Investigations on Phrase-based Decoding with Recurrent Neural Network Language and Translation Models

Tamer Alkhouli, Felix Rietig, and Hermann Ney
Human Language Technology and Pattern Recognition Group
RWTH Aachen University, Aachen, Germany
{surname}@cs.rwth-aachen.de

Abstract

This work explores the application of recurrent neural network (RNN) language and translation models during phrase-based decoding. Due to their use of unbounded context, the decoder integration of RNNs is more challenging compared to the integration of feedforward neural models. In this paper, we apply approximations and use caching to enable RNN decoder integration, while requiring reasonable memory and time resources. We analyze the effect of caching on translation quality and speed, and use it to integrate RNN language and translation models into a phrase-based decoder. To the best of our knowledge, no previous work has discussed the integration of RNN translation models into phrase-based decoding. We also show that a special RNN can be integrated efficiently without the need for approximations. We compare decoding using RNNs to rescoring n-best lists on two tasks: IWSLT 2013 German→English, and BOLT Arabic→English. We demonstrate that the performance of decoding with RNNs is at least as good as using them in rescoring.

1 Introduction

Applying neural networks to statistical machine translation has been gaining increasing attention recently. Neural network language and translation models have been successfully applied to rescore the first-pass decoding output (Le et al., 2012; Sundermeyer et al., 2014; Hu et al., 2014; Guta et al., 2015). These models include feedforward and recurrent neural networks.

A more ambitious move is to apply neural networks directly during decoding, which in principle should give the models a better chance to influence translation in comparison to rescoring, as rescoring is limited to scoring and reranking fixed n-best lists. Recently, neural networks were used for standalone decoding using a simple beam-search word-based decoder (Sutskever et al., 2014; Bahdanau et al., 2015). Another approach is to apply neural models directly in a phrase-based decoder. We focus on this approach, which is challenging since phrase-based decoding typically involves generating tens or even hundreds of millions of partial hypotheses. Scoring such a number of hypotheses using neural models is expensive, mainly due to the usually large output layer. Nevertheless, decoder integration has been done in (Vaswani et al., 2013) for feedforward neural language models. Devlin et al. (2014) integrate feedforward translation models into phrase-based decoding reporting major improvements, which highlight the strength of the underlying models.

In related fields like e. g. language modeling, RNNs has been shown to perform considerably better than standard feedforward architectures (Mikolov et al., 2011; Arisoy et al., 2012; Sundermeyer et al., 2013; Liu et al., 2014). Sundermeyer et al. (2014) also show that RNN translation models outperform feedforward networks in rescoring. Given the success of feedforward translation models in phrase-based decoding, it is natural to ask how RNN translation models perform if they are integrated in decoding.

This paper investigates the performance of RNN language and translation models in phrase-based decoding. For RNNs that depend on an unbounded target context, their integration into a phrase-based decoder employing beam search requires relaxing the pruning parameters, which makes translation inefficient. Therefore, we apply approximations to integrate RNN translation models during phrase-based decoding. Auli and Gao (2014) use approximate scoring to integrate

Proceedings of the Tenth Workshop on Statistical Machine Translation, pages 294–303,
Lisboa, Portugal, 17-18 September 2015. ©2015 Association for Computational Linguistics.

an RNN language model (LM), but to the best of our knowledge, no work yet has explored the integration of RNN translation models. In addition to approximate models, we integrate a special RNN model that only depends on the source context, allowing for exact, yet efficient integration into the decoder. We provide a detailed comparison between using the RNN models in decoding vs. rescoring on two tasks: IWSLT 2013 German→English and BOLT Arabic→English. In addition, we analyze the approximation effect on translation speed and quality.

Our integration follows (Huang et al., 2014), which uses caching strategies to apply an RNN LM in speech recognition. This can be viewed as a modification of the approximation introduced by Auli and Gao (2014), allowing for a flexible choice between translation quality and speed. We choose to integrate the word-based RNN translation models that were introduced in (Sundermeyer et al., 2014), due to their success in rescoring n-best lists.

The rest of this paper is structured as follows. In Section 2 we review the related work. The RNN LM integration and caching strategies are discussed in Section 3, while Section 4 discusses the integration of exact and approximate RNN translation models. We analyze the effect of approximation and caching on translation quality and speed in Section 5. The section also includes the translation experiments comparing decoding vs. rescoring. Finally we conclude with Section 6.

2 Related Work

Schwenk (2012) proposed a feedforward network that predicts phrases of a fixed maximum length, such that all phrase words are predicted at once. The prediction is conditioned on the source phrase. The model was used to compute additional phrase table scores, and the phrase table was used for decoding. No major difference was reported compared to rescoring using the model. Our work focuses on neural network scoring performed online during decoding, capturing dependencies that extend beyond phrase boundaries.

