Non-projective Dependency-based Pre-Reordering with Recurrent Neural Network for Machine Translation

Antonio Valerio Miceli-Barone
Università di Pisa
Largo B. Pontecorvo, 3
56127 Pisa, Italy
miceli@di.unipi.it

Giuseppe Attardi
Università di Pisa
Largo B. Pontecorvo, 3
56127 Pisa, Italy
attardi@di.unipi.it

Abstract

The quality of statistical machine translation performed with phrase based approaches can be increased by permuting the words in the source sentences in an order which resembles that of the target language. We propose a class of recurrent neural models which exploit source-side dependency syntax features to reorder the words into a target-like order. We evaluate these models on the German-to-English language pair, showing significant improvements over a phrase-based Moses baseline, obtaining a quality similar or superior to that of hand-coded syntactical reordering rules.

1 Introduction

Statistical machine translation is typically performed using phrase-based systems (Koehn et al., 2007). These systems can usually produce accurate local reordering but they have difficulties dealing with the long-distance reordering that tends to occur between certain language pairs (Birch et al., 2008).

The quality of phrase-based machine translation can be improved by reordering the words in each sentence of source-side of the parallel training corpus in a "target-like" order and then applying the same transformation as a pre-processing step to input strings during execution.

When the source-side sentences can be accurately parsed, pre-reordering can be performed using hand-coded rules. This approach has been successfully applied to German-to-English (Collins et al., 2005) and other languages. The main issue with it is that these rules must be designed for each specific language pair, which requires considerable linguistic expertise.

Fully statistical approaches, on the other hand, learn the reordering relation from word alignments. Some of them learn reordering rules on the constituency (Dyer and Resnik, 2010) (Khalilov and Fonollosa, 2011) or projective dependency (Genzel, 2010), (Lerner and Petrov, 2013) parse trees of source sentences. The permutations that these methods can learn can be generally non-local (i.e. high distance) on the sentences but local (parent-child or sibling-sibling swaps) on the parse trees. Moreover, constituency or projective dependency trees may not be the ideal way of representing the syntax of non-analytic languages, which could be better described using non-projective dependency trees (Bosco and Lombardo, 2004). Other methods, based on recasting reordering as a combinatorial optimization problem (Tromble and Eisner, 2009), (Visweswariah et al., 2011), can learn to generate in principle arbitrary permutations, but they can only make use of minimal syntactic information (part-of-speech tags) and therefore can't exploit the potentially valuable structural syntactic information provided by a parser.

In this work we propose a class of reordering models which attempt to close this gap by exploiting rich dependency syntax features and

Proceedings of SSST-9, Ninth Workshop on Syntax, Semantics and Structure in Statistical Translation, pages 10–20,
Denver, Colorado, June 4, 2015.

at the same time being able to process non-projective dependency parse trees and generate permutations which may be non-local both on the sentences and on the parse trees.
We represent these problems as sequence prediction machine learning tasks, which we address using recurrent neural networks.

We applied our model to reorder German sentences into an English-like word order as a pre-processing step for phrase-based machine translation, obtaining significant improvements over the unreordered baseline system and quality comparable to the hand-coded rules introduced by Collins et al. (2005).

2 Reordering as a walk on a dependency tree

In order to describe the non-local reordering phenomena that can occur between language pairs such as German-to-English, we introduce a reordering framework similar to (Miceli Barone and Attardi, 2013), based on a *graph walk* of the dependency parse tree of the source sentence. This framework doesn't restrict the parse tree to be projective, and allows the generation of arbitrary permutations.

Let $f \equiv (f_1, f_2, \ldots, f_{L_f})$ be a source sentence, annotated by a rooted dependency parse tree: $\forall j \in 1, \ldots, L_f, h_j \equiv PARENT(j)$

We define a *walker* process that walks from word to word across the edges of the parse tree, and at each steps optionally *emits* the current word, with the constraint that each word must be eventually emitted exactly once.
Therefore, the final string of emitted words f' is a permutation of the original sentence f, and any permutation can be generated by a suitable walk on the parse tree.

