The AMU-UEDIN Submission to the WMT16 News Translation Task: Attention-based NMT Models as Feature Functions in Phrase-based SMT

Marcin Junczys-Dowmunt[1,2], Tomasz Dwojak[1], Rico Sennrich[2]
[1]Faculty of Mathematics and Computer Science, Adam Mickiewicz University in Poznań
[2]School of Informatics, University of Edinburgh
junczys@amu.edu.pl t.dwojak@amu.edu.pl
rico.sennrich@ed.ac.uk

Abstract

This paper describes the AMU-UEDIN submissions to the WMT 2016 shared task on news translation. We explore methods of decode-time integration of attention-based neural translation models with phrase-based statistical machine translation. Efficient batch-algorithms for GPU-querying are proposed and implemented. For English-Russian, our system stays behind the state-of-the-art pure neural models in terms of BLEU. Among restricted systems, manual evaluation places it in the first cluster tied with the pure neural model. For the Russian-English task, our submission achieves the top BLEU result, outperforming the best pure neural system by 1.1 BLEU points and our own phrase-based baseline by 1.6 BLEU. After manual evaluation, this system is the best restricted system in its own cluster. In follow-up experiments we improve results by additional 0.8 BLEU.

1 Introduction

This paper describes the AMU-UEDIN submissions to the WMT 2016 shared task on news translation. We explore methods of decode-time integration of attention-based neural translation models with phrase-based decoding. Experiments have been conducted for the English-Russian language pair in both translation directions.

For these experiments we re-implemented the inference step of the models described in Bahdanau et al. (2015) (more exactly the DL4MT[1] variant also present in Nematus[2]) in efficient C++/CUDA code that can be directly compiled as a Moses feature function. The GPU-based computations come with their own peculiarities which we reconcile with the two most popular phrase-based decoding algorithms — stack-decoding and cube-pruning.

While it seems at first that for English-Russian our phrase-based system is holding back the neural models in terms of BLEU, the manual evaluation reveals that our systems is tied with the pure neural systems, occupying the same top cluster for restricted systems with an even slightly higher TrueSkill score. We achieve the top BLEU result for the Russian-English task, outperforming the best pure neural system by 1.1 BLEU points and our own phrase-based baseline by 1.6 BLEU. After manual evaluation, this system is the best restricted system in its own cluster.

Our implementation is available as a Moses fork from `https://github.com/emjotde/mosesdecoder_nmt`

2 Preprocessing

As we reuse the neural systems from Sennrich et al. (2016), we follow their preprocessing scheme for the phrase-based systems as well. All data is tokenized with the Moses tokenizer, for English the Penn-format tokenization scheme has been used. Tokenized text is true-cased.

Sennrich et al. (2016) use byte-pair-encoding (BPE) to achieve open-vocabulary translation with a fixed vocabulary of subword symbols (Sennrich et al., 2015b). For English, the vocabulary size is limited to 50,000 units, for Russian to 100,000. This has the interesting consequence of using subword units for phrase-based SMT. Although SMT seems to be better equipped to handle large vocabularies, the case of Russian still poses problems which are usually solved with transliteration

[1]`https://github.com/nyu-dl/dl4mt-tutorial`
[2]`https://github.com/rsennrich/nematus`

mechanisms (Durrani et al., 2014). Resorting to subword units eliminates the need for these.[3]

3 Neural translation systems

As mentioned before, we reuse the English-Russian and Russian-English NMT models from Sennrich et al. (2016) and refer the reader to that paper for a more detailed description of these systems. In this section we give a short summarization for the sake of completeness.