Online usage of neural networks during decoding requires tackling the costly output normalization step. Vaswani et al. (2013) avoid this step by training feedforward neural language models using *noise contrastive estimation*. Auli and Gao (2014) propose an expected BLEU criterion instead of the usual cross-entropy. They train recurrent neural LMs without the need to normalize the output layer, but training becomes computationally more expensive as each training example is an n-best list instead of a sentence. At decoding time, however, scoring with the neural network is faster since normalization is not needed. Furthermore, they integrate cross-entropy RNNs without affecting state recombination. They report results over a baseline having a LM trained on the target side of the parallel data. The results for the RNN LM trained with cross-entropy indicated that decoding improves over rescoring, with the difference ranging from 0.4% to 0.8% BLEU. In this work, we stick to RNNs trained using cross-entropy, with a class-factored output layer to reduce the normalization cost.

Devlin et al. (2014) augment the cross-entropy training objective function to produce approximately normalized scores directly. They also precompute the first hidden layer beforehand, resulting in large speedups. Major improvements over strong baselines were reported. While their work focuses on feedforward translation models, we investigate the decoder integration of RNN models instead, which poses additional challenges due to the unbounded history used by RNNs.

Huang et al. (2014) truncate the history and use it to cache the hidden RNN states, the normalization factors and the probability values. This is applied to an RNN LM in a speech recognition task. In this work, we apply these caching strategies to a recurrent LM for translation tasks. Furthermore, we analyze the degree of approximation and its influence on the search problem. We also extend caching and apply it to RNN translation models that are conditioned on source and target words.

Sundermeyer et al. (2014) proposed word- and phrase-based RNN translation models and applied them to rescore n-best lists, reporting major improvements. The RNN word-based models were shown to outperform a feedforward neural network. This work aims to enable the use of the word-based RNN models in phrase-based decoding, and to explore their effect on the search space during decoding.

3 RNN Language Model Integration

In this section we discuss the integration of the RNN LM using caching in details. These caching techniques will also be applied to the joint

RNN translation model in Section 4.2 with minor changes.

First, we will briefly introduce the RNN LM. The LM probability $p(e_i|e_1^{i-1})$ of the target word e_i at position i depends on the unbounded target history e_1^{i-1}. The probability can be computed using an RNN LM of a single hidden layer as follows:

$$y_{i-1} = A_1 \hat{e}_{i-1} \quad (1)$$

$$h(e_1^{i-1}) = \xi(y_{i-1}; A_2, h(e_1^{i-2})) \quad (2)$$

$$o(e_1^{i-1}) = A_3 h(e_1^{i-1}) \quad (3)$$

$$Z(e_1^{i-1}) = \sum_{w=1}^{|V|} e^{o_w(e_1^{i-1})} \quad (4)$$

$$p(e_i|e_1^{i-1}) = \frac{e^{o_{e_i}(e_1^{i-1})}}{Z(e_1^{i-1})} \quad (5)$$

where A_1, A_2 and A_3 denote the neural network weight matrices, $\hat{e}_{i-1}$ is the one-hot vector encoding the word e_{i-1}, and y_{i-1} is its word embedding vector. h is a vector of the hidden layer activations depending on the unbounded context, and it is computed recurrently using the function ξ, which we use to represent a generic recurrent layer. $o \in \mathbb{R}^{|V|}$ is a $|V|$-dimensional vector containing the raw unnormalized output layer values, where $|V|$ is the vocabulary size. The probability in Eq. 5 is computed using the softmax function, which requires the normalization factor Z. In this work, we use a class-factored output layer consisting of a class layer and a word layer (Goodman, 2001; Morin and Bengio, 2005). In this case, the LM probability is the product of the two:

$$p(e_i|e_1^{i-1}) = p(e_i|c(e_i), e_1^{i-1}) \cdot p(c(e_i)|e_1^{i-1})$$

where c denotes a word mapping from any target word to its unique class. Such factorization is used to reduce the normalization cost.

Phrase-based decoding involves the generation of a search graph consisting of nodes. Each node represents a search state uniquely identified by a triple $(C, \tilde{e}, j)$, where C denotes the coverage set, $\tilde{e}$ is the language model history, and j is the position of the last translated source word. During decoding, equivalent nodes are recombined. The degree of recombination is affected by the order of the LM history, where higher orders result in fewer recombinations. Our phrase-based decoder is based on beam search, where the search space is pruned and a limit is imposed on the number of hypotheses to explore. Since an RNN LM depends on the full target history e_1^{i-1}, a naïve integration of the RNN LM would define $\tilde{e} = e_1^{i-1}$, but this leads to an explosion in the number of nodes in the search graph, which in turn leads to reducing the variance between the hypotheses lying within the beam, and focusing the decoding effort on hypotheses that are similar to each other.

