Locally Non-Linear Learning via Feature Induction and Structured Regularization in Statistical Machine Translation

Jonathan H. Clark

CMU-LTI-15-002

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

Thesis Committee:

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

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in Language and Information Technologies

Abstract

Linear models, which support efficient learning and inference, are the workhorses of statistical machine translation; however, linear decision rules are less attractive from a modeling perspective. The combination of 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 assumptions of linear models – a highly unintuitive and error-prone task.

We relax three assumptions of linear models by abandoning a static set of features and expanding the feature set as part of learning; we do this using two types of **feature induction operators**. First, we show that by augmenting the feature set with **conjunctions** of features, we can learn interactions among multiple features. We then show that by **discretizing** real-valued features into a finite set of indicator features, we can achieve non-linear transformations of individual features. This technique 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. 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, exposing which non-linearities are important. Throughout this thesis, we apply **structured regularization**, which considers relationships among parameters when assigning their values; this technique mitigates the data sparsity that accompanies the larger parameter sets produced by feature induction.

For Libby.

Acknowledgements

It's been a wild ride and I owe my thanks to my friends, family, and colleagues for helping me finish this thesis. First, I would like to thank my advisor Alon Lavie for encouraging me to pursue everything from linguistic syntax to machine learning and workflow management while always reminding me that machine translation is really about language and real people. He also taught me to think deeply and technically about syntax and translation.

I would also like to thank my thesis committee for their support – and their patience while I delayed this work to focus on Skype Translator. Chris Dyer has had a very active role in this thesis. He helped me see much deeper, more formal connections between machine translation and machine learning than I had previously considered. Noah Smith introduced me to modern methods in structured prediction and turned me on to declarative algorithms.[1] Thanks to Jaime Carbonell for helping me see the broader academic context of this work and to Wolfgang Macherey for helping me think about its broader industry impact.

I also owe my gratitude to Lori Levin and Bob Frederking who guided me through my first year of graduate school. Lori taught me to understand linguistic typology, the complexity of language, and how to speak the non-terminal symbols of language. Bob taught me to think about problem solving in terms of search. I am also grateful for my interactions with Stephan Vogel who on many occasions showed me the nuances (and highly effective hacks) in modern SMT decoders.

My fellow grad students at CMU have been strong collaborators and excellent friends, helping me refine many ideas in this work: Vamshi Ambati, Waleed Ammar, Nguyen Bach, Cari Bader, Victor Chahuneau, Michael Denkowski, Jeff Flannigan, Qin Gao, Kevin Gimpel, Paco Guzman, Matt Marge, Christian Monson, Alok Parlikar, Erik Peterson, Danny Rashid, Greg Hanneman, Kenneth Heafield, Michael Heilman, Austin Matthews, Jose Pablo-Gonzalez, Avneesh Saluja, Nathan Schneider, Narges Sharif-Razavian, Tae Yano, Reza Zadeh, and Andreas Zollmann.

I feel privileged to have worked at Safaba Translation Solutions during my last two years in Pittsburgh. Alon Lavie and Bob Olszewski gently pushed us to build a customized online post-editing system from the ground up in just a few months. Yet somehow the process seemed calm, orderly, and downright fun, adding to my experience of grad school instead of taking away. It was a pleasure brainstorming and hacking with you: Ryan Carlson, Pradeep Dasigi, Michael Denkowski, Greg Hanneman, Austin Matthews, Kenton Murray, and Aaron Phillips.

[1] And footnotes.

My semester as a teaching assistant for the Principles of Imperative Computation was also a great experience. Frank Pfenning, Tom Cortina, Rob Simmons, Carsten Varming, and all of the undergraduate TA's made this both a fun and intellectually challenging experience. Hilary Schuldt at the Eberly Future Faculty Program also provided some great teaching guidance, which almost certainly improved the quality of life for my undergraduates.

My interactions with external collaborators have also been very valuable. Thanks for many helpful and insightful conversations: Ondřej Bojar, Chris Callison-Burch, and Juri Ganitkevitch. A special thanks to Lane Schwartz for his help in thinking through the design of the ducttape workflow manager and implementing its new AST parser.

For the last two years, my colleagues at Microsoft Research have been a fantastic source of inspiration and intellectual stimulation. Thanks to Arul Menezes for providing a great work environment and in-depth conversations on syntax, speech recognition, and user experiences. Thanks to Chris Quirk for many whiteboard conversations including discretization strategies, neural networks, search algorithms, and software engineering. To everybody on the Skype Translator team, thank you for making the roller coaster ride toward a public product fun along the way: Anthony Aue, Vishal Chowdhary, Jacob Devlin, Sauleh Eetemadi, Christian Federmann, Qin Gao, Hany Hassan, Will Lewis, Lee Schwartz, Frank Seide, and Donovan Smith. Thanks to colleagues in MSR for many great conversations: Chris Brockett, Michel Galley, Xiaodong He, Meg Mitchell, and Kristina Toutanova.

I was lucky to have a supportive group of professors during my undergraduate work at TCU. Thanks to Dick Rinewalt, Lisa Ball, and Antonio Sanchez for helping to guide me to a career path that I love. In particular, I thank Charles Hannon who got me interested in NLP and spent a good deal of his time educating me in how to do research – without him I likely never would have followed this path. He also encouraged me to intern at Language Computer Corporation. There, during the summers of 2006 and 2007, Marian Olteanu gave me my first taste of machine translation and I've been hooked ever since.

My family has been a source of constant love and support since I can remember. My parents Lee and Emma encouraged me to pursue my dreams, letting me accumulate computer parts in my bedroom and practice a drum set in close quarters. Robert Carruthers, Linda Carruthers, Diane Hoelzeman, Chris Larson, and Melissa Larson have also been a source of support, love, and levity over the years, encouraging me to persevere.

Finally, to Libby: thank you for enduring the journey of grad school with me including the stress of deadlines, the joy of success, and the frustration of failures. You have done far more than your fair share of bringing happiness to our daughter Paige's life while I wrote this thesis. Thank you for helping me hold on to the last of my sanity and for making me smile when I need it most.

Contents

Table of Notation[1]

x	A scalar
$\mathbf{x}$	A vector
$\overline{\overline{\mathbf{x}}}$	The length of the vector $\mathbf{x}$
$\lvert\mathbf{x}\rvert$	The length of the vector $\mathbf{x}$
$\lvert x\rvert$	The absolute value of the scalar x
$\lVert\mathbf{x}\rVert_p$	The ℓ_p norm of the vector $\mathbf{x}$
$[\![\pi]\!]$	Returns 1 if the predicate π is true, 0 otherwise
$\mathcal{R}$	A regularization function
β	A regularization constant
$\mathcal{L}$	A loss function
$\mathbf{f}$	Vector of source (foreign/"French") words
$\mathbf{e}$	Vector of target ("English") words
$\mathbf{w}$	Vector of real-valued model weights
$\hat{\mathbf{e}}(\mathbf{f})$	Estimator of $\mathbf{f}$'s translation
$\tilde{\mathbf{e}}$	A human (empirical) translation
D	A complete hypothesis derivation
d	A derivation for a partial hypothesis
$H(D)$	A feature vector for a complete derivation
$h(d)$	A feature vector for a partial derivation
$\Phi_d(h)$	Feature induction operator on $h(d)$
$\triangleq$	Equal by definition
$i \frown j$	Concatenation of feature identifiers i and j
$\nabla g(x)$	Gradient of the function $g(x)$
$\mathbb{R}$	The set of real numbers
$\mathbb{N}$	The set of integers
$\mathcal{Y}(\mathbf{f})$	The set of possible target outputs for $\mathbf{f}$
$\{x_i : \pi_i\}$	Set containing all x_i that satisfy predicate π_i
$\langle a, b\rangle$	Tuple with elements a and b

[1] See also Appendix B for a comprehensive explanation of all notation and symbols.

1 Background

I like the dreams of the future better than the history of the past.

—Thomas Jefferson

Every man is a quotation from all his ancestors.

—Ralph Waldo Emerson

As the state-of-the-art in machine translation has evolved, each innovation in isolation is typically accompanied by theory, empirical evidence, and contemporary assumptions. In this chapter, we review how prevailing models for SMT evolved with an emphasis on how they have become more general – and in doing so we point out some historical conventions that have lingered despite current advances. These historical conventions include the use of a log transform to convert translation model probabilities into features and the dubious equal treatment of dense and sparse features. 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 arbitrary objective function (e.g. BLEU). Finally, we will touch on recent work that seeks to handle large numbers of features while optimizing toward a surrogate loss function (e.g. MIRA and PRO), which emulate optimization toward an arbitrary objective function.

1.1 A Brief History of Modeling in SMT

Brown et al. (1993) present a probabilistic 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 $\hat{\mathbf{e}}$:

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

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. This allows much larger monolingual data to be used to estimate the language model.

Och and Ney (2002)[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 the set of all possible translations of $\mathbf{f}$ as generated by the function $\mathcal{Y}(\mathbf{f})$, ensuring that each conditional distribution sums to unity:

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

[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 (2002) originally present these equations without the latent alignment variable $\mathbf{a}$. We present it here for completeness.

This exponential formulation has two major advantages:

1. the model allows the system designer to add arbitrary features that expose to the model information about different aspects of translation hypotheses; and

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

Och 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 exponential 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, 2002):

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

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

Och also noted that comparable performance could be achieved by using the feature $h_{TM}(\mathbf{f}, \mathbf{a}, \mathbf{e}) = \log P_{\mathbf{a}}(\mathbf{e}|\mathbf{f})$. While this formulation is incorrect from the perspective of Bayes' Rule, it empirically supports 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: It is now both a decision rule for inference and an objective function for training. Since Och's model (Equation 1.3) has the free parameters $\mathbf{w}$ (which are tuned), it was also used as an objective function for optimization using Generalized Iterative Scaling (GIS). Further, the features used in this model are unusual in that they are real-valued; though log-linear models are typically defined as having real-valued features, in practice it is more common for only indicator features to be used.

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 internal 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 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 by a model with a particular set of weights $\mathbf{w}$ according to an arbitrary error function E, using a corpus of reference translations $\tilde{\mathbf{e}}$:[4]

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

Here, we overload the notation $\hat{\mathbf{e}}$ to indicate the decoding of the entire tuning corpus.

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 a log-linear model (Equation 1.3) becomes more tenuous – in fact, MERT is simply optimizing a linear model (which also happens to share the decision rule of Equation 1.7):

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

For models that have been directly optimized toward BLEU, 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 compels us to believe that an exponential model such as Equation 1.3 is a good measure of translation quality (i.e. that it should be our objective function). In this context, the log transform of probability models into features begins to seem arbitrary:

[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.

[4] This is often notated as $\hat{\mathbf{w}} = \operatorname{argmin}_{\mathbf{w}} \sum_{(\mathbf{f},\tilde{\mathbf{e}})\in\mathbf{S}} E\left(\tilde{\mathbf{e}}, \hat{\mathbf{e}}(\mathbf{f}; \mathbf{w})\right)$, which implies that the error function must decompose linearly over sentences. This is not the case for BLEU's brevity penalty. In practice, the sufficient statistics tend to decompose linearly, making computation of the final corpus-level error surface efficient.

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). While it has been observed empirically that many common features produce better translation quality in log space rather than probability space, the question still remains whether a log transform is optimal. We will return to this question in Section 2.1.

Besides the questionable log transform, the standard model formulation also hides many parameters from the final pass optimizer. In some sense, this is desirable 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 Features on Structured Hypotheses for Efficient Inference

So far, we have establish the modern feature-based view of modeling. In this section, we briefly define attributes of features and the interplay between the features and inference. This will inform our choices in constructing feature induction operators in the next chapter. 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)

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 problem of decoding is decomposed into subproblems. For example, in a phrase-based decoder, any feature that is computable given only a single phrase pair and 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.[5]

Next, we consider the features of a standard machine translation system using a factor graph (Kschischang, Frey, and Loeliger, 2001) as a visual representation in Figure 1.1 with a focus on understanding how the features and the inference procedure are intertwined. The undirected edges in this graph should not be interpreted as a generative story nor a particular ordering of events. Note that the source sentence $\mathbf{f}$ is fully observed and so is constant for any partial derivation – that is, we may incorporate it into any feature without additional inference cost. 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 the random variable D making D both latent and structured. For each the n phrases (or grammar rules) chosen by D, each phrase is associated with some number of target terminals m_i with lexical forms e_i. The feasible assignment to each of these random vari-

[5] 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.

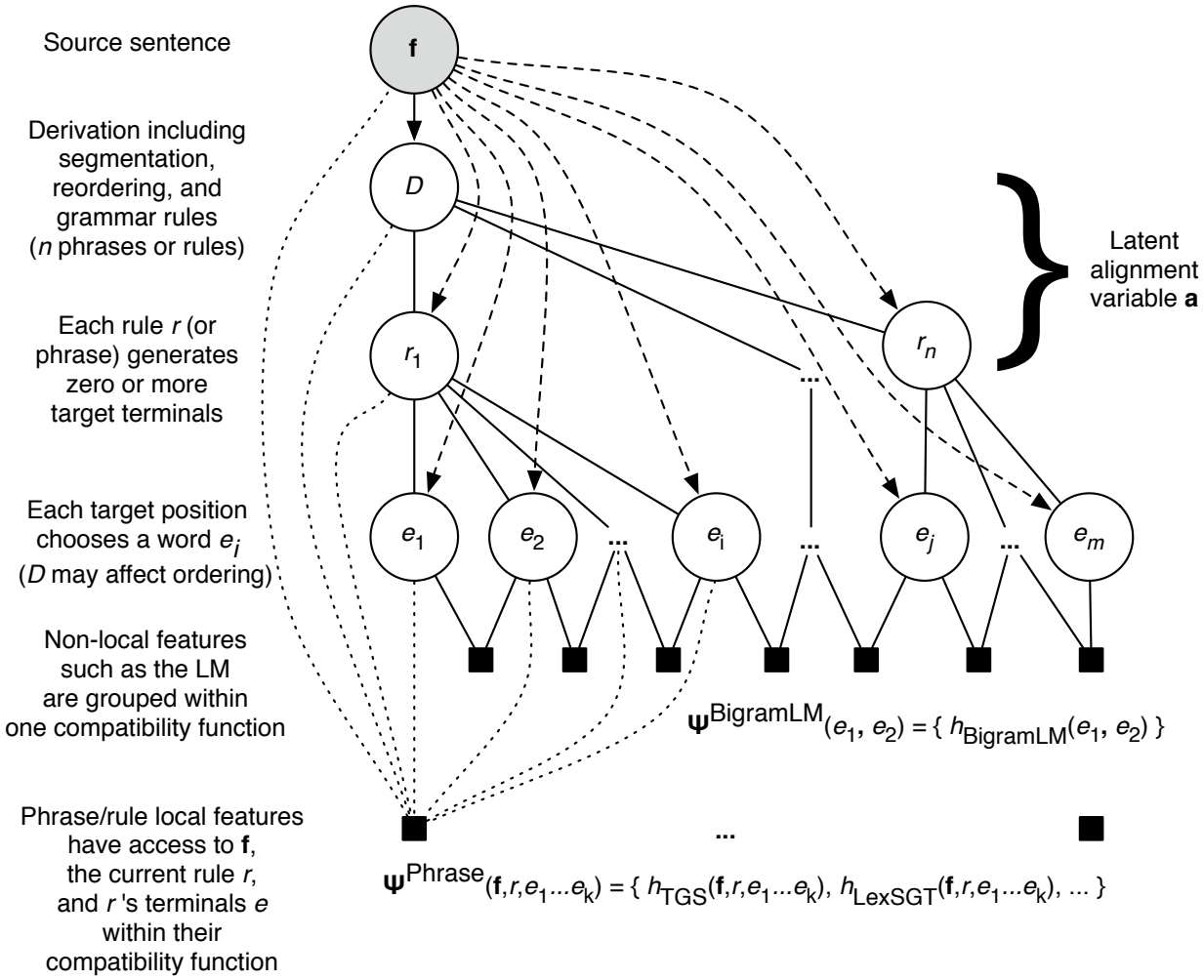


Figure 1.1: The features of a typical statistical machine translation visualized as a factor graph. The factors (square boxes) indicate phrase-local and rule-local features $\Psi^{\text{Phrase}}(\mathbf{f}, r, e_1 \ldots e_k)$ and a non-local bigram language model feature $\Psi^{\text{BigramLM}}(e_1, e_2)$. Local features are distinguished from non-local features in that they do not cross any rule boundaries (they can be computed without any state s). Throughout this thesis, we will use it to our advantage that as long as new features do not create any new factors, inference complexity remains the same. 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.

ables is heavily constrained by the grammar rules or phrase table, as well as the reordering hyperparameters of the system (reordering length limit, etc.).[6]

1.3 Inference: Linear Models over Structured Hypotheses

The formulation of features from the previous section is based largely on the needs of efficient inference. In this section, we describe inference in modern decoders since we seek to keep inference efficient throughout this thesis. Decoding a given source sentence $\mathbf{f}$ can be expressed as search over target hypotheses $\mathbf{e}$, each with an associated complete derivation D. To find the best-scoring hypothesis $\hat{\mathbf{e}}(\mathbf{f})$, a linear model applies a set of weights $\mathbf{w}$ to a complete hypothesis' feature vector $\mathbf{H}$ (Equation 1.7):

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

However, this hides many of the realities of performing inference in modern decoders. Traditional inference would be intractable if every feature were allowed access to the entire derivation D and its associated target hypothesis $\mathbf{e}$. Decoders take advantage of the fact that local features decompose over partial derivations d. For a complete derivation D, the global features $H(D)$ are an efficient summation over local features $h(d)$:

$$\hat{\mathbf{e}}(\mathbf{f}) = \operatorname*{argmax}_{\mathbf{e},D} \sum_{i=0}^{|\mathbf{H}|} w_i \underbrace{\sum_{d \in D} h_i(d)}_{H_i(D)} \tag{1.10}$$

However, this view ignores non-local features such as the language model (LM), which cannot be exactly calculated given an arbitrary partial hypothesis, which may lack both left and right context.[7] Such features require special handling including future cost estimation. In this study, we limit ourselves to local features, leaving the traditional non-local LM feature

[6] For those readers more familiar with the language of Markov random fields, since the factorization of a MRF is a priori ambiguous, we organize the features on vertices $\mathbf{V}$ into **compatibility functions**, also frequently called factors (not to be confused with factored machine translation). Each compatibility function $\Psi_{\mathbf{v}} \in \mathcal{F}(\mathbf{d})$ 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 (also sometimes referred to as potential functions), which may be applied to any random variables of the correct type. When we instantiate this random field for a 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.

[7] This is especially problematic for chart-based decoders.

unchanged. In general, feature locality is relative to a particular **structured search space**, and is not necessarily related to the **structured features** described in Section 6.3.2.

Decoders incorporate non-local features by maintaining a state object s, which includes the minimal information from nearby partial derivations required to compute all features (local features never use s). As the state object becomes more complex, the search for a best derivation is prone to become slower or lossier. Incorporating the state s, we reach a final formulation:[8]

$$\hat{\mathbf{e}}(\mathbf{f}) = \operatorname*{argmax}_{\mathbf{e},D} \sum_{i=0}^{|\mathbf{H}|} w_i \underbrace{\sum_{\langle d,s \rangle \in D} h_i(d,s)}_{H_i(D)} \tag{1.11}$$

This view of the inference problem provides us with a clear picture of how decoders make inference efficient. The decoder can quickly aggregate the scores of local features from local derivations with state from nearby partial derivations updating the ranking as the search progresses. 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. The non-local state s is the key reason why exact dynamic programming is no longer possible in MT decoding. 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 accommodate 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**[9] 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

[8] From the view of Markov random fields and compatibility functions, we could substitute the state object for a decomposition over compatibility functions Ψ: $\hat{\mathbf{e}}(\mathbf{f}) = \operatorname{argmax}_{\mathbf{d},\mathbf{e}} \sum_{i=0}^{|\mathbf{H}|} w_i \underbrace{\sum_{\Psi_\mathbf{v} \in \mathcal{F}(\mathbf{d})} h_i(\Psi_\mathbf{v})}_{H_i(\mathbf{d})}$

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

stack comparable so that the decoder does not opportunistically make local choices that will inevitably lead to worse translations in a larger context.

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: From this perspective, we see a clear distinction between the global features H, which is important from the perspective of the final linear model, and the myopic local feature functions h, which is important for inference efficiency (and often the way features are formally described in the literature). This distinction will become important in Chapter 6.

Not only are current decoding algorithms efficient, but they also produce empirically good results. While exact decoding algorithms exist for both phrasal and syntactic machine translation models (Rush, Chang, and Collins, 2013), when Chang and Collins (2011) applied Lagrangian Relaxation to perform exact decoding of phrase-based models, they observed that the quality of phrase-based decoding with a beam size of 100 was effectively the same (insignificantly better) as using *exact* decoding, according to BLEU.

1.4 Learning: Non-Convex Objectives in High Dimensions

Throughout this thesis, we will apply the techniques of feature induction (Section 2.3) and regularization (Section 2.4). While MERT has proven to be a strong baseline, traditional MERT does not scale to larger feature sets (such as those resulting from feature induction) in terms of both inefficiency and overfitting (Hopkins and May, 2011).

Because of this, we use the PRO optimizer (Hopkins and May, 2011) as our baseline learner since it has been shown to perform comparably to MERT for a small number of features, and to significantly outperform MERT for a large number of features (Hopkins and May, 2011; Ganitkevitch et al., 2012). PRO works by sampling pairs of hypotheses from the decoder's k-best list and then providing these pairs of binary training examples (where the gold binary label is a correctly ranked or not) to a standard binary classifier to obtain a new set of weights for the linear model. This procedure of sampling training pairs and then optimizing the pairwise rankings is repeated for a specified number of iterations. We select and sample exemplars uniformly at random to avoid pathologically long negative samples (Nakov, Guzmán, and Vogel, 2013).

Other recent MT optimizers such as MIRA (Chiang, Marton, and Resnik, 2008), the linear structured SVM (Cherry and Foster, 2012a), Rampion (Gimpel and Smith, 2012), HOLS (Flanigan, Dyer, and Carbonell, 2013), AROW (Chiang, 2012) and regularized MERT (Galley et al., 2013) have also been shown to be effective in optimizing larger feature sets. Any of these optimizers that are scale insensitive or can be reformulated as scale insensitive

(Galley et al., 2013) are compatible with the structured regularization techniques described in this thesis.[10]

[10] Since we eventually dispense with nearly all of the original dense features in Chapter 6 and all of our regularization terms are scale sensitive, one would need to use the ℓ_1-renormalized variant of regularized MERT.

2 Overview

A theory must be tempered with reality.

—Jawaharlal Nehru

So once you know what the question actually is, you'll know what the answer means.

—Douglas Adams, The Hitchhiker's Guide to the Galaxy

FEATURE ENGINEERING remains a significant human cost in the development of state-of-the-art machine translation systems. While this is partly due to the complex nature of human language, known theoretical limitations of current modeling and learning procedures are also to blame. This thesis seeks to reduce the cost of feature engineering and improve translation quality by addressing those issues through improved learning. In this chapter, we outline those limitations, discuss how they are a burden in practice, and state the claims and contributions of this thesis.

2.1 Theoretical Motivation: Assumptions and Pitfalls of Linear Modeling

We follow Nguyen, Mahajan, and He (2007) in enumerating the restrictions placed upon a feature set within a linear model, which are assumed to 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 $P(\mathbf{e})$.

[1] The authors originally presented these in the context of a log-linear model. However, these properties also hold for linear models.

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 most important when considering **initially**[2] **continuous features** such as probabilities and counts since these assumptions are trivially satisfied for **initially discrete features** such as 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 fail 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. 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.

	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.

[2]We distinguish initial features from the induced features described in Section 2.3.

	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 produce 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 interdependence 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.

2.1.1 Feature Non-Linearity and Separability

Unlike models that rely primarily on a large number of sparse indicator features, state-of-the-art machine translation systems rely heavily on a small number of dense real-valued features. However, unlike indicator features, real-valued features are more prone to the limitations of linear models, as we just discussed.

Decoders use a linear model to rank hypotheses, selecting the highest-*ranked* derivation. Since the absolute score of the model is irrelevant, non-linear responses are useful only in cases where they elicit novel rankings. In this section, we will discuss these cases in terms of separability. Here, we are separating the correctly ranked pairs of hypotheses from the incorrect in the implicit pairwise rankings defined by the total ordering on hypotheses provided by our model.

When the local feature vectors h of each oracle-best[3] hypothesis (or hypotheses) are distinct from those of all other competing hypotheses, we say that the inputs are **oracle separable** given the feature set. If there exists a weight vector that distinguishes the oracle-best ranking from all other rankings under a linear model, we say that the inputs are **linearly separable** given the feature set. If the inputs are oracle separable but not linearly separable, we say that there are **uncaptured non-linearities** that are unexplained by the feature set. For example, this can happen if a feature is positively related to quality in some regions but negatively related in other regions (i.e. non-monotonicity with regard to the objective).

As we add more sentences to our corpus, the likelihood of achieving separability decreases. For a given corpus, if all hypotheses are oracle separable, we can always produce the oracle translation – assuming an optimal (and potentially very complex, overparameterized) model and weight vector. If our hypothesis space also contains all reference translations, we can always recover the reference. In practice, both of these conditions are typically

[3]We define the oracle-best hypothesis in terms of some external quality measure such as BLEU.

violated to a certain degree. However, if we modify our feature set such that some lower-ranked higher-quality hypothesis can be separated from all higher-ranked lower-quality hypotheses, then we can improve translation quality. For this reason, we argue that separability remains an informative tool for thinking about modeling in MT.

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 assumptions 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 on the human system designers.

2.3 Approach: Feature Induction to Combat Underfitting

The limitations of linear models make them subject to **underfitting** – not fitting the tuning data as tightly as desired. In the process of relaxing the assumptions of linear models, we will apply feature induction to provide the model with more degrees of freedom. Our feature induction operators produce features that are more specific than the input feature set – as if the features have been split into more fine-grained categories:

- The **conjoin** operator will multiply two real-valued features or compute the logical conjunction of two indicator features; this relaxes the linear assumption of independence.

- The **discretize** operator will split a single real-valued feature into bins; this relaxes the linear assumptions of linearity and monotonicity.

This enables our transformed model to produce non-linear responses with regard to the initial feature set **H** while inference remains linear with regard to the optimized parameters $\mathbf{w}'$. We visualize the difference between the original linear model and our formulation of learning with feature induction in Figure 2.5. Importantly, our transformed feature set requires no additional non-local state information for inference. In our experiments, we perform feature induction once at the beginning of learning, though other formulations are possible.

We define **feature induction** as a function $\Phi(\mathbf{h})$ that takes the result of the feature vector $\mathbf{h}(\mathbf{f}, d, s) \in \mathbb{R}^{|\mathbf{h}|}$ and returns a set of $|\mathbf{h}'|$ tuples $\langle y', j \rangle$ where $y' \in \mathbb{R}$ is a transformed feature value and j is the transformed feature index. Building on equation 1.10, we can apply feature induction as follows:

$$\hat{\mathbf{e}}(\mathbf{f}) = \underset{\mathbf{e},\mathbf{D}}{\operatorname{argmax}} \sum_{\langle d,s \rangle \in D} \underbrace{\sum_{\langle y',j \rangle \in \Phi(\mathbf{h}(d,s))} w'_j y'}_{\mathbf{h}'(\mathbf{f},\mathbf{e},\mathbf{d})} \tag{2.1}$$

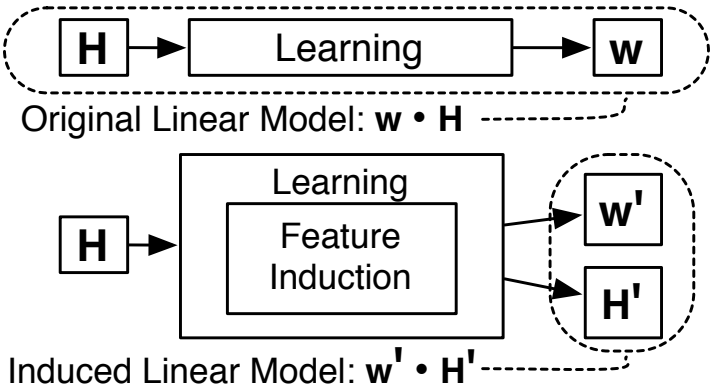


Figure 2.5: **Top:** A traditional learning procedure, assigning a set of weights to a fixed feature set. **Bottom:** Our feature induction technique expands the feature set as part of learning, while still producing a linear model for inference, albeit with more features.

2.4 Challenges: Regularization to Combat Overfitting

While feature induction will allow us to more tightly fit the tuning data, we also run the risk of **overfitting** the tuning data such that the learned parameters no longer generalize to held-out test data. To combat the overfitting tendencies introduced by feature induction, we will apply various types of **regularization**, which encourage the learning procedure to make use of the induced features only as necessary. When the distribution encouraged by the regularizer matches that of the population data (i.e. a held-out test set), we can optimize larger feature sets on smaller data while still improving the quality of our predictions. We will return to this basic trade off between underfitting and overfitting as a theme throughout this thesis.

2.5 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 due to uncaptured non-linearities.

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

1. by expanding the feature set during learning we can render the learned model non-linear with regard to the initial feature set;

2. discretization of initial features will result in a significant improvement in overall translation quality as measured by automatic metric scores; and

3. conjunction of initial features will result in a significant improvement in overall translation quality as measured by automatic metric scores.

To support these claims, this thesis makes the following contributions:

1. a method of feature discretization to relax the properties of linearity and monotonicity (Chapter 6);

2. a method of feature conjunction to relax the property of independence that performs feature induction learning (Chapter 4);

3. evidence 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 (Chapter 6);

4. evidence that conjunctions of features can improve translation quality (Chapters 4 and 5); and

5. three real-world scenarios where these theoretical tools improve machine translation quality (Chapters 4,5, and 6).

In order to achieve these goals, we further contribute:

6. a method of evaluating the impact of experimental results that is robust to optimizer instability such as randomized optimization procedures and experimental configurations having high variance due to large feature sets (Chapter 3);

7. a method of regularizing discretized features, allowing their weights to be reliably estimated (Chapter 6); and

8. a method of regularizing conjoined feature sets having intuitive groupings (Chapter 4).

9. a method of regularizing very large conjoined feature sets without *a priori* obvious groupings, allowing them to be used with conventionally-sized tuning sets (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.

2.6 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 disambiguation 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. Additionally, 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 as 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 some 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. Second, traditional learning procedures for neither MT nor decision trees can be applied due to the structured hypothesis space.

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 and Elisseeff (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, Mahajan, and He (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.

Neural networks have recently gained popularity as a means of learning new feature representations with non-linear relationships. Recent work has investigated recurrent neural networks (Schwenk, 2012; Auli et al., 2013; Kalchbrenner and Blunsom, 2013), which add non-local state to inference, and additive neural networks (Liu et al., 2013a), which learn locally non-linear mappings. While these models have shown promise as methods of augmenting existing models, they have not yet offered a path for replacing or transforming existing real-valued features. This may be partially due to requiring many more internal parameters to perform otherwise simple transformations that are better represented as simple discretization or conjunction operations (Chapter 6). The uninterpretable nature of representations learned by neural networks also means that (1) they are opaque to post-hoc analysis; and (2) it is more difficult to encode prior knowledge using techniques such as regularization, making them less appropriate as data becomes more sparse.

2.7 Roadmap

The remainder of this thesis is structured as follows.

