Undirected Machine Translation with Discriminative Reinforcement Learning

Andrea Gesmundo
Google Inc.
andrea.gesmundo@gmail.com

James Henderson
Xerox Research Centre Europe
james.henderson@xrce.xerox.com

Abstract

We present a novel Undirected Machine Translation model of Hierarchical MT that is not constrained to the standard bottom-up inference order. Removing the ordering constraint makes it possible to condition on top-down structure and surrounding context. This allows the introduction of a new class of contextual features that are not constrained to condition only on the bottom-up context. The model builds translation-derivations efficiently in a greedy fashion. It is trained to learn to choose jointly the best action and the best inference order. Experiments show that the decoding time is halved and forest-rescoring is 6 times faster, while reaching accuracy not significantly different from state of the art.

1 Introduction

Machine Translation (MT) can be addressed as a structured prediction task (Brown et al., 1993; Yamada and Knight, 2001; Koehn et al., 2003). MT's goal is to learn a mapping function, f, from an input sentence, x, into $y = (t, h)$, where t is the sentence translated into the target language, and h is the hidden correspondence structure (Liang et al., 2006). In Hierarchical MT (HMT) (Chiang, 2005) the hidden correspondence structure is the synchronous-tree composed by instantiations of synchronous rules from the input grammar, G.

Statistical models usually define f as: $f(x) = \arg\max_{y \in \mathcal{Y}} \mathsf{Score}(x, y)$, where $\mathsf{Score}(x, y)$ is a function whose parameters can be learned with a specialized learning algorithm. In MT applications, it is not possible to enumerate all $y \in \mathcal{Y}$. HMT decoding applies pruning (e.g. Cube Pruning (Huang and Chiang, 2005)), but even then HMT has higher complexity than Phrase Based MT (PbMT) (Koehn et al., 2003). On the other hand, HMT improves over PbMT by introducing the possibility of exploiting a more sophisticated reordering model not bounded by a window size, and producing translations with higher syntactic-semantic quality. In this paper, we present the Undirected Machine Translation (UMT) framework, which retains the advantages of HMT and allows the use of a greedy decoder whose complexity is lower than standard quadratic beam-search PbMT.

UMT's fast decoding is made possible through even stronger pruning: the decoder chooses a single action at each step, never retracts that action, and prunes all incompatible alternatives to that action. If this extreme level of pruning was applied to the CKY-like beam-decoding used in standard HMT, translation quality would be severely degraded. This is because the bottom-up inference order imposed by CKY-like beam-decoding means that all pruning decisions must be based on a bottom-up approximation of contextual features, which leads to search errors that affect the quality of reordering and lexical-choice (Gesmundo and Henderson, 2011). UMT solves this problem by removing the bottom-up inference order constraint, allowing many different inference orders for the same tree structure, and learning the inference order where the decoder can be the most confident in its pruning decisions.

Removing the bottom-up inference order constraint makes it possible to condition on top-down structure and surrounding context. This undirected approach allows us to integrate contextual features such as the Language Model (LM) in a more flex-

Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, pages 10–19,
Gothenburg, Sweden, April 26-30 2014.

ible way. It also allows us to introduce a new class of undirected features. In particular, we introduce the Context-Free Factor (CFF) features. CFF features compute exactly and efficiently a bound on the context-free cost of a partial derivation's missing branches, thereby estimating the future cost of partial derivations. The new class of undirected features is fundamental for the success of a greedy approach to HMT, because the additional non-bottom-up context is sometimes crucial to have the necessary information to make greedy decisions.

Because UMT prunes all but the single chosen action at each step, both choosing a good inference order and choosing a correct action reduce to a single choice of what action to take next. To learn this decoding policy, we propose a novel Discriminative Reinforcement Learning (DRL) framework. DRL is used to train models that construct incrementally structured output using a local discriminative function, with the goal of optimizing a global loss function. We apply DRL to learn the UMT scoring function's parameters, using the BLEU score as the global loss function. DRL learns a weight vector for a linear classifier that discriminates between decisions based on which one leads to a complete translation-derivation with a better BLEU score. Promotions/demotions of translations are performed by applying a Perceptron-style update on the sequence of decisions that produced the translation, thereby training local decisions to optimize the global BLEU score of the final translation, while keeping the efficiency and simplicity of the Perceptron Algorithm (Rosenblatt, 1958; Collins, 2002).

Our experiments show that UMT with DRL reduces decoding time by over half, and the time to rescore translations with the Language Model by 6 times, while reaching accuracy non-significantly different from the state of the art.

2 Undirected Machine Translation

In this section, we present the UMT framework. For ease of presentation, and following synchronous-grammar based MT practice, we will henceforth restrict our focus to binary grammars (Zhang et al., 2006; Wang et al., 2007).

