Structured Ramp Loss Minimization for Machine Translation

Kevin Gimpel and **Noah A. Smith**
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{kgimpel,nasmith}@cs.cmu.edu

Abstract

This paper seeks to close the gap between training algorithms used in statistical machine translation and machine learning, specifically the framework of empirical risk minimization. We review well-known algorithms, arguing that they do not optimize the loss functions they are assumed to optimize when applied to machine translation. Instead, most have implicit connections to particular forms of **ramp loss**. We propose to minimize ramp loss directly and present a training algorithm that is easy to implement and that performs comparably to others. Most notably, our structured ramp loss minimization algorithm, RAMPION, is less sensitive to initialization and random seeds than standard approaches.

1 Introduction

Every statistical MT system relies on a training algorithm to fit the parameters of a scoring function to examples from parallel text. Well-known examples include MERT (Och, 2003), MIRA (Chiang et al., 2008), and PRO (Hopkins and May, 2011). While such procedures can be analyzed as machine learning algorithms—e.g., in the general framework of **empirical risk minimization** (Vapnik, 1998)—their *procedural* specifications have made this difficult. From a practical perspective, such algorithms are often complex, difficult to replicate, and sensitive to initialization, random seeds, and other hyperparameters.

In this paper, we consider training algorithms that are first specified *declaratively*, as **loss functions** to be minimized. We relate well-known training algorithms for MT to particular loss functions. We show that a family of **structured ramp** loss functions (Do et al., 2008) is useful for this analysis. For example, McAllester and Keshet (2011) recently suggested that, while Chiang et al. (2008, 2009) described their algorithm as "MIRA" (Crammer et al., 2006), in fact it targets a kind of ramp loss. We note here other examples: Liang et al. (2006) described their algorithm as a variant of the perceptron (Collins, 2002), which has a unique loss function, but the loss actually optimized is closer to a particular ramp loss (that differs from the one targeted by Chiang et al.). Och and Ney (2002) sought to optimize log loss (likelihood in a probabilistic model; Lafferty et al., 2001) but actually optimized a version of the *soft* ramp loss.

Why isn't the application of ML to MT more straightforward? We note two key reasons: (i) ML generally assumes that the correct output can always be scored by a model, but in MT the reference translation is often unreachable, due to a model's limited expressive power or search error, requiring the use of "surrogate" references; (ii) MT models nearly always include latent derivation variables, leading to non-convex losses that have generally received little attention in ML. In this paper, we discuss how these two have caused a disconnect between the loss function minimized by an algorithm in ML and the loss minimized when it is adapted for MT.

From a practical perspective, our framework leads to a simple training algorithm for structured ramp loss based on general optimization techniques. Our algorithm is simple to implement and, being a *batch* algorithm like MERT and PRO, can easily be inte-

grated with any decoder. Our experiments show that our algorithm, which we call RAMPION, performs comparably to MERT and PRO, is less sensitive to randomization and initialization conditions, and is robust in large-feature scenarios.

2 Notation and Background

Let $\mathcal{X}$ denote the set of all strings in a source language and, for a particular $\boldsymbol{x} \in \mathcal{X}$, let $\mathcal{Y}(\boldsymbol{x})$ denote the set of its possible translations (correct and incorrect) in the target language. In typical models for machine translation, a **hidden variable** is assumed to be constructed during the translation process.[1] Regardless of its specific form, we will refer to it as a **derivation** and denote it $\boldsymbol{h} \in \mathcal{H}(\boldsymbol{x})$, where $\mathcal{H}(\boldsymbol{x})$ is the set of possible values of $\boldsymbol{h}$ for the input $\boldsymbol{x}$. Derivations will always be coupled with translations and therefore we define the set $\mathcal{T}(\boldsymbol{x}) \subseteq \mathcal{Y}(\boldsymbol{x}) \times \mathcal{H}(\boldsymbol{x})$ of valid **output pairs** $\langle \boldsymbol{y}, \boldsymbol{h} \rangle$ for $\boldsymbol{x}$.

To model translation, we use a linear model parameterized by a parameter vector $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. Given a vector $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{h})$ of feature functions on $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{h}$, and assuming $\boldsymbol{\theta}$ contains a component for each feature function, output pairs $\langle \boldsymbol{y}, \boldsymbol{h} \rangle$ for a given input $\boldsymbol{x}$ are selected using a simple argmax decision rule: $\langle \boldsymbol{y}^*, \boldsymbol{h}^* \rangle = \operatorname{argmax}_{\langle \boldsymbol{y}, \boldsymbol{h} \rangle \in \mathcal{T}(\boldsymbol{x})} \underbrace{\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{h})}_{\text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{h}; \boldsymbol{\theta})}$.

The training problem for machine translation corresponds to choosing $\boldsymbol{\theta}$. There are many ways to do this, and we will describe each in terms of a particular **loss function** $\text{loss} : \mathcal{X}^N \times \mathcal{Y}^N \times \boldsymbol{\Theta} \to \mathbb{R}$ that maps an input corpus, its reference translations, and the model parameters to a real value indicating the quality of the parameters. **Risk minimization** corresponds to choosing

$$\operatorname{argmin}_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \; \mathbb{E}_{p(\boldsymbol{X}, \boldsymbol{Y})} \left[\text{loss} \left(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta} \right) \right] \qquad (1)$$

where $p(\boldsymbol{X}, \boldsymbol{Y})$ is the (unknown) true joint distribution over corpora. We note that the loss function depends on the entire corpus, while the decoder operates independently on one sentence at a time. This is done to fit the standard assumptions in MT systems: the evaluation metric (e.g., BLEU) depends on the entire corpus and does not decompose linearly, while the model score does. Since in practice we do not know $p(\boldsymbol{X}, \boldsymbol{Y})$, but we do have access to an actual corpus pair $\langle \tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}} \rangle$, where $\tilde{\boldsymbol{X}} = \{\boldsymbol{x}^{(i)}\}_{i=1}^N$ and $\tilde{\boldsymbol{Y}} = \{\boldsymbol{y}^{(i)}\}_{i=1}^N$, we instead consider **regularized empirical risk minimization**:

$$\operatorname{argmin}_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \; \text{loss}(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}}, \boldsymbol{\theta}) + R(\boldsymbol{\theta}) \qquad (2)$$

where $R(\boldsymbol{\theta})$ is the **regularization function** used to mitigate overfitting. The regularization function is frequently a squared norm of the parameter vector, such as the ℓ_1 or ℓ_2 norm, but many other choices are possible. In this paper, we use ℓ_2.