2.1 Reordering automaton

We formalize the walker process as a non-deterministic finite-state automaton.
The state v of the automaton is the tuple $v \equiv (j, E, a)$ where $j \in 1, \ldots, L_f$ is the current vertex (word index), E is the set of emitted vertices, a is the last action taken by the automaton.
The initial state is: $v(0) \equiv (root_f, \{\}, null)$ where $root_f$ is the root vertex of the parse tree.

At each step t, the automaton chooses one of the following actions:

- $EMIT$: emit the word f_j at the current vertex j. This action is enabled only if the current vertex has not been already emitted:

$$\frac{j \notin E}{(j, E, a) \overset{EMIT}{\rightarrow} (j, E \cup \{j\}, EMIT)} \quad (1)$$

- UP: move to the parent of the current vertex. Enabled if there is a parent and we did not just come down from it:

$$\frac{h_j \neq null, a \neq DOWN_j}{(j, E, a) \overset{UP}{\rightarrow} (h_j, E, UP_j)} \quad (2)$$

- $DOWN_{j'}$: move to the child j' of the current vertex. Enabled if the subtree $s(j')$ rooted at j' contains vertices that have not been already emitted and if we did not just come up from it:

$$\frac{h_{j'} = j, a \neq UP_{j'}, \exists k \in s(j') : k \notin E}{(j, E, a) \overset{DOWN_{j'}}{\rightarrow} (j', E, DOWN_{j'})} \quad (3)$$

The execution continues until all the vertices have been emitted.

We define the sequence of states of the walker automaton during one run as an *execution* $\bar{v} \in GEN(f)$. An execution also uniquely specifies the sequence of actions performed by the automation.

The preconditions make sure that all execution of the automaton always end generating a permutation of the source sentence. Furthermore, no cycles are possible: progress is made at every step, and it is not possible to enter in an execution that later turns out to be invalid.
Every permutation of the source sentence can be generated by some execution. In fact, each permutation f' can be generated by exactly one execution, which we denote as $\bar{v}(f')$.

We can split the execution $\bar{v}(f')$ into a sequence of L_f *emission fragments* $\bar{v}_j(f')$, each ending with an $EMIT$ action.
The first fragment has zero or more $DOWN_*$ actions followed by one $EMIT$ action, while each

other fragment has a non-empty sequence of UP and $DOWN_*$ actions (always zero or more UPs followed by zero or more DOWNs) followed by one $EMIT$ action.

Finally, we define an action in an execution as *forced* if it was the only action enabled at the step where it occurred.

2.2 Application

Suppose we perform reordering using a typical syntax-based system which processes source-side projective dependency parse trees and is limited to swaps between pair of vertices which are either in a parent-child relation or in a sibling relation. In such execution the UP actions are always forced, since the "walker" process never leaves a subtree before all its vertices have been emitted.

Suppose instead that we could perform reordering according to an "oracle". The executions of our automaton corresponding to these permutations will in general contain *unforced* UP actions. We define these actions, and the execution fragments that exhibit them, as *non-tree-local*.

In practice we don't have access to a reordering "oracle", but for sentences pairs in a parallel corpus we can compute heuristic "pseudo-oracle" reference permutations of the source sentences from word-alignments.

Following (Al-Onaizan and Papineni, 2006), (Tromble and Eisner, 2009), (Visweswariah et al., 2011), (Navratil et al., 2012), we generate word alignments in both the source-to-target and the target-to-source directions using IBM model 4 as implemented in GIZA++ (Och et al., 1999) and then we combine them into a symmetrical word alignment using the "grow-diag-final-and" heuristic implemented in Moses (Koehn et al., 2007).

Given the symmetric word-aligned corpus, we assign to each source-side word an integer index corresponding to the position of the left-most target-side word it is aligned to (attaching unaligned words to the following aligned word) and finally we perform a stable sort of source-side words according to this index.

2.3 Reordering example

Consider the segment of a German sentence shown in fig. 1. The English-reordered segment "**die Währungsreserven anfangs lediglich dienen sollten zur Verteidigung**" corresponds to the English: "**the reserve assets were originally intended to provide protection**".