A UMT decoder can be formulated as a function, f, that maps a source sentence, $x \in \mathcal{X}$, into a structure defined by $y = (t, h) \in \mathcal{Y}$, where t is the translation in the target language, and h is the synchronous tree structure generating the input sentence on the source side and its translation on the target side. Synchronous-trees are composed of instantiations of synchronous-rules, r, from a grammar, G. A UMT decoder builds synchronous-trees, h, by recursively expanding partial synchronous-trees, τ. τ includes a partial translation. Each τ is required to be a connected sub-graph of some synchronous-tree h. Thus, τ is composed of a subset of the rules from any h that generates x on the source side, such that there is a connected path between any two rules in τ. Differently from the partial structures built by a bottom-up decoder, τ does not have to cover a contiguous span on x. Formally, τ is defined by:
1) The set of synchronous-rule instantiations in τ: $I \equiv \{r_1, r_2, \cdots, r_k | r_i \in G, 1 \leq i \leq k\}$;
2) The set of connections among the synchronous-rule instantiations, C.
Let $c_i = (r_i, r_{j_i})$ be the notation to represent the connection between the i-th rule and the rule r_{j_i}. The set of connections can be expressed as:
$C \equiv \{(r_1, r_{j_1}), (r_2, r_{j_2}), \cdots, (r_{k-1}, r_{j_{k-1}})\}$
3) The postcondition set, P, which specifies the non-terminals in τ that are available for creating new connections. Each postcondition, $p_i = (r_x, X_{\boxed{y}})_i$, indicates that the rule r_x has the non-terminal $X_{\boxed{y}}$ available for connections. The index $\boxed{y}$ identifies the non-terminal in the rule. In a binary grammar $\boxed{y}$ can take only 3 values: $\boxed{1}$ for the first non-terminal (the left child of the source side), $\boxed{2}$ for the second non-terminal, and $\boxed{h}$ for the head. The postcondition set can be expressed as:
$P \equiv \{(r_{x_1}, X_{y_1})_1, \cdots, (r_{x_m}, X_{y_m})_m\}$
4) The set of carries, K. We define a different carry, κ_i, for each non-terminal available for connections. Each carry stores the extra information required to correctly score the non-local interactions between τ and the rule that will be connected at that non-terminal. Thus $|K| = |P|$. Let κ_i be the carry associated with the postcondition p_i. The set of carries can be expressed as:
$K \equiv \{\kappa_1, \kappa_2, \cdots, \kappa_m\}$

Partial synchronous-trees, τ, are expanded by performing connection-actions. Given a τ we can connect to it a new rule, $\hat{r}$, using one available non-terminal represented by postcondition, $p_i \in P$, and obtain a new partial synchronous-tree $\hat{\tau}$. Formally: $\hat{\tau} \equiv \langle \tau \prec \hat{a} \rangle$, where, $\hat{a} = [\hat{r}, p_i]$, represents the connection-action.

Algorithm 1 UMT Decoding

1: **function** Decoder (x; $\mathbf{w}$, G) : (t,h)
2: $\tau.\{I, C, P, K\} \leftarrow \{\emptyset, \emptyset, \emptyset, \emptyset\}$;
3: $\mathbf{Q} \leftarrow$ LeafRules(G);
4: **while** $|\mathbf{Q}| > 0$ **do**
5: $[\hat{r}, p_i] \leftarrow$ PopBestAction ($\mathbf{Q}$,$\mathbf{w}$);
6: $\tau \leftarrow$ CreateConnection($\tau, \hat{r}, p_i$);
7: UpdateQueue($\mathbf{Q}$, $\hat{r}$, p_i);
8: **end while**
9: Return(τ);

10: **procedure** CreateConnection(τ, $\hat{r}, p_i$) : $\hat{\tau}$
11: $\hat{\tau}.I \leftarrow \tau.I + \hat{r}$;
12: $\hat{\tau}.C \leftarrow \tau.C + (\hat{r}, r_{p_i})$;
13: $\hat{\tau}.P \leftarrow \tau.P - p_i$;
14: $\hat{\tau}.K \leftarrow \tau.K - \kappa_i$;
15: $\hat{\tau}.K$.UpdateCarries($\hat{r}, p_i$);
16: $\hat{\tau}.P$.AddAvailableConnectionsFrom($\hat{r}, p_i$);
17: $\hat{\tau}.K$.AddCarriesForNewConnections($\hat{r}, p_i$);
18: Return($\hat{\tau}$);

19: **procedure** UpdateQueue($\mathbf{Q}$, $\hat{r}$, p_i) :
20: $\mathbf{Q}$.RemoveActionsWith(p_i);
21: $\mathbf{Q}$.AddNewActions($\hat{r}$, p_i);