- In Chapter 3, we discuss how to evaluate machine translation experiments given a randomized optimizer

We then explore increasingly complex feature induction operators along with corresponding regularization techniques.

- In Chapter 4, we explore a domain adaptation task using conjunctions between source domain indicator features and traditional dense features. When combined with a group regularizer that takes feature complexity into account, we observe an improvement.
- In Chapter 5, we add sparse source-target lexical indicator conjunctions and target-target lexical indicator conjunctions. By applying a regularization technique that induces groups during learning, we are able to optimize our parameters on a traditional tuning data set while retaining nearly 1 million active features.
- In Chapter 6, we replace the traditional dense features with discretized features; rather than discovering structure among features, we induce features that are by definition structurally related. We find that when this structure is leveraged by neighbor regularization, a novel regularization strategy, we can discard the traditional log transform while still observing improvements in quality.
- In Chapter 7, we discuss how to build conjunctions on top of discretized features and generalize neighbor regularization across multiple spatial relationships simultaneously.

Finally, we summarize and discuss future directions in Chapter 8.

Evaluation: Robust Hypothesis Testing under Randomized Optimization[1]

3

Insanity: doing the same thing over and over and expecting different results.

—John Dryden, The Spanish Friar

OBJECTS IN MIRROR ARE CLOSER THAN THEY APPEAR.

—U.S. Federal Vehicle Safety Standards, Section 571.111

*Trust, but verify. (*Доверяй, но проверяй*)*

—Old Russian Proverb

WHEN ANALYZING THE RESULTS of a statistical machine translation experiment, a researcher seeks to determine whether some innovation (e.g., a new feature, model, or inference algorithm) improves translation quality in comparison to a baseline system. To answer this question, he runs an experiment to evaluate the behavior of the two systems on held-out data. In this chapter, we consider how to make such experiments more statistically reliable. We provide a systematic analysis of the effects of optimizer instability—an extraneous variable that has previously been seldom controlled for—on experimental outcomes, and outline a methodology for reporting results more accurately. While these controls are generally good practice for the field, they are particularly important in enabling us to draw strong conclusions about the results in this thesis since we push the limits of optimizers with high dimensional feature sets.

The need for statistical hypothesis testing for machine translation (MT) has been acknowledged since at least Och (2003). In that work, the proposed method was based on bootstrap resampling and was designed to improve the statistical reliability of results by controlling for randomness across test sets. However, there has been no consistently used

[1]Elements of this chapter were originally published as J. Clark, C. Dyer, A. Lavie, N. Smith, "Better Hypothesis Testing for Statistical Machine Translation: Controlling for Optimizer Instability", *ACL*. July 2011.

strategy that controls for the effects of unstable estimates of model parameters.[2] While the existence of optimizer instability has long been an acknowledged problem, it was previously only infrequently discussed in relation to the reliability of experimental results, and, to our knowledge, this is the first systematic study of its effects on hypothesis testing. In this chapter, we present a series of experiments demonstrating that optimizer instability can account for substantial amount of variation in translation quality, which, if not controlled for, could lead to incorrect conclusions both by automatic metrics and human evaluators. We then show that it is possible to control for this variable with a high degree of confidence with only a few replications of the experiment and close this chapter by outlining best practices for significance testing for machine translation that take these factors into consideration. Following the initial publication of this work, the research community has largely embraced this methodology with over 125 recent publications citing the recommendations in Clark et al. (2011).

3.1 Nondeterminism and Other Optimization Pitfalls

Statistical machine translation systems consist of a model whose parameters are estimated to maximize some objective function on a set of development data. Because the standard objectives (e.g., 1-best BLEU, expected BLEU, marginal likelihood) are not convex, only approximate solutions to the optimization problem are available, and the parameters learned are typically only locally optimal and may strongly depend on parameter initialization and search hyperparameters. Additionally, stochastic optimization and search techniques, such as minimum error rate training (Och, 2003) and Markov chain Monte Carlo methods (Arun et al., 2010), constitute a second, more obvious source of noise in the optimization procedure. Online subgradient techniques such as MIRA (Crammer et al., 2006; Chiang, Marton, and Resnik, 2008) have an implicit stochastic component as well based on the order of the training examples.

This variation in the parameter vector affects the quality of the model measured on both development data and held-out test data, independently of any experimental manipulation. Thus, when trying to determine whether the difference between two measurements is significant, it is necessary to control for variance due to noisy parameter estimates. This can be done by replication of the optimization procedure with different starting conditions (e.g., by running MERT many times).

[2] We hypothesize that the convention of "trusting" BLEU score improvements of greater than 1.0 BLEU, is not merely due to an appreciation of what qualitative difference a particular quantitative improvement will have, but also an implicit awareness that ignoring optimizer instability leads to results that are not consistently reproducible.

Unfortunately, common practice in reporting machine translation results was previously to run the optimizer once per system configuration and to draw conclusions about the experimental manipulation from this single sample. However, it could be that a particular sample is on the "low" side of the distribution over optimizer outcomes (i.e., it results in relatively poorer scores on the test set) or on the "high" side. The danger here is obvious: a high baseline result paired with a low experimental result could lead to a useful experimental manipulation being incorrectly identified as useless. On the other hand, a low baseline paired with a high experimental result could lead to a negligible gain (or even a degradation) being reported as beneficial. We now turn to the question of how to reduce the probability of falling into this trap.

3.2 Related Work

The use of statistical hypothesis testing has grown apace with the adoption of empirical methods in natural language processing. Bootstrap techniques (Efron, 1979; Wasserman, 2003) are widespread in many problem areas, including for confidence estimation in speech recognition (Bisani and Ney, 2004), and to determine the significance of MT results (Och, 2003; Koehn, 2004; Zhang, Vogel, and Waibel, 2004; Zhang and Vogel, 2010). The bootstrap relies on subsampling to create virtual test corpora by subsampling the actual test corpus; these virtual test sets are then used to estimate the probability of the observed difference occurring due to chance. Such methods are particularly adept at detecting when a small number of data points are disproportionately affecting the results, which suggests the results will not generalize (we quantify this phenomena as s_{sel}, below).

Approximate randomization (AR) has been proposed as a more reliable technique for MT significance testing, and evidence suggests that it yields fewer type I errors (i.e., claiming a significant difference where none exists; Riezler and Maxwell, 2005). AR creates virtual test sets by repeatedly shuffling the hypotheses of two systems between two virtual sets; the difference between these virtual sets is calculated to estimate the differences we would observe by chance (as opposed to measuring the systematic effects of an experimenter's intervention). AR is discussed in further detail in Section 3.3.3.

Other uses of significance tests in NLP include the MUC-6 evaluation (Chinchor, 1993) and parsing (Cahill et al., 2008). Yeh (2000) also discusses several alternatives for significance testing in NLP. However, these previous methods have all been applied in ways that assume model parameters are elements of the system rather than extraneous variables. This is unrealistic since downstream consumers of these reported innovations typically do *not* take the model parameters as part of the implementation (in fact, full model parameters are rarely, if ever published – nor should we expect them to be). Downstream consumers

will be subject to any optimizer noise, which is a motivating factor for quantifying and reporting it.

Prior work on optimizer noise in MT has focused primarily on *reducing* optimizer instability (whereas our concern is how to deal with optimizer noise, when it exists). Foster and Kuhn (2009) measured the instability of held-out BLEU scores across 10 MERT runs to improve tune/test set correlation. However, they only briefly mention the implications of the instability on significance. Cer, Jurafsky, and Manning (2008) explored regularization of MERT to improve generalization on test sets. Moore and Quirk (2008) explored strategies for selecting better random "restart points" in optimization. Cer, Manning, and Jurafsky (2010) measured the standard deviation over 5 MERT runs as part of exploring which metric provided the best translation quality when used as the objective function of optimization.

Following the initial publication of this work, others have considered how to control for this variation by randomly subsampling and permuting the optimization corpus (Cherry and Foster, 2012b).

3.3 Experiments

In our experiments, we ran the MERT optimizer to optimize BLEU on a held-out development set many times to obtain a set of optimizer samples on two different pairs of systems (4 configurations total). Each pair consists of a baseline system (System A) and an "experimental" system (System B), which previous research has suggested will perform better.

The first system pair contrasts a baseline phrase-based system (Moses) and experimental hierarchical phrase-based system (Hiero), which were constructed from the Chinese-English BTEC corpus (0.7M words), the later of which was decoded with the cdec decoder (Koehn et al., 2007; Chiang, 2007; Dyer et al., 2010). The second system pair contrasts two German-English Hiero/cdec systems constructed from the WMT11 parallel training data (98M words).[3] The baseline system was trained on unsegmented words, and the experimental system was constructed using the most probable segmentation of the German text according to the CRF word segmentation model of Dyer (2009). The Chinese-English systems were optimized 300 times, and the German-English systems were optimized 50 times.

Our experiments used the default implementation of MERT that accompanies each of the two decoders. The Moses MERT implementation uses 20 random restart points per iteration, drawn uniformly from the default ranges for each feature, and, at each iteration,

[3] http://statmt.org/wmt11/

Metric	System	Avg	$\overline{s_{\text{sel}}}$	s_{dev}	s_{test}
BTEC Chinese-English ($n = 300$)					
BLEU ↑	System A	48.4	1.6	0.2	0.5
	System B	49.9	1.5	0.1	0.4
MET ↑	System A	63.3	0.9	-	0.4
	System B	63.8	0.9	-	0.5
TER ↓	System A	30.2	1.1	-	0.6
	System B	28.7	1.0	-	0.2
WMT German-English ($n = 50$)					
BLEU ↑	System A	18.5	0.3	0.0	0.1
	System B	18.7	0.3	0.0	0.2
MET ↑	System A	49.0	0.2	-	0.2
	System B	50.0	0.2	-	0.1
TER ↓	System A	65.5	0.4	-	0.3
	System B	64.9	0.4	-	0.4

Table 3.1: Measured standard deviations of different automatic metrics due to test-set and optimizer variability. s_{dev} is reported only for the tuning objective function BLEU.

200-best lists were extracted with the current weight vector (Bertoldi and Federico, 2009). The cdec MERT implementation performs inference over the decoder search space which is structured as a hypergraph (Kumar et al., 2009). Rather than using restart points, in addition to optimizing each feature independently, it optimizes in 5 random directions per iteration by constructing a search vector by uniformly sampling each element of the vector from $(-1, 1)$ and then renormalizing so it has length 1. For all systems, the initial weight vector was manually initialized so as to yield reasonable translations as this is common practice. However, we note that even more optimizer instability could have been elicited by starting from uniform weights. Results are reported using BLEU (Papineni et al., 2002), METEOR[4] (Banerjee and Lavie, 2005; Denkowski and Lavie, 2010), and TER (Snover et al., 2006).

3.3.1 Extraneous variables in one system

In this section, we describe and measure (on the example systems just described) three extraneous variables that should be considered when evaluating a translation system. We quantify these variables in terms of standard deviation s, since it is expressed in the same units as the original metric. Refer to Table 3.1 for the statistics.

[4] METEOR version 1.2 with English ranking parameters and all modules.

Local optima effects s_{dev} The first extraneous variable we discuss is the stochasticity of the optimizer. As discussed above, different optimization runs find different local maxima. The noise due to this variable can depend on many number of factors, including the number of random restarts used (in MERT), the number of features in a model, the number of references, the language pair, the portion of the search space visible to the optimizer (e.g. 10-best, 100-best, a lattice, a hypergraph), and the size of the tuning set. Unfortunately, there is no proxy to estimate this effect as with bootstrap resampling. To control for this variable, we must run the optimizer multiple times to estimate the spread it induces on the *development set*. Using the n optimizer samples, with m_i as the translation quality measurement of the development set for the i^{th} optimization run, and $\overline{m}$ is the average of all m_i's, we report the standard deviation over the tuning set as s_{dev}:

$$s_{\text{dev}} \triangleq \sqrt{\sum_{i=1}^{n} \frac{(m_i - \overline{m})^2}{n-1}}$$

A high s_{dev} value may indicate that the optimizer is struggling with local optima and changing hyperparameters (e.g. more random restarts in MERT) could improve system performance.

Overfitting effects s_{test} As with any optimizer, there is a danger that the optimal weights for a tuning set may not generalize well to unseen data (i.e., we overfit). For a randomized optimizer, this means that parameters can generalize to different degrees over multiple optimizer runs. We measure the spread s_{test} induced by optimizer randomness on the test set metric score, as opposed to the overfitting effect in isolation. The computation of s_{test} is identical to s_{dev} except that the m_i are the translation metrics calculated on the *test set*. In Table 3.1, we observe that $s_{\text{test}} > s_{\text{dev}}$, indicating that optimized parameters are likely not generalizing well.

Test set selection $\overline{s_{\text{sel}}}$ The final extraneous variable we consider is the selection of the test set itself. A good test set should be representative of the domain or language for which experimental evidence is being considered. However, with only a single test corpus, we may have unreliable results because of idiosyncrasies in the test set. This can be mitigated in two ways. First, replication of experiments by testing on multiple, non-overlapping test sets can eliminate it directly. Since this is not always practical (more test data may not be available), the widely-used bootstrap resampling method (Section 3.2) also controls for test set effects by resampling multiple "virtual" test sets from a single set, making it possible to infer distributional parameters such as the standard deviation of the translation metric over

(very similar) test sets.[5] Furthermore, this can be done for each of our optimizer samples. By averaging the bootstrap-estimated standard deviations over optimizer samples, we have a statistic that jointly quantifies the impact of test set effects and optimizer instability on a test set. We call this statistic $\overline{s_{\text{sel}}}$. Different values of this statistic can suggest methodological improvements. For example, a large $\overline{s_{\text{sel}}}$ indicates that more replications will be necessary to draw reliable inferences from experiments on this test set, so a larger test set may be helpful.

To compute $\overline{s_{\text{sel}}}$, assume we have n independent optimization runs which produced weight vectors that were used to translate a test set n times. The test set has ℓ segments with references $\mathbf{R} = \langle R_1, R_2, \ldots, R_\ell \rangle$. Let $\mathcal{X} = \langle \mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n \rangle$ where each $\mathbf{X}_i = \langle X_{i1}, X_{i2}, \ldots, X_{i\ell} \rangle$ is the list of translated segments from the i^{th} optimization run list of the ℓ translated segments of the test set. For each hypothesis output $\mathbf{X}_i$, we construct k bootstrap replicates by drawing ℓ segments uniformly, *with replacement*, from $\mathbf{X}_i$, together with its corresponding reference. This produces k virtual test sets for each optimization run i. We designate the score of the j^{th} virtual test set of the i^{th} optimization run with m_{ij}. If $\overline{m_i} = \frac{1}{k}\sum_{j=1}^{k} m_{ij}$, then we have:

$$s_i \triangleq \sqrt{\sum_{j=1}^{k} \frac{\left(m_{ij} - \overline{m_i}\right)^2}{k-1}}$$

$$\overline{s_{\text{sel}}} \triangleq \frac{1}{n}\sum_{i=1}^{n} s_i$$

3.3.2 Comparing Two Systems

In the previous section, we gave statistics about the distribution of evaluation metrics across a large number of experimental samples (Table 3.1). Because of the large number of trials we carried out, we can be extremely confident in concluding that for both pairs of systems, the experimental manipulation accounts for the observed metric improvements, and furthermore, that we have a good estimate of the magnitude of that improvement. However, it is not generally feasible to perform as many replications as we did, so here we turn to the question of how to compare two systems, accounting for optimizer noise, but without running 300 replications.

We begin with a visual illustration how optimizer instability affects test set scores when comparing two systems. Figure 3.1 plots the histogram of the 300 optimizer samples each

[5] Unlike actually using multiple test sets, bootstrap resampling does not help to re-estimate the *mean* metric score due to test set spread since the mean over bootstrap replicates is approximately the aggregate metric score.

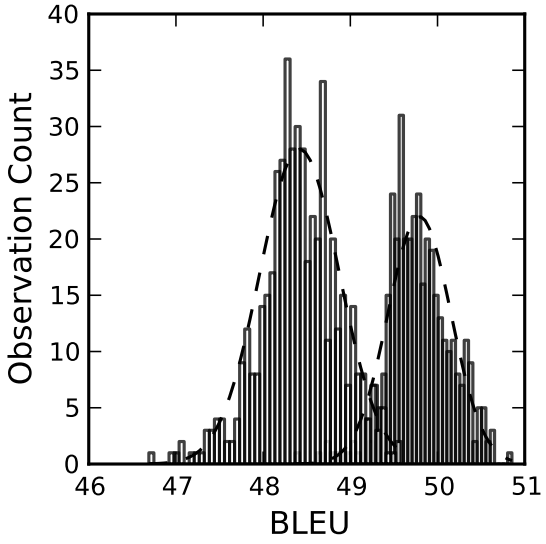


Figure 3.1: Histogram of test set BLEU scores for the BTEC phrase-based system (left) and BTEC hierarchical system (right). While the difference between the systems is 1.5 BLEU in expectation, there is a non-trivial region of overlap indicating that some random outcomes will result in little to no difference being observed.

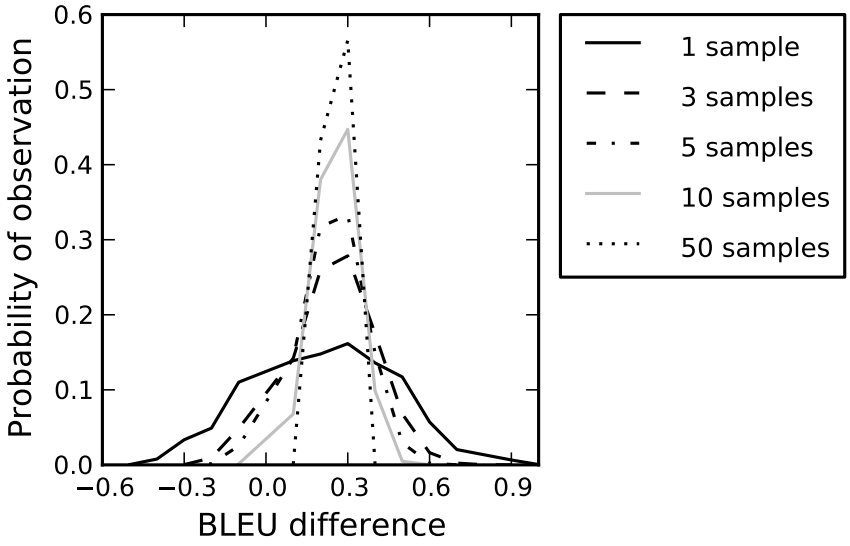


Figure 3.2: Relative frequencies of obtaining differences in BLEU scores on the WMT system as a function of the number of optimizer samples. The expected difference is 0.2 BLEU. While there is a reasonably high chance of observing a non-trivial improvement (or even a decline) for 1 sample, the distribution quickly peaks around the expected value given just a few more samples.

			p-value	
n	System A	System B	BTEC	WMT
1	high	low	0.25	0.95
1	median	median	0.15	0.13
1	low	high	0.0003	0.003
			p-value (95% CI)	
5	**random**	**random**	0.001–0.034	0.001–0.38
50	**random**	**random**	0.001–0.001	0.001–0.33

Table 3.2: Two-system analysis: AR p-values for three different "single sample" scenarios that illustrate different pathological scenarios that can result the optimizer produces a solution that is on the "low" or "high" side of the distribution over test set scores. For "random," we simulate an experiment with n optimization replications by drawing n optimized system outputs from our pool and performing AR; this simulation was repeated 250 times and the 95% CI of the AR p-values is reported.

from the two BTEC Chinese-English systems. The phrase-based system's distribution is centered at the sample mean 48.4, and the hierarchical system is centered at 49.9, a difference of 1.5 BLEU, corresponding to the widely replicated result that hierarchical phrase-based systems outperform conventional phrase-based systems in Chinese-English translation. Crucially, although the distributions are distinct, there is a non-trivial region of overlap, and experimental samples from the overlapping region could suggest the opposite conclusion!

To further underscore the risks posed by this overlap, Figure 3.2 plots the relative frequencies with which different BLEU score deltas will occur, as a function of the number of optimizer samples used.

3.3.3 When is a difference significant?

To determine whether an experimental manipulation results in a statistically reliable difference for an evaluation metric, we use a stratified approximate randomization (AR) test. This is a nonparametric test that approximates a paired permutation test by sampling permutations (Noreen, 1989). AR estimates the probability $p(\text{CHANCE})$, also known as a p-value, that a measured difference in metric scores arose by chance by randomly exchanging sentences between the two systems. If there is no significant difference between the systems (i.e., the null hypothesis is true), then this shuffling should not change the computed metric score. Crucially, this assumes that the samples being analyzed are representative of all extraneous variables that could affect the outcome of the experiment. Therefore, we must include multiple optimizer replications. Also, since metric scores (such as BLEU) are in general not comparable across test sets, we stratify, exchanging only hypotheses that cor-

Algorithm 1 APPROXIMATERANDOMIZATION

$c \leftarrow 0$
▷ Compute observed difference o between systems A and B using metric m
(e.g. BLEU averaged over replicas)
$o \leftarrow |m(A) - m(B)|$
▷ Shuffle data N times
for $i \in [1, N]$ **do**
▷ For each sentence in the test corpus
for $j \in [1, |\mathbf{R}|]$ **do**
Shuffle hypothesis j between virtual system A and B with probability 0.5
end for
▷ Compute difference between test statistics for this virtual by-chance trial
$v \leftarrow |m(A_i) - m(B_i)$
▷ Accumulate count of by-chance trials where the difference exceeds the observed difference
if $v > o$ **then**
$c \leftarrow c + 1$
end if
end for
▷ Compute relative frequency to estimate the probability that observed difference occurred by chance
$p(\text{CHANCE}) \leftarrow \frac{c+1}{N+1}$
return $p(\text{CHANCE})$

respond to the same sentence. We sketch the algorithm for AR in Algorithm 1, following Riezler and Maxwell (2005).

Table 3.2 shows the p-values computed by AR, testing the significance of the differences between the two systems in each pair. The first three rows illustrate "single sample" testing practice. Depending on luck with MERT, the results can vary widely from insignificant (at $p > .05$) to highly significant.

The last two lines summarize the results of the test when a small number of replications are performed, as ought to be reasonable in a research setting. In this simulation, we randomly selected n optimizer outputs from our large pool and ran the AR test to determine the significance. We repeated this procedure 250 times and analyze the 95% confidence interval (CI) – the middle 95% of the samples. The p-values reported are the p-values at the edges of the 95% confidence interval according to AR seen in the 250 simulated comparison scenarios.

These results indicate that we are very likely to observe a significant difference for BTEC at $n = 5$, and a very significant difference by $n = 50$ (Table 3.2). Similarly, we see this trend in the WMT system: performing more replications leads to more significant results

(equivalently, the probability of the result occurring due to chance falls), which will be easier to reproduce.

Based on the average (true) performance of the systems reported in Table 3.1, we *expect* significance over a large enough number of independent trials. The number of independent trials to achieve significance will depend on the magnitude of the true difference between the systems and the magnitude of the noise generated by the optimizer. A larger number of independent trials will have the following effects:

- reduce the likelihood of either incorrectly reporting a positive result due to optimizer noise, a "lucky" result that the community would find difficult to reproduce;
- reduce the likelihood failing to observe a positive result due to optimizer noise, an "unlucky" result that might cause a researcher to abandon a potentially fruitful line of research;
- increase the accuracy of the observed metric score (averaged over N replicas) with regard to the true metric score (averaged over an infinite number of replicas).

3.3.4 The Limitations of p-values

As with any model, p-values will sometimes be a poor estimate of the phenomenon they seek to predict. In this case, we are attempting to predict whether or not the magnitude of a difference between two metric scores is due to chance. Recently, some fields have begun moving away from p-values, citing the inability of many practitioners to understand and interpret them (Trafimow and Marks, 2015). In this section, we discuss some common misunderstandings and offer some advice on how to avoid misinterpreting p-values.

The most important points are:

- a p-value can indicate whether a difference of this magnitude is likely to be generated again by some random process (a randomized optimizer)
- a p-value does not indicate whether a difference of this magnitude is meaningful (in terms of translation quality)

So even though a large difference may more frequently correspond to smaller p-values, this is not guaranteed. In fact, small differences can be quite significant and vice versa. For example:

- Consider a deterministic optimizer and a null experimental manipulation. This means we have two exactly duplicate sets of system outputs. There will of course be a zero

difference in metric scores, but also a p-value of zero. This is because shuffling hypotheses between the systems produces no change, indicating that this difference (of zero) is likely to be reproducible.

- Consider a pathologically unstable optimizer and a null experimental manipulation. The outputs both between the replicas and between the two "systems" will be very different. For a small number of replicas, variance will be high and so shuffling is likely to change the difference in scores, giving an insignificant p-value, though this could be reduced toward zero with sufficiently many replications. This correctly mirrors that more replicas will be needed to increase the probability of repeating this experiment.

- Consider a comparison with system A being a real MT system and system B being the references. In this case, shuffling hypotheses will almost certainly degrade system B. Since the difference of deltas between the original A-B and the shuffled A-B will be large, we will observe a small, significant p-value.

These examples illustrate a few points about our p-values:

- that this significance test does not account for the user giving it too few (optimizer) samples, which is why it's important to report how many optimizer samples were used;

- that this significance test is still subject to errors in the experimental setup including failing to represent all confounding variables (for example, if the same random seed is erroneously used in multiple replicas); and

- that the test only provides information about the replicability of a delta, not whether or not the magnitude can be assigned external meaning (in terms of translation quality).

When computed properly and interpreted thoughtfully, the p-value described here can be informative tool for reasoning about whether or not an experiment might be repeatable. However, interpretation is not meaningful in isolation and factors such as the number of replicas must also be considered.

3.4 Chapter Summary

No experiment can completely control for all possible confounding variables. Nor are metric scores (even if they are statistically reliable) a substitute for thorough human analysis.

However, we believe that the impact of optimizer instability has been neglected by standard experimental methodology in MT research, where single-sample measurements are too often used to assess system differences. In this chapter, we have provided evidence that optimizer instability can have a substantial impact on results. However, we have also shown that it is possible to control for it with very few replications (Table 3.2). We therefore advocate the following methodology, which we apply throughout this thesis:

- Optimization (e.g. MERT, MIRA, PRO, etc.) and test set evaluation should be run at least three times to produce average metric scores; more replications may be necessary for experimental manipulations with more subtle effects;[6]
- Report either p-values that take multiple optimizer replicas into account or report the standard deviation on the test set over optimizer replicas s_{test}; and
- When *manually* analyzing system output, use the median system according to a trusted metric; preferably, the median should be determined based on one test set and a second test set should be manually analyzed.

[6] Source code to carry out the AR test for multiple optimizer samples on the three metrics in this chapter is available from http://github.com/jhclark/multeval.

4 Open-Domain Machine Translation using Conjunctions[1]

Coming together is a beginning; keeping together is progress; working together is success.

—Henry Ford

Conjunction junction, what's your function?

—Jack Sheldon (Schoolhouse Rock)

TRANSLATION STYLE CAN CHANGE DRAMATICALLY among texts from different domains. Rather than focusing on adapting the dense component features of the MT system to capture the style each domain, we augment the standard translation model and language model features with conjunctive features and optimize weights on a single tuning set that contains a mixture of domains. In this chapter, we show that this approach is simple, fast, and effective in improving translation quality.

4.1 Introduction

Machine translation systems are often used for information assimilation, which allows users to make sense of information written in various languages they do not speak. This use case is particularly important for translation web services, such as Bing Translator and Google Translate, which seek to make more of the web accessible to more users. A crucial challenge facing such systems is that they must translate documents from a variety of different domains, but it has been observed that the performance of statistical systems can suffer substantially when testing conditions deviate from training conditions (Bertoldi and Federico, 2009).

[1]Elements of this chapter were originally published as J. Clark, C. Dyer, and A. Lavie, "One System, Many Domains: Open-Domain Statistical Machine Translation via Feature Augmentation", *AMTA*.2012.

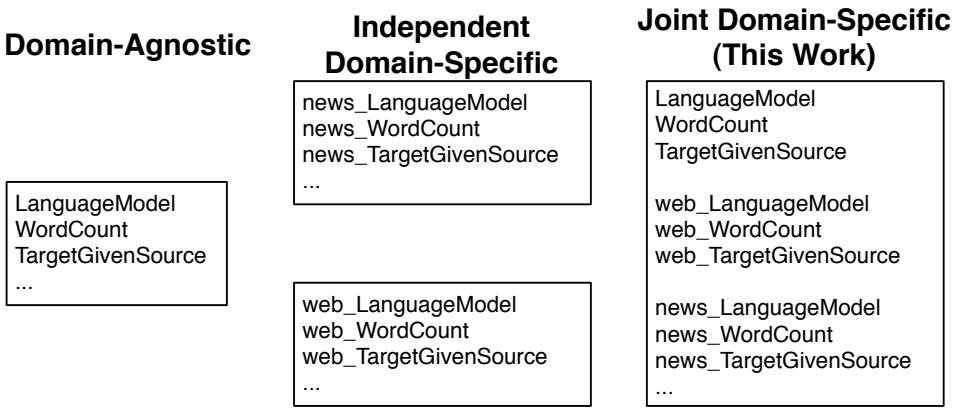


Figure 4.1: A comparison of feature sets from a baseline domain-agnostic system (left), the features sets of an approach to domain adaptation using m domain-specific systems (middle), and the feature set used in this work (right). Each feature type (e.g. `LanguageModel`) may have many conjoined instances (e.g. `news_LanguageModel`, `web_LanguageModel`), but all conjoined instances of the feature fire the same *value* as the domain agnostic feature – however, in the right two scenarios, each feature instance may have a different *weight*. Only 3 of the 7 typical baseline features (Section 4.4) are shown for brevity.

However, it is not always possible or cost-effective to collect a large body of training data for all of the desired application domains. In fact, training data tends to be collected opportunistically from any available sources rather than from curated sources that match the distribution of test data (Koehn and Schroeder, 2007). As a result, inputs from each application domain may frequently be very different from the training data. With such domain mismatches being commonplace, this chapter looks at a way of adapting the behavior of a translation system based on the *domain* of the input documents, which can be matched during both tuning and test time.

One of the key trade offs in designing a statistical model is the balance between bias and variance. In domain adaptation, we would like to *bias* each domain's model toward the distribution of that specific domain, yet we also desire models with low *variance*. Since each domain has less tuning data individually versus the aggregation of all the domains, this gives us statistically less reliable estimates for each domain's parameters.

So what options do we have for making this trade off when faced with m domains? If we desire lower variance, we may ignore the domains entirely and build a single system with $|\mathbf{H}|$ features; however, it will suffer from a bias toward some average over the domains. Another design would be to separately optimize a set of feature weights for each of the m domains, effectively creating m independent translation systems that share a com-

mon translation model and language model. This strategy expects that each component feature (the language model, lexical probabilities, etc.) would have varying contributions among the different domains. However, from an optimization perspective, this means that we would have m distinct sets of domain-specific tuning data and that the optimization procedures for each domain cannot share statistics among the domains. In this scenario, we would suffer from higher variance – each model's parameters would be more sparsely estimated while removing the possibility of relying on the more reliably estimated features of the non-adapted model.

In this work, we propose a solution that allows us to incrementally make this trade off in a soft, principled way, without entirely committing to one extreme or the other. First, we describe a simple method of feature augmentation in which we create a single system with $|\mathbf{H}|(m+1)$ features and optimize its weights using a regularization strategy that allows the optimizer to prefer the better-estimated domain-agnostic features (Section 4.2). Next, we evaluate our approach on a Czech→English system and a Arabic→English system (Section 4.4) and observe a significant improvement of up to 1.0 BLEU, controlling for both test set variation and optimizer instability (Section 4.5). Finally, we compare our work with previous approaches (Section 4.6) and conclude (Section 4.7).

