A Localized Prediction Model for Statistical Machine Translation

Christoph Tillmann and Tong Zhang
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598 USA
{ctill,tzhang}@us.ibm.com

Abstract

In this paper, we present a novel training method for a localized phrase-based prediction model for statistical machine translation (SMT). The model predicts blocks with orientation to handle local phrase re-ordering. We use a maximum likelihood criterion to train a log-linear block bigram model which uses real-valued features (e.g. a language model score) as well as binary features based on the block identities themselves, e.g. block bigram features. Our training algorithm can easily handle millions of features. The best system obtains a 18.6 % improvement over the baseline on a standard Arabic-English translation task.

1 Introduction

In this paper, we present a block-based model for statistical machine translation. A block is a pair of phrases which are translations of each other. For example, Fig. 1 shows an Arabic-English translation example that uses 4 blocks. During decoding, we view translation as a block segmentation process, where the input sentence is segmented from left to right and the target sentence is generated from bottom to top, one block at a time. A monotone block sequence is generated except for the possibility to swap a pair of neighbor blocks. We use an orientation model similar to the lexicalized block re-ordering model in (Tillmann, 2004; Och et al., 2004): to generate a block b with orientation o relative to its predecessor block b'. During decoding, we compute the probability $P(b_1^n, o_1^n)$ of a block sequence b_1^n with orientation o_1^n as a product of block bigram probabilities:

$$P(b_1^n, o_1^n) \approx \prod_{i=1}^{n} p(b_i, o_i | b_{i-1}, o_{i-1}), \qquad (1)$$

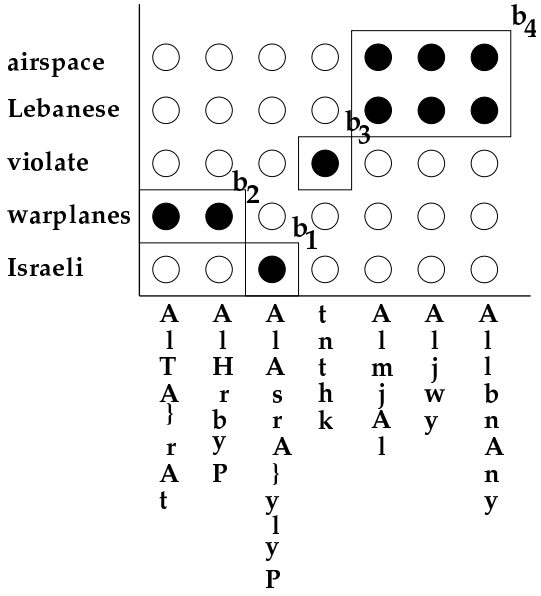

Figure 1: An Arabic-English block translation example, where the Arabic words are romanized. The following orientation sequence is generated: $o_1 = N, o_2 = L, o_3 = N, o_4 = R$.

where b_i is a block and $o_i \in \{L(\text{eft}), R(\text{ight}), N(\text{eutral})\}$ is a three-valued orientation component linked to the block b_i (the orientation o_{i-1} of the predecessor block is currently ignored.). Here, the block sequence with orientation (b_1^n, o_1^n) is generated under the restriction that the concatenated source phrases of the blocks b_i yield the input sentence. In modeling a block sequence, we emphasize adjacent block neighbors that have **Right** or **Left** orientation. Blocks with neutral orientation are supposed to be less strongly 'linked' to their predecessor block and are handled separately. During decoding, most blocks have right orientation $(o = R)$, since the block translations are mostly monotone.

Proceedings of the 43rd Annual Meeting of the ACL, pages 557–564,
Ann Arbor, June 2005.

The focus of this paper is to investigate issues in discriminative training of decoder parameters. Instead of directly minimizing error as in earlier work (Och, 2003), we decompose the decoding process into a sequence of local decision steps based on Eq. 1, and then train each local decision rule using convex optimization techniques. The advantage of this approach is that it can easily handle a large amount of features. Moreover, under this view, SMT becomes quite similar to sequential natural language annotation problems such as part-of-speech tagging, phrase chunking, and shallow parsing.
The paper is structured as follows: Section 2 introduces the concept of block orientation bigrams. Section 3 describes details of the localized log-linear prediction model used in this paper. Section 4 describes the online training procedure and compares it to the well known perceptron training algorithm (Collins, 2002). Section 5 shows experimental results on an Arabic-English translation task. Section 6 presents a final discussion.