2.1 Decoding Algorithm

Algorithm 1 gives details of the UMT decoding algorithm. The decoder takes as input the source sentence, x, the parameters of the scoring function, $\mathbf{w}$, and the synchronous-grammar, G. At $line$ 2 the partial synchronous-tree τ is initialized by setting I, C, P and K to empty sets $\emptyset$. At $line$ 3 the queue of candidate connection-actions is initialized as $\mathbf{Q} \equiv \{ [r_{leaf}, \mathbf{null}] \mid r_{leaf}$ is a leaf rule$\}$, where **null** means that there is no postcondition specified, since the first rule does not need to connect to anything. A leaf rule r_{leaf} is any synchronous rule with only terminals on the right-hand sides. At $line$ 4 the main loop starts. Each iteration of the main loop will expand τ using one connection-action. The loop ends when $\mathbf{Q}$ is empty, implying that τ covers the full sentence and has no more missing branches or parents. The best scoring action according to the parameter vector $\mathbf{w}$ is popped from the queue at $line$ 5. The scoring of connection-actions is discussed in details in Section 3.2. At $line$ 6 the selected connection-action is used to expand τ. At $line$ 7 the queue of candidates is updated accordingly (see $lines$ 19-21). At $line$ 8 the decoder iterates the main loop, until τ is complete and is returned at $line$ 9.

$Lines$ 10-18 describe the CreateConnection($\cdot$) procedure, that connects the partial synchronous-tree τ to the selected rule $\hat{r}$ via the postcondition p_i specified by the candidate-action selected in $line$ 5. This procedure returns the resulting partial synchronous-tree: $\hat{\tau} \equiv \langle \tau \prec [\hat{r}, p_i] \rangle$. At $line$ 11, $\hat{r}$ is added to the rule set I. At $line$ 12 the connection between $\hat{r}$ and r_{p_i} (the rule specified in the postcondition) is added to the set of connections C. At $line$ 13, p_i is removed from P. At $line$ 14 the carry k_i matching with p_i is removed from K. At $line$ 15 the set of carries K is updated, in order to update those carries that need to provide information about the new action. At $line$ 16 new postconditions representing the non-terminals in $\hat{r}$ that are available for subsequent connections are added in P. At $line$ 17 the carries associated with these new postconditions are computed and added to K. Finally at $line$ 18 the updated partial synchronous-tree is returned.

In the very first iteration, the CreateConnection($\cdot$) procedure has nothing to compute for some lines. $Line$ 11 is not executed since the first leaf rule needs no connection and has nothing to connect to. $lines$ 12-13 are not executed since P and K are $\emptyset$ and p_i is not specified for the first action. $Line$ 15 is not executed since there are no carries to be updated. $Lines$ 16-17 only add the postcondition and carry relative to the leaf rule head link.

The procedure used to update $\mathbf{Q}$ is reported in $lines$ 19-21. At $line$ 20 all the connection-actions involving the expansion of p_i are removed from $\mathbf{Q}$. These actions are the incompatible alternatives to the selected action. In the very first iteration, all actions in $\mathbf{Q}$ are removed because they are all incompatible with the connected-graph constraint. At $line$ 21 new connection-actions are added to $\mathbf{Q}$. These are the candidate actions proposing a connection to the available non-terminals of the selected action's new rule $\hat{r}$. The rules used for these new candidate-actions must not be in conflict with the current structure of τ (e.g. the rule cannot generate a source side terminal that is already covered by τ).

3 Discriminative Reinforcement Learning

Training a UMT model simply means training the parameter vector $\mathbf{w}$ that is used to choose the best scoring action during decoding. We propose a novel method to apply a kind of minimum error rate training (MERT) to $\mathbf{w}$. Because each action choice must be evaluated in the context of the complete translation-derivation, we formalize this method in terms of Reinforcement Learning. We propose Discriminative Reinforcement Learning as an appropriate way to train a UMT model to maximize the BLEU score of the complete derivation. First we define DRL as a novel generic training framework.

3.1 Generic Framework of DRL

RL can be applied to any task, $\mathcal{T}$, that can be formalized in terms of:
1) The set of states S[1];
2) A set of actions A_s for each state $s \in S$;
3) The transition function $T : S \times A_s \rightarrow S$, that specifies the next state given a source state and performed action[2];
4) The reward function, $R : S \times A_s \rightarrow \mathbb{R}$;
5) The discount factor, $\gamma \in [0, 1]$.

A policy is defined as any map $\pi : S \rightarrow A$. Its value function is given by:

$$V^{\pi}(s_0) = \sum_{i=0}^{\sigma} \gamma^i R(s_i, \pi(s_i)) \qquad (1)$$

where $\mathsf{path}(s_0|\pi) \equiv \langle s_0, s_1, \cdots, s_\sigma|\pi\rangle$ is the sequence of states determined by following policy π starting at state s_0. The Q-function is the total future reward of performing action a_0 in state s_0 and then following policy π:

$$Q^{\pi}(s_0, a_0) = R(s_0, a_0) + \gamma V^{\pi}(s_1) \qquad (2)$$

Standard RL algorithms search for a policy that maximizes the given reward.