Models are evaluated using a task-specific notion of error, here encoded as a **cost function**, $\text{cost} : \mathcal{Y}^N \times \mathcal{Y}^N \to \mathbb{R}_{\geq 0}$, such that the worse a translation is, the higher its cost. The cost function will typically make use of an automatic evaluation metric for machine translation; e.g., cost might be 1 minus the BLEU score (Papineni et al., 2001).[2]

We note that our analysis in this paper is applicable for understanding the loss function being optimized *given a fixed set of k-best lists*.[3] However, most training procedures periodically invoke the decoder to generate new k-best lists, which are then typically merged with those from previous training iterations. It is an open question how this practice affects the loss function being optimized by the procedure as a whole.

Example 1: MERT. The most commonly-used training algorithm for machine translation is **minimum error rate training**, which seeks to directly minimize the cost of the predictions on the training data. This idea has been used in the pattern recognition and speech recognition communities (Duda and Hart, 1973; Juang et al., 1997); its first application to MT was by Och (2003). The loss function takes the following form: $\text{loss}_{\text{cost}} \left(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}}, \boldsymbol{\theta} \right) =$

$$\text{cost} \left(\tilde{\boldsymbol{Y}}, \left\{ \operatorname*{argmax}_{\langle \boldsymbol{y}, \boldsymbol{h} \rangle \in \mathcal{T}(\boldsymbol{x}^{(i)})} \text{score}(\boldsymbol{x}^{(i)}, \boldsymbol{y}, \boldsymbol{h}; \boldsymbol{\theta}) \right\}_{i=1}^N \right) \qquad (3)$$

[1] For phrase-based MT, a segmentation of the source and target sentences into phrases and an alignment between them (Koehn et al., 2003). For hierarchical phrase-based MT, a derivation under a synchronous CFG (Chiang, 2005).

[2] We will abuse notation and allow cost to operate on both sets of sentences as well as individual sentences. For notational convenience we also let cost accept hidden variables but assume that the hidden variables do not affect the value; i.e., $\text{cost}(\langle \boldsymbol{y}, \boldsymbol{h} \rangle, \langle \boldsymbol{y}', \boldsymbol{h}' \rangle) = \text{cost}(\boldsymbol{y}, \langle \boldsymbol{y}', \boldsymbol{h}' \rangle) = \text{cost}(\boldsymbol{y}, \boldsymbol{y}')$.

[3] Cherry and Foster (2012) have concurrently performed a similar analysis.

MERT directly minimizes the corpus-level cost function of the best outputs from the decoder without any regularization (i.e., $R(\boldsymbol{\theta}) = 0$).[4] The loss is non-convex and not differentiable for cost functions like BLEU, so Och (2003) developed a coordinate ascent procedure with a specialized line search.

MERT avoids the need to compute feature vectors for the references (§1(i)) and allows corpus-level metrics like BLEU to be easily incorporated. However, the complexity of the loss and the difficulty of the search lead to instabilities during learning. Remedies have been suggested, typically involving additional search directions and experiment replicates (Cer et al., 2008; Moore and Quirk, 2008; Foster and Kuhn, 2009; Clark et al., 2011). But despite these improvements, MERT is ineffectual for training weights for large numbers of features; in addition to anecdotal evidence from the MT community, Hopkins and May (2011) illustrated with synthetic data experiments that MERT struggles increasingly to find the optimal solution as the number of parameters grows.

Example 2: Probabilistic Models. By exponentiating and normalizing $\text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{h}; \boldsymbol{\theta})$, we obtain a conditional log-linear model, which is useful for training criteria with probabilistic interpretations:

$$p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{h} | \boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x}, \boldsymbol{\theta})} \exp\{\text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{h}; \boldsymbol{\theta})\} \quad (4)$$

The **log loss** then defines $\text{loss}_{\log}(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}}, \boldsymbol{\theta}) = -\sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(i)} \mid \boldsymbol{x}^{(i)})$.

Example 3: Bayes Risk. The term "risk" as used above should not be confused with the **Bayes risk** framework, which uses a probability distribution (Eq. 4) and a cost function to define a loss:

$$\text{loss}_{\text{B_risk}} = \sum_{i=1}^{N} \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{h} | \boldsymbol{x}^{(i)})}[\text{cost}(\boldsymbol{y}^{(i)}, \boldsymbol{y})] \quad (5)$$

The use of this loss is often simply called "risk minimization" in the speech and MT communities. Bayes risk is non-convex, whether or not latent variables are present. Like MERT, it naturally avoids the need to compute features for $\boldsymbol{y}^{(i)}$ and uses a cost function, making it appealing for MT. Bayes risk minimization first appeared in the speech recognition community (Kaiser et al., 2000; Povey and Woodland, 2002) and more recently has been applied to MT (Smith and Eisner, 2006; Zens et al., 2007; Li and Eisner, 2009).

[4] However, Cer et al. (2008) and Macherey et al. (2008) achieved a sort of regularization by altering MERT's line search.

3 Training Methods for MT

In this section we consider other ML-inspired approaches to MT training, situating each in the framework from §2: ramp, perceptron, hinge, and "soft" losses. Each of the first three kinds of losses can be understood as a way of selecting, for each $\boldsymbol{x}^{(i)}$, two candidate translation/derivation pairs: $\langle \boldsymbol{y}^{\uparrow}, \boldsymbol{h}^{\uparrow} \rangle$ and $\langle \boldsymbol{y}^{\downarrow}, \boldsymbol{h}^{\downarrow} \rangle$. During training, the loss function can be improved by increasing the score of the former and decreasing the score of the latter, through manipulation of the parameters $\boldsymbol{\theta}$. Figure 1 gives a general visualization of some of the key output pairs that are considered for these roles. Learning alters the score function, or, in the figure, moves points *horizontally* so that scores approximate negated costs.

3.1 Structured Ramp Loss Minimization

The **structured ramp loss** (Do et al., 2008) is a non-convex loss function with certain attractive theoretical properties. It is an upper bound on $\text{loss}_{\text{cost}}$ (Eq. 3) and is a tighter bound than other loss functions (Collobert et al., 2006). Ramp loss has been shown to be **statistically consistent** in the sense that, in the limit of infinite training data, minimizing structured ramp loss reaches the minimum value of $\text{loss}_{\text{cost}}$ that is achievable with a linear model (McAllester and Keshet, 2011). This is true whether or not latent variables are present.

Consistency in this sense is not a common property of loss functions; commonly-used convex loss functions such as the perceptron, hinge, and log losses (discussed below) are *not* consistent, because they are all sensitive to outliers or otherwise noisy training examples. Ramp loss is better at dealing with outliers in the training data (Collobert et al., 2006).

