

---

# Toward Purely Discriminative Training for Tree-Structured Translation Models

Benjamin Wellington  
Joseph Turian  
I. Dan Melamed

---

## 7.1 Introduction

Discriminative training methods have recently led to significant advances in the state of the art of machine translation (MT). Another promising trend is the incorporation of syntactic information into MT systems. Combining these trends is difficult for reasons of system complexity and computational complexity. This study makes progress toward a syntax-aware MT system whose every component is trained discriminatively. Our main innovation is an approach to discriminative learning that is computationally efficient enough for large statistical MT systems, yet whose accuracy on translation subtasks is near the state of the art.

Our approach to predicting a translation string is to predict its parse tree, and then read the string off the tree. Predicting a target tree given a source tree is equivalent to predicting a synchronous tree (*bitree*) that is consistent with the source tree. Our method for training tree transducers was to train an inference engine to predict bitrees. The *inference engine* employs the traditional AI technique of predicting a structure by searching over possible sequences of inferences, where each inference predicts a part of the eventual structure. Thus, to train a model for predicting bitrees, it is sufficient to train it to predict correct inferences. However, unlike most approaches employed in natural language processing (NLP), the proposed method makes no independence assumptions: the function that evaluates each inference can use arbitrary information not only from the input but also from all previous inferences.

Let us define some terms to help explain how our algorithm predicts a tree. An *item* is a node in the tree. Every *state* in the search space consists of a set of items, representing nodes that have been inferred since the algorithm started. States whose items form a complete tree are final states. An *inference* is a (state, item) pair, i.e., a state and an item to be added to it. Each inference represents a transition from one state to another. A state is *correct* if it is possible to infer zero or more items

to obtain the final state that corresponds to the training data tree. Similarly, an inference is correct if it leads to a correct state.

Given input  $s$ , the inference engine searches the possible complete trees  $T(s)$  for the tree  $\hat{t} \in T(s)$  that has minimum *cost*  $C_\Theta(t)$  under model  $\Theta$ :

$$\hat{t} = \arg \min_{t \in T(s)} C_\Theta(t) = \arg \min_{t \in T(s)} \left( \sum_{j=1}^{|t|} c_\Theta(i_j) \right). \quad (7.1)$$

The  $i_j$  are the inferences involved in constructing tree  $t$ .  $c_\Theta(i)$  is the cost of an individual inference  $i$ . The number of states in the search space is typically exponential in the size of the input. The freedom to compute  $c_\Theta(i)$  using arbitrary nonlocal information from anywhere in inference  $i$ 's state precludes exact solutions by ordinary dynamic programming. We know of two effective ways to approach such large search problems. The first is to restrict the order in which items can be inferred, for example, bottom-up. The second is to make the simplifying assumption that the cost of adding a given item to a state is the same for all states. Under this assumption, the fraction of any state's cost due to a particular item can be computed just once per item, instead of once per state. However, in contrast to traditional context-free parsing algorithms, that computation can involve context-sensitive features.

## 7.2 Related Work

A probabilistic treatment of Eq. (7.1) uses  $C_\Theta(t) = \Pr(t|s)$ , the probability that a text  $s$  in a source language will translate into a text  $t$  in the target language. Brown et al. (1993) used Bayes's rule to decompose this posterior into a language model  $\Pr(t)$  and a translation model  $\Pr(s|t)$ :

$$\Pr(t|s) = \frac{\Pr(t) \Pr(s|t)}{\sum_t \Pr(t) \Pr(s|t)} \propto \Pr(t) \Pr(s|t). \quad (7.2)$$

The problem with this model is that finding the maximum probability tree is difficult. Foster (2000) instead directly modeled the posterior  $\Pr(t|s)$  using a chain-rule expansion:

$$\Pr(t|s) = \prod_{i=1}^{|t|} \Pr(t_i | t_{i-1}, \dots, t_1, s), \quad (7.3)$$

which is a special case of Eq. (7.1). In Foster (2000), each inference  $t_i$  adds the  $i$ th token in  $t$ . Hence, the translation text was predicted one word at a time, where each prior word in the translation text was known at the point of guessing the next. Foster was the first to build a large-scale log-linear translation model. More specifically, Foster modeled the probability of individual decisions  $\Pr(t_i | t_{i-1}, \dots, t_1, s)$  using an

unregularized locally normalized log-linear model (Berger et al., 1996). There are several differences between our approach and Foster’s:

- His learning method was unregularized. Unregularized models tend to overfit, especially in the presence of useful features that always occur with a certain target output. Our learning approach uses  $\ell_1$ -regularization.
- To reduce overfitting, Foster performed automatic feature selection as a separate step before training. This approach can inadvertently remove important features. In contrast, our learning method does automatic feature selection during training.
- Foster’s model is induced over just the *atomic* features he defined. Our approach achieves more powerful modeling by combining *atomic* features in useful ways.
- Foster’s model was locally normalized, i.e., there was conservation of mass among the candidate inferences at each state. Local normalization can cause label bias (Lafferty et al., 2001). Our models are unnormalized, which avoids label bias (Turian, 2007, sections 4.1 and 7.1).
- The inference process of Foster is to infer tokens in the target string, one by one. Our inference process is to build a target-side syntax tree bottom-up, one node at a time, which can be viewed as translation by parsing (Melamed, 2004).

Och and Ney (2002) described a generalization of the approach of Brown et al. (1993), which allowed different information sources in the form of features. Och and Ney’s approach was to perform minimum error-rate training (MERT), using the loss function of one’s choice. One such approach was that of Koehn et al. (2003), who built a phrase-based statistical MT system that worked by calculating the probability that one phrase would translate to another and compiling those probabilities into a phrase table. In their model, discriminative methods were used to tune a handful of metaparameters, namely the parameters of (1) the log probability of the output sentence  $t$  under a language model, (2) the score of translating a source phrase into a target phrase based on the phrase table just described, (3) a distortion score, and (4) a length penalty. Several authors have even applied this method to syntax-aware systems (Chiang, 2005; Quirk et al., 2005). However, while MERT was a significant step forward in applying discriminative training to statistical MT, it can be performed reliably on only a small number of parameters. For this reason, it cannot be used with approaches like that of Foster (2000), in which there are tens of thousands of features, let alone approaches like ours involving millions of features. Another limitation of MERT is that the submodels may or may not be optimized for the same objective as the metaparameters. On a task like machine translation between common languages, where training data is abundant, a more elegant and more accurate system might be obtained by training all parts of the system with a single regularized objective.

