Scalable Purely-Discriminative Training for Word and Tree Transducers

Benjamin Wellington, Joseph Turian, Chris Pike, and I. Dan Melamed *
Computer Science Department
New York University
{lastname}@cs.nyu.edu

Abstract

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. The present study makes progress towards 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 sub-tasks is near the state of the art. Our source code is downloadable from http://nlp.cs.nyu.edu/GenPar/.

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. The present study makes progress towards 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 sub-tasks is near the state of the art. The core of our proposal is described in Section 2, and evaluated in Section 3.

Several studies have explored discriminative training for machine translation, following the method of Och and Ney (2002). Several authors have even applied this method to syntax-aware systems (Chiang, 2005; Quirk et al., 2005). However, Och's method applies discriminative training only to a handful of meta-parameters that are used to combine information from a variety of sub-models. The sub-models may or may not be optimized for the same objective as the meta-parameters. On a task like machine translation of common languages, where training data is abundant, a more elegant and more accurate system might be obtained by training all parts of the system discriminatively (Ng and Jordan, 2002). Foster (2000) built such a system, but could not fully exploit the benefits of discriminative training for lack of a suitable regularization scheme.

2 Training Method

2.1 The Training Set

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. Our method for predicting bitrees is to make a sequence of multiclass classification decisions called **inferences**. Each inference predicts an unstructured part of the eventual bitree. Thus, to train a model for predicting bitrees, it is sufficient to train it to predict correct inferences.

* Many thanks to Leon Bottou for help with LaSVM. This research was supported by NSF grants #0238406 and #0415933.

Proceedings of the 7th Conference of the Association for Machine Translation in the Americas, pages 251-260,
Cambridge, August 2006.

To generate training examples for predicting correct inferences, each training bitree t is decomposed into a partially-ordered set ("poset") of inferences, which jointly predict t and no other bitree. Next, a completely ordered sequence of correct inferences is chosen randomly, among all complete orderings that are consistent with the poset. All the inferences that can possibly follow each prefix of the correct sequence become part of the training set. All but one of the inferences generated for each prefix will be incorrect. 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 disadvantage of this approach is that it does not teach the model to recover from mistakes.

Our approach to training multiclass classifiers is to decompose each *multiclass* training inference into a set of training examples of *binary* classification decisions, all but one of which are negative examples. For example, one binary decision might be about whether a given source word should translate to a given target word. Another might be whether two constituents should switch order in translation. After the decomposition from multiclass to binary, the training set I consists of training examples i, where each i is 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 $y(i) = -1$ if not.

2.2 Objective Function

The training algorithm induces a hypothesis $h_\Theta(i)$, which is a real-valued example scoring function, parameterized by a real vector Θ. Θ has one entry for each possible feature f. In the present work, h_Θ is a linear model:

$$h_\Theta(i) = \Theta \cdot X(i) = \sum_f \Theta_f \cdot X_f(i) \qquad (1)$$

The sign of $h_\Theta(i)$ predicts the y-value of i and the magnitude gives the confidence in this prediction.

The training algorithm adjusts Θ to minimize the **objective** function, which is the expected risk $R_\Theta(I)$ over training set I:

$$R_\Theta(I) = L_\Theta(I) + \lambda \cdot \Omega_\Theta \qquad (2)$$

L_Θ above is the **loss** function and Ω_Θ is the **regularization** term, which penalizes complex models to reduce overfitting and generalization error. λ is a parameter that controls the strength of the regularizer.

In principle, L_Θ can be any loss function, but in the present work we use the log-loss (e.g., Collins et al. (2002)). Let the **margin** of inference i under the current model be

$$\mu_\Theta(i) = y(i) \cdot h_\Theta(i). \qquad (3)$$

Then the log-loss of i is

$$\sigma_\Theta(i) = \ln(1 + \exp(-\mu_\Theta(i))) \qquad (4)$$

and the loss of a set of inferences is just the sum of their individual losses:

$$L_\Theta(I) = \sum_{i \in I} \sigma_\Theta(i) \qquad (5)$$

We also use ℓ_1 regularization:

$$\Omega_\Theta = \sum_f |\Theta_f|. \qquad (6)$$

This choice of objective R_Θ was motivated by Ng (2004), who suggested that, given a learning setting where the number of irrelevant features is exponential in the number of training examples, we can still learn effectively by minimizing the ℓ_1-regularized log-loss. On the other hand, Ng (2004) suggested that the following algorithms will overfit in this setting: unregularized logistic regression, logistic regression with an ℓ_2 penalty (i.e. a Gaussian prior), SVMs using most kernels, multilayer neural nets trained by back-propagation, and the perceptron algorithm. This list includes most of the discriminative learning algorithms commonly used in NLP.