2 Block Orientation Bigrams

This section describes a phrase-based model for SMT similar to the models presented in (Koehn et al., 2003; Och et al., 1999; Tillmann and Xia, 2003). In our paper, phrase pairs are named blocks and our model is designed to generate block sequences. We also model the position of blocks relative to each other: this is called orientation. To define block sequences with orientation, we define the notion of block orientation bigrams. Starting point for collecting these bigrams is a block set $\Gamma = \{b = (S,T) = (s_1^J\ ,\ t_1^I)\}$. Here, b is a block consisting of a source phrase S and a target phrase T. J is the source phrase length and I is the target phrase length. Single source and target words are denoted by s_j and t_i respectively, where $j = 1,\cdots,J$ and $i = 1,\cdots,I$. We will also use a special single-word block set $\Gamma_1 \subseteq \Gamma$ which contains only blocks for which $J = I = 1$. For the experiments in this paper, the block set is the one used in (Al-Onaizan et al., 2004). Although this is not investigated in the present paper, different blocksets may be used for computing the block statistics introduced in this paper, which may effect translation results.

For the block set Γ and a training sentence pair, we carry out a two-dimensional pattern matching algorithm to find adjacent matching blocks along with their position in the coordinate system defined by source and target positions (see Fig. 2). Here, we do not insist on a consistent block coverage as one would do during decoding. Among the matching blocks, two blocks b' and b are adjacent if the target phrases T and T' as well as the source phrases S and S' are adjacent. b' is predecessor of block b if b' and b are adjacent and b' occurs below b. A right adjacent successor block b is said to have right orientation $o = R$. A left adjacent successor block is said to have left orientation $o = L$. There are matching blocks b that have no predecessor, such a block has neutral orientation ($o = N$). After matching blocks for a training sentence pair, we look for adjacent block pairs to collect block bigram orientation events e of the type $e = (b', o, b)$. Our model to be presented in Section 3 is used to predict a **future** block orientation pair (b, o) given its predecessor block **history** b'. In Fig. 1, the following block orientation bigrams occur: $(\cdot, N, b_1)$,(b_1, L, b_2),$(\cdot, N, b_3)$,(b_3, R, b_4). Collecting orientation bigrams on all parallel sentence pairs, we obtain an orientation bigram list e_1^N:

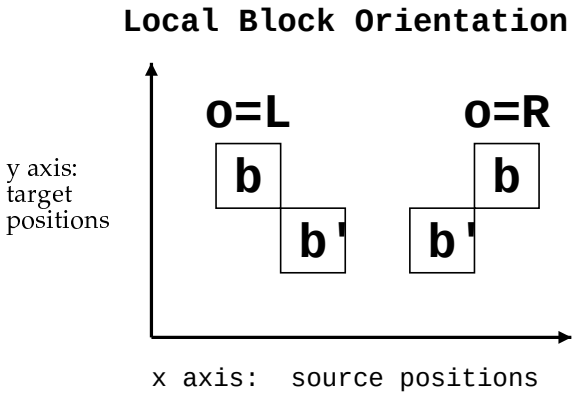

Figure 2: Block b' is the predecessor of block b. The successor block b occurs with either left $o = L$ or right $o = R$ orientation. 'left' and 'right' are defined relative to the x axis ; 'below' is defined relative to the y axis. For some discussion on global re-ordering see Section 6.

$$e_1^N \quad = \quad [e_1^{n_s}]_{s=1}^{S} = [\,(b_i^{'}, o_i, b_i)_{i=1}^{n_s}]_{s=1}^{S} \qquad (2)$$

Here, n_s is the number of orientation bigrams in the s-th sentence pair. The total number N of orientation bigrams $N = \sum_{s=1}^{S} n_s$ is about $N = 7.8$ million for our training data consisting of $S = 273\,000$ sentence pairs. The orientation bigram list is used for the parameter training presented in Section 3. Ignoring the bigrams with neutral orientation N reduces the list defined in Eq. 2 to about 5.0 million orientation bigrams. The **Neutral** orientation is handled separately as described in Section 5. Using the reduced orientation bigram list, we collect unigram orientation counts $N_o(b)$: how often a block occurs with a given orientation $o \in \{L, R\}$. $N_L(b) > 0.25 \cdot N_R(b)$ typically holds for blocks b involved in block swapping and the orientation model $p_o(b)$ is defined as:

$$p_o(b) \quad = \quad \frac{N_o(b)}{N_L(b) + N_R(b)}.$$

In order to train a block bigram orientation model as described in Section 3.2, we define a successor set $\delta_s(b')$ for a block b' in the s-th training sentence pair:

$$\delta_s(b') = \{ \text{ number of triples of type } (b', L, b) \text{ or type } (b', R, b) \in e_1^{n_s} \}$$

The successor set $\delta(b')$ is defined for each event in the list e_1^N. The average size of $\delta(b')$ is 1.5 successor blocks. If we were to compute a Viterbi block alignment for a training sentence pair, each block in this block alignment would have at most 1 successor: Blocks may have several successors, because we do not inforce any kind of consistent coverage during training.