More recent attempts to take full advantage of discriminative methods are the perceptron-based approaches of Tillmann and Zhang (2005) and Cowan et al. (2006), and the perceptron-like approach of Liang et al. (2006). All of these took advantage of discriminative methods to learn parameters for models with millions

of features. All of them were limited by the instability of the perceptron algorithm. Tillmann and Zhang (2006) used a more principled approach, involving stochastic gradient descent over a regularized loss function in a reranking framework. From a machine-learning point of view, this latter work was perhaps the most advanced (Bottou and Bousquet, 2008). However, all of these recent efforts were applied only to finite-state translation models.

Our work is the first, to our knowledge, to apply principled discriminative learning directly to the millions of parameters of a tree-structured translation model.

---

## 7.3 Learning Method

The goal of learning is to induce  $c_{\Theta}$ , which is the cost function for individual inferences.  $c_{\Theta}$  is used in Eq. (7.1) to find the minimum cost output structure. An important design decision in learning the inference cost function  $c_{\Theta}$  is the choice of feature set. Given the typically large number of possible features, the learning method must satisfy two criteria. First, it must be able to learn effectively even if the number of irrelevant features is exponential in the number of examples. It is too time-consuming to manually figure out the right feature set for such problems. Second, the learned function must be sparse. Otherwise, it would be too large for the memory of an ordinary computer, and therefore impractical. In this section, we describe how  $c_{\Theta}$  is induced.

### 7.3.1 Problem Representation

The training data used for translation initially comes in the form of bitrees. These gold-standard trees are used to generate training examples, each of which is a candidate inference: starting at the initial state, we randomly choose a sequence of correct inferences that lead to the (gold-standard) final state.<sup>1</sup> All the candidate inferences that can possibly follow each state in this sequence become part of the training set. The vast majority of these inferences will lead to incorrect states, which makes them negative examples. More specifically, the training set  $I$  consists of candidate inferences. Each inference  $i$  in the training set  $I$  can be expanded to a tuple  $\langle X(i), y(i) \rangle$ .  $X(i)$  is a feature vector describing  $i$ , with each element in  $\{0, 1\}$ . We will use  $X_f(i)$  to refer to the element of  $X(i)$  that pertains to feature  $f$ .  $y(i) = +1$  if  $i$  is correct, and is  $y(i) = -1$  if not.

An advantage of this method of generating training examples is that it does not require a working inference engine and can be run prior to any training. A

---

1. The order of inferences is nondeterministic, so there may be many paths to the gold-standard final state. Although the monolingual experiments of Turian (2007, section 6.4) indicate that having a deterministic inference logic is preferable, for the task of tree transduction it is not clear if there is a sensible canonical ordering.

disadvantage of this approach is that it does not teach the model to recover from mistakes. This is because example generation does not use the model in any way. Our training set is identical to the inferences that would be scored during search using an oracle cost function. The oracle cost function  $\hat{c}(i)$  is  $\infty$  if  $y(i) = -1$  and 0 if  $y(i) = +1$ . If we wanted to improve accuracy by teaching the inference engine to recover from its mistakes, we could use inference during example generation and iterate training. With a deterministic logic, using the oracle cost function to generate training examples is similar to the first iteration of SEARN (Daumé et al., 2005, 2006).

### 7.3.2 Objective Function

The training method induces a real-valued inference evaluation function  $h_{\Theta}(i)$ . We will describe how to transform  $h_{\Theta}(i)$  to  $c_{\Theta}(i)$  in Eq. (7.5). In this chapter,  $h_{\Theta}$  is a linear model parameterized by a real vector  $\Theta$ , which has one entry for each feature  $f$ :  $h_{\Theta}(i) = \Theta \cdot X(i) = \sum_f \Theta_f \cdot X_f(i)$ . The sign of  $h_{\Theta}(i)$  predicts the  $y$ -value of  $i$  and the magnitude gives the confidence in this prediction. The training procedure adjusts  $\Theta$  to minimize the expected risk  $R_{\Theta}$  over training set  $I$ .  $R_{\Theta}$  is the *objective* or *risk* function, which is the sum of *loss* function  $L_{\Theta}$  and *regularization* term  $\Omega_{\Theta}$ . The loss measures the extent to which the model underfits the training data. The regularization term is a measure of model complexity, and is used to avoid overfitting the training data. We use the log-loss and  $\ell_1$ -regularization, so we have

$$\begin{aligned} R_{\Theta}(I) &= L_{\Theta}(I) + \Omega_{\Theta} = \left[ \sum_{i \in I} l_{\Theta}(i) \right] + \Omega_{\Theta} \\ &= \left[ \sum_{i \in I} [\ln(1 + \exp(-\mu_{\Theta}(i)))] \right] + \left[ \lambda \cdot \sum_f |\Theta_f| \right]. \end{aligned} \quad (7.4)$$

$\lambda$  is a parameter that controls the strength of the regularizer and  $\mu_{\Theta}(i) = y(i) \cdot h_{\Theta}(i)$  is the *margin* of example  $i$ . The tree cost  $C_{\Theta}$  (Eq. (7.1)) is obtained by computing the objective function with  $y(i) = +1$  for every inference in the tree, and treating the penalty term  $\Omega_{\Theta}$  as constant:

$$c_{\Theta}(i) = l_{\Theta}(\langle X(i), y = +1 \rangle) = \ln(1 + \exp(-h_{\Theta}(i))). \quad (7.5)$$

The experiments in section 7.4.2 motivate the use of  $\ell_1$  regularization instead of  $\ell_2$  regularization.