Learning in an exponentially-sized feature space is the very setting we have in mind. It is time-consuming and error-prone to do feature selection by hand for problems as complex as MT. We prefer to start with a very large set of all the features that we can think of, and let the learning algorithm find the useful ones. *A priori*, we define only a set A of simple **atomic** features (given in Section 3). The learning algorithm then induces **compound** features, each of which is a conjunction of possibly negated atomic features. Each atomic feature can

```
procedure TRAIN(I)
    ensemble ← ∅
    λ ← ∞
    repeat
        t ← tree with one (root) node
        GrowTree(λ, t, I)
        append t to ensemble
        measure tuning set loss
    until tuning set loss stops decreasing

procedure GROWTREE(λ, t, I)
    while the root of t cannot be split do
        decay regularization parameter λ
    while some leaf in t can be split do
        split the leaf to maximize gain
    percolate every i ∈ I to a leaf node
    for each leaf n in t do
        update Θ_φ(n) to minimize R_Θ
```

Figure 1: Outline of training algorithm.

have one of three values (yes/no/don't care), so the size of the compound feature space is $3^{|A|}$, exponential in the number of atomic features. In our experiments, it was also exponential in the number of training examples, because in our experiments $|A| > |I|$.

2.3 Boosting ℓ_1-Regularized Decision Trees

We use an ensemble of confidence-rated decision trees (Schapire and Singer, 1999) to represent h_Θ.[1] The path from the root to a node n in a decision tree corresponds to a compound feature f, and we write $\varphi(n) = f$. Decision trees partition the example space so that each example i falls into exactly one leaf of a decision tree. An example i percolates down to node n iff $X_a = 1$ for all atomic features a such that either a or $\neg a$ is part of $\varphi(n)$. To score an example i using a decision tree, we percolate the example's features $X(i)$ down to a leaf n and return the confidence $\Theta_{\varphi(n)}$. The score $h_\Theta(i)$ given to an example i by the whole ensemble is the sum of the confidences returned by the trees in the ensemble.

Figure 1 presents our training algorithm. At the beginning of training, the ensemble is empty, $\Theta = \mathbf{0}$, and the regularization hyper-parameter λ is set to ∞. The main loop runs until no further accuracy improvement can be found on a development data set. Each iteration of the main loop initializes a trivial decision tree, consisting of only a root node. It then calls the GROWTREE procedure, which can grow the trivial tree into a non-trivial one. The tree is then appended to the ensemble.

Each invocation of GROWTREE starts by decaying λ until it is low enough to permit the root of t to be split. The purpose of this step is to optimize the hyper-parameter λ concurrently with the rest of the model parameters, in contrast to the common practice of optimizing such hyper-parameters independently and/or heuristically. After λ gets low enough, GROWTREE chooses some compound features that have high magnitude gradient with respect to the objective function (see below). The result is a new decision tree whose leaves represent those compound features.[2] After the tree is built, GROWTREE percolates the training examples down to their appropriate leaf nodes. It then chooses for each leaf node n the parameter $\Theta_{\varphi(n)}$ that minimizes the objective over the examples in that leaf. A convenient property of decision trees is that each example will fall into exactly one leaf of a given tree. Therefore, the leaf confidences can be directly optimized independently of each other using a line search over the objective. In this manner, compound feature selection is performed incrementally during training, as opposed to *a priori*.