There are three forms of latent structured ramp loss: Eq. 6–8 (Fig. 2). Ramp losses are appealing for MT because they do not require computing the feature vector of $\boldsymbol{y}^{(i)}$ (§1(i)). The first form, Eq. 6, sets $\langle \boldsymbol{y}^{\uparrow}, \boldsymbol{h}^{\uparrow} \rangle$ to be the current model prediction ($\langle \hat{\boldsymbol{y}}, \hat{\boldsymbol{h}} \rangle$ in Fig. 1) and $\langle \boldsymbol{y}^{\downarrow}, \boldsymbol{h}^{\downarrow} \rangle$ to be an output that is both favored by the model *and* has high cost. Such an

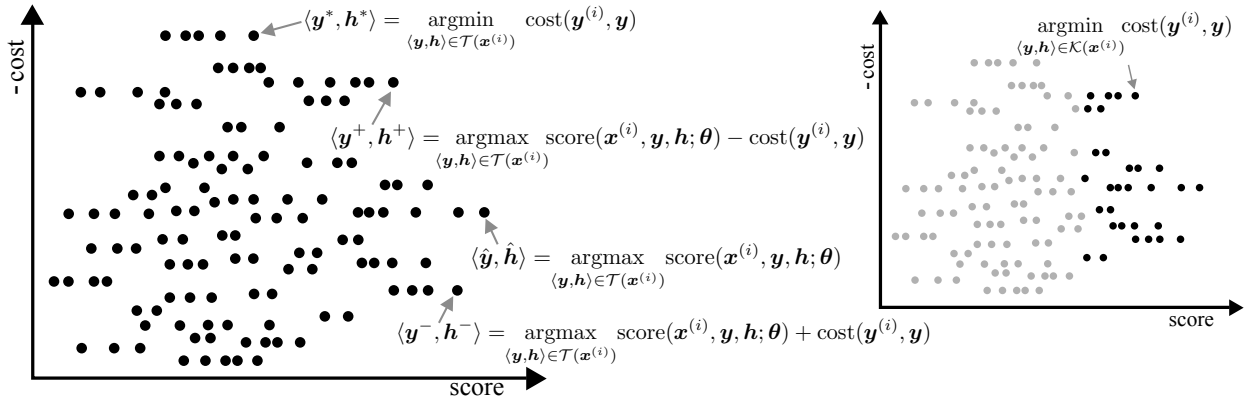


Figure 1: Hypothetical output space of a translation model for an input sentence $\boldsymbol{x}^{(i)}$. Each point corresponds to a single translation/derivation output pair. Horizontal "bands" are caused by output pairs with the same translation (and hence the same cost) but different derivations. The left plot shows the entire output space and the right plot highlights outputs in the k-best list. Choosing the output with the lowest cost in the k-best list is similar to finding $\langle \boldsymbol{y}^+, \boldsymbol{h}^+ \rangle$.

output is shown as $\langle \boldsymbol{y}^-, \boldsymbol{h}^- \rangle$ in Fig. 1; finding $\boldsymbol{y}^\downarrow$ is often called **cost-augmented decoding**, which is also used to define hinge loss (§3.3).

The second form, Eq. 7, *penalizes* the model prediction ($\langle \boldsymbol{y}^\downarrow, \boldsymbol{h}^\downarrow \rangle = \langle \hat{\boldsymbol{y}}, \hat{\boldsymbol{h}} \rangle$) and favors an output pair that has both high model score and low cost; this is the converse of cost-augmented decoding and therefore we call it **cost-*diminished* decoding**; $\langle \boldsymbol{y}^\uparrow, \boldsymbol{h}^\uparrow \rangle = \langle \boldsymbol{y}^+, \boldsymbol{h}^+ \rangle$ in Fig. 1. The third form, Eq. 8, sets $\langle \boldsymbol{y}^\uparrow, \boldsymbol{h}^\uparrow \rangle = \langle \boldsymbol{y}^+, \boldsymbol{h}^+ \rangle$ and $\langle \boldsymbol{y}^\downarrow, \boldsymbol{h}^\downarrow \rangle = \langle \boldsymbol{y}^-, \boldsymbol{h}^- \rangle$. This loss underlies RAMPION. It is similar to the loss optimized by the MIRA-inspired algorithm used by Chiang et al. (2008, 2009).

Optimization The ramp losses are continuous but non-convex and non-differentiable, so gradient-based optimization methods are not available.[5] Fortunately, Eq. 8 can be optimized by using a concave-convex procedure (CCCP; Yuille and Rangarajan, 2002). CCCP is a batch optimization algorithm for any function that is the the sum of a concave and a convex function. The idea is to approximate the sum as the convex term plus a tangent line to the concave function at the current parameter values; the resulting sum is convex and can be optimized with (sub)gradient methods.

With our loss functions, CCCP first imputes the outputs in the concave terms in each loss (i.e., solves the negated max expressions) for the entire training set and then uses an optimization procedure to optimize the loss with the imputed values fixed. Any convex optimization procedure can be used once the negated $\max$ terms are solved; we use stochastic subgradient descent (SSD) but MIRA could be easily used instead.

The CCCP algorithm we use for optimizing $\text{loss}_{\text{ramp}\,3}$, which we call RAMPION, is shown as Alg. 1. Similar algorithms can easily be derived for the other ramp losses. The first step done on each iteration is to generate k-best lists for the full tuning set (line 3). We then run CCCP on the k-best lists for T' iterations (lines 4–15). This involves first finding the translation to update towards for all sentences in the tuning set (lines 5–7), then making parameter updates in an online fashion with T'' epochs of stochastic subgradient descent (lines 8–14). The subgradient update for the ℓ_2 regularization term is done in line 11 and then for the loss in line 12.[6]

Unlike prior work that targeted similar loss functions (Watanabe et al., 2007; Chiang et al., 2008; Chiang et al., 2009), we do not use a fully online algorithm such as MIRA in an outer loop because we are not aware of an online learning algorithm with theoretical guarantees for non-differentiable, non-convex loss functions like the ramp losses. CCCP

[5] For non-differentiable, continuous, *convex* functions, **subgradient**-based methods are available, such as stochastic subgradient descent (SSD), and it is tempting to apply them here. However, non-convex functions are not everywhere subdifferentiable and so a straightforward application of SSD may encounter problems in practice.