During decoding, we generate a list of block orientation bigrams as described above. A DP-based beam search procedure identical to the one used in (Tillmann, 2004) is used to maximize over all oriented block segmentations (b_1^n, o_1^n). During decoding orientation bigrams (b', L, b) with left orientation are only generated if $N_L(b) \geq 3$ for the successor block b.

3 Localized Block Model and Discriminative Training

In this section, we describe the components used to compute the block bigram probability $p(b_i, o_i | b_{i-1}, o_{i-1})$ in Eq. 1. A block orientation pair $(o', b'; o, b)$ is represented as a feature-vector $f(b, o; b', o') \in \mathbb{R}^d$. For a model that uses all the components defined below, d is 6. As feature-vector components, we take the negative logarithm of some block model probabilities. We use the term 'float' feature for these feature-vector components (the model score is stored as a float number). Additionally, we use binary block features. The letters (a)-(f) refer to Table 1:

Unigram Models: we compute (a) the unigram probability $p(b)$ and (b) the orientation probability $p_o(b)$. These probabilities are simple relative frequency estimates based on unigram and unigram orientation counts derived from the data in Eq. 2. For details see (Tillmann, 2004). During decoding, the unigram probability is normalized by the source phrase length.

Two types of Trigram language model: (c) probability of predicting the first target word in the target clump of b_i given the final two words of the target clump of b_{i-1}, (d) probability of predicting the rest of the words in the target clump of b_i. The language model is trained on a separate corpus.

Lexical Weighting: (e) the lexical weight $p(S \mid T)$ of the block $b = (S, T)$ is computed similarly to (Koehn et al., 2003), details are given in Section 3.4.

Binary features: (f) binary features are defined using an indicator function $f(b, b')$ which is 1 if the block pair (b, b') occurs more often than a given threshold N, e.g $N = 2$. Here, the orientation o between the blocks is ignored.

$$f(b, b') = \begin{cases} 1 & N(b, b') > N \\ 0 & \text{else} \end{cases} \quad (3)$$

3.1 Global Model

In our linear block model, for a given source sentence s, each translation is represented as a sequence of block/orientation pairs $\{b_1^n, o_1^n\}$ consistent with the source. Using features such as those described above, we can parameterize the probability of such a sequence as $P(b_1^n, o_1^n | w, s)$, where w is a vector of unknown model parameters to be estimated from the training data. We use a log-linear probability model and maximum likelihood training— the parameter w is estimated by maximizing the joint likelihood over all sentences. Denote by $\Delta(s)$ the set of possible block/orientation sequences $\{b_1^n, o_1^n\}$ that are consistent with the source sentence s, then a log-linear probability model can be represented as

$$P(b_1^n, o_1^n | w, s) = \frac{\exp(w^T f(b_1^n, o_1^n))}{Z(s)}, \quad (4)$$

where $f(b_1^n, o_1^n)$ denotes the feature vector of the corresponding block translation, and the partition function is:

$$Z(s) = \sum_{\{b'^m_1, o'^m_1\} \in \Delta(s)} \exp(w^T f(b'^m_1, o'^m_1)).$$

A disadvantage of this approach is that the summation over $\Delta(s)$ can be rather difficult to compute. Consequently some sophisticated approximate inference methods are needed to carry out the computation. A detailed investigation of the global model will be left to another study.

3.2 Local Model Restrictions

In the following, we consider a simplification of the direct global model in Eq. 4. As in (Tillmann, 2004), we model the block bigram probability as $p(b_i, o_i \in \{L, R\} | b_{i-1}, o_{i-1})$ in Eq. 1. We distinguish the two cases (1) $o_i \in \{L, R\}$, and (2) $o_i = N$. Orientation is modeled only in the context of immediate neighbors for blocks that have left or right orientation. The log-linear model is defined as:

$$p(b, o \in \{L, R\} \mid b', o'; w, s) = \frac{\exp(w^T f(b, o; b', o'))}{Z(b', o'; s)}, \quad (5)$$

where s is the source sentence, $f(b, o; b', o')$ is a locally defined feature vector that depends only on the current and the previous oriented blocks (b, o) and (b', o'). The features were described at the beginning of the section. The partition function is given by

$$Z(b', o'; s) = \sum_{(b,o) \in \Delta(b', o'; s)} \exp(w^T f(b, o; b', o')). \quad (6)$$