Because we are taking a discriminative approach to learn $\mathbf{w}$, we formalize our optimization task similarly to an inverse reinforcement learning problem (Ng and Russell, 2000): we are given information about the optimal action sequence and we want to learn a discriminative reward function. As in other discriminative approaches, this approach simplifies the task of learning the reward function in two respects: the learned reward function only needs to be monotonically related to the true reward function, and this property only needs to hold for the best competing alternatives. This is all we need in order to use the discriminative reward function in an optimal classifier, and this simplification makes learning easier in cases where the true reward function is too complicated to model directly.

Algorithm 2 Discriminative RL

```
1: function Trainer (φ, T, D ) : w
2: repeat
3:    s ←SampleState(S);
4:    â ← π_w(s);
5:    a' ←SampleAction(A_s);
6:    if Q^{π_w}(s, â) < Q^{π_w}(s, a') in D then
7:       w ← w + Φ^w(s, a') − Φ^w(s, â);
8:    end if
9: until convergence
10: Return(w);
```

In RL, an optimal policy π^* is one which, at each state s, chooses the action which maximizes the future reward $Q^{\pi^*}(s, a)$. We assume that the future discriminative reward can be approximated with a linear function $\tilde{Q}^{\pi}(s, a)$ in some feature-vector representation $\phi : S \times A_s \rightarrow \mathbb{R}^d$ that maps a state-action pair to a d-dimensional features vector:

$$\tilde{Q}^{\pi}(s, a) = \mathbf{w}\ \phi(s, a) \qquad (3)$$

where $\mathbf{w} \in \mathbb{R}^d$. This gives us the following policy:

$$\pi_{\mathbf{w}}(s) = \arg\max_{a \in A_s}\ \mathbf{w}\ \phi(s, a) \qquad (4)$$

The set of parameters of this policy is the vector $\mathbf{w}$. With this formalization, all we need to learn is a vector $\mathbf{w}$ such that the resulting decisions are compatible with the given information about the optimal action sequence. We propose a Perceptron-like algorithm to learn these parameters.

Algorithm 2 describes the DRL meta-algorithm. The Trainer takes as input ϕ, the task $\mathcal{T}$, and a generic set of data D describing the behaviors we want to learn. The output is the weight vector $\mathbf{w}$ of the learned policy that fits the data D. The algorithm consists in a single training loop that is repeated until convergence ($lines$ 2-9). At $line$ 3 a state, s, is sampled from S. At $line$ 4, $\hat{a}$ is set to

[1] S can be either finite or infinite.

[2] For simplicity we describe a deterministic process. To generalize to the stochastic process, replace the transition function with the transition probability: $P_{sa}(s'),\ s' \in S$.

be the action that would be preferred by the current $\mathbf{w}$-policy. At $line\ 5$ an action, a', is sampled from A_s such that $a' \neq \hat{a}$. At $line\ 6$ the algorithm checks if preferring $\mathtt{path}(T(s,\hat{a}),\pi_{\mathbf{w}})$ over $\mathtt{path}(T(s,a'),\pi_{\mathbf{w}})$ is a correct choice according to the behaviors data D that the algorithm aims to learn. If the current $\mathbf{w}$-policy contradicts D, $line\ 7$ is executed to update the weight vector to promote $\Phi^{\mathbf{w}}(s,a')$ and penalize $\Phi^{\mathbf{w}}(s,\hat{a})$, where $\Phi^{\mathbf{w}}(s,a)$ is the summation of the features vectors of the entire derivation path starting at (s,a) and following policy $\pi_{\mathbf{w}}$. This way of updating $\mathbf{w}$ has the effect of increasing the $\tilde{Q}(\cdot)$ value associated with all the actions in the sequence that generated the promoted structure, and reducing the $\tilde{Q}(\cdot)$ value of the actions in the sequence that generated the penalized structure[3].

We have described the DRL meta-algorithm to be as general as possible. When applied to a specific problem, more details can be specified: **1)** it is possible to choose specific sampling techniques to implement $lines\ 3$ and 5; **2)** the test at $line\ 6$ needs to be detailed according to the nature of $\mathcal{T}$ and D; **3)** the update statement at $line\ 7$ can be replaced with a more sophisticated update approach. We address these issues and describe a range of alternatives as we apply DRL to UMT in Section 3.2.