[6] ℓ_2 regularization done here regularizes toward $\boldsymbol{\theta}_0$, not $\mathbf{0}$.

$$\text{loss}_{\text{ramp 1}} = \sum_{i=1}^{N} -\max_{\langle \boldsymbol{y},\boldsymbol{h}\rangle \in \mathcal{T}_i} (\text{score}_i(\boldsymbol{y},\boldsymbol{h};\boldsymbol{\theta})) + \max_{\langle \boldsymbol{y},\boldsymbol{h}\rangle \in \mathcal{T}_i} (\text{score}_i(\boldsymbol{y},\boldsymbol{h};\boldsymbol{\theta}) + \text{cost}_i(\boldsymbol{y})) \tag{6}$$

$$\text{loss}_{\text{ramp 2}} = \sum_{i=1}^{N} -\max_{\langle \boldsymbol{y},\boldsymbol{h}\rangle \in \mathcal{T}_i} (\text{score}_i(\boldsymbol{y},\boldsymbol{h};\boldsymbol{\theta}) - \text{cost}_i(\boldsymbol{y})) + \max_{\langle \boldsymbol{y},\boldsymbol{h}\rangle \in \mathcal{T}_i} (\text{score}_i(\boldsymbol{y},\boldsymbol{h};\boldsymbol{\theta})) \tag{7}$$

$$\text{loss}_{\text{ramp 3}} = \sum_{i=1}^{N} -\max_{\langle \boldsymbol{y},\boldsymbol{h}\rangle \in \mathcal{T}_i} (\text{score}_i(\boldsymbol{y},\boldsymbol{h};\boldsymbol{\theta}) - \text{cost}_i(\boldsymbol{y})) + \max_{\langle \boldsymbol{y},\boldsymbol{h}\rangle \in \mathcal{T}_i} (\text{score}_i(\boldsymbol{y},\boldsymbol{h};\boldsymbol{\theta}) + \text{cost}_i(\boldsymbol{y})) \tag{8}$$

$$\text{loss}_{\text{perc}} = \sum_{i=1}^{N} -\max_{\boldsymbol{h}:\langle \boldsymbol{y}^{(i)},\boldsymbol{h}\rangle \in \mathcal{T}_i} \text{score}_i(\boldsymbol{y}^{(i)},\boldsymbol{h};\boldsymbol{\theta}) + \max_{\langle \boldsymbol{y},\boldsymbol{h}\rangle \in \mathcal{T}_i} \text{score}_i(\boldsymbol{y},\boldsymbol{h};\boldsymbol{\theta}) \tag{9}$$

$$\text{loss}_{\text{perc } k\text{best}} = \sum_{i=1}^{n} -\text{score}\left(\boldsymbol{x}^{(i)}, \operatorname*{argmin}_{\langle \boldsymbol{y},\boldsymbol{h}\rangle \in \mathcal{K}_i} (\text{cost}_i(\boldsymbol{y}))\,; \boldsymbol{\theta}\right) + \max_{\langle \boldsymbol{y},\boldsymbol{h}\rangle \in \mathcal{T}_i} \text{score}_i(\boldsymbol{y},\boldsymbol{h};\boldsymbol{\theta}) \tag{10}$$

$$\approx \sum_{i=1}^{N} -\max_{\langle \boldsymbol{y},\boldsymbol{h}\rangle \in \mathcal{T}_i} (\text{score}_i(\boldsymbol{y},\boldsymbol{h};\boldsymbol{\theta}) - \gamma_i \text{cost}_i(\boldsymbol{y})) + \max_{\langle \boldsymbol{y},\boldsymbol{h}\rangle \in \mathcal{T}_i} \text{score}_i(\boldsymbol{y},\boldsymbol{h};\boldsymbol{\theta}) \tag{11}$$

Figure 2: Formulae mentioned in text for latent-variable loss functions. Each loss is actually a function $\text{loss}(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}}, \boldsymbol{\theta})$; we suppress the arguments for clarity. "$\mathcal{T}_i$" is shorthand for "$\mathcal{T}(\boldsymbol{x}^{(i)})$." "$\mathcal{K}_i$" is shorthand for the k-best list for $\boldsymbol{x}^{(i)}$. "$\text{cost}_i(\cdot)$" is shorthand for "$\text{cost}(\boldsymbol{y}^{(i)}, \cdot)$." "$\text{score}_i(\cdot)$" is shorthand for "$\text{score}(\boldsymbol{x}^{(i)}, \cdot)$." As noted in §3.4, any operator of the form $\max_{s\in\mathcal{S}}$ can be replaced by $\log \sum_{s\in\mathcal{S}} \exp$, known as softmax, giving many additional loss functions.

is fundamentally a batch optimization algorithm and has been used for solving many non-convex learning problems, such as latent structured SVMs (Yu and Joachims, 2009).

3.2 Structured Perceptron

The stuctured perceptron algorithm (Collins, 2002) was considered by Liang et al. (2006) as an alternative to MERT. It requires only a decoder and comes with some attractive guarantees, at least for models without latent variables. Liang et al. modified the perceptron in several ways for use in MT. The first was to generalize it to handle latent variables. The second change relates to the need to compute the feature vector for the reference translation $\boldsymbol{y}^{(i)}$, which may be unreachable (§1(i)). To address this, researchers have proposed the use of surrogates that are both favored by the current model parameters and similar to the reference. Och and Ney (2002) were the first to do so, using the translation on a k-best list with the highest evaluation metric score as $\boldsymbol{y}^{\uparrow}$. This practice was followed by Liang et al. (2006) and others with success (Arun and Koehn, 2007; Watanabe et al., 2007).[7]

Perceptron Loss Though typically described and analyzed procedurally, it is straightforward to show that Collins' perceptron (without latent variables) equates to SSD with fixed step size 1 on loss:

$$\sum_{i=1}^{N} -\text{score}(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}; \boldsymbol{\theta}) + \max_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x}^{(i)})} \text{score}(\boldsymbol{x}^{(i)}, \boldsymbol{y}; \boldsymbol{\theta}) \tag{12}$$

This loss is convex but ignores cost functions. In our notation, $\boldsymbol{y}^{\uparrow} = \boldsymbol{y}^{(i)}$ and $\boldsymbol{y}^{\downarrow} = \text{argmax}_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x}^{(i)})} \text{score}(\boldsymbol{x}^{(i)}, \boldsymbol{y}; \boldsymbol{\theta})$.