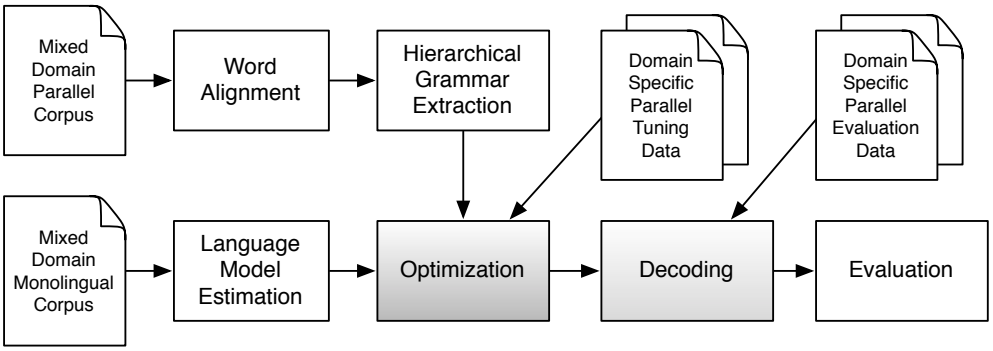


Figure 4.2: Our experimental pipeline. Shaded components are affected by this work with the optimization component being the primary focus. The optimizer and its dependent steps are run multiple times to control for optimizer instability (Chapter 3).

4.2 Feature Induction Approach

Our approach can be effectively described as two modifications to a conventional statistical system, which define a simple feature induction approach:

1. For each sentence in the tuning set and evaluation set, annotate it with its domain as an observable indicator feature such as `news=true`

2. In addition to the original features, conjoin every feature in the initial model (e.g. `WordCount=3`, `LanguageModel=0.9`) with the domain indicator feature, resulting in features such as `news_WordCount=3` and `news_LanguageModel=0.9` (Figure 4.1)

This approach corresponds to a feature induction operator:

$$\Phi^{\text{Domain}}(\mathbf{h}) \triangleq \left\{ \langle H_i \cdot h_j, i \frown j \rangle : H_i \in \mathbf{H}^{\text{Domains}}, h_j \in \mathbf{h}^{\text{Dense}} \right\} \tag{4.1}$$

where the $\frown$ operator indicates concatenation of a feature identifier with a bin identifier to form a new, unique feature identifier. Notice that the domain indicator features $\mathbf{H}^{\text{Domain}}$ are fully observed and so are constant over the entire derivation while the dense features $\mathbf{h}^{\text{Dense}}$ vary within partial derivations.

This imposes minimal requirements on the data: the parallel and monolingual training data can be completely unlabeled, while only the tuning and test sets need domain labels for each sentence. This is easy to satisfy in many scenarios. For example, if we are translating data for many different product marketing campaigns, each will have their own specific style of language targeting their consumers. It is likely we will only have a small sample of tuning and test data for each campaign, but it is easy to associate each with a particular domain label.

Unlike approaches that build m domain-specific systems and optimize weights for each of them separately (the middle scenario in Figure 4.1), our approach allows the optimizer to be informed by the data from all domains when estimating weights for the domain-agnostic features shared by all domains (the features at the top right of Figure 4.1). This framework also opens up new possibilities including having multiple granularities of domains such as `news` and `news-political` retaining the coarser features to combat data sparsity while using the finer features to fit the data more tightly. Similarly, we could encode multiple communication mediums in combination with topics as in `news-political` and `blog-political`.

Similarly, Daumé III (2007) describes a simple kernelized method for feature augmentation in which the features of each data point are labeled as either "general domain" or with a specific target domain. Like Daumé, we accomplish domain adaptation via feature augmentation. However, SMT models are more complex than those studied by Daumé. Particular characteristics that could affect performance include:

- separate training data sets for the (closed-form) estimation of the huge number of parameters in the translation model and language model

- a much smaller tuning data set for estimating the relatively few number of parameters in the final linear model that aggregates the features exposed by these component models[2]

- non-local features (e.g. the language model), which require special handling beyond the simple preprocessing step proposed by Daumé

Our approach is distinguished from previous approaches primarily in its simplicity. We do not construct domain-specific translation models nor domain-specific language models – our feature augmentation affects only the tuning and evaluation data, not the training data (Figure 4.2). Instead, we merely estimate domain-specific weights for each feature (in addition to domain-agnostic weights). However, we do this estimation jointly for all domains such that the general domain weights can be estimated using the tuning data from all domains.

4.2.1 Ease of Implementation

Translation Model Augmentation: For a batch system in which we have access to all of the sentences to be translated ahead of time, we can first extract sentence-level grammars.[3] In this case, feature augmentation of the translation model can be implemented as a preprocessing step similar to the "10 lines of Perl" described by Daumé (2007): Each sentence's grammar will have additional features appended to each grammar rule such that they contain a general domain version of each feature and a feature conjoined with the current sentence's domain. For realtime systems,[4] in which we do not have sentence-level translation models, the feature augmentation must be performed at runtime, requiring a simple decoder modification.

Language Model Augmentation: Given that sentence-level language models are generally not used,[5] domain augmentation of the language model feature cannot be implemented as a preprocessing step. Therefore, the domain of each input sentence must be passed to the decoder, and the language model must be modified such that it returns two features for each sentence: `LanguageModel` $= p$ and `domain-LanguageModel` $= p$ where p is the lan-

[2] For example, the translation model is one component model – its parameters are estimated from the parallel training data, and it typically exposes about 7 features to the aggregate model

[3] This is the default usage pattern for the cdec decoder.

[4] For example, Moses does not use sentence-level phrase tables.

[5] We are not aware of any major decoders that support sentence-level language models. This is likely due to sentence-level LMs being impractical due to their large size, which results from the large space of target translations.

guage model probability of each partial hypothesis. In the case of the cdec decoder, which was used to implement this work, the modification involved only a few lines of code.

4.2.2 Impact on Time and Space Requirements

In this section, we describe the minimal impact that this technique has on development and runtime time and space requirements. During system development, no additional time nor space is required for building additional translation models or language models since we construct only one per language pair. This also holds at runtime, which can be important if multiple deployed translation systems are competing for CPU and RAM resources on a shared server.

The main burden introduced by feature augmentation is the larger number of features itself. As discussed above in Section 1.4, this is not an issue for PRO since it scales robustly to very large feature sets. However, traditional MERT would likely fare poorly as its time complexity scales linearly with the number of features in the model.

The space requirement induced by these extra features is minimal. As noted by Daumé (2007), for an initial feature set with $|\mathbf{H}|$ features our feature space is in $\mathbb{R}^{|\mathbf{H}|}$. Then for 2 domains, this gives us a feature space $\mathbb{R}^{3|\mathbf{H}|}$ or for m domains, our feature vector will be in $\mathbb{R}^{(m+1)|\mathbf{H}|}$ where the constant of 1 represents the background domain while the $1 \ldots m$ sets of additional parameters correspond to specific domains.

4.2.3 Learning

In this chapter, we use the PRO optimizer as described in Section 1.4. We use L-BFGS to optimize our binary classifier. The procedure of sampling training pairs and then optimizing the pairwise rankings is repeated for 30 iterations. Our baseline system includes a ℓ_2 regularizer (the ℓ_2 norm is discussed further in Chapter 5).

4.3 Feature Complexity Regularization

We note that in the absence of other evidence we prefer to give weight to the background features, since they have more data to estimate them and are therefore more likely to generalize well to unseen data. We encode this preference as a regularizer (equivalently, as a prior). Let $\mathbf{w}$ be the weight vector and let the function $\textsc{Specific}(H)$ return true iff a feature H is a domain-specific feature. Then we define the regularization hyperparameter β_{C} and regularization term in our objective function as:

$$\mathcal{R}_{\text{complexity}} \triangleq \beta_{\text{C}} \sum_{i=0}^{|\mathbf{H}|} [\![\textsc{Specific}(H_i)]\!] \cdot w_i^2 \tag{4.2}$$

Since β_C is constant with regard to $\mathbf{w}$, this regularization term differs from the squared ℓ_2 norm by only a constant. Alternatively, we can view complexity regularization as partitioning the feature set into 2 groups (domain-agnostic and domain-specific) and then applying a ℓ_2 regularizer with weight β_{C} to the domain-specific group. This preserves the convexity[6] of the objective function and makes it easy to incorporate into the gradient-based updates of PRO. With this in mind, the gradient is:

$$\nabla_{\mathbf{w}} \mathcal{R}_{\text{complexity}} = \beta_{\text{C}} \sum_{i=0}^{|\mathbf{H}|} 2 \cdot [\![\textsc{Specific}(H_i)]\!] \cdot w_i \tag{4.3}$$

We apply this penalty to the domain-specific features *in addition* to the default ℓ_2 regularizer.

Complexity regularization is similar to the group lasso (a mixed-norm $\ell_{1,2}$) with two groups (domain-agnostic versus domain-specific), though we use a squared ℓ_2 penalty. The group lasso is discussed further in Section 5.2.3.

4.4 Experimental Setup

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 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 implementing these techniques.

Decoder: For decoding, we will use cdec (Dyer et al., 2010), a multi-pass decoder that supports syntactic translation models and sparse features.

[6] Each internal iteration of PRO solves a convex binary classification problem. The end-to-end optimization procedure is not convex.

	Sentences	Translations
NIST Train		
Total	5.4M	1
MT06 (tune)		
Newswire	1033	4
Web	764	4
Total	1797	4
MT08 (test)		
Newswire	813	4
Web	547	4
Total	1360	4
MT09 (test)		
Newswire	586	4
Web	727	4
Total	1313	4

Figure 4.3: Corpus statistics for Arabic→English experiments. Note that the training data was sampled in different quantities to create learning curves, simulating lower data scenarios.

Optimizer: Optimization is performed using PRO (Hopkins and May, 2011) as implemented by the cdec decoder. We run 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. Any additional regularization (as described in Section 4.3) is explicitly noted.

Baseline Features: We use the baseline features produced by Lopez' suffix array grammar extractor (Lopez, 2008a; Lopez, 2007; Lopez, 2008b), which is distributed with cdec:

- $\log P_{\text{coherent}}(e|f)$: The coherent phrase-to-phrase translation probability (Lopez, 2008b, 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

	Sentences
CzEng Train *(Sections 1-97)**	1M

	Tune (Sents) *Sec 98*	Test (Sents) *Sec 99*
Fiction	500	1000
EU Legislation	500	1000
Subtitles	500	1000
Parallel Web	500	1000
Tech Docs	500	1000
News	500	1000
Total	3000	6000

Figure 4.4: Corpus statistics for Czech→English experiments. The tuning and test sets have one reference. We use the * to indicate that sections were sampled to create a more balanced data set. Project Navajo data was omitted due to its small size. No domain-specific information from the training set was used in the construction of our models.

- $c(\texttt{glue})$ 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)

Domain-Specific Features: In addition to the baseline features, as standalone features with each of the 7 features conjoined with each of the m domains, for a total of $7(m+1)$ features overall.

Complexity Regularization: In our experiments, we used a hyperparameter value of $\beta_C = 5000$, 10 times the default global ℓ_2 regularizer constant of $\beta_2 = 500$. This value worked well and so no tuning (manual or otherwise) was performed.

Arabic Resources: We build an Arabic→English system, training on the large NIST MT 2009 constrained training corpus[7] of approximately 5 million sentence pairs with about

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

181 million English words. We tune on the NIST MT 2006 dataset and test on NIST MT 2008 and 2009.[8] Full details are shown in Figure 4.3.

Czech resources: We also construct a Czech→English system based on the CzEng 1.0 data (Bojar et al., 2012). Both sides of the data were lowercased as a preprocessing step. For our experiments, we sampled a training set of 1M sentences and training and tuning sets that are evenly balanced among the various domains contained in CzEng. Full details are shown in Figure 4.4.

Evaluation: We quantify increases in translation quality using automatic metrics including BLEU (Papineni et al., 2002). We control for test set variation and optimizer instability by measuring statistical significance according to approximate randomization (Clark et al., 2011).[9] Evaluation is performed on tokenized lowercased references.

4.5 Results and Analysis

We show the results of using feature augmentation for domain adaptation to an Arabic→English system in Figure 4.5. There, we see an overall improvement of only 0.2 - 0.3 BLEU. Interestingly, we see an improvement of up to 0.7 BLEU in the weblog domain while we see no improvement in the newswire domain. We suspect this is due to the domain of the training data being more aligned with the newswire data.

However, on the Czech→English system in Figure 4.6, we see an improvement of 1.0 BLEU on the test set consisting of the 6 domains listed in Figure 4.4. In this case, we suspect that the much larger number of domains hinders the domain agnostic system from being able to tightly fit to any of the individual domains as well as the domain augmented system.

We also did an analysis of the Czech results with and without complexity regularization. We did not observe any significant improvements with the extra conjunctive features unless complexity regularization was enabled. That is, complexity regularization was necessary to achieve improvements with this many domains. Despite some domains improving, others regressed, indicating higher variance. Complexity regularization was effective in reducing this variance, allowing for gains in aggregate.

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

[9]MultEval 0.5.1 is available at `http://github.com/jhclark/multeval`

	Baseline	Domain Augmented
MT08		
News	47.8	47.8 (±)
Web	30.5	**31.0** (+0.5)
All	40.5	**40.7** (+0.2)
MT09		
News	51.6	51.5 (±)
Web	31.6	**32.3** (+0.7)
All	41.9	**42.2** (+0.3)

Figure 4.5: Results of multi-domain experiments on the large NIST MT Arabic→English data set as measured by the BLEU metric. Differences in BLEU scores are shown in parentheses. The domain augmented system has 21 features. Both the Meteor and TER evaluation metrics agreed with BLEU in the conditions where BLEU improved. Boldfaced results indicate significance at the 0.01 level according to approximate randomization over 5 optimizer replications.

	Baseline	Domain Augmented
Test (All Domains)	46.5	**47.5** (+1.0)

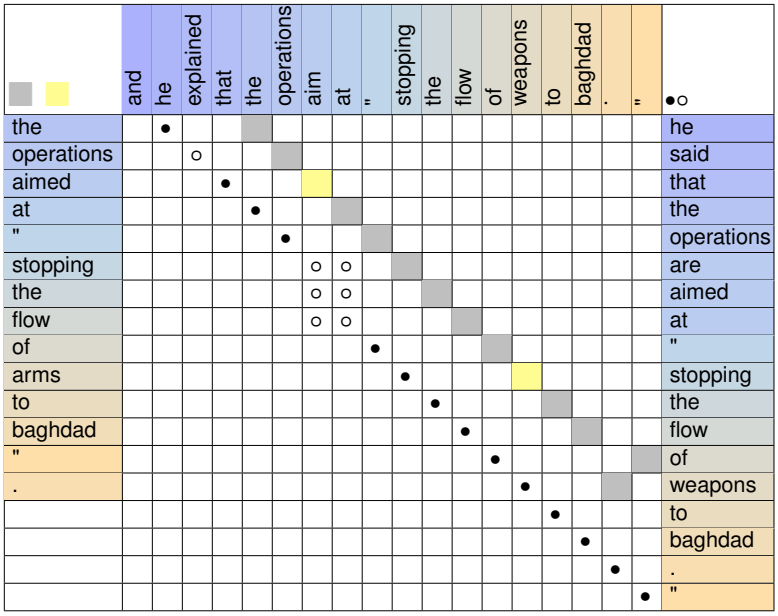
Figure 4.6: Results of feature augmentation experiments on the 1M sentence Czech→English data set as measured by the BLEU metric. Differences in BLEU scores are shown in parentheses. The domain augmented system has 49 features. Both the Meteor and TER evaluation metrics also showed improvements. Results are significant at the 0.01 level according to approximate randomization over 3 optimizer replications.

To gain a deeper understanding of the changes in translation quality resulting from our modifications, we used the Meteor X-Ray visualization tool (Denkowski and Lavie, 2010). We present two improved examples from the MT08 newswire test set in Figure 4.7. In general, improvements follow the pattern of these examples, remaining subtle yet consistent.

4.6 Related Work

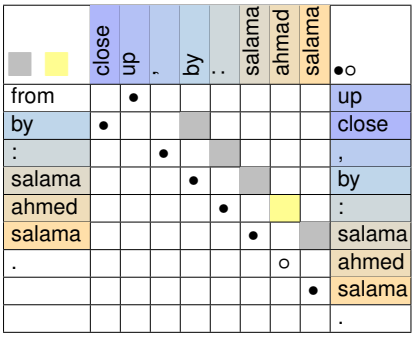
Domain adaptation in statistical machine translation has been widely investigated.

Component model adaptation: Perhaps the largest body of work focuses on adapting the translation model and language model. Koehn and Schroeder (2007) explore several techniques for domain adaptation in SMT including multiple translation models (via multiple



Segment 207

P:	0.931	vs	0.920	:	**-0.011**
R:	0.756	vs	0.881	:	**0.125**
Frag:	0.441	vs	0.338	:	**-0.102**
Score:	0.435	vs	0.587	:	**0.152**



Segment 341

P:	0.754	vs	0.871	:	**0.117**
R:	0.613	vs	0.925	:	**0.312**
Frag:	0.435	vs	0.493	:	**0.058**
Score:	0.356	vs	0.465	:	**0.108**

Figure 4.7: A Meteor X-Ray visualization of 2 improved examples, cherry picked from the MT08 newswire test set. The alignment to the reference (top border) for the baseline system (left border) is shown as shaded blocks while the alignment between the reference and the augmented system (right border) is shown using dots. Improvements are generally subtle, but consistent.

factored decoding paths), interpolated language models, and multiple language models. Xu et al. (2007) build a general domain translation system and then construct domain-specific language models and tune domain-specific feature weights to aggregate the component models. Unlike the work in this chapter, the optimization procedure for training these final feature weights cannot share statistics among domains. Foster and Kuhn (2007) train independent models on each domain and use a mixture model (both linear and log-linear) to weight the component models (both the translation model and language model) appropriately for the current context. This was later extended by Foster, Goutte, and Kuhn (2010) to examine fine-grained instance-level characteristics rather than requiring each domain to have a distinct model.

Word alignment: Civera and Juan (2007) explore an extension of the HMM alignment model that performs domain adaptation using mixture modeling.

Automatic Post-Editing: Isabelle, Goutte, and Simard (2007) use an automatic post-editor to accomplish domain adaptation, effectively "translating" from a domain-agnostic version of the target language into a domain-adapted version of the target language.

Monolingual data: Others have used monolingual data to improve in-domain performance. Ueffing, Gholamreza, and Sarkar (2007) use source monolingual data in various ways to improve target domain performance, focusing on corpus filtering techniques such that more relevant data is included in the models. Bertoldi and Federico (2009) saw large improvements by using automatic translations of monolingual in-domain data to augment the training data of their original system.

Domain identification: Closely related to domain adaptation is domain identification. Banerjee et al. (2010) focus on the problem of determining the domain of the input data so that the appropriate domain-specific translation system can be used.

EBMT: Phillips (2012) describes the Cunei framework, which natively uses instance-based features to determine which translation examples are most relevant given the current context. One major application area of this framework is domain adaptation.

Multi-Task Learning: Simianer, Wäschle, and Riezler (2011) propose a variant of minimum error rate training (MERT) that allows for multi-task learning. Like our work, multi-

task MERT allows sharing of common parameters among various task-specific optimization problems.

NLP: Other areas of NLP have also seen work on domain adaptation. For instance, Dredze et al. (2007) report their experiments in a shared task for domain adaptation of dependency parsing in which they explored adding features more likely to transfer across domains and removing features less likely to transfer.

Machine learning: Domain adaptation has also been well-studied from a more general machine learning perspective. Daumé (2006) points out that "the most basic assumption used in statistical learning theory is that training data and test data are drawn from the same underlying distribution" and goes on to formulate the MEGA (Maximum Entropy Genre Adaptation) model. Blitzer, McDonald, and Pereira (2006; Blitzer (2007) describe structural correspondence learning, which automatically discovers features that behave similarly in both a source and target domain. Ben-David et al. (2010) provide formal bounds on source and target error for various discriminative learning criteria. Huang and Yates (2012) propose a domain adaptation method of representation learning, which automatically discovers features more likely to generalize across domains. By using posterior regularization to bias the process of representation learning, they observe improvements on a part of speech tagging task and a named entity recognition task. Domain adaptation has also been framed as a semi-supervised learning problem in which unlabeled in-domain data is used to augment out-of-domain labeled data (Daumé III, Kumar, and Saha, 2010).

Perhaps most similar to this work is Daumé (2007), which proposes a method for feature augmentation in which the features of each data point augmented with a label of either "general domain" or a specific target domain.

4.7 Chapter Summary

In this chapter, we presented a very simple technique for performing domain adaptation by inducing conjunction features consisting of domain indicator features and traditional dense features. In order to mitigate the additional data sparsity of this larger feature set, we applied complexity regularization. We observed improvements of 1.0 BLEU using these techniques together. In the future, we hope to apply this technique to domains of varying granularity and combine it with more complex feature sets. Note that it is possible to have hierarchical domain structures or even overlapping domains due to the simplicity and flexibility of conjunctions. We have shown that technique is easy to interpret, simple to implement, low-cost to deploy, and effective in terms of improving translation quality.

Structure-Inducing Regularization: Inducing Large Feature Sets on Small Data[1]

5

Alone we can do so little; together we can do so much.

—Helen Keller

Great acts are made up of small deeds.

—Lao Tzu

As feature sets have become larger and sparser, our relatively small tuning data sets have been pushed to the limits of their ability to reliably estimate feature weights. Previous regularization techniques such as ℓ_1 assign zero weight to many features during optimization, allowing for better generalization by discarding features. In this chapter, we show it is possible to reduce the count of *unique weights* rather than the count of *non-zero weights*, allowing the model to retain more sparse features while still generalizing and improving translation quality on unseen data. This allows us to more tightly fit the high quality, task-specific human translations used in tuning data sets rather than resorting to larger training data whose quality often cannot be closely controlled by system builders. Using this technique, we are able to induce nearly 1 million sparse features and estimate weights on a traditional tuning set.

5.1 Introduction

In the previous chapter, we described how simple conjunctions between source features and standard MT features can be exploited to better fit the data. In this chapter, we consider conjunctions on a much larger scale. Source and target lexical indicators are conjoined with their neighbors to form features ranging from unigram indicators through very sparse rule indicators. Typically, it would not be advisable to use such a large feature set on a standard

[1] Elements of this chapter are in preparation as "Conjoined Features in Statistical Machine Translation with Pairwise ℓ_∞ Regularization" *Forthcoming*.

tuning set of 3000 sentences or less. Since most features would be observed only a few times, the optimizer is apt to learn poorly estimated weights that do not generalize well.

Others have approached this issue by directly decreasing the number of features they attempt to tune on the development set. It is not uncommon to use around 10,000 features on the development set. However, for larger feature sets, recent research has focused on scalable optimization methods that allow weights to be estimated directly from the training data (Dyer, Simianer, and Riezler, 2012; Flanigan, Dyer, and Carbonell, 2013; Yu et al., 2013). However, this makes a strong assumption about the training data: that it is drawn from the same distribution as the system's test data.[2]

However, in many real-world machine translation scenarios, this assumption does not hold. The vast majority of parallel and monolingual training data results from processes such as parliamentary proceedings and Internet forums. This data is often available free of charge to the machine translation community. Far less data may exist for a specific task due to the cost of human translation. For example, given a marketing campaign, a company may wish to use a very specific vocabulary to describe their products when disseminating their marketing materials in multiple languages. Similarly, for a conversational translation task, tightly fitting the translation style of parliamentary proceedings will yield undesirable results (e.g. for pro-drop languages, the system will be biased toward third person rather than second person).

Two phase optimization has long enabled machine translation systems to cheaply estimate features from a large, potentially out-of-domain training corpus while tightly fitting a smaller high-quality task-specific tuning corpus using more expensive optimization procedures. Rather than abandon this framework, we explore techniques to make it more robust.

In this chapter, we describe a method for tuning nearly 1 million features on a standard tuning set by adding an intuitive, well-defined, and fast regularization term to the PRO objective. This regularizer induces groups as part of learning and, as a result:

- favors fewer unique feature weights rather than fewer non-zero features
- decides group membership using an error signal directly from a task-specific objective such as BLEU rather than using a surrogate loss function or heuristic prior to learning or requiring group membership to be specified manually

This grouping effect allows us to reduce variance due to overfitting while introducing less bias than a traditional regularizer since we can more tightly fit the distribution of the tuning data.

[2]The i.i.d. (independently and identically distributed) assumption is a widespread condition for generalization according to machine learning theory.

In the remainder of this chapter, we first review ℓ_p and mixed norm regularization (Section 5.2) and introduce the OSCAR regularizer including an efficient proximal implementation and its integration with PRO (Section 5.3). Next, we present our experimental setup (Section 5.4) and examine the results (Section 5.6). Finally, we discuss related work (Section 5.7) and conclude (Section 5.8).

5.2 Background

5.2.1 ℓ_p Regularization

Regularization is already a widely used technique to avoid overfitting. Perhaps the most intuitive criterion is the ℓ_0 **"norm"**:[3]

$$\|\mathbf{w}\|_0 \triangleq \lim_{\epsilon \to 0} \sum_{i=1}^{N} |w_i|^{\epsilon} \tag{5.1}$$

$$= |\{i : w_i \neq 0\}| \tag{5.2}$$

which adds a penalty directly on the count of active features. This regularizer directly encodes the assumption that using less features in the model will lead to less overfitting. It is one of many approaches used for **feature subset selection**. Regularizers are considered an **embedded** method of selection, though other methods are possible (Guyon and Elisseeff, 2003). Unfortunately, exact optimization of this norm is NP-hard (Ge, Jiang, and Ye, 2011). While smoothed approximations do exist (Mohimani, Babaie-zadeh, and Jutten, 2007; Hyder and Mahata, 2009), the simpler regularizers are generally favored in the literature.

The ℓ_1 **regularizer** adds a penalty term for the magnitude of each feature weight:

$$\|\mathbf{w}\|_1 \triangleq \sum_{i=1}^{|\mathbf{w}|} |w_i| \tag{5.3}$$

Since ℓ_1 is non-smooth, it requires a specialized solver. Algorithms for efficiently incorporating ℓ_1 into various regression objectives have been widely studied (Park and Hastie, 2007; Andrew and Gao, 2007) including the well-known Lasso algorithm for least squares regression (Tibshirani, 1996). Despite its differences from the ℓ_0 norm, in practice, ℓ_1 tends to produce solutions with **parameter sparsity**, those having fewer non-zero weights. In

[3] Strictly speaking, ℓ_0 is not a norm since it fails 2 of the 3 conditions for a norm. ℓ_0 is not subadditive (i.e. the triangle inequality $f(x+y) \leq f(x) + f(y)$) nor is it absolutely homogeneous ($f(cx) = |c|f(x)$). In fact, the only property of a norm that ℓ_0 satisfies is positivity (it assigns non-negative values to non-negative vectors and zero to zero vectors).

fact, in many cases, the solution to the ℓ_1 optimization problem has equivalent sparseness to the ℓ_0 optimization problem (Donoho, 2004). From a Bayesian perspective, a ℓ_1 penalty is equivalent to placing a Laplace prior on the weights.

The squared ℓ_2 regularizer adds a penalty term for the magnitude of each feature weight:

$$\|\mathbf{w}\|_2^2 \triangleq \sum_{i=1}^{|\mathbf{w}|} w_i^2 \tag{5.4}$$

In the statistics literature, it is often called **ridge regression** when the w_i are coefficients in a linear regression model. From a Bayesian perspective, it is equivalent to placing a zero-mean normal distribution as a prior on the weights. The ℓ_2 norm is perhaps the most attractive from the perspective of simplicity in that it is strictly convex and smooth, which allows it to be readily incorporated into any objective function without special solvers. However, the ℓ_2 norm is known to produce less parameter sparsity, leaving more active features overall.

The ℓ_∞ **regularizer** (also known as the maximum norm or uniform norm) adds a penalty term for the magnitude of the largest active feature weight:

$$\|\mathbf{w}\|_\infty \triangleq \lim_{x \to \infty} \|\mathbf{w}\|_\infty \tag{5.5}$$

$$= \max_i |w_i| \tag{5.6}$$

This regularizer is rarely used by itself in practice. In theory, this pressure on the feature with the greatest magnitude will encourage equality of the largest parameters given that they are equally helpful in minimizing the overall objective function. However, most batch learning strategies will naturally produce such a solution. Furthermore, there are only marginal benefits to reducing the magnitudes of the largest model parameters; this regularizer is not effective in producing solutions with parameter sparsity.

We graphically summarize these regularization strategies in Figure 5.1, showing the constraint regions for a model with two parameters. As both parameter weights approach zero (the axes), the regularizers behave similarly by being permissive. However, as the parameter magnitudes increase toward the corners of the plots, the regularizers become increasingly permissive from left to right.

Any of these regularizers can be written in one of two equivalent forms (Kloft et al., 2009). Given a convex loss function $\mathcal{L}$ and a regularization term $\mathcal{R}$, the two forms are

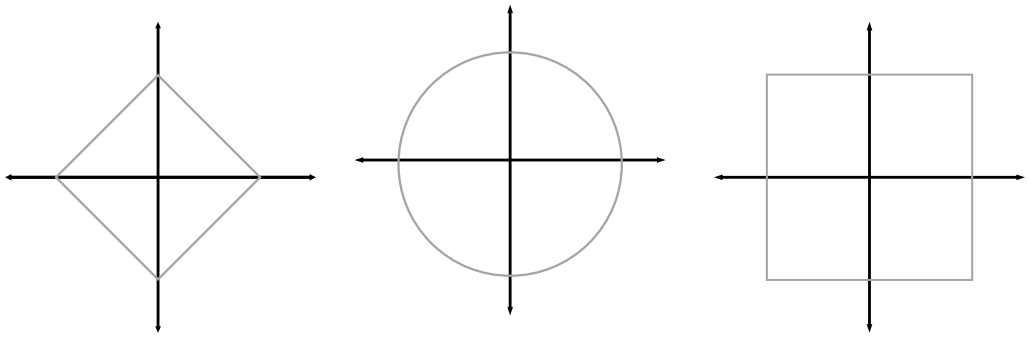


Figure 5.1: Visualizations of constraint regions for a model with two parameters for three standard ℓ_p regularizers: ℓ_1 (left), ℓ_2 (center), and ℓ_∞ (right). ℓ_0 is omitted since it does not directly constrain the model weights but rather the count of non-zero model parameters.

penalty form (Tikhonov Regularization):

$$\hat{\mathbf{w}} \triangleq \underset{\mathbf{w}}{\operatorname{argmin}} \mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{w}) + \mathcal{R}(\mathbf{w}) \tag{5.7}$$

and **constraint form** (Ivanov Regularization):

$$\hat{\mathbf{w}} \triangleq \underset{\mathbf{w}}{\operatorname{argmin}} \mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{w}) \quad \text{s.t.} \quad \mathcal{R}(\mathbf{w}) \leq \tau \tag{5.8}$$

where τ is a constraint on the maximum value of the regularization term. In this work, we will typically write the penalty form, but may visualize and discuss the effects of regularizers in constraint form for the sake of exposition.