### 7.3.3 Minimizing the Risk

We minimize the risk  $R_{\Theta}$  using a form of forward stagewise additive modeling, a procedure Guyon and Elisseeff (2003) refer to as *embedded feature selection*. At

each iteration in training, we pick one or more parameters of the model to adjust (*feature selection*), then adjust these parameters (*model update*).

### Feature Selection

We want to choose parameters that allow us to decrease the risk quickly. We define the *gain* of a feature  $f$  as

$$G_{\Theta}(I; f) = \max(\text{decrease in risk as we increase } f\text{'s parameter value,} \\ \text{decrease in risk as we decrease } f\text{'s parameter value,} \\ \text{decrease in risk if we leave } f\text{'s parameter value unchanged}). \quad (7.6)$$

More specifically, we pick the features with the steepest gradients (Mason et al., 1999, 2000; Perkins et al., 2003):

$$G_{\Theta}(I; f) = \max\left(\lim_{\epsilon \rightarrow 0^+} \frac{\partial -R_{\Theta}}{\partial \Theta_f}(\Theta_f + \epsilon), \lim_{\epsilon \rightarrow 0^-} \frac{\partial -R_{\Theta}}{\partial -\Theta_f}(\Theta_f + \epsilon), 0\right). \quad (7.7)$$

The limits are taken from above and below, respectively. This analysis technique allows us to determine the gain for any continuous function, regardless of whether it is continuously differentiable or not. To determine the gain function in Eq. (7.7) for a particular risk, we consider two cases:

- If we are using the  $\ell_1$  penalty ( $\Omega_{\Theta} = \sum_{f \in F} |\Theta_f|$ ) and  $\Theta_f = 0$ , then we are at the gradient discontinuity and we have

$$G_{\Theta}(I; f) = \max\left(\left|\frac{\partial L_{\Theta}}{\partial \Theta_f}\right| - \lambda, 0\right). \quad (7.8)$$

Observe that unless the magnitude of the gradient of the empirical loss  $|\partial L_{\Theta}(I)/\partial \Theta_f|$  exceeds the penalty term  $\lambda$ , the gain is zero and the risk increases as we adjust parameter  $\Theta_f$  away from zero in either direction. In other words, the parameter value is trapped in a “corner” of the risk. In this manner the polyhedral structure of the  $\ell_1$  norm tends to keep the model sparse (Riezler and Vasserman, 2004). Dudik et al. (2007) offer another perspective, pointing out that  $\ell_1$  regularization is “truly sparse”: if some feature’s parameter value is zero when the risk is minimized, then the optimal parameter value will remain at zero even under slight perturbations of the feature’s expected value and of the regularization penalty. However, if the gain is nonzero,  $G_{\Theta}(I; f)$  is the magnitude of the gradient of the risk as we adjust  $\Theta_f$  in the direction that reduces  $R_{\Theta}$ .

- If we are using the  $\ell_2$  penalty ( $\Omega_{\Theta} = \sum_{f \in F} \Theta_f^2$ ), then we have

$$G_{\Theta}(I; f) = \left|\frac{\partial L_{\Theta}}{\partial \Theta_f} + \lambda \cdot 2 \cdot \Theta_f\right|. \quad (7.9)$$

The  $\ell_2$ -regularization term disappears when  $\Theta_f = 0$ , and so—for unused features— $\ell_2$ -regularized feature selection is indistinguishable from unregularized feature selection. The only difference is that the  $\ell_2$ -penalty term reduces the magnitude of the optimal parameter setting for each feature. This is why  $\ell_2$ -regularization typically leads to models where many features are active (nonzero).

Let us define the *weight* of an example  $i$  under the current model as the rate at which loss decreases as the margin of  $i$  increases:

$$w_{\Theta}(i) = -\frac{\partial l_{\Theta}(i)}{\partial \mu_{\Theta}(i)} = \frac{1}{1 + \exp(\mu_{\Theta}(i))}. \quad (7.10)$$

To determine  $\frac{\partial L_{\Theta}}{\partial \Theta_f}$ , the gradient of the unpenalized loss  $L_{\Theta}$  with respect to the parameter  $\Theta_f$  of feature  $f$ , which is used in Eq. (7.8) and (7.9), we have

$$\begin{aligned} \frac{\partial L_{\Theta}(I)}{\partial \Theta_f} &= \sum_{i \in I} \frac{\partial l_{\Theta}(i)}{\partial \Theta_f} = \sum_{i \in I} \frac{\partial l_{\Theta}(i)}{\partial \mu_{\Theta}(i)} \cdot \frac{\partial \mu_{\Theta}(i)}{\partial \Theta_f} \\ &= - \sum_{i \in I} w_{\Theta}(i) \cdot [y(i) \cdot X_f(i)] = - \sum_{\substack{i \in I: \\ X_f(i)=1}} w_{\Theta}(i) \cdot y(i). \end{aligned} \quad (7.11)$$

Turian (2007, section 4.2) contains detailed derivations of the gain functions in this section, as well as more discussion and comparison to related learning approaches.

### Compound Feature Selection

Although  $h_{\Theta}$  is just a linear discriminant, it can nonetheless learn effectively if feature space  $F$  is high-dimensional. Features encode information about the inference in question. A priori, we define only a set  $A$  of simple atomic features (sometimes also called *attributes* or *primitive* features).

Feature *construction* or *induction* methods are learning methods in which we induce a more powerful machine than a linear discriminant over just the attributes, and the power of the machine can be data-dependent.