Adaptation for MT We chart the transformations from Eq. 12 toward the loss Liang et al.'s algorithm actually optimized. First, generalize to latent variables; see Eq. 9 (Fig. 2), sacrificing convexity. Second, to cope with unreachable references, use a k-best surrogate as shown in Eq. 10 (Fig. 2), where $\mathcal{K}_i \in \mathcal{T}(\boldsymbol{x}^{(i)})^k$ is a set containing the k best output pairs for $\boldsymbol{x}^{(i)}$. Now the loss only depends on $\boldsymbol{y}^{(i)}$ through the cost function. (Even without hidden variables, this loss can only be convex when the k-best list is fixed, keeping $\boldsymbol{y}^{\uparrow}$ unchanged across iterations. Updating the k-best lists makes $\boldsymbol{y}^{\uparrow}$ depend on $\boldsymbol{\theta}$, resulting in a non-convex loss.)

It appears that Eq. 10 (Fig. 2) is the loss that Liang et al. (2006) *sought* to optimize, using SSD. In light of footnote 5 and the non-convexity of Eq. 10 (Fig. 2), we have no theoretical guarantee that such an algorithm will find a (local) optimum.

[7] Liang et al. (2006) also tried a variant that updated directly to the reference when it is reachable ("bold updating"), but they and others found that Och and Ney's strategy worked better.

Input: inputs $\{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$, references $\{\boldsymbol{y}^{(i)}\}_{i=1}^{N}$, init. weights $\boldsymbol{\theta}_0$, k-best list size k, step size η, ℓ_2 reg. coeff. C, # iters T, # CCCP iters T', # SSD iters T''
Output: learned weights: $\boldsymbol{\theta}$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$;
for $iter \leftarrow 1$ **to** T **do**
  $\{\mathcal{K}_i\}_{i=1}^{N} \leftarrow \texttt{Decode}(\{\boldsymbol{x}^{(i)}\}_{i=1}^{N}, \boldsymbol{\theta}, k)$;
  for $iter' \leftarrow 1$ **to** T' **do**
    for $i \leftarrow 1$ **to** N **do**
      $\langle \boldsymbol{y}_i^+, \boldsymbol{h}_i^+ \rangle \leftarrow$ $\operatorname{argmax}_{\langle \boldsymbol{y}, \boldsymbol{h} \rangle \in \mathcal{K}_i} \operatorname{score}_i(\boldsymbol{y}, \boldsymbol{h}; \boldsymbol{\theta}) - \operatorname{cost}_i(\boldsymbol{y})$;
    end
    for $iter'' \leftarrow 1$ **to** T'' **do**
      for $i \leftarrow 1$ **to** N **do**
        $\langle \boldsymbol{y}^-, \boldsymbol{h}^- \rangle \leftarrow$ $\operatorname{argmax}_{\langle \boldsymbol{y}, \boldsymbol{h} \rangle \in \mathcal{K}_i} \operatorname{score}_i(\boldsymbol{y}, \boldsymbol{h}; \boldsymbol{\theta}) + \operatorname{cost}_i(\boldsymbol{y})$;
        $\boldsymbol{\theta} \mathrel{-}= \eta C \left(\frac{\boldsymbol{\theta} - \boldsymbol{\theta}_0}{N}\right)$;
        $\boldsymbol{\theta} \mathrel{+}= \eta\big(\boldsymbol{f}(\boldsymbol{x}^{(i)}, \boldsymbol{y}_i^+, \boldsymbol{h}_i^+) - \boldsymbol{f}(\boldsymbol{x}^{(i)}, \boldsymbol{y}^-, \boldsymbol{h}^-)\big)$;
      end
    end
  end
end
return $\boldsymbol{\theta}$;

Algorithm 1: RAMPION.

We note that Eq. 10 is similar to Eq. 11 (Fig. 2), where each γ is used to trade off between model and cost. Fig. 1 illustrates the similarity by showing that the min-cost output on a k-best list resides in a similar region of the output space as $\langle \boldsymbol{y}^+, \boldsymbol{h}^+ \rangle$ computed from the full output space. While it is not the case that we can always choose γ_i so as to make the two losses equivalent, they are similar in that they update towards some $\boldsymbol{y}^\uparrow$ with high model score and low cost. Eq. 11 corresponds to Eq. 7 (Fig. 2), the second form of the latent structured ramp loss.

Thus, one way to understand Liang et al.'s algorithm is as a form of structured ramp loss. However, another interpretation is given by McAllester et al. (2010), who showed that procedures like that used by Liang et al. approach direct cost minimization in the limiting case.

3.3 Large-Margin Methods

A related family of approaches for training MT models involves the **margin-infused relaxed algorithm** (MIRA; Crammer et al., 2006), an online large-margin training algorithm. It has recently shown success for MT, particularly when training models with large feature sets (Watanabe et al., 2007; Chiang et al., 2008; Chiang et al., 2009). In order to apply it to MT, Watanabe et al. and Chiang et al. made modifications similar to those made by Liang et al. for perceptron training, namely the extension to latent variables and the use of a surrogate reference with high model score and low cost.

Hinge Loss It can be shown that 1-best MIRA corresponds to dual coordinate ascent for the **structured hinge loss** when using ℓ_2 regularization (Martins et al., 2010). The structured hinge is the loss underlying maximum-margin Markov networks (Taskar et al., 2003): setting $\boldsymbol{y}^\uparrow = \boldsymbol{y}^{(i)}$ and:

$$\boldsymbol{y}^\downarrow = \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x}^{(i)})} \Big(\operatorname{score}(\boldsymbol{x}^{(i)}, \boldsymbol{y}; \boldsymbol{\theta}) + \operatorname{cost}(\boldsymbol{y}^{(i)}, \boldsymbol{y})\Big) \tag{13}$$

Unlike the perceptron losses, which penalize the highest-scoring outputs, hinge loss penalizes an output that is both favored by the model *and* has high cost. Such an output is shown as $\langle \boldsymbol{y}^-, \boldsymbol{h}^- \rangle$ in Fig. 1; the structured hinge loss focuses on pushing such outputs to the left. As mentioned in §3.1, finding $\boldsymbol{y}^\downarrow$ is often called **cost-augmented decoding**.

Structured hinge loss is convex, can incorporate a cost function, and can be optimized with several algorithms, including SSD (Ratliff et al., 2006).

Adaptation for MT While prior work has used MIRA-like algorithms for training machine translation systems, the proposed algorithms did not actually optimize the structured hinge loss, for similar reasons to those mentioned above for the perceptron: latent variables and surrogate references. Incorporating latent variables in the hinge loss results in the **latent structured hinge loss** (Yu and Joachims, 2009). Like the latent perceptron, this loss is non-convex and inappropriate for MT because it requires computing the feature vector for $\boldsymbol{y}^{(i)}$. By using a surrogate instead of $\boldsymbol{y}^{(i)}$, the actual loss optimized becomes closer to Eq. 8 (Fig. 2), the third form of the latent structured ramp loss.