5.2.2 Elastic Net Regularization: Combining ℓ_1 and ℓ_2

The **elastic net** regularizer combines ℓ_1 and ℓ_2 (Zou and Hastie, 2005), preserving some of the most important qualities of both. Formally, it is defined as:

$$\mathcal{R}_{\text{Elastic}} \triangleq \|\mathbf{w}\|_1 + \|\mathbf{w}\|_2^2 \tag{5.9}$$

In particular, it is strictly convex like ℓ_2 (ℓ_1 does not necessarily have a unique solution) while still retaining the parameter sparsity property of ℓ_1. These properties can be observed in the Elastic Net's constraint regions in Figure 5.2. Yet unlike either of the methods in isolation, it exhibits a soft grouping property – parameters tend to fall to zero together. In the extreme case when features are *perfectly* correlated, the features within each perfectly

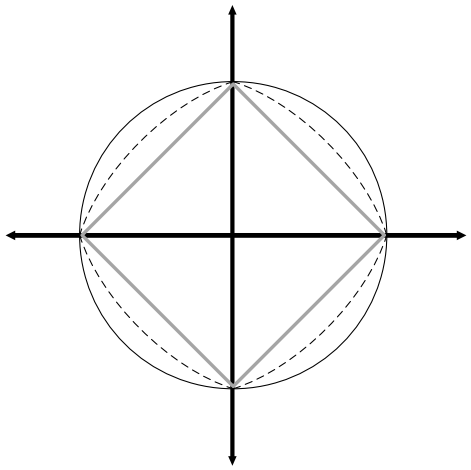


Figure 5.2: The constraint region of a two dimensional model regularized by the Elastic Net with varying choices of regularization constants. The middle dotted line represents a standard choice that takes advantage of the properties of the Elastic Net. The inner gray line recovers ℓ_1 while the outermost circle recovers ℓ_2.

correlated group will receive exactly the same weight. However, in general, within each automatically discovered feature group, the parameters tend to be highly correlated, though not necessarily exactly equal. We will return to this issue in Section 5.3.

5.2.3 The Group Lasso: A mixed norm

The group lasso (Yuan and Lin, 2006) and the sparse group lasso (Friedman, Hastie, and Tibshirani, 2010) allow users to embed prior knowledge about group structure into the optimization problem.

Given a partitioning of features into groups $g \in \mathcal{G}$ where λ_g denotes the regularization constant for g and $\mathbf{w}_g$ denotes the subset of weights corresponding to group g, the group lasso regularizer is defined as:

$$\mathcal{R}_{\text{group}}(\mathbf{w}, \mathcal{G}) \triangleq \sum_{g \in \mathcal{G}} \lambda_g \|\mathbf{w}_g\|_2 \tag{5.10}$$

Note that the iteration over the groups actually makes this a mixed $\ell_{2,1}$ norm. If each group contains exactly one feature s.t. $\mathcal{G}_i = \{w_i\}$, then the group lasso simplifies to ℓ_1. On the other hand, if a single group contains all features $\mathcal{G}_1 = \{1, \ldots, N\}$, then we recover a ℓ_2 regularizer (Martins et al., 2011).

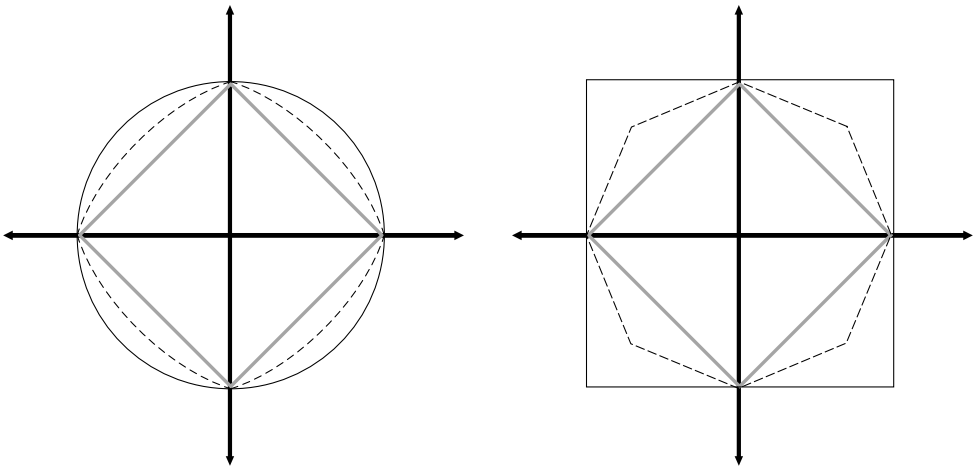


Figure 5.3: The constraint region of a two dimensional model comparing the Elastic Net (left) and OSCAR (right). **Left:** The inner solid gray line is the constraint region for Lasso (ℓ_1). The dashed shape represents a typical configuration of the elastic net. The outer black circle recovers ℓ_2. The sharp corners of the ℓ_1 constraint region incident on the axes represent exact zero values for one or both model parameters. **Right:** For OSCAR, the additional 4 corners shown in the dotted line represent exact equality in magnitude of the two model parameters, which is encouraged by OSCAR. The outer black line recovers a ℓ_∞ regularizer.

The group lasso has been extended to support non-overlapping groups, including tree- and graph-structured groups (Jacob, Obozinski, and Vert, 2009). It has also found some popularity in structured prediction (Martins et al., 2011).

Unlike the soft groups induced by the elastic net, the group lasso requires these groups to be pre-specified. Further, a hyperparameter λ_g must be learned or specified for each group. If such knowledge is straightforward and low-cost to encode, the group lasso is likely to result in a better model. Otherwise, a poor choice of groups may result in either no improvement if the hyperparameters are learned or — if the hyperparameters are also poorly specified — a degradation in model quality.

5.3 OSCAR: Inducing and regularizing groups exactly

OSCAR (Octagonal Selection and Clustering Algorithm for Regression) is a combination of a ℓ_2 regularizer, which seeks to control the overall number of active features, and a pairwise ℓ_∞ regularizer, which seeks to control the overall degrees of freedom (unique feature weights) by grouping features with similar weights.

To encourage this, the combination of a ℓ_1 and pairwise ℓ_∞ term called OSCAR has been proposed (Bondell and Reich, 2008). It is defined as:

$$\mathcal{R}_{\text{OSCAR}} = \beta_1 \sum_{i=1}^{|\mathbf{w}|} |w_i| + \beta_\infty \sum_{i=1}^{|\mathbf{w}|} \sum_{j=1}^{i-1} \max\left(|w_i|, |w_j|\right) \tag{5.11}$$

These constraint regions are visualized in Figure 5.3. The octagonal shape of the constraint region gives OSCAR its name. Compared to the soft grouping property of the Elastic Net, OSCAR promotes *exact* grouping of feature weights, which are visualized as the pointed corners furthest from the axes.

Note that in two dimensions, the pairwise formulation has no effect beyond a standard ℓ_∞ regularizer. However, with more parameters, the move to a pairwise formulation allows more than one group to be formed. That is, without the pairwise term, the exact grouping property would only apply to a single group that must include the feature with the greatest magnitude.

This penalty form may also be simplified to depend on a single parameter at a time, if we first sort $\mathbf{w}$ by the magnitude of each parameter weight such that $w_1 \leq w_2 \leq \ldots \leq w_N$:

$$\mathcal{R}_{\text{OSCAR}} = \beta_1 \sum_{i=1}^{|\mathbf{w}|} \left(\beta_\infty (i-1) + 1\right) |w_i| \tag{5.12}$$

This form highlights that the contribution pairwise ℓ_∞ penalty increases linearly as we iterate through the sorted weights. We will elaborate on this and use it to our advantage in Section 5.3.2.

While this objective is convex, it is non-smooth and so requires a specialized solver. Unfortunately, the originally proposed solver is cast as a huge quadratic program, which the authors admit may overwhelm standard solvers. A more efficient max-flow solver was proposed in Mairal et al. (2010), but the resulting complexity remains $O(n^5)$ for n features. Both solvers are prohibitively expensive for the nearly 1 million features we are seeking to optimize.

5.3.1 Background: Proximal Gradient Methods

Proximal gradient methods are useful for efficiently solving optimization problems that can be posed as the summation of a smooth convex component $f(x)$ and a non-smooth convex component $r(x)$ (Combettes and Pesquet, 2010; Parikh and Boyd, 2013):

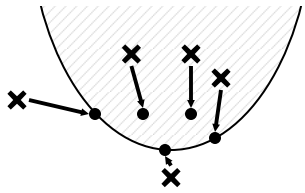


Figure 5.4: Visualization of a prox operator moving initial points v (x's) to proximal points (filled circles) that are closer to the optimum of $r(x)$ and still within its convex set $r(x)$ (shaded area).

$$\min f(\mathbf{w}) + r(\mathbf{w}) \tag{5.13}$$

In this work, $r(\mathbf{w})$ will correspond to the OSCAR regularization term. In a proximal gradient method, each optimization iteration first takes a step toward minimizing $f(\mathbf{w})$ and then takes another step that incorporates the contributions of $r(\mathbf{w})$. Formally, the proximal operator solves the optimization problem of moving a weight vector v (that has been updated based on $f(\mathbf{w})$) toward the optimum of $r(\mathbf{w})$:

$$\mathbf{prox}_r(\mathbf{v}) = \underset{\mathbf{w}}{\operatorname{argmin}} \left(r(\mathbf{w}) + \|\mathbf{w} - \mathbf{v}\|_2 \right) \tag{5.14}$$

Because this effectively projects the initial weight vector v toward the minimum of the function $r(\mathbf{w})$ while still compromising between remaining near to v, $\mathbf{prox}_r(\mathbf{v})$ is sometimes called a **proximal point** with regard to $r(\mathbf{w})$ (Parikh and Boyd, 2013). Figure 5.4 visualizes the effects of a proximal operator.

The proximal operator may also be interpreted as an step in the direction of the gradient of $r(x)$ for some small constant c:

$$\mathbf{prox}_r(\mathbf{v}) = \mathbf{v} - c\nabla r(\mathbf{w}) \tag{5.15}$$

Proximal methods (including projected gradient methods) can also be used to minimize any of the ℓ_p penalties mentioned so far. Duchi and Singer (2009) describe the Forward Backward Splitting (FOBOS) framework along with concrete projection algorithms for ℓ_1, ℓ_2, ℓ_∞ and mixed-norm $\ell_{1,2}$. We use this formulation for both ℓ_1 and the elastic net in our experiments.

5.3.2 FastOSCAR: A proximal solver for OSCAR

FastOSCAR is a proximal method that defines an $O(N)$ proximal operator for OSCAR (Zhong and Kwok, 2011; Zhong and Kwok, 2012),[4] making it very attractive for very large feature sets.[5] Despite its efficiency, FastOSCAR still provides an exact solution.

Within each instance of the proximal step, FastOSCAR begins with each feature weight having its own cluster where the clusters are sorted by decreasing magnitudes of the current pre-proximal feature weights. From this point on, each proximal feature weight includes a ℓ_1 and pairwise ℓ_∞ penalty and each cluster is defined to have a common weight equal to the average of its component proximal feature weights. Importantly, if the value drops under zero, it is clipped as with ℓ_1 solvers. FastOSCAR then hypothesizes merging the current cluster with the next cluster until the common weight of the current cluster (including newly added regularization penalties) is no longer greater than the weight of the next cluster (this works due to the pre-sorted nature of the feature weights). One of the keys to FastOSCAR's efficiency is that the merge operation can be implemented as fixed number of addition operations (updating number of members in the group and a sum of weights).

Let $\mathbf{q}$ be the current iteration's weight vector $\mathbf{w}$ sorted by weight magnitude. Then the common weight value for the group spanning from i to j in $\mathbf{q}$ is given by:

$$\textsc{CommonValue}(\mathcal{G}_{i:j}, \mathbf{q}) = \max\left(\frac{\sum_{k\in\mathcal{G}_{i:j}} q_k - \beta_1 - \beta_\infty(N-k)}{j-i+1}, 0\right) \tag{5.16}$$

We provide a formal algorithm for the FastOSCAR proximal operator in Algorithm 2.

5.4 Experimental Setup

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. 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.

[4] FastOSCAR is also re-derived in a more general form and called FISTA-GPO by Zeng and Figueiredo (2014)

[5] We have released a generic implementation of FastOSCAR at `http://github.com/jhclark/prunejuice/` as part of this work. It has been integrated into the cdec MT decoder at `http://github.com/jhclark/cdec/`

Algorithm 2 FASTOSCAR Proximal Operator

Input: Pre-proximal (unregularized) weight vector $\mathbf{w}$
Output: Post-proximal (regularized) weight vector $\mathbf{w}'$
Let $N = |\mathbf{w}|$
Compute $\mathbf{a}$ as $a_i = |w_i|$
$\triangleright$ We retain a mapping between indices in $\mathbf{w}$ and $\mathbf{q}, \mathcal{G}$
$\mathbf{q} \leftarrow \text{SORTDECREASING}(\mathbf{a})$
From the sorted indices $\mathbf{q}$, create groups $\mathcal{G}_{1:1} \dots \mathcal{G}_{N:N}$
Initialize a stack $\mathcal{S} = [\mathcal{G}_{1:1}]$
for $i \in [2, N]$ **do**
  $\mathcal{G}' \leftarrow \mathcal{G}_{i:i}$
  while $\overline{\overline{\mathcal{S}}} > 0$ **and** $\text{COMMONVALUE}(\mathcal{G}', \mathbf{q}) \geq \text{COMMONVALUE}(\mathcal{G}_{\text{top}}, \mathbf{q})$ **do**
    $\mathcal{G}' \leftarrow \mathcal{G}' \cup \mathcal{G}_{\text{top}}$
    Pop $\mathcal{G}_{\text{top}}$ from $\mathcal{S}$
  end while
  Push $\mathcal{G}$ onto $\mathcal{S}$
end for
$z_i \leftarrow \text{COMMONVALUE}(\mathcal{G}_k)$ where $w_i \in \mathcal{G}_k$
$w'_i \leftarrow \text{sign}(w_i) \cdot z_i$
return $\mathbf{w}'$

Decoder: For decoding, we will use cdec (Dyer et al., 2010), a multi-pass decoder that supports syntactic translation models and sparse features.

Optimizer: Optimization is performed using PRO (Hopkins and May, 2011) as implemented by the cdec decoder. We run PRO for 30 iterations as suggested by Hopkins and May (2011). Our baseinle PRO optimizer internally uses an ADAGRAD optimizer with the default ℓ_2 regularization implemented in cdec. Any additional regularization is explicitly noted. The hyperparameters for each regularizer are tuned using grid search on data kept separate from the dev and test sets.

Baseline Features: We use the baseline features produced by Lopez' suffix array grammar extractor (Lopez, 2008a), which is distributed with cdec. The addition of sparse features in other experimental conditions is noted explicitly. These baseline features are:

- $\log P_{\text{coherent}}(e|f)$: The coherent phrase-to-phrase translation probability (Lopez, 2008b).
- $\log P_{\text{lex}}(e|f)$, $\log P_{\text{lex}}(f|e)$: The lexical alignment probabilities within each translation rule, as computed by a maximum likelihood estimation over the Viterbi alignments

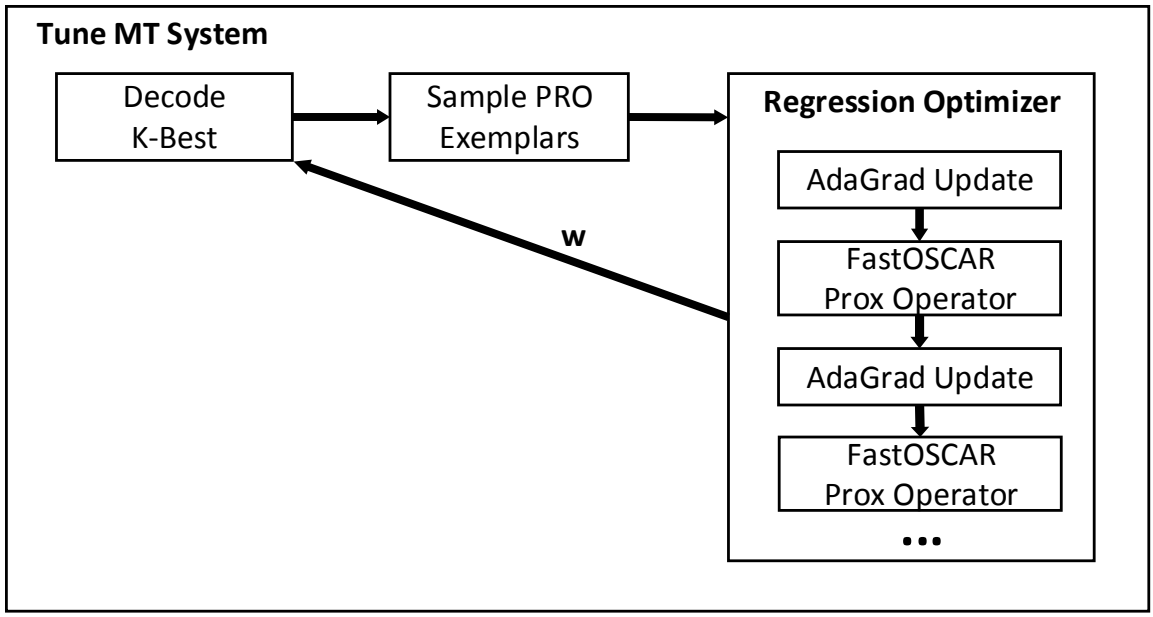


Figure 5.5: Our modified PRO optimization procedure including the FastOSCAR proximal operator.

- $\log P_{\text{LM}}(e)$: 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)

	Zh→En	**Ar→En**	**Cz→En**
Train	303K	5.4M	1M
WeightTune	1664	1797	3000
HyperTune	1085	1056	2000
Test	1357	1313	2000

Table 5.1: Corpus statistics: number of parallel sentences.

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[6] of approximately 300,000 sentence pairs with about 9.4 million English words. We tune weights on the NIST MT 2006 dataset, tune hyperparameters on NIST MT05, and test on NIST MT 2008. See Table 5.1.

Arabic Resources: We build an Arabic→English system, training on the large NIST MT 2009 constrained training corpus[7] of approximately 5 million sentence pairs with about 181 million English words. We tune weights on the NIST MT 2006 dataset, tune hyperparameters on NIST MT 2005, and test on NIST MT 2008.[8] See Table 5.1.

Czech resources: We also construct a Czech→English system based on the CzEng 1.0 data (Bojar et al., 2012). First, we lowercased and performed sentence-level deduplication of the data.[9] Then, we uniformly sampled a training set of 1M sentences (sections 1 – 97) along with a weight-tuning set (section 98), hyperparameter-tuning (section 99), and test set (section 99) from the paraweb domain contained of CzEng.[10] Sentences having length less than 5 were discarded due to their noisy nature. See Table 5.1.

Evaluation: We quantify increases in translation quality using case-insensitive BLEU (Papineni et al., 2002). We control for test set variation and optimizer instability by averaging over multiple optimizer replicas (Chapter 3).[11]

5.4.1 Sparse Features: Simple Lexical Conjunctions

To test the utility of OSCAR in machine translation, we add several categories of sparse feature templates already in use in machine translation. For the sake of exposition, we cast these feature templates as instances of conjunctions.

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

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

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

[9] CzEng is distributed deduplicated at the document level, leading to very high sentence-level overlap.

[10] The section splits recommended by Bojar et al. (2012).

[11] MultEval 0.5.1: github.com/jhclark/multeval

We begin with lexical word identities on the source and target. We then add conjunctions of these features using two operations:

- **Conjoin across:** This produces lexical translation indicators (e.g. does "gato" translate as "cat"?) and may apply inside any rule.
- **Conjoin neighbor:** Conjoin with the lexical indicator on the left or right. This produces n-gram features.

On the source, we limit this to rule boundaries while we allow it to cross rule boundaries on the target side (like a standard LM). In the extreme, these two types of conjunctions also recover the widely used rule identity indicator features that use an indicator for each unique SCFG translation rule.

This corresponds to the feature induction operators:

$$\Phi^{\text{Across}}(\mathbf{h}) \triangleq \left\{ \langle h_i \cdot h_j, i \frown j \rangle : h_i \in \mathbf{h}^{\text{SrcUnigrams}}, h_j \in \mathbf{h}^{\text{TgtUnigrams}} \right\} \tag{5.17}$$

$$\Phi^{\text{Target}}(\mathbf{h}) \triangleq \left\{ \langle h_i \cdot h_j, i \frown j \rangle : h_i \in \mathbf{h}^{\text{TgtUnigrams}}, h_j \in \mathbf{h}^{\text{TgtUnigrams}} \right\} \tag{5.18}$$

where the $\frown$ operator indicates concatenation of a feature identifier with a bin identifier to form a new, unique feature identifier. The union of these induced features along with the original dense features are included in our models.

We view this process as a series of steps:

- Simple initial features are specified by the user
- Conjunctions are applied to generate a much larger, more specific feature space (intuitively, a splitting operation)
- The regularizer induces feature groups, discarding overly specific distinctions (intuitively, a merge operation)

5.5 Hyperparameter Tuning

For the systems with ℓ_1 regularization, we tuned the hyperparameter β_1 by sweeping through values in the set $\{1e^{-1}, 1e^{-2}, \ldots, 1e^{-7}\}$. We ran PRO to completion on the tuning set and selected the value that maximized BLEU on the hyperparameter sweep data set (see above). For our reported results, we sweep each language individually.

For the systems with OSCAR regularization, we performed some preliminary investigation on the tuning set of the Arabic→English data set. With the goal of controlling the number of unique weights rather than the number of active features, we noticed that $\beta_1 = 1e^{-6}$

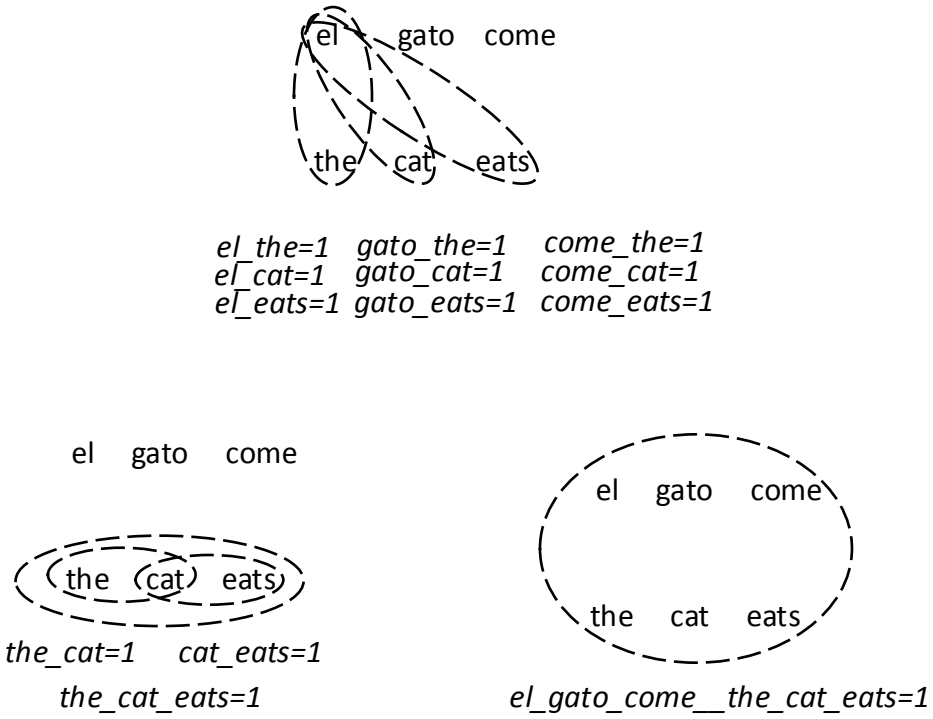


Figure 5.6: Diagrams of indicator features. Conjoining across the rule on yields lexical translation features (right). Conjoining with neighboring words yields n-gram features (center). Combining these two operators eventually leads to full-rule features (right).

was very stable across multiple feature sets. We use this value in all of our experiments with OSCAR. On the other hand, β_∞ was far more sensitive, particularly in the region $\{1e^{-8}, 2e^{-8}, \ldots, 9e^{-8}\}$. We performed a parameter sweep on β_∞ for all of our experiments with OSCAR. However, we also observed that $\beta_\infty = 3e^{-8}$ gave good performance across data sets. The relative unimportance of β_1 in the context of OSCAR is intuitive given our new focus on reducing the number of unique weights rather than reducing the number of active features.

5.6 Results

Table 5.2 shows the experimental results for sparse features with ℓ_1 and OSCAR. In our first comparison, we add the sparse features described in Section 5.4.1 along with a ℓ_1 regularizer (row 2). In the Arabic-English system, while the feature space contains nearly

Condition	#NonZero	#Unique	Tune BLEU	Test BLEU
Zh→En				
Baseline (ℓ_2)	8	8	29.4	23.5
SparseFeats (ℓ_1 + ℓ_2)	75,952	55,746	36.0	22.8* (-0.7)
SparseFeats (OSCAR)	677,728	922	34.4	**24.1*** (+0.6)
Ar→En				
Baseline (ℓ_2)	8	8	42.2	47.7
SparseFeats (ℓ_1 + ℓ_2)	79,961	56,657	50.5	48.3* (+0.6)
SparseFeats (OSCAR)	824,563	608	48.6	**49.4*** (+1.7)
Cz→En				
Baseline (ℓ_2)	8	8	33.3	38.5
SparseFeats (ℓ_1 + ℓ_2)	115,210	77,732	36.9	38.3* (-0.2)
SparseFeats (OSCAR)	826,440	2,273	35.8	**38.7*** (+0.2)

Table 5.2: Translation quality for systems using a standard feature set and PRO with ℓ_2 regularization (row1) compared to systems using sparse conjoined features (rows 2 – 3). Differences in BLEU are shown in parentheses. Row 2 shows the effect of ℓ_1 regularization whereas row 3 includes OSCAR. All scores are averaged over 3 end-to-end optimizer replications. * denotes significantly different than the baseline (row 1) with $p(\text{CHANCE}) < 0.01$ under Clark et al. (2011).

1 million features, the ℓ_1 regularizer selects less than 100k as active features, with almost all of those having unique feature weights. However, after applying the OSCAR regularizer, nearly all the features become active while a mere 608 receive unique feature weights (i.e. were placed into different groups). The benefits of this grouping effect can be seen in the model's ability to generalize. Despite the OSCAR model preforming worse than the ℓ_1 model on the tuning set, it still outperforms ℓ_1 on the test set.

5.6.1 Analysis of Chinese→English ℓ_1 Result

The significant underperformance of ℓ_1 for the Chinese→English experiments merits some special attention. Intuitively, one might expect that as we increase β_1, the penalty will eventually push the newly added sparse features toward weights of zero, allowing us to recover levels of quality comparable with the baseline system in the worse case. In practice, we observed that on the tuning set, we do recover a BLEU score comparable to the baseline system; however, the optimizer achieves this via a somewhat pathological set of feature weights, including assigning negative weights to $\log P_{\text{rule}}(t|s)$, $\log P_{\text{lex}}(t|s)$, and $\log P_{\text{lex}}(s|t)$ (positive weights should be expected as these are generally strong features). We made several attempts to improve the performance of the ℓ_1 system including initializing PRO using the weights from other strong systems, attempting multiple restarts within PRO, doing

additional finer-grained hyperparameter sweeps on β_1, and even tuning β_1 directly on the test set. None of these led to any meaningful improvements and so we conclude that PRO with standard ℓ_1 simply did not manage to prevent overfitting in this case.

For Chinese→English, we were able to recover the baseline performance using a Group Lasso regularizer. We divided the features into sparse and dense groups and allowed different regularization penalties for each group. We fixed the regularization penalties for the dense features to those of the baseline system and conducted a hyperparameter sweep for the regularization penalties of the sparse features. However, this resulted in a very strong ℓ_1 penalty, which set *all* of the sparse features to zero. While we did exactly recover the baseline performance (as opposed to the drop of -0.7 BLEU observed for traditional ℓ_1), this very harsh hyperparameter setting is a disappointing result for ℓ_1.

5.6.2 Analysis of Arabic→English OSCAR Result

We also analyzed some of the translations from the Arabic→English test set to better understand the effects of OSCAR. In Figure 5.7, we show four hand-picked examples of improvements. In the first example, we recover a missing determiner. The derivation contains a bigram feature for `in-a`, which receives a small positive weight, shared with over 10,000 other features, including `in-an` and `in-the`. We also noticed that the n-gram feature `in-to` shares the same *magnitude*, but with a *negative* sign (perhaps because the system should prefer "into"). In the second example, we recover the word "might". The derivation for this hypothesis contains a feature for the Arabic قد (*qd*) translating as "might". Again, this feature receives a small positive weight, shared with many other features.

5.7 Related Work

Feature grouping has also been explored in NLP. Suzuki and Nagata (2013) propose an efficient feature grouping algorithm for named entity recognition and dependency parsing tasks with up to 15 million features. Unlike OSCAR, their algorithm requires both the number and values of the allowable weights to be pre-specified.

More recently, more advanced regularization techniques have been explored in MT. Vaswani, Huang, and Chiang (2012) used a smoothed ℓ_0 "norm" in the IBM word alignment models and observed improvements in both alignment and translation quality. Their implementation uses a projected gradient algorithm (a subclass of proximal gradient methods). Liu et al. (2013b) perform small-scale experiments on IWSLT data with an OSCAR-like objective. However, their algorithm requires repeatedly decoding the entire training data

REF: the jordanian king heads to canada **in a** business trip
BASELINE: jordanian king to visit canada *in* working visit.
OSCAR: jordanian king to visit canada **in a** working visit.

REF: sarkozy **might** visit libya "soon" (presidency).
BASELINE: sarkozy *was to* visit libya "soon" (presidency).
OSCAR: sarkozy **might** visit libya "soon" (presidency).

REF: however difficult the fight is in iraq, **we** must **win** it. "
BASELINE: whatever the difficult struggle in iraq *should be on our victory*.
OSCAR: whatever the difficult struggle in iraq **we win**.

REF: **the pakistan people's party** decided to reserve its decision.
BASELINE: *ppp* decided to retain the resolution.
OSCAR: **pakistan people's party** decided to retain the resolution.

Figure 5.7: Hand-picked examples in which OSCAR improved over the baseline. Often, these improvements are the result of features having small positive weights.

and their feature grouping criterion is not driven by the top-level objective function (i.e. PRO/BLEU).

Green et al. (2013) has also explored large feature sets using PRO. Similar to our contrastive baselines, Green applied ℓ_1 regularization with FOBOS and analyzed feature sets up to 500k features. Unlike the work in this chapter, they excluded features whose count in the training data fell below a certain threshold. Green found that their algorithm "overfits badly to MT06" and found that it was necessary to combine several tuning sets in order to avoid this overfitting. They observed that using rule ID features on an Arabic→English task yielded about 0.5 BLEU improvement on average over a PRO baseline with dense features, which is consistent with our experiments. They also observed gains in a phrase-based scenario when adding sparse lexicalized reordering features.

Another recent line of work has involved optimizing feature weights directly on the training data. Yu et al. (2013) tuned 20M sparse features directly on the training data, noting improvements over PRO tuned on small tuning data. They also observed that PRO without strong regularization began to degrade with tens of thousands of features. Auli, Galley, and Gao (2014) observed improvements when turning a phrase-based system with a discriminative reordering model having nearly 3M features using an expected BLEU (minimum risk) objective. This required n-best lists to be produced for the entire training data.

5.8 Chapter Summary

In this chapter, we have described the integration of the FastOSCAR regularizer into PRO. We observed improvements of up to 1.7 BLEU while estimating weights for nearly 1 million features, given as few as 1300 parallel sentences. Such large feature sets have previously only been successfully used in MT when doing learning on the entire training data. We have also argued that due to the frequent scarcity of high-quality truly task-specific data, it can be advantageous to leverage a small amount of data. We conclude that OSCAR's prior toward fewer degrees of freedom is an effective tool for tuning a large number of features on a small amount of data for a MT system.