Our learner induces *compound* features, each of which is a conjunction of possibly negated atomic features. Each atomic feature can have one of three values (yes/no/don't care), so the compound feature space  $F$  has size  $3^{|A|}$ , exponential in the number of atomic features. Each feature selection iteration selects a set of *domain-partitioning* features. Domain-partitioning features  $\tilde{F}$  cover the domain and are nonoverlapping for every possible inference  $i$  in the space of inferences:

**cover** :  $\exists f \in \tilde{F}$  s.t.  $X_f(i) = 1$ .

**nonoverlapping** :  $\forall f, f' \in \tilde{F}$ , if  $X_f(i) = 1$  and  $X_{f'}(i) = 1$  then  $f = f'$ .

In other words,  $\tilde{F}$  partitions the inference space if for any  $i$ , there is a unique feature  $f$  in the partition  $\tilde{F}$  such that  $X_f(i) = 1$ , and  $X_{f'}(i) = 0$  for all other features.

One way to choose a set of domain-partitioning compound features is through greedy splitting of the inference space. This is the approach taken by decision trees,

for some particular splitting criterion. Since we wish to pick splits that allow us to reduce risk, our splitting criterion uses the gain function. In particular, assume that we have two different two-feature partitions of some subspace of the inference space. The features  $f_1$  and  $f_2$  are one partition, and  $f_1'$  and  $f_2'$  are the other. We treat the gain of each partition as  $G_\Theta(f_1) + G_\Theta(f_2)$  and  $G_\Theta(f_1') + G_\Theta(f_2')$ , respectively, and prefer the partition with higher gain. The work of Mason et al. (1999, 2000), who studied a related problem, motivate this choice theoretically using a first-order Taylor expansion.

### ***Model Update***

After we have selected one or more high-gain features, we update the model. *Parallel* update methods can adjust the parameter values of overlapping features to minimize the risk. *Stagewise* or *sequential* update methods adjust the parameter values of nonoverlapping features to minimize the risk, each of which can be optimized independently, e.g., using line search. Since domain-partitioning features are nonoverlapping, we use sequential updates to choose parameter values.

### ***Boosting Regularized Decision Trees***

To summarize, our approach to minimizing the risk is divided into feature selection and model update. Feature selection is performed using gradient descent in the compound feature space through a greedy splitting procedure to partition the inference space. Model update is performed using a sequential update method.

Our specific implementation of this risk minimization approach is to boost an ensemble of confidence-rated decision trees. This boosted ensemble of confidence-rated decision trees represents  $\Theta$ . We write  $\Delta\Theta(t)$  to represent the parameter values chosen by tree  $t$ , and for ensemble  $T$  we have  $\Theta = \sum_{t \in T} \Delta\Theta(t)$ . Each internal node is split on an atomic feature. The path from the root to each node  $n$  in a decision tree corresponds to a *compound* feature  $f$ , in which case we write  $\varphi(n) = f$ . An example  $i$  percolates down to node  $n$  iff  $X_{\varphi(n)} = 1$ . Each leaf node  $n$  keeps track of delta-parameter value  $\Delta\Theta_{\varphi(n)}(t)$ . To score an example  $i$  using a decision tree, we percolate the example down to a leaf  $n$  and return confidence  $\Delta\Theta_{\varphi(n)}(t)$ . The score  $h_\Theta(i)$  given to an example  $i$  by the whole ensemble is the sum of the confidences returned by all trees in the ensemble.

Figure 7.1 presents our training algorithm. At the beginning of training, the ensemble is empty,  $\Theta = \mathbf{0}$ , and  $\lambda$  is set to  $\infty$ . We grow the ensemble until the risk cannot be further reduced for the current choice of  $\lambda$ . So for some choice of penalty factor  $\lambda'$ , our model is the ensemble up until when  $\lambda$  was decayed below  $\lambda'$ . We then relax the regularizer by decreasing  $\lambda$  and continue training. We use the decay factor  $\eta = 0.9$  as our *learning rate*. In this way, instead of choosing the best  $\lambda$  heuristically, we can optimize it during a single training run. This approach of progressively relaxing the regularizer during a single training run finds the entire

**Algorithm 7.1** Outline of the training algorithm.

---

```

procedure TRAIN( $I$ )
  ensemble  $\leftarrow \emptyset$ 
  regularization parameter  $\lambda \leftarrow \infty$ 
  while not converged do
     $g \leftarrow \max_{a \in A} (|\partial L_{\Theta} / \partial \Theta_{\emptyset}|, |\partial L_{\Theta} / \partial \Theta_a|, |\partial L_{\Theta} / \partial \Theta_{-a}|)$   $\triangleright$  Find the best
     $\lambda \leftarrow \min(\lambda, \eta \cdot g)$   $\triangleright$  unpenalized gain of any root split
     $\triangleright$  Maybe decay the penalty parameter
     $\triangleright$  so that training can progress

     $\Delta\Theta(t) \leftarrow \text{BUILDTREE}_{\Theta}(\lambda, I)$ 
    if  $R_{\Theta + \Delta\Theta(t)} < R_{\Theta} + \epsilon$  then  $\triangleright$  If we have reduced loss by some threshold
       $\Theta \leftarrow \Theta + \Delta\Theta(t)$   $\triangleright$  Then keep the tree and update the model
    else  $\triangleright$  Otherwise, we have converged for this choice of  $\lambda$ 
       $\lambda \leftarrow \eta \cdot \lambda$   $\triangleright$  Decay the penalty parameter

procedure BUILDTREE $_{\Theta}(\lambda, I)$ 
   $t \leftarrow$  root node only
  while some leaf in  $t$  can be split do  $\triangleright$  See Eq. (7.13)
    split the leaf to maximize gain  $\triangleright$  See Eq. (7.12)
    percolate every  $i \in I$  to a leaf node
    for each leaf  $n$  in  $t$  do
       $\Delta\Theta_{\varphi(n)}(t) \leftarrow \arg \min_{\Delta\Theta_{\varphi(n)}(t)} (R_{\Theta + \Delta\Theta_{\varphi(n)}(t)}(I_{\varphi(n)}))$   $\triangleright$  Choose  $\Delta\Theta_{\varphi(n)}(t)$  to
       $\triangleright$  minimize  $R$  using a line search

  return  $\Delta\Theta(t)$ 

```

---

regularization path (e.g., Hastie et al., 2004) and is a form of continuation method for global optimization (e.g., Chapelle et al., 2006).