Our strategy for feature selection is a variant of steepest descent (Perkins et al., 2003). A natural criterion for choosing features is the degree to which changing a feature parameter Θ_f can change the objective, i.e. the absolute value of the gradient of the objective with respect to the parameter. Therefore, we define the gain so that it is never negative:

$$G_\Theta(I; f) = \max\left(0, \left|\frac{\partial L_\Theta(I)}{\partial \Theta_f}\right| - \lambda\right). \quad (7)$$

To compute the gain, we note that

$$\frac{\partial L_\Theta(I)}{\partial \Theta_f} = \sum_{i \in I} \frac{\partial \sigma_\Theta(i)}{\partial \Theta_f} = \sum_{i \in I} \frac{\partial \sigma_\Theta(i)}{\partial \mu_\Theta(i)} \cdot \frac{\partial \mu_\Theta(i)}{\partial \Theta_f}. \quad (8)$$

[1] Turian and Melamed (2005) found that training progressed more quickly and resulted in more accurate classifiers when using full decision trees rather than decision stumps.

[2] A compound feature can appear in more than one tree.

The boosting literature usually refers to the negative of the first factor above as the **weight** of an example. It can be computed by differentiating Equation 4:

$$w_\Theta(i) = -\frac{\partial \sigma_\Theta(i)}{\partial \mu_\Theta(i)} = \frac{1}{1 + \exp(\mu_\Theta(i))} \quad (9)$$

We also have from Equations 1 and 3 that

$$\frac{\partial \mu_\Theta(i)}{\partial \Theta_f} = y(i) \cdot X_f(i) \quad (10)$$

Recall that $X_f(i)$ is either 0 or 1. Combining Equations 8–10 gives:

$$\frac{\partial L_\Theta(I)}{\partial \Theta_f} = -\sum_{\substack{i \in I \\ X_f(i)=1}} y(i) \cdot w_\Theta(i) \quad (11)$$

We grow each decision tree by recursively splitting each node by a feature that will allow us to maximize the gain in Equation 7. For node n with $\varphi(n) = f$, the best atomic splitting feature is

$$\hat{a} = \arg\max_{a \in A} \left[G_\Theta(I; f \wedge a) + G_\Theta(I; f \wedge \neg a) \right] \quad (12)$$

After finding $\hat{a}$, we decide whether n should be split. If $G_\Theta(I; f \wedge \hat{a}) + G_\Theta(I; f \wedge \neg \hat{a}) > G_\Theta(I; f)$ then n spawns child nodes n_1 and n_2, with $\varphi(n_1) = f \wedge \hat{a}$ and $\varphi(n_2) = f \wedge \neg \hat{a}$. Otherwise, the current tree is likely to reduce loss more quickly if n were left unsplit, so n becomes a leaf node of the decision tree and $\Theta_{\varphi(n)}$ becomes one of the values to be optimized during the parameter update step. To our knowledge, this is a novel way to use regularization to control the depth of a decision tree. Using ℓ_1 regularization, in particular, keeps the trees relatively small and the ensembles relatively sparse. If no root node split has positive gain, then training has converged for the current hyper-parameter λ.

3 Experiments

3.1 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 1.

	sent.	English words		French words	
	pairs	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

Table 1: Data sizes in 000's.

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.

3.2 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 con-

sistent 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 multi-word "phrases" as possible translations, because we do not yet understand how methods for compiling such phrases interact with our discriminative training methods and our objective function. In future work, we intend to explore discriminative phrase induction techniques. In the present 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.[3] 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 2. The second was a method similar to Vickrey et al. (2005). Both of these methods use logistic regression. Their main difference is that the second method used ℓ_2 regularization, but the first method used ℓ_1 The third method was LaSVM (Bordes et al., 2005), an online SVM algorithm designed for large datasets.

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 2-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 4 "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 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 4 configurations. The best configurations of the two logistic regression methods far exceed the baseline, but otherwise they were statistically indistinguishable. The

	W	W+D	W+C	W+D+C
10,000 training sentences — baseline = 47.54				
ℓ_2	54.09	54.33	52.36	52.88
ℓ_1	53.96	54.13	53.29	53.75
LaSVM	53.38	51.93	49.13	50.71
pruned ℓ_2	47.37	46.01	46.68	45.01
ℓ_1 size	54.1K	41.7K	37.8K	38.7K
ℓ_2 size	1.67M	2.51M	5.63M	6.47M
pruned ℓ_2 size	54.1K	41.7K	37.8K	38.7K
100,000 training sentences — baseline = 51.94				
ℓ_1	62.00	62.42	61.98	62.40
ℓ_1 size	736K	703K	316K	322K

Table 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. ℓ_1 size is the number of compound feature types.