Watanabe et al. (2007) and Arun and Koehn (2007) used k-best oracles like Liang et al., but Chiang et al. (2008, 2009) used a different approach, explicitly defining the surrogate as $\langle \boldsymbol{y}^+, \boldsymbol{h}^+ \rangle$ in Fig. 1. While the method of Chiang et al. showed impres-

sive performance improvements, its implementation is non-trivial, involving a complex cost function and a parallel architecture, and it has not yet been embraced by the MT community. Indeed, the complexity of Chiang et al's algorithm was one of the reasons cited for the development of PRO (Hopkins and May, 2011). In this paper, we have sought to isolate the loss functions used in prior work like that by Chiang et al. and identify simple, generic optimization procedures for optimizing them. We offer RAMPION as an alternative to Chiang et al's MIRA that is simpler to implement and achieves empirical success in experiments (§4).

3.4 Likelihood and Softened Losses

We can derive new loss functions from the above by converting any "max" operator to a "softmax" ($\log \sum \exp$, where the set of elements under the summation is the same as under the max). For example, the softmax version of the perceptron loss is the well-known **log loss** (§2, Ex. 2), the loss underlying the **conditional likelihood** training criterion which is frequently used when a probabilistic interpretation of the learned model is desired, as in conditional random fields (Lafferty et al., 2001).

Och and Ney (2002) popularized the use of log-linear models for MT and initially sought to optimize log loss, but by using the min-cost translation on a k-best list as their surrogate, we argue that their loss was closer to the **soft ramp loss** obtained by softening the second max in $\text{loss}_{\text{ramp }2}$ in Eq. 7 (Fig. 2). The same is true for others who aimed to optimize log loss for MT (Smith and Eisner, 2006; Zens et al., 2007; Cer, 2011).

The softmax version of the latent variable perceptron loss, Eq. 9 (Fig. 2), is the **latent log loss** inherent in latent-variable CRFs (Quattoni et al., 2004). Blunsom et al. (2008) and Blunsom and Osborne (2008) actually did optimize latent log loss for MT, discarding training examples for which $\boldsymbol{y}^{(i)}$ was unreachable by the model.

Finally, we note that "softening" the ramp loss in Eq. 6 (Fig. 2) results in the **Jensen risk bound** from Gimpel and Smith (2010), which is a computationally-attractive upper bound on the Bayes risk.

4 Experiments

The goal of our experiments is to compare RAMPION (Alg. 1) to state-of-the-art methods for training MT systems. RAMPION minimizes $\text{loss}_{\text{ramp }3}$, which we found in preliminary experiments to work better than other loss functions tested.[8]

System and Datasets We use the Moses phrase-based MT system (Koehn et al., 2007) and consider Urdu→English (UR→EN), Chinese→English (ZH→EN) translation, and Arabic→English (AR→EN) translation.[9] We trained a Moses system using default settings and features, except for setting the distortion limit to 10. Word alignment was performed using GIZA++ (Och and Ney, 2003) in both directions, the `grow-diag-final-and` heuristic was used to symmetrize the alignments, and a max phrase length of 7 was used for phrase extraction. We estimated 5-gram language models using the SRI toolkit (Stolcke, 2002) with modified Kneser-Ney smoothing (Chen and Goodman, 1998). For each language pair, we used the English side of the parallel text and 600M words of randomly-selected sentences from the Gigaword v4 corpus (excluding NYT and LAT).

For UR→EN, we used parallel data from the NIST MT08 evaluation consisting of 1.2M Urdu words and 1.1M English words. We used half of the documents (882 sentences) from the MT08 test set for tuning. We used the remaining half for one test set ("MT08*") and MT09 as our other test set. For ZH→EN, we used 303k sentence pairs from the FBIS corpus (LDC2003E14). We segmented the Chinese data using the Stanford Chinese segmenter (Chang et al., 2008) in "CTB" mode, giving us 7.9M Chinese words and 9.4M English words. We used MT03 for tuning and used MT02 and MT05 for testing.

For AR→EN, we used data provided by the LDC

[8] We only present full results using $\text{loss}_{\text{ramp }3}$. We found that minimizing $\text{loss}_{\text{ramp }1}$ did poorly, resulting in single-digit BLEU scores, and that $\text{loss}_{\text{ramp }2}$ reached high BLEU scores on the tuning data but failed to generalize well. Softened versions of the ramp losses performed comparably to $\text{loss}_{\text{ramp }3}$ but were slightly worse on both tuning and held-out data.

[9] We found similar trends for other language pairs and systems, including Hiero (Chiang, 2005). A forthcoming report will present these results, as well as experiments with additional loss functions, in detail.

for the NIST evaluations, including 3.29M sentence pairs of UN data and 982k sentence pairs of non-UN data. The Arabic data was preprocessed using an HMM segmenter that splits off attached prepositional phrases, personal pronouns, and the future marker (Lee et al., 2003). The common stylistic sentence-initial *wa#* (*and ...*) was removed from the training and test data. The resulting corpus contained 130M Arabic tokens and 130M English tokens. We used MT06 for tuning and three test sets: MT05, the MT08 newswire test set ("MT08 NW"), and the MT08 weblog test set ("MT08 WB").

For all languages we evaluated translation output using case-insensitive IBM BLEU (Papineni et al., 2001).

Training Algorithms Our baselines are MERT and PRO as implemented in the Moses toolkit.[10] PRO uses the hyperparameter settings from Hopkins and May (2011), including k-best lists of size 1500 and 25 training iterations.[11] MERT uses k-best lists of size 100 and was run to convergence. For both MERT and PRO, previous iterations' k-best lists were merged in.

For RAMPION, we used $T = 20$, $T' = 10$, $T'' = 5$, $k = 500$, $\eta = 0.0001$, and $C = 1$. Our cost function is $\alpha(1 - \text{BLEU}_{+1}(\boldsymbol{y}, \boldsymbol{y}'))$ where $\text{BLEU}_{+1}(\boldsymbol{y}, \boldsymbol{y}')$ returns the BLEU_{+1} score (Lin and Och, 2004) for reference $\boldsymbol{y}$ and hypothesis $\boldsymbol{y}'$. We used $\alpha = 10$. We used these same hyperparameter values for all experiments reported here and found them to perform well across other language pairs and systems.[12]

4.1 Results

Table 1 shows our results. MERT and PRO were run 3 times with differing random seeds and averages and standard deviations are shown. The three algorithms perform very similarly on the whole, with certain algorithms performing better on certain languages. MERT shows larger variation across random seeds, as reported by many others in the community. On average across all language pairs and test sets, RAMPION leads to slightly higher BLEU scores.