Each invocation of BUILDTREE has several steps. First, we choose some compound features that will allow us to decrease the risk function. We do this by building a decision tree whose leaf node paths represent the chosen compound features. Second, we confidence-rate each leaf to minimize the risk over the examples that percolate down to that leaf. Finally, we return the decision tree. TRAIN appends it to the ensemble and update parameter vector  $\Theta$  accordingly. In this manner, compound feature selection is performed incrementally during training, as opposed to a priori.

The construction of each decision tree begins with a sole leaf node, the root node, which corresponds to a dummy “always true” feature  $\emptyset$ . By “always true”, we mean that  $X_{\emptyset}(i) = 1$  for any example  $i$ . We recursively split leaf nodes by choosing the best atomic splitting feature that will allow us to increase the gain. Specifically, we consider splitting each leaf node  $n$  using atomic feature  $\hat{a}$ , where  $f = \varphi(n)$  and

$$\hat{a} = \arg \max_{a \in A} [G_{\Theta}(I; f \wedge a) + G_{\Theta}(I; f \wedge \neg a)]. \quad (7.12)$$

(We use  $f$  to mean an actual feature, not a feature index.) Splitting using  $\hat{a}$  would create children nodes  $n_1$  and  $n_2$ , with  $\varphi(n_1) = f \wedge \hat{a}$  and  $\varphi(n_2) = f \wedge \neg \hat{a}$ . We split

**Table 7.1** Data sizes in 000's

	sent. pairs	English words		French words	
		types	tokens	types	tokens
training1	10	11	210	14	232
training2	100	29	2100	38	2300
tuning	1	3.5	21	4.2	23
devel	1	3.5	21	4.1	23
test	1	3.5	21	4.1	23

node  $n$  using  $\hat{a}$  only if the total gain of these two children exceeds the gain of the unsplit node, i.e., if

$$G_{\Theta}(I; f \wedge \hat{a}) + G_{\Theta}(I; f \wedge \neg \hat{a}) > G_{\Theta}(I; f). \quad (7.13)$$

Otherwise,  $n$  remains a leaf node of the decision tree, and  $\Theta_{\varphi(n)}$  becomes one of the values to be optimized during the parameter update step.

Parameter update is done sequentially on only the most recently added compound features, which correspond to the leaves of the new decision tree. After the entire tree is built, we percolate each example down to its appropriate leaf node. As indicated earlier, a convenient property of decision trees is that the leaves' compound features are mutually exclusive, so their parameters can be directly optimized independently of each other. We use a line search to choose for each leaf node  $n$  the parameter  $\Theta_{\varphi(n)}$  that minimizes the risk over the examples in  $n$ .

## 7.4 Experiments

### 7.4.1 Data

The data for our experiments came from the English and French components of the EuroParl corpus (Koehn, 2005). From this corpus, we extracted sentence pairs where both sentences had between 5 and 40 words, and where the ratio of their lengths was no more than 2:1. We then extracted disjoint training, tuning, development, and test sets. The tuning, development, and test sets were 1000 sentence pairs each. For some experiments we used 10,000 sentence pairs of training data; for others we used 100,000. Descriptive statistics for these corpora are in table 7.1.

We parsed the English half of the training, tuning, development, and test bitexts using Dan Bikel's parser (Bikel, 2004), which was trained on the Penn treebank (Marcus et al., 1993). On each of our two training sets, we induced word alignments using the default configuration of GIZA++ (Och and Ney, 2003). The training set word alignments and English parse trees were fed into the default French-English

hierarchical alignment algorithm distributed with the GenPar system (Burbank et al., 2005), to produce binarized tree alignments.

#### 7.4.2 Word Transduction

Our first set of experiments evaluated our approach on the task of translating individual words from English to French. The input was a single English word, which we’ll call the “focus” word, along with a vector of features (described below). The output was a single French word, possibly NULL. The proposed translation was compared to a gold-standard translation.

The gold-standard word pairs that we used for this task were extracted from the tree alignments described above. Thus, the gold standard was a set of GIZA++ Viterbi word alignments filtered by a tree cohesion constraint. Regardless of whether they are created manually or automatically, word alignments are known to be highly unreliable. This property of the data imposed a very low artificial ceiling on all of our results, but it did not significantly interfere with our goal of controlled experiments to compare learning methods. To keep our measurements consistent across different training data sizes, the word alignments used for testing were the ones induced by GIZA++ when trained on the larger training set. The number of trials was equal to the number of source words for which GIZA++ predicts an alignment.

In contrast to Vickrey et al. (2005), we did not allow multiword “phrases” as possible translations. Our hypothesis, based on some evidence in section 7.4.3 and in Quirk and Menezes (2006), is that the best MT systems of the future will not need to deal in such objects. In our study, phrases might have raised our absolute scores, but they would have confounded our understanding of the results. Our experiment design also differs from Vickrey et al. (2005) in that we trained classifiers for *all* words in the training data.<sup>2</sup> There were 161K word predictions in the smaller (10,000 sentence pairs) training set, 1866K in the larger training set, 17.8K predictions in the tuning set, 14.2K predictions in the development set, and 17.5K predictions in the test set.

Using the smaller training set and guessing the most frequent translation of each source word achieves a baseline accuracy of 47.54% on the development set. With this baseline, we compared three methods for training word transducers on the word alignments described above. The first was the method described in section 7.3 for inducing  $\ell_1$ -regularized log-linear models over the compound feature space. The second method was similar to Vickrey et al. (2005): induce  $\ell_2$ -regularized log-linear models over the *atomic* feature space.<sup>3</sup> The third method was LaSVM (Bordes et al., 2005), an online SVM algorithm designed for large data sets.

