008). BigTable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4.

[32] Chappelier, J.-C., Rajman, M., et al. (1998). A generalized CYK algorithm for parsing stochastic CFG. *TAPD*, 98:133–137.

[33] Chen, S. F. and Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393.

[34] Cherry, C. and Foster, G. (2012). Batch tuning strategies for statistical machine translation. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 427–436, Montréal, Canada. Association for Computational Linguistics.

[35] Chiang, D. (2005). A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 263–270, Ann Arbor, Michigan. Association for Computational Linguistics.

[36] Chiang, D. (2007). Hierarchical phrase-based translation. *Computational Linguistics*, 33.

[37] Chiang, D. (2012). Hope and fear for discriminative training of statistical translation models. *The Journal of Machine Learning Research*, 13(1):1159–1187.

[38] Chiang, D., DeNeefe, S., and Pust, M. (2011). Two easy improvements to lexical weighting. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 455–460, Portland, Oregon, USA. Association for Computational Linguistics.

[39] Chiang, D., Knight, K., and Wang, W. (2009). 11,001 new features for statistical machine translation. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 218–226, Boulder, Colorado. Association for Computational Linguistics.

[40] Chiang, D., Marton, Y., and Resnik, P. (2008). Online large-margin training of syntactic and structural translation features. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 224–233, Honolulu, Hawaii. Association for Computational Linguistics.

[41] Chomsky, N. (2002). *Syntactic Structures*. Mouton classic. Bod Third Party Titles.

[42] Connolly, T. M. and Begg, C. E. (2005). *Database systems: a practical approach to design, implementation, and management*. Pearson Education.

[43] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. MIT Press.

[44] Cox, D. A., Little, J., and O'Shea, D. (2007). *Ideals, varieties, and algorithms: an introduction to computational algebraic geometry and commutative algebra*. Springer.

[45] 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.

[46] de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. (2008). *Computational Geometry: Algorithms and Applications*. Springer-Verlag, third edition.

[47] de Gispert, A., Iglesias, G., Blackwood, G., Banga, E. R., and Byrne, W. (2010a). Hierarchical phrase-based translation with weighted finite-state transducers and shallow-n grammars. *Computational Linguistics*, 36(3):505–533.

[48] de Gispert, A., Pino, J., and Byrne, W. (2010b). Hierarchical phrase-based translation grammars extracted from alignment posterior probabilities. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 545–554, Cambridge, MA. Association for Computational Linguistics.

[49] Dean, J. and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

[50] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38.

[51] Deng, Y. and Byrne, W. (2005). HMM word and phrase alignment for statistical machine translation. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 169–176, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

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

[53] 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. Morgan Kaufmann Publishers Inc.

[54] Dyer, C. (2013). Minimum error rate training and the convex hull semiring. *CoRR*, abs/1307.3675.

[55] Dyer, C., Cordova, A., Mont, A., and Lin, J. (2008). Fast, easy, and cheap: Construction of statistical machine translation models with MapReduce. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 199–207, Columbus, Ohio. Association for Computational Linguistics.

[56] Dyer, C., Lopez, A., Ganitkevitch, J., Weese, J., Ture, F., Blunsom, P., Setiawan, H., Eidelman, V., and Resnik, P. (2010). cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of the ACL 2010 System Demonstrations*, pages 7–12, Uppsala, Sweden. Association for Computational Linguistics.

[57] Flanigan, J., Dyer, C., and Carbonell, J. (2013). Large-scale discriminative training for statistical machine translation using held-out line search. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 248–258, Atlanta, Georgia. Association for Computational Linguistics.

[58] Fredkin, E. (1960). Trie memory. *Communications of the ACM*, 3(9):490–499.

[59] Fukuda, K. (2004). From the zonotope construction to the Minkowski addition of convex polytopes. *Journal of Symbolic Computation*, 38(4):1261–1272.

[60] Fukuda, K. (2014). Frequently asked questions in polyhedral computation. `ftp://ftp.ifor.math.ethz.ch/pub/fukuda/reports/polyfaq040618.pdf`.