Discretization: Inducing Non-Linear Transforms of Initially-Continuous Features[1]

6

The straight line leads to the downfall of humanity.
—Friedensreich Hundertwasser

LINEAR MODELS, which support efficient learning and inference, are the workhorses of statistical machine translation; however, linear decision rules are less attractive from a modeling perspective. In this chapter, we introduce a technique for learning arbitrary, rule-local, non-linear feature transforms that improve model expressivity, but do not sacrifice the efficient inference and learning associated with linear models. To demonstrate the value of our technique, we discard the customary log transform of lexical probabilities and drop the phrasal translation probability in favor of raw counts. We observe that our algorithm learns a variation of a log transform that leads to better translation quality compared to the explicit log transform. We conclude that non-linear responses play an important role in SMT, an observation that we hope will inform the efforts of feature engineers.

6.1 Introduction

Linear models using log-transformed probabilities as features have emerged as the dominant model in MT systems. This practice can be traced back to the IBM noisy channel models (Brown et al., 1993), which decompose decoding into the product of a translation model (TM) and a language model (LM), motivated by Bayes' Rule. When Och and Ney (2002) introduced a log-linear model for translation (a linear sum of log-space features), they noted that the noisy channel model was a special case of their model using log probabilities. This same formulation persisted even after the introduction of MERT (Och, 2003), which optimizes a linear model; again, using two log probability features (TM and LM) with

[1]Elements of this chapter were originally published as J. Clark, C. Dyer, and A. Lavie, "Locally Non-Linear Learning for Statistical Machine Translation via Discretization and Structured Regularization", *TACL*. 2014.

equal weight recovered the noisy channel model. Yet systems now use many more features, some of which are not even probabilities. Nor do we now believe that equal weights between the TM and LM provides optimal translation quality; the probabilities in the TM do not obey the chain rule nor Bayes' rule, nullifying several theoretical mathematical justifications for multiplying probabilities. The story of multiplying probabilities may just amount to heavily penalizing small values.

The community has abandoned the original motivations for a linear interpolation of log-transformed features. Is there empirical evidence that we should continue using this particular transformation? Do we have any reason to believe it is better than other non-linear transformations? To answer these questions, we will explore the issue of non-linearity in models for MT. In the process, we will discuss the impact of linearity on feature engineering and develop a general mechanism for learning a class of non-linear transformations of real-valued features.

Applying a non-linear transformation such as log to features is one way of achieving a non-linear response function, even if those features are aggregated in a linear model. Alternatively, we could achieve a non-linear response using a natively non-linear model such as a SVM (Wang, Shawe-Taylor, and Szedmak, 2007) or RankBoost (Sokolov, Wisniewski, and Yvon, 2012). However, MT is a structured prediction problem, in which a full hypothesis is composed of partial hypotheses. MT decoders take advantage of the fact that the model score decomposes as a linear sum over both local features and partial hypotheses to efficiently perform inference in these structured spaces (Section 1.3) – currently, there are no scalable solutions to integrating the hypothesis-level non-linear feature transforms typically associated with kernel methods while still maintaining polynomial time search. Another alternative is incorporating a recurrent neural network (Schwenk, 2012; Auli et al., 2013; Kalchbrenner and Blunsom, 2013) or an additive neural network (Liu et al., 2013a). While these models have shown promise as methods of augmenting existing models, they have not yet offered a path for replacing or transforming existing real-valued features.

In the remainder of this chapter, we describe local discretization, our approach to learning non-linear transformations of individual features, compare it with globally non-linear models (Section 6.2), present our experimental setup (Section 6.4), empirically verify the importance of non-linear feature transformations in MT and demonstrate that discretization can be used to recover non-linear transformations (Section 6.5), discuss related work (Section 6.6), sketch some qualitative comparisons with other non-linear methods (Section 6.7), and conclude (Section 6.8).

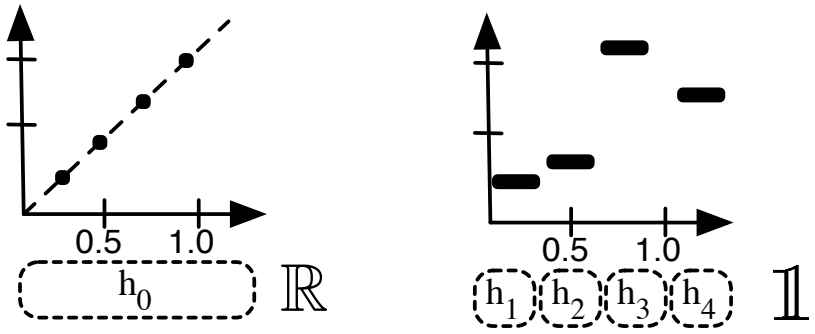


Figure 6.1: **Left:** A real-valued feature. Bold dots represent points where we could imagine bins being placed. However, since we may only adjust w_0, these "bins" will be rigidly fixed along the feature function's value. **Right:** After discretizing the feature into 4 bins, we may now adjust 4 weights independently, to achieve a non-linear re-shaping of the function.

6.2 Discretization and Feature Induction

Recall from Section 2.1.1 that a feature set that is too coarse can cause the data to be less separable than desired, resulting in uncaptured non-linearities. In this section, we propose a feature induction technique based on discretization that produces a feature set that is allows the resulting model to fit more uncaptured non-linearities.

Discretization allows us to avoid many non-linearities (Section 2.1.1) while preserving the fast inference provided by feature locality (Section 1.2). In general, feature locality is relative to a particular **structured hypothesis space**, and is unrelated to the structured features described in Section 6.3.2. We 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 (Figure 2.5). This technique is sometimes referred to as binning and is closely related to quantization. Effectively, discretization allows us to re-shape a feature function (Figure 6.1). In fact, given an infinite number of bins, we can perform any non-linear transformation of the original function.

For now, we assume a binning function $\textsc{Bin}(x) \in \mathbb{R} \rightarrow \mathbb{N}$, which we define in Section 6.2.2. To simplify notation, we define a helper function $\varphi_i(x)$, which generates a new binned feature identifier given an initial feature h_i with value x:

$$\varphi_i(x) \triangleq i \frown \textsc{Bin}(x) \tag{6.1}$$

where the $\frown$ operator indicates concatenation of a feature identifier with a bin identifier to form a new, unique feature identifier.

Our feature induction function operator Φ^{Disc} iterates over the initially continuous features, maps each value to a binned feature identifier, and assigns it the value 1:

$$\Phi^{\text{Disc}}(\mathbf{h}) \triangleq \{\langle 1, \varphi_i(h_i)\rangle : h_i \in \mathbf{h}\} \tag{6.2}$$

6.2.1 Local Discretization

Unlike other approaches to non-linear learning in MT, we perform non-linear transformation on *partial* hypotheses as in equation 2.1 where discretization is applied as $\Phi_i(h_i(d))$, which allows **locally non-linear** transformations, instead of applying Φ to complete hypotheses as in $\Phi_i(H_i(D))$, which would allow **globally non-linear** transformations. This enables our transformed model to produce non-linear responses with regard to the initial feature set $\mathbf{H}$ while inference remains linear with regard to the optimized parameters $\mathbf{w}'$. Importantly, our transformed feature set requires no additional non-local information for inference.

By performing transformations within a local context, we effectively reinterpret the feature set. For example, the familiar target word count feature found in many modern MT systems is often conceptualized as "what is the count of target words in the complete hypothesis?" A hypothesis-level view of discretization would view this as "Did this hypothesis have 5 target words?". Only one such feature will fire for each hypothesis. However, local discretization reinterprets this feature as "How many phrases in the complete hypothesis have 1 target word?" Many such features are likely to fire for each hypothesis. We provide a further example of this technique in Figure 6.2. In this study, we limit ourselves to local features, leaving the traditional non-local LM feature unchanged.

In terms of predictive power, this transformation can provide the learned model with increased ability to discriminate between hypotheses. This is primarily a result of moving to a higher-dimensional feature space. As we introduce new parameters, we expect that some hypotheses that were previously indistinguishable under $\mathbf{H}$ become separable under $\mathbf{H}'$ (Section 2.1.1). We show specific examples comparing linear, locally non-linear, and globally non-linear models in Figures 6.4 - 6.8. As seen in these examples, locally non-linear models (Equation 2.1, 6.2) are not an approximation nor a subset of globally non-linear models, but rather a different class of models.

We summarize this section with the Venn diagram in Figure 6.3, comparing the strengths of globally versus locally non-linear models. The overlap represents the space of data sets separable by both approaches given some feature set. Here, we emphasize that local feature induction is not an approximation of traditional feature induction, but rather a distinct class of models.

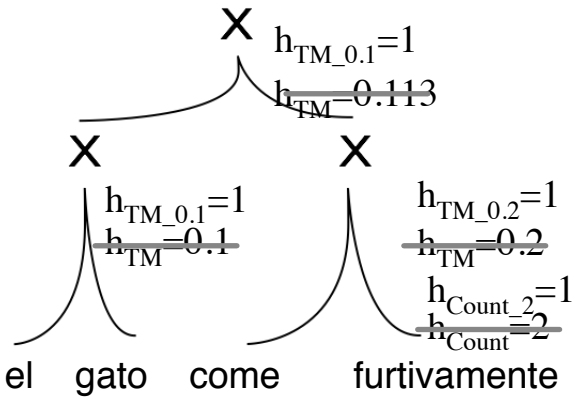


Figure 6.2: We perform discretization *locally* on each grammar rule or phrase pair, operating on the local feature vectors $\mathbf{h}$. In this example, the original real-valued features are crossed out with a solid gray line and their discretized indicator features are written above. When forming a complete hypothesis from partial hypotheses, we sum the counts of these indicator features to obtain the complete feature vector $\mathbf{H}$. In this example, $\mathbf{H} = \{H_{\text{TM\_0.1}} : 2, H_{\text{TM\_0.2}} : 1, H_{\text{Count\_2}} : 1\}$

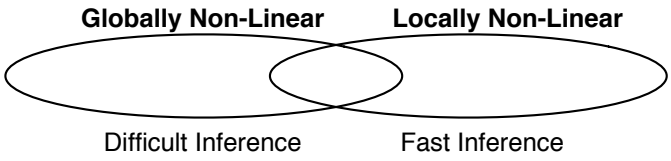


Figure 6.3: We use a Venn diagram to conceptualize the space of non-linear transformations that are possible under local (transforming h) versus global (transforming H) discretization. While many of the same rankings are reachable (overlapping region), sometimes locally transformation is capable of distinctions that global transformation is not, and sometimes the opposite is true.

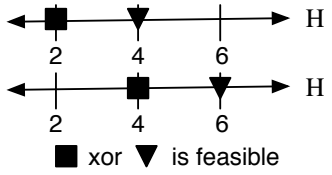
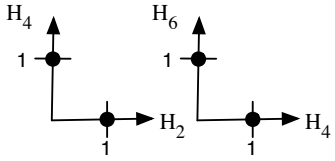
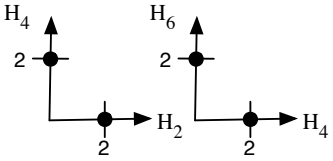
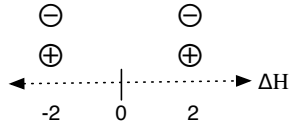
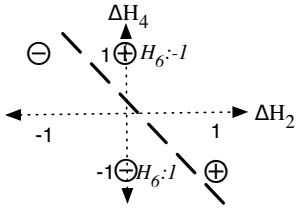
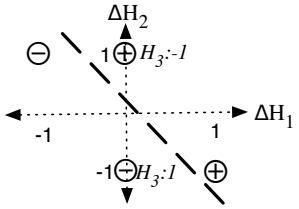
	Linear	**Globally Non-Linear**	**Locally Non-Linear**
Ranking	$^*S_3^1$ he says h:1.0 h:1.0 = {H:2.0} S_3^2 she said h:2.0 h:2.0 = {H:4.0} S_4^1 small kitten h:2.0 h:2.0 = {H:4.0} $^*S_4^2$ big lion h:3.0 h:3.0 = {H:6.0}	$^*S_3^1$ he says h:1.0 h:1.0 = {H_2:1} S_3^2 she said h:2.0 h:2.0 = {H_4:1} S_4^1 small kitten h:2.0 h:2.0 = {H_4:1} $^*S_4^2$ big lion h:3.0 h:3.0 = {H_6:1}	$^*S_3^1$ he says h_1:1 h_1:1 = {H_1:2} S_3^2 she said h_2:1 h_2:1 = {H_2:2} S_4^1 small kitten h_2:1 h_2:1 = {H_2:2} $^*S_4^2$ big lion h_3:1 h_3:1 = {H_3:2}
Viterbi		 S_3^1, S_4^2 **Feasible**	 S_3^1, S_4^2 **Feasible**
Pairs	(S_3^1, S_3^2) {ΔH:-2.0} ⊕ (S_3^2, S_3^1) {ΔH:2.0} ⊖ (S_4^2, S_4^1) {ΔH:-2.0} ⊖ (S_4^1, S_4^2) {ΔH:2.0} ⊕	(S_3^1, S_3^2) {ΔH_2:1, ΔH_4:-1} ⊕ (S_3^2, S_3^1) {ΔH_2:-1, ΔH_4:1} ⊖ (S_4^2, S_4^1) {ΔH_4:1, ΔH_6:-1} ⊖ (S_4^1, S_4^2) {ΔH_4:-1, ΔH_6:1} ⊕	(S_3^1, S_3^2) {ΔH_1:2, ΔH_2:-2} ⊕ (S_3^2, S_3^1) {ΔH_1:-2, ΔH_2:2} ⊖ (S_4^2, S_4^1) {ΔH_2:2, ΔH_3:-2} ⊖ (S_4^1, S_4^2) {ΔH_2:-2, ΔH_3:2} ⊕
Pairwise Ranking	 **Inseparable**	 **Separable**	 **Separable**

Figure 6.4: An example showing a **collinearity** over multiple input sentences S_3, S_4 in which the oracle-best hypothesis is "trapped" along a line with other lower quality hypotheses in the linear model's output space. **Ranking** shows how the hypotheses would appear in a k-best list with each partial derivation having its partial feature vector h under it; the complete feature vector H is shown to the right of each hypothesis and the oracle-best hypothesis is notated with a *. **Pairs** explicates the implicit pairwise rankings. **Pairwise Ranking** graphs those pairs in order to visualize whether or not the hypotheses are separable. ($\oplus$ indicates that the pair of hypotheses is ranked correctly according to the extrinsic metric and $\ominus$ indicates the pair is ranked incorrectly. In the pairwise ranking row, some $\oplus$ and $\ominus$ points are annotated with their positions along the third axis H_3 (omitted for clarity). Collinearity can also occur with a single input having at least 3 hypotheses.

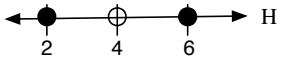
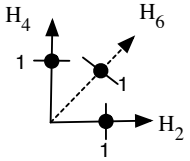
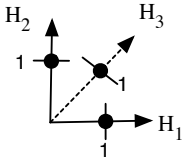
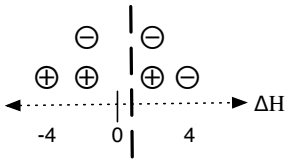
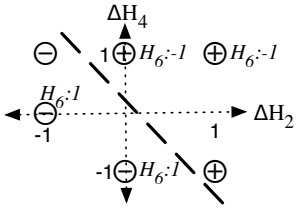
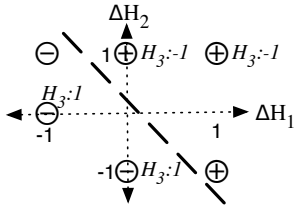
	Linear	**Globally Non-Linear**	**Locally Non-Linear**
Ranking	S_1^1 the cat h:1.0 h:1.0 = {H:2.0} $^*S_1^2$ a kitty h:2.0 h:2.0 = {H:4.0} S_1^3 an animal h:3.0 h:3.0 = {H:6.0}	S_1^1 the cat h:1.0 h:1.0 = {H_2:1} $^*S_1^2$ a kitty h:2.0 h:2.0 = {H_4:1} S_1^3 an animal h:3.0 h:3.0 = {H_6:1}	S_1^1 the cat h_1:1 h_1:1 = {H_1:2} $^*S_1^2$ a kitty h_2:1 h_2:1 = {H_2:2} S_1^3 an animal h_3:1 h_2:1 = {H_3:2}
Viterbi	 S_1^2 **Infeasible**	 S_1^2 **Feasible**	 S_1^2 **Feasible**
Pairs	(S_1^1, S_1^2) {ΔH:-2.0} ⊕ (S_1^2, S_1^1) {ΔH:2.0} ⊖ (S_1^2, S_1^3) {ΔH:-2.0} ⊕ (S_1^3, S_1^2) {ΔH:2.0} ⊖ (S_1^1, S_1^3) {ΔH:-4.0} ⊕ (S_1^3, S_1^1) {ΔH:4.0} ⊖	(S_1^1, S_1^2) {ΔH_2:1, ΔH_4:-1} ⊕ (S_1^2, S_1^1) {ΔH_2:-1, ΔH_4:1} ⊖ (S_1^2, S_1^3) {ΔH_4:1, ΔH_6:-1} ⊕ (S_1^3, S_1^2) {ΔH_4:-1, ΔH_6:1} ⊖ (S_1^1, S_1^3) {ΔH_2:1, ΔH_6:-1} ⊕ (S_1^3, S_1^1) {ΔH_2:-1, ΔH_6:1} ⊖	(S_1^1, S_1^2) {ΔH_1:2, ΔH_2:-2} ⊕ (S_1^2, S_1^1) {ΔH_1:-2, ΔH_2:2} ⊖ (S_1^2, S_1^3) {ΔH_2:2, ΔH_3:-2} ⊕ (S_1^3, S_1^2) {ΔH_2:-2, ΔH_3:2} ⊖ (S_1^1, S_1^3) {ΔH_1:2, ΔH_3:-2} ⊕ (S_1^3, S_1^1) {ΔH_1:-2, ΔH_3:2} ⊖
Pairwise Ranking	 **Inseparable**	 **Separable**	 **Separable**

Figure 6.5: An example showing a **collinearity** for a single sentence.

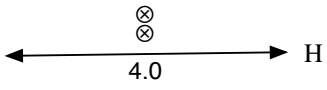
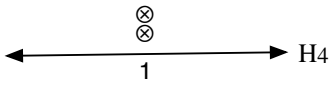
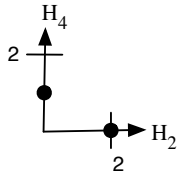
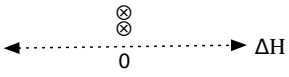
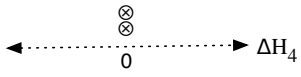
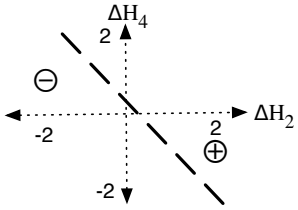
	Linear	**Globally Non-Linear**	**Locally Non-Linear**
Ranking	$^*S_2^1$ some things h:2.0 h:2.0 = {H:4.0} S_2^2 something h:4.0 = {H:4.0}	$^*S_2^1$ some things h:2.0 h:2.0 = {H_4:1} S_2^2 something h:4.0 = {H_4:1}	$^*S_2^1$ some things h_2:1 h_2:1 = {H_2:2} S_2^2 something h_4:1 = {H_4:1}
Viterbi	 **Infeasible**	 **Infeasible**	 **Feasible**
Pairs	(S_2^1, S_2^2) {ΔH:0.0} ⊕ (S_2^2, S_2^1) {ΔH:0.0} ⊖	(S_2^1, S_2^2) {ΔH_4:0} ⊕ (S_2^2, S_2^1) {ΔH_4:0} ⊖	(S_2^1, S_2^2) {ΔH_2:2, ΔH_4:-1} ⊕ (S_2^2, S_2^1) {ΔH_2:-2, ΔH_4:1} ⊖
Pairwise Ranking	 **Inseparable**	 **Inseparable**	 **Separable**

Figure 6.6: An example showing a trivial "collision" in which two hypotheses of differing quality receive the same model score until local discretization is applied. The two hypotheses are indistinguishable under a linear model with the feature set H, as shown by the zero-difference in the "pairs" row. While a globally non-linear transformation does not yield any improvement, local discretization allows the hypotheses to be properly ranked due to the higher-dimensional feature space H_2, H_4. See Figure 6.4 for an explanation of notation.

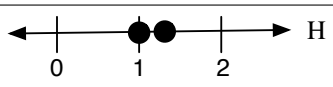
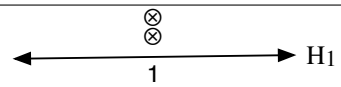
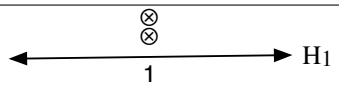
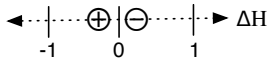
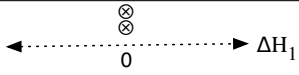
	Linear	**Globally Non-Linear**	**Locally Non-Linear**
Ranking	$^*S_5^1$ hi h:1.0 = {H:1.0} S_5^2 hello h:1.1 = {H:1.1} **extrinsic best*	$^*S_5^1$ hi h:1.0 = {H_1:1} S_5^2 hello h:1.1 = {H_1:1} **extrinsic best*	$^*S_5^1$ hi h_1:1 = {H_1:1} S_5^2 hello h_1:1 = {H_1:1} **extrinsic best*
Viterbi	 **Feasible**	 **Infeasible**	 **Infeasible**
Pairs	(S_2^1, S_2^2) {ΔH:-0.1} ⊕ (S_2^2, S_2^1) {ΔH:0.1} ⊖	(S_2^1, S_2^2) {ΔH_1:0} ⊕ (S_2^2, S_2^1) {ΔH_1:0} ⊖	(S_2^1, S_2^2) {ΔH_1:0} ⊕ (S_2^2, S_2^1) {ΔH_1:0} ⊖
Pairwise Ranking	 **Inseparable**	 **Inseparable**	 **Separable**

Figure 6.7: A example in which the initial binning step of discretization *prevents* the model from distinguishing two hypotheses that were previously separable by causing them to collide. This can happen when two feature scores are sufficiently close that they fall in the same bin. Such occurrences are mitigated by the presence of other features. Experimentally we observe that this does not affect aggregate translation quality.

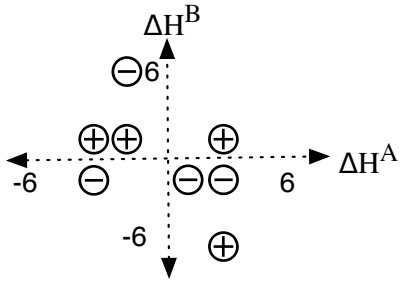
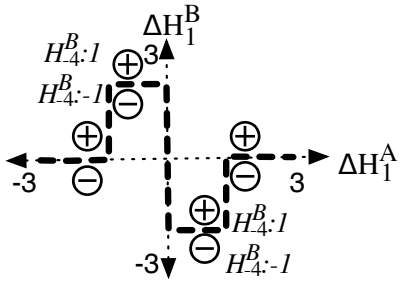
	Linear	**Locally Non-Linear**
Ranking	$*S_6^1$ kitten h^B:1.0 = {H^B:1.0} S_6^2 small cat h^A:1.0 h^A:1.0 = {H^A:2.0} --- $*S_7^1$ big marine mammal h^A:1.0 h^A:1.0 h^B:1.0 = {H^A:2.0, H^B:1.0} S_7^2 manitee h^A:0.0 = {} --- $*S_8^1$ the common house cat h^B:-4.0 h^B:1.0 h^B:1.0 h^B:1.0 = {H^B:-1.0} S_8^2 feline h^A:1.0 = {H^A:1.0} --- $*S_9^1$ the kanine h^B:-4.0 h^A:1.0 = {H^A:1.0, H^B:-4.0} S_9^2 big dog h^B:1.0 h^B:1.0 = {H^B:2.0}	$*S_6^1$ kitten h_1^B:1 = {H_1^B:1} S_6^2 small cat h_1^A:1 h_1^A:1 = {H_1^A:2} --- $*S_7^1$ big marine mammal h_1^A:1 h_1^A:1 h_1^B:1 = {H_1^A:2, H_1^B:1} S_7^2 manitee h_1^A:0 = {} --- $*S_8^1$ the common house cat h_{-4}^B:1 h_1^B:1 h_1^B:1 h_1^B:1 = {H_{-4}^B:1, H_1^B:3} S_8^2 feline h_1^A:1 = {H_1^A:1} --- $*S_9^1$ the kanine h_{-4}^B:1 h_1^A:1 = {H_1^A:1, H_{-4}^B:1} S_9^2 big dog h_1^B:1 h_1^B:1 = {H_1^B:2}
Pairs	(S_6^1, S_6^2) {ΔH^A:-2.0, ΔH^B:1.0} $\oplus$ (S_6^2, S_6^1) {ΔH^A:2.0, ΔH^B:-1.0} $\ominus$ (S_7^1, S_7^2) {ΔH^A:2.0, ΔH^B:1.0} $\oplus$ (S_7^2, S_7^1) {ΔH^A:-2.0, ΔH^B:-1.0} $\ominus$ (S_8^1, S_8^2) {ΔH^A:-1.0, ΔH^B:1.0} $\oplus$ (S_8^2, S_8^1) {ΔH^A:1.0, ΔH^B:-1.0} $\ominus$ (S_9^1, S_9^2) {ΔH^A:1.0, ΔH^B:-6.0} $\oplus$ (S_9^2, S_9^1) {ΔH^A:-1.0, ΔH^B:6.0} $\ominus$	(S_6^1, S_6^2) {ΔH_1^A:-2, ΔH_1^B:1} $\oplus$ (S_6^2, S_6^1) {ΔH_1^A:2, ΔH_1^B:-1} $\ominus$ (S_7^1, S_7^2) {ΔH_1^A:2, ΔH_1^B:1} $\oplus$ (S_7^2, S_7^1) {ΔH_1^A:-2, ΔH_1^B:-1} $\ominus$ (S_8^1, S_8^2) {ΔH_1^A:-1, ΔH_1^B:3, ΔH_{-4}^B:1} $\oplus$ (S_8^2, S_8^1) {ΔH_1^A:1, ΔH_1^B:-3, ΔH_{-4}^B:-1} $\ominus$ (S_9^1, S_9^2) {ΔH_1^A:1, ΔH_1^B:-2, ΔH_{-4}^B:1} $\oplus$ (S_9^2, S_9^1) {ΔH_1^A:-1, ΔH_1^B:2, ΔH_{-4}^B:-1} $\ominus$
Pairwise Ranking	 **Inseparable**	 **Separable**

Figure 6.8: An example demonstrating a non-linear decision boundary induced by discretization. The non-linear nature of the decision boundary can be seen clearly when the induced feature set H_1^A, H_1^B, H_{-4}^B (right) is considered in the original feature space H^A, H^B (left). In the **pairwise ranking** row, two axes (H_1^A, H_1^B) are plotted while the third axis H_{-4}^B is indicated only as stand-off annotations for clarity. Given a larger number of hypotheses, such situations could also arise within a single sentence. See Figure 6.4 for an explanation of notation.

6.2.2 Binning Algorithm

To initialize the learning procedure, we construct the binning function BIN used by the indicator dicretizer Φ. We have two desiderata: (1) any monotonic transformation of a feature should not affect the induced binning since we should not require feature engineers to determine the optimal feature transformation and (2) no bin's data should be so sparse that the optimizer cannot reliably estimate a weight for each bin. Therefore, we construct bins that are (i) populated uniformly subject to (ii) each bin containing no more than one feature value. We call this approach **uniform population feature binning**.

While one could consider the predictive power of the features when determining bin boundaries, this would suggest that we should jointly optimize and determine bin boundaries, which is beyond the scope of this work. This problem has recently been considered for NLP by Suzuki and Nagata (2013) and for MT by Liu et al. (2013b), though the latter involves decoding the entire training data.

Let $\mathcal{X}$ be the list of feature values to bin where i indexes feature values $x_i \in \mathcal{X}$ and their associated frequencies f_i. We want each bin to have a uniform size u. For the sake of simplifying our final algorithm, we first create adjusted frequencies f'_i so that very frequent feature values will not occupy more than 100% of a bin via the following algorithm, which iterates over k:

$$u^k \triangleq \frac{1}{|\mathcal{X}|} \sum_{i=1}^{|\mathcal{X}|} f_i^k \tag{6.3}$$

$$f_i^{k+1} \triangleq \min(f_i^k, u^k) \tag{6.4}$$

which returns $u' = u^k$ when $f_i^k < u^k \; \forall i$. Next, we solve for a binning B of N bins where $\overline{\overline{b_j}}$ is the population of each bin:

$$\operatorname*{argmin}_B \frac{1}{N} \sum_{j=1}^{N} |\overline{\overline{b_j}} - u'| \tag{6.5}$$

We use Algorithm 3 to produce this binning. In our experiments, we construct a translation model for each sentence in our tuning corpus; we then add a feature value instances to $\mathcal{X}$ for each rule instance.

6.3 Structure-Aware Regularization

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

Algorithm 3 POPULATEBINSUNIFORMLY($\mathcal{X}, N$)

$\triangleright$ Remaining values for b_j, s.t. $\overline{\overline{b_k}} > 0 \; \forall k$
def $R(j) = |\mathcal{X}| - (N - j - 1)$
$\triangleright$ Remaining frequency mass within ideal bound
def $C(j) = j \cdot u' - \sum_k^j \overline{\overline{b_k}}$
$i \leftarrow 1$ $\triangleright$ Current feature value
for $j \in [1, N]$ **do**
  while $i \leq R(j)$ **and** $f_i \leq C(j)$ **do**
    $b_j \leftarrow b_j \cup \{x_i\}$
    $i \leftarrow i + 1$
  end while
  $\triangleright$ Handle value that straddles ideal boundaries
    by minimizing its violation of the ideal
  if $i \leq R(j)$ **and** $\frac{f_i - C(j)}{f_i} < 0.5$ **then**
    $b_j \leftarrow b_j \cup \{x_i\}$
    $i \leftarrow i + 1$
  end if
end for
return B

- **Fidelity.** 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, we 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.

While one may be tempted to simply throw more data or millions of sparse features at the problem, we elect to more strategically use the existing data, since (1) large in-domain tuning data is not always readily available, and (2) when it is available, it can add considerable computational expense. In this section, we explore methods for mitigating data sparsity by embedding more knowledge into the learning procedure.