[3] David Vickrey (p.c.) informed us that Vickrey et al. (2005) omitted punctuation and function words, which are the most difficult in this task.

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 ℓ_2-regularized models were bigger than the ℓ_1-regularized models by two orders of magnitude. The ℓ_2-regularized models grew in size to accommodate each new feature type. In contrast, the ℓ_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 run-time complexity.

To scale up to 100,000 sentence pairs of training data with just the window features, the ℓ_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 heaviest feature weights to zero. We tried this on 10,000 sentence pairs of training data. The number of features allowed to remain active in each ℓ_2 classifier was the number of active features in the ℓ_1 classifier. Table 2 shows the accuracy of these "pruned" ℓ_2-regularized classifiers on the development set, when trained on 10,000 sentence pairs. With the playing field leveled, the ℓ_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, C. Tillmann informed us (p.c.) 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 ℓ_1 regularized log-loss for the rest of our experiments. Table 2 shows the accuracy and model size of the ℓ_1-regularized classifier on the development set, when trained on 100,000 sentence pairs, using each of the 4 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.

3.3 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 4 configurations of our ℓ_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 4 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 confuse our study with heuristic methods, so we avoided "phrase" induction, word class induction, non-discriminatively trained target language models, etc. On the other hand, modern MT systems

	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 3: (P)recision, (R)ecall and (F)-measure for bag transduction of the development set. The discriminative transducers were trained with ℓ_1 regularization.

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.

	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

Table 4: (P)recision, (R)ecall and (F)-measure of bag transducers on the test set.

The results are in Table 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 1-to-1 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 significantly higher than Pharaoh's when they compete on a level playing field (without a language model). Table 4 shows the accuracy of the best models on the test set, where the numbers closely follow those on the development set.

This result suggests that it might not be necessary to induce "phrases" on the source side[4]. 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 (but see the next section). The only advantage of phrases on this task is in proposing more words on the target side, which eliminates the 1-to-1 upper bound on recall.

3.4 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 3.2. 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 it 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

[4]Quirk and Menezes (2006) offer additional evidence for this hypothesis.

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 (Collins et al., 2005; Xia and McCord, 2004).

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 6 possible permutations, requiring a 6-class classifier. After binarization, the same 6 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.

To make the tree transducer deterministic, we add a procedure that searches for the tree with minimum total loss, among all possible output trees:

$$\widehat{T} = \arg\min_{T \in \mathcal{T}} \sum_{i \in T} l(i) \qquad (13)$$

where $\mathcal{T}$ is the set of all possible output trees, $i \in T$ are the inferences used to build the tree T and $l(i)$ is the loss associated with inference i. We compared two models of inference loss —one generative and one discriminative.

The generative model was based on a top-down tree transducer (Comon et al., 1997) 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 π be the function that maps every node to its parent, and let η be the function that maps every target node to 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

$$\Pr(T) = \Pr(n_0|\eta(n_0)) \cdot \prod_{n \neq n_0 \in T} \Pr(n|\pi(n), \eta(n))$$

For the generative model, the loss of an inference i is the negative logarithm of the probability of the node $n(i)$ that it infers:

$$l(i) = -\log \Pr[n(i)|\pi(n(i)), \eta(n(i))]. \qquad (14)$$

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 2, with $l(i) = \sigma_\Theta(i)$. 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 3.2. 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,

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	**62.36**	39.06	**48.04**	**28.02**	17.55	**21.59**
	Pharaoh (w/o LM)	32.19	**54.62**	40.51	12.37	**20.99**	15.57

Table 5: (P)recision, (R)ecall, and (F)-measure of transducers using 100,000 sentence pairs of training data.

monolexical, and unlexicalized forms. These kinds of feature combinations are very difficult to model in the traditional generative framework. 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. In principle, our step-wise approach to structured inference is also more flexible than the approach of Taskar et al. (2004), because we are not limited to objective functions that can be decomposed along subsets of the relevant features.