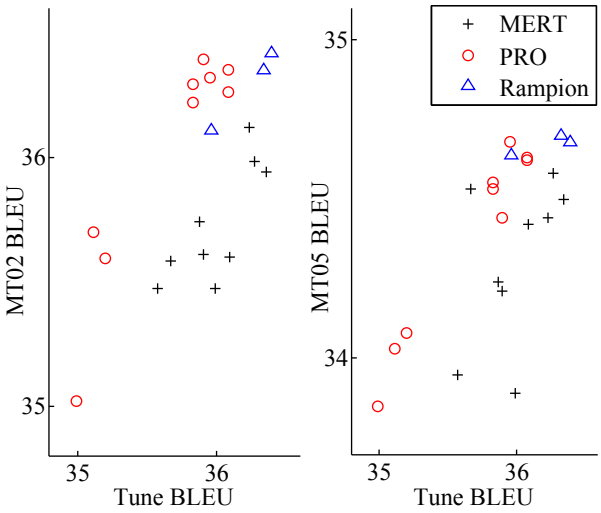


Figure 3: ZH→EN training runs. The cluster of PRO points to the left corresponds to one of the random initial models; MERT and RAMPION were able to recover while PRO was not.

4.2 Sensitivity Analysis

We now measure the sensitivity of these training methods to different initializers and to randomness in the algorithms. RAMPION is deterministic, but MERT uses random starting points and search directions and PRO uses random sampling to choose pairs for training its binary classifier.

For initial models, we used the default parameters in Moses as well as two randomly-generated models.[13] We ran RAMPION once with each of the three initial models, and MERT and PRO three times with each. This allows us to compare variance due to initializers as well as due to the nondeterminism in each algorithm. Fig. 3 plots the results. While PRO exhibits a small variance for a given initializer, as also reported by Hopkins and May (2011), it had

[10] The PRO algorithm samples pairs of translations from k-best lists on each iteration and trains a binary classifier to rank pairs according to the cost function. The loss function underlying PRO depends on the choice of binary classifier and also on the sampling strategy. We leave an analysis of PRO's loss function to future work.

[11] Hopkins and May used 30 iterations, but showed that training had converged by 25.

[12] We found performance to be better when using a smaller value of T'; we suspect that using small T' guards against overfitting to any particular set of k-best lists. We also found the value of α to affect performance, although $\alpha \in \{1, 5, 10\}$ all worked well. Performance was generally insensitive to C. We fixed $\eta = 0.0001$ early on and did little tuning to it.

[13] The default weights are 0.3 for reordering features, 0.2 for phrase table features, 0.5 for the language model, and -1 for the word penalty. We generated each random model by sampling each feature weight from a $\mathcal{N}(\mu, \sigma^2)$ with μ equal to the default weight for that feature and $\sigma = |\mu/2|$.

Method	UR→EN		ZH→EN		AR→EN			avg
	MT08*	MT09	MT02	MT05	MT05	MT08 NW	MT08 WB	
MERT	**24.5** (0.1)	**24.6** (0.0)	35.7 (0.3)	34.2 (0.2)	55.0 (0.7)	**49.8** (0.3)	**32.6** (0.2)	36.6
PRO	24.2 (0.1)	24.2 (0.1)	36.3 (0.1)	34.5 (0.0)	**55.6** (0.1)	49.6 (0.0)	31.7 (0.0)	36.6
RAMPION	**24.5**	**24.6**	**36.4**	**34.7**	55.5	**49.8**	32.1	**36.8**

Table 1: %BLEU on several test sets for UR→EN, ZH→EN, and AR→EN translation. Algorithms with randomization (MERT and PRO) were run three times with different random seeds and averages are shown in each cell followed by standard deviations in parentheses. All results in this table used a single initial model (the default Moses weights). The final column shows the average %BLEU across all individual test set scores, so 21 scores were used for MERT and PRO and 7 for RAMPION.

Method	UR→EN			ZH→EN		
	Tune	MT08*	MT09	Tune	MT02	MT05
PRO	**29.4**	22.3	23.0	**40.9**	35.7	33.6
RAMPION	27.8	**24.2**	**24.6**	38.8	**36.2**	**34.3**

Table 2: %BLEU with large feature sets.

trouble recovering from one of the random initializers. Therefore, while the within-initializer variance for PRO tended to be smaller than that of MERT, PRO's overall range was larger. RAMPION found very similar weights regardless of $\boldsymbol{\theta}_0$.

4.3 Adding Features

Finally, we compare RAMPION and PRO with an extended feature set; MERT is excluded as it fails in such settings (Hopkins and May, 2011).

We added count features for common monolingual and bilingual lexical patterns from the parallel corpus: the 1k most common bilingual word pairs from phrase extraction, 200 top unigrams, 1k top bigrams, 1k top trigrams, and 4k top trigger pairs extracted with the method of Rosenfeld (1996), ranked by mutual information. We integrated the features with our training procedure by using Moses to generate lattices instead of k-best lists. We used cube pruning (Chiang, 2007) to incorporate the additional (potentially non-local) features while extracting k-best lists from the lattices to pass to the training algorithms.[14]

Results are shown in Table 2. We find that PRO finds much higher BLEU scores on the tuning data but fails to generalize, leading to poor performance on the held-out test sets. We suspect that incorporating regularization into training the binary classifier within PRO may mitigate this overfitting. RAMPION is more stable by contrast. This is a challenging learning task, as lexical features are prone to overfitting with a small tuning set. Hopkins and May (2011) similarly found little gain on test data when using extended feature sets in phrase-based translation for these two language pairs.

Results for AR→EN translation were similar and are omitted for space; these and additional experiments will be included in a forthcoming report.

[14] In cube pruning, each node's local n-best list had $n = 100$.

5 Conclusion

We have framed MT training as empirical risk minimization and clarified loss functions that were optimized by well-known procedures. We have proposed directly optimizing the structured ramp loss implicit in prior work with a novel algorithm—RAMPION—which performs comparably to state-of-the-art training algorithms and is empirically more stable. Our source code, which integrates easily with Moses, is available at `www.ark.cs.cmu.edu/MT`.

Acknowledgments

We thank Colin Cherry, Chris Dyer, Joseph Keshet, David McAllester, and members of the ARK research group for helpful comments that improved this paper. This research was supported in part by the NSF through CAREER grant IIS-1054319, the U. S. Army Research Laboratory and the U. S. Army Research Office under contract/grant number W911NF-10-1-0533, and Sandia National Laboratories (fellowship to K. Gimpel).