6.3.1 Overlapping Bins

One simple way we could combat sparsity is to extend the edges of each bin such that they cover their neighbors' values (see Equation 6.2):

$$\Phi^{\text{Overlap}}(\mathbf{h}) \triangleq \left\{ \left\langle \left[\!\left[\text{BIN}_i(h_i) \in \cup_{k=i-1}^{i+1} \text{BIN}_k(h_i) \right]\!\right], \varphi_i(h_i) \right\rangle \; : h_i \in \mathbf{h} \right\} \tag{6.6}$$

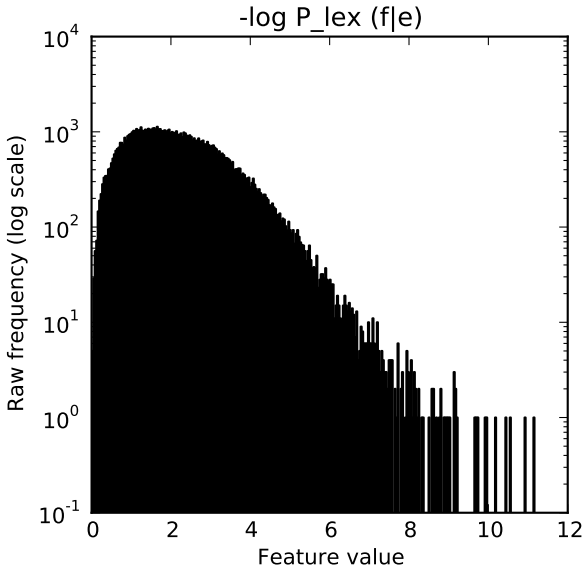


Figure 6.9: A fine-grained histogram of the number of grammar rules having each value of the $p_{\text{lex}}(f|e)$ feature. The bin boundaries are placed at intervals along the feature value such that the population is distributed evenly among the bins

This way, each bin will have more data points to estimate its weight, reducing data sparsity, and the bins will mutually constrain each other, reducing the ability to overfit. We include this technique as a contrastive baseline for structured regularization.[2]

6.3.2 Linear 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.

Unregularized discretization is potentially 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. These features make up a **structured feature space**. With these things in mind, we propose linear neighbor regularization, a structured regularizer that 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

[2]We also performed some preliminary experiments with simple thresholds where each bin includes all values less than some threshold, but did not observe any differences with overlapping bins.

the indicator features that represent neighboring values of the original real-valued feature such that the resulting transformation appears somewhat smooth.

To incorporate this knowledge of nearby bins, the linear neighbor regularizer $\mathcal{R}_{\text{LNR}}$ uses a set of 3-tuples that indicate each feature's left and right neighbors $\langle l, k, r\rangle \in \mathcal{N}$ to penalize each feature's weight by the squared amount it differs from its neighbors' midpoint:

$$\mathcal{R}_{\text{LNR}}(\mathbf{w}, l, k, r) \triangleq \left(\frac{1}{2}(w_l + w_r) - w_k\right)^2 \tag{6.7}$$

$$\mathcal{R}_{\text{LNR}}(\mathbf{w}; \mathcal{N}) \triangleq \beta \sum_{\langle l,k,r\rangle \in \mathcal{N}} \mathcal{R}_{\text{LNR}}(\mathbf{w}, l, k, r) \tag{6.8}$$

This is a special case of the feature network regularizer of Sandler (2010) and is similar to the fused lasso (Tibshirani and Saunders, 2005). Unlike traditional regularizers, we do not hope to reduce the active feature count. With the basic PRO loss function l and a ℓ_2 regularization term $\mathcal{R}_2$, the complete modified loss function of the convex optimization problem internal to each iteration of PRO is then:

$$\mathcal{L}(\mathbf{w}) = l(\mathbf{x}, \mathbf{y}; \mathbf{w}) + \mathcal{R}_2(\mathbf{w}) + \mathcal{R}_{\text{LNR}}(\mathbf{w}) \tag{6.9}$$

6.3.3 Monotone Neighbor Regularization

However, as $\beta \to \infty$, the linear neighbor regularizer $\mathcal{R}_{\text{LNR}}$ forces a linear arrangement of weights – this violates our premise that we should be agnostic to non-linear transformations. We now describe a structured regularizer $\mathcal{R}_{\text{MNR}}$ whose limiting solution is any monotone arrangement of weights. We augment $\mathcal{R}_{\text{LNR}}$ with a smooth damping term $\mathcal{D}(\mathbf{w}, l, k, r)$, which has the shape of a bathtub curve with steepness γ:

$$\mathcal{D}(\mathbf{w}, l, k, r) \triangleq \tanh^{2\gamma} \frac{\frac{1}{2}(w_l + w_r) - w_k}{\frac{1}{2}(w_l - w_r)} \tag{6.10}$$

$$\mathcal{R}_{\text{MNR}}(\mathbf{w}, \mathcal{N}) \triangleq \beta \sum_{\langle l,k,r\rangle \in \mathcal{N}} \mathcal{D}(\mathbf{w}, l, k, r)\mathcal{R}_{\text{LNR}}(\mathbf{w}, l, k, r) \tag{6.11}$$

$\mathcal{D}$ is nearly zero while $w_k \in [w_l, w_r]$ and nearly one otherwise. Briefly, the numerator measures how far w_k is from the midpoint of w_l and w_r while the denominator scales that distance by the radius from the midpoint to the neighboring weight.

6.4 Experimental Setup

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. 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.

Decoder: For decoding, we will use cdec (Dyer et al., 2010), a multi-pass decoder that supports syntactic translation models and sparse features.

Optimizer: Optimization is performed using PRO (Hopkins and May, 2011) as implemented by the cdec decoder. We run PRO for 30 iterations as suggested by Hopkins and May (2011). The PRO optimizer internally uses a L-BFGS optimizer with the default ℓ_2 regularization implemented in cdec. Any additional regularization is explicitly noted.

Baseline Features: We use the baseline features produced by Lopez' suffix array grammar extractor (Lopez, 2008a), which is distributed with cdec. Note that these features may be simplified or removed as specified in each experimental condition. These features are:

- $\log P_{\text{coherent}}(e|f)$: The coherent phrase-to-phrase translation probability (Lopez, 2008b).
- $\log P_{\text{lex}}(e|f)$, $\log P_{\text{lex}}(f|e)$: 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)

Chinese Resources: For the Chinese→English experiments, including the completed work presented in this proposal, we train on the Foreign Broadcast Information Service (FBIS)

	Zh→En	Ar→En	Cz→En
Train	303K	5.4M	1M
WeightTune	1664	1797	3000
HyperTune	1085	1056	2000
Test	1357	1313	2000

Table 6.1: Corpus statistics: number of parallel sentences.

corpus[3] of approximately 300,000 sentence pairs with about 9.4 million English words. We tune weights on the NIST MT 2006 dataset, tune hyperparameters on NIST MT05, and test on NIST MT 2008. See Table 6.1.

Arabic Resources: We build an Arabic→English system, training on the large NIST MT 2009 constrained training corpus[4] of approximately 5 million sentence pairs with about 181 million English words. We tune weights on the NIST MT 2006 dataset, tune hyperparameters on NIST MT 2005, and test on NIST MT 2008.[5] See Table 6.1.

Czech resources: We also construct a Czech→English system based on the CzEng 1.0 data (Bojar et al., 2012). First, we lowercased and performed sentence-level deduplication of the data.[6] Then, we uniformly sampled a training set of 1M sentences (sections 1 – 97) along with a weight-tuning set (section 98), hyperparameter-tuning (section 99), and test set (section 99) from the paraweb domain contained of CzEng.[7] Sentences having length less than 5 were discarded due to their noisy nature. See Table 6.1.

Evaluation: We quantify increases in translation quality using case-insensitive BLEU (Papineni et al., 2002). We control for test set variation and optimizer instability by averaging over multiple optimizer replicas (Clark et al., 2011).[8]

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

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

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

[6]CzEng is distributed deduplicated at the document level, leading to very high sentence-level overlap.

[7]The section splits recommended by Bojar et al. (2012).

[8]MultEval 0.5.1: github.com/jhclark/multeval

Bits	4	8	12
Features	101	1302	12,910
Test BLEU	36.4	36.6	36.8

Table 6.2: Translation quality for Cz→En system with varying bits for discretization. For all other experiments, we tune the number of bits on held-out data.

6.5 Results

6.5.1 Does Non-Linearity Matter?

In our first set of experiments, we seek to answer "Does non-linearity matter?" by starting with our baseline system of 7 typical features (the log probability system) and we then remove the log transform from all of the log probability features in our grammar (the Probs. system). The results are shown in Table 6.3 (rows 1, 2). If a naïve feature engineer were to remove the non-linear log transform, the systems would degrade between 1.1 BLEU and 3.6 BLEU. From this, we conclude that non-linearity does affect translation quality. This is a potential pitfall for any novel real-valued feature including probability features, count features, similarity measures, etc.

6.5.2 Learning Non-Linear Transformations

Next, we evaluate the effects of discretization (Disc), overlapping bins (Over.), linear neighbor regularization (LNR), and monotone neighbor regularization (MNR) on three language pairs: a small Zh→En system, a large Ar→En system and a large Cz→En system. In the first row of Table 6.3, we use raw probabilities rather than log probabilities for $p_{\text{coherent}}(t|s)$, $p_{\text{lex}}(t|s)$, and $p_{\text{lex}}(s|t)$. In rows 3 – 7, all translation model features (without the log-transformed features) are then discretized into indicator features.[9] The number of bins and the structured regularization strength were tuned on the hyperparameter tuning set.

Discretization alone does not consistently recover the performance of the log transformed features (row 3). The naïve overlap strategy in fact degrades performance (row 4). Linear neighbor regularization (row 5) behaves more consistently than discretization alone, but is consistently outperformed by the monotone neighbor regularizer (row 6), which is able to meet or significantly exceed the performance of the log transformed system. Importantly, this is done without any knowledge of the correct non-linear transformation. In the final row, we go a step further by removing $p_{\text{coherent}}(t|s)$ altogether and replacing it with simple count features: $c(s)$ and $c(s,t)$, with slight to no degradation in quality. We

[9] We also keep a real-valued copy of the word penalty to help normalize the language model.

Condition	Zh→En	Ar→En	Cz→En
P	20.8* (-2.7)	44.3* (-3.6)	36.5* (-1.1)
log P	23.5†	47.9†	37.6†
Disc P	23.4† (-0.1)	47.2† (-0.7)	36.8* (-0.8)
Over. P	20.7* (-2.8)	44.6* (-3.3)	36.6* (-1.0)
LNR P	23.1*† (-0.4)	48.0† (+0.1)	37.3 (-0.3)
MNR P	23.8† (+0.3)	48.7*† (+0.8)	37.6† (±)
MNR C	23.6† (±)	48.7*† (+0.8)	37.4† (-0.2)

Table 6.3: **Top:** Translation quality for systems with and without the typical log transform. **Bottom:** Translation quality for systems using discretization and structured regularization with probabilities P or counts C as the input of discretization. MNR P consistently recovers or outperforms a state-of-the-art system, but without any assumptions about how to transform the initial features. Differences in BLEU versus the log P system are shown in parentheses. Note that we observe much larger improvements (up to 4.4 BLEU) when a good transform is unknown (compare to raw P system). All scores are averaged over 3 end-to-end optimizer replications. * denotes significantly different than log probs (row 2) with $p(\text{CHANCE}) < 0.01$ under Clark et al. (2011) and † is likewise used with regard to P (row 1).

take this as evidence that a feature engineer developing a new real-valued feature may find discretization and monotone neighbor regularization useful.

We also observe that different data sets benefit from non-linear feature transformation to different degrees (Table 6.3, rows 1, 2). We noticed that discretization with monotone neighbor regularization is able to improve over a log transform (rows 2, 6) in proportion to the improvement of a log transform over probability-based features (rows 1, 2).

To provide insight into how translation quality can be affected by the number of bits for discretization, we offer Table 6.2.

In Figure 6.10, we present the weights learned by the Ar→En system for probability-based features. We see that even without a bias toward a log transform, a log-like shape still emerges for many SMT features based only on the criteria of optimizing BLEU and a preference for monotonicity. However, the optimizer chooses some important variations on the log curve, especially for low probabilities, that lead to improvements in translation quality.

6.6 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

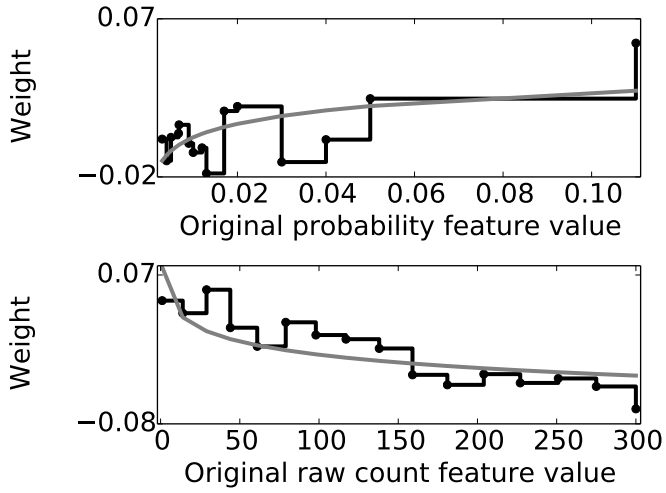


Figure 6.10: Plots of weights learned for the discretized $p_{\text{coherent}}(e|f)$ (top) and $c(f)$ (bottom) for the Ar→En system with 4 bits and monotone neighbor regularization. $p(e|f) > 0.11$ is omitted for exposition as values were constant after this point. The gray line fits a log curve to the weights. The system learns a shape that deviates from the log in several regions. Each non-monotonic segment represents the learner choosing to better fit the data while paying a strong regularization penalty.

real-valued inputs or perform suboptimally given real-valued inputs (Dougherty, Kohavi, and Sahami, 1995; Kotsiantis and Kanellopoulos, 2006).

Decision trees and random forests have been successfully used in language modeling (Jelinek et al., 1994; Xu and Jelinek, 2004) and parsing (Charniak, 2010; Magerman, 1995).

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 non-linear kernel PCA for word sense disambiguation. Tsochantaridis et al. (2004) describe a structured SVM for grammar learning, named-entity recognition, text classification, and sequence alignment. 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. Taskar, Guestrin, and Koller (2003) describe a method for incorporating kernels into structured Markov networks. Tsochantaridis et al. (2004) then proposed a structured SVM for grammar learning, named-entity recognition, text classification, and sequence alignment. This was followed by a structured SVM

with inexact inference (Finley and Joachims, 2008) and the latent structured SVM (Yu and Joachims, 2009).

In MT, Giménez and Màrquez (2007) used a SVM to annotate a phrase table with binary features indicating whether or not a phrase translation was appropriate in context. Nguyen, Mahajan, and He (2007) also applied non-linear features for SMT n-best reranking.

Perhaps most similar to this work is that of Toutanova and Ahn (2013), which uses gradient-boosting machines, a form of regression decision trees, to induce locally non-linear features. They apply these features in a n-best reranking framework. He and Deng (2012) directly optimize the lexical and phrasal probability features using expected BLEU to generate a new phrase table. Nguyen, Mahajan, and He (2007) explore bin features, Gaussian mixture models, and Parzen windows for global non-linear learning in a re-ranking framework. Nelakanti et al. (2013) use tree-structured ℓ_p regularizers to train language models and observe improvements in perplexity over a Kneser-Ney. Yogatama and Smith (2014) apply a variety of structured regularizers based on the group lasso to achieve improvements on topic classification, sentiment analysis, and forecasting.

Learning parameters under weak order restrictions has also been studied for regression. Isotonic regression (Barlow et al., 1972; Robertson, Wright, and Dykstra, 1988; Silvapulle and Sen, 2005) fits a curve to a set of data points such that each point in the fitted curve is greater than or equal to the previous point in the curve. Nearly isotonic regression allows violations in monotonicity (Tibshirani, Hoefling, and Tibshirani, 2011).

6.7 Qualitative Comparison with Other Non-Linear Models

While many models are capable of producing non-linear responses to input features, the specific class of non-linear responses can often differ significantly. In this section, we compare several well-known non-linear models while considering the following:

- **Expressivity:** What are the set of outputs that can be produced by the model? VC dimensionality is one way that this is often quantified. However, to avoid being sensitive to pathological arrangements of inputs, which may not be able to be shattered, we instead discuss specific classes of non-linear responses relevant to understanding discretization.

- **Overfitting:** Does the model have a large number of parameters? Even if the model's expressivity theoretically allows for tightly fitting the tuning data, is it easy to avoid overfitting with regard to held-out test data?

- **Underfitting:** It it theoretically and practically easy to find a good parameterization that maximizes the objective function on the tuning data? Underfitting can be a concern for some non-convex loss functions.
- **Feature locality:** What are the considerations that must be made in inference and learning for structured prediction?

6.7.1 Neural Networks

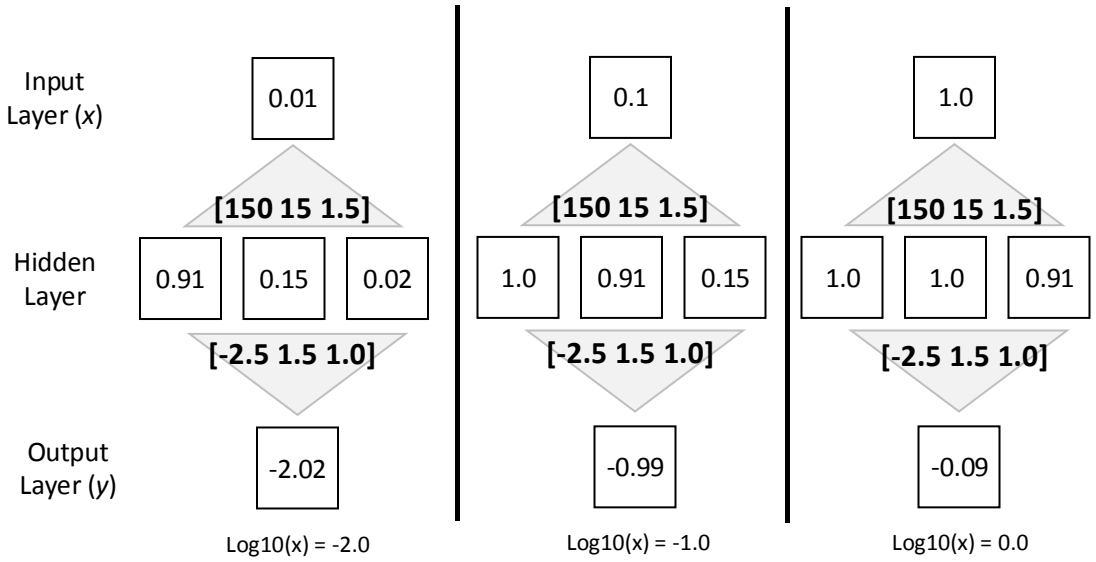


Figure 6.11: A neural network simulating discretization with a single real-valued input neuron (top), single tanh hidden layer with 3 neurons (middle), and a single softmax real-valued output (bottom). The log10 transformation of the input that the network seeks to approximate is shown below the output neuron. The projection matrix for each layer is shown in bold above each layer. The hidden layer uses a tanh function while the output layer uses a softmax function. The network is instantiated three times with different input values: 0.01 (left), 0.1 (middle), and 1.0 (right) to show points of interest in the log10 transformation. See section 6.7.1 for a full discussion.

Neural networks (NN's) have recently gained popularity in NLP and MT (Schwenk, 2012; Auli et al., 2013; Kalchbrenner and Blunsom, 2013). While they are known to be capable of producing non-linear responses to inputs, we will specifically compare their responses to the class of non-linear responses that are possible via discretization. Neural networks are directed graphical models, which typically consist of an input layer analogous to the initial features H, one or more hidden layers that learn a new representation of the

initial features, and an output layer analogous to the final scoring function. Often, a non-linear transformation such as tanh is applied to each neuron of the hidden layer (with the notable exception of log bi-linear models); in practice, tanh tends to act as a differentiable proxy for indicator variables since it asymptotically approaches 0 or 1 very quickly. This makes the neurons in the hidden layer analogous to the transformation Φ into indicator bins performed by discretization.

NN's are known to be universal function approximators(Scarselli and Tsoi, 1998), but these proofs often come with caveats such as requiring a prohibitively large number of neurons in the hidden layer(s), further motivating us to more transparently compare NN's with discretization.

With this in mind, there are direct similarities between the expressivity of NN with a single hidden layer and discretization. To make this concrete, figure 6.11 shows how a single-layer NN might accomplish a log10 transformation using a manually chosen set of weights. Unlike our implementation of discretization, the "bins" formed by the hidden layer are cumulative in nature (i.e. they are active if the input is greater than some value as opposed to if the input is in some range). The single neuron in the input layer represents a single real-valued feature. It is then projected into a new feature representation in the hidden layer; the first projection matrix has been chosen such that each of the three hidden neurons become active for inputs of 0.01, 0.1, and 1.0, respectively (see the network instances, left to right). The output projection matrix is analogous to the typical weights of a linear model or bin weights in the case of discretization.

- **Expressivity:** Not only is the neural network able to approximate a log transform and many other of non-linear transformations enabled by discretization, as a universal function approximator (Scarselli and Tsoi, 1998), it is able to approximate an even wider range of functions.

- **Overfitting:** With this additional expressivity comes an enhanced ability to overfit the tuning data. In this thesis, we mitigate the overfitting due to discretization by applying a structured regularizer. However, in the case of neural networks, it is less clear how to apply such a regularizer since we have no reason to believe a priori that the weights projecting the hidden layer to the output layer should be spatially related to each other.

- **Underfitting:** To avoid underfitting due to non-convexity, neural networks require additional consideration during learning. For example, Liu et al. (2013a) use both pre-training and post-training to avoid poor local optima.

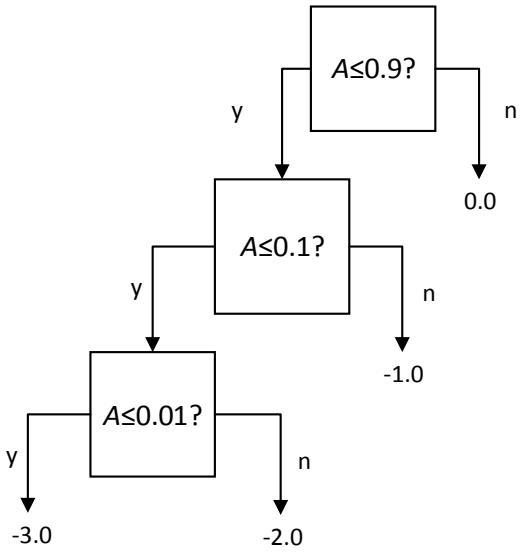


Figure 6.12: A decision tree, showing a possible emulation of a discretized log transform for a single real-valued feature (attribute A). See section 6.7.2 for a full discussion.

- **Feature locality:** As with global discretization, exact application of NN's at the hypothesis-level during decoding is computationally infeasible. Liu et al. (2013a) propose additive NN's, which address this issue by applying non-linear transformations only within rule-local contexts, consistent with the concept of locally non-linear models presented in this chapter.

While neural networks are a promising means of non-linear modeling, their very broad expressive power requires additional considerations during learning. Because of this, the same structured regularization tools that make discretization practical for SMT are not readily applicable to NN's.

6.7.2 Decision Trees

Decision trees are another popular non-linear model. They are constructed by iteratively adding levels to the decision tree by selecting the attribute that maximizes some splitting criterion such as information gain or gain ratio. For regression, each leaf of the decision tree can then produce an output value.

The default tests considered by C4.5, a popular decision tree induction algorithm, are (Quinlan, 1996):

- $A = x$ for a discrete (categorical) attribute A for each outcome x. (These are typically implemented as indicator features in modern MT systems.)

- $A \leq x$ for a continuous (real-valued) attribute A. To find the threshold x that maximizes the splitting criterion, the observed unique values of A are sorted. The midpoint value between each of the observed values is considered a candidate threshold. The threshold that maximizes the splitting criterion is selected. (This procedure results in a sort of discretization).

Comparing the model characteristics of decision trees to discretization:

- **Expressivity:** Given unbounded tree depth, the tree can exactly create arbitrary mappings of the training data. Thus, decision trees can express transformations beyond what is possible with discretization. When a path from the root of the decision tree to a leaf includes multiple attributes, the decision tree can also model conjunctions, a topic we will return to in the next chapter.

- **Overfitting:** The ability to arbitrarily partition the output space enables some particularly pathological solutions and so pruning heuristics are critical to prevent overfitting.

- **Underfitting:** Despite the highly non-convex parameter space of decision trees, they are capable of very tightly fitting the tuning data. Underfitting is only a concern insomuch as the pruning heuristics cause the learned model to be overly simple. This is often addressed by using random decision forests (Ho, 1995), which aggregate many simple decision trees to reduce bias on both tuning and test data while still constraining the ability to overfit.

- **Feature locality:** The same principles of local non-linearity and a PRO-like objective can be leveraged in the context of decision trees (Toutanova and Ahn, 2013).

Toutanova and Ahn (2013) explored this direction using gradient boosting machines, a form of regression decision trees along with a PRO-like objective to induce decision trees that use only phrase-local features. The decision trees were first learned in isolation and were exposed as new features into the overall MT linear model as a second optimization pass in which the decision trees were held fixed.

In future work, it may be possible to leverage the fact that decision trees can exactly represent discretization to apply structured regularization to regression decision trees.

6.7.3 Kernel Methods in Max-Margin Learning

Kernel methods are used in large margin learning, which includes the popular Support Vector Machine (SVM). However, the underlying difference in model expressivity in these techniques is due to the choice of kernel – a SVM with a linear model is still simply a method for learning the weights of a linear model.

The polynomial kernel is not capable of re-shaping individual real-valued features in a manner similar to discretization; its primary strength lies in its ability to model conjunctions. Similarly, the Gaussian Radial Basis Function (RBF) Kernel is effectively a scaled measure of the Euclidean distance between two feature vectors, which does not enable many of the transformations possible with discretization. While the sigmoid (hyperbolic tangent) kernel is equivalent to a trivial neural network with zero hidden layers and a tanh output layer, this lack of a hidden layer makes it incapable of re-shaping individual features. NLP-specific convolution kernels such as the tree kernel and string kernel have also been proposed (Collins and Duffy, 2001). For these reasons, contemporary kernel methods are not directly comparable to discretization.

6.7.4 Splines and Kernel Density Estimation

A **spline** is a piecewise polynomial function in which the pieces are smoothly connected. They are often used for non-linear curve fitting. Yu and Deng (2009) describe the use of splines for arbitrary non-linear estimation problems and suggest their usefulness in signal processing. Along this same line of work, both bins and splines are mentioned as possible ways of handling continuous features in maximum entropy models (Yu, Deng, and Acero, 2008) though empirical results were not presented. Splines have also found use in speech recognition for reparameterizing HMMs (Yu et al., 2008). Splines may be seen as a smooth alternative to the piecewise constant function resulting from the discretization approach described in this chapter.

Closely related to splines is **kernel density estimation** (also called the Parzen-Rosenblatt window), which uses a kernel function to estimate a probability distribution given a sample of data from that distribution. While this is not the same estimation problem we face in maximizing an objective function such as PRO in a structured prediction task, there are some similarities worth mentioning. Namely, when using the uniform kernel, a piecewise constant function much like our implementation of discretization can be recovered; when using continuous kernels (e.g. Epanechnikov, Gaussian, or Logistic), a smooth spline-like curve can be produced.

In the future, such methods may allow for smooth generalizations of discretization. However, curve fitting and density estimation are inherently different problems than feature

induction, so significant work remains to adapt these techniques for use in regression or ranking for structured prediction.

6.8 Chapter Summary

In the absence of highly refined knowledge about a feature, discretization with structured regularization enables higher quality impact of new feature sets that contain non-linearities. In our experiments, we observed that discretization out-performed naïve features lacking a good non-linear transformation by up to 4.4 BLEU and that it can outperform a baseline by up to 0.8 BLEU while dropping the log transform of the lexical probabilities and removing the phrasal probabilities in favor of counts. Looking beyond this basic feature set, non-linear transformations could be the difference between showing quality improvements or not for novel features. As researchers include more real-valued features including counts, similarity measures, and separately-trained models with millions of features, we suspect this will become an increasingly relevant issue. We conclude that non-linear responses play an important role in SMT, even for a commonly-used feature set, an observation that we hope will inform feature engineers.

Combining Conjunctions and Discretization with Structured Regularization

7

Nature's patterns sometimes reflect two intertwined features.

—Brian Greene

Innovation is about bad ideas, or ideas that look like bad ideas.

—Ben Horowitz

DISCRETIZATION AND CONJUNCTION along with appropriate structured regularization have proven to be useful tools, improving translation quality while requiring less of feature engineers. Discretization provides additional freedom for the model to re-shape initially continuous features while conjunctions allow us to leverage interactions among features. In this chapter, we combine these two techniques to capture interactions among specific sub-regions of multiple initially continuous features. Toward this goal, we generalize neighbor regularization to encode spatial relationships among discretized features that have been conjoined. While we do not observe any improvements in translation quality, we offer some future directions for this line of work.

7.1 Introduction

Relaxing independence assumptions has often proven to be an effective technique for improving the accuracy of a model, provided that learning can cope with the resulting sparsity. In this thesis, we have discussed methods for relaxing the independence assumptions between initially discrete and initially continuous features (Chapter 4) and among multiple initially discrete features (Chapter 5). In Chapter 6, we relaxed the assumptions of linearity and monotonicity. Yet we are still left with an independence assumption among initially continuous features.

When might this matter? Consider discretized features for source and target length. A discretized target length feature may be able to strongly favor using long contiguous blocks of human-authored text. However, this may be a bad idea when the source phrase is only 1 word long (this can happen frequently due to the garbage collector effect[1] in word alignment); ideally, a good model would learn to penalize 1 word being translated as 5 words. By creating a conjunction on top of discretized phrase lengths, we could arrive at features such as `SrcLen1_TgtLen5`, giving the model the capability to directly penalize such bad mappings.

Similarly, consider the translation of a source word with count 10 as a target word with count 10,000. Typically, we should expect that low-frequency words will translate as low-frequency words – otherwise, we may be translating a content word as a function word. Again, a conjunction of discretized features would allow us to directly learn that such mappings do not correspond to high quality translations. In this chapter, we will formally describe how to combine discretization, conjunction, and their associated regularization strategies to allow the model to explore these possibilities.

7.2 Conjunctions on Discretized Features

Recall that we defined the discretization of initially continuous features into indicator features Φ^{Disc} in terms of a binning function $\textsc{Bin}_i(x) \in \mathbb{R} \to \mathbb{N}$. If $\varphi_i(x) = i \frown \textsc{Bin}(x)$, then the feature set inducted by discretization is:

$$h'_k = [\![\varphi_i(h_i) = k]\!] \quad \forall i \in [0, |\mathbf{h}|] \tag{7.1}$$

And our feature induction operator for discretization is:

$$\Phi^{\text{Disc}}(\mathbf{h}) \triangleq \{\langle 1, \varphi_i(h_i)\rangle : h_i \in \mathbf{h}\} \tag{7.2}$$

We extend this to conjunctions on top of these discretized features by inducing a secondary feature set. For conjunctions of order 3, we would have the feature set that takes the shape of a tensor:

$$h''_{i,j,k} = h'_i \cdot h'_j \cdot h'_k \quad \forall i, j, k \in \left[0, |\mathbf{h}'|\right] \tag{7.3}$$

A conjunction of 3 discretized features is visualized in Figure 7.1.

[1] The garbage collector effect refers to infrequent words being aligned to several words in difficult-to-align sentence pairs.