[61] Gale, W. A. and Church, K. W. (1993). A program for aligning sentences in bilingual corpora. *Computational linguistics*, 19(1):75–102.

[62] Galley, M. and Quirk, C. (2011). Optimal search for minimum error rate training. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 38–49, Edinburgh, Scotland, UK. Association for Computational Linguistics.

[63] Galley, M., Quirk, C., Cherry, C., and Toutanova, K. (2013). Regularized minimum error rate training. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1948–1959, Seattle, Washington, USA. Association for Computational Linguistics.

[64] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design patterns: elements of reusable object-oriented software*. Pearson Education.

[65] Ganitkevitch, J., Cao, Y., Weese, J., Post, M., and Callison-Burch, C. (2012). Joshua 4.0: Packing, pro, and paraphrases. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 283–291, Montréal, Canada. Association for Computational Linguistics.

[66] Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003). The Google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM.

[67] Gimpel, K. and Smith, N. A. (2012). Structured ramp loss minimization for machine translation. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 221–231, Montréal, Canada. Association for Computational Linguistics.

[68] Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3-4):237–264.

[69] Google Inc. (2014). Cloud datastore - NoSQL database for cloud data storage - google cloud platform. `https://cloud.google.com/datastore/`.

[70] Goyal, A., Daume III, H., and Venkatasubramanian, S. (2009). Streaming for large scale NLP: Language modeling. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 512–520, Boulder, Colorado. Association for Computational Linguistics.

[71] Green, S., Wang, S., Cer, D., and Manning, C. D. (2013). Fast and adaptive online training of feature-rich translation models. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 311–321, Sofia, Bulgaria. Association for Computational Linguistics.

[72] Gritzmann, P. and Sturmfels, B. (1992). Minkowski addition of polytopes: Computational complexity and applications to Gröbner bases. *SIAM Journal on Discrete Mathematics*, 6(2).

[73] Guthrie, D. and Hepple, M. (2010). Storing the web in memory: Space efficient language models with constant time retrieval. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 262–272, Cambridge, MA. Association for Computational Linguistics.

[74] Heafield, K. (2011). Kenlm: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland. Association for Computational Linguistics.

[75] Heafield, K., Pouzyrevsky, I., Clark, J. H., and Koehn, P. (2013). Scalable modified Kneser-Ney language model estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 690–696, Sofia, Bulgaria.

[76] Hopkins, M. and May, J. (2011). Tuning as ranking. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1352–1362, Edinburgh, Scotland, UK. Association for Computational Linguistics.

[77] Huang, L. (2008). Advanced dynamic programming in semiring and hypergraph frameworks. In *Coling 2008: Advanced Dynamic Programming in Computational Linguistics: Theory, Algorithms and Applications - Tutorial notes*, pages 1–18, Manchester, UK. Coling 2008 Organizing Committee.

[78] Huang, X., Acero, A., and Hon, H.-W. (2001). *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

[79] Iglesias, G., Allauzen, C., Byrne, W., de Gispert, A., and Riley, M. (2011). Hierarchical phrase-based translation representations. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1383, Edinburgh, Scotland, UK. Association for Computational Linguistics.

[80] Iglesias, G., de Gispert, A., Banga, E. R., and Byrne, W. (2009a). Rule filtering by pattern for efficient hierarchical translation. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 380–388, Athens, Greece. Association for Computational Linguistics.

[81] Iglesias, G., de Gispert, A., R. Banga, E., and Byrne, W. (2009b). Hierarchical phrase-based translation with weighted finite state transducers. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 433–441, Boulder, Colorado. Association for Computational Linguistics.

[82] Jelinek, F. (1997). *Statistical methods for speech recognition*. MIT Press, Cambridge, MA, USA.

[83] Jurafsky, D. and Martin, J. H. (2008). *Speech and Language Processing*. Prentice Hall, 2nd edition.

[84] Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM.

[85] Katz, S. M. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. In *IEEE Transactions on Acoustics, Speech and Signal Processing*, pages 400–401.

[86] Klemperer, P. (2010). The product-mix auction: A new auction design for differentiated goods. *Journal of the European Economic Association*, 8(2-3):526–536.