References

A. Arun and P. Koehn. 2007. Online learning methods for discriminative training of phrase based statistical machine translation. In *Proc. of MT Summit XI*.

P. Blunsom and M. Osborne. 2008. Probabilistic inference for machine translation. In *Proc. of EMNLP*.

P. Blunsom, T. Cohn, and M. Osborne. 2008. A discriminative latent variable model for statistical machine translation. In *Proc. of ACL*.

D. Cer, D. Jurafsky, and C. Manning. 2008. Regularization and search for minimum error rate training. In *Proc. of ACL-2008 Workshop on Statistical Machine Translation*.

D. Cer. 2011. *Parameterizing Phrase Based Statistical Machine Translation Models: An Analytic Study*. Ph.D. thesis, Stanford University.

P. Chang, M. Galley, and C. Manning. 2008. Optimizing Chinese word segmentation for machine translation performance. In *Proc. of ACL-2008 Workshop on Statistical Machine Translation*.

S. Chen and J. Goodman. 1998. An empirical study of smoothing techniques for language modeling. Technical report 10-98, Harvard University.

C. Cherry and G. Foster. 2012. Batch tuning strategies for statistical machine translation. In *Proc. of NAACL*.

D. Chiang, Y. Marton, and P. Resnik. 2008. Online large-margin training of syntactic and structural translation features. In *Proc. of EMNLP*.

D. Chiang, W. Wang, and K. Knight. 2009. 11,001 new features for statistical machine translation. In *Proc. of NAACL-HLT*.

D. Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proc. of ACL*.

D. Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

J. H. Clark, C. Dyer, A. Lavie, and N. A. Smith. 2011. Better hypothesis testing for statistical machine translation: Controlling for optimizer instability. In *Proc. of ACL*.

M. Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proc. of EMNLP*.

R. Collobert, F. Sinz, J. Weston, and L. Bottou. 2006. Trading convexity for scalability. In *ICML*.

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

C. B. Do, Q. Le, C. H. Teo, O. Chapelle, and A. Smola. 2008. Tighter bounds for structured estimation. In *Proc. of NIPS*.

R. O. Duda and P. E. Hart. 1973. *Pattern classification and scene analysis*. John Wiley, New York.

G. Foster and R. Kuhn. 2009. Stabilizing minimum error rate training. In *Proc. of Fourth Workshop on Statistical Machine Translation*.

K. Gimpel and N. A. Smith. 2010. Softmax-margin CRFs: Training log-linear models with cost functions. In *Proc. of NAACL*.

M. Hopkins and J. May. 2011. Tuning as ranking. In *Proc. of EMNLP*.

B. H. Juang, W. Chou, and C. H. Lee. 1997. Minimum classification error rate methods for speech recognition. *Speech and Audio Processing, IEEE Transactions on*, 5(3):257–265, may.

J. Kaiser, B. Horvat, and Z. Kacic. 2000. A novel loss function for the overall risk criterion based discriminative training of hmm models. In *Proc. of ICSLP*.

P. Koehn, F. J. Och, and D. Marcu. 2003. Statistical phrase-based translation. In *Proc. of HLT-NAACL*.

P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proc. of ACL (demo session)*.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML*.

Y. Lee, K. Papineni, S. Roukos, O. Emam, and H. Hassan. 2003. Language model based Arabic word segmentation. In *Proc. of ACL*.

Z. Li and J. Eisner. 2009. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proc. of EMNLP*.

P. Liang, A. Bouchard-Côté, D. Klein, and B. Taskar. 2006. An end-to-end discriminative approach to machine translation. In *Proc. of COLING-ACL*.

Chin-Yew Lin and Franz Josef Och. 2004. Orange: a method for evaluating automatic evaluation metrics for machine translation. In *Proc. of Coling*.

W. Macherey, F. Och, I. Thayer, and J. Uszkoreit. 2008. Lattice-based minimum error rate training for statistical machine translation. In *EMNLP*.

A. F. T. Martins, K. Gimpel, N. A. Smith, E. P. Xing, P. M. Q. Aguiar, and M A. T. Figueiredo. 2010. Learning structured classifiers with dual coordinate descent. Technical report, Carnegie Mellon University.

D. McAllester and J. Keshet. 2011. Generalization bounds and consistency for latent structural probit and ramp loss. In *Proc. of NIPS*.

D. McAllester, T. Hazan, and J. Keshet. 2010. Direct loss minimization for structured prediction. In *Proc. of NIPS*.

R. C. Moore and C. Quirk. 2008. Random restarts in minimum error rate training for statistical machine translation. In *Proc. of Coling*.

F. J. Och and H. Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proc. of ACL*.

F. J. Och and H. Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1).

F. J. Och. 2003. Minimum error rate training for statistical machine translation. In *Proc. of ACL*.

K. Papineni, S. Roukos, T. Ward, and W.J. Zhu. 2001. BLEU: a method for automatic evaluation of machine translation. In *Proc. of ACL*.

D. Povey and P. C. Woodland. 2002. Minimum phone error and I-smoothing for improved discrimative training. In *Proc. of ICASSP*.

A. Quattoni, M. Collins, and T. Darrell. 2004. Conditional random fields for object recognition. In *NIPS 17*.

N. Ratliff, J. A. Bagnell, and M. Zinkevich. 2006. Subgradient methods for maximum margin structured learning. In *ICML Workshop on Learning in Structured Output Spaces*.

R. Rosenfeld. 1996. A maximum entropy approach to adaptive statistical language modeling. *Computer, Speech and Language*, 10(3).

D. A. Smith and J. Eisner. 2006. Minimum risk annealing for training log-linear models. In *Proc. of COLING-ACL*.

A. Stolcke. 2002. SRILM—an extensible language modeling toolkit. In *Proc. of ICSLP*.

B. Taskar, C. Guestrin, and D. Koller. 2003. Max-margin Markov networks. In *Advances in NIPS 16*.

V. Vapnik. 1998. *Statistical learning theory*. Wiley.

T. Watanabe, J. Suzuki, H. Tsukada, and H. Isozaki. 2007. Online large-margin training for statistical machine translation. In *Proc. of EMNLP-CoNLL*.

C. J. Yu and T. Joachims. 2009. Learning structural SVMs with latent variables. In *Proc. of ICML*.

A. L. Yuille and Anand Rangarajan. 2002. The concave-convex procedure (CCCP). In *Proc. of NIPS*. MIT Press.

R. Zens, S. Hasan, and H. Ney. 2007. A systematic comparison of training criteria for statistical machine translation. In *Proc. of EMNLP*.