Non-Linear Learning via Feature Induction in Statistical Machine Tranlsation

Jonathan H. Clark
Language Technologies Institute
School of Computer Science
Carnegie Mellon University

Ph.D. Thesis Proposal
March 30, 2012

Thesis Committee:

Alon Lavie (chair), Carnegie Mellon University
Jaime Carbonell, Carnegie Mellon University
Chris Dyer, Carnegie Mellon University
Noah Smith, Carnegie Mellon University
Wolfgang Macherey, Google, Inc.

Abstract

Many popular machine learning algorithms used in statistical machine translation minimize a loss function representative of translation quality by fitting a linear model comprised of a real-valued weight vector associated with a static set of features. However, the combination of such a simple learning technique and such a simple model means the overall learning process may ignore useful non-linear information present in the feature set. This places significant and undue burden on those tasked with improving translation systems to carefully design features that fall within the constraints of linear models – a highly unintuitive and error-prone task, as we will see.

We propose to relax three constraints of linear models by abandoning a static set of features and dynamically expanding the feature set during learning. We first show that by expanding real-valued features into a finite set of indicator features, we can achieve non-linear transformations of individual features. This transformation enables us to recover a translation system performing on par with the state-of-the-art without the the usual (arbitrary) transformation from probabilities to log-probabilities – in fact, we out-perform the state-of-the-art, demonstrating that the customary log transform is not always optimal. Second, we show that by expanding the feature set with conjunctions of indicator features, we can learn interactions among multiple features. These two techniques are not mutually exclusive in that indicator features resulting from the expansion of real-valued features can also participate in conjunctions. Further, these techniques produce interpretable models, which provide valuable insight into how the learner chose to fit the data, exposing which non-linearities are important. In the course of this thesis, the effectiveness of these techniques will be explored through several applications in machine translation using at least three language pairs. Experiments so far on a large scale Arabic→English task have shown gains of up to 2.4 BLEU using discretization and 1.2 BLEU using conjunctions.

Contents

Chapter 1

Background

1.1 A Brief History of Modeling in SMT

We begin by reviewing how prevailing models for SMT evolved with an emphasis on how they have become more general and which assumptions have fallen away – and in doing so we will reveal some historical conventions that have remained despite current advances. These historical conventions include the use of a log transform to convert translation model probabilities into features and the summarization of monolingual language modeling statistics into a single feature. We will discuss the original generative Brown model, Och's log-linear model optimized toward corpus probability, and Och's linear model directly optimized toward an arbitary objective function (e.g. BLEU). Finally, we will touch on recent work that seeks to handle large numbers of features while optimizing toward a proxy objective function (e.g. MIRA and PRO).

Brown et al. (1993) present a probabalistic model of translation, which defines the probability of an English translation $\mathbf{e}$ of a foreign sentence $\mathbf{f}$ under the distribution $p(\mathbf{e}|\mathbf{f})$. This can be expanded according to Bayes' Rule, consistent with their source-channel interpretation of translation:

$$P(\mathbf{e}|\mathbf{f}) = \frac{P(\mathbf{e})P(\mathbf{f}|\mathbf{e})}{P(\mathbf{f})} \tag{1.1}$$

The Brown paper then quickly moves on to transform this into the "fundamental equation of statistical machine translation", which shows the inference-time decision rule that seeks the most likely English translation $\hat{\mathbf{e}}$:

$$\hat{\mathbf{e}}(\mathbf{f}) = \operatorname*{argmax}_{\mathbf{e}} P(\mathbf{e})P(\mathbf{f}|\mathbf{e}) \tag{1.2}$$

Notice that the denominator $P(\mathbf{f})$ falls away as it is a constant for each $\mathbf{f}$. $P(\mathbf{e})$ corresponds to a language model and $P(\mathbf{f}|\mathbf{e})$ corresponds to a translation model (in reverse, as explained by the noisy-channel model). From a modern perspective, this model "weights" each of these component models equally. While this may empirically be suboptimal because of model error or estimation error, it is theoretically optimal under Bayes' Rule. Importantly, it allows the translation model and language model to be trained separately.

Och and Ney (2001)[1] move away from this generative source-channel approach in favor of a log-linear model with features $\mathbf{H}$, corresponding weights $\mathbf{w}$, and a latent variable aligning phrases $\mathbf{a}$:[2]

$$P(\mathbf{e}, \mathbf{a}|\mathbf{f}) = \frac{\exp \sum_{i=0}^{|\mathbf{H}|} w_i H_i(\mathbf{f}, \mathbf{a}, \mathbf{e})}{Z(\mathbf{f})} \tag{1.3}$$

where $Z(\mathbf{f})$ is a normalizer, which sums over all possible translations of $\mathbf{f}$ as generated by the function GEN, ensuring that each conditional distribution sums to unity:

$$Z(\mathbf{f}) = \sum_{\mathbf{e}', \mathbf{a}' \in \texttt{GEN}(\mathbf{f})} \exp \sum_{i=0}^{|\mathbf{H}|} w_i H_i(\mathbf{f}, \mathbf{a}', \mathbf{e}') \tag{1.4}$$

This exponential formulation has two major advantages:

1. The model allows the system designer to add arbitrary features that provide the model with faceted information about translation hypotheses

2. It introduces optimizable parameters $\mathbf{w}$ for each of the features in a principled way

Och so notes that the Brown model does not account for the translation model and language model being imperfectly estimated and that we can correct for this in the overall model by weighting the components. Conveniently, this log-linear model generalizes the source-channel model in that we can recover Brown's model by using the following feature set (Och and Ney, 2001):

$$h_{LM}(\mathbf{f}, \mathbf{a}, \mathbf{e}) = \log P(\mathbf{e}) \tag{1.5}$$

$$h_{TM}(\mathbf{f}, \mathbf{a}, \mathbf{e}) = \log P_{\mathbf{a}}(\mathbf{f}|\mathbf{e}) \tag{1.6}$$

Och also noted that comparable performance could be achieved by using the feature $h_{TM}(\mathbf{f}, \mathbf{a}, \mathbf{e}) = \log P_{\mathbf{a}}(\mathbf{e}|\mathbf{f})$, which is further evidence in support of this feature-based view of translation over the source-channel model.

Besides the change in form, an additional purpose has been assigned to the model's definition: since Och's model (Equation 1.3) has the free parameters $\mathbf{w}$, which must be tuned, it was also used as an objective function for optimization using Generalized Iterative Scaling (GIS). Further, the features used in this model are perhaps unusual as they are real-valued; whereas log-linear models are typically defined as having real-valued features, in practice it is more common for only indicator features to be used.

[1] Papineni, Roukos, and Ward (1998) had previously proposed such an exponential model with discriminative training, but in the context of translating from a natural language to a formal language.

[2] Och and Ney (2001) originally present these equations without the latent alignment variable $\mathbf{a}$. We present it here for completeness.

With Och's log-linear model, we see that there are now two layers of optimization:

- first, the translation model parameters and language model parameters are each estimated separately using relative frequencies; and
- second, the parameters of the log-linear model $\mathbf{w}$ are optimized toward Equation 1.3 (holding the parameters of the translation model and language model fixed). Note that in this context, the translation model and language model have been re-cast as real-valued features.

As we did with the Brown model, we can remove all terms that are constant with regard to $\mathbf{f}$ to obtain the model's inference-time decision rule:[3]

$$\hat{\mathbf{e}}(\mathbf{f}) = \underset{\mathbf{a},\mathbf{e}}{\operatorname{argmax}} \sum_{i=0}^{|\mathbf{H}|} w_i H_i(\mathbf{f}, \mathbf{a}, \mathbf{e}) \tag{1.7}$$

It is perhaps striking how much simpler the decision rule is versus the overall model's objective function. Another disconnect in this log-linear setup is that the objective function used during optimization (likelihood) is entirely different than the evaluation function (e.g. BLEU). This was motivation for Minimum Error Rate Training (MERT) (Och, 2003), which directly minimizes the number of errors produced over a set of parallel tuning sentences $\mathbf{S}$ by a model with a particular set of weights $\mathbf{w}$ according to an arbitrary error function E, using a set of reference translations $\mathbf{r}$:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{(\mathbf{f},\mathbf{r})\in\mathbf{S}} E\left(\mathbf{r}, \hat{\mathbf{e}}(\mathbf{f}; \mathbf{w})\right) \tag{1.8}$$

Och (2003) present MERT as an optimizer for the log-linear model of Equation 1.3, and in fact the decision rule remains the same as Equation 1.7. However, when using MERT as an optimizer, the relation to Equation 1.3 is perhaps more for historical reasons – in fact, MERT is simply optimizing a linear model (which also happens to share the decision rule of Equation 1.7:

$$\text{score}(\mathbf{f}, \mathbf{a}, \mathbf{e}) = \sum_{i=0}^{|\mathbf{H}|} w_i H_i(\mathbf{f}, \mathbf{a}, \mathbf{e}) \tag{1.9}$$

With this view the term log-linear becomes overly specific. Throughout the rest of this work, we will refer to this more generally as a linear model.

MERT no longer compells us to believe that an exponential model such as Equation 1.3 is a good measure of translation quality. In this context, the log transform of probability models into features begins to seem arbitrary: Since we are no longer tied to the Brown model nor the log-linear model of translation, we are left with little theoretical justification for our log transform. In fact, many systems actually use raw probabilities as features (Gimpel and Smith, 2009). Yet it has been observed empirically that many common

[3]Notice that decoding uses the Viterbi derivation (as denoted by $\mathbf{a}$ being included in argmax) rather than the Viterbi target string (which would correspond to summing over all $\mathbf{a}$). This is common to most modern decoders due to the burdens of non-local features.

features produce better translation quality in log space rather than probability space. However, the question still remains whether a log transform is optimal. We will return to this question in Section 2.1.

MERT also no longer compells us to believe that an exponential model such as Equation 1.3 is a good measure of translation quality. However, considering that some features are actually real-valued models themselves (Section 2.4), it does this only in the context of the *second* stage of optimization. These component models are trained as first stage of parameter estimation toward some proxy estimator such as maximum likelihood estimates (phrase tables) or perplexity (language models) – this is almost certainly not optimal in the context of the final objective function used in MERT. This shortcoming is primarily a symptom of the component models hiding a large number of statistics from the MERT procedure. In some sense, this is desireable since MERT only scales to a small number of features (Hopkins and May, 2011) and so summarizing a large number of features into a single real-value makes MERT more efficient and less prone to overfitting. However, to be optimal in terms of MERT's objective function, the model score (which summarizes these many statistics) must satisfy several properties (Section 2.1). In practice, satisfying these properties can be very difficult. Further, it can even be difficult to know *if* these properties are satisfied.

Recent work in optimization for SMT has focused on enabling the use of more features (Shen, Va, and Rey, 2004; Liang et al., 2006; Chiang, Marton, and Resnik, 2008; Hopkins and May, 2011). These typically trade some approximation of the objective function for the ability to scale to large feature sets. We will return to such optimizers in Section 1.4. With the general ability to add many features to models, this brings into question a second historical practice: previously, many of the free parameters of child models were hidden from the second-stage optimizer (MERT) via a first-stage optimization procedure; why should we still continue to optimize these parameters toward a first-pass proxy objective function such as likelihood when we now have the ability to jointly optimize them toward a more reliable objective?

1.2 The Anatomy of the Feature Space

In this section, we briefly define attributes of features that we will reference throughout the rest of this proposal. First, we define three types of **feature visibility**:

- **Observable features**. Direct attributes of the source input (e.g. sentence-level source language model, source LDA topic)
- **Output features**. Attributes of the model's output (e.g. language model, word penalty, lexical probabilities)
- **Latent features**. Not visible in either the input nor output, but only in latent variables of the model (e.g. segmentation, source tree, target tree, entropy, source phrase count, language model probability of a particular phrase, $P_{phr}(T|S)$)

Together, we refer to the output features and latent features as **non-observable features**.

We also define two types of **feature locality**:

- **Local features**. Stateless features (e.g. phrase features) that do not require additional state information in the dynamic program
- **Non-local features**. Stateful features (e.g. target language model) that do require additional state information in the dynamic program – these will always be hypothesis or latent features

Feature locality is relative to the way in which the inference method of a decoder is factored. For example, in a phrase-based decoder, any feature that is computable given only a single phrase pair or the input sentence is local, while features that cross phrase boundaries are non-local. Similarly, in a chart-based decoder, local features must be computable from a single grammar rule or the input sentence. Factoring features in this way allows for efficient inference by allowing search to be decomposed into smaller reusable subproblems in the spirit of dynamic programming.[4]

Next, we present the features of a standard machine translation system using the terminology of markov random fields (MRF) in Figure 1.1 (Smith, 2011, pp. 27-29). As the edges in this graph are undirected, this should not be interpreted as a generative story nor a particular ordering of events. In this graphical model, the source sentence $\mathbf{f}$ is fully observed. We then choose a derivation D of this sentence including how to segment the phrases in the case of a phrase-based system or which grammar rules to apply in the case of a syntactic system. We also assign the task of performing any reordering to D. Note that D is both latent and structured. For each the n phrases (or grammar rules) chosen by D, each phrase choses to generate some number of target terminals m_i. Finally, we choose which target terminal belongs in at e_i. The feasible assignment to each of these random variables is heavily constrained by the grammar rules or phrase table, as well as, the reordering hyperparameters of the system.