Since the RNN LM computes a hidden state $h(e_1^{i-1})$ encoding the sequence e_1^{i-1}, another way is to extend the search state of the node to $(C, \tilde{e}, j, h(e_1^{i-1}))$. However, such extension would pose the same problem for recombination as the one encountered if the full history sequence is stored. Therefore, we resort to approximate RNN LM evaluation in decoding. An approximation proposed in (Auli et al., 2013) is to extend the search node with the RNN hidden state, but to ignore the hidden state when deciding which nodes to recombine. That is, two search nodes are deemed equivalent if they share the same triple $(C, \tilde{e}, j)$, even if they have different RNN hidden states. Upon recombination, one of the two hidden states is kept and stored in the resulting recombined node.

3.1 Caching Strategies

In this work, we follow a modification of the approach by (Auli et al., 2013). Instead of storing the RNN hidden state in the search nodes, we truncate the RNN history to the most recent n words e_{i-n}^{i-1}, and store this word sequence in the node instead. As in (Auli et al., 2013), the added information is ignored when deciding on recombination. When the RNN hidden state is needed, it is retrieved from a cache using the truncated history as a key. The cache is shared between all nodes. While this might seem as an unnecessary complication, it introduces the flexibility of choosing the degree of approximation. The parameter n can be used to control the trade-off between accuracy and speed; more accurate RNN scores are obtained if n is set to a large value, or faster decoding is achieved if n is set to a small value. In principle, we can still simulate the case of storing the hidden RNN state directly in the search nodes by using large n values as we will see later. We will refer to n as the *caching order*.

During decoding, we use the cache C_{state} to store the hidden state $h(e_1^{i-1})$ using the key e_{i-n}^{i-1}.

The state $h(e_1^{i-1})$ is computed once, using the hidden state $h(e_1^{i-2})$ as given in Eq. 2, and the cached state is reused whenever it is needed. Note that the n-gram key is only used to look up the hidden state, which will have been computed recurrently encoding an unbounded history. This is different from a feedforward network which uses the n-gram as direct input to compute its output. We also introduce a cache C_{norm} to store the output layer's normalization factor $Z(e_1^{i-1})$ using the same key e_{i-n}^{i-1}, hence avoiding the sum of Eq. 4, which requires the expensive computation of the full raw output layer values $o(e_1^{i-1})$ using Eq. 3. If the normalization factor is found in the cache, computing Eq. 5 only requires the output value o_{e_i} corresponding to word e_i, which involves a dot product rather than a matrix-vector product. Since we use a class-factored output layer, we cache the normalization factor of the class layer. Finally, the cache C_{prob} is introduced to store the word probability using the caching key (e_i, e_{i-n}^{i-1}).

Fig. 1 shows the percentage of cache hits for different caching orders. We count a cache hit if a look up is performed on that cache and the entry is found, otherwise the look up counts as a cache miss. We observe high hit ratios even for high caching orders. This is due to the fact that most of the hits occur upon node expansion, where a node is extended by a new phrase, and where all candidates share the same history. We also observe that word probabilities are retrieved from the cache 70% of the time for high enough caching orders, which can be explained due to the similarities between the phrase candidates in their first word. Note also that the reported C_{norm} hit ratio is for the cases where the cache C_{prob} produces a cache miss. We report this hit ratio since the original C_{norm} hit ratio is equal to C_{state}'s hit ratio as they both use the same caching key.

We report the effect of caching on translation speed in Tab. 1, where we use a large caching order of 30 to simulate the search space covered when no caching is used. Using none of the caches and storing the hidden state in the search node instead has a speed of 0.03 words per second. This increases to 0.05 words per second when caching the hidden state. This is because caching computes each hidden state once, while storing the hidden state in the search node may lead to computing the same hidden state multiple times, as no global view of what has been computed is available. Caching the normalization constant yields a large speedup, due to the reduction of the number of times the full output layer is computed. Finally, caching word probabilities does not speed up translation further. This is due to the class-factored output layer we use, where computing the softmax for the word layer part (given a word class) uses a small matrix corresponding to the words that belong to the same class of the word in question. Overall, a speedup factor of 6 is achieved over the case where no caching is used. Achieving this speedup does not lead to a loss in translation quality, in fact, for all cases, the translation quality is the same due to the large caching order used.