3.2 Application of DRL to UMT

To apply DRL we formalize the task of translating x with UMT as $\mathcal{T} \equiv \{S, \{A_s\}, T, R, \gamma\}$:
1) The set of states S is the space of all possible UMT partial synchronous-trees, τ;
2) The set $A_{\tau,x}$ is the set of connection-actions that can expand τ connecting new synchronous-rule instantiations matching the input sentence x on the source side;
3) The transition function T is the connection function $\hat{\tau} \equiv \langle\, \tau \prec a \,\rangle$ formalized in Section 2 and detailed by the procedure CreateConnection($\cdot$) in Algorithm 1;
4) The true reward function R is the BLEU score. BLEU is a loss function that quantifies the difference between the reference translation and the output translation t. The BLEU score can be computed only when a terminal state is reached and a full translation is available. Thus, the rewards are all zero except at terminal states, called a Pure Delayed Reward function;
5) Considering the nature of the problem and reward function, we choose an undiscounted setting: $\gamma = 1$.

Next we specify the details of the DRL algorithm. The data D consists of a set of pairs of sentences, $D \equiv \{(x, t^*)\}$, where x is the source sentence and t^* is the reference translation. The feature-vector representation function ϕ maps a pair (τ, a) to a real valued vector having any number of dimensions. Each dimension corresponds to a distinct feature function that maps: $\{\tau\} \times A_{\tau,x} \rightarrow \mathbb{R}$. Details of the features functions implemented for our model are given in Section 4. Each loop of the DRL algorithm analyzes a single sample $(x, t^*) \in D$. The state s is sampled from a uniform distribution over $\langle s_0, s_1, \cdots, s_\sigma | \pi \rangle$. The action a' is sampled from a Zipfian distribution over $\{A_{\tau,x} - \hat{a}\}$ sorted with the $\tilde{Q}^{\pi_{\mathbf{w}}}(s,a)$ function. In this way actions with higher score have higher probability to be drawn, while actions at the bottom of the rank still have a small probability to be selected. The **if** at $line\ 6$ tests if the translation produced by $\mathtt{path}(T(s,a'),\pi_{\mathbf{w}})$ has higher BLEU score than the one produced by $\mathtt{path}(T(s,\hat{a}),\pi_{\mathbf{w}})$.

For the update statement at $line\ 7$ we use the Averaged Perceptron technique (Freund and Schapire, 1999). Algorithm 2 can be easily adapted to implement the efficient Averaged Perceptron updates (e.g. see Section 2.1.1 of (Daumé III, 2006)). In preliminary experiments, we found that other more aggressive update technique, such as Passive-Aggressive (Crammer et al., 2006), Aggressive (Shen et al., 2007), or MIRA (Crammer and Singer, 2003), lead to worst accuracy. To see why this might be, consider that a MT decoder needs to learn to construct structures (t, h), while the training data specifies the gold translation t^* but gives no information on the hidden-correspondence structure h. As discussed in (Liang et al., 2006), there are output structures that match the reference translation using a wrong internal structure (e.g. assuming wrong internal alignment). While in other cases the output translation can be a valid alternative translation but gets a low BLEU score because it differs from t^*. Aggressively promoting/penalizing structures whose correctness can be only partially verified can be expected to harm generalization ability.

[3] Preliminary experiments with updating only the features for $\hat{a}$ and a' produced substantially worse results.

4 Undirected Features

In this section we show how the features designed for bottom-up HMT can be adapted to the undirected approach, and we introduce a new feature from the class of undirected features that are made possible by the undirected approach.

Local features depend only on the action rule r. These features can be used in the undirected approach without adaptation, since they are independent of the surrounding structure. For our experiments we use a standard set of local features: the probability of the source phrase given the target phrase; the lexical translation probabilities of the source words given the target words; the lexical translation probabilities of the target words given the source words; and the Word Penalty feature.

Contextual features are dependent on the interaction between the action rule r and the available context. In UMT all the needed information about the available context is stored in the carry κ_i. Therefore, the computation of contextual features whose carry's size is bounded (like the LM) requires constant time.

The undirected adaptation of the LM feature computes the scores of the new n-grams formed by adding the terminals of the action rule r to the current partial translation τ. In the case that the action rule r is connected to τ via a child non-terminal, the carry is expressed as $\kappa_i \equiv ([W_L \star W_R])$. Where W_L and W_R are respectively the left and right boundary target words of the span covered by τ. This notation is analogous to the standard star notation used for the bottom-up decoder (e.g. (Chiang, 2007) Section 5.3.2). In the case that r is connected to τ via the head non-terminal, the carry is expressed as $\kappa_i \equiv (W_R]\text{-}[W_L)$. Where W_L and W_R are respectively the left and right boundary target words of the surrounding context provided by τ. The boundary words stored in the carry and the terminals of the action rule are all the information needed to compute and score the new n-grams generated by the connection-action.