Algorithm 4 CONJOINDISCRETIZEDFEATS($\mathbf{h}$)

$\mathbf{h}' \leftarrow \{\}$
▷ Iterate over all pairs of features (for order 2 conjunctions)
for $h_i \in \mathbf{h}$ **do**
  for $h_j \in \mathbf{h}$ **do**
    ▷ Iterate over the bins generated by discretizing each feature
    for $i' \in \varphi_i(h_i)$ **do**
      for $j' \in \varphi_j(h_j)$ **do**
        ▷ Conjoined the discretized features into a new feature ID
        $k \leftarrow i' \frown j'$
        ▷ Fire the conjoined feature
        $\mathbf{h}' \leftarrow \mathbf{h}' \cup \{\langle 1, k\rangle\}$
      end for
    end for
  end for
end for
return $\mathbf{h}'$

Without loss of generality, we define the feature induction operator for conjoined discretized features of order 3 as:

$$\Phi^{\text{DiscConj}}(\mathbf{h}) \triangleq \left\{\left\langle 1, \varphi_i(h_i) \frown \varphi_j(h_j) \frown \varphi_k(h_k)\right\rangle : h_i \in \mathbf{h}, h_j \in \mathbf{h}, h_k \in \mathbf{h}, h_i \neq h_j \neq h_k\right\} \quad (7.4)$$

While this equation shows an order 3 conjunction, arbitrary orders may be defined similarly. For example, a conjunction of order 2 would result in a 2 dimensional matrix of conjunctions while order 3 would imply a 3 dimensional tensor. For exposition, we also provide an imperative algorithm for equation 7.4 in algorithm 4.

7.3 Structured Regularization

In order to combat the increased sparsity that results from the additional free parameters of our induced features, we again turn to structured regularization. There are several regularization strategies we could consider:

1. **Smooth toward neighboring features.** We already pursued a strategy of this sort in Chapter 6, but with tensor-structured features we now have multiple directions that could be considered neighbors. For example, in Figure 7.1, the inner-most feature $w_{1,1,1}$ would have 6 immediate neighbors.

2. **Smooth toward parent features.** We could also consider smoothing toward features in lower dimensional spaces, which should have more data to estimate their param-

eters. For example, for a conjoined feature `SrcLen2_TgtLen3`, we could consider encouraging its weight to be similar to the weights of `SrcLen2` and `TgtLen3`.

3. **Smooth toward zero.** If we view these conjoined features as small corrections for cases where feature independence does not hold, then we should consider the features' weights to be zero-based. This standard regularization strategy contradicts the previous strategy in that they encourage similarity toward different conflicting points.

4. **Minimize the number of degrees of freedom.** We could consider reducing the number of overall unique weights instead of pushing the weights toward zero. In this case, there is no conflict with smoothing toward neighbors. (Chapter 5)

We chose to view these conjunctions as zero-based corrections rather than inheriting from their weights from lower dimensional parent features. The main reason is that all conjunctions will have at least two parents and the parents' weights can be very different. For example, `SrcLen2` may have a small positive weight and `TgtLen5` may have a very positive weight while the actual weight we desire for the conjoined feature may be negative (since a very different length ratio is symptomatic of garbage collection in word alignment). Because of this, we chose not to explore strategy 2 any further.

Under our perspective of zero-based corrections, we start with the assumption that feature-wise independence is good enough in most cases. We enforce this by smoothing toward zero. In cases where the data motivates feature-wise dependence, we further assume that these corrections will behave similarly for neighboring features. For example, we expect that the features `SrcLen2_TgtLen2`, `SrcLen2_TgtLen3`, and `SrcLen3_TgtLen2` should all behave similarly while `SrcLen1_TgtLen5` would behave similarly to `SrcLen1_TgtLen4`.

In this chapter, we consider strategy 1 in a technique we call tensor neighbor regularization (Section 7.3.1) and strategy 3 using complexity regularization in Section 7.3.2. We also use strategy 4 in our experiments by directly using the OSCAR regularizer described in Chapter 5.

7.3.1 Tensor Neighbor Regularization

Recall that monotone neighbor regularization encourages each feature's weight to fall between the weights of its immediate neighbors:

$$\mathcal{R}_{\text{MNR}}(\mathbf{w};\mathcal{N}) \triangleq \beta \sum_{\langle l,i,r\rangle\in\mathcal{N}} \mathcal{D}(\mathbf{w},l,i,r)\mathcal{R}_{\text{LNR}}(\mathbf{w},l,i,r)$$

In cases where w_i falls between its left and right neighbors w_l and w_r, the penalty will be nearly zero. Because we now have tensor-structured features, we generalize from the

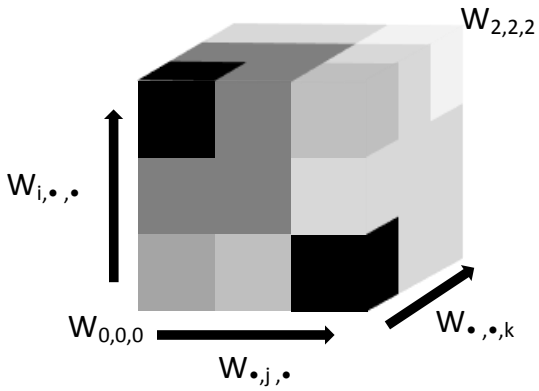


Figure 7.1: Visualization of conjunction of 3 discretized features. We use darker colors to indicate higher weights. Monotone tensor neighbor regularization will not penalize the gradual weight decay originating in the upper left corner. However, the high weight in the bottom right corner would be strongly penalized since it is very different from its neighbors and would only be learned if it was strongly supported by the data.

concept of left and right neighbors to any direct neighbors, by building on the neighbors from the discretization of individual features $\mathcal{N}^{\text{Disc}}$. Without loss of generality, the neighbors for order 2 conjunctions of discretized features $\mathcal{N}^{\text{DiscConj}}$ are formed by visiting each conjoined feature and produces a tuple for each direction where neighbors exist as shown in Figure 7.2:

$$\mathcal{N}^{\text{DiscConj}}(\mathbf{h}) \triangleq \bigcup_{\langle a_l, a_k, a_r \rangle \in \mathcal{N}^{\text{Disc}}} \bigcup_{\langle b_l, b_k, b_r \rangle \in \mathcal{N}^{\text{Disc}}} \left\{ \left\langle a_l \frown b_k, a_k \frown b_k, a_k \frown b_r \right\rangle, \left\langle a_k \frown b_l, a_k \frown b_k, a_k \frown b_r \right\rangle \right\}$$

7.3.2 Complexity Regularization

Just as we did in Section 4.3, we prefer to give higher weight to simpler features since they will have more data to estimate them and are more likely to generalize well to unseen data. This is also consistent with our view of the conjunctions of discretized features as corrections to be made on top of the initial feature set — in situations where the initial feature set is good enough, these features should be assigned zero weight. Given a function $\textsc{Order}(h_i)$, which returns the number of initial features used in a conjoined feature, we use

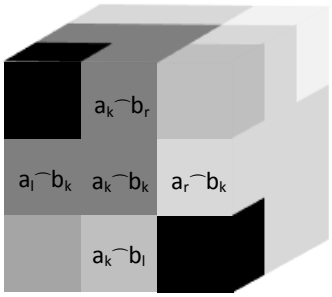


Figure 7.2: Visualization of how the set of neighbors $\mathcal{N}$ is computed in Equation 7.5 for tensor neighbor regularization. For conjunctions of order 2 (as shown on the face of this tensor), each feature will have 4 neighbors, represented by two 3-tuples. For conjunctions of order 3, each feature would have 6 neighbors, represented by three tuples.

a conjunction complexity regularizer with hyperparameter β_C:

$$\mathcal{R}_{\text{complexity}} \triangleq \beta_C \sum_{i=0}^{|\mathbf{h}|} [\![\textsc{Order}(h_i) > 1]\!] \cdot w_i^2 \tag{7.5}$$

7.4 Experimental Setup

Since the experiments in this chapter depend strongly on discretization, we use an identical experimental setup to Chapter 6. Our formalism is 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. 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. For decoding, we use cdec (Dyer et al., 2010), a multi-pass decoder that supports syntactic translation models and sparse features. Optimization is performed using PRO (Hopkins and May, 2011) as implemented by the cdec decoder, with the addition of the regularizers and feature induction techniques described in this chapter. We run PRO for 30 iterations as suggested by Hopkins and May (2011).

The PRO optimizer internally uses a L-BFGS optimizer with the default ℓ_2 regularization implemented in cdec. Any additional regularization is explicitly noted.

We use the baseline features produced by Lopez' suffix array grammar extractor (Lopez, 2008a), which is distributed with cdec. Note that these features may be simplified or removed as specified in each experimental condition. These features are:

- $\log P_{\text{coherent}}(e|f)$: The coherent phrase-to-phrase translation probability (Lopez, 2008b).
- $\log P_{\text{lex}}(e|f)$, $\log P_{\text{lex}}(f|e)$: 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 each translation rule
- $c(\texttt{glue})$: 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)

We also add source features that are useful only when conjunction is applied:

- $c(f)$: The count of source words (terminals) in each translation rule

For the experiments in this chapter, we use the discretized real-valued features from the previous chapter as our baseline. This gives us the following intermediate features when applying conjunction on top of discretization:

- discretized $P_{\text{coherent}}(e|f)$: The coherent phrase-to-phrase translation probability (Lopez, 2008b).
- discretized $P_{\text{lex}}(e|f)$, discretized $P_{\text{lex}}(f|e)$: The lexical alignment probabilities within each translation rule, as computed by a maximum likelihood estimation over the Viterbi alignments
- discretized $c(f)$: The count of source words (terminals) in each translation rule
- discretized $c(e)$: The count of target words (terminals) in each translation rule
- discretized $c(\texttt{glue})$: The count of glue rules used in the derivation

- discretized $c(\texttt{OOV}_{\text{TM}})$: The count of source tokens that were not recognized by the translation model (out of vocabulary) and were therefore passed through
- discretized $c(\texttt{OOV}_{\text{LM}})$: The count of target tokens that were not recognized by the language model (out of vocabulary)
- $\log P_{\text{LM}}(e)$: The log probability of the target translation hypothesis under a language model

All of bins from all of the features (except the language model) are then conjoined with each other as formally described in equation 7.4.

We quantify changes in translation quality using case-insensitive BLEU (Papineni et al., 2002). We control for test set variation and optimizer instability by averaging over multiple optimizer replicas (Clark et al., 2011).[2]

7.5 Results

We present the results of our experiments in Table 7.1. The first row is a standard system with log transformed features while the second row uses discretized probabilities with monotone neighbor regularization exactly as described in Chapter 6. We compare against the stronger discretized system since this chapter builds on discretization. Unfortunately, we did not observe any improvements over discretization in isolation. To mitigate potential local optima issues, we tried initializing PRO with best weights for the discretized system, but did not see any positive effects. We also experimented with applying OSCAR on top of tensor neighbor regularization to reduce the overall number of free parameters, but did not observe any improvements over discretization in isolation.

To further analyze the effects of conjunctions on top of discretization, we compare the BLEU score of the systems during tuning (Table 7.2) to testing (Table 7.1). Despite observing no improvements on the test set, we do observe improvements on the tuning set. When we look into the relative feature weights in the Arabic→English system, we find that aside from the language mode, the most highly weighted features are conjunctions of the target word count and the lexical translation probability. For example, TgtCount1_LexFGivenE0.5 receives a positive weight of 0.07 while TgtCount4_LexFGivenE0.5 receives a negative weight of -0.11, which we might interpret as meaning we should give a boost to single word translations when they have a very good inverse lexical translation probability. On the other hand, we see a negative weight of -0.06 for TgtCount1_LexFGivenE0.001 and a positive weight of 0.05 for TgtCount4_LexFGivenE0.001, which could be interpreted as

[2]MultEval 0.5.1: github.com/jhclark/multeval

Condition	Zh→En	Ar→En	Cz→En
log P	23.5	47.9	37.6
Disc P + MNR	23.8	48.7	37.6
Disc P + Conj + TNR	23.3* (-0.5)	48.6 (-0.1)	37.7 (+0.1)

Table 7.1: **Top:** Test set translation quality for systems with the typical log transform and for discretized probabilities **Bottom:** Test set translation quality for systems using conjunctions on discretized probabilities and tensor neighbor regularization.

Condition	Zh→En	Ar→En	Cz→En
log P	29.2	42.9	37.8
Disc P + MNR	29.8	42.9	38.3
Disc P + Conj + TNR	30.8 (+1.0)	45.5 (+2.6)	38.8 (+0.5)

Table 7.2: **Top:** *Tuning set* translation quality for systems with the typical log transform and for discretized probabilities **Bottom:** *Tuning set* translation quality for systems using conjunctions on discretized probabilities and tensor neighbor regularization.

penalizing short translations with a poor inverse lexical translation probability and giving a boost to longer translations, even if they have a poor inverse lexical translation probability. While these initially sound promising, further investigation reveals that these weights are strongly correlated with that of their components. For example, TgtCount1 already has a weight of -0.6, TgtCount4 has a weight of 0.03, good lexical probabilities are encouraged, and bad lexical probabilities are discouraged. It appears that we learn to more tightly fit local "bumps" in the conjoined feature space, but in a way that only helps the tuning set. After hyperparameter tuning, structured regularization is successful in preventing this overfitting from being overly harmful to test set generalization. However, we do not observe any inter-dependence among the discretized real-valued features in the standard MT feature set.

7.6 Related Work and Techniques

Toutanova and Ahn (2013) used gradient-boosting machines, a form of regression decision trees, to induce locally non-linear features. They apply these features in a n-best reranking framework. As a form of decision trees, these models would have also been able to accomplish both discretization and conjunction, though do not easily allow integration of structured regularization.

We also provide some concrete examples showing how other well-known non-linear learners might encode the source-to-target length balancing phenomena that we have used as a running example throughout this chapter. Figure 7.3 shows a neural network formulation in which the first hidden layer approximates discretization and the second hidden layer

performs an XOR operator much like a conjunction. Similarly, figure 7.4 shows a decision tree formulation. Notice that neither of these models allow for the same straightforward application of structured regularization nor a straightforward path of joint optimization alongside the other components of the MT system. See section 6.7 for a more in-depth discussion of the trade-offs between these models and discretization with structured regularization.

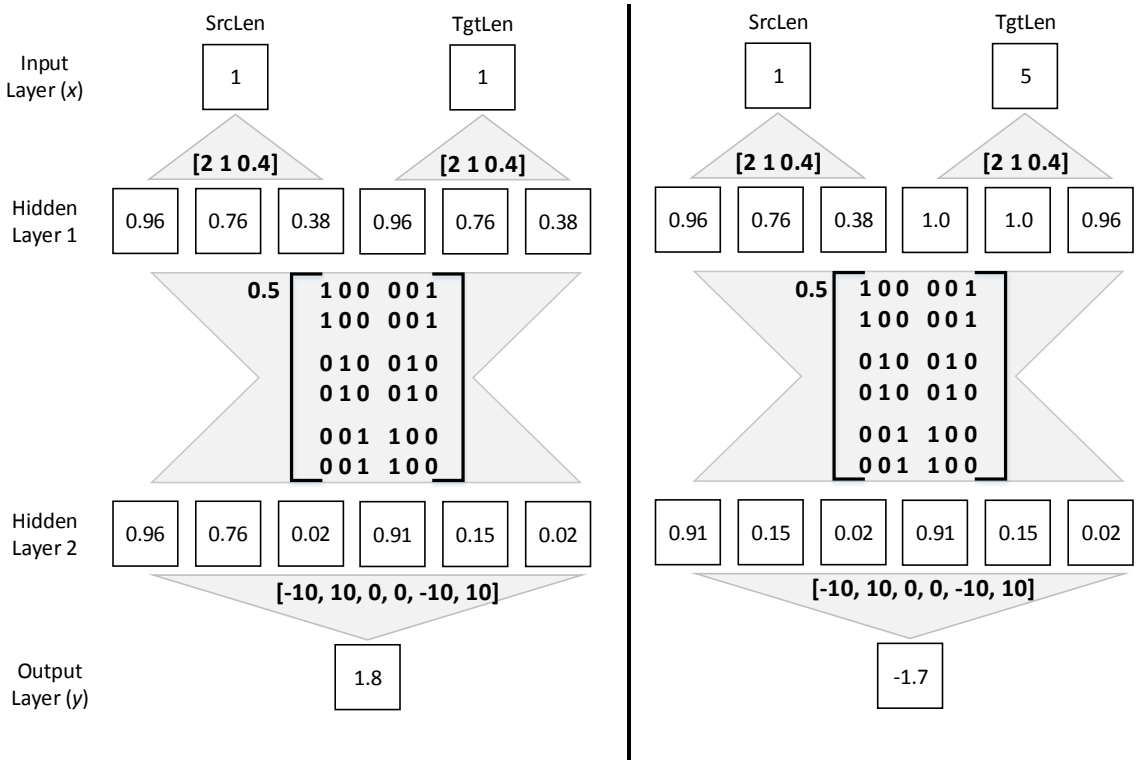


Figure 7.3: A neural network design that attempts to balance source length with target length. The network has two real-valued input neurons (top), a first tanh hidden layer imitating **discretization** with 3 neurons per input neuron, a second tanh hidden layer imitating the XOR function as **conjunctions** could do, and a single softmax real-valued output (bottom). The projection matrix for each layer is shown in bold above each layer. The network is instantiated twice with different input values: a well-matched source-to-target length that scores well (left) and a poorly matched source-to-target length that scores poorly (right). See section 6.7.1 for a full discussion.

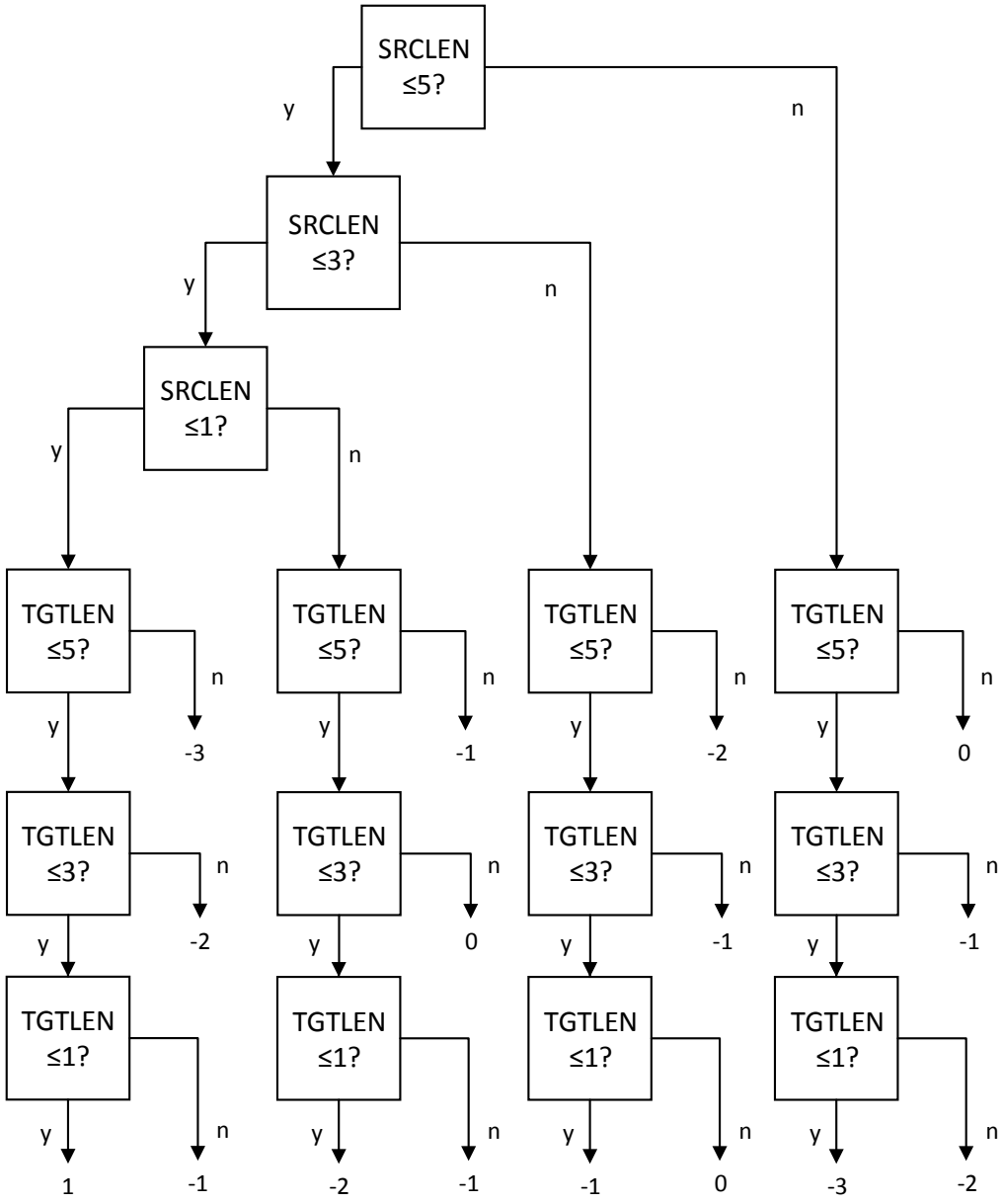


Figure 7.4: A decision tree, showing an attempt at balancing the source and target length. The top branches on `SrcLen` and each of the 4 columns that branch on `TgtLen` emulate discretization. Conjunction is emulated by duplicating the "discretized" `TgtLen` tree in sequence after the `SrcLen` decisions. See section 6.7.2 for a full discussion.

7.7 Chapter Summary

The combination of discretization and conjunction into a single feature induction operator allows us to relax the assumptions of linearity, monotonicity, and independently for initially continuous features. Tensor neighbor regularization provides a framework for coping with the resulting sparsity. While we still imagine that there may be feature sets for which this may be useful, we do not observe improvements beyond using discretization in isolation, suggesting that there may not be strong interdependence among the traditional real-valued features in MT.

8 Conclusions

There will come a time when you believe everything is finished. Yet that will be the beginning.

—Louis L'Amour

What you get by achieving your goals is not as important as what you become by achieving your goals.

—Henry David Thoreau

I hate quotations. Tell me what you know.

—Ralph Waldo Emerson

FEATURE INDUCTION when combined with appropriate structured regularization is an effective technique for improving the quality of a statistical machine translation system while simultaneously reducing sensitivity to the form of manually engineered features. In this thesis, we have shown that discretization and conjunctions are useful feature induction operators and we have observed gains on induced feature sets ranging from a few dozen features to nearly a million features. Due to strong regularization, none of these scenarios necessitated moving to a significantly larger set of tuning data.

We have argued that the premise of log-transformed features is now largely historical with only anecdotal empirical support, prior to this thesis (Chapter 1). We pointed out several axiomatic limitations of linear models and made the case for feature induction as a method for relaxing those assumptions while still preserving efficient inference (Chapter 2). In order to reliably evaluate our experiments that estimate large parameter sets on relatively small data, we proposed a technique for controlling for optimizer instability (Chapter 3), which has now been widely adopted by the machine translation community, including over 125 recent citations. Next, we explored conjunctions as a way of approaching domain adaptation in machine translation (Chapter 4) and as a means of tightly fitting

a single domain using nearly 1 million induced features (Chapter 5). We then explored discretization (Chapter 6), which enables local feature transformations during optimization. Finally, we combined these techniques using a regularizer that is aware of its neighbors in multi-dimensional space (Chapter 7).

We summarize the feature induction operator and regularization techniques used in each chapter in Figure 8.1. Similarly, we summarize the observed differences in translation quality in Figure 8.2.

To promote adoption in the research community, the code for all of these experiments is publicly released[1] including a stand-alone regularization library[2] and an evaluation library for multiple optimizer replicas.[3]

	Feature induction operators (Φ)	**Regularization** ($\mathcal{R}$)
Chapter 4	Conjunctions: InitiallyDiscrete $\times$ InitiallyContinuous	Complexity regularization
Chapter 5	Conjunctions: InitiallyDiscrete $\times$ InitiallyDiscrete	OSCAR
Chapter 6	Discretization: InitiallyContinuous	Linear & monotone neighbor regularization
Chapter 7	Conjunctions on discretization: InitiallyContinuous $\times$ InitiallyContinuous	Tensor regularization

Figure 8.1: A summary of the feature induction techniques and regularization techniques from each chapter.

	BLEU Change
Chapter 4	+1.0
Chapter 5	+1.7
Chapter 6	+0.8 vs baseline +4.4 vs no non-linear transform
Chapter 7	$\pm$

Figure 8.2: A summary of the quality improvements observed in each chapter.

[1] http://github.com/jhclark/cdec
[2] http://github.com/jhclark/prunejuice
[3] http://github.com/jhclark/multeval

8.1 Summary

We outlined three assumptions 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 divided features into two broad categories:

- **Initially discrete.** These indicator features may suffer from the independence assumption of linear models.

- **Initially continuous.** These features including probabilities and counts are subject to all three assumptions of linear models.

And we defined two classes of feature induction operators, which we applied to relax the assumptions of linear models:

- **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 all fire for a hypothesis, rather than simply returning the dot product of the individual features and their weights.

8.2 Contributions

This thesis makes the following contributions to the field of machine translation:

1. a method of feature conjunction to relax the property of independence between initially discrete and initially continuous features (Chapter 4);

2. a method of feature conjunction to relax the property of independence between initially discrete and initially discrete features (Chapter 5);

3. evidence 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 (Chapter 6);

4. a method of feature discretization of initially continuous features to relax the properties of linearity and monotonicity (Chapter 6);

5. a method of combining feature discretization and conjunction between initially continuous and initially continuous features to relax the properties of linearity, monotonicity, and independence (Chapter 7);

6. evidence that conjunctions of features can improve translation quality (Chapters 4 and 5); and

7. three real-world applications of these theoretical tools in machine translation (Chapters 4,5, and 6).

In order to achieve these, we also contribute:

6. a method of evaluating the impact of experimental results that is robust to optimizer instability such as randomized optimization procedures and experimental configurations having high variance due to large feature sets (Chapter 3);

7. a method of regularizing discretized features, allowing their weights to be reliably estimated (Chapter 6); and

8. a method of regularizing conjoined feature sets with intuitive groupings (Chapter 4).

9. a method of regularizing very large conjoined feature sets without *a priori* obvious groupings, allowing them to be used with conventionally-sized tuning sets (Chapter 5).

8.3 The Future of Feature Induction in Machine Translation

Looking beyond the scope of this work, we make connections to very recent work and offer some suggestions for future directions that may bring together several independent lines of work.

8.3.1 Neural networks

Near the completion of this thesis, several positive results have been published showing improvements due to the addition of neural network models (Devlin et al., 2014; Sutskever, Vinyals, and Le, 2014). These hidden layers of these networks are generally regarded as a type of **representation learning** or **feature learning** with an output layer that performs a softmax operation over the learned features (Bengio, Courville, and Vincent, 2013). In fact, standard machine translation models can be seen as a trivial neural network with no hidden layer where softmax serves as our decision rule. These recent methods have focused on representation learning on the training data and use millions of words to estimate their parameters. However, current models still remain slightly modified versions of standard neural networks. We speculate that making these models more aware of the structure of the machine translation problem including alignment, reordering, and hierarchical structure they will become even more useful tools. As these models evolve, the community will also have to reconcile how to maintain fast inference while models require increasingly more context. For example, recurrent neural networks theoretically have unbounded left state whereas current MT decoding algorithms assume bounded left state in order to provide fast inference.

Auto-encoder networks are another technique for representation learning that produce **embeddings** as output rather than a predictive model. For example, Zou et al. (2013) learned bilingual word embeddings for phrase-based machine translation and observed small improvements by exposing a single feature based on cosine similarity. Such instances where arbitrary feature transformations such as cosine similarity must be chosen are a good opportunity to applying discretization to achieve a better fit during the final optimization pass. We suggest that embeddings and the feature induction techniques proposed in this thesis may be combined to achieve an even greater impact.

However, neural networks may still benefit from manually engineered feature sets alongside learned representations (Mikolov and Zweig, 2012; Auli et al., 2013). We suggest that such strategies are also a good match for lower data scenarios where overfitting is a concern as it allows the neural network to shrink along with the data size.

Neural networks also suffer from highly non-convex objective functions, even when optimizing corpus likelihood. This is problematic to the degree that optimizers find poor local optima due to the bumpy nature of the objective function. We suggest that initializing neural networks with a structure similar to discretization or conjunction may be a useful method of avoiding particularly bad local optima.

Finally, neural networks may benefit from feature transformations when their input is real-valued. Despite being universal function approximators, a neural network may need to use a large number of parameters to perform certain non-linear transformations of real-valued input neurons. This provides a middle ground compared to directly adding manually engineered features to a neural network – initial features may be discretized and then fed into a neural network to learn a further improved representation. This also opens up the possibility of applying structured regularization such as monotone neighbor regularization directly to the parameters of a neural network.

8.3.2 Jointly Optimized Hybrid Models

Model design presents a basic tradeoff among model expressivity, data sparsity, and the human cost of embedding domain knowledge. Given larger data, more expressive models have the capacity to learn nuances of the problem. In such cases, far less domain knowledge may be required, as demonstrated by the recent successes of neural networks. When faced with smaller data, highly expressive models with too many parameters are prone to overfit. This requires more domain knowledge to be built into the model – this can be in the form of careful feature engineering (a common approach so far) or informative regularization (the approach advocated by this thesis).

Yet it is misleading to cast this as a strict dichotomy – often we have a large amount of out-of-domain data and a small amount of in-domain data. For years, the community has trained coarse features on the out-of-domain data and performed final tuning on the in-domain set. Throughout this thesis, we have focused on improving our ability to fit the in-domain data. This staged pipeline approach hides many aspects of the out-of-domain training data from the in-domain optimization phase. Other recent work has demonstrated improvements in optimization of large feature sets or neural networks directly on the training data. However, this approach ignores the utility of small in-domain data.

Future work will likely benefit from combining these approaches. Rather than a staged approach, this could take the form of joint optimization toward a combined objective function that considers both large out-of-domain data and small in-domain data. We envision a tuned hyperparameter that balances the contributions of the out-of-domain and in-domain data while automatically learning a feature representation that leverages both. Such a

model could include manually engineered features, features induced by discretization and conjunction, and fully learned representations as in a locally additive neural network or a source context neural network. Formally, we envision the neural network as a new feature induction operator Φ_{NN} with projection matrix $\mathbf{A}$ and bias vector $\mathbf{B}$:

$$\Phi_{\text{NN}}(\mathbf{h}) = \tanh(\mathbf{A}\mathbf{h} + \mathbf{B}) \tag{8.1}$$

which can be combined into a composite feature induction operator $\Phi^* = \{\Phi_{\text{Conj}}, \Phi_{\text{Disc}}, \Phi_{\text{NN}}\}$ as in equation 2.1:

$$\hat{\mathbf{e}}(\mathbf{f}) = \operatorname*{argmax}_{\mathbf{e},\mathbf{D}} \sum_{d \in D} \underbrace{\sum_{\langle y', j \rangle \in \Phi^*(\mathbf{h}(d))} w'_j y'}_{\mathbf{h}'(\mathbf{d})}$$

Inference remains straightforward. However, unlike the feature induction operators in the rest of this thesis, the derivatives for learning this model now depend on the identities of the individual rules in the derivations. Learning with this new feature induction operator would require storing these when sampling exemplars for PRO. Backpropagation would also be required to update the projection matrix $\mathbf{A}$. A small number of tuned hyperparameters would allow heavier regularization of the sparser induced features, allowing these hybrid models to be highly expressive while getting the most out of any amount of data.

8.4 Conclusions

We have discussed feature induction as a technique for reducing a system's sensitivity to an initial manually engineered feature set. We observed that it can improve translation quality when used with appropriate regularization in multiple scenarios. We hope that these techniques will enable systems to more fully exploit the information within data and features while reducing human effort.

List of Tables

List of Figures

List of Algorithms

References

Andrew, Galen and Jianfeng Gao. 2007. Scalable Training of L1-Regularized Log-Linear Models. In *International Conference on Machine Learning*.

Arun, Abhishek, Barry Haddow, Philipp Koehn, Adam Lopez, Chris Dyer, and Phil Blunsom. 2010. Monte Carlo techniques for phrase-based translation. *Machine Translation*, 24(2):103–121, June.