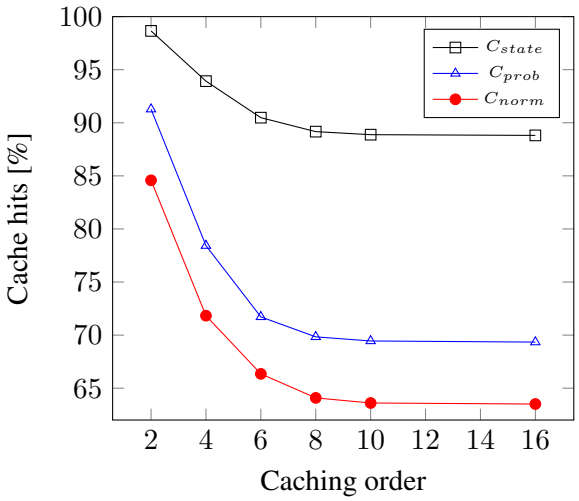

Figure 1: Cache hits for different caching orders when using an RNN LM in the decoder.

Cache	Speed [words/second]
none	0.03
C_{state}	0.05
$C_{state} + C_{norm}$	0.19
$C_{state} + C_{norm} + C_{prob}$	0.19

Table 1: The effect of using caching on translation speed. A large caching order of 30 is used to reduce the approximation effect of caching, leading to the same translation quality for all table entries.

4 RNN Translation Model Integration

One of the main contributions of this work is to integrate RNN translation models into phrase-based decoding. To the best of our knowledge, no such integration has been done before. We integrate two models that work on the word level. The mod-

els were proposed in (Sundermeyer et al., 2014). They assume a one-to-one alignment between the source sentence $f_1^I = f_1 \dots f_I$ and the target sentence $e_1^I = e_1 \dots e_I$. Such alignment is obtained using heuristics that make use of IBM 1 lexica. In the following, we discuss the integration of the bidirectional translation model (BTM), which can be done exactly and efficiently without resorting to approximations. In addition, we propose an approximate integration of the joint model (JM) which makes use of the same caching strategies discussed in Section 3.

4.1 Bidirectional Translation Model

The bidirectional translation model (BTM) is conditioned on the full source sentence, without dependence on previously predicted target words:

$$p(e_1^I|f_1^I) \approx \prod_{i=1}^{I} p(e_i|f_1^I). \quad (6)$$

This equation is realized by a network that uses forward and backward recurrent layers to capture the complete source sentence. The forward layer is a recurrent hidden layer that processes the source sequence from left to right, while a backward layer does the processing backwards, from right to left. The source sentence is basically split at a given position i, then past and future representations of the sentence are recursively computed by the forward and backward layers, respectively. Due to recurrency, the forward layer encodes f_1^i, and the backward layer encodes f_i^I, and together they encode the full source sentence f_1^I, which is used to score the output target word e_i.

Including the BTM in the decoder is efficient and scores can be computed exactly. This is because the model has no dependence on previous target words hypothesized during decoding. For a sentence of length I, and a target vocabulary size $|V|$, the number of distinct evaluations is at most $I \cdot |V|$. The term I corresponds to the number of possibilities where the source sentence may be split into past and future parts, and the term $|V|$ is the different possible target words that may be hypothesized. In phrase-based decoding, the number of distinct evaluations is in the order of thousands, as the number of target word candidates per sentence is limited by the phrase table. Since the input to the network is completely known at the beginning of decoding, it is enough that the full network is computed I times per source sentence, once per split position i for $1 \leq i \leq I$. Computing $p(e_i = e|f_1^I)$ amounts to looking up the normalized output layer value corresponding the word e from the network computed using the split position i.

4.2 Joint Model

The joint model (JM) conditions target word predictions on the hypothesized target history in addition to the source history and the current source word:[1]

$$p(e_1^I|f_1^I) = \prod_{i=1}^{I} p(e_i|e_1^{i-1}, f_1^i). \quad (7)$$

This equation can be modeled using a network similar to the RNN LM. While the RNN LM has the previous target word e_{i-1} as direct input to score the current target word e_i, the JM aggregates the word embeddings of the previous target word e_{i-1} and the current source word f_i. Due to recurrency, the hidden state will encode the sequence pair (e_1^{i-1}, f_1^i).

Since the JM is similar to the RNN LM in its dependence on the unbounded history, we apply caching strategies similar to those used with the RNN LM. JM computations are shared between instances that have a truncated source and target history in common. The cache key in this case is $(e_{i-n}^{i-1}, f_{i-n+1}^i)$ for the C_{state} and C_{norm} caches, and $(e_i, e_{i-n}^{i-1}, f_{i-n+1}^i)$ for the C_{prob} cache.

5 Experiments

5.1 Setup

We carry out experiments on the IWSLT 2013 German→English shared translation task.[2] The baseline system is trained on all available bilingual data, 4.3M sentence pairs in total, and uses a 4-gram LM with modified Kneser-Ney smoothing (Kneser and Ney, 1995; Chen and Goodman, 1998), trained with the SRILM toolkit (Stolcke, 2002). As additional data sources for the LM, we selected parts of the Shuffled News and LDC English Gigaword corpora based on the cross-entropy difference (Moore and Lewis, 2010), resulting in a total of 1.7 billion running words for

[1] We use a unidirectional rather than a bidirectional JM, dropping the future source information f_{i+1}^I. This is because the models we integrate reorder the source sentence following the target order, which can only be done for the past part of the source sentence at decoding time.