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 bi-tree 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 re-tokenizing 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 using the method of Section 2. Table 5 shows the results. The discriminatively-trained transducer far outperformed the generatively trained transducer on both precision and recall. In addition, the discriminatively-trained transducer performed better when it started with parse trees from a purely discriminative parser. To our knowledge, these are the first reported results for a syntax-driven SMT system that makes no use of generative models.

4 Conclusion

We have presented a method for training a syntax-aware statistical machine translation system in a fully discriminative manner. The system outperforms a generative baseline, despite not using the standard trick of bootstrapping from a generative model. 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 for doing so. Our source code is downloadable from `http://nlp.cs.nyu.edu/GenPar/`.

References

D. Bikel. 2004. A distributional analysis of a lexicalized statistical parsing model. In *EMNLP*.

A. Bordes, S. Ertekin, J. Weston, and L. Bottou. 2005. Fast kernel classifi ers with online and active learning. *Journal of Machine Learning Research* 6.

A. Burbank, M. Carpuat, S. Clark, M. Dreyer, P. Fox, D. Groves, K. Hall, M. Hearne, I. D. Melamed, Y. Shen, A. Way, B. Wellington, and D. Wu. 2005. Final report on statistical machine translation by parsing. Technical report, Johns Hopkins University Center for Speech and Language Processing. `http://www.clsp.jhu.edu/ws2005/groups/statistical/report.html`.

C.Callison-Burch, M. Osborne, and P. Koehn. 2006. Re-evaluating the role of Bleu in machine translation research. In *EACL*.

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

M. Collins, R. Schapire, and Y. Singer. 2002. Logistic regression, AdaBoost and Bregman distances. *Machine Learning*, 48(1-3).

M. Collins, P. Koehn, and I. Kucerova. 2005. Clause restructuring for statistical machine translation. In *ACL*.

G. Foster. 2000. A maximum entropy / minimum divergence translation model. In *ACL*.

M. Galley, M. Hopkins, K. Knight, and D. Marcu. 2004. What's in a translation rule? In *HLT-NAACL*.

J. Graehl and K. Knight. 2004. Training tree transducers. In *HLT-NAACL*.

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

P. Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *MT Summit X*.

M. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2).

I. Dan Melamed. 2004. Statistical machine translation by parsing. In *ACL*.

A. Ng and M. Jordan. 2002. On discriminative vs. generative classifi ers: A comparison of logistic regression and naive bayes. In *NIPS*.

A. Ng. 2004. Feature selection, ℓ_1 vs. ℓ_2 regularization, and rotational invariance. In *ICML*.

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

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

K. Papineni, S. Roukos, T. Ward, and W. Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *ACL*.

S. Perkins, K. Lacker, and J. Theiler. 2003. Grafting: Fast, incremental feature selection by gradient descent in function space. *Journal of Machine Learning Research*, 3.

C. Quirk and A. Menezes. 2006. Do we need phrases? Challenging the conventional wisdom in statistical machine translation. In *NAACL*.

C. Quirk, A. Menezes, and C. Cherry. 2005. Dependency treelet translation: Syntactically informed phrasal SMT. In *ACL*

R. Schapire and Y. Singer. 1999. Improved boosting using confi dence-rated predictions. *Machine Learning*, 37(3).

B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. 2004. Max-margin parsing. In *EMNLP*.

A. Tewari and P. Bartlett. 2005. On the consistency of multiclass classifi cation methods. In *COLT*.

C. Tillmann and T. Zhang. 2005. A localized prediction model for statistical machine translation. In *ACL*.

J. Turian and I. D. Melamed. 2005. Constituent parsing by classifi cation. In *IWPT*.

J. Turian and I. D. Melamed. 2006. Advances in discriminative parsing. In *COLING/ACL*.

J. Turian, L. Shen, and I. D. Melamed. 2003. Evaluation of machine translation and its evaluation. In *MT Summit IX*.

D. Vickrey, L. Biewald, M. Teyssier, and D. Koller. 2005. Word-sense disambiguation for machine translation. In *EMNLP*.

D. Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3).

F. Xia and M. McCord. 2004. Improving a statistical MT system with automatically learned rewrite patterns. In *COLING*.

H. Zhang and D. Gildea. 2005. Stochastic lexicalized inversion transduction grammar for alignment. In *ACL*.