In order to compose this segment from the original German, the reordering automaton described in our framework must perform a complex sequence of moves on the parse tree:

- Starting from "**sollten**", descend to "**dienen**", descent to "**Währungsreserven**" and finally to "**die**". Emit it, then go up to "**Währungsreserven**", emit it and go up to "**dienen**" and up again to "**sollten**". Note that the last UP is *unforced* since "**dienen**" has not been emitted at that point and has also unemitted children. This unforced action indicates non-tree-local reordering.

- Go down to "**anfangs**". Note that the in the parse tree edge crosses another edge, indicating non-projectivity. Emit "**anfangs**" and go up (forced) back to "**sollten**".

- Go down to "**dienen**", down to "**zur**", down to "**lediglich**" and emit it. Go up (forced) to "**zur**", up (unforced) to "**dienen**", emit it, go up (unforced) to "**sollten**", emit it. Go down to "**dienen**", down to "**zur**" emit it, go down to "**Verteidigung**" and emit it.

Correct reordering of this segment would be difficult both for a phrase-based system (since the words are further apart than both the typical maximum distortion distance and maximum phrase length) and for a syntax-based system (due to the presence of non-projectivity and non-tree-locality).

3 Recurrent Neural Network reordering models

Given the reordering framework described above, we could try to directly predict the ex-

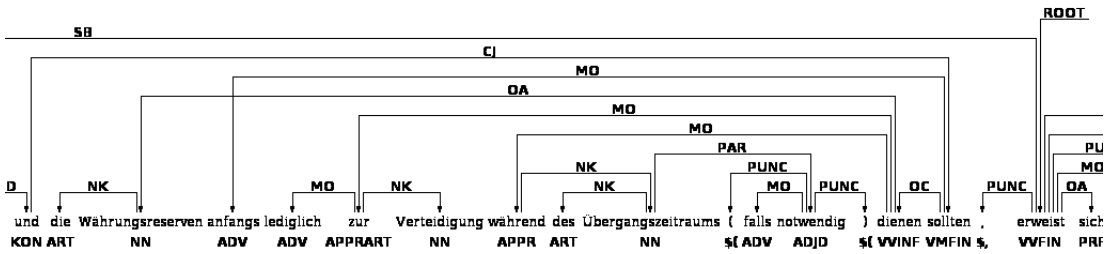

Figure 1: Section of the dependency parse tree of a German sentence.

ecutions as Miceli Barone and Attardi (2013) attempted with their version of the framework. However, the executions of a given sentence can have widely different lengths, which could make incremental inexact decoding such as beam search difficult due to the need to prune over partial hypotheses that have different numbers of emitted words.

Therefore, we decided to investigate a different class of models which have the property that state transition happen only in correspondence with word emission. This enables us to leverage the technology of incremental *language models*.

Using language models for reordering is not something new (Feng et al., 2010), (Durrani et al., 2011), (Bisazza and Federico, 2013), but instead of using a more or less standard n-gram language model, we are going to base our model on *recurrent neural network language models* (Mikolov et al., 2010).

Neural networks allow easy incorporation of multiple types of features and can be trained more specifically on the types of sequences that will occur during decoding, hence they can avoid wasting model space to represent the probabilities of non-permutations.

3.1 Base RNN-RM

Let $f \equiv (f_1, f_2, \ldots, f_{L_f})$ be a source sentence.
We model the reordering system as a deterministic single hidden layer recurrent neural network:

$$v(t) = \tau(\Theta^{(1)} \cdot x(t) + \Theta^{REC} \cdot v(t-1)) \quad (4)$$

where $x(t) \in \mathcal{R}^n$ is a feature vector associated to the t-th word in a permutation f', $v(0) \equiv v_{init}$, $\Theta^{(1)}$ and Θ^{REC} are parameters[1] and $\tau(\cdot)$ is the hyperbolic tangent function.