[2] http://www.iwslt2013.org

LM training. The state-of-the-art baseline is a standard phrase-based SMT system (Koehn et al., 2003) tuned with MERT (Och, 2003). It contains a hierarchical reordering model (Galley and Manning, 2008) and a 7-gram word cluster language model (Wuebker et al., 2013). All neural networks are trained on the TED portion of the data (138K segments). The experiments are run using an observation histogram size of 100, with a maximum of 16 lexical hypotheses per source coverage and a maximum of 32 reordering alternatives per source cardinality.

Additional experiments are performed on the Arabic→English task of the DARPA BOLT project. The system is a standard phrase-based decoder trained on 921K segments, amounting to 15.5M running words, and using 17 dense features. The neural network training is performed using the same data. We evaluate results on two data sets from the 'discussion forum' domain, `test1` and `test2`. The sizes of the data sets are: 1219 (`dev`), 1510 (`test1`), and 1137 (`test2`) segments. An additional development set containing 2715 segments is used during RNN training. The experiments are run using an observation histogram size of 100, with a maximum of 32 lexical hypotheses per source coverage and a maximum of 8 reordering alternatives per source cardinality.

The BTM consists of a linear projection layer, forward and backward long-short term memory (LSTM) layers and an additional LSTM to combine them. Each of the LM and JM has a projection layer and a single LSTM layer. All layers have 200 nodes, with 2000 classes used for the class-factored output layer.

All results are measured in case-insensitive BLEU [%] (Papineni et al., 2002) and TER [%] (Snover et al., 2006) on a single reference. The reported decoding results are averages of 3 MERT optimization runs. Rescoring experiments are performed using 1000-best lists (without duplicates), where an additional MERT iteration is performed. 20 such trials are carried out and the average results are reported. We used the multeval toolkit (Clark et al., 2011) for evaluation.

5.2 Approximation Analysis

First, we will analyze the caching impact on decoding. We compare RNN LM rescoring and decoding by marking a win for the method finding the better search score. Decoding with the RNN LM uses approximate scores while rescoring with same model uses the exact scores. Fig. 2 shows the percentage of improved hypotheses for RNN decoding (compared to RNN rescoring) and for RNN rescoring (compared to RNN decoding). The figure does not include tie cases, which occur when RNN LM decoding and rescoring yield the same hypothesis as their best finding. For the caching order $n = 8$, decoding finds a better search score for 33% of the sentences compared to rescoring, while rescoring has a better score in 17% of the cases compared to decoding. The remaining 50% cases (not shown in the figure) correspond to ties where both search methods select the same hypotheses. Increasing the caching order improves the decoding quality. For the caching order $n = 16$, rescoring outperforms decoding in 12% of the cases, i.e. for the remaining 88% cases, decoding is at least as good as rescoring.

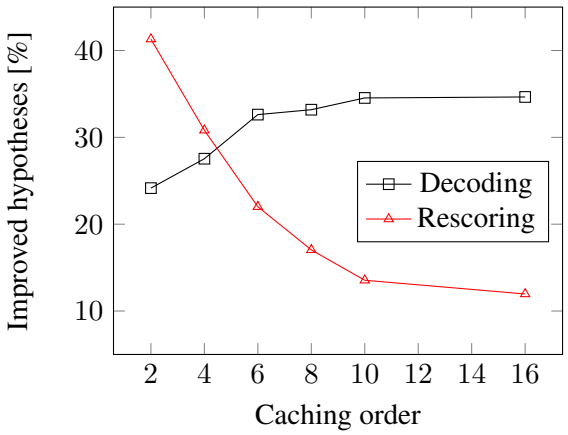

Figure 2: A comparison between applying the RNN LM in decoding using caching, and applying it exactly in rescoring.

Even for high caching orders, we observe that decoding does not completely beat rescoring. This can be attributed to the recombination approximation, as recombination disregards the RNN history. We performed another experiment to determine the effect of recombination on the RNN scores. In this experiment the RNN hidden state is stored in the search nodes, and no caching is used. This leaves recombination as the only source of approximation. We generated 1000-best lists using the approximate RNN LM scores during decoding. Afterwards, we computed the exact RNN scores of the 1000-best lists and compared them to the approximate scores. Fig. 3 shows the cumulative distribution of the absolute relative dif-