The neural machine translation system is an attentional encoder-decoder (Bahdanau et al., 2015), which has been trained with Nematus. Additional parallel training data has been produced by automatically translating a random sample (2 million sentences) of the monolingual Russian News Crawl 2015 corpus into English (Sennrich et al., 2015a), which has been combined with the original parallel data in a 1-to-1 ratio.[4] The same has been done for the other direction. We used mini-batches of size 80, a maximum sentence length of 50, word embeddings of size 500, and hidden layers of size 1024. We clip the gradient norm to 1.0 (Pascanu et al., 2013). Models were trained with Adadelta (Zeiler, 2012), reshuffling the training corpus between epochs. The models have been trained model for approximately 2 weeks, saving every 30000 mini-batches.

For our experiments with PB-SMT integration, we chose the same four models that constituted the best-scoring ensemble from Sennrich et al. (2016). If less than four models were used, we chose the models with the highest BLEU scores among these four models as measured on a development set.

4 Phrase-Based baseline systems

We base our set-up on a Moses system (Koehn et al., 2007) with a number of additional feature functions. Apart from the default configuration with a lexical reordering model, we add a 5-gram operation sequence model (Durrani et al., 2013).

We perform no language-specific adaptations or modifications. The two systems differ only with respect to translation direction and the available (monolingual) training data. For domain-adaptation, we rely solely on parameter tuning with Batch-Mira (Cherry and Foster, 2012) and on-line log-linear interpolation. Binary domain-indicators for each separate parallel corpus are introduced to the phrase-tables (four indicators) and a separate language model per parallel and monolingual resource is trained (en:16 and ru:12). All language models are 5-gram models with Modified Kneser-Ney smoothing and without pruning thresholds (Heafield et al., 2013). We treat different years of the News Crawl data as different domains to take advantage of possible recency-based effects. During parameter tuning on the newstest-2014 test set, we can unsurprisingly observe that weights for the last three LMs (2013, 2014, 2015) are much higher than for the remaining years.

After concatenating all resources, a large 5-gram background language model is trained, with 3-grams or higher n-gram orders being pruned if they occur only once. The same concatenated files and pruning settings are used to create a 9-gram word-class language model with 200 word-classes produced by word2vec (Mikolov et al., 2013).

5 NMT as Moses feature functions

As mentioned in the introduction, we implemented a C++/CUDA version of the inference step for the neural models trained with DL4MT or Nematus, which can be used directly with our code. One or multiple models can be added to the Moses log-linear model as different instances of the same feature, which during tuning can be separately weighted. Adding multiple models as separate features becomes thus similar to ensemble translation with pure neural models.

In this section we give algorithmic details about integrating GPU-based soft-attention neural translation models into Moses as part of the feature function framework. Our work differs from Alkhouli et al. (2015) in the following aspects:

1. While Alkhouli et al. (2015) integrate RNN-based translation models in phrase-based decoding, this work is to our knowledge the first to integrate soft-attention models.

2. Our implementation is GPU-based and our algorithms being tailored towards GPU computations require very different caching strategies from those proposed in Alkhouli et

[3]In experiments not described in this paper, we tried BPE encoding for the English-German language pair and found subword units to cope well with German compound nouns when used for phrase-based SMT.

[4]This artificial data has not been used for the creation of the phrase-based system, but it might be worthwhile to explore this possibility in the future. It might enable the phrase-based system to produce translation that are more similar to the neural output.