In addition, we introduce the Context-Free Factor (CFF) features. An action rule r is connected to τ via one of r's non-terminals, $X_{r,\tau}$. Thus, the score of the interaction between r and the context structure attached to $X_{r,\tau}$ can be computed exactly, while the score of the structures attached to other r nonterminals (i.e. those in postconditions) cannot be computed since these branches are missing. Each of these postcondition nonterminals has an associated CFF feature, which is an upper bound on the score of its missing branch. More precisely, it is an upper bound on the context-free component of this score. This upper bound can be exactly and efficiently computed using the Forest Rescoring Framework (Huang and Chiang, 2007; Huang, 2008). This framework separates the MT decoding in two steps. In the first step only the context-free factors are considered. The output of the first step is a hypergraph called the context-free-forest, which compactly represents an exponential number of synchronous-trees. The second step introduces contextual features by applying a process of state-splitting to the context-free-forest, rescoring with non-context-free factors, and efficiently pruning the search space.

To efficiently compute CFF features we run the Inside-Outside algorithm with the $(max, +)$ semiring (Goodman, 1999) over the context-free-forest. The result is a map that gives the maximum Inside and Outside scores for each node in the context-free forest. This map is used to get the value of the CFF features in constant time while running the forest rescoring step.

5 Experiments

We implement our model on top of Cdec (Dyer et al., 2010). Cdec provides a standard implementation of the HMT decoder (Chiang, 2007) and MERT training (Och, 2003) that we use as baseline.

We experiment on the NIST Chinese-English parallel corpus. The training corpus contains $239k$ sentence pairs with $6.9M$ Chinese words and $8.9M$ English words. The test set contains 919 sentence pairs. The hierarchical translation grammar was extracted using the Joshua toolkit (Li et al., 2009) implementation of the suffix array rule extractor algorithm (Callison-Burch et al., 2005; Lopez, 2007).

Table 1 reports the decoding time measures. HMT with $beam1$ is the fastest possible configuration for HMT, but it is 71.59% slower than UMT. This is because HMT $b1$ constructs $O(n^2)$ subtrees, many of which end up not being used in the final result, whereas UMT only constructs the rule instantiations that are required. HMT with $beam30$ is the fastest configuration that reaches state of the art accuracy, but increases the average time per sentence by an additional 131.36% when compared with UMT. The rescoring time is

Model	sent. t.	sent. t. var.	resc. t.	resc. t. var.
UMT	135.2ms	-	38.9 ms	-
HMT $b1$	232.0ms	+71.59%	141.3 ms	+263.23%
HMT $b30$	312.8ms	+131.36%	226.9 ms	+483.29%

Table 1: Decoding speed comparison.

Model	sent. t.	sent. t. var.
UMT with DRL	267.4 ms	-
HMT $b1$	765.2 ms	+186.16%
HMT $b30$	1153.5 ms	+331.37%

Table 2: Training speed comparison.

Model	BLEU	relative loss	p-value
UMT with DRL	30.14	6.33%	0.18
HMT $b1$	30.87	4.07%	0.21
HMT $b30$	32.18	-	-

Table 3: Accuracy comparison.

the average time spent on the forest rescoring step, which is the only step where the decoders actually differ. This is the step that involves the integration of the Language Model and other contextual features. For HMT $b30$, rescoring takes two thirds of the total decoding time. Thus rescoring is the most time consuming step in the pipeline. The rescoring time comparison shows even bigger gains for UMT. HMT $b30$ is almost 6 times slower than UMT.

Table 2 reports the training time measures. These results show HMT $b30$ training is more than 4 times slower than UMT training with DRL. Comparing with Table 1, we notice that the relative gain on average training time is higher than the gain measured at decoding time. This is because MERT has an higher complexity than DRL. Both of the training algorithms requires 10 training epochs to reach convergence.

Table 3 reports the accuracy measures. As expected, accuracy degrades the more aggressively the search space is pruned. UMT trained with DRL loses 2.0 BLEU points compared to HMT $b30$. This corresponds to a relative-loss of 6.33%. Although not inconsequential, this variation is not considered big (e.g. at the WMT-11 Machine Translation shared task (Callison-Burch et al., 2011)). To measure the significance of the variation, we compute the sign test and measure the one-tail p-value for the presented models in comparison to HMT $b30$. From the values reported in the fourth column, we can observe that the BLEU score variations would not normally be considered significant. For example, at WMT-11 two systems were considered equivalent if $p > 0.1$, as in these cases. The accuracy cannot be compared in terms of search score since the models we are comparing are trained with distinct algorithms and thus the search scores are not comparable.

To test the impact of the CFF features, we trained and tested UMT with DRL with and without these features. This resulted in an accuracy decrease of 2.3 BLEU points. Thus these features are important for the success of the greedy approach. They provide an estimate of the score of the missing branches, thus helping to avoid some actions that have a good local score but lead to final translations with low global score.