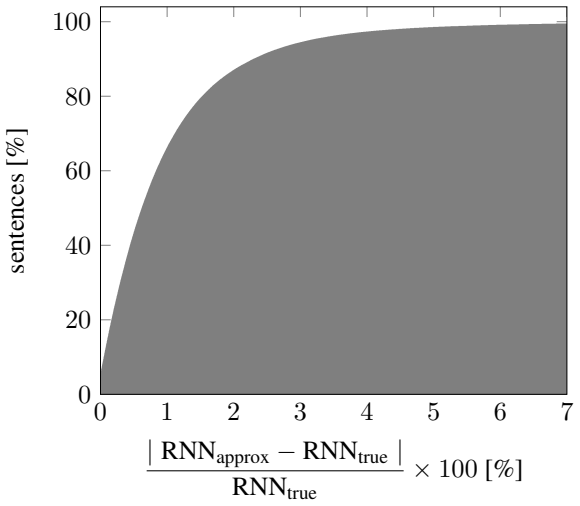

Figure 3: The cumulative distribution of the absolute relative difference between the approximate and true RNN score with respect to the true score. The distribution was generated using around 233k sentences, obtained from n-best lists generated by decoding the `dev` set of the IWSLT task.

ference between the approximate and true RNN scores with respect to the true scores. The figure suggests that recombining search nodes while ignoring the RNN hidden state leads to inexact RNN scores in most cases. For 66% of the cases the absolute relative error is at most 1%, and the error is at most 6% for 99% of the cases. As expected, ignoring the RNN during recombination leads to inexact RNN scores.

In Tab. 2, we compare between caching the RNN hidden state and the approach proposed in (Auli et al., 2013), which stores the RNN hidden state in the search node. The experiment aims to compare the two approaches in terms of translation quality. If the caching order is at least 6, no considerable difference is observed. This result is in favor of caching due to the speedup it achieves (cf. Tab. 1).

5.3 Translation Results

The IWSLT results are given in Tab. 3. We observe that decoding with RNNs improves the baseline by $1.0 - 1.7\%$ BLEU and $0.9 - 1.9\%$ TER on the `test` set. These improvements are at least as good as those of rescoring. This applies both for the exact BTM as well as the approximate LM and JM cases. In the case of BTM decoding, we observe an improvement of 0.1 BLEU and 0.5 TER compared to the corresponding rescoring experiment. The decoding improvements in the LM and JM cases are minor compared to rescoring. We also experimented with rescoring the RNN decoding output, where rescoring was performed using the same RNN used in decoding to obtain exact scores. We took the best on `dev` among the 3 MERT runs and rescored it. This is indicated by the "+" sign. The results show that RNN LM rescoring can be improved if decoding is performed including the RNN LM. On `test` the gain is 0.4 BLEU and 0.2 TER, while the improvement is 0.2 BLEU and 0.4 TER in the JM case. This indicates that using the RNN model in decoding improves the n-best lists, allowing rescoring afterwards to choose better hypotheses. Overall, BTM decoding improves over the baseline by 1.7 BLEU and 1.9 TER.

Caching Order	`dev`	`test`
2	33.1	30.8
4	33.4	31.2
6	33.9	31.6
8	33.9	31.5
16	34.0	31.5
30	33.9	31.5
-	33.9	31.5

Table 2: A comparison between storing the RNN state in the search nodes (last entry) or caching it using different caching orders (remaining entries). We report the BLEU [%] scores for the IWSLT 2013 German→English task.

	`dev`		`test`	
	BLEU	TER	BLEU	TER
baseline	33.4	46.1	30.6	49.2
LM Resc.	34.1	45.7	31.5	48.6
LM Dec.	33.9	45.7	31.6	48.3
+LM Resc.	34.1	45.8	31.9	48.4
BTM Resc.	34.4	45.3	32.2	47.8
BTM Dec.	34.4	44.9	**32.3**	**47.3**
JM Resc.	34.3	45.4	31.6	48.3
JM Dec.	34.4	45.6	31.6	48.2
+ JM Resc.	34.6	45.3	31.8	47.9

Table 3: IWSLT 2013 German→English results. Caching orders: $n = 8$ (LM), $n = 5$ (JM).

	test1		test2	
	BLEU	TER	BLEU	TER
baseline	23.9	59.7	26.4	59.8
LM Resc.	24.3	59.3	26.9	59.3
LM Dec.	24.6	59.0	27.0	59.2
+LM Resc.	**25.0**	**58.8**	27.2	59.1
BTM Resc.	24.7	58.9	27.0	**58.9**
BTM Dec.	24.8	58.9	27.0	**58.9**
JM Resc.	24.4	59.0	27.2	59.0
JM Dec.	24.5	59.0	**27.3**	59.0
+ JM Rec.	24.5	59.0	**27.3**	59.0

Table 4: BOLT Arabic→English results. Caching orders: $n = 8$ (LM), $n = 10$ (JM).