The set $\Delta(b', o'; s)$ is a restricted set of possible successor oriented blocks that are consistent with the current block position and the source sentence s, to be described in the following paragraph. Note that a straightforward normalization over all block orientation pairs in Eq. 5 is not feasible: there are tens of millions of possible successor blocks b (if we do not impose any restriction). For each block $b = (S, T)$, aligned with a source sentence s, we define a source-induced alternative set:

$$\Gamma(b) = \{ \text{ all blocks } b'' \in \Gamma \text{ that share an identical source phrase with } b \}$$

The set $\Gamma(b)$ contains the block b itself and the block target phrases of blocks in that set might differ. To restrict the number of alternatives further, the elements of $\Gamma(b)$ are sorted according to the unigram count $N(b'')$ and we keep at most the top 9 blocks for each source interval s. We also use a modified alternative set $\Gamma_1(b)$, where the block b as well as the elements in the set $\Gamma_1(b)$ are single word blocks. The partition function is computed slightly differently during training and decoding:

Training: for each event (b', o, b) in a sentence pair s in Eq. 2 we compute the successor set $\delta_s(b')$. This defines a set of 'true' block successors. For each true successor b, we compute the alternative set $\Gamma(b)$. $\Delta(b', o'; s)$ is the union of the alternative set for each successor b. Here, the orientation o from the true successor b is assigned to each alternative in $\Gamma(b)$. We obtain on the average 12.8 alternatives per training event (b', o, b) in the list e_1^N.

Decoding: Here, each block b that matches a source interval following b' in the sentence s is a potential successor. We simply set $\Delta(b', o'; s) = \Gamma(b)$. Moreover, setting $Z(b', o'; s) = 0.5$ during decoding does not change performance: the list $\Gamma(b)$ just restricts the possible target translations for a source phrase.

Under this model, the log-probability of a possible translation of a source sentence s, as in Eq. 1, can be written as

$$\begin{aligned} \ln P(b_1^n, o_1^n | w, s) = & \qquad (7) \\ = & \sum_{i=1}^{n} \ln \frac{\exp(w^T f(b_i, o_i; b_{i-1}, o_{i-1}))}{Z(b_{i-1}, o_{i-1}; s)}. \end{aligned}$$

In the maximum-likelihood training, we find w by maximizing the sum of the log-likelihood over observed sentences, each of them has the form in Eq. 7. Although the training methodology is similar to the global formulation given in Eq. 4, this localized version is computationally much easier to manage since the summation in the partition function $Z(b_{i-1}, o_{i-1}; s)$ is now over a relatively small set of candidates. This computational advantage is the main reason that we adopt the local model in this paper.

3.3 Global versus Local Models

Both the global and the localized log-linear models described in this section can be considered as maximum-entropy models, similar to those used in natural language processing, e.g. maximum-entropy models for POS tagging and shallow parsing. In the parsing context, global models such as in Eq. 4 are sometimes referred to as *conditional random field* or CRF (Lafferty et al., 2001).

Although there are some arguments that indicate that this approach has some advantages over localized models such as Eq. 5, the potential improvements are relatively small, at least in NLP applications. For SMT, the difference can be potentially more significant. This is because in our current localized model, successor blocks of different sizes are directly compared to each other, which is intuitively not the best approach (i.e., probabilities of blocks with identical lengths are more comparable). This issue is closely related to the phenomenon of multiple counting of events, which means that a source/target sentence pair can be decomposed into different oriented blocks in our model. In our current training procedure, we select one as the truth, while consider the other (possibly also correct) decisions as non-truth alternatives. In the global modeling, with appropriate normalization, this issue becomes less severe. With this limitation in mind, the localized model proposed here is still an effective approach, as demonstrated by our experiments. Moreover, it is simple both computationally and conceptually. Various issues such as the ones described above can be addressed with more sophisticated modeling techniques, which we shall be left to future studies.

3.4 Lexical Weighting

The lexical weight $p(S \mid T)$ of the block $b = (S, T)$ is computed similarly to (Koehn et al., 2003), but the lexical translation probability $p(s|t)$ is derived from the block set itself rather than from a word alignment, resulting in a simplified training. The lexical weight is computed as follows:

$$\begin{aligned} p(S \mid T) &= \prod_{j=1}^{J} \frac{1}{N_\Gamma(s_j, T)} \sum_{t=1}^{I} p(s_j \mid t_i) \\ p(s_j|t_i) &= \frac{N(b)}{\sum_{b' \in \Gamma_1(b)} N(b')} \end{aligned}$$

Here, the single-word-based translation probability $p(s_j \mid t_i)$ is derived from the block set itself. $b = (s_j, t_i)$ and $b' = (s_j, t_k)$ are single-word blocks, where source and target phrases are of length 1. $N_\Gamma(s_j, t_1^I)$ is the number of blocks $b_k = (s_j, t_k)$ for $k \in 1, \cdots, I$ for which $p(s_j|t_k) > 0.0$.