If we know the first $t-1$ words of the permutation f' in order to compute the probability distribution of the t-th word we do the following:

- Iteratively compute the state $v(t-1)$ from the feature vectors $x(1), \ldots, x(t-1)$.
- For the all the indices of the words that haven't occurred in the permutation so far $j \in J(t) \equiv ([1, L_f] - \bar{i}_{t-1:})$, compute a score $h_{j,t} \equiv h_o(v(t-1), x_o(j))$, where $x_o(\cdot)$ is the feature vector of the candidate target word.
- Normalize the scores using the logistic softmax function: $P(\bar{I}_t = j | f, \bar{i}_{t-1:}, t) = \frac{\exp(h_{j,t})}{\sum_{j' \in J(t)} \exp(h_{j',t})}$.

The scoring function $h_o(v(t-1), x_o(j))$ applies a feed-forward hidden layer to the feature inputs $x_o(j)$, and then takes a weighed inner product between the activation of this layer and the state $v(t-1)$. The result is then linearly combined to an additional feature equal to the logarithm of the remaining words in the permutation $(L_f - t)$,[2] and to a bias feature:

$$\begin{aligned} h_{j,t} \equiv & < \tau(\Theta^{(o)} \cdot x_o(j)), \theta^{(2)} \odot v(t-1) > \\ & + \theta^{(\alpha)} \cdot \log(L_f - t) + \theta^{(bias)} \end{aligned} \quad (5)$$

where $h_{j,t} \equiv h_o(v(t-1), x_o(j))$.

[1] we don't use a bias feature since it is redundant when the layer has input features encoded with the "one-hot" encoding

[2] since we are then passing this score to a softmax of variable size $(L_f - t)$, this feature helps the model to keep the score already approximately scaled.

We can compute the probability of an entire permutation f' just by multiplying the probabilities for each word: $P(f'|f) = P(\bar{I} = \bar{i}|f) = \prod_{t=1}^{L_f} P(\bar{I}_t = \bar{i}_t|f, t)$

3.1.1 Training

Given a training set of pairs of sentences and reference permutations, the training problem is defined as finding the set of parameters $\theta \equiv (v_{init}, \Theta^{(1)}, \theta^{(2)}, \Theta^{REC}, \Theta^{(o)}, \theta^{(\alpha)}, \theta^{(bias)})$ which minimize the per-word empirical cross-entropy of the model w.r.t. the reference permutations in the training set. Gradients can be efficiently computed using *backpropagation through time* (BPTT).

In practice we used the following training architecture:
Stochastic gradient descent, with each training pair (f, f') considered as a single minibatch for updating purposes. Gradients computed using the automatic differentiation facilities of Theano (Bergstra et al., 2010) (which implements a generalized BPTT). No truncation is used. L2-regularization [3]. Learning rates dynamically adjusted per scalar parameter using the *AdaDelta* heuristic (Zeiler, 2012). Gradient clipping heuristic to prevent the "exploding gradient" problem (Graves, 2013). Early stopping w.r.t. a validation set to prevent overfitting. Uniform random initialization for parameters other than the recurrent parameter matrix Θ^{REC}. Random initialization with *echo state property* for Θ^{REC}, with contraction coefficient $\sigma = 0.99$ (Jaeger, 2001), (Gallicchio and Micheli, 2011).

Training time complexity is $O(L_f^2)$ per sentence, which could be reduced to $O(L_f)$ using truncated BTTP at the expense of update accuracy and hence convergence speed. Space complexity is $O(L_f)$ per sentence.

3.1.2 Decoding

In order to use the RNN-RM model for prereordering we need to compute the most likely permutation $\overset{*}{f'}$ of the source sentence f:

$$\overset{*}{f'} \equiv \underset{f' \in GEN(f)}{\text{argmax}} \; P(f'|f) \tag{6}$$

Solving this problem to the global optimum is computationally hard[4], hence we solve it to a local optimum using a *beam search* strategy.

We generate the permutation incrementally from left to right. Starting from an initial state consisting of an empty string and the initial state vector v_{init}, at each step we generate all possible successor states and retain the B-most probable of them (histogram pruning), according to the probability of the entire prefix of permutation they represent.

Since RNN state vectors do not decompose in a meaningful way, we don't use any hypothesis recombination.
At step t there are $L_f - t$ possible successor states, and the process always takes exactly L_f steps[5], therefore time complexity is $O(B \cdot L_f^2)$ and space complexity is $O(B)$.