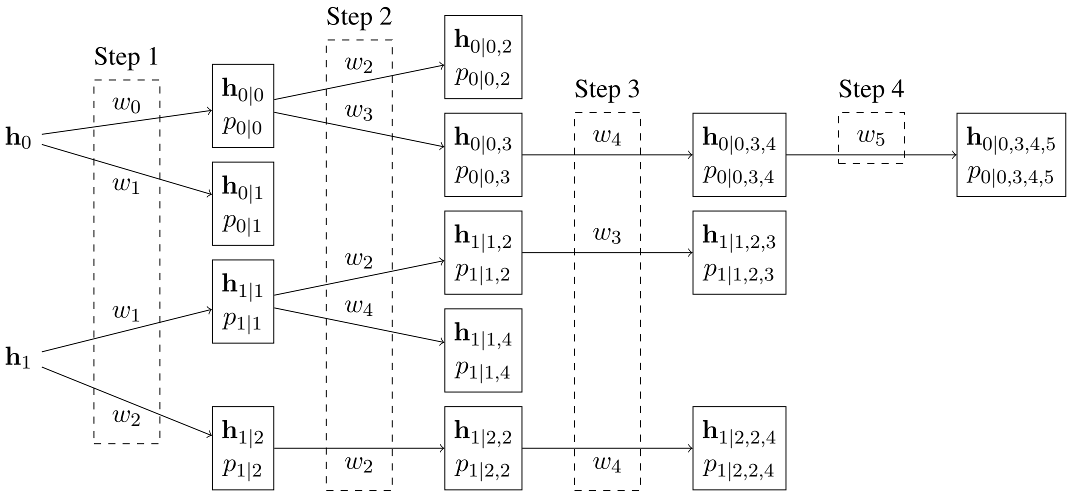
Step 1
Step 2
Step 3
Step 4
$\mathbf{h}_0$
$\mathbf{h}_1$
w_0
w_1
w_1
w_2
$\mathbf{h}_{0|0}$ $p_{0|0}$
$\mathbf{h}_{0|1}$ $p_{0|1}$
$\mathbf{h}_{1|1}$ $p_{1|1}$
$\mathbf{h}_{1|2}$ $p_{1|2}$
w_2
w_3
w_2
w_4
w_2
$\mathbf{h}_{0|0,2}$ $p_{0|0,2}$
$\mathbf{h}_{0|0,3}$ $p_{0|0,3}$
$\mathbf{h}_{1|1,2}$ $p_{1|1,2}$
$\mathbf{h}_{1|1,4}$ $p_{1|1,4}$
$\mathbf{h}_{1|2,2}$ $p_{1|2,2}$
w_4
w_3
w_4
$\mathbf{h}_{0|0,3,4}$ $p_{0|0,3,4}$
$\mathbf{h}_{1|1,2,3}$ $p_{1|1,2,3}$
$\mathbf{h}_{1|2,2,4}$ $p_{1|2,2,4}$
w_5
$\mathbf{h}_{0|0,3,4,5}$ $p_{0|0,3,4,5}$

Figure 1: SCOREBATCH procedure for a forest consisting of two per-hypothesis prefix trees. Words are collected at the same tree depths across all trees in the forest.

al. (2015). Our implementation seems to be about 10 times faster on one GPU, 30 times faster when three GPUs are used.

5.1 Scoring hypotheses and their expansions

We assume through-out this section that the neural model has already been initialized with the source sentence and that the source sentence context is available at all time.

In phrase-based machine translation, a pair consisting of a translation hypothesis and a chosen possible target phrase that expands this hypothesis to form a new hypothesis can be seen as the smallest unit of computation. In the typical case they are processed independently from other hypothesis-expansion pairs until they are put on a stack and potentially recombined. Our aim is to run the computations on one or more GPUs. This makes the calculation of scores per hypothesis-expansion pair (as done for instance during n-gram language model querying) unfeasible as repeated GPU-access with very small units of computation comes with a very high overhead.

In neural machine translation, we treat neural states to be equivalent to hypotheses, but they are extended only by single words, not phrases, by performing computations over the whole target vocabulary. In this section, we present a batching and querying scheme that aims at taking advantage of the capabilities of GPUs to perform batched calculations efficiently, by combining the approaches from phrase-based and neural decoding.