4 Online Training of Maximum-entropy Model

The local model described in Section 3 leads to the following abstract maximum entropy training formulation:

$$\hat{w} = \arg\min_{w} \sum_{i=1}^{m} \ln \frac{\sum_{j\in\Delta_i} \exp(w^T x_{i,j})}{\exp(w^T x_{i,y_i})}. \quad (8)$$

In this formulation, w is the weight vector which we want to compute. The set Δ_i consists of candidate labels for the i-th training instance, with the true label $y_i \in \Delta_i$. The labels here are block identities , Δ_i corresponds to the alternative set $\Delta(b', o'; s)$ and the 'true' blocks are defined by the successor set $\delta(b')$. The vector $x_{i,j}$ is the feature vector of the i-th instance, corresponding to label $j \in \Delta_i$. The symbol x is short-hand for the feature-vector $f(b, o; b', o')$. This formulation is slightly different from the standard maximum entropy formulation typically encountered in NLP applications, in that we restrict the summation over a subset Δ_i of all labels.

Intuitively, this method favors a weight vector such that for each i, $w^T x_{i,y_i} - w^T x_{i,j}$ is large when $j \neq y_i$. This effect is desirable since it tries to separate the correct classification from the incorrect alternatives. If the problem is completely separable, then it can be shown that the computed linear separator, with appropriate regularization, achieves the largest possible separating margin. The effect is similar to some multi-category generalizations of support vector machines (SVM). However, Eq. 8 is more suitable for non-separable problems (which is often the case for SMT) since it directly models the conditional probability for the candidate labels.

A related method is multi-category perceptron, which explicitly finds a weight vector that separates correct labels from the incorrect ones in a mistake driven fashion (Collins, 2002). The method works by examining one sample at a time, and makes an update $w \rightarrow w + (x_{i,y_i} - x_{i,j})$ when $w^T(x_{i,y_i} - x_{i,j})$ is not positive. To compute the update for a training instance i, one usually pick the j such that $w^T(x_{i,y_i} - x_{i,j})$ is the smallest. It can be shown that if there exist weight vectors that separate the correct label y_i from incorrect labels $j \in \Delta_i$ for all $j \neq y_i$, then the perceptron method can find such a separator. However, it is not entirely clear what this method does when the training data are not completely separable. Moreover, the standard mistake bound justification does not apply when we go through the training data more than once, as typically done in practice. In spite of some issues in its justification, the perceptron algorithm is still very attractive due to its simplicity and computational efficiency. It also works quite well for a number of NLP applications.

In the following, we show that a simple and efficient online training procedure can also be developed for the maximum entropy formulation Eq. 8. The proposed update rule is similar to the perceptron method but with a soft mistake-driven update rule, where the influence of each feature is weighted by the significance of its mistake. The method is essentially a version of the so-called *stochastic gradient descent method*, which has been widely used in complicated stochastic optimization problems such as neural networks. It was argued recently in (Zhang, 2004) that this method also works well for standard convex formulations of binary-classification problems including SVM and logistic regression. Convergence bounds similar to perceptron mistake bounds can be developed, although unlike perceptron, the theory justifies the standard practice of going through the training data more than once. In the non-separable case, the method solves a regularized version of Eq. 8, which has the statistical interpretation of estimating the conditional probability. Consequently, it does not have the potential issues of the perceptron method which we pointed out earlier. Due to the nature of online update, just like perceptron, this method is also very simple to implement and is scalable to large problem size. This is important in the SMT application because we can have a huge number of training instances which we are not able to keep in memory at the same time.

In stochastic gradient descent, we examine one training instance at a time. At the i-th instance, we derive the update rule by maximizing with respect to the term associated with the instance

$$L_i(w) = \ln \frac{\sum_{j\in\Delta_i} \exp(w^T x_{i,j})}{\exp(w^T x_{i,y_i})}$$

in Eq. 8. We do a gradient descent localized to this instance as $w \rightarrow w - \eta_i \frac{\partial}{\partial w} L_i(w)$, where $\eta_i > 0$ is a parameter often referred to as the learning rate. For Eq. 8, the update rule becomes:

$$w \rightarrow w + \eta_i \frac{\sum_{j\in\Delta_i} \exp(w^T x_{i,j})(x_{i,y_i} - x_{i,j})}{\sum_{j\in\Delta_i} \exp(w^T x_{i,j})}. \quad (9)$$

Similar to online algorithms such as the perceptron, we apply this update rule one by one to each training instance (randomly ordered), and may go-through data points repeatedly. Compare Eq. 9 to the perceptron update, there are two main differences, which we discuss below.