---

2. David Vickrey (personal communication) informed us that Vickrey et al. (2005) omitted punctuation and function words, which are the most difficult in this task.

3. We do not induce  $\ell_2$ -regularized log-linear models over the *compound* feature space because Turian (2007, section 6.4) found that these models were too large even for monolingual parsing experiments. To induce  $\ell_2$ -regularized log-linear models over the

**Table 7.2** Percent accuracy on the development set and sizes of word-to-word classifiers trained on 10K or 100K sentence pairs. The feature sets used were (W)indow, (D)ependency, and (C)o-occurrence.  $\ell_1$  size is the number of compound feature types.

	W	W+D	W+C	W+D+C
10,000 training sentences — baseline = 47.54				
$\ell_2$	54.09	54.33	52.36	52.88
$\ell_1$	53.96	54.13	53.29	53.75
LaSVM	53.38	51.93	49.13	50.71
pruned $\ell_2$	47.37	46.01	46.68	45.01
$\ell_1$ size	54.1K	41.7K	37.8K	38.7K
$\ell_2$ size	1.67M	2.51M	5.63M	6.47M
pruned $\ell_2$ size	54.1K	41.7K	37.8K	38.7K
100,000 training sentences — baseline = 51.94				
$\ell_1$	62.00	62.42	61.98	62.40
$\ell_1$ size	736K	703K	316K	322K

For each training method, we experimented with several kinds of features, which we call “window,” “co-occurrence,” and “dependency.” Window features included source words and part-of-speech (POS) tags within a two-word window around the focus word, along with their relative positions (from  $-2$  to  $+2$ ). Co-occurrence features included all words and POS tags from the whole source sentence, without position information. Dependency features were compiled from the automatically generated English parse trees. The dependency features of each focus word were

- the label of its maximal projection (i.e., the highest node that has the focus word as its lexical head, which might be a leaf, in which case that label is a POS tag);
- the label and lexical head of the parent of the maximal projection;
- the label and lexical head of all dependents of the maximal projection;
- all the labels of all head-children (recursively) of the maximal projection.

The window features were present in all experimental conditions. The presence/absence of co-occurrence and dependency features yielded four “configurations.”

Using each of these configurations, each training method produced a confidence-rating binary classifier for each translation of each English word seen in the training data. In all cases, the test procedure was to choose the French word predicted with the highest confidence. All methods, including the baseline, predicted NULL for source words that were not seen in training data.

Table 7.2 shows the size and accuracy of all three methods on the development set, after training on 10,000 sentence pairs, for each of the four configurations. The best configurations of the two logistic regression methods far exceed the baseline,

---

*atomic* feature space, we used MegaM by Hal Daumé, available at <http://www.cs.utah.edu/~hal/megam/>

but otherwise they were statistically indistinguishable. The accuracy of LaSVM was similar to the regression methods when using only the window features, but it was significantly worse with the larger feature sets.

More interesting were the differences in model sizes. The  $\ell_2$ -regularized models were bigger than the  $\ell_1$ -regularized models by two orders of magnitude. The  $\ell_2$ -regularized models grew in size to accommodate each new feature type. In contrast, the  $\ell_1$ -regularized models *decreased* in size when given more useful features, without significantly losing accuracy. This trend was even stronger on the larger training set, where more of the features were more reliable.

The size of models produced by LaSVM grew linearly with the number of examples, because for source words like “the,” about 90% of the examples became support vectors. This behavior makes it infeasible to scale up LaSVM to significantly larger data sets, because it would need to compare each new example to all support vectors, resulting in near-quadratic runtime complexity.

To scale up to 100,000 sentence pairs of training data with just the window features, the  $\ell_2$  classifiers would need about 25 billion parameters, which could not fit in the memory of our computers. To make them fit, we could set all but the largest feature parameters to zero. We tried this on 10,000 training sentence pairs. The number of features allowed to remain active in each  $\ell_2$  classifier was the number of active features in the  $\ell_1$  classifier. Table 7.2 shows the accuracy of these “pruned”  $\ell_2$ -regularized classifiers on the development set, when trained on 10,000 sentence pairs. With the playing field leveled, the  $\ell_1$  classifiers were far more effective.

In preliminary experiments, we also tried perceptron-style updates, as suggested by Tillmann and Zhang (2005). However, for reasons given by Tewari and Bartlett (2005), the high-entropy decisions involved in our structured prediction setting often prevented convergence to useful classifiers. Likewise, Christoph Tillmann informed us (personal communication) that, to ensure convergence, he had to choose features very carefully, even for his finite-state MT system.

Regularization schemes that don’t produce sparse representations seem unsuitable for problems on the scale of machine translation. For this reason, we used only  $\ell_1$ -regularized log-loss for the rest of our experiments. Table 7.2 shows the accuracy and model size of the  $\ell_1$ -regularized classifier on the development set, when trained on 100,000 sentence pairs, using each of the four configurations. Our classifier far exceeded the baseline. The test set results for the best models (window + dependency features) were quite close to those on the development set: 54.64% with the smaller training set, and 62.88% with the larger.

### 7.4.3 Bag Transduction

The word-to-word translation task is a good starting point, but any conclusions that we might draw from it are inherently biased by the algorithm used to map source words to target words in the test data. Our next set of experiments was on a task with more external validity – predict a translation for *each* source word in the test data, regardless of whether GIZA++ predicted an alignment for it. The

difficulty with this task, of course, is that we have no deterministic word alignment to use as a gold standard. Our solution was to pool the word translations in each source sentence and compare them to the bag of words in the target sentence. We still predicted exactly one translation per source word, and that translation could be NULL. Thus, the number of target words predicted for each source sentence was less than or equal to the number of words in that source sentence. The evaluation measures for this experiment were precision, recall, and F-measure, with respect to the bag of words in the test target sentence.