[87] Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86.

[88] Koehn, P. (2010). *Statistical Machine Translation*. Cambridge University Press.

[89] Koehn, P., Axelrod, A., Callison-Burch, C., Osborne, M., and Talbot, D. (2005). Edinburgh system description for the 2005 IWSLT speech translation evaluation. In *International Workshop on Spoken Language Translation, volume 8.*

[90] 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, Prague, Czech Republic. Association for Computational Linguistics.

[91] Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 48–54. Association for Computational Linguistics.

[92] Kumar, S. and Byrne, W. (2003). A weighted finite state transducer implementation of the alignment template model for statistical machine translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 63–70. Association for Computational Linguistics.

[93] Kumar, S. and Byrne, W. (2004). Minimum Bayes-risk decoding for statistical machine translation. In Susan Dumais, D. M. and Roukos, S., editors, *HLT-NAACL 2004: Main Proceedings*, pages 169–176, Boston, Massachusetts, USA. Association for Computational Linguistics.

[94] Kumar, S., Macherey, W., Dyer, C., and Och, F. (2009). Efficient minimum error rate training and minimum Bayes-risk decoding for translation hypergraphs and lattices. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 163–171, Suntec, Singapore. Association for Computational Linguistics.

[95] Levenberg, A. and Osborne, M. (2009). Stream-based randomised language models for SMT. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 756–764, Singapore. Association for Computational Linguistics.

[96] Lewis II, P. M. and Stearns, R. E. (1968). Syntax-directed transduction. *Journal of the ACM (JACM)*, 15(3):465–488.

[97] Li, Z., Callison-Burch, C., Dyer, C., Khudanpur, S., Schwartz, L., Thornton, W., Weese, J., and Zaidan, O. (2009). Joshua: An open source toolkit for parsing-based machine translation. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 135–139, Athens, Greece. Association for Computational Linguistics.

[98] Library of Parliament (2014). Historical debates of the parliament of canada. `http://parl.canadiana.ca`.

[99] Lin, J. and Dyer, C. (2010). *Data-intensive Text Processing with MapReduce*. Morgan & Claypool.

[100] Lopez, A. (2007). Hierarchical phrase-based translation with suffix arrays. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 976–985, Prague, Czech Republic. Association for Computational Linguistics.

[101] Luenberger, D. G. and Ye, Y. (2008). *Linear and nonlinear programming*, volume 116. Springer.

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

[103] Manber, U. and Myers, G. (1993). Suffix arrays: a new method for on-line string searches. *siam Journal on Computing*, 22(5):935–948.

[104] Manku, G. S. and Motwani, R. (2002). Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pages 346–357. VLDB Endowment.

[105] McDonald, R., Crammer, K., and Pereira, F. (2005). Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98, Ann Arbor, Michigan. Association for Computational Linguistics.

[106] Microsoft Corporation (2014). Get started using HBase with Hadoop in HDInsight | Azure. `http://azure.microsoft.com/en-us/documentation/articles/hdinsight-hbase-get-started/`.

[107] Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational linguistics*, 23(2):269–311.

[108] Mohri, M. (2002). Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350.

[109] Mohri, M. (2003). Edit-distance of weighted automata: General definitions and algorithms. *International Journal of Foundations of Computer Science*, 14(6):957–982.

[110] Mohri, M., Pereira, F. C. N., and Riley, M. (2008). Speech recognition with weighted finite-state transducers. *Handbook on Speech Processing and Speech Communication*.

[111] Mohri, M. and Riley, M. (2002). An efficient algorithm for the n-best-strings problem. In *Proceedings of the International Conference on Spoken Language Processing 2002*.

[112] Motzkin, T. S., Raiffa, H., Thompson, G. L., and Thrall, R. M. (1953). The double description method. In *Contributions to the Theory of Games, Vol. II (Kuhn, HW and AW Tucker, eds.), Annals of Math 28*, pages 81–103. Princeton University Press, Princeton.