The first difference is the weighting scheme. Instead of putting the update weight to a single (most mistaken) feature component, as in the perceptron algorithm, we use a soft-weighting scheme, with each feature component j weighted by a factor $\exp(w^T x_{i,j}) / \sum_{k\in\Delta_i} \exp(w^T x_{i,k})$. A component j with larger $w^T x_{i,j}$ gets more weight. This effect is in principle similar to the perceptron update. The smoothing effect in Eq. 9 is useful for non-separable problems

since it does not force an update rule that attempts to separate the data. Each feature component gets a weight that is proportional to its conditional probability.

The second difference is the introduction of a learning rate parameter η_i. For the algorithm to converge, one should pick a decreasing learning rate. In practice, however, it is often more convenient to select a fixed $\eta_i = \eta$ for all i. This leads to an algorithm that approximately solve a regularized version of Eq. 8. If we go through the data repeatedly, one may also decrease the fixed learning rate by monitoring the progress made each time we go through the data. For practical purposes, a fixed small η such as $\eta = 10^{-5}$ is usually sufficient. We typically run forty updates over the training data. Using techniques similar to those of (Zhang, 2004), we can obtain a convergence theorem for our algorithm. Due to the space limitation, we will not present the analysis here.

An advantage of this method over standard maximum entropy training such as GIS (generalized iterative scaling) is that it does not require us to store all the data in memory at once. Moreover, the convergence analysis can be used to show that if m is large, we can get a very good approximate solution by going through the data only once. This desirable property implies that the method is particularly suitable for large scale problems.

5 Experimental Results

The translation system is tested on an Arabic-to-English translation task. The training data comes from the UN news sources. Some punctuation tokenization and some number classing are carried out on the English and the Arabic training data. In this paper, we present results for two test sets: (1) the devtest set uses data provided by LDC, which consists of $1\,043$ sentences with $25\,889$ Arabic words with 4 reference translations. (2) the blind test set is the MT03 Arabic-English DARPA evaluation test set consisting of 663 sentences with $16\,278$ Arabic words with also 4 reference translations. Experimental results are reported in Table 2: here cased BLEU results are reported on MT03 Arabic-English test set (Papineni et al., 2002). The word casing is added as post-processing step using a statistical model (details are omitted here).

In order to speed up the parameter training we filter the original training data according to the two test sets: for each of the test sets we take all the Arabic substrings up to length 12 and filter the parallel training data to include only those training sentence pairs that contain at least one out of these phrases: the 'LDC' training data contains about 273 thousand sentence pairs and the 'MT03' training data contains about 230 thousand sentence pairs. Two block sets are derived for each of the training sets using a phrase-pair selection algorithm similar to (Koehn et al., 2003; Tillmann and Xia, 2003). These block sets also include blocks that occur only once in the training data. Additionally, some heuristic filtering is used to increase phrase translation accuracy (Al-Onaizan et al., 2004).

5.1 Likelihood Training Results

We compare model performance with respect to the number and type of features used as well as with respect to different re-ordering models. Results for 9 experiments are shown in Table 2, where the feature types are described in Table 1. The first 5 experimental results are obtained by carrying out the likelihood training described in Section 3. Line 1 in Table 2 shows the performance of the baseline block unigram '*MON*' model which uses two 'float' features: the unigram probability and the boundary-word language model probability. No block re-ordering is allowed for the baseline model (a **monotone** block sequence is generated). The '*SWAP*' model in line 2 uses the same two features, but neighbor blocks can be swapped. No performance increase is obtained for this model. The '*SWAP* & *OR*' model uses an orientation model as described in Section 3. Here, we obtain a small but significant improvement over the baseline model. Line 4 shows that by including two additional 'float' features: the lexical weighting and the language model probability of predicting the second and subsequent words of the target clump yields a further significant improvement. Line 5 shows that including binary features and training their weights on the training data actually decreases performance. This issue is addressed in Section 5.2.

The training is carried out as follows: the results in line 1-4 are obtained by training 'float' weights only. Here, the training is carried out by running only once over 10 % of the training data. The model including the binary features is trained on the entire training data. We obtain about 3.37 million features of the type defined in Eq. 3 by setting the threshold $N = 3$. Forty iterations over the training data take about 2 hours on a single Intel machine. Although the online algorithm does not require us to do so, our training procedure keeps the entire training data and the weight vector w in about 2 gigabytes of memory.