We compared the four configurations of our  $\ell_1$ -regularized classifiers on this task to the most-frequent-translation baseline. We also evaluated a mixture model, where a classifier for each source word was chosen from the best one of the four configurations, based on that configuration’s accuracy on that source word in the tuning data. As an additional gauge of external validity, we performed the same task using the best publicly available machine translation system (Koehn et al., 2003). This comparison was enlightening but necessarily unfair. As mentioned above, our long-term goal is to build a system whose every component is discriminatively trained to optimize the objective function. We did not want to confound our study with techniques such as “phrase” induction, word-class induction, nondiscriminatively trained target language models, etc. On the other hand, modern MT systems are designed for use with such information sources, and cannot be fairly evaluated without them. So, we ran Pharaoh in two configurations. The first used the default system configuration, with a target language model trained on the target half of the training data. The second allowed Pharaoh to use its phrase tables but without a target language model. This second configuration allowed us to compare the accuracy of our classifiers to Pharaoh specifically on the subtask of MT for which they were designed.

The results are shown in table 7.3. The table shows that our method far exceeds the baseline. Since we predict only one French target word per English source word, the recall of our bag transducer was severely handicapped by the tendency of French sentences to be longer than their English equivalents. This handicap is reflected in the one-to-one upper bound shown in the table. With a language model, Pharaoh’s recall exceeded that of our best model by slightly less than this 13.7% handicap. However, we were surprised to discover that the bag transducer’s precision was much higher than Pharaoh’s when they compete on a more level playing field (without a language model), and higher even when Pharaoh was given a language model. Table 7.4 shows the accuracy of the best models on the test set. These results closely follow those on the development set.

This result suggests that it might not be necessary to induce “phrases” on the source side. Quirk and Menezes (2006) offer additional evidence for this hypothesis. After all, the main benefits of phrases on the source side are in capturing lexical context and local word reordering patterns. Our bag transducers capture lexical context in their feature vectors. Word order is irrelevant for bag transduction.

The main advantage of phrase-based models on this task is in proposing more words on the target side, which eliminates the one-to-one upper bound on recall.

**Table 7.3** (P)recision, (R)ecall, and (F)-measure for bag transduction of the development set. The discriminative transducers were trained with  $\ell_1$  regularization.

	P	R	F
training on 10,000 sentence pairs			
baseline	48.09	41.26	44.42
window only	53.93	42.81	47.73
dependency	53.97	42.72	47.69
co-occurrence	53.31	42.45	47.26
co-oc. + dep.	53.58	42.67	47.50
mixture model	54.05	42.95	47.87
training on 100,000 sentence pairs			
baseline	51.91	40.79	45.68
window only	58.79	44.26	50.50
dependency	59.12	44.57	50.82
co-occurrence	59.06	44.36	50.67
co-oc. + dep.	59.19	44.60	50.87
mixture model	59.03	44.55	50.78
Pharaoh w/o LM	32.20	54.62	40.51
Pharaoh with LM	56.20	57.49	56.84
1-to-1 upper bound	100.00	86.31	92.65

**Table 7.4** (P)recision, (R)ecall, and (F)-measure of bag transducers on the test set

	P	R	F
training on 10,000 sentence pairs			
dependency	54.36	42.75	47.86
mixture model	54.27	42.81	47.86
training on 100,000 sentence pairs			
co-oc. + dep.	59.49	44.19	50.71
mixture model	59.62	44.38	50.88
Pharaoh w/o LM	32.55	54.62	40.80
Pharaoh with LM	57.01	57.84	57.45

Another advantage is that phrase-based models require no parsing, and thus require no hand-annotated treebank to be available for training a parser for the source language. Note, however, that when training on 100,000 sentences, our best model not relying on parse tree information (the co-oc. model) was only a tiny bit less accurate than the best model overall (the co-oc. + dep. model).

#### 7.4.4 Tree Transduction

We experimented with a simplistic tree transducer, which involves only two types of inference. The first type transduces leaves; the second type transduces internal

nodes. The transduction of leaves is exactly the word-to-word translation task described in section 7.4.2. Leaves that are transduced to NULL are deterministically erased. Internal nodes are transduced merely by permuting the order of their children, where one of the possible permutations is to retain the original order. This transducer is grossly inadequate for modeling real bitext (Galley et al., 2004): it cannot account for many kinds of noise and for many real translingual phenomena, such as head-switching and discontinuous constituents, which are important for accurate MT. It cannot even capture common phrasal translations such as *there is / il y a*. However, it is sufficient for controlled comparison of learning methods. The learning method will be the same when we use more sophisticated tree transducers. Another advantage of this experimental design is that it uses minimal linguistic cleverness and is likely to apply to many language pairs, in contrast to other studies of constituent/dependent reordering that are more language-specific (Xia and McCord, 2004; Collins et al., 2005).

To reduce data sparseness, each internal node with more than two children was binarized, so that the multiclass permutation classification for the original node was reduced to a sequence of binary classifications. This reduction is different from the usual multiclass reduction to binary: in addition to making the classifier binary instead of multiclass, the reduction decomposes the label so that some parts of it can be predicted before others. For example, without this reduction, a node with children  $\langle A, B, C \rangle$  can be transduced to any of six possible permutations, requiring a six-class classifier. After binarization, the same six possible permutations can be obtained by first permuting  $\langle A, B \rangle$ , and then permuting the result with  $C$ , or by first permuting  $\langle B, C \rangle$  and then permuting the result with  $A$ . This reduction eliminates some of the possible permutations for nodes with four or more children (Wu, 1997).

Our monolingual parser indicated which node is the head-child of each internal node. Some additional permutations were filtered out using this information: two sibling nodes that were *not* the head-children of their parent were not allowed to participate in a permutation until one of them was permuted with the head-child sibling. Thus, if  $C$  was the head-child in the previous example, then  $\langle A, B \rangle$  could not be permuted first;  $\langle B, C \rangle$  had to be permuted first, before permuting with  $A$ .

We search for the tree with minimum total cost, as specified in Eq. (7.1). We compared two models of inference cost—one generative and one discriminative.