3.1.3 Features

We use two different feature configurations: *unlexicalized* and *lexicalized*.

In the *unlexicalized* configuration, the state transition input feature function $x(j)$ is composed by the following features, all encoded using the "one-hot" encoding scheme:

- Unigram: $POS(j)$, $DEPREL(j)$, $POS(j) * DEPREL(j)$. Left, right and parent unigram: $POS(k)$, $DEPREL(k)$, $POS(k) * DEPREL(k)$, where k is the index of respectively the word at the left (in the original sentence), at the right and the dependency parent of word j. Unique tags are used for padding.

- Pair features: $POS(j) * POS(k)$, $POS(j) * DEPREL(k)$, $DEPREL(j) * POS(k)$, $DEPREL(j) * DEPREL(k)$, for k defined as above.

[3] $\lambda = 10^{-4}$ on the recurrent matrix, $\lambda = 10^{-6}$ on the final layer, per minibatch.

[4] NP-hard for at least certain choices of features and parameters

[5] actually, $L_f - 1$, since the last choice is forced

- Triple features $POS(j) * POS(left_j) * POS(right_j)$, $POS(j) * POS(left_j) * POS(parent_j)$, $POS(j) * POS(right_j) * POS(parent_j)$.
- Bigram: $POS(j) * POS(k)$, $POS(j) * DEPREL(k)$, $DEPREL(j) * POS(k)$ where k is the previous emitted word in the permutation.
- Topological features: three binary features which indicate whether word j and the previously emitted word are in a parent-child, child-parent or sibling-sibling relation, respectively.

The target word feature function $x_o(j)$ is the same as $x(j)$ except that each feature is also conjoined with a quantized signed distance[6] between word j and the previous emitted word. Feature value combinations that appear less than 100 times in the training set are replaced by a distinguished "rare" tag.

The *lexicalized* configuration is equivalent to the unlexicalized one except that $x(j)$ and $x_o(j)$ also have the surface form of word j (not conjoined with the signed distance).

3.2 Fragment RNN-RM

The *Base RNN-RM* described in the previous section includes dependency information, but not the full information of reordering fragments as defined by our automaton model (sec. 2). In order to determine whether this rich information is relevant to machine translation pre-reordering, we propose an extension, denoted as *Fragment RNN-RM*, which includes reordering fragment features, at expense of a significant increase of time complexity.

We consider a hierarchical recurrent neural network. At top level, this is defined as the previous RNN. However, the $x(j)$ and $x_o(j)$ vectors, in addition to the feature vectors described as above now contain also the final states of another recurrent neural network.

This internal RNN has a separate clock and a separate state vector. For each step t of the top-level RNN which transitions between word $f'(t-1)$ and $f'(t)$, the internal RNN is reinitialized to its own initial state and performs multiple internal steps, one for each action in the fragment of the execution that the walker automaton must perform to walk between words $f'(t-1)$ and $f'(t)$ in the dependency parse (with a special shortcut of length one if they are adjacent in f with monotonic relative order).

The state transition of the inner RNN is defined as:

$$v_r(t) = \tau(\Theta^{(r_1)} \cdot x_r(t_r) + \Theta^{r_{REC}} \cdot v_r(t_r - 1)) \quad (7)$$

where $x_r(t_r)$ is the feature function for the word traversed at inner time t_r in the execution fragment. $v_r(0) = v_r^{init}$, $\Theta^{(r_1)}$ and $\Theta^{r_{REC}}$ are parameters.

Evaluation and decoding are performed essentially in the same was as in Base RNN-RM, except that the time complexity is now $O(L_f^3)$ since the length of execution fragments is $O(L_f)$.

Training is also essentially performed in the same way, though gradient computation is much more involved since gradients propagate from the top-level RNN to the inner RNN. In our implementation we just used the automatic differentiation facilities of Theano.

3.2.1 Features

The *unlexicalized* features for the inner RNN input vector $x_r(t_r)$ depend on the current word in the execution fragment (at index t_r), the previous one and the action label: UP, $DOWN$ or $RIGHT$ (shortcut). $EMIT$ actions are not included as they always implicitly occur at the end of each fragment.