Under this view, we can now more precisely identify what information each feature function requires. Since the factorization of a MRF is a priori ambiguous, we organize our features over vertices $\mathbf{V}$ into **compatibility functions**[5] $\Psi_{\mathbf{v}} \in \mathcal{F}(\mathbf{d})$ where $\mathbf{v} \subseteq \mathbf{V}$ defines the scope of the compatibility function and $\mathcal{F}$ is responsible for taking a derivation $\mathbf{d}$ and factorizing it into compatibility functions (Sutton and Mccallum, 2010). In Figure 1.1, each compatibility function is represented by a small square box. In the abstract, a compatibility function is composed of several feature functions,[6] which may be applied to any random variables of the correct type. When we instantiate this graphical model for particular input sentence, then each compatibility function is bound to a particular instances of its random variables and its constituent feature functions are likewise bound to a subset of those random variable instances.

We divide the popular features of machine translation into two abstract compatibility functions: phrase and rule-local features Ψ^{Local} (e.g. phrase translation probabilities and lexical probabilities) and Ψ^{LM}, which includes only the language model. As shown in Figure 1.1, the rule-local features are distinguished from the LM features in that their compatibility functions never cross a rule boundary, making them amenable to dynamic programming. Because different classes of feature functions require different information, they bear different function signatures including $h_{\text{Phrase}}(\mathbf{f}, r, e_1 \ldots, e_k)$ and $h_{\text{BigramLM}}(e_1, e_2)$. When speaking about features in general without regard to a specic feature class, we denote the feature function as taking the random variables $\mathbf{v}$ of its compatibility function as an argument $h(\mathbf{v}), h \in \Psi_{\mathbf{v}}$.

[4]Due to the presence of non-local features, we cannot actually perform dynamic programming since the best solution to each subproblem is not guaranteed to be globally optimal. This is, in fact, the primary reason that machine translation decoders typically use inexact inference.

[5]The compatibility functions Ψ are also frequently called factors. However, we avoid this terminology due to the alternative meaning of the word factor in the machine translation community.

[6]Feature functions are also sometimes referred to as potential functions in the machine learning literature.

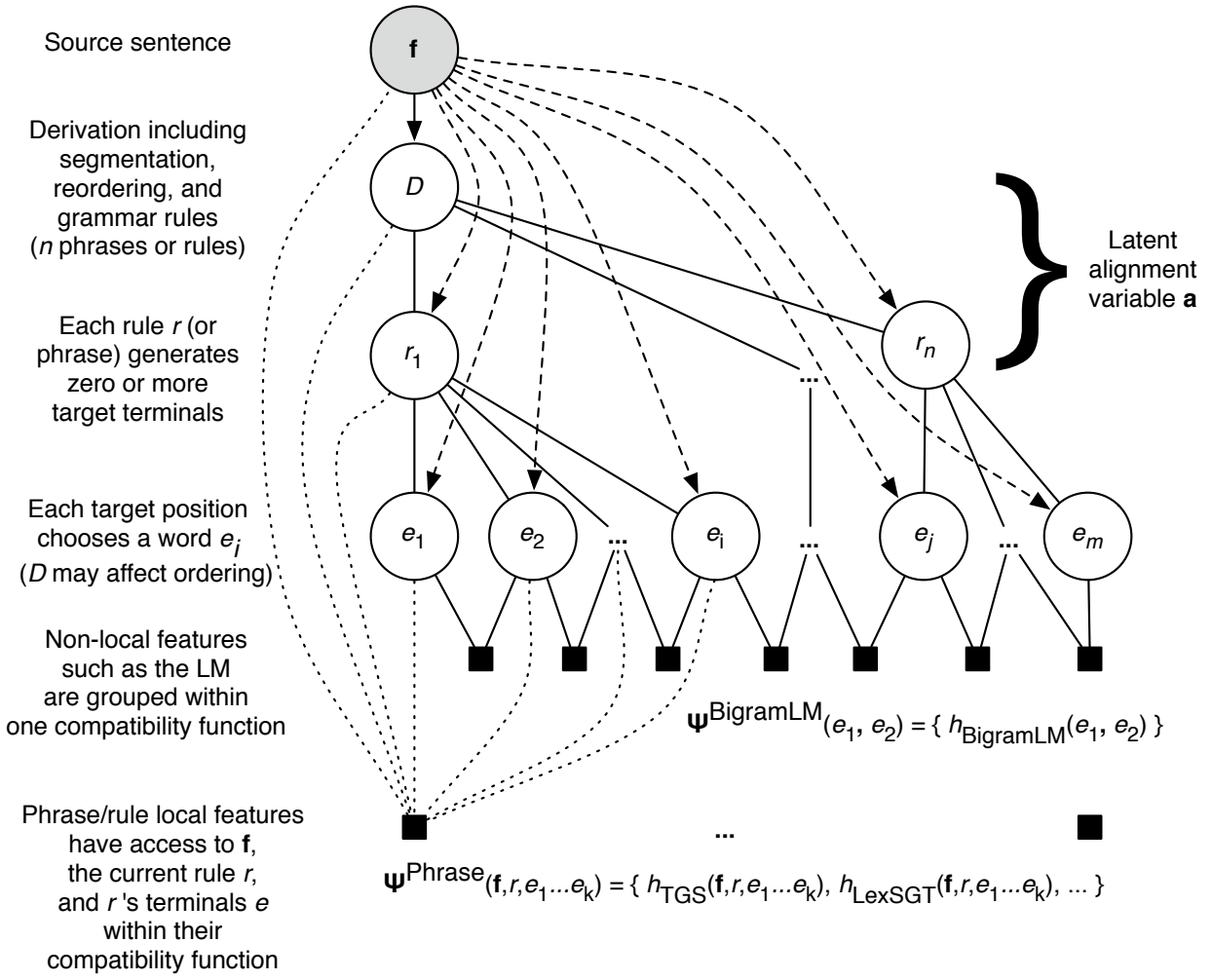

Figure 1.1: The features of a typical statistical machine translation visualized as a graphical model. We use factor graph notation to indicate the compatibility functions for phrase-local and rule-local features $\Psi^{\text{Phrase}}(\mathbf{f}, r, e_1 \dots e_k)$ and by a non-local bigram language model feature $\Psi^{\text{BigramLM}}(e_1, e_2)$. Local features are distinguished from non-local features in that they do not cross any rule boundaries. We use D to indicate a structured random variable over the derivation including segmentation, choice of grammar rules (or source phrases), and reordering constraints. We use a bigram LM as an example here for simplicity – in our experiments, we use larger LMs. Lines are dashed solely for readability.

1.3 Inference Framework: Linear Models in Structured Prediction

We have already presented the decision rule of linear models used by many MT systems in Equation 1.7:

$$\hat{\mathbf{e}}(\mathbf{f}) = \operatorname*{argmax}_{\mathbf{a},\mathbf{e}} \sum_{i=0}^{|\mathbf{H}|} w_i H_i(\mathbf{f}, \mathbf{a}, \mathbf{e})$$

However, this hides many of the realities of performing inference in modern decoders. Inference would be intractable if every feature were allowed access to the entire alignment $\mathbf{a}$ and the entire target hypothesis $\mathbf{e}$. In this section, we refine this decision rule using the fine-grained distinctions among features presented in the previous section.

First, we define how the inference mechanism aggregates the feature values within each compatibility function, as introduced in the previous section. Let "$i \in \Psi_{\mathbf{v}}$" indicate iteration over the feature indices i in the compatibility function $\Psi_{\mathbf{v}}$ composed of the random variables $\mathbf{v}$. Then our linear model aggregates the compatibility function as:

$$\Psi_{\mathbf{v}}(\mathbf{v}) = \sum_{i \in \Psi_{\mathbf{v}}} w_i h_i(\mathbf{v}) \tag{1.10}$$

We pass $\mathbf{v}$ as an argument to indicate that this is a call to Ψ returning a value, not a definition of the members of Ψ.

This then allows us to rewrite the decision rule of Equation 1.7 in terms of these compatibility functions $\Psi_{\mathbf{v}}$ resulting from the factorization of the derivation $\mathbf{d}$ according to $\mathcal{F}$:

$$\hat{\mathbf{e}}(\mathbf{f}) = \operatorname*{argmax}_{\mathbf{d},\mathbf{e}} \sum_{\Psi_{\mathbf{v}} \in \mathcal{F}(\mathbf{d})} \Psi_{\mathbf{v}}(\mathbf{v}) \tag{1.11}$$

$$= \operatorname*{argmax}_{\mathbf{d},\mathbf{e}} \sum_{\Psi_{\mathbf{v}} \in \mathcal{F}(\mathbf{d})} \underbrace{\sum_{i \in \Psi_{\mathbf{v}}} w_i h_i(\mathbf{v})}_{\Psi_{\mathbf{v}}(\mathbf{v})} \tag{1.12}$$

This view of the inference problem provides us with a clear picture of how decoders make inference efficient. The decoder can aggregate the score from each of these compatibility functions once per partial hypothesis. Since many of these partial hypotheses are shared among larger hypotheses, these costs can be amortized in a dynamic programming fashion. In practice, the scope of the language model feature is too broad to allow dynamic programming to be efficient, and so the inference problem is organized such that it factors over rules or phrases. This implies that some features are local (the rule-local features in Ψ^{Phrase}) and others have now become non-local (the LM features in Ψ^{LM}). It also implies that exact dynamic programming is no longer possible. Instead, approximate inference is typically carried out using a modified beam search that includes cube pruning as a nested search problem over which partial hypotheses should be considered for combination (Huang and Chiang, 2007).

With the introduction of this approximate inference to accomodate non-local features, we are now left with the possibility of **search errors**, which can arise when a poor decision early in the search procedure discards a partial hypothesis that would later have participated in the hypothesis with the best model score (Auli et al., 2009). To mitigate (but certainly not overcome) this danger, decoders require non-local feature

functions to perform **future cost estimation**[7] in which features provide an estimate of their value once the full hypothesis has been completed, but within the context of only a partial hypothesis (Moore and Quirk, 2007; Zens, 2008; Zens and Ney, 2008). Alternatively, this can be viewed as a mechanism of making the scores of partial hypotheses within the same search stack comparable so that the decoder does not opportunistically make local choices that will inevitably lead to worse translations in a larger context.

While this view from the perspective of compatibility functions is useful for understanding the efficiency of inference (which will become important in Section 4.1), it obscures the behavior of the overall model. To shed light on the high-level behavior of the model, we first define two stages of **feature vector aggregation**:

- the Ψ**-local feature vector**, denoted $\mathbf{h}$, which is the feature vector returned by a local or non-local feature as evaluated within its compatibility function Ψ; and
- the **global feature vector**, denoted $\mathbf{H}$, which is the feature vector obtained by the decoder by summing the Ψ-local feature vectors.

The inference mechanism computes the value of a global feature $H(\mathbf{d})$ for some derivation $\mathbf{d}$ by summing over all instances of Ψ-local features across the various compatibility functions Ψ:

$$H_i(\mathbf{d}) = \sum_{\Psi_\mathbf{v} \in \mathcal{F}(\mathbf{d})} h_i(\mathbf{v}) \tag{1.13}$$

If we view the inference problem as hypergraph decoding, then we can view the local features Ψ_{Local} as decomposing additively over hyperedges (Dyer, 2010, p. 57).

Finally, substituting this definition of the global feature vector back into our original decision rule, we obtain this high-level view of our linear model:

$$\hat{\mathbf{e}}(\mathbf{f}) = \operatorname*{argmax}_{\mathbf{d},\mathbf{e}} \sum_{i=0}^{|\mathbf{H}|} w_i \underbrace{\sum_{\Psi_\mathbf{v} \in \mathcal{F}(\mathbf{d})} h_i(\mathbf{v})}_{H_i(\mathbf{d})} \tag{1.14}$$

From this perspective, we see a clear distinction between the global feature H, which is important from the perspective of the linear model, and the myopic Ψ-local feature h, which is often how features are formally described. This distinction will become important in Chapter 3.

1.4 Learning Framework: Pairwise Ranking Optimization

As types of feature induction, discretization and conjunction will both result in much larger feature sets to be optimized. While MERT has proven to be a strong baseline, it does not scale to larger feature sets in terms of both inefficiency and overfitting. While MIRA has been shown to be effective over larger feature sets, as an on-line margin learning algorithm, it is more difficult to regularize – this will become important for neighbor regularization in Section 3.3. Therefore, we use the PRO optimizer as our baseline learner. PRO works by sampling pairs of hypotheses from the decoder's k-best list and then providing these pairs as a binary training examples (correctly ranked or not) to a standard binary classifier to obtain a new set of weights for the linear model. In our case, this classifier is trained using L-BFGS. This procedure of sampling training pairs and

[7] Future cost estimation is also often referred to as rest cost estimation.

then optimizing the pairwise rankings is repeated for a specified number of iterations. It has been shown to perform comparably to MERT for small number of features, and to significantly outperform MERT for large number of features (Hopkins and May, 2011).

Chapter 2

Overview

2.1 Theoretical Motivation: Assumptions and Pitfalls of Linear Modeling

We have reviewed how linear models have emerged as the dominant models of statistical machine translation. While these models have allowed much progress in the field, they are not without their limitations. We follow Nguyen et al. (2007) in enumerating the restrictions placed upon a feature set within a linear model, which hold whenever a linear model is used as an inference mechanism:[1]

1. **Linearity.** A feature must be linearly correlated with the objective function. That is, there should be no region that matters less than other regions. For example, this forces a system designer to decide use between $\log P(\mathbf{e})$ and $P(\mathbf{e})$.