For blocks with neutral orientation $o = N$, we train a separate model that does not use the orientation model feature or the binary features. E.g. for the results in line 5 in Table 2, the neutral model would use the features $(a), (c), (d), (e)$, but not (b) and (f). Here, the neutral model is trained on the neutral orientation bigram subsequence that is part of Eq. 2.

5.2 Modified Weight Training

We implemented the following variation of the likelihood training procedure described in Section 3, where we make use of the 'LDC' devtest set. First, we train a model on the 'LDC' training data using 5 float features and the binary features. We use this model to decode

Table 1: List of feature-vector components. For a description, see Section 3.

Description
(a) Unigram probability
(b) Orientation probability
(c) LM first word probability
(d) LM second and following words probability
(e) Lexical weighting
(f) Binary Block Bigram Features

Table 2: Cased BLEU translation results with confidence intervals on the MT03 test data. The third column summarizes the model variations. The results in lines 8 and 9 are for a cheating experiment: the float weights are trained on the test data itself.

	Re-ordering	Components	BLEU
1	'*MON*'	(a),(c)	32.3 ± 1.5
2	'*SWAP*'	(a),(c)	32.3 ± 1.5
3	'*SWAP* & *OR*'	(a),(b),(c)	33.9 ± 1.4
4	'*SWAP* & *OR*'	(a)-(e)	37.7 ± 1.5
5	'*SWAP* & *OR*'	(a)-(f)	37.2 ± 1.6
6	'*SWAP* & *OR*'	(a)-(e) (ldc devtest)	37.8 ± 1.5
7	'*SWAP* & *OR*'	(a)-(f) (ldc devtest)	38.2 ± 1.5
8	'*SWAP* & *OR*'	(a)-(e) (mt03 test)	39.0 ± 1.5
9	'*SWAP* & *OR*'	(a)-(f) (mt03 test)	39.3 ± 1.6

the devtest 'LDC' set. During decoding, we generate a 'translation graph' for every input sentence using a procedure similar to (Ueffing et al., 2002): a translation graph is a compact way of representing candidate translations which are close in terms of likelihood. From the translation graph, we obtain the $1\,000$ best translations according to the translation score. Out of this list, we find the block sequence that generated the top BLEU-scoring target translation. Computing the top BLEU-scoring block sequence for all the input sentences we obtain:

$$e_1^{N'} \quad = \quad [\,(b_i^{'}, o_i, b_i)_{i=1}^{n_{s'}}]_1^{S'}\,, \qquad (10)$$

where $N' \approx 9400$. Here, N' is the number of blocks needed to decode the entire devtest set. Alternatives for each of the events in $e_1^{N'}$ are generated as described in Section 3.2. The set of alternatives is further restricted by using only those blocks that occur in some translation in the $1\,000$-best list. The 5 float weights are trained on the modified training data in Eq. 10, where the training takes only a few seconds. We then decode the 'MT03' test set using the modified 'float' weights. As shown in line 4 and line 6 there is almost no change in performance between training on the original training data in Eq. 2 or on the modified training data in Eq. 10. Line 8 shows that even when training the float weights on an event set obtained from the test data itself in a cheating experiment, we obtain only a moderate performance improvement from 37.7 to 39.0. For the experimental results in line 7 and 9, we use the same five float weights as trained for the experiments in line 6 and 8 and keep them fixed while training the binary feature weights only. Using the binary features leads to only a minor improvement in BLEU from 37.8 to 38.2 in line 7. For this best model, we obtain a 18.6 % BLEU improvement over the baseline.

From our experimental results, we draw the following conclusions: (1) the translation performance is largely dominated by the 'float' features, (2) using the same set of 'float' features, the performance doesn't change much when training on training, devtest, or even test data. Although, we do not obtain a significant improvement from the use of binary features, currently, we expect the use of binary features to be a promising approach for the following reasons:

- The current training does not take into account the block interaction on the sentence level. A more accurate approximation of the global model as discussed in Section 3.1 might improve performance.
- As described in Section 3.2 and Section 5.2, for efficiency reasons alternatives are computed from source phrase matches only. During training, more accurate local approximations for the partition function in Eq. 6 can be obtained by looking at block translations in the context of translation sequences. This involves the computationally expensive generation of a translation graph for each training sentence pair. This is future work.
- As mentioned in Section 1, viewing the translation process as a sequence of local discussions makes it similar to other NLP problems such as POS tagging, phrase chunking, and also statistical parsing. This similarity may facilitate the incorporation of these approaches into our translation model.