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

[114] Och, F. J. and Ney, H. (2002). Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 295–302, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

[115] Pachter, L. and Sturmfels, B. (2004). Parametric inference for biological sequence analysis. *Proceedings of the National Academy of Sciences of the United States of America*, 101(46):16138–16143.

[116] Pachter, L. and Sturmfels, B. (2007). The mathematics of phylogenomics. *SIAM review*, 49(1):3–31.

[117] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

[118] Papineni, K. A. (1999). Discriminative training via linear programming. In *Acoustics, Speech, and Signal Processing, 1999. Proceedings., 1999 IEEE International Conference on*, volume 2, pages 561–564. IEEE.

[119] Parker, R., Graff, D., Kong, J., Chen, K., and Maeda, K. (2011). English Gigaword fifth edition. In *Linguistic Data Consortium, Phildelphia*.

[120] Patterson, D. A. and Hennessy, J. L. (2009). *Computer Organization and Design: The Hardware/software Interface*. Morgan Kaufmann.

[121] Pauls, A. and Klein, D. (2011). Faster and smaller n-gram language models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 258–267, Portland, Oregon, USA. Association for Computational Linguistics.

[122] Pino, J. (2014). *Refinements in Hierarchical Phrase-Based Translation*. PhD thesis, University of Cambridge, Cambridge, United Kingdom.

[123] Pino, J., Iglesias, G., de Gispert, A., Blackwood, G., Brunning, J., and Byrne, W. (2010). The cued hifst system for the wmt10 translation shared task. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, pages 155–160, Uppsala, Sweden. Association for Computational Linguistics.

[124] Pino, J., Waite, A., and Byrne, W. (2012). Simple and efficient model filtering in statistical machine translation. *The Prague Bulletin of Mathematical Linguistics*, 98:5–24.

[125] Pino, J., Waite, A., Xiao, T., de Gispert, A., Flego, F., and Byrne, W. (2013). The University of Cambridge Russian-English system at WMT13. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 200–205, Sofia, Bulgaria. Association for Computational Linguistics.

[126] Press, W. H., Vetterling, W. T., Teukolsky, S. A., and Flannery, B. P. (2002). *Numerical Recipes in C++: the art of scientific computing*. Cambridge University Press.

[127] Przywara, Č. and Bojar, O. (2011). eppex: Epochal phrase table extraction for statistical machine translation. *The Prague Bulletin of Mathematical Linguistics*, 96:89–98.

[128] Ratliff, N., Bagnell, J. A., and Zinkevich, M. (2006). Subgradient methods for maximum margin structured learning. In *ICML Workshop on Learning in Structured Output Spaces*, volume 46. Citeseer.

[129] Richter-Gebert, J., Sturmfels, B., and Theobald, T. (2005). First steps in tropical geometry. In *Idempotent mathematics and mathematical physics*.

[130] Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., and Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471.

[131] Simianer, P., Riezler, S., and Dyer, C. (2012). Joint feature selection in distributed stochastic learning for large-scale discriminative training in smt. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11–21, Jeju Island, Korea. Association for Computational Linguistics.

[132] Smith, D. A. and Eisner, J. (2006). Minimum risk annealing for training log-linear models. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 787–794, Sydney, Australia. Association for Computational Linguistics.

[133] Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Makhoul, J. (2006). A study of translation edit rate with targeted human annotation. In *Proceedings of association for machine translation in the Americas*, pages 223–231.

[134] Sokolov, A. and Yvon, F. (2011). Minimum error rate training semiring. In *Proceedings of the European Association for Machine Translation*.

[135] Speyer, D. and Sturmfels, B. (2009). Tropical mathematics. *Mathematics Magazine*.

[136] Stolcke, A. (2000). Entropy-based pruning of backoff language models. *CoRR*, cs.CL/0006025.

[137] Talbot, D. and Brants, T. (2008). Randomized language models via perfect hash functions. In *Proceedings of ACL-08: HLT*, pages 505–513, Columbus, Ohio. Association for Computational Linguistics.

[138] Talbot, D. and Osborne, M. (2007a). Randomised language modelling for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 512–519, Prague, Czech Republic. Association for Computational Linguistics.