Specifically the features, encoded with the "one-hot" encoding are: $A * POS(t_r) * POS(t_r - 1)$, $A * POS(t_r) * DEPREL(t_r - 1)$, $A * DEPREL(t_r) * POS(t_r - 1)$, $A * DEPREL(t_r) * DEPREL(t_r - 1)$.

These features are also conjoined with the quantized signed distance (in the original sentence) between each pair of words.

The *lexicalized* features just include the surface form of each visited word at t_r.

[6] values greater than 5 and smaller than 10 are quantized as 5, values greater or equal to 10 are quantized as 10. Negative values are treated similarly.

3.3 Base GRU-RM

We also propose a variant of the Base RNN-RM where the standard recurrent hidden layer is replaced by a *Gated Recurrent Unit* layer, recently proposed by Cho et al. (2014) for machine translation applications.
The Base GRU-RM is defined as the Base RNN-RM of sec. 3.1, except that the recurrence relation 4 is replaced by fig. 2

Features are the same of unlexicalized Base RNN-RM (we experienced difficulties training the Base GRU-RM with lexicalized features).
Training is also performed in the same way except that we found more beneficial to convergence speed to optimize using *Adam* (Kingma and Ba, 2014) [7] rather than AdaDelta.
In principle we could also extend the Fragment RNN-RM into a Fragment GRU-RM, but we did not investigate that model in this work.

4 Experiments

We performed German-to-English pre-reordering experiments with Base RNN-RM (both unlexicalized and lexicalized), Fragment RNN-RM and Base GRU-RM .

4.1 Setup

The baseline phrase-based system was trained on the German-to-English corpus included in Europarl v7 (Koehn, 2005). We randomly split it in a 1,881,531 sentence pairs training set, a 2,000 sentence pairs development set (used for tuning) and a 2,000 sentence pairs test set. The English language model was trained on the English side of the parallel corpus augmented with a corpus of sentences from AP News, for a total of 22,891,001 sentences.
The baseline system is phrase-based Moses in a default configuration with maximum distortion distance equal to 6 and lexicalized reordering enabled. Maximum phrase size is equal to 7.
The language model is a 5-gram IRSTLM/KenLM.
The pseudo-oracle system was trained on the training and tuning corpus obtained by permuting the German source side using the heuristic described in section 2.2 and is otherwise equal to the baseline system.
In addition to the test set extracted from Europarl, we also used a 2,525 sentence pairs test set (”news2009”) a 3,000 sentence pairs ”challenge” set used for the WMT 2013 translation task (”news2013”).

We also trained a Moses system with pre-reordering performed by Collins et al. (2005) rules, implemented by Howlett and Dras (2011).
Constituency parsing for Collins et al. (2005) rules was performed with the Berkeley parser (Petrov et al., 2006), while non-projective dependency parsing for our models was performed with the DeSR transition-based parser (Attardi, 2006).

For our experiments, we extract approximately 300,000 sentence pairs from the Moses training set based on a heuristic confidence measure of word-alignment quality (Huang, 2009), (Navratil et al., 2012). We randomly removed 2,000 sentences from this filtered dataset to form a validation set for early stopping, the rest were used for training the pre-reordering models.

4.2 Results

The hidden state size s of the RNNs was set to 100 while it was set to 30 for the GRU model, validation was performed every 2,000 training examples. After 50 consecutive validation rounds without improvement, training was stopped and the set of training parameters that resulted in the lowest validation cross-entropy were saved.
Training took approximately 1.5 days for the unlexicalized Base RNN-RM, 2.5 days for the lexicalized Base RNN-RM and for the unlexicalized Base GRU-RM and 5 days for the unlexicalized Fragment RNN-RM on a 24-core machine without GPU (CPU load never rose to more than 400%).