Given is a set of pairs (h, t) where h is a decoding hypothesis and t a target phrase expanding the hypothesis. In a naive approach (corresponding to unmodified stack decoding) the number of queries to the GPU would be equal to the total number of words in all expansions. A better algorithm might take advantage of common target phrase prefixes per hypothesis. The number of queries would be reduced to the number of collapsed edges in the per-hypothesis prefix-tree forest.

```
1: procedure SCOREBATCH($L$, NMT)
2:     Create forest of per-hypothesis prefix trees from all hypotheses and expansions in $L$
3:     for $i$ from 1 to maximum tree depth do
4:         Construct embedding matrix $E_i$ from all edge labels at depth $i$
5:         Construct row-wise corresponding state matrix $H_{i-1}$ from source nodes
6:         Compute forward step: $(H_i, P_i) \leftarrow \text{NMT}(H_{i-1}, E_i)$
7:         Cache state pointers and probabilities at target nodes
```

Figure 2: Scoring of hypothesis expansion pairs

By explicitly constructing this forest of prefix trees where a single prefix tree encodes all target phrases that expand the same hypothesis, we can actually reduce the number of queries to the neural model to the maximum depth of any of the trees (i.e. the maximum target phrase length) as illustrated in Figures 1 and 2.

Target phrases t are treated as sequences of words w. Rectangles at tree nodes should be imagined to be empty before the preceding step has been performed. The first embedding matrix E_1 is constructed by concatenating embedding vectors $\mathbf{e}_i \leftarrow \textsc{Lookup}(w_i)$ as rows of the matrix, for all w_i marked in the first dashed rectangle. The initial state matrix H_0 is a row-wise concatenation of the neural hypothesis states, repeated for each outgoing edge. Thus, the embedding matrix and state matrix have the same number of corresponding rows. Example matrices for the first step take the following form:

$$E_1 = \begin{bmatrix} \mathbf{e}_0 \\ \mathbf{e}_1 \\ \mathbf{e}_1 \\ \mathbf{e}_2 \end{bmatrix} \quad H_0 = \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_0 \\ \mathbf{h}_1 \\ \mathbf{h}_1 \end{bmatrix}$$

Given the precomputed source context state, we can now perform one forward step in the neural network which yields a matrix of output states and a matrix of probabilities, both corresponding row-wise to the input state matrix and embedding matrix we constructed earlier. The target nodes for each edge pointed to after the first step are filled. Probabilities will be queried later during phrase-based scoring, neural hypothesis states are reused to construct the state matrix of the next step and potentially as initial states when scoring another batch of hypotheses at later time.

5.2 Two-pass stack decoding

Standard stack decoding still scores hypotheses one-by-one. In order to limit the number of modifications of Moses to a minimum, we propose two-pass stack decoding where the first pass is a hypothesis and expansions collection step and the second pass is the original expansion and scoring step. Between the two steps we pre-calculate per-hypothesis scores with the procedure described above. The data structure introduced in Figure 1 is then reused for probability look-up during the scoring phrase of stack decoding as if individual hypotheses where scored on-the-fly.

Figure 3 contains our complete proposal for two-pass stack decoding, a modification of the original stack decoding algorithm described in Koehn (2010). We dissect stack decoding into smaller reusable pieces that can be passed functors to perform different tasks for the same sets of hypotheses. The main reason for this is the small word “applicable” in line 12, which hides a complicated set of target phrase choices based on reordering limits and coverage vectors which should not be discussed here. This allows our algorithm to collect exactly the set of hypotheses and expansions for score pre-calculation that will be used during the second expansion step.

```
1: procedure TwoPassStackDecoding
2:     Place empty hypothesis $h_0$ into stack $S_0$
3:     for stack $S$ in stacks do
4:         $L \leftarrow \emptyset$
5:         ProcessStack($S$, Gather{$L$})
6:         $C \leftarrow$ ScoreBatch($L$, Nmt)
7:         ProcessStack($S$, Expand{$C$})
8:
9: procedure ProcessStack($S$, $f$)
10:     for hypothesis $h$ in $S$ do
11:         for target phrase $t$ do
12:             if applicable then
13:                 Apply functor $f(h, t)$
14:
15: procedure Gather($h$, $t$)
16:     $L \leftarrow L \cup \{(h, t)\}$
17:
18: procedure Expand($h$, $t$)
19:     Look-up $p$ for $(h, t)$ in $C$
20:     Create new hypothesis $\hat{h}$ from $(h