[139] Talbot, D. and Osborne, M. (2007b). Smoothed Bloom filter language models: Tera-scale LMs on the cheap. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 468–476, Prague, Czech Republic. Association for Computational Linguistics.

[140] Taskar, B., Guestrin, C., and Koller, D. (2004a). Max-margin Markov networks. *Advances in neural information processing systems*, 16:25.

[141] Taskar, B., Klein, D., Collins, M., Koller, D., and Manning, C. (2004b). Max-margin parsing. In Lin, D. and Wu, D., editors, *Proceedings of EMNLP 2004*, pages 1–8, Barcelona, Spain. Association for Computational Linguistics.

[142] The Linux Documentation Project (2014). The buffer cache. `http://www.tldp.org/LDP/sag/html/buffer-cache.html`.

[143] Tromble, R., Kumar, S., Och, F., and Macherey, W. (2008). Lattice Minimum Bayes-Risk decoding for statistical machine translation. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 620–629, Honolulu, Hawaii. Association for Computational Linguistics.

[144] Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, page 104. ACM.

[145] Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large margin methods for structured and interdependent output variables. In *Journal of Machine Learning Research*, pages 1453–1484.

[146] Ueffing, N., Och, F. J., and Ney, H. (2002). Generation of word graphs in statistical machine translation. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 156–163. Association for Computational Linguistics.

[147] Venugopal, A. and Zollmann, A. (2009). Grammar based statistical mt on hadoop. *The Prague Bulletin of Mathematical Linguistics*, 91.

[148] Vogel, S., Ney, H., and Tillmann, C. (1996). HMM-based word alignment in statistical translation. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 2*, COLING '96, pages 836–841, Stroudsburg, PA, USA. Association for Computational Linguistics.

[149] Waite, A., Blackwood, G., and Byrne, W. (2012). Lattice-based minimum error rate training using weighted finite-state transducers with tropical polynomial weights. In *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing*, pages 116–125, Donostia–San SebastiÃąn. Association for Computational Linguistics.

[150] Watanabe, T. (2012). Optimized online rank learning for machine translation. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 253–262, Montréal, Canada. Association for Computational Linguistics.

[151] Watanabe, T., Suzuki, J., Tsukada, H., and Isozaki, H. (2007). Online large-margin training for statistical machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 764–773.

[152] Weese, J., Ganitkevitch, J., Callison-Burch, C., Post, M., and Lopez, A. (2011). Joshua 3.0: Syntax-based machine translation with the Thrax grammar extractor. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 478–484, Edinburgh, Scotland. Association for Computational Linguistics.

[153] Weibel, C. (2010). Implementation and parallelization of a reverse-search algorithm for minkowski sums. In *In Proceedings of the 12th Workshop on Algorithm Engineering and Experiments (ALENEX 2010)*, pages 34–42. SIAM.

[154] Woods, W. A. (1970). Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606.

[155] Yu, X. (2008). Estimating language models using hadoop and hbase. Master's thesis, University of Edinburgh, Edinburgh.

[156] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10.

[157] Zens, R., Hasan, S., and Ney, H. (2007). A systematic comparison of training criteria for statistical machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 524–532, Prague, Czech Republic. Association for Computational Linguistics.

[158] Zens, R. and Ney, H. (2007). Efficient phrase-table representation for machine translation with applications to online MT and speech translation. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 492–499, Rochester, New York. Association for Computational Linguistics.

[159] Zhang, Y., Hildebrand, A. S., and Vogel, S. (2006). Distributed language modeling for n-best list re-ranking. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 216–223, Sydney, Australia. Association for Computational Linguistics.

[160] Zhang, Y. and Vogel, S. (2005). An efficient phrase-to-phrase alignment model for arbitrarily long phrase and large corpora. In *Proceedings of the 10th Conference of the European Association for Machine Translation (EAMT-05)*, pages 294–301.

[161] Zhang, Y. and Vogel, S. (2006). Suffix array and its applications in empirical natural language processing. Technical Report CMU-LTI-06-010, Language Technologies Institute, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA.

[162] Ziegler, G. (1995). *Lectures on Polytopes*. Springer-Verlag.