Tab. 4 shows the results of the Arabic→English BOLT task. Again, the LM, JM and BTM models in decoding are at least as good as in rescoring. For the LM, we observe an improvement of 0.7 BLEU when LM rescoring is applied on the LM decoding output. The best result improves the baseline by 1.1 BLEU on test1 and 0.9 BLEU on test2.

In a final experiment to examine the power of the recurrent neural translation models, we performed phrase-based decoding without the conventional phrasal and lexical translation scores. Instead, we performed decoding with the BTM as described in Section 4.1, and augmented the phrase table with four additional features derived from the bidirectional translation model, the joint model, and the phrase-based translation and joint models described in (Sundermeyer et al., 2014). This was done by scoring each phrase pair in the phrase table as if it were a sentence pair. For this specific experiment, we trained the phrase-based models on phrase pairs obtained from forced-decoding the training data. That is, each training instance was a phrase pair instead of a sentence pair. For the sake of comparison, we trained the baseline translation model on the TED portion of the data; the same data used for neural training. The results are shown in Tab. 5. We observe a gap of only 1.2 BLEU on dev and 1.0 BLEU on test, with almost no difference in TER. We consider this an encouraging result, as it is possible that the word-based recurrent neural models used here are not capable of expressing their full potential due to their use in the phrase-based framework,

	dev		test	
	BLEU	TER	BLEU	TER
baseline	32.2	46.6	30.5	48.7
RNN	31.0	46.8	29.5	48.6

Table 5: The in-domain baseline has a translation model trained on the TED portion of the data only, while RNN denotes decoding with the BTM, in addition to 4 offline word- and phrase-based neural scores in the phrase table . The phrase-based models were trained on forced-aligned phrase-pairs rather than full sentences.

which only allows phrases given by the phrase table. Therefore, it would be interesting to examine the performance of the models outside the phrase-based framework.

5.4 Discussion

We observe that integrating RNN models into phrase-based decoding slightly outperforms applying them in a rescoring step. This is unlike the case of feedforward networks, which where integrated into phrase-based decoding in (Devlin et al., 2014), and resulted in large improvements compared to rescoring. Even when we use large caching orders, we observe no major improvements over rescoring. This can be attributed to the fact that deciding on recombining search nodes completely ignores the RNN hidden state, which could be a harsh approximation, given that the RNN hidden state encodes the complete history. We experimented with changing the LM order used to make recombination decisions, which we refer to as the recombination order. However, simply increasing the recombination order does not enhance the translation quality, and it starts to even have a negative impact. This can be explained due to the fact that our phrase-based decoder is based on beam search, which has fixed pruning parameters that allow a fixed number of hypotheses to be explored. Simply increasing the recombination order limits the variety in the beam. When the beam size is doubled,[3] both RNN decoding and rescoring improve, but the difference between them is still insignificant. To be able to benefit from the increase in recombination order, the beam size

[3]We doubled each of the observation histogram size, the number of lexical hypotheses per source coverage and the number of reordering alternatives per source cardinality.

should be appropriately increased. But using large beam sizes makes translation costly and infeasible. This calls for other more selective ways to make recombination decisions dependent on the RNN hidden state.

6 Conclusion

We investigated the integration of RNN language and translation models into a phrase-based decoder. We integrated exact RNN translation models that are conditioned on the source context only, and used caching to integrate approximate RNN translation models that are dependent on the target context. This is the first time RNN translation models are integrated into phrase-based decoding. We analyzed the effect of caching on translation quality and speed, and demonstrated that it achieves equivalent translation results compared to having the RNN hidden states stored in the decoder's search nodes, while being 6 times faster. Translation results indicated that applying the models in decoding is at least as good as applying them in rescoring n-best lists, but we observed no major advantage for RNN decoding. Future work will investigate approaches to make recombination dependent on the RNN hidden state in a feasible way, furthermore, we will explore how the RNN models perform in word-based decoding.

Acknowledgments

This material is partially based upon work supported by the DARPA BOLT project under Contract No. HR0011- 12-C-0015. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA. The research leading to these results has also received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement n^{o} 645452 (QT21). We would like to thank Joern Wuebker for many insightful discussions.

References

Ebru Arisoy, Tara N. Sainath, Brian Kingsbury, and Bhuvana Ramabhadran. 2012. Deep neural network language models. In *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, pages 20–28, Montréal, Canada, June.

Michael Auli and Jianfeng Gao. 2014. Decoder Integration and Expected BLEU Training for Recurrent Neural Network Language Models. In *Annual Meeting of the Association for Computational Linguistics*, pages 136–142, Baltimore, MD, USA, June.

Michael Auli, Michel Galley, Chris Quirk, and Geoffrey Zweig. 2013. Joint Language and Translation Modeling with Recurrent Neural Networks. In *Conference on Empirical Methods in Natural Language Processing*, pages 1044–1054, Seattle, USA, October.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, San Diego, Calefornia, USA, May.