To validate the results, additional experiments were executed on the French to Italian portion of the Europarl corpus v6. This portion contains $190k$ pairs of sentences. The first $186k$ sentences were used to extract the grammar and train the two models. The final tests were performed on the remaining $4k$ sentence pairs. With this corpus we measured a similar speed gain. HMT $b30$ is 2.3 times slower at decoding compared to UMT, and 6.1 times slower at rescoring, while UMT loses 1.1 BLEU points in accuracy. But again the accuracy differences are not considered significant. We measured a p-value of 0.25, which is not significant at the 0.1 level.

6 Related Work

Models sharing similar intuitions have been previously applied to other structure prediction tasks. For example, Nivre et al. (2006) presents a linear time syntactic dependency parser, which is constrained in a left-to-right decoding order. This model offers a different accuracy/complexity balance than the quadratic time graph-based parser of Mcdonald et al. (2005).

Other approaches learning a model specifically for greedy decoding have been applied with suc-

cess to other less complex tasks. Shen et al. (2007) present the Guided Learning (GL) framework for bidirectional sequence classification. GL successfully combines the tasks of learning the order of inference and training the local classifier in a single Perceptron-like algorithm, reaching state of the art accuracy with complexity lower than the exhaustive counterpart (Collins, 2002).

Goldberg and Elhadad (2010) present a similar training approach for a Dependency Parser that builds the tree-structure by recursively creating the easiest arc in a non-directional manner. This model also integrates the tasks of learning the order of inference and training the parser in a single Perceptron. By "non-directional" they mean the removal of the constraint of scanning the sentence from left to right, which is typical of shift-reduce models. However this algorithm still builds the tree structures in a bottom-up fashion. This model has a $O(n \log n)$ decoding complexity and accuracy performance close to the $O(n^2)$ graph-based parsers (Mcdonald et al., 2005).

Similarities can be found between DRL and previous work that applies discriminative training to structured prediction: Collins and Roark (2004) present an Incremental Parser trained with the Perceptron algorithm. Their approach is specific to dependency parsing and requires a function to test exact match of tree structures to trigger parameter updates. On the other hand, DRL can be applied to any structured prediction task and can handle any kind of reward function. LASO (Daumé III and Marcu, 2005; Daumé III et al., 2005) and SEARN (Daumé III et al., 2009; Daumé III et al., 2006) are generic frameworks for discriminative training for structured prediction: LASO requires a function that tests correctness of partial structures to trigger early updates, while SEARN requires an optimal policy to initialize the learning algorithm. Such a test function or optimal policy cannot be computed for tasks such as MT where the hidden correspondence structure h is not provided in the training data.

7 Discussion and Future Work

In general, we believe that greedy-discriminative solutions are promising for tasks like MT, where there is not a single correct solution: normally there are many correct ways to translate the same sentence, and for each correct translation there are many different derivation-trees generating that translation, and each correct derivation tree can be built greedily following different inference orders. Therefore, the set of correct decoding paths is a reasonable portion of UMT's search space, giving a well-designed greedy algorithm a chance to find a good translation even without beam search.

In order to directly evaluate the impact of our proposed decoding strategy, in this paper the only novel features that we consider are the CFF features. But to take full advantage of the power of discriminative training and the lower decoding complexity, it would be possible to vastly increase the number of features. The UMT's undirected nature allows the integration of non-bottom-up contextual features, which cannot be used by standard HMT and PbMT. And the use of a history-based model allows features from an arbitrarily wide context, since the model does not need to be factorized. Exploring the impact of this advantage is left for future work.

8 Conclusion

The main contribution of this work is the proposal of a new MT model that offers an accuracy/complexity balance that was previously unavailable among the choices of hierarchical models.

We have presented the first Undirected framework for MT. This model combines advantages given by the use of hierarchical synchronous-grammars with a more efficient decoding algorithm. UMT's nature allows us to design novel undirected features that better approximate contextual features (such as the LM), and to introduce a new class of undirected features that cannot be used by standard bottom-up decoders. Furthermore, we generalize the training algorithm into a generic Discriminative Reinforcement Learning meta-algorithm that can be applied to any structured prediction task.

References

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 19:263–311.

Chris Callison-Burch, Colin Bannard, and Josh Schroeder. 2005. Scaling phrase-based statistical machine translation to larger corpora and longer

phrases. In *ACL '05: Proceedings of the 43rd Conference of the Association for Computational Linguistics*, Ann Arbor, MI, USA.