2. **Monotonicity.** A feature value of x_1 for a hypothesis h_1 must indicate that h_1 is better than any other hypothesis h_i having a lesser value of that feature $x_i < x_1$. For example, consider a simple length penalty implemented as a *signed* difference: $\texttt{WordCount}(\mathbf{f}) - \texttt{WordCount}(\mathbf{e})$. To meet the monotonicity requirement, a system designer must use the absolute value to indicate that deviating from the length of the source sentence in *either* direction is bad.

3. **Independence.** Inter-dependence among features is ignored by the model. That is, a feature f_1 may not contribute more toward the model score in the presence of another feature f_2 than when f_2 is not active. For example, if a system has 3 language models, each trained on a different genre, each of those language models contributes equally to a hypothesis' score, regardless of the genre of the source sentence.

Linearity and monotonicity are perhaps most important when considering real-valued features as they are trivially satisfied for indicator features.

Independence is not strictly a *requirement* of a linear model nor most popular learners used in SMT including MERT, MIRA, and PRO. That is, the optimization procedure will not perform worse when dependent features are added to an otherwise independent feature set. However, when inter-dependence exists between features, the model cannot take advantage of this information. Independence manifests itself differently for real-valued versus indicator features. For indicator features (the boolean case), the classic example of a function that cannot be captured by a linear model is XOR, which is shown in Figures 2.1 and 2.2.

[1] They actually present this in the context of a log-linear model. However, these properties also hold for linear models.

Yet for such simple boolean features, it can be fairly easy for a human engineer to design such features. However, when considering real-valued features, interactions between features become much less intuitive. Interactions between real-valued features manifest when particular values of multiple features *together* indicate that a hypothesis has a different quality than a simple weighted sum of those values, as shown in Figures 2.3 and 2.4. Concrete applications of this idea in MT are described in Chapter 5.

	f_1=0	f_1=1
f_2=0	0	1
f_2=1	1	2

Figure 2.1: An example of how a linear model combines indicator features using the weights w_1=w_2=1

	f_1=0	f_1=1
f_2=0	0	1
f_2=1	1	**0**

Figure 2.2: An example of the XOR function. Notice that there exists no weight vector for a linear model that can produce this outcome.

	f_1=0.1	f_1=0.2	...
f_2=0.1	0.2	0.3	...
f_2=0.2	0.3	0.4	...
...			

Figure 2.3: A feasible scenario in a linear model. A linear model can produece the outcomes in this table given the weights w_1=w_2=1

	f_1=0.1	f_1=0.2	...
f_2=0.1	0.2	0.3	...
f_2=0.2	0.3	**0.5**	...
...			

Figure 2.4: A model that allows inter-dependence between features. There exists no set of weights $\mathbf{w}$ such that $\mathbf{w} \cdot \mathbf{f}$ will produce the values in this table, indicating that it is not possible in a linear model.

2.2 Practical Motivation: A View from Feature Engineering

We have pointed out several theoretical limitations of linear models. We now turn our attention to how these restrictions can negatively impact the development of statistical machine translation systems. When constructing MT systems, system developers often wish to improve system performance via feature Engineering, the process of designing features to contribute to the system's predictive model.

A common development pattern when improving SMT systems is to 1) create a new translation feature that (hopefully) predicts some desirable (or undesirable) facet of translations and then 2) expose this as a new feature whose weight can be tuned by an optimizer. These component translation features, which are increasingly complex models on their own, are not optimized directly toward our true objective function (e.g. BLEU), yet once we have the opportunity to optimize directly to our objective, we fit a global linear model, a rather blunt one-size-fits-all approach. But what if a feature has the potential to improve quality in some contexts, but not others? This is likely a common scenario since during feature engineering, developers may look through system output and devise models to address specific observed shortcomings; however, it is impossible for feature developers to consider all of the situations in which their new models might have negative side effects. We address these shortcomings by allowing translation features to be reweighted depending on context. This context could range from document topic to the average number of letters per word or combinations thereof. As discussed in Section 2.1, linear models are simply incapable of such fine distinctions unless they are explicitly encoded into the feature set, placing this burden squarely upon the human system designers.

2.3 Thesis Statement

We have introduced linear models as the state-of-the-art in statistical machine translation. We have also suggested that direct optimization of a linear model may produce suboptimal results in a SMT system by ignoring useful non-linear information present in the feature set. Currently, this shortcoming is addressed by human system engineers manually developing increasingly complex models.

In this thesis, we propose to develop methods that address these shortcomings of linear learning. We claim that:

- by dynamically expanding the feature set during learning we can render the learned model non-linear with regard to the initial feature set;
- discretization of initial features will result in a statistically significant improvement in overall translation quality as measured by automatic metric scores; and
- conjunction of initial features will result in a statistically significant improvement in overall translation quality as measured by automatic metric scores.

To support these claims, we propose to:

- develop a method of feature discretization to relax the properties of linearity and monotonicity (Chapter 3);
- develop a method of feature conjunction to relax the property of independence that dynamically performs feature induction during learning (Chapter 4);
- demonstrate that discretization can be used to recover a system with at least state-of-the-art performance by performing a non-linear transformation of the standard feature set *without* the typical log transform applied (Section 3.2);
- demonstrate that conjunctions of features can improve translation quality (Chapter 5); and
- contribute three real-world applications of these theoretical tools in machine translation tasks (Chapter 5).

From an engineering perspective, the resulting model will still be directly usable in conventional decoders with minimal to no modification. From a scientific perspective, the resulting model will be *interpretable*, giving us insight into which non-linear factors are most important in achieving high quality translations.

2.4 Experimental Setup

To empirically verify these claims, we will perform experiments in Chinese→English, Arabic→English, and one additional language pair, as well as the reverse direction from English→Foreign for one language pair. These language pairs exhibit variety in the quality achievable by state-of-the-art systems; under our data scenario, a baseline Chinese→English typically scores around 30 BLEU while a baseline Arabic→English system scores around 50 BLEU. In turn, this can cause features to have very different behaior inside the system. For example, a feature that looks at source phrase length on the foreign side will be fairly uninteresting

in the Chinese case since source phrase match lengths are usually around one word while the feature could be important given the longer source spans matched in the Arabic system. The additional language pair will be selected based on preliminary results from the Chinese and Arabic experiments so as to best demonstrate the strengths of our approach. For example, it may turn out that the proposed techniques work best with mixed genres, a low-resource scenario, or in morphologically rich languages.

Chinese Resources: For the Chinese→English experiments, including the completed work presented in this proposal, we train on the Foreign Broadcast Information Service (FBIS) corpus[2] of approximately 300,000 sentence pairs with about 9.4 million English words. We tune on the NIST MT 2006 dataset and test on NIST MT 2008. Full details are shown in Figure 2.5.

	Sentences	Words (avg per reference)		Translations
		Chinese	English	
FBIS	303K	7.9M	9.4M	1
MT06 (tune)	1664	38.8K	47.1K	4
MT08 (test)	1357	32.5K	40.6K	4

Figure 2.5: Corpus statistics for Chinese→English experiments.

Arabic Resources: For the Arabic→English experiments, we train on the much larger NIST MT 2009 constrained training corpus [3] of approximately 5 million sentence pairs with about 181 million English words. We tune on the NIST MT 2006 dataset and test on NIST MT 2005 and 2008.[4] [5] Full details are shown in Figure 2.5.

	Sentences	Words (avg per reference)		Translations
		Arabic	English	
NIST Train	5.4M	184.5M	181.4M	1
MT06 (tune)	1797	49.0K	59.2K	4
MT05 (test)	1056	32.6K	36.2K	4
MT08 (test)	1360	45.0K	51.6K	4
MT09 (test)	1313	39.7K	47.3K	4

Figure 2.6: Corpus statistics for Arabic→English experiments.

Formalism: In our experiments, we use a hierarchical phrase-based translation model (Chiang, 2007). A corpus of parallel sentences is first word-aligned and then phrase translations are extracted heuristically. In addition, hierarchical grammar rules are extracted where phrases are nested. Such aligned subphrases are

[2]Distributed as part of the NIST Open MT Evaluation as Linguistic Data Consortium catalog number LDC2003E14

[3]A list of the resources available as part of the NIST MT 2009 constrained training resources is availble at `http://www.itl.nist.gov/iad/mig/tests/mt/2009/MT09_ConstrainedResources.pdf`

[4]The NIST MT test sets are available from the LDC as catalog numbers LDC2010T{10,11,12,13,17,21,23}

[5]One of the four references for the Arabic MT08 weblog data was not processed correctly in the officially released XML document and is mismatched with regard to the source sentences. There is no obvious way of reversing this error. However, since three references are still valid, this should have negligible impact on the results.

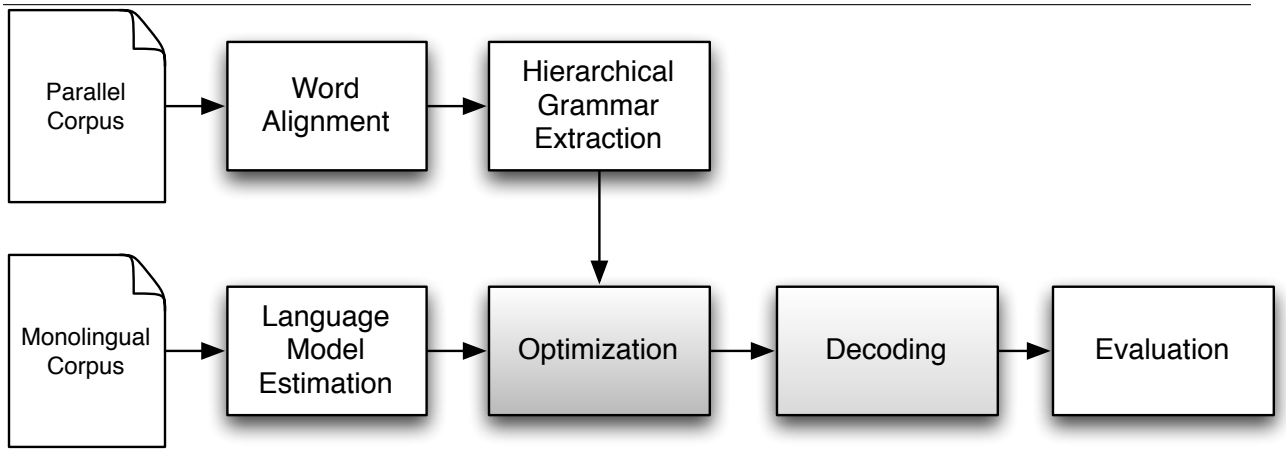

Figure 2.7: Our experimental pipeline. Shaded components fall within the scope of the proposed work. The optimization component will be the primary focus. Minor modifications to the decoder will be necessary to support non-local conjunctive features. Note that the optimizer is run multiple times to control for optimizer instability (Clark et al., 2010).

used to generalize their parent phrases by being substituted as a single non-terminal symbol X. In general, our choice of formalism is rather unimportant – our techniques should apply to most common phrase-based and chart-based paradigms including hiero and syntactic systems. Our decision to use hiero was primarily motivated by the cdec decoder's API being most amenable to our proposed work.

Tools: After initial pre-processing such as tokenization and segmentation, we perform word alignment using MGIZA++ (Gao and Vogel, 2008) and then extract and score a hierarchical phrase-based grammar using a suffix array data-structures that samples a few hundred target phrases for each matched source phrase (Lopez, 2008; Lopez, 2007; Lopez, 2008). For decoding, we will use cdec (Dyer et al., 2010), a multi-pass decoder that supports syntactic translation models and sparse features. Our primary optimizer will be PRO (Hopkins and May, 2011) as implemented by the cdec decoder.

Baseline Features: As this thesis is largely about inducing enhanced feature sets, feature sets will vary. Here, we describe our baseline feature set, which is included in all experiments unless otherwised stated explicitly. We use the baseline features produced by Lopez' suffix array grammar extractor, which is distributed with cdec:

- $\log P_{\text{coherent}}(e|f)$: The coherent phrase-to-phrase translation probability (Lopez, 2008, p. 103). The phrasal probability of each English SCFG antecedent (e.g. "el [X] gato") given a particular foreign SCFG antecedent "the [X] cat" combined with *coherence*, the ratio of successful source extractions to the number of attempted extractions
- $\log P_{\text{lex}}(e|f)$, $\log P_{\text{lex}}(f|e)$: The lexical alignment probabilities within each translation rule, as computed by a maximum likelihood estimation over the Viterbi alignments
- $\log P_{\text{LM}}(e)$: The log probability of the target translation hypothesis under a language model

- $c(e)$ The count of target words (terminals) in the target translation hypothesis
- $c(\texttt{glue})$ The count of glue rules used in the derivation
- $c(\texttt{OOV}_{\text{TM}})$ The count of source tokens that were not recognized by the translation model (out of vocabulary) and were therefore passed through
- $c(\texttt{OOV}_{\text{LM}})$ The count of target tokens that were not recognized by the language model (out of vocabulary)

Hardware: We will continue to carry out experimentation on resources provided under the Extreme Science and Engineering Discovery Environment (XSEDE), a program of the National Science Foundation. This includes the use of the Trestles cluster at the San Diego Supercomputing Center, which offers many nodes having 32 cores and 64GB RAM, and the Blacklight cluster at the Pittsburgh Supercomputing Center, which offers nodes having 16 cores and 128GB RAM.

Evaluation: We will quantify increases in translation quality using automatic metrics including BLEU (Papineni et al., 2002) and METEOR (Denkowski and Lavie, 2010; Banerjee and Lavie, 2005). We will control for test set variation by measure the statistical significance over bootstrap replicas of the test set (Koehn, 2004) and control for optimizer instability by measuring variance and statistical significance according to approximate randomization (Clark et al., 2010). We will further mitigate the risk of confounds by performing a targeted automated analysis, breaking down improvements by category.