Decoding was performed with a beam size of 4. Decoding the whole corpus took about 1.0-1.2 days for all the models except Fragment

[7] with learning rate $2 \cdot 10^{-5}$ and all the other hyperparameters equal to the default values in the article.

$$
\begin{aligned}
v_{rst}(t) &= \pi(\Theta^{(1)}_{rst} \cdot x(t) + \Theta^{REC}_{rst} \cdot v(t-1)) \\
v_{upd}(t) &= \pi(\Theta^{(1)}_{upd} \cdot x(t) + \Theta^{REC}_{upd} \cdot v(t-1)) \\
v_{raw}(t) &= \tau(\Theta^{(1)} \cdot x(t) + \Theta^{REC} \cdot v(t-1) \odot v_{upd}(t)) \\
v(t) &= v_{rst}(t) \odot v(t-1) + (1 - v_{rst}(t)) \odot v_{raw}(t)
\end{aligned}
\tag{8}
$$

Figure 2: GRU recurrence equations. $v_{rst}(t)$ and $v_{upd}(t)$ are the activation vectors of the "reset" and "update" gates, respectively, and $\pi(\cdot)$ is the logistic sigmoid function.

.

Reordering	BLEU	improvement
none	62.10	
unlex. Base RNN-RM	64.03	+1.93
lex. Base RNN-RM	63.99	+1.89
unlex. Fragment RNN-RM	64.43	+2.33
unlex. Base GRU-RM	64.78	+2.68

Figure 3: "Monolingual" reordering scores (upstream system output vs. "oracle"-permuted German) on the Europarl test set. All improvements are significant at 1% level.

RNN-RM for which it took about 3 days.

Effects on monolingual reordering score are shown in fig. 3, effects on translation quality are shown in fig. 4.

4.3 Discussion and analysis

All our models significantly improve over the phrase-based baseline, performing as well as or almost as well as (Collins et al., 2005), which is an interesting result since our models doesn't require any specific linguistic expertise.

Surprisingly, the lexicalized version of Base RNN-RM performed worse than the unlexicalized one. This goes contrary to expectation as neural language models are usually lexicalized and in fact often use nothing but lexical features.

The unlexicalized Fragment RNN-RM was quite accurate but very expensive both during training and decoding, thus it may not be practical.

The unlexicalized Base GRU-RM performed very well, especially on the Europarl dataset (where all the scores are much higher than the other datasets) and it never performed significantly worse than the unlexicalized Fragment RNN-RM which is much slower.

We also performed exploratory experiments with different feature sets (such as lexical-only features) but we couldn't obtain a good training error. Larger network sizes should increase model capacity and may possibly enable training on simpler feature sets.

5 Conclusions

We presented a class of statistical syntax-based pre-reordering systems for machine translation.

Our systems processes source sentences parsed with non-projective dependency parsers and permutes them into a target-like word order, suitable for translation by an appropriately trained downstream phrase-based system.

The models we proposed are completely trained with machine learning approaches and is, in principle, capable of generating arbitrary permutations, without the hard constraints that are commonly present in other statistical syntax-based pre-reordering methods.

Practical constraints depend on the choice of features and are therefore quite flexible, allowing a trade-off between accuracy and speed.

In our experiments with the RNN-RM and

Test set	system	BLEU	improvement
Europarl	baseline	33.00	
Europarl	"oracle"	41.80	+8.80
Europarl	Collins	33.52	+0.52
Europarl	unlex. Base RNN-RM	33.41	+0.41
Europarl	lex. Base RNN-RM	33.38	+0.38
Europarl	unlex. Fragment RNN-RM	33.54	+0.54
Europarl	**unlex. Base GRU-RM**	34.15	**+1.15**
news2013	baseline	18.80	
news2013	Collins	NA	NA
news2013	unlex. Base RNN-RM	19.19	+0.39
news2013	lex. Base RNN-RM	19.01	+0.21
news2013	unlex. Fragment RNN-RM	19.27	+0.47
news2013	**unlex. Base GRU-RM**	19.28	**+0.48**
news2009	baseline	18.09	
news2009	**Collins**	18.74	**+0.65**
news2009	unlex. Base RNN-RM	18.50	+0.41
news2009	lex. Base RNN-RM	18.44	+0.35
news2009	unlex. Fragment RNN-RM	18.60	+0.51
news2009	unlex. Base GRU-RM	18.58	+0.49

Figure 4: RNN-RM translation scores. All improvements are significant at 1% level.

GRU-RM models we managed to achieve translation quality improvements comparable to those of the best hand-coded pre-reordering rules.