Chris Callison-Burch, Philipp Koehn, Christof Monz, and Omar Zaidan. 2011. Findings of the 2011 workshop on statistical machine translation. In *WMT '11: Proceedings of the 6th Workshop on Statistical Machine Translation*, Edinburgh, Scotland.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *ACL '05: Proceedings of the 43rd Conference of the Association for Computational Linguistics*, Ann Arbor, MI, USA.

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

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *ACL '04: Proceedings of the 42rd Conference of the Association for Computational Linguistics*.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *EMNLP '02: Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, Philadelphia, PA, USA.

Koby Crammer and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991.

Koby Crammer, Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585.

Hal Daumé III and Daniel Marcu. 2005. Learning as search optimization: approximate large margin methods for structured prediction. In *ICML '05: Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany.

Hal Daumé III, John Langford, and Daniel Marcu. 2005. Search-based structured prediction as classification. In *ASLTSP '05: Proceedings of the NIPS Workshop on Advances in Structured Learning for Text and Speech Processing*, Whistler, British Columbia, Canada.

Hal Daumé III, John Langford, and Daniel Marcu. 2006. Searn in practice. Technical report.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Submitted to Machine Learning Journal*.

Hal Daumé III. 2006. Practical structured learning techniques for natural language processing. Ph.D. thesis, University of Southern California.

Chris Dyer, Adam Lopez, Juri Ganitkevitch, Jonathan Weese, Hendra Setiawan, Ferhan Ture, Vladimir Eidelman, Phil Blunsom, and Philip Resnik. 2010. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *ACL '10: Proceedings of the ACL 2010 System Demonstrations*, Uppsala, Sweden.

Yoav Freund and Robert E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

Andrea Gesmundo and James Henderson. 2011. Heuristic Search for Non-Bottom-Up Tree Structure Prediction. In *EMNLP '11: Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, Edinburgh, Scotland, UK.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *NAACL '10: Proceedings of the 11th Conference of the North American Chapter of the Association for Computational Linguistics*, Los Angeles, CA, USA.

Joshua Goodman. 1999. Semiring parsing. *Computational Linguistics*, 25:573–605.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *IWPT '05: Proceedings of the 9th International Workshop on Parsing Technology*, Vancouver, British Columbia, Canada.

Liang Huang and David Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *ACL '07: Proceedings of the 45th Conference of the Association for Computational Linguistics*, Prague, Czech Republic.

Liang Huang. 2008. Forest-based algorithms in natural language processing. Ph.D. thesis, University of Pennsylvania.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *NAACL '03: Proceedings of the 4th Conference of the North American Chapter of the Association for Computational Linguistics*, Edmonton, Canada.

Zhifei Li, Chris Callison-Burch, Chris Dyer, Sanjeev Khudanpur, Lane Schwartz, Wren Thornton, Jonathan Weese, and Omar Zaidan. 2009. Joshua: An open source toolkit for parsing-based machine translation. In *WMT '09: Proceedings of the 4th Workshop on Statistical Machine Translation*, Athens, Greece.

Percy Liang, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar. 2006. An end-to-end discriminative approach to machine translation. In *COLING-ACL '06: Proceedings of the 21st International Conference on Computational Linguistics and the 44th Conference of the Association for Computational Linguistics*, Sydney, Australia.

Adam Lopez. 2007. Hierarchical phrase-based translation with suffix arrays. In *EMNLP-CoNLL '07: Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, Prague, Czech Republic.

Ryan Mcdonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *ACL '05: Proceedings of the 43rd Conference of the Association for Computational Linguistics*, Ann Arbor, MI, USA.

Andrew Y. Ng and Stuart Russell. 2000. Algorithms for inverse reinforcement learning. In *ICML '00: Proceedings of the 17th International Conference on Machine Learning*, Stanford University, CA, USA.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006. Maltparser: A data-driven parser-generator for dependency parsing. In *LREC '06: Proceedings of the 5th International Conference on Language Resources and Evaluation*, Genoa, Italy.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *ACL '03: Proceedings of the 41st Conference of the Association for Computational Linguistics*, Sapporo, Japan.

Frank Rosenblatt. 1958. The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Libin Shen, Giorgio Satta, and Aravind Joshi. 2007. Guided learning for bidirectional sequence classification. In *ACL '07: Proceedings of the 45th Conference of the Association for Computational Linguistics*, Prague, Czech Republic.

Wei Wang, Kevin Knight, and Daniel Marcu. 2007. Binarizing syntax trees to improve syntax-based machine translation accuracy. In *EMNLP-CoNLL '07: Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, Prague, Czech Republic.

Kenji Yamada and Kevin Knight. 2001. A syntax-based statistical translation model. In *ACL '01: Proceedings of the 39th Conference of the Association for Computational Linguistics*, Toulouse, France.

Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous binarization for machine translation. In *NAACL '06: Proceedings of the 7th Conference of the North American Chapter of the Association for Computational Linguistics*, New York, New York.