Workflow management: To manage our experiments in an orderly fashion and ensure that our work can be reproduced by the broader community, we use the Ducttape Workflow Manager, a successor of the LoonyBin HyperWorkflow Manager (Clark et al., 2009; Clark and Lavie, 2010).

2.5 Related Work

Previous work on feature discretization in machine learning has focused on the conversion of real-valued features into discrete values for learners that are either incapable of handling real-valued inputs or perform suboptimally given real-valued inputs (Dougherty, Kohavi, and Sahami, 1995; Kotsiantis and Kanellopoulos, 2006). Such learners include decision trees, bayesian networks, and rule learners.

Kernel methods such as support vector machines (SVMs) are often considered when non-linear interactions between features are desired since they allow for easy usage of non-linear kernels. Wu, Su, and Carpuat (2004) showed improvements using a non-linear kernel PCA method for word sense disambiguiation versus a support vector machine model. Tsochantaridis et al. (2004) introduce a support vector machine for structured output spaces and show applications in grammar learning, named-entity recognition, text classification, and sequence alignment. This line of work later included application to information retrieval, protein alignment, and classification with unbalanced classes (Joachims et al., 2009). Giménez and Màrquez (2007) used a support vector machine to annotate a phrase table with binary features indicating whether or not a phrase translation was appropriate given the context. Wang, Shawe-Taylor, and Szedmak (2007) present a string-kernel SVM model of machine translation, but apply it only on very small data due to inherent scaling difficulties. Even within kernel methods, learning non-linear mappings with kernels remains an open

area of research; For example, Cortes, Mohri, and Rostamizadeh (2009) investigated learning non-linear combinations of kernels.

However, kernel methods remain slow due to complex training and inference requirements. This poses a challenge in machine translation, which is a complex structured prediction problem involving non-local features. Kernel methods also make it difficult to map back to the primal form of the problem to recover features and their weights – that is they are generally not interpretable. Further complicating matters is the issue of future cost estimation, which requires a model to estimate the future values of non-local features to avoid excessive search errors during approximate inference. Nor would such an approach be compatible with most decoders available, slowing adoption of the work. Aditionally, much work has already been done to make inference fast with linear models in current decoders.

Decision trees and random forests are also commonly used non-linear learners. Decision trees have been successfully used in many areas of natural language processing including language modeling (Jelinek et al., 1994; Xu and Jelinek, 2004) and parsing (Charniak, 2010; Magerman, 1995). While they are considerably easier to interpret than kernel methods, they have two important deficiencies in the context of this work. As with kernel methods, it is again very difficult to perform future cost estimation for such models, a requirement of our modern decoding algorithms.

Feature augmentation using conjunctions of features has been explored in various areas of machine learning. Della Pietra, Della Pietra, and Lafferty (1995) describe a greedy algorithm for iteratively adding features to a random field based on expected gain (Della Pietra, Della Pietra, and Lafferty, 1997). Mccallum (2003) extends this work to conditional random fields, but also constructs the conjunctions iteratively. Guyon (2003) describes a checklist for applying machine learning to a problem, which recommends using conjunctive features if one suspects inter-dependence among the features. Guyon also notes that while a model may be linear in its parameters, the model may be rendered non-linear in its input variables via feature construction – the approach we take in this work.

Research has also been done on "unpacking" real-valued generative features into a few sufficient statistics. These unpacked statistics can then be optimized directly alongside other features in the translation system, providing the opportunity for a tighter fit. For example, Chen et al. (2011) explored unpacking the sufficient statistics of various phrase table smoothing methods and demonstrated an improvement in translation quality.

The most closely related work to this proposal is that of Nguyen et al. (2007). Nguyen attempted to use discretized real-valued features using 20-50 bins. However, with this small number of bins they saw no improvement. Nguyen also did not yet have access to reliable methods for optimizing large numbers of features available at that time, so they instead used a modified version of minimum risk annealing (Smith and Eisner, 2005). Nguyen did observe a small improvement when attempting to relax the feature independence constraint using Gaussian Mixture Model (GMM) features, which were separately trained toward a likelihood-based objective and then added as features into a standard MT model.

Chapter 3

Discretization: Inducing Non-Linear Transformations of Real-valued Features

We introduced the limitations of a static feature set within a linear model in Section 2.1. We then proposed feature induction as a means for overcoming these limitations in Section 2.3. In this chapter, we describe discretization (Section 3.1), which allows us to relax the linearity and monotonicity constraints imposed on the initial set of features. First, we provide some results using a very basic discretization technique (Section 3.2). We then describe neighbor regularization (Section 3.3), which will allow the optimizer to reliably estimate weights for the significantly larger number of features generated by discretization.

3.1 Discretization Approach

We now turn to the problem of learning non-linear transformations of real-valued features so those features need not be linear nor monotonic with regard to the objective function. At first glance, one might be tempted to simply apply a few non-linear functions on top of each function (e.g. $\log(x), \exp(x), \sin(x), x^n$). However, even if we were to restrict ourselves to some "standard" set of non-linear functions, many of these functions have hyperparameters that are not directly tunable by conventional optimizers (e.g. what period and amplitude for $\sin$, what value of n in x^n). Further, continuous transformations may simply be inadequate to model sudden changes in the behavior of the response.

Our approach is to first discretize real-valued features into a set of indicator features and then use a conventional optimizer to learn a weight for each indicator feature. This technique is sometimes referred to as binning and is closely related to quantization. In particular, we will transform the Ψ-local features $\mathbf{h}$, rather than the global features $\mathbf{H}$ such that no additional non-local information is required during inference; this is important to maintaining efficient inference (see Section 1.3).

Effectively, discretization allows us to re-shape a feature function. In fact, given an infinite number of bins, we can perform any non-linear transformation of the original function.[1]

However, choosing the right number of bins can have important effects on the model:

- **Lossiness.** If we choose too few bins, we risk degrading the model's performance by discarding important distinctions encoded in fine differences between the feature values. In the extreme case, we

[1] This is similar to the idea of Riemann Sums in calculus, in which some number of partitions is used to approximate the area under a curve; taking the limit as the number of partitions goes to infinity yields the true area.

could reduce a real-valued feature to a single indicator feature.

- **Sparsity.** If we choose too many bins, we risk making each indicator feature too sparse, which is likely to result in the optimizer overfitting such that we generalize poorly to unseen data

First, we will experimentally explore these effects in Section 3.2. Then, we will explore how to mitigate this sparsity in Section 3.3.

3.2 Completed Work: Static Discretization

In this section, we experimentally evaluate the performance of discretization using some fixed number of bins. We construct a Chinese→English system using the FBIS training data, NIST MT06 for tuning, and NIST MT08 for evaluation, as described in Section 2.4. We tune using PRO for 30 iterations as suggested by Hopkins and May (2011), though analysis indicates that the parameters converged much earlier. The PRO optimizer internally uses a L-BFGS optimizer with the default ℓ_2 regularization implemented in cdec. We begin with the baseline set of 7 common features described in Section 2.4, and modify them as described below.

First, we remove the phrase translation probability features and the lexical translation probability features and instead replace them as follows. We run two baseline experiments (the top two lines of Figure 3.1): One with the $P_{\text{phr}}(t|s)$ feature in log space, as usual, and another with this feature untransformed, in probability space. As expected, the feature in log space performs much better. We then run 4 scenarios using static discretization by representing the probability-space feature using only 3, 4, 5, or 6 base ten digits. For example, in the case of discretizing to 3 digits, the individual features in the model might include `TGS@0.111`, `TGS@0.112`, and `TGS@1.000`.

Condition	Tune	Test
Prob	21.6	15.7
Log	26.2	18.4
Bin3	26.9	**18.8**
Bin4	25.7	17.8
Bin5	25.2	17.0
Bin6	25.0	17.4

Figure 3.1: Results of static discretization experiments as measured by the BLEU metric.

The results of these experiments are shown in Figure 3.1. We see a clear trend of underperformance when using a larger number of bins – binning tends to do worse than the log tranform when using more than 3 decimal places to represent our feature. This trend is mirrored when we examine the feature weights assigned to each of the binned features in Figure 3.2 – the weights are dominated by overfitting noise when we use too many bins. Surprisingly, we observe lower translation quality on both the tuning and test data. We hypothesize that this is due to the changing samples between iterations of the PRO optimizer; that is, the discretized features become so sparse that their weights fail to generalize even between different samples from k-best lists. However, we can see that we overcome the effects of both lossiness and sparsity when using 3 decimal places – in fact, we do better than the features in log space, providing evidence that the log tranform is not somehow theoretically optimal, but merely an empirical convenience that often performs well.

While there are certainly more principled ways of choosing bin boundaries, which we plan to explore through the course of this work, these simple experiments demonstrate both the potential gains and the challenges we expect to face in discretization. For example, bin boundaries could be chosen by uniformly dividing the values of the feature or by examining the density of each feature value in the tuning data. It may also be beneficial to specialize bins near boundaries to be very narrow – this might allow zero and one counts to be weighted more freely.

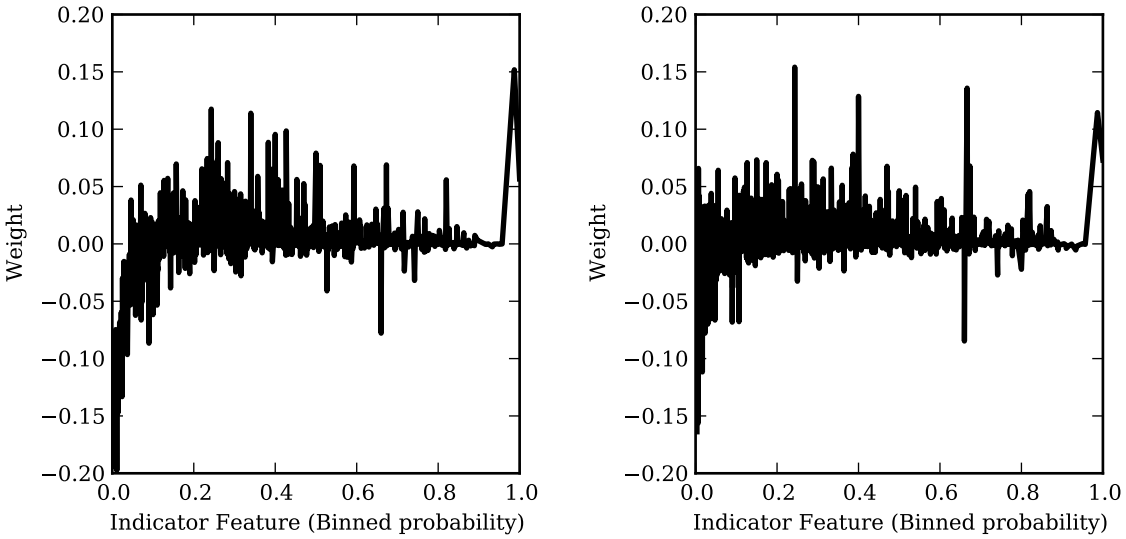

Figure 3.2: Plots of the weights of the indicator features representing the binned $P_{\text{rule}}(t|s)$ feature. Notice that the bin3 experiment (left) is beginning to show a log-like shape while the bin6 experiment (right) is dominated by noise due to overfitting. 794 features are represented along the x-axis for the bin3 experiment and 4532 features are represented for the bin6 experiment.

3.3 Proposed Work: Neighbor Regularization

Regularization has long been used to discourage optimization solutions that give too much weight to any one feature. This encodes our prior knowledge that such solutions are unlikely to generalize. Regularization terms such as the ℓ_p norm are frequently used in gradient-based optimizers including our baseline implementation of PRO.

As we just saw, static discretization is very brittle with regard to the number of bins chosen. Primarily, it suffers from sparsity. At the same time, we note that we know much more about discretized features than initial features since we control how they are formed. With these things in mind, we propose neighbor regularization, which embeds a small amount of knowledge into the objective function: that the indicator features resulting from the discretization of a single real-valued feature are spatially related. We expect similar weights to be given to the indicator features that represent neighboring values of the original real-valued feature such that the resulting transformation appears somewhat smooth.[2]

To incorporate this knowledge into our objective function, we use the method of Sandler et al. (2008) for regularization over feature networks (Sandler, 2010). In terms of learning, we are making the usual trade-off of regularization: higher bias (we may not fit the tuning data as tightly) for lower variance (different tuning sets should yield more similar weights than without regularization) and better generalization to unseen data.

As an introduction to describing neighbor regularization, we first note that the ℓ_2 regularizer can be interpreted probabalistically as adding a prior to the initial likelihood objective. This prior has the form of a multivariate Gaussian with a zero mean (to encourage the feature weight to remain small) and a variance inversely proportional to the regularization constant α. The regularization term added to the objective is then:

$$\mathcal{R}_2(\mathbf{w}) = \alpha \sum_{j=1}^{|\mathbf{h}|} w_j^2 \tag{3.1}$$

Since the Gaussian is multivariate, its variance parameter is in fact a covariance matrix.[3] For the ℓ_2 regularizer, this matrix is diagonal and can be calculated as the regularization constant α multiplied by the identity matrix I, encoding the knowledge that we do not believe any features to be related *a priori*. This allows us to rewrite the regularization term as a sequence of matrix operations:

$$\mathcal{R}_2(\mathbf{w}) = \mathbf{w}^\top M_2 \mathbf{w} \tag{3.2}$$

$$M_2 = \alpha I \tag{3.3}$$

[2] This is similar to the method of time series regularization, introduced by Yogatama et al. (2011).