6 Discussion and Future Work

In this paper we proposed a method for discriminatively training the parameters of a block SMT decoder. We discussed two possible approaches: global versus local. This work focused on the latter, due to its computational advantages. Some limitations of our approach have also been pointed out, although our experiments showed that this simple method can significantly improve the baseline model.

As far as the log-linear combination of float features is concerned, similar training procedures have been proposed in (Och, 2003). This paper reports the use of 8

features whose parameter are trained to optimize performance in terms of different evaluation criteria, e.g. BLEU. On the contrary, our paper shows that a significant improvement can also be obtained using a likelihood training criterion.

Our modified training procedure is related to the discriminative re-ranking procedure presented in (Shen et al., 2004). In fact, one may view discriminative reranking as a simplification of the global model we discussed, in that it restricts the number of candidate global translations to make the computation more manageable. However, the number of possible translations is often exponential in the sentence length, while the number of candidates in a typically reranking approach is fixed. Unless one employs an elaborated procedure, the candidate translations may also be very similar to one another, and thus do not give a good coverage of representative translations. Therefore the reranking approach may have some severe limitations which need to be addressed. For this reason, we think that a more principled treatment of global modeling can potentially lead to further performance improvements.

For future work, our training technique may be used to train models that handle global sentence-level reorderings. This might be achieved by introducing orientation sequences over phrase types that have been used in ((Schafer and Yarowsky, 2003)). To incorporate syntactic knowledge into the block-based model, we will examine the use of additional real-valued or binary features, e.g. features that look at whether the block phrases cross syntactic boundaries. This can be done with only minor modifications to our training method.

Acknowledgment

This work was partially supported by DARPA and monitored by SPAWAR under contract No. N66001-99-2-8916. The paper has greatly profited from suggestions by the anonymous reviewers.

References

Yaser Al-Onaizan, Niyu Ge, Young-Suk Lee, Kishore Papineni, Fei Xia, and Christoph Tillmann. 2004. IBM Site Report. In *NIST 2004 Machine Translation Workshop*, Alexandria, VA, June.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proc. EMNLP'02*.

Philipp Koehn, Franz-Josef Och, and Daniel Marcu. 2003. Statistical Phrase-Based Translation. In *Proc. of the HLT-NAACL 2003 conference*, pages 127–133, Edmonton, Canada, May.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML-01*, pages 282–289.

Franz-Josef Och, Christoph Tillmann, and Hermann Ney. 1999. Improved Alignment Models for Statistical Machine Translation. In *Proc. of the Joint Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC 99)*, pages 20–28, College Park, MD, June.

Och et al. 2004. A Smorgasbord of Features for Statistical Machine Translation. In *Proceedings of the Joint HLT and NAACL Conference (HLT 04)*, pages 161–168, Boston, MA, May.

Franz-Josef Och. 2003. Minimum Error Rate Training in Statistical Machine Translation. In *Proc. of the 41st Annual Conf. of the Association for Computational Linguistics (ACL 03)*, pages 160–167, Sapporo, Japan, July.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a Method for Automatic Evaluation of machine translation. In *Proc. of the 40th Annual Conf. of the Association for Computational Linguistics (ACL 02)*, pages 311–318, Philadelphia, PA, July.

Charles Schafer and David Yarowsky. 2003. Statistical Machine Translation Using Coercive Two-Level Syntactic Translation. In *Proc. of the Conf. on Empirical Methods in Natural Language Processing (EMNLP 03)*, pages 9–16, Sapporo, Japan, July.

Libin Shen, Anoop Sarkar, and Franz-Josef Och. 2004. Discriminative Reranking of Machine Translation. In *Proceedings of the Joint HLT and NAACL Conference (HLT 04)*, pages 177–184, Boston, MA, May.

Christoph Tillmann and Fei Xia. 2003. A Phrase-based Unigram Model for Statistical Machine Translation. In *Companian Vol. of the Joint HLT and NAACL Conference (HLT 03)*, pages 106–108, Edmonton, Canada, June.

Christoph Tillmann. 2004. A Unigram Orientation Model for Statistical Machine Translation. In *Companian Vol. of the Joint HLT and NAACL Conference (HLT 04)*, pages 101–104, Boston, MA, May.

Nicola Ueffing, Franz-Josef Och, and Hermann Ney. 2002. Generation of Word Graphs in Statistical Machine Translation. In *Proc. of the Conf. on Empirical Methods in Natural Language Processing (EMNLP 02)*, pages 156–163, Philadelphia, PA, July.

Tong Zhang. 2004. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *ICML 04*, pages 919–926.