Auli, Michael, Michel Galley, and Jianfeng Gao. 2014. Large-scale Expected BLEU Training of Phrase-based Reordering Models. In *Empirical Methods in Natural Language Processing*.

Auli, Michael, Michel Galley, Chris Quirk, and Geoffrey Zweig. 2013. Joint Language and Translation Modeling with Recurrent Neural Networks. In *Empirical Methods in Natural Language Processing*, pages 1044–1054.

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

Banerjee, Pratyush, Jinhua Du, Baoli Li, Sudip Kr Naskar, Andy Way, and Josef Van Genabith. 2010. Combining Multi-Domain Statistical Machine Translation Models using Automatic Classifiers. In *Association for Machine Translation in the Americas*.

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*.

Barlow, R. E., D. Bartholomew, J. M. Bremner, and H. D. Brunk. 1972. *Statistical inference under order restrictions; the theory and application of isotonic regression*. Wiley.

Ben-David, Shai, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. 2010. A Theory of Learning from Different Domains. *Machine Learning*, 79(1-2):151–175, October.

Bengio, Yoshua, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis & Machine Intelligence,*, 35(8):1798–1828.

Bertoldi, Nicola and Marcello Federico. 2009. Domain Adaptation for Statistical Machine Translation with Monolingual Resources. In *Workshop on Statistical Machine Translation*, pages 182–189.

Bisani, M and H Ney. 2004. Bootstrap Estimates for Confidence Intervals in ASR Performance Evaluation. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 409–412.

Blitzer, John. 2007. *Domain Adaptation of Natural Language Processing Systems*. Ph.D. thesis, University of Pennsylvania.

Blitzer, John, Ryan McDonald, and Fernando Pereira. 2006. Domain Adaptation with Structural Correspondence Learning. In *Empirical Methods in Natural Language Processing*, pages 120–128, Sydney.

Bojar, Ondřej, Zdeněk Žabokrtský, Ondřej Dušek, Petra Galuščáková, Martin Majliš, David Mareček, Jiří Maršík, Michal Novák, Martin Popel, and Aleš Tamchyna. 2012. The Joy of Parallelism with CzEng 1 .0. In *Conference on Language Resources and Evaluation (LREC)*, Istanbul, Turkey. European Language Resources Association.

Bondell, Howard D and Brian J Reich. 2008. Simultaneous regression shrinkage variable selection and supervised clustering of predictors with OSCAR. *Biometrics*, 64(1):115–23, March.

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.

Cahill, Aoife, Michael Burke, Ruth O'Donovan, Stefan Riezler, Josef van Genabith, and Andy Way. 2008. Wide-Coverage Deep Statistical Parsing Using Automatic Dependency Structure Annotation. *Computational Linguistics*, 34(1):81–124, March.

Cer, Daniel, Daniel Jurafsky, and Christopher D. Manning. 2008. Regularization and search for minimum error rate training. In *Workshop on Statistical Machine Translation*, pages 26–34, Morristown, NJ, USA. Association for Computational Linguistics.

Cer, Daniel, Christopher D Manning, and Daniel Jurafsky. 2010. The Best Lexical Metric for Phrase-Based Statistical MT System Optimization. In *Proceedings of the North American Association for Computational Linguistics (NAACL)*, pages 555–563.

Chang, Yin-Wen and Michael Collins. 2011. Exact Decoding of Syntactic Translation Models through Lagrangian Relaxation. In *Empirical Methods in Natural Language Processing*, pages 72–82.

Charniak, Eugene. 2010. Top-Down Nearly-Context-Sensitive Parsing. In *Empirical Methods in Natural Language Processing*, 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.

Cherry, Colin and George Foster. 2012a. Batch Tuning Strategies for Statistical Machine Translation. In *Proceedings of the North American Association for Computational Linguistics*, pages 427–436.

Cherry, Colin and George Foster. 2012b. Batch Tuning Strategies for Statistical Machine Translation. In *Proceedings of the North American Association for Computational Linguistics*.

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

Chiang, David. 2012. Hope and fear for discriminative training of statistical translation models. *Journal of Machine Learning Research*, 13:1159–1187.

Chiang, David, Yuval Marton, and Philip Resnik. 2008. Online large-margin training of syntactic and structural translation features. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 224–233, Morristown, NJ, USA. Association for Computational Linguistics.

Chinchor, Nancy. 1993. The statistical significance of the MUC-5 results. *Conference on Message Understanding (MUC5)*, page 79.

Civera, Jorge and Alfons Juan. 2007. Domain Adaptation in Statistical Machine Translation with Mixture Modelling. In *Workshop on Statistical Machine Translation*, pages 177–180, Prague.

Clark, Jonathan H, Chris Dyer, and Alon Lavie. 2014. Locally Non-Linear Learning for Statistical Machine Translation via Discretization and Structured Regularization. *Transactions of the Association for Computational Linguistics*, 2:393–404.

Clark, Jonathan H, Chris Dyer, Alon Lavie, and Noah A Smith. 2011. Better Hypothesis Testing for Statistical Machine Translation: Controlling for Optimizer Instability. In *Association for Computational Linguistics*.

Clark, Jonathan H, Alon Lavie, and Chris Dyer. 2012. One System, Many Domains: Open-Domain Statistical Machine Translation via Feature Augmentation. *Association for Machine Translation in the Americas*.

Collins, Michael and Nigel Duffy. 2001. Parsing with a Single Neuron: Convulution Kernels for Natural Language Problems. In *Advances in Neural Information Processing Systems (NIPS)*.

Combettes, Patrick L. and Jean-Christophe Pesquet. 2010. Proximal Splitting Methods in Signal Processing. *arXiv*, December.

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

Crammer, Koby, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research*, 7:551–585.

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

Daumé III, Hal, Abhishek Kumar, and Avishek Saha. 2010. Co-regularization Based Semi-supervised Domain Adaptation. In *Advances in Neural Information Processing*, pages 1–9.

Daumé III, Hal and Daniel Marcu. 2006. Domain Adaptation for Statistical Classifiers. *Journal of Artificial Intelligence Research*, 26:101–126.

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*.

Devlin, Jacob, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. 2014. Fast and Robust Neural Network Joint Models for Statistical Machine Translation. In *Association for Computational Linguistics*, pages 1370–1380.

Donoho, David L. 2004. For Most Large Undetermined Systems of Linear Equations of Minimal L1 Norm Solution is Also the Sparsest Solution. Technical report, Stanford University.

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.

Dredze, Mark, John Blitzer, Partha Pratim Talukdar, Kuzman Ganchev, V João Graça, and Fernanda Pereira. 2007. Frustratingly Hard Domain Adaptation for Dependency Parsing. In *Proceedings of the Conference on Natural Language Learning (CoNLL)*.

Duchi, John and Yoram Singer. 2009. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10:2899–2934.

Dyer, Chris. 2009. Using a maximum entropy model to build segmentation lattices for MT. In *Proceedings of the North American Association for Computational Linguistics (NAACL)*, pages 406–414.

Dyer, Chris, Patrick Simianer, and Stefan Riezler. 2012. Joint Feature Selection in Distributed Stochastic Learning for Large-Scale Discriminative Training in SMT. In *Association for Computational Linguistics*, pages 11–21.

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*, pages 7–12.

Efron, B. 1979. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1–26.

Finley, Thomas and Thorsten Joachims. 2008. Training structural SVMs when exact inference is intractable. In *Proceedings of the International Conference on Machine Learning*, pages 304–311, New York, New York, USA. ACM Press.

Flanigan, Jeffrey, Chris Dyer, and Jaime Carbonell. 2013. Large-Scale Discriminative Training for Statistical Machine Translation Using Held-Out Line Search. In *North American Association for Computational Linguistics*, pages 248–258.

Foster, George, Cyril Goutte, and Roland Kuhn. 2010. Discriminative Instance Weighting for Domain Adaptation in Statistical Machine Translation. In *Empirical Methods in Natural Language Processing*, pages 451–459.

Foster, George and Roland Kuhn. 2007. Mixture-Model Adaptation for SMT. In *Proceedings of the Workshop on Statistical Machine Translation*.

Foster, George and Roland Kuhn. 2009. Stabilizing minimum error rate training. *orkshop on Statistical Machine Translation*, page 242.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2010. A note on the group lasso and a sparse group lasso. Technical report, Stanford University.

Galley, Michel, Chris Quirk, Colin Cherry, and Kristina Toutanova. 2013. Regularized Minimum Error Rate Training. In *Empirical Methods in Natural Language Processing*.

Ganitkevitch, Juri, Yuan Cao, Jonathan Weese, Matt Post, and Chris Callison-Burch. 2012. Joshua 4.0: Packing, PRO, and Paraphrases. In *Workshop on Statistical Machine Translation*, pages 283–291.

Ge, Dongdong, Xiaoye Jiang, and Yinyu Ye. 2011. A note on the complexity of Lp minimization. *Mathematical Programming*, 129(2):285–299, June.

Giménez, Jesús and Lluís Màrquez. 2007. Context-aware Discriminative Phrase Selection for Statistical Machine Translation. In *Workshop on Statistical Machine Translation*, 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*.

Gimpel, Kevin and Noah A Smith. 2012. Structured Ramp Loss Minimization for Machine Translation. In *North American Association for Computational Linguistics*.

Green, Spence, Sida Wang, Daniel Cer, and Christopher D Manning. 2013. Fast and Adaptive Online Training of Feature-Rich Translation Models. In *Association for Computational Linguistics*, pages 311–321.

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

He, Xiaodong and Li Deng. 2012. Maximum Expected BLEU Training of Phrase and Lexicon Translation Models. In *Proceedings of the Association for Computational Linguistics*, Jeju Island, Korea. Microsoft Research.

Ho, Tin Kam. 1995. Random Decision Forests. In *International Conference on Document Analysis and Recognition*, volume 47, pages 278–282, Montreal.

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

Huang, Fei and Alexander Yates. 2012. Biased Representation Learning for Domain Adaptation. In *Empirical Methods in Natural Language Processing*.

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

Hyder, Mashud and Kaushik Mahata. 2009. An Approximate L0 Norm Minimization Algorithm for Compressed Sensing. In *International Conference on Acoustics Speech and Signal Processing*, pages 3365–3368.

Isabelle, Pierre, Cyril Goutte, and Michel Simard. 2007. Domain adaptation of MT systems through automatic post-editing. In *Machine Translation Summit XI*, Copenhagen.

Jacob, Laurent, Guillaume Obozinski, and Jean-Philippe Vert. 2009. Group Lasso with Overlap and Graph Lasso. In *International Conference on Machine Learning*.

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 Association for Computing Machinery*, 52(11):97–104.

Kalchbrenner, Nal and Phil Blunsom. 2013. Recurrent Continuous Translation Models. In *Empirical Methods in Natural Language Processing*.

Kloft, Marius, Ulf Bredfeld, Soren Sonnenburg, Pavel Laskov, Klaus-Robert Muller, and Alexander Zien. 2009. Efficient and Accurate Lp-Norm Multiple Kernel Learning. In *Neural Information Processing Systems*, pages 1–9.

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

Koehn, Philipp, Alexandra Birch, Chris Callison-burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Christine Moran, Chris Dyer, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open Source Toolkit for Statistical Machine Translation. In *Association for Computational Linguistics*, pages 177–180.

Koehn, Philipp and Josh Schroeder. 2007. Experiments in Domain Adaptation for Statistical Machine Translation. In *Workshop on Statistical Machine Translation*, pages 224–227, Prague.

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

Kschischang, Frank R, Brendan J Frey, and Hans-Andrea Loeliger. 2001. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2).

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 *Association for Computational Linguistics and International Joint Conference on Natural Language Processing*, pages 163–171, Morristown, NJ, USA. Association for Computational Linguistics.

Liang, Percy, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar. 2006. An End-to-End Discriminative Approach to Machine Translation. *Computational Linguistics*, pages 761–768.

Liu, Lemao, Taro Watanabe, Eiichiro Sumita, and Tiejun Zhao. 2013a. Additive Neural Networks for Statistical Machine Translation. In *Association for Computational Linguistics*.

Liu, Lemao, Tiejun Zhao, Taro Watanabe, and Eiichiro Sumita. 2013b. Tuning SMT with A Large Number of Features via Online Feature Grouping. In *International Joint Conference on Natural Language Processing*.

Lopez, Adam. 2007. Hierarchical Phrase-Based Translation with Suffix Arrays. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Language Learning*, pages 976–985.

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

Lopez, Adam David. 2008b. *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.

Mairal, Julien, Rodolphe Jenatton, Guillaume Obozinski, and Francis Bach. 2010. Network Flow Algorithms for Structured Sparsity. In *Neural Information Processing Systems*, pages 1–9.

Martins, Andre F. T., Noah Smith, Pedro M. Q. Aguiar, and Mario A. T. Figueiredo. 2011. Structured Sparsity in Structured Prediction. In *Empirical Methods in Natural Language Processing*, pages 1500–1511.

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

Mikolov, Tomas and Geoffrey Zweig. 2012. Context dependent recurrent neural network language model. *2012 IEEE Spoken Language Technology Workshop (SLT)*, pages 234–239, December.

Mohimani, G Hosein, Massoud Babaie-zadeh, and Christian Jutten. 2007. Fast Sparse Representation Based on Smoothed L0 Norm. In *Independent Component Analysis and Signal Separation*, pages 0–8.

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

Moore, Robert C and Chris Quirk. 2008. Random Restarts in Minimum Error Rate Training for Statistical Machine Translation. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pages 585–592.

Nakov, Preslav, Francisco Guzmán, and Stephan Vogel. 2013. A Tale about PRO and Monsters.

Nelakanti, Anil, Cedric Archambeau, Julien Mairal, Francis Bach, and Guillaume Bouchard. 2013. Structured Penalties for Log-linear Language Models. In *Empirical Methods in Natural Language Processing*, Seattle, WA.

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

Noreen, Eric W. 1989. *Computer-Intensive Methods for Testing Hypotheses: An Introduction*. Wiley-Interscience, April.

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

Och, Franz Josef and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the Association for Computational Linguistics*, page 295, Morristown, NJ, USA. Association for Computational Linguistics.

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

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

Parikh, Neal and Stephen Boyd. 2013. Proximal Algorithms. In *Foundations and Trends in Optimization: Proximal Algorithms*. Stanford University, pages 123–231.

Park, Mee Young and Trevor Hastie. 2007. L1 Regularization Path Algorithm for Generalized Linear Models. *Journal of the Royal Statistical Society*, pages 1–25.

Phillips, Aaron B. 2012. *Modeling Relevance in Statistical Machine Translation: Scoring Alignment, Context, and Annotations of Translation Instances*. Ph.D. thesis, Carnegie Mellon University.

Quinlan, J R. 1996. Improved Use of Continuous Attributes in C4.5. *Journal of Artificial Intelligence Research*, pages 77–90.

Riezler, Stefan and John T Maxwell. 2005. On Some Pitfalls in Automatic Evaluation and Significance Testing for MT. In *Workshop on Intrinsic and Extrinsic Evaluation Methods for MT and Summarization at ACL*.

Robertson, Tim, F. T. Wright, and R. L. Dykstra. 1988. *Order Restricted Statistical Inference*. Wiley.

Rush, Alexander M, Yin-Wen Chang, and Michael Collins. 2013. Optimal Beam Search for Machine Translation. In *Empirical Methods in Natural Language Processing*, pages 210–221.

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

Scarselli, Franco and Ah Chung Tsoi. 1998. Universal Approximation Using Feedforward Neural Networks: A Survey of Some Existing Methods and Some New Results. *Neural Networks*, 11(1):15–37, January.

Schwenk, Holger. 2012. Continuous Space Translation Models for Phrase-Based Statistical Machine Translation. In *International Conference on Computational Linguistics (COLING)*, pages 1071–1080, Mumbai, India.

Shen, Libin, B C Va, and Marina Rey. 2004. Discriminative Reranking for Machine Translation. In *North American Association for Computational Linguistics*.

Silvapulle, Mervyn J. and Pranab K. Sen. 2005. *Constrained Statistical Inference: Order, Inequality, and Shape Constraints*. Wiley.

Simianer, Patrick, Katharina Wäschle, and Stefan Riezler. 2011. Multi-Task Minimum Error Rate Training for SMT. *Prague Bulletin of Mathematical Linguistics*, pages 99–108.

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

Snover, Matthew, Bonnie Dorr, College Park, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A Study of Translation Edit Rate with Targeted Human Annotation. In *Proceedings of Association for Machine Translation in the Americas*, pages 223–231.

Sokolov, Artem, Guillaume Wisniewski, and François Yvon. 2012. Non-linear N-best List Reranking with Few Features. In *Association for Machine Translation in the Americas*.

Sutskever, Ilya, Oriol Vinyals, and Quov V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Neural Information Processing Systems*, pages 1–9.

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

Suzuki, Jun and Masaaki Nagata. 2013. Supervised Model Learning with Feature Grouping based on a Discrete Constraint. In *Association for Computational Linguistics*, pages 18–23.

Taskar, Ben, Carlos Guestrin, and Daphne Koller. 2003. Max-Margin Markov Networks. In *Neural Information Processing Systems*.

Tibshirani, Robert. 1996. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society*, 58(1):267–288.

Tibshirani, Robert and Michael Saunders. 2005. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society*, pages 91–108.

Tibshirani, Ryan J, Holger Hoefling, and Robert Tibshirani. 2011. Nearly-Isotonic Regression. *Technometrics*, 53(1):54–61.

Toutanova, Kristina and Byung-Gyu Ahn. 2013. Learning Non-linear Features for Machine Translation Using Gradient Boosting Machines. In *Proceedings of the Association for Computational Linguistics*.

Trafimow, David and Michael Marks. 2015. Editorial. *Basic and Applied Social Psychology*, 37(1):1–2.

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)*.

Ueffing, Nicola, Haffari Gholamreza, and Anoop Sarkar. 2007. Transductive learning for statistical machine translation. In *Workshop on Statistical Machine Translation*.

Vaswani, Ashish, Liang Huang, and David Chiang. 2012. Smaller Alignment Models for Better Translations: Unsupervised Word Alignment with the L0-norm. In *Association for Computational Linguistics*.

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

Wasserman, Larry. 2003. *All of Statistics: A Concise Course in Statistical Inference (Springer Texts in Statistics)*. Springer, December.

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

Xu, Jia, Yonggang Deng, Yuqing Gao, and Hermann Ney. 2007. Domain Dependent Statistical Machine Translation. In *MT Summit*.

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

Yeh, Alexander. 2000. More accurate tests for the statistical significane of result differences. In *International Conference on Computational Linguistics (COLING)*, February.

Yogatama, Dani and Noah A Smith. 2014. Linguistic Structured Sparsity in Text Categorization. In *Association for Computational Linguistics*, pages 786–796.

Yu, Chun-Nam John and Thorsten Joachims. 2009. Learning structural SVMs with latent variables. In *nternational Conference on Machine Learning (ICML)*, pages 1–8, New York, New York, USA. ACM Press.

Yu, Dong and Li Deng. 2009. Solving Nonlinear Estimation Problems Using Splines. *IEEE Signal Processing*, 6(July):86–90.

Yu, Dong, Li Deng, and Alex Acero. 2008. The Maximum Entropy Model with Continuous Features. In *Neural Information Processing Systems*.

Yu, Dong, Li Deng, Yifan Gong, and Alex Acero. 2008. Discriminative Training of Variable-Parameter HMMs for Noise Robust Speech Recognition. In *Interspeech 2008*, pages 285–288.

Yu, Heng, Liang Huang, Hiatao Mi, and Kai Zhao. 2013. Max-Violation Perceptron and Forced Decoding for Scalable MT Training. In *Empirical Methods in Natural Language Processing*, pages 1112–1123.

Yuan, Ming and Yi Lin. 2006. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society*, pages 49–67.

Zeng, Xiangrong and Mario A. T. Figueiredo. 2014. Solving OSCAR regularization problems by proximal splitting algorithms. *Digital Signal Processing*, 31:124–135.

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

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.

Zhang, Ying and Stephan Vogel. 2010. Significance Tests of Automatic Machine Translation Evaluation Metrics. *Machine Translation*, 24(1):51–65.

Zhang, Ying, Stephan Vogel, and Alex Waibel. 2004. Interpreting BLEU / NIST Scores: How Much Improvement Do We Need to Have a Better System? In *Language Resources and Evaluation Conference*.

Zhong, Leon Wenliang and James T Kwok. 2011. Efficient Sparse Modeling with Automatic Feature Grouping. In *International Conference on Machine Learning*.

Zhong, Leon Wenliang and James T. Kwok. 2012. Efficient Sparse Modeling With Automatic Feature Grouping. *IEEE Transactions on Neural Networks and Learning Systems*, 23(9):1436–1447, September.

Zou, Hui and Trevor Hastie. 2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society*, 67(2):301–320, April.

Zou, Will Y, Richard Socher, Daniel Cer, and Christopher D Manning. 2013. Bilingual Word Embeddings for Phrase-Based Machine Translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)*.

Additional Notes on Neighbor Regularization

A.1 Derivation of $\nabla\mathcal{R}_{\text{MNR}}$

In this section, we provide the gradient of the monotone monotone neighbor regularizer $\mathcal{R}_{\text{MNR}}$, which is needed for implementing the technique in most optimizers. First, we duplicate the definitions of $\mathcal{R}_{\text{MNR}}$ (see p.81) then show its gradient.

We begin with the inner term of linear neighbor regularization, which is the basis for monotone neighbor regularization:

$$\mathcal{R}_{\text{LNR}}(\mathbf{w}, j) \triangleq (\mathcal{M}(\mathbf{w}, j))^2 \tag{A.1.1}$$

Next, we define the distance of the current weight w_j from the midpoint of its adjacent weights:

$$\mathcal{M}(\mathbf{w}, j) \triangleq \frac{1}{2}(w_{j-1} + w_{j+1}) - w_j \tag{A.1.2}$$

We also define the span covered by the adjacent weights as:

$$\mathcal{S}(\mathbf{w}, j) \triangleq \frac{1}{2}(w_{j-1} - w_{j+1}) + \epsilon \tag{A.1.3}$$

Note that we include a small positive quantity ϵ so that this quantity is guaranteed to be non-zero since it is frequently used as a denominator.

Using these two terms, we define the smooth damping factor of monotone neighbor regularization:

$$\mathcal{D}(\mathbf{w}, j) \triangleq \tanh^{2\gamma} \frac{\mathcal{M}(\mathbf{w}, j)}{\mathcal{S}(\mathbf{w}, j)} \tag{A.1.4}$$

Finally, we combine the damping factor with linear neighbor regularization to obtain the monotone neighbor regularizer:

$$\mathcal{R}_{\text{MNR}}(\mathbf{w}) \triangleq \beta \sum_{j=2}^{|\mathbf{h}|-1} \mathcal{D}(\mathbf{w}, j)\mathcal{R}_{\text{LNR}}(\mathbf{w}, j) \tag{A.1.5}$$

Prior to differentiating, we expand all terms that include w_j since it is non-constant in the context of this partial derivative:

$$\mathcal{R}_{\text{MNR}}(\mathbf{w}) = \beta \sum_{j=2}^{|\mathbf{h}|-1} \left[\tanh^{2\gamma} \ \frac{\frac{1}{2}(w_{j-1} + w_{j+1}) - w_j}{\frac{1}{2}(w_{j-1} - w_{j+1}) + \epsilon}\right] \left[\frac{1}{2}(w_{j-1} + w_{j+1}) - w_j\right]^2 \tag{A.1.6}$$

By applying standard differentiation rules, we obtain the partial derivative for each of the weights w_j, which together form the gradient for monotone neighbor regularization:

$$\begin{aligned}\frac{\partial\mathcal{R}_{\text{MNR}}(\mathbf{w})}{\partial w_j} =& -2\left[\tanh\left(\frac{\frac{1}{2}(w_{j+1} + w_{j-1}) - w_j}{\frac{1}{2}(w_{j+1} - w_{j-1}) + \epsilon}\right)\right] \cdot \left[\frac{1}{2}(w_{j-1} + w_{j+1}) - w_j\right] \\ &- \frac{2\gamma}{\frac{1}{2}(w_{j+1} - w_{j-1}) + \epsilon} \cdot \left[\operatorname{sech}\left(\frac{\frac{1}{2}(w_{j+1} + w_{j-1}) - w_j}{\frac{1}{2}\,w_{j+1} - w_{j-1} + \epsilon}\right)\right]^2 \\ &\cdot \left[\tanh\left(\frac{\frac{1}{2}(w_{j+1} + w_{j-1}) - w_j}{\frac{1}{2}\,w_{j+1} - w_{j-1} + \epsilon}\right)\right]^{2\gamma-1} \cdot \left[\frac{1}{2}(w_{j-1} + w_{j+1}) - w_j\right]^2\end{aligned} \tag{A.1.7}$$

Finally, we simplify using previously defined variables:

$$\begin{aligned}\frac{\partial\mathcal{R}_{\text{MNR}}(\mathbf{w})}{\partial w_j} =& -2\left[\tanh\left(\frac{\mathcal{M}(\mathbf{w}, j)}{\mathcal{S}(\mathbf{w}, j)}\right)\right] \cdot [\mathcal{M}(\mathbf{w}, j)] \\ &- \frac{2\gamma}{\mathcal{S}(\mathbf{w}, j)} \cdot \left[\operatorname{sech}\left(\frac{\mathcal{M}(\mathbf{w}, j)}{\mathcal{S}(\mathbf{w}, j)}\right)\right]^2 \\ &\cdot \left[\tanh\left(\frac{\mathcal{M}(\mathbf{w}, j)}{\mathcal{S}(\mathbf{w}, j)}\right)\right]^{2\gamma-1} \cdot [\mathcal{M}(\mathbf{w}, j)]^2\end{aligned} \tag{A.1.8}$$

In practice, floating point arithmetic can cause $\frac{\mathcal{M}(\mathbf{w},j)}{\mathcal{S}(\mathbf{w},j)}$ to be rounded to zero, which causes $\tanh\left(\frac{\mathcal{M}(\mathbf{w},j)}{\mathcal{S}(\mathbf{w},j)}\right)$ to be undefined. In such cases, we define $\frac{\partial\mathcal{R}_{\text{MNR}}(\mathbf{w})}{\partial w_j}$ to be zero. For example, a weight may differ only slightly from the midpoint of its neighbors and when divided by a reasonably large span will round to zero. In this case, we intuitively do not want to penalize this w_j since it is already close to a desirable value.

B Notation

B.1 Evaluation and Hypothesis Testing

s — Standard deviation (p. 26)

s_{dev} — Standard deviation over optimizer replicas on the development set (p. 26)

s_{test} — Standard deviation over optimizer replicas on the test set (p. 27)

$\overline{s_{\text{sel}}}$ — Average over optimizer replicas of the standard deviation according to bootstrap resampling of the hypotheses within each optimizer replica (p. 27)

n — Number of optimizer replicas (p. 26)

m_i — Translation quality measurement of the i^{th} optimization run (p. 26)

ℓ — In the context of evaluation, the number of segments in the test set (p. 28). See also ℓ_p regularization in the context of optimization.

$\mathbf{R}$ — In the context of evaluation, the references for each segment in the test set (p. 28)

$\mathcal{X}$ — In the context of evaluation, the test set translations over all optimizer replicas (p. 28)

p — In the context of hypothesis testing, the p-value or $p(\textsc{Chance})$, the probability that the magnitude of a difference is due to chance (p. 30)

B.2 Features and Modeling

$\mathbf{w}$ — The weight vector isomorphic to a feature set. (p. 8)

$\mathbf{f}$ — A source sentence, a vector of word tokens. (p. 8). f is used by a convention following early work in MT in which the source language was French.

f — A particular source word token. (p. 8)

$\mathbf{e}$ — A target sentence, a vector of word tokens. (p. 8). e is used by a convention following early work in MT in which the target language was English.

e — A particular target word token (p. 8)

D — A complete derivation of some hypothesis. (p. 8)

d — A partial derivation of some hypothesis. (p. 8)

$\hat{\mathbf{e}}(\mathbf{f})$ — The best-scoring hypothesis for some source sentence. (p. 8)

$H(D)$ or H — The global feature vector associated with some complete derivation. (p. 8)

$h(d)$ or h — The local feature vector associated with some partial derivation. (p. 8)

h' — A new feature set resulting from feature induction on h. (p. 16)

w' — A set of weights learned for the feature set h'. (p. 16)

$\Phi(y)$ — A feature transformation function that takes the result of the feature function $y = h(x) \in \mathbb{R}$ and returns a tuple $\langle y', j \rangle$ (p. 16)

ℓ_p — Specifies the norm of a regularizer with regard to the ℓ_p ball. (p. 10)

$\textsc{Bin}_i(x)$ — A binning function that takes a feature value $x \in \mathbb{R}$ for a specific feature with index i and returns a bin index. (p. 70)

$\mathcal{X}$ — A list of feature values to bin. (p. 78)

f_i — A frequency associated with some feature value with index i inside the list of observed feature values $\mathcal{X}$ (p. 78)

u' — The uniform bin size where u' is the number of feature values that will reside inside each bin if the bins are to have (nearly) equal size. (p. 78)

$\overline{\overline{x}}$ — The cardinality (number of items in the set) for some set x. We use this notation to distinguish from absolute value. (p. 78)

$|x|$ — The absolute value of the scalar x. (p. 78)

B — A binning for a particular feature that is a total mapping on ranges of a real-valued feature to bin indices. (p. 78)

N — The desired number of bins for each feature. (p. 78)

$\mathcal{R}_2$ — An ℓ_2 regularization term. (p. 81)

$\mathcal{R}_{\text{LNR}}$ — A linear neighbor regularization term. (p. 81)

$\mathcal{R}_{\text{MNR}}$ — A monotone neighbor regularization term. (p. 81)

$\mathcal{D}$ — A smooth damping term used within monotone neighbor regularization. (p. 81)

β — A regularization constant / regularization weight. (p. 81)

γ — A steepness for the bathtub curve in the dampening term for monotone neighbor regularization. (p. 81)

$\mathcal{L}$ — A loss function. (p. 81)

$c(x)$ — A count of some property of a translation rule, used for computing features. (p. 59)

A — An attribute in a decision tree. (p. 90)

B.3 Figure Notation

In the context of visuzlizations in figures...

$\oplus$ — Indicates that the pair of hypotheses is ranked correctly according to the extrinsic metric. (p. 73)

$\ominus$ — Indicates that the pair of hypotheses is ranked incorrectly according to the extrinsic metric. (p. 73)

C Terminology and Abbreviations

C.1 Generic

MACHINE TRANSLATION (MT)

STATISTICAL MACHINE TRANSLATION (SMT)

BILINGUAL EVALUATION UNDERSTUDY (BLEU): A commonly-used quality evaluation metric for MT.

METRIC FOR EVALUATION OF TRANSLATION WITH EXPLICIT ORDERING (METEOR): A commonly-used quality evaluation metric for MT.

TRANSLATION ERROR RATE (TER): A commonly-used quality evaluation metric for MT.

MINIMUM ERROR RATE TRAINING (MERT): An optimization algorithm for SMT systems.

BASIC TRAVEL EXPRESSION CORPUS (BTEC): One of the data sets used in our experiments

WORKSHOP ON MACHINE TRANSLATION (WMT): A yearly evaluation workshop that releases data used in our experiments

C.2 Evaluation and Hypothesis Testing

APPROXIMATE RANDOMIZATION (AR) [p. 30]

CONFIDENCE INTERVAL (CI) [p. 31]