References

Yaser Al-Onaizan and Kishore Papineni. 2006. Distortion models for statistical machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, ACL-44, pages 529–536, Stroudsburg, PA, USA. Association for Computational Linguistics.

Giuseppe Attardi. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pages 166–170, Stroudsburg, PA, USA. Association for Computational Linguistics.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. 2010. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June. Oral Presentation.

Alexandra Birch, Miles Osborne, and Philipp Koehn. 2008. Predicting success in machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 745–754, Stroudsburg, PA, USA. Association for Computational Linguistics.

Arianna Bisazza and Marcello Federico. 2013. Efficient solutions for word reordering in German-English phrase-based statistical machine translation. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 440–451, Sofia, Bulgaria, August. Association for Computational Linguistics.

Cristina Bosco and Vincenzo Lombardo. 2004. Dependency and relational structure in treebank annotation. In *COLING 2004 Recent Advances in Dependency Grammar*, pages 1–8.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Michael Collins, Philipp Koehn, and Ivona Kučerová. 2005. Clause restructuring for statistical machine translation. In *Proceedings of*

the 43rd annual meeting on association for computational linguistics, pages 531–540. Association for Computational Linguistics.

Nadir Durrani, Helmut Schmid, and Alexander Fraser. 2011. A joint sequence translation model with integrated reordering. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1045–1054. Association for Computational Linguistics.

Chris Dyer and Philip Resnik. 2010. Context-free reordering, finite-state translation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 858–866, Stroudsburg, PA, USA. Association for Computational Linguistics.

Minwei Feng, Arne Mauser, and Hermann Ney. 2010. A source-side decoding sequence model for statistical machine translation. In *Conference of the Association for Machine Translation in the Americas (AMTA)*.

C. Gallicchio and A. Micheli. 2011. Architectural and markovian factors of echo state networks. *Neural Networks*, 24(5):440 – 456.

Dmitriy Genzel. 2010. Automatically learning source-side reordering rules for large scale machine translation. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 376–384, Stroudsburg, PA, USA. Association for Computational Linguistics.

Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Susan Howlett and Mark Dras. 2011. Clause restructuring for SMT not absolutely helpful. In *Proceedings of the 49th Annual Meeting of the Assocation for Computational Linguistics: Human Language Technologies*, pages 384–388.

Fei Huang. 2009. Confidence measure for word alignment. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 932–940. Association for Computational Linguistics.

Herbert Jaeger. 2001. The echo state approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148:34.

Maxim Khalilov and José AR Fonollosa. 2011. Syntax-based reordering for statistical machine translation. *Computer speech & language*, 25(4):761–788.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 177–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *Conference Proceedings: the tenth Machine Translation Summit*, pages 79–86, Phuket, Thailand. AAMT, AAMT.

Uri Lerner and Slav Petrov. 2013. Source-side classifier preordering for machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP '13)*.

Antonio Valerio Miceli Barone and Giuseppe Attardi. 2013. Pre-reordering for machine translation using transition-based walks on dependency parse trees. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 164–169, Sofia, Bulgaria, August. Association for Computational Linguistics.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048.

Jiri Navratil, Karthik Visweswariah, and Ananthakrishnan Ramanathan. 2012. A comparison of syntactic reordering methods for english-german machine translation. In *COLING*, pages 2043–2058.

Franz Josef Och, Christoph Tillmann, Hermann Ney, et al. 1999. Improved alignment models for statistical machine translation. In *Proc. of the Joint SIGDAT Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 20–28.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 433–440. Association for Computational Linguistics.

Roy Tromble and Jason Eisner. 2009. Learning linear ordering problems for better translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2 - Volume 2*, EMNLP '09, pages 1007–1016, Stroudsburg, PA, USA. Association for Computational Linguistics.

Karthik Visweswariah, Rajakrishnan Rajkumar, Ankur Gandhe, Ananthakrishnan Ramanathan, and Jiri Navratil. 2011. A word reordering model for improved machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 486–496, Stroudsburg, PA, USA. Association for Computational Linguistics.

Matthew D Zeiler. 2012. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.