The generative model was based on a top-down tree transducer (Comon et al., 2002) that stochastically generates the target tree given the source tree. The generative process starts by generating the target root given the source root. It then proceeds top-down, generating every target node conditioned on its parent and on the corresponding node in the source tree. Let  $\pi$  be the function that maps every node to its parent, and let  $\eta$  be the function that maps every target node to

**Table 7.5** (P)recision, (R)ecall, and (F)-measure of transducers using 100,000 sentence pairs of training data

Source parser	Transduction model	Exponent 1			Exponent 2		
		P	R	F	P	R	F
generative	generative	51.29	38.30	43.85	22.62	16.90	19.35
generative	discriminative	59.89	39.53	47.62	26.94	17.78	21.42
discriminative	generative	50.51	37.76	43.21	22.04	16.47	18.85
discriminative	discriminative	<b>62.36</b>	39.06	<b>48.04</b>	<b>28.02</b>	17.55	<b>21.59</b>
	Pharaoh (w/o LM)	32.19	<b>54.62</b>	40.51	12.37	<b>20.99</b>	15.57

its corresponding source. If we view the target tree as consisting of nodes  $n$  with  $n_0$  being the root node, then the probability of the target tree  $t$  is

$$\Pr(t) = \Pr(n_0|\eta(n_0)) \cdot \prod_{n \neq n_0 \in t} \Pr(n|\pi(n), \eta(n)). \quad (7.14)$$

For the generative model, the cost of an inference  $i$  is the negative logarithm of the probability of the node  $n(i)$  that it infers:  $l(i) = -\log \Pr[n(i)|\pi(n(i)), \eta(n(i))]$ . We estimated the parameters of this transducer using the Viterbi approximation of the inside-outside algorithm described by Graehl and Knight (2004). Following Zhang and Gildea (2005), we lexicalized the nodes so that their probabilities capture bilexical dependencies.

The discriminative model was trained using the method in section 7.3, with the inference cost computed as described in Eq. (7.5). A separate classifier was induced for each possible translation of each source word seen in training data, to evaluate candidate transductions of leaf nodes. Additional classifiers were induced to confidence-rate candidate permutations of sibling nodes. Recall that each permutation involved a head-child node and one of its siblings. Since our input trees were lexicalized, it was easy to determine the lexical head of both the head-child and the other node participating in each permutation. Features were then compiled separately for each of these words according to the “window” and “dependency” feature types described in section 7.4.2. Since the tree was transduced bottom-up, the word-to-word translation of the lexical head of any node was already known by the time it participated in a permutation. So, in addition to dependents on the source side, there were also features to encode their translations. The final kind of feature used to predict permutations was whole synchronous context-free production rules, in bilexical, monolexical, and unlexicalized forms. We cannot imagine a generative process that could explain the data using this combination of features. Our hypothesis was that the discriminative approach would be more accurate, because its evaluation of each inference could take into account a great variety of information in the tree, including its entire yield (string), not just the information in nearby nodes.

For both models, the search for the optimal tree was organized by an agenda, as is typically done for tree inference algorithms. For efficiency, we used a chart, and

pruned items whose score was less than  $10^{-3}$  times the score of the best item in the same chart cell. We also pruned items from cells whenever the number of items in the same cell exceeded 40. Our entire tree transduction algorithm can be viewed as translation by parsing (Melamed, 2004) where the source side of the output bitree was constrained by the input (source) tree.

We compared the generative and discriminative models by reading out the string encoded in their predicted trees, and comparing that string to the target sentence in the test corpus. In pilot experiments we used the BLEU measure commonly used for such comparisons (Papineni et al., 2002). To our surprise, BLEU reported unbelievably high accuracy for our discriminative transducer, exceeding the accuracy of Pharaoh even with a language model. Subsequently, we discovered that BLEU was incorrectly inflating our scores by internally retokenizing our French output. This behavior, together with the growing evidence against using BLEU for syntax-aware MT (Callison-Burch et al., 2006), convinced us to use the more transparent precision, recall, and F-measure, as computed by GTM (Turian et al., 2003). With the exponent set to 1.0, the F-measure is essentially the unigram overlap ratio, except it avoids double-counting. With a higher exponent, the F-measure accounts for overlap of all n-grams (i.e., for all values of n), again without double-counting.

During testing, we compared two kinds of input parse trees for each kind of tree transducer. The first kind was generated by the parser of Bikel (2004). The second kind was generated by the parser of Turian and Melamed (2006), which was trained in a purely discriminative manner. Table 7.5 shows the results. The discriminatively trained transducer far outperformed the generatively trained transducer on all measures, at a statistical significance level of 0.001 using the Wilcoxon signed ranks test. In addition, the discriminatively trained transducer performed better when it started with parse trees from a purely discriminative parser.

---

## 7.5 Conclusion

We have presented a method for training all the parameters of a syntax-aware statistical machine translation system in a discriminative manner. The system outperforms a generative syntax-aware baseline. We have not yet added all the standard information sources that are necessary for a state-of-the-art MT system, but the scalability of our system suggests that we have overcome the main obstacle to doing so.

Our next step will be to generalize the tree transducer into a bitree transducer, so that it can modify the target side of the bitree after it is inferred from the source side. In particular, we shall add a new type of inference to predict additional words on the target side, starting from words that are already there, and from all the information available on the source side (cf. Toutanova and Suzuki, 2007). This kind of inference will enable the transducer to translate strings with fewer words into strings with more words. Item costs assigned by a target language model will

likely be an important source of information for this kind of inference, just as it is in most other kinds of MT systems.

---

### Acknowledgments

We would like to thank Yoshua Bengio, Léon Bottou, Patrick Haffner, Fernando Pereira, Christopher Pike, Cynthia Rudin, the anonymous reviewers, and the editors for their helpful comments and constructive criticism. This research was sponsored by NSF grants #0238406 and #0415933. The work described in this chapter was carried out while all of the authors were at New York University.