[3] We specify dependencies largely in terms of the precision matrix P as it allows us to specify global dependencies while keeping the overall matrix sparse. For example, if feature 1 is related to feature 2, 2 to 3, and 3 to 4 (as if they were balls connected by springs), then we could specify this using only 6 entries in the precision matrix (specifying which balls are connected by springs), while we would need the full cross product of 16 entries in the covariance matrix (specifying every ball that moves when each other ball moves).

For example, for a set of 4 features, the matrix M_2 is:

$$\begin{bmatrix} \alpha & 0 & 0 & 0 \\ 0 & \alpha & 0 & 0 \\ 0 & 0 & \alpha & 0 \\ 0 & 0 & 0 & \alpha \end{bmatrix} \tag{3.4}$$

Now we formally define neighbor regularization as an instance of feature network regularization (Sandler et al., 2008; Sandler, 2010). We consider our features as vertices $V = \{1, \ldots, |\mathbf{h}|\}$ in a graph G with edges linking feature weights that we believe to be similar. These edges are non-negative with increasing weights indicating greater similarity. We encode these edge weights in the square matrix P where each $P_{ij} \geq 0$ contains the weight of the directed edge from vertex i to vertex j. Following Sandler, we constrain the sum of outgoing edges to sum to one, $\sum_i P_{ij} = 1$, so that no feature dominates the graph. To enforce the semantics that linked features should have similar weights while maintaining a convex objective function, our neighbor regularizer $\mathcal{R}_{\text{NR}}$ penalizes each feature's weight by the squared amount it differs from its neighbors:

$$\mathcal{R}_{\text{NR}}(\mathbf{w}) = \beta \sum_{j=1}^{|\mathbf{h}|} (w_j - \sum_{k=1}^{|\mathbf{h}|} P_{jk} w_k)^2 \tag{3.5}$$

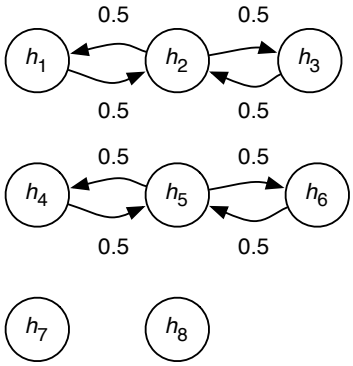

Figure 3.3: An example feature network implementing neighbor regularization. Discretized features resulting from the same initial real-valued feature are linked by edges as shown by h_1, h_2, h_3 and h_4, h_5, h_6. Initial features that are not discretized do not receive edges as shown by h_7, h_8.

Alternatively, this can be rewritten as a series of matrix multiplications:

$$\mathcal{R}_{\text{NR}}(\mathbf{w}) = \mathbf{w}^\top M_{\text{NR}} \mathbf{w} \tag{3.6}$$

$$M_{\text{NR}} = \beta (I - P)^\top (I - P) \tag{3.7}$$

In the case of neighbor regularization, only a few features, the neighboring features, will have edges between them. This will cause the matrix P to be tridiagonal. Continuing the example from Figure 3.3, the matrix would be:

$$\begin{array}{c} \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \end{array} \begin{array}{c} \begin{array}{cccccccc} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 \end{array} \\ \begin{pmatrix} 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \tag{3.8}$$

For reference, we present the complete modified loss function of the convex optimization problem internal to each iteration of PRO. For n sampled pairwise rankings, the loss function will take the following form when combined with the usual ℓ_2 regularizer and neighbor regularization:

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{n} l(\mathbf{x}_i, y_i; \mathbf{w}) + \mathcal{R}_2(\mathbf{w}) + \mathcal{R}_{\text{NR}}(\mathbf{w}) \tag{3.9}$$

The gradient update to minimize the loss $\mathcal{L}$ is then:

$$\nabla_{\mathbf{w}} \mathcal{L} = \sum_{i=1}^{n} \nabla_{\mathbf{w}} l(\mathbf{x}_i, y_i; \mathbf{w}) + 2(M_2 + M_{\text{NR}})\mathbf{w} \tag{3.10}$$

To evaluate the performance of neighbor regularization, we will carry out experiments on the various language pairs described in Section 2.4 using various schemes of static discretization (Section 3.2). Hyperparameters will be tuned on a separate held-out data set. We will evaluate each experimental condition of neighbor regularization using automatic metrics and perform significance testing to determine if improvements are meaningful. We will also investigate its effectiveness qualitatively, as we did with static discretization, using feature weight visualizations of the type shown in Figure 3.2.

Chapter 4

Conjunction: Inducing Dependencies Among Multiple Features

We have described how feature discretization can be used to relax the linearity and monotonicity requirements on (initially) real-valued features. We now turn our attention toward allowing our learning procedure to take advantage of inter-dependence among features.[1] While this is strictly not possible given a linear model with a fixed number of features and parameters, we propose to again augment the initial feature set such that our learned model can take advantage of interactions among features (those that can reasonably be expected to generalize to unseen data).

Later, we will describe applications of both discretization and conjunctions including results demonstrating that conjunctions among features can improve translation quality, a key claim of this thesis. In preliminary work, these conjunctions were manually selected. However, another goal of this work is to take as much burden off of feature engineers as possible. Toward that end, we now consider how to dynamically learn conjunctions during learning.

A naïve implementation would directly insert the full cross-product of all features into learning, making the size of the feature set $O(2^{|\mathbf{h}|})$. This would place large computational burdens on both learning and inference. Further, it would exponentially increase the number of free parameters in the model; the curse of dimensionality predicts that such a model would also perform poorly from a learning perspective due to overfitting.

With these issues in mind, we propose methods for:

- Enumerating a set of candidate feature conjunctions that can be efficiently computed by the decoder
- Selecting a set of candidate feature conjunctions that is both feasibly small for learning and that has enough training examples to be estimated reliably
- Incrementally adding these conjoined features to the model during optimization
- Reliably learning a set of weights for these complex conjoined features

[1] Inter-dependence among features can also be viewed as a non-linear combination of features rather than the usual linear combination.

4.1 Proposed Work: Feature Selection by Inference Complexity

In the context of inference complexity, if we consider the space of all possible $2^{|\mathbf{h}|}$ conjunctions among initial features, it becomes apparent that many of these conjoined features cross many more rule boundaries than before (see Section 1.3 and Figure 1.1). That is, the factorization $\mathcal{F}$ of a derivation into compatibility functions Ψ that was valid under the initial feature set, is no longer valid under the new conjoined feature set. This $2^{|\mathbf{h}|}$ conjoined feature set would require a new factorization in which each compatibility function has a much larger scope. In fact, the scope of some conjoined features would span the entire sentence. By separating partial hypotheses that were originally in the same equivalence classes, this would allow for less substructure during inference, causing decoding to become far more costly.

Clearly, this is not desireable if we seek efficient inference. Therefore, we perform feature selection based on which conjoined features will not enlarge the scope of any compatibility functions. First, we consider three sets of initial features:

- **Observable initial features**. Examine only the source sentence, placing them within the compatibility function $\Psi^{\text{Obs}}(\mathbf{f})$. Since these features are constant with regard to the source sentence, they will never affect the ranking of target hypotheses, unless conjoined with a feature from another category.

- **Local initial features**. Examine random variables local to a single rule or phrase, placing them within the compatibility function $\Psi^{\text{Local}}(\mathbf{f}, r, e_1 \dots e_k)$.

- **Non-local initial features**. Examine random variables that span multiple rules or phrases. Such features include the language model, whose compatibility function is $\Psi^{\text{LM}}(e_1 \dots e_k)$

We then consider the following conjunctions among these feature sets:

- Local initial features with themselves

- Local initial features with non-local initial features

- Observable initial features with all of the above conjunctions

- Observable initial features with local initial features

- Observable initial features with non-local initial features

By selecting features only from this set, we guarantee that no additional non-local state information must be kept at inference time. In the remainder of this section, we will explore the technical details of how to enumerate this seemingly simple set of conjunctions.

We specify the space of candidate conjunctions that we will consider declaratively in the framework of semirings. Semirings have been used in computational linguistics for quite some time (Goodman, 1999; Eisner, 2001; Mohri, 2002) and more recently in statistical machine translation (Lopez, 2009; Kumar et al., 2009; Li and Eisner, 2009; Dyer et al., 2010; Dyer, 2010).[2] A semiring K is defined by a 5-tuple $\langle \mathbb{K}, \otimes, \oplus, \overline{0}, \overline{1} \rangle$:

- $\mathbb{K}$: The set of values.
- $\oplus$: A commutative addition-like operator (a monoid) used for aggregating the results of $\otimes$ with other previous results within a chart cell.
- $\otimes$: A multiplication-like operator (a monoid), used for aggregating neighboring antecedents in a production rule as well as the weight of the production rule.
- $\overline{0}$: An annhilator such that $\overline{0} \otimes x = \overline{0}$ (used for initializing blank chart entries in a hypergraph decoder or empty stacks in a phrase-based decoder).
- $\overline{1}$: An identity element $\overline{1} \otimes x = x$ (used for seeding agenda with axioms such as input words).

First, we will address how to enumerate conjunctions of local features with themselves. We will approach the enumeration of local feature conjunctions in two phases: First, we will enumerate the set of rules or phrases that could participate in the translation of a sentence and then we will extract all local conjunctions over features *within* each rule.

After collecting rules and/or phrases from the entire tuning corpus,[3] we enumerate all local conjunctions over these local non-observable features. We will refer to the initial feature set (the set of all features reachable in our hypothesis space) as $\mathbf{h}_{\text{Init}}$ and we will use a function $\texttt{COMB}(X, n)$, which enumerates all combinations of length n over the set X. We define the local conjunction semiring K_{conj} as:

- $\mathbb{K}_{\text{conj}}$: Let C be a conjoined feature, represented as a set containing component indicator features. The semantics of the set C are that all features in C participate in a single conjunctive feature by applying the feature operator $\wedge$ so that the feature C is true iff all component features in C are true. Initial non-conjoined features are represented as a set C of size one. Then $\mathbb{K}$ is a set containing instances of conjoined features C.
- $\oplus_{\text{conj}}$: For operands A and B, the set-wise union over two sets of conjoined features: $A \cup B$
- $\otimes_{\text{conj}}$: For operands A and B, the set of all conjunctions over (potentially) conjoined features: $\{a \cup b | \forall a \in A, \forall b \in B\}$
- $\overline{0}_{\text{conj}}$: The empty set: $\emptyset$
- $\overline{1}_{\text{conj}}$: A set containing one conjoined feature, composed of zero initial features: $\{\emptyset\}$

For example, consider two initial features h_{TGS} and h_{Count}. We can represent each of these as a conjoined feature of order 1: $C_{\text{TGS}} = \{h_{\text{TGS}}\}$ and $C_{\text{Count}} = \{h_{\text{Count}}\}$. Then, we the following would be valid

[2] We also rely on the language of hypergraphs for machine translation. Hypergraphs were introduced to the parsing community by Klein and Manning (2001) and have since gained popularity in chart-based statistical machine translation (Zollmann and Venugopal, 2006; Huang and Chiang, 2007).

[3] The task of collecting all grammar rules that can participate in valid derivations of a tuning set is also a trivial semiring, the rule forest semiring K_{rules}

operations under our semiring:

$$K_1 = \{C_{\text{TGS}}\} = \{\{h_{\text{TGS}}\}\}, K_1 \in \mathbb{K}$$
$$K_2 = \{C_{\text{Count}}\} = \{\{h_{\text{Count}}\}\}, K_2 \in \mathbb{K}$$
$$K_{\text{union}} = K_1 \oplus K_2 = \{\{h_{\text{TGS}}\}, \{h_{\text{Count}}\}\}$$
$$K_{\text{cross}} = K_1 \otimes K_2 = \{\{h_{\text{TGS}}, h_{\text{Count}}\}\}$$

Notice that K_{union} is a set containing two conjoined features and K_{cross} is a set containing one conjoined feature of order two.

Using this semiring K_{conj}, we can now enumerate all conjunctions of local non-observable features $\mathbf{h}_{\text{LocalNonObs}}$ over the rules $\mathbf{r}$. If $\mathbf{h}_r$ is the set of local initial features associated with a rule r, then:

$$\mathbf{C}_{\text{LocalNonObs}} = \bigoplus_{r \in \mathbf{r}} \bigoplus_{i=1}^{|\mathbf{h}_r|} \bigoplus_{C \in \texttt{COMB}(\mathbf{h}_r, i)} \bigotimes_{h \in C} \{h\} \tag{4.1}$$

Similarly, we can obtain the set of local observable features:

$$\mathbf{C}_{\text{Obs}} = \bigoplus_{i=1}^{|\mathbf{h}_{\text{Obs}}|} \bigoplus_{C \in \texttt{COMB}(\mathbf{h}_{\text{Obs}}, i)} \bigotimes_{h \in C} \{h\} \tag{4.2}$$

Recall that conjunctions among observable features $\mathbf{h}_{\text{Obs}}$ will still result in features that cannot improve the model (since such a source observation will be constant with regard to translation hypotheses, not affecting their ranking). However, they can provide information when conjoined with latent and hypothesis features. Taking these into account (but still neglecting non-local features), we can express the space of conjunctions among local features as:

$$\mathbf{C}_{\text{Local}} = \mathbf{C}_{\text{Obs}} \otimes \mathbf{C}_{\text{LocalNonObs}} \tag{4.3}$$

At this point, we have enumerated all conjunctions among local features, both observable and non-observable. Now we turn our attention toward forming conjunctions among local and non-local features. When considering which local features should be conjoined with non-local features, we must define the set of local features that fall within the scope of each non-local feature instance – that is, which local features should be conjoined with each non-local feature.