Stanley F. Chen and Joshua Goodman. 1998. An Empirical Study of Smoothing Techniques for Language Modeling. Technical Report TR-10-98, Computer Science Group, Harvard University, Cambridge, MA, August.

Jonathan H. Clark, Chris Dyer, Alon Lavie, and Noah A. Smith. 2011. Better hypothesis testing for statistical machine translation: Controlling for optimizer instability. In *49th Annual Meeting of the Association for Computational Linguistics*, pages 176–181, Portland, Oregon, June.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. 2014. Fast and Robust Neural Network Joint Models for Statistical Machine Translation. In *52nd Annual Meeting of the Association for Computational Linguistics*, pages 1370–1380, Baltimore, MD, USA, June.

Michel Galley and Christopher D. Manning. 2008. A simple and effective hierarchical phrase reordering model. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 848–856, Honolulu, Hawaii, USA, October.

Joshua Goodman. 2001. Classes for fast maximum entropy training. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, volume 1, pages 561–564. IEEE.

Andreas Guta, Tamer Alkhouli, Jan-Thorsten Peter, Joern Wuebker, and Hermann Ney. 2015. A Comparison between Count and Neural Network Models Based on Joint Translation and Reordering Sequences. In *Conference on Empirical Methods on Natural Language Processing*, page to appear, Lisbon, Portugal, September.

Yuening Hu, Michael Auli, Qin Gao, and Jianfeng Gao. 2014. Minimum translation modeling with recurrent neural networks. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 20–29, Gothenburg, Sweden, April.

Zhiheng Huang, Geoffrey Zweig, and Benoit Dumoulin. 2014. Cache based recurrent neural network language model inference for first pass speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 6404–6408, Florence, Italy, May.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for M-gram language modeling. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 181–184, May.

P. Koehn, F. J. Och, and D. Marcu. 2003. Statistical Phrase-Based Translation. In *Proceedings of the 2003 Meeting of the North American chapter of the Association for Computational Linguistics*, pages 127–133, Edmonton, Alberta, May/June.

Hai Son Le, Alexandre Allauzen, and François Yvon. 2012. Continuous Space Translation Models with Neural Networks. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 39–48, Montreal, Canada, June.

X. Liu, Y. Wang, X. Chen, M. J. F. Gales, and P. C. Woodland. 2014. Efficient lattice rescoring using recurrent neural network language models. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 4941–4945, Florence, Italy, May.

Tomas Mikolov, Stefan Kombrink, Lukas Burget, JH Cernocky, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531, Prague, Czech Republic, May.

R.C. Moore and W. Lewis. 2010. Intelligent Selection of Language Model Training Data. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 220–224, Uppsala, Sweden, July.

Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *Proceedings of the international workshop on artificial intelligence and statistics*, pages 246–252, Barbados, January.

Franz Josef Och. 2003. Minimum Error Rate Training in Statistical Machine Translation. In *Proc. of the 41th Annual Meeting of the Association for Computational Linguistics (ACL)*, 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 *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July.

Holger Schwenk. 2012. Continuous Space Translation Models for Phrase-Based Statistical Machine Translation. In *25th International Conference on Computational Linguistics (COLING)*, pages 1071–1080, Mumbai, India, December.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A Study of Translation Edit Rate with Targeted Human Annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas*, pages 223–231, Cambridge, Massachusetts, USA, August.

Andreas Stolcke. 2002. SRILM – An Extensible Language Modeling Toolkit. In *Proc. of the Int. Conf. on Speech and Language Processing (ICSLP)*, volume 2, pages 901–904, Denver, CO, September.

Martin Sundermeyer, Ilya Oparin, Jean-Luc Gauvain, Ben Freiberg, Ralf Schlüter, and Hermann Ney. 2013. Comparison of feedforward and recurrent neural network language models. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 8430–8434, Vancouver, Canada, May.

Martin Sundermeyer, Tamer Alkhouli, Joern Wuebker, and Hermann Ney. 2014. Translation Modeling with Bidirectional Recurrent Neural Networks. In *Conference on Empirical Methods on Natural Language Processing*, pages 14–25, Doha, Qatar, October.

Ilya Sutskever, Oriol Vinyals, and Quoc V. V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112, Montréal, Canada, December.

Ashish Vaswani, Yinggong Zhao, Victoria Fossum, and David Chiang. 2013. Decoding with large-scale neural language models improves translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1387–1392, Seattle, Washington, USA, October.

Joern Wuebker, Stephan Peitz, Felix Rietig, and Hermann Ney. 2013. Improving statistical machine translation with word class models. In *Conference on Empirical Methods in Natural Language Processing*, pages 1377–1381, Seattle, USA, October.