First, it is potentially intractable to enumerate all of the non-local features that could occur in a sentence as this is tantamount to enumerating all of the hypotheses that a sentence can generate.[4] Therefore, we sample the non-local features that we expect to see during the current iteration of optimization by examining the k-best derivations of each sentence, recovering the non-local features that fired for each hypothesis.

[4]Imagine an algorithm that attempts to efficiently enumerate all non-local features. It might attempt to run a semiring over a hypergraph that stores (1) all of the features seen under the current vertex along with the conjunction context in which that feature was seen and (2) a *set* containing all of the left and right non-local feature states that could possibly be produced under the current vertex. In the worst case, no recombination is possible and at the sentence-spanning vertex, the set must contain an entry for every hypothesis that can be produced in the hypergraph. Storing such a large number of hypotheses would be unacceptably inefficient.

Second, by definition, the scope of a non-local feature is not dictated by how the model is factored. In fact, the upper bound on a non-local feature's scope is an entire sentence. We might take this as evidence that we should conjoin all local features that might occur within a sentence with our sample of the non-local features for that sentence. However, this greatly overstates which local features are within the scope of each non-local features. Generally, non-local feature functions are conceptualized as managing their own state – they receive antecedent state objects and return a consequent state object along with any relevant feature scores. Therefore, we defer to each non-local feature function to provide us with this information – that is, we require each non-local indicator feature function to provide an `EnumerateScope` mechanism for returning both *if* it fires and which local features fall within its scope. Each hypothesis in a k-best list generated using the `EnumerateScope` function of each non-local feature will be annotated with the a list whose elements are tuples $(h_{\text{NonLocal}}, \mathbf{h}_{\text{LocalScope}})$ where h_{NonLocal} is a non-local feature associated with the hypothesis and $\mathbf{h}_{\text{LocalScope}}$ is the set of local initial features that should be considered for conjunction with it. Consider the example in Figure 4.1 this would allow a discriminative lexical n-gram language model to return the feature ID of a match $h_{\text{NonLocal}} =$ `LM-cat-eats` and the local features associated with the words in that n-gram. In this case, we might return an unaligned words indicator feature, such that the returned tuple would be: (`LM-cat-eats`, [`Aligned`, `Unaligned`]). In general, if the language model match spans a phrase boundary, $\mathbf{h}_{\text{LocalScope}}$ may contain features from multiple phrases. We refer to this overall procedure that enumerates a k-best list containing non-local features and the local features falling within their scope as `ScopedKbest`.

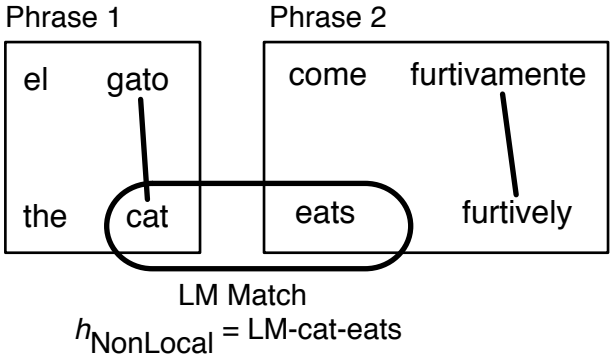

Figure 4.1: An example of how `EnumerateScope` will behave. A discriminative LM matches `LM-eats-futively`, `EnumerateScope` might indicate that the features `Aligned` and `Unaligned` are within the scope of the feature `LM-cat-eats`.

With this mechanism in place, we finally obtain $\mathbf{C}_{\text{NonLocal}}$ as the set of all non-local features conjoined with in-scope local features for a single sentence:

$$\mathbf{C}_{\text{NonLocal}} = \bigoplus_{(h_{\text{NonLocal}}, \mathbf{h}_{\text{LocalScope}}) \in \texttt{ScopedKbest}} \left(\{h_{\text{NonLocal}}\} \otimes \bigoplus_{i=1}^{|\mathbf{h}_{\text{LocalScope}}|} \bigoplus_{C \in \texttt{COMB}(\mathbf{h}_{\text{LocalScope}}, i)} \bigotimes_{h \in C} \{h\} \right) \quad (4.4)$$

Finally, we can incorporate all of these observable, local, and non-local features into the full space of conjunctions that we will be considering by building on Equation 4.4:

$$\mathbf{C}_{\text{Conj}} = \mathbf{C}_{\text{Obs}} \otimes (\mathbf{C}_{\text{LocalNonObs}} \oplus \mathbf{C}_{\text{NonLocal}}) \tag{4.5}$$

This technique of enumerating conjunctions provides us with a set of features that can be efficiently computed by the decoder without adding any additional non-local state information. However, it is likely that many of these conjoined features are very sparse and are not likely to be reliably estimated. With this in mind, we do not expect this set of conjunctions to perform well in isolation and we plan to evaluate them only as a baseline for the methods presented in the following sections.

4.2 Proposed Work: Feature Selection by Sparsity

Previously, we described how to enumerate a set candidate feature conjunctions that are amenable to efficient decoding and occur within the tuning data. Most of these conjoined features will occur infrequently in the tuning data and even less frequently within each of PRO's samples of ranked hypothesis pairs, giving us little hope of estimating their weights reliably. To address this, at each iteration of PRO, we examine the hypothesis pairs selected by PRO's sampler and collect two counts:

- $c_{\text{pairs}}(h)$ The count of sampled pairs in which the feature h occurs with a non-zero difference (pairs in which the feature has the same value contribute nothing to the gradient and provide no information to optimization)
- $c_{\text{sentences}}(h)$ The count of tuning sentences containing having a non-zero $c_{\text{pairs}}(h)$

We then set thresholds on each of these $\theta_{\text{pairs}}, \theta_{\text{sentences}}$, which we will adjust as hyperparameters. We then use this thresholding mechanism to perform feature selection, throwing away features that are too sparse to be reliably estimated.

To evaluate the performance of this technique, we will carry out experiments on the various language pairs described in Section 2.4. Hyperparameters will be tuned on a separate held-out data set. We will use initial features from the applications described in Chapter 5. Comparisons will be made to a baseline without conjunctions and a system using conjunctions selected for decoder efficiency but not for sparsity. We will evaluate each experimental condition using automatic metrics and perform significance testing to determine if improvements are meaningful.

4.3 Proposed Work: Incremental Feature Selection via Grafting

We propose to control the number of features in our model by grafting, as described by Perkins (2003) and Riezler and Vasserman (2003). Feature grafting incrementally selects features to be added to the model based on the amount that each feature will contribute the the objective function; features that do not contribue enough are not added to the model. As a side effect, this makes the learned model more interpretable by reducing the number of correlated features in the model. Like an ℓ_0 or ℓ_1 regularizer, grafting strongly reduces the overall number of features in a model. However, it does not give up the convex objective function

of the ℓ_2 regularizer. Previously, Okanohara (2009) have applied grafting in the context of combination features on a parsing task with positive results.

To evaluate the performance of grafting, we will carry out experiments on the various language pairs described in Section 2.4. Hyperparameters will be tuned on a separate held-out data set. We will use initial features from the applications described in Chapter 5. Comparisons will be made to a baseline without conjunctions and a system using conjunctions with features selected for inference efficiency and for sparsity. We will evaluate each experimental condition using automatic metrics and perform significance testing to determine if improvements are meaningful.

4.4 Proposed Work: Feature Complexity Regularization

Intuitively, conjoined features made up of a very large number of component features are less likely to generalize. Further, such complex features are likely to outnumber less complex features due to the combinatorial explosion of feature combinations; even small weights on such a large number of features could cause them to overpower less complex, potentially more reliable features. However, we do not wish to discard such features outright since such features could forseeably be valuablue. Our solution is to introduce a regularization hyperparameter γ and regularization term into our objective function using N as the maximum number of component features involved in any conjoined feature:

$$\mathcal{R}_{\text{complexity}} = \gamma \sum_{j=0}^{N} j^2 \sum_{\substack{i=0 \\ \text{iff } \texttt{ORDER}(h_i)=j}}^{|\mathbf{h}|} ||w_i||_2 \tag{4.6}$$

We use $\texttt{ORDER}(h)$ to indicate the number of component features involved in a conjoined feature. Since j is constant with regard to $\mathbf{w}$, this regularization term differs from the ℓ_2 norm by only a constant. Alternatively, we can view complexity regularization as partitioning the feature set into N groups and then applying a ℓ_2 regularizer to each group where the regularization weight of each group is γj^2. This preserves the convexity of the objective function and makes it easy to incorporate into the gradient-based updates of PRO. With this in mind, the gradient is:

$$\nabla_{\mathbf{w}} \mathcal{R}_{\text{complexity}} = \gamma \sum_{j=0}^{N} j^2 \sum_{\substack{i=0 \\ \text{iff } \texttt{ORDER}(h_i)=j}}^{|\mathbf{h}|} 2w_i \tag{4.7}$$

To evaluate the performance of feature complexity regularization, we will carry out experiments on the various language pairs described in Section 2.4. Hyperparameters will be tuned on a separate held-out data set. We will use initial features from the applications described in Chapter 5. Comparisons will be made to a baseline without conjunctions and a system using conjunctions selected for inference efficiency and for sparsity. We will evaluate each experimental condition using automatic metrics and perform significance testing to determine if improvements are meaningful.

Chapter 5

Applications

In previous chapters, we described a theoretical framework for non-linear learning in machine translation. We now present concrete tasks that make use of these tools.

5.1 Completed Work: Direct Application to Simple, Targeted Features

In our first application, we consider what we expect to be a rather common feature engineering scenario: A feature engineer adds to a baseline translation system a few simple features that are dense (occur frequently in the tuning data) and that target common intuitive phenomena. We use the following *indicator* features as an example of such a scenario.

We include the following feature templates:

- **Nonterminal Count (Nonterm).** One feature for each count of non-terminals in the grammar rule (This can be considered a discretization of the commonly used (This can be considered a discretization of the commonly used `ArityPenalty`, whose value is the count of non-terminals in the grammar rule)

- **Source and Target Word Count (WC).** One feature for each count of terminals (words) on each side of the grammar rule (The target side can be considered a discretization of the commonly used `WordPenalty`, whose value is the count of terminals in the output string)

- **Source and Target High Frequency Word Count (HFWC).** One feature for each count of terminals (words) on the source side of the grammar rule that are high-frequency words

- **Source and Target High Frequency Words (HFW$_n$).** One feature (the lexical identity) of each source word that is a high-frequency word. We collect a set of high-frequency words from the training data by rankinking them by frequency and thresholding at n

Since these are indicator features, each count and each high-frequency lexical item is a separate feature such as `SrcWC-1`, `SrcWC-2`, and `TgtHFWLex-the`. Notice that although some of these features are "source" features in that they examine source words, they *can* have an impact on the model since they are actually latent features using knowledge of how the source sentence has been segmented into phrases and/or rules and therefore can improve the model even without conjunctions.

To experimentally evaluate the performance of this feature set under discretization and conjunction, we construct an Arabic→English system using the standard NIST training data, NIST MT06 for tuning, and NIST MT08 for evaluation, as described in Section 2.4. We tune using PRO for 30 iterations as suggested by Hopkins and May (2011), though analysis indicates that the parameters converged much earlier. The PRO optimizer internally uses a L-BFGS optimizer with the default ℓ_2 regularization implemented in cdec. All experimental conditions below include the 7 common features described in Section 2.4 with some subset of the additional targeted features described in this section.

Condition	Tune	MT05	MT08
Baseline	40.4	54.5	47.2
WC, Nonterm	41.5	**56.9**	48.6
WC, Nonterm, Conj4	42.2	55.5	48.6
WC, Nonterm, HFWC	42.0	55.9	49.5
WC, Nonterm, HFWC, Conj4	43.2	56.4	49.2
WC, Nonterm, HFWC, HFW_{20}	42.5	55.8	49.9
WC, Nonterm, HFWC, HFW_{20}, Conj4	43.4	55.8	49.9
WC, Nonterm, HFWC, HFW_{50}	42.8	**56.8**	49.6
WC, Nonterm, HFWC, HFW_{50}, Conj4	**43.8**	56.2	**50.4**

Figure 5.1: Results of initial conjunction experiments on the NIST MT Arabic→English data set as measured by the BLEU metric. Both the Meteor and TER evaluation metrics also showed similar improvements.

As shown by the result "WC, Nonterm" in Figure 5.1, by using simple discretizations of the `ArityPenalty` and `WordPenalty` features, we can achieve improvements of +2.4 BLEU on MT05 and +1.4 BLEU on MT08. However, when we add conjunctions of up to order 4 (Conj4), we see consistent improvements only on the tune set, but often hurting the test set on MT05 – this highlights the potential for overfitting with conjunctions and the need for careful feature selection as described in Section 4.1.

5.2 Completed Work: Single System, Multiple Domain Adaptation

Machine translation systems are often used for information assimilation as in the case of one person seeking to understand information or varying genres stored in many languages. This is the use case for most modern translation web services including Google Translate, Microsoft Translator, and Yahoo Babel Fish.

It is possible to optimize a separate set of weights for each genre. However, taken from the view of our conjunction framework, such a model implies that *all* of the features of the model must be, in effect, conjoined with each genre and none of the original features may share statistics among genres. This can result in the model's parameters being more sparsely estimated while removing the possiblity of relying on the better estimated non-conjoined features.

With these issues in mind, we propose to use conjunctions of observable genre features with initial model features. This opens up new possibilities as well, such as having multiple granularities of genres such as `news` and `news-political` or even using communication modalities in combination with topics as in `text-political` and `speech-political`. This would give us resulting features such as `news-LanguageModel`, `news-WordPenalty`, and `speech-WordPenalty`.

A similar conjunctive approach to domain adaptation (using "feature augmentation") has been successfully applied in other areas of computational linguistics by Daumé III (2007).

To experimentally evaluate the performance of single system, multiple domain adaptation, we construct an Arabic→English system using the standard NIST training data, NIST MT06 for tuning, and NIST MT08 for evaluation, as described in Section 2.4. We tune using PRO for 30 iterations as suggested by Hopkins and May (2011), though analysis indicates that the parameters converged much earlier. The PRO optimizer internally uses a L-BFGS optimizer with the default ℓ_2 regularization implemented in cdec. All experimental conditions below include the 7 common features described in Section 2.4 as standalone features with each of the 7 features conjoined with each of the 2 genres, for a total of 21 features overall.

Condition	Tune (MT06)	MT08			MT09		
		News	Web	All	News	Web	All
Baseline	40.4	47.2	30.4	40.1	50.9	30.9	41.2
Genre Conjunctions	**41.3**	**48.6**	**30.8**	**41.1**	**52.2**	**32.0**	**42.4**

Figure 5.2: Results of initial multi-genre experiments on the large NIST MT Arabic→English data set as measured by the BLEU metric. Both the Meteor and TER evaluation metrics also showed improvements.

We present the results of genre conjunctions on the large NIST Arabic→English system in Figure 5.2. The conjunctions provide gains of 1.0 BLEU on MT08 and 1.2 BLEU on MT09.

5.3 Proposed Work: A Jointly Optimized Discriminative Language Model

In this section, we explore how to accomplish discretization of an important non-local feature: the language model. We will then build interpretable conjunctions on top of this in Section 5.4. Work has already been done in improving over generative language models in favor of discriminative equivalents (Roark et al., 2000; Roark, Saraclar, and Collins, 2007) including in the context of machine translation (Li and Khudanpur, 2008).

However, generative language models typically have millions or billions of free parameters – one for each n-gram in the monolingual corpus that it is trained on. Yet previous attempts to learn discriminative language models in the context of machine translation have typically relied on a much smaller number of parameters. Li and Khudanpur (2008) used only unigram and bigram features in their discriminative model and later used only the top 250 most frequent words in their "rule bigram" features, giving them only about 500 features in their discriminative model (Li et al., 2010).

These methods use a pipeline approach, taking one of the following forms:

- First, generate training data for the discriminative LM learner using a baseline MT system. Then, learn a n-best reranking model. (Li and Khudanpur, 2008)
- First, generate training data for the discriminative LM learner using a baseline MT system. Then, learn the parameters of the discriminative LM. Finally, learn a new set of weights for the MT system with the discriminative LM incorporated. (Li et al., 2010)

These pipeline methods each have their own shortcomings. First, training a n-best reranking model over the entire training data can be quite expensive as it typically involves decoding the entirety of the training data. N-best reranking is also at the mercy of incorrectable errors made by the baseline MT system; if an otherwise good hypothesis is dropped from the n-best list, there is no hope of recovery. This can be a significant source of error since even large n-best lists represent a tiny fraction of the overall hypothesis space.

On the other hand, training a discriminative LM that is later integrated into the decoder has access to the full hypothesis space. However, the parameters of the discriminative LM are fixed during the optimization of the final MT system's parameters. This can be viewed as an additional constraint on the combined parameter space, which can lead to underfitting.

Our approach is to jointly optimize the features of both the discriminative LM and the MT system using only the development set. Because deveopment sets are typically smaller than the training set, optimization requires less CPU time. However, this also means that we are unlikely to have enough data to reliably estimate millions of parameters. Even including only bi-grams, there are potentially millions of features on the target side of the Arabic→English. Therefore, we propose to use a small set of carefully engineered features in our model.

We plan to rely on a generatively estimated language model as one of these features for two reasons:

- Current approximate inference techniques in machine translation rely heavily on rest cost (future cost) estimation to achieve good approximations of the model-best decoding. Currently, feature-based language models are generally not able to efficiently providing accurate future cost estimates.

- Generative language models can reliably estimate many more parameters due to the amount of knowledge that has been manually embedded into their estimation formulae; discriminative models generally do not have the benefits of such knowledge-rich constraints.

As a starting point, we propose two basic feature sets for our discriminative LM:

- The n most frequent bigrams found in a monolingual data source

- Discretized log probabilities (match features) and backoffs (miss features) as indicators, separated by length. For example:

  - For a trigram match: `LM-Match-Len3-VeryLikely`, `LM-Match-Len2-SomewhatLikely`, `LM-Match-Len1-VeryLikely`
  - For a trigram miss: `LM-Miss-Len3-VeryLikely`, `LM-Match-Len2-SomewhatLikely`, `LM-Match-Len1-VeryLikely`

These bigram features build on the earlier work in discriminative language models described above. Other work in language model quantization shows that only a few discrete values of the LM probabilities are able to produce most of the signal provided by the language model (Federico and Bertoldi, 2006; Whittaker and Raj, 2001a; Whittaker and Raj, 2001b); We are in fact performing a type of quantization with our discretization of generative LM features. However, our discriminative LM features now have the opportunity to (re-)estimate the importance of matches and misses of different orders and different magnitude.

To evaluate the performance of our jointly optimized discriminative language model, we will carry out experiments on the various language pairs described in Section 2.4. Hyperparameters will be tuned on a separate held-out data set. Comparisons will be made to a baseline without the discriminative language model. We will evaluate each experimental condition using automatic metrics and perform significance testing to determine if improvements are meaningful.

5.4 Proposed Work: Context-Rich Language Modeling

Previously, we described how to convert a typical generative language model into a set of features that more directly and interpretably exposes the preferences of the learned model. We now describe how to enrich such a model with context information from the translation model using feature conjunctions.

First, we consider incorporating phrase boundary information into the language model. Phrase-based machine translation systems memorize and reproduce translation fragments seen in the training data. The model determines which translation fragments are chosen. Each of these fragments was a fluent and correct in its original context, but when the decoder combines these fragments in a new context, we can no longer be sure that the hypothesis will be fluent as a whole. We refer to this phenomena of disfluency at phrase boundaries as boundary friction (Brown et al., 2003). Normally, the phrase penalty, which encourages the use of larger phrases, and the language model are the primary tools that combat boundary friction. Yet even though the language model is a word-based feature, it has no knowledge of whether or not it is scoring fluent memorized text (as it was trained to do) or scoring across potentially disfluent phrase boundaries. We propose to conjoin the discriminative language model features from Section 5.3 with phrase boundary indicator features so that language modeling predictions can weigh more heavily where we expect boundary friction and can reduce their influence when the translation model is reciting already-fluent memorized phrases.

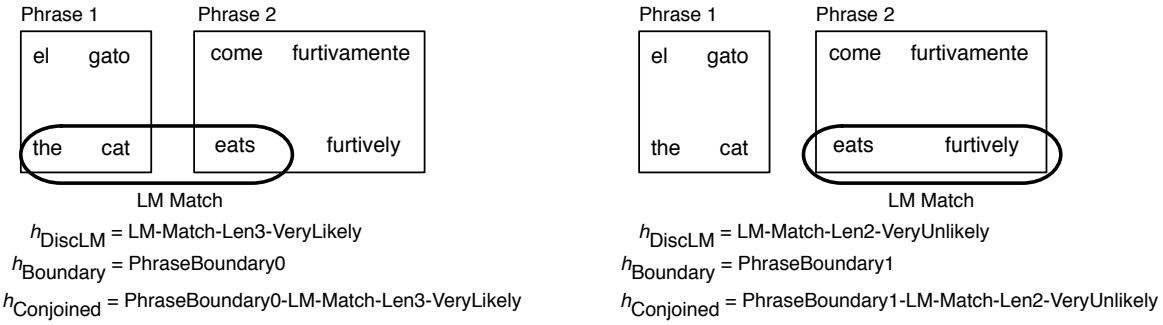

Figure 5.3: Examples of how phrase boundary features might be conjoined with the features of our discriminative language model from the previous section.

For example, consider Figure 5.3. Building on our feature set from Section 5.3, we may have features such as `PhraseBoundary0-LM-Match-Len3-VeryLikely` to indicate that we are just to the right of a phrase boundary and the language model found a trigram match bridging this phrase boundary; or we might have a feature such as `PhraseBoundary1-LM-Match-Len2-VeryUnlikely`, which indicates that we are deeper within a phrase but found a smaller, unlikely match – without the phrase boundary information, this would look like a bad choice, but with the large preceeding phrase context, we might expect the translation model to do a good job of determining if this is a good word choice.

We will also consider adding translation model information to the language model in a more direct fashion: by conjoining with a binary feature that indicates whether or not each target word should be considered an "island of reliability", following the concept from parsing noisy speech recognizer output (Miller, 1974). First, we apply a (very fast) binary classifier to classify each target word in the hypothesis space as either reliable or not. As a baseline, we propose to simply use lexical probabilities with a threshold for both probability and count – that is, we will consider lexical items that are either low count or low probability to be

unreliable. Then, during the decoder's search for the best hypothesis, we conjoin each LM feature with its island of reliability indicator feature. This allows the language model to weigh more heavily when the translation model is not confident. It also may prevent the language model from guiding the search away from confident, correct translations, instead choosing words that are simply more frequent. For example, conjoined features of this sort might include `Reliable-LM-Match-Len1-VeryUnlikely`, which would normally be penalized heavily by the model, but might be more likely to participate in a correct hypothesis given that it is reliable.

To evaluate the performance of our context-rich language model, we will carry out experiments on the various language pairs described in Section 2.4. Hyperparameters will be tuned on a separate held-out data set. Comparisons will be made to a baseline without any discriminative language model from the previous section and to a baseline with only the discriminative language model (but without any contextual conjunctions). We will evaluate each experimental condition using automatic metrics and perform significance testing to determine if improvements are meaningful.

Chapter 6

Summary and Timeline

6.1 Summary

We outlined three restrictions on feature sets imposed by linear models and argued that they unduely burden feature engineers:

- **Linearity.** Within each translation unit (each source sentence), if plot the objective function score of each translation hypotheses versus a feature's score of that translation hypothesis, they must form a straight line. Visually, we must be able to plot of $y = mx + b$ where y is the value of the objective function and x is the value of the feature for some m and b.
- **Monotonicity.** Within each translation unit, if we rank the translation hypotheses by an objective function, a feature must return monotonically increasing or decreasing values as we iterate over the ranked translation hypotheses.
- **Independence.** A feature's contribution to the model score may not depend on the presence of any other feature. That is, inter-dependence between features is ignored.

We then proposed two feature induction techniques to address these restrictions:

- **Discretization.** By transforming a single real-valued feature into a larger number of features, we can piece-wise learn a non-linear transformation of each of those initial features using the induced feature set. This relaxes the linearity and monotonicity restrictions on the initial feature set. We then use neighbor regularization to combat the resulting sparsity of the model.
- **Conjunction.** To learn inter-dependencies among initial features, we form conjunctions among all features that are dense enough to be reliably estimated. This enables our model to respond differently when multiple features are all fire for a hypothesis, rather than simply returning the dot product of the individual features and their weights.

Finally, we sketched three scenarios in which these techniques can be applied:

- **Simple, Targeted Features.** By discretizing and conjoining a small set of simple, intuitive features, we produce a more powerful, more effective feature set.

- **Context Rich Language Modeling.** By conjoining a small number of features of a discriminative language model with other features such as phrase boundary information, we can learn a model that behaves differently depending on the current context. For example, whether it is scoring within already-fluent phrases or accross potentially disfluent phrase boundaries. Alternatively, we might conjoining features from the translation model and language model, so that we can learn a model that behaves differently depending on how confident the translation model is, allowing the language model to speak softly in the case of phrases that are confidently translated, but not otherwise likely in monolingual training data.
- **Single system, multiple domain adaptation.** By using conjunctive features, we can learn a single model that responds differently depending on the domain or genre of the text to be translated. In the past, domain adaptation has been performed largely by training entirely new systems and models for each domain.

Discretization	
Static binning	Completed
Neighbor regularization	Proposed
Conjunctions	
Feature Selection by Inference Complexity	Proposed
Feature Selection by Sparsity	Proposed
Incremental Feature Selection via Grafting	Proposed
Feature Complexity Regularization	Proposed
Applications	
Simple, Targeted Features	Completed
Single System, Multiple Domain Adaptation	Completed
Jointly Optimized Discriminative LM	Proposed
Context Rich Language Modeling	Proposed

Figure 6.1: A summary of current progress on work falling within the scope of this thesis.

6.2 Contributions

This work will result in the following contributions to the field of statistical machine translation:

- Demonstrating that discretization can improve translation quality when compared with standard real-valued features (Sections 3.2 and 5.1)
- Demonstrating that conjunctions of features can improve translation quality (Sections 5.2, 5.3, and 5.4)
- Providing a method for reliably estimating the parameters of a discretized feature set learning (Chapter 3)
- Providing a method for dynamically performing induction of conjunctions during learning, easing the burden on feature engineers (Chapter 4)
- Providing three concrete examples of how these theoretical tools can be efficiently applied in common machine translation tasks (Chapter 5)

6.3 Timeline

We expect work on this thesis to be completed by August 2012, with incremental progress being made according to the following 17-month timeline:

April – May 2012:
Implementation of neighbor regularization (Section 3.3).
All discretization experiments on FBIS Chinese→English data set (Chapter 3).

Summer 2012:
Scale up discretization implemntation to large data sets.
Discretization experiments on all data sets.
Checkpoint: Completion of NAACL 2013 paper on discretization and neighbor regularization.

Fall 2012:
Implement iterative, dynamic induction of conjunctive features and complexity regularization.
Carry out experiments with dynamic learning of conjunctive features.
Checkpoint: Completion of ACL 2013 paper on conjunctive features.

Spring 2013:
Finalize any remaining tasks in each of the three application areas (Chapter 5).
Finalization of all experiments on individual techniques.
Begin job search.
Checkpoint: Completion of WMT 2013 paper on MT applications of discretization and conjunction.

Summer 2013:
Finalization of systems and experimentation.
Thesis writing.
Checkpoint: Circulate draft of thesis document to committee for review (July).

August 2013:
Checkpoint: Thesis defense.

References

Auli, Michael, Adam Lopez, Hieu Hoang, and Philipp Koehn. 2009. A Systematic Analysis of Translation Model Search Spaces. In *Workshop on Statistical Machine Translation*, number March, pages 224–232.

Banerjee, Satanjeev and Alon Lavie. 2005. METEOR : An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In *Proceedings of the ACL 2005 Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization*.

Brown, Peter E, Stephen A Della Pietra, Vincent J Della Pietra, and Robert L Mercer. 1993. The Mathematics of Statistical Machine Translation : Parameter Estimation. *Computational Linguistics*, 10598.

Brown, Ralf D, Rebecca Hutchinson, Paul N Bennett, Jaime G Carbonell, Peter J Jansen, and Peter Jansen. 2003. Reducing Boundary Friction Using Translation- Fragment Overlap. In *MT Summit*, pages 24–31, New Orleans.

Charniak, Eugene. 2010. Top-Down Nearly-Context-Sensitive Parsing. In *Empirical Methods in Natural Language Processing*, number October, pages 674–683.

Chen, Boxing, Roland Kuhn, George Foster, and Howard Johnson. 2011. Unpacking and Transforming Feature Functions : New Ways to Smooth Phrase Tables. In *MT Summit*, pages 269–275.

Chiang, David. 2007. Hierarchical Phrase-Based Translation. *Computational Linguistics*, 33(2):201–228, June.

Chiang, David, Yuval Marton, and Philip Resnik. 2008. Online large-margin training of syntactic and structural translation features. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing - EMNLP '08*, page 224, Morristown, NJ, USA. Association for Computational Linguistics.

Clark, Jonathan H, Chris Dyer, Alon Lavie, and Noah A Smith. 2010. Better Hypothesis Testing for Statistical Machine Translation : Controlling for Optimizer Instability. In *Association for Computational Linguistics*.

Clark, Jonathan H and Alon Lavie. 2010. LoonyBin : Keeping Language Technologists Sane through Automated Management of Experimental (Hyper) Workflows. In *Conference on Language Resources and Evaluation (LREC)*.

Clark, Jonathan H, Jonathan Weese, Byung Gyu Ahn, Andreas Zollmann, Qin Gao, Kenneth Heafield, and Alon Lavie. 2009. The Machine Translation Toolpack for LoonyBin: Automated Management of Experimental Machine Translation HyperWorkflows. *Prague Bulletin of Mathematical Linguistics*, (December):1–10.

Cortes, Corinna, Mehryar Mohri, and Afshin Rostamizadeh. 2009. Learning Non-Linear Combinations of Kernels. In *Advances in Neural Information Processing Systems (NIPS 2009)*, pages 1–9, Vancouver, Canada.

Daumé III, Hal. 2007. Frustratingly Easy Domain Adaptation. In *Association for Computational Linguistics*, number June, pages 256–263, Prague.

Della Pietra, Stephen, Vincent Della Pietra, and John Lafferty. 1995. Inducing features of random fields. Technical report, Carnegie Mellon University.

Della Pietra, Stephen, Vincent Della Pietra, and John Lafferty. 1997. Inducing Features of Random Fields. *Analysis*, 19(4):1–13.

Denkowski, Michael and Alon Lavie. 2010. Extending the METEOR Machine Translation Evaluation Metric to the Phrase Level. In *North American Association for Computational Linguistics*.

Dougherty, James, Ron Kohavi, and Mehran Sahami. 1995. Supervised and Unsupervised Discretization of Continuous Features. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 194–202, San Francisco, CA.

Dyer, Chris, Jonathan Weese, Adam Lopez, Vladimir Eidelman, Phil Blunsom, and Philip Resnik. 2010. cdec : A Decoder , Alignment , and Learning Framework for Finite-State and Context-Free Translation Models. In *Association for Computational Linguistics*, number July, pages 7–12.

Dyer, Christopher James. 2010. *A Formal Model of Ambiguity and Its Applications in Machine Translation*. Ph.D. thesis, University of Maryland.

Eisner, Jason. 2001. Expectation Semirings : Flexible EM for Learning Finite-State Transducers . *Methods*, (August):1–5.

Federico, Marcello and Nicola Bertoldi. 2006. How Many Bits Are Needed To Store Probabilities for Phrase-Based Translation? In *Workshop on Statistical Machine Translation*, number June, pages 94–101.

Gao, Qin and Stephan Vogel. 2008. Parallel Implementations of Word Alignment Tool. In *Association for Computational Linguistics Computational Linguistics*, number June, pages 49–57.

Giménez, Jesús and Lluís Màrquez. 2007. Context-aware Discriminative Phrase Selection for Statistical Machine Translation. In *Workshop on Statistical Machine Translation*, number June, pages 159–166.

Gimpel, Kevin and Noah A Smith. 2009. Feature-Rich Translation by Quasi-Synchronous Lattice Parsing. In *Empirical Methods in Natural Language Processing*.

Goodman, Joshua. 1999. Semiring Parsing. *Computational Linguistics*.

Guyon, Isabelle. 2003. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3:1157–1182.

Hopkins, Mark and Jonathan May. 2011. Tuning as Ranking. *Computational Linguistics*, pages 1352–1362.

Huang, Liang and David Chiang. 2007. Forest Rescoring: Faster Decoding with Integrated Language Models. In *Association for Computational Linguistics*, number June, pages 144–151.

Jelinek, Frederick, John Lafferty, David Magerman, Robert Mercer, Adwait Ratnaparkhi, and Salim Roukos. 1994. Decision Tree Parsing using a Hidden Derivation Model. In *Workshop on Human Language Technologies (HLT)*.

Joachims, Thorsten, Thomas Hofmann, Yisong Yue, and Chun-Nam Yu. 2009. Predicting Structured Objects with Support Vector Machines. *Communications of the ACM*, 52(11):97–104.

Klein, Dan and Christopher D Manning. 2001. Parsing and hypergraphs. *Science*, (c).

Koehn, Philipp. 2004. Statistical Significance Tests for Machine Translation Evaluation. In *Empirical Methods in Natural Language Processing*.

Kotsiantis, Sotiris and Dimitris Kanellopoulos. 2006. Discretization Techniques : A recent survey. In *GESTS International Transactions on Computer Science and Engineering*, volume 32, pages 47–58.

Kumar, Shankar, Wolfgang Macherey, Chris Dyer, and Franz Och. 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: Volume 1 - ACL-IJCNLP '09*, number August, pages 163–171, Morristown, NJ, USA. Association for Computational Linguistics.

Li, Zhifei and Jason Eisner. 2009. First- and Second-Order Expectation Semirings with Applications to Minimum-Risk Training on Translation Forests . *Language*.

Li, Zhifei and Sanjeev Khudanpur. 2008. Large-scale Discriminative n-gram Language Models for Statistical Machine Translation. In *Methods*.

Li, Zhifei, Ziyuan Wang, Sanjeev Khudanpur, and Jason Eisner. 2010. Unsupervised Discriminative Language Model Training for Machine Translation using Simulated Confusion Sets. In *Coling*, number August, pages 656–664, Bejing.

Liang, Percy, Alexandre Bouchard-c, Dan Klein, and Ben Taskar. 2006. An End-to-End Discriminative Approach to Machine Translation e. *Computational Linguistics*, (July):761–768.

Lopez, Adam. 2007. Hierarchical Phrase-Based Translation with Suffix Arrays. *Computational Linguistics*, (June):976–985.

Lopez, Adam. 2008. Tera-Scale Translation Models via Pattern Matching. In *Association for Computational Linguistics Computational Linguistics*, number August, pages 505–512.

Lopez, Adam. 2009. Translation as Weighted Deduction. *Computational Linguistics*, (April):532–540.

Lopez, Adam David. 2008. *Machine Translation by Pattern Matching*. Ph.D. thesis, University of Maryland.

Magerman, David M. 1995. Statistical Decision-Tree Models for Parsing. In *Association for Computational Linguistics*, pages 276–283.

Mccallum, Andrew. 2003. Efficiently Inducing Features of Conditional Random Fields. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Miller, Perry Lowell. 1974. A locally-organized parser for spoken input. *Communications of the ACM*, 17(11):621–630, November.

Mohri, Mehryar. 2002. Semiring Frameworks and Algorithms for Shortest-Distance Problems. *Jornal of Automata, Languages, and Combinatorics*, 7(3):321–350.

Moore, Robert C and Chris Quirk. 2007. Faster Beam-Search Decoding for Phrasal Statistical Machine Translation. In *MT Summit*.

Nguyen, Patrick, Milind Mahajan, Xiaodong He, and Microsoft Way. 2007. Training Non-Parametric Features for Statistical Machine Translation. In *Association for Computational Linguistics*.

Och, Franz J. 2003. Minimum Error Rate Training in Statistical Machine Translation. In *Association for Computational Linguistics*, number July, pages 160–167.

Och, Franz Josef and Hermann Ney. 2001. Discriminative training and maximum entropy models for statistical machine translation. *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, (July):295.

Okanohara, Daisuke. 2009. Learning Combination Features with L 1 Regularization. *Computational Linguistics*, (June):97–100.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-jing Zhu. 2002. BLEU : a Method for Automatic Evaluation of Machine Translation. In *Computational Linguistics*, number July, pages 311–318.

Papineni, Kishore A., Salim Roukos, and R. T. Ward. 1998. Maximum Likelihood and Discriminative Training of Direct Translation Models. *City*.

Perkins, Simon. 2003. Grafting : Fast , Incremental Feature Selection by Gradient Descent in Function Space. 3:1333–1356.

Riezler, Stefan and Alexander Vasserman. 2003. Incremental Feature Selection and 1 Regularization for Relaxed Maximum-Entropy Modeling. *Intelligence*, (1996).

Roark, B, M Saraclar, and M Collins. 2007. Discriminative n-gram language modeling. *Computer Speech & Language*, 21(2):373–392, April.

Roark, Brian, Murat Saraclar, Michael Collins, and Mark Johnson. 2000. Discriminative Language Modeling with Conditional Random Fields and the Perceptron Algorithm. *ReCALL*, (1995).

Sandler, S Ted. 2010. *Regularized Learning with Feature Networks*. Ph.D. thesis, University of Pennsylvania.

Sandler, Ted, Partha Pratim Talukdar, Lyle H Ungar, and John Blitzer. 2008. Regularized Learning with Networks of Features. In *Advances in Neural Information Processing Systems (NIPS 2008)*.

Shen, Libin, B C Va, and Marina Rey. 2004. Discriminative Reranking for Machine Translation. In *NLT-NAACL*.

Smith, David A and Jason Eisner. 2005. Minimum Risk Annealing for Training Log-Linear Models . *Language and Speech*.

Smith, Noah A. 2011. *Linguistic Structure Prediction*. Morgan and Claypool.

Sutton, Charles and Andrew Mccallum. 2010. An Introduction to Conditional Random Fields. *Foundations and Trends in Machine Learning (FnT ML)*, To appear.

Tsochantaridis, Ioannis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. 2004. Support Vector Machine Learning for Interdependent and Structured Output Spaces. In *International Conference on Machine Learning (ICML)*.

Wang, Zhuoran, John Shawe-Taylor, and Sandor Szedmak. 2007. Kernel Regression Based Machine Translation. In *North American Association for Computational Linguistics*, number April, pages 185–188, Rochester, N.

Whittaker, Edward and Bhiksha Raj. 2001a. Comparison of Width-wise and Length-wise Language Model Compression. *Language*.

Whittaker, Edward and Bhiksha Raj. 2001b. Quantization-based language model compression. *Language*.

Wu, Dekai, Weifeng Su, and Marine Carpuat. 2004. A Kernel PCA Method for Superior Word Sense Disambiguation. In *Association for Computational Linguistics*, Barcelona.

Xu, Peng and Frederick Jelinek. 2004. Random Forests in Language Modeling. In *Empirical Methods in Natural Language Processing*.

Yogatama, Dani, Michael Heilman, Brendan O Connor, Chris Dyer, Noah A Smith, and Bryan R Routledge. 2011. Predicting a Scientific Community's Response to an Article. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Zens, Richard. 2008. *Phrase-based Statistical Machine Translation: Models, Search, Training*. Ph.D. thesis.

Zens, Richard and Hermann Ney. 2008. Improvements in Dynamic Programming Beam Search for Phrase-based Statistical Machine Translation. In *International Workshop on Spoken Language Translation (IWSLT)*, Honolulu, HI.

Zollmann, Andreas and Ashish Venugopal. 2006. Syntax Augmented Machine Translation via Chart Parsing. *Computational Linguistics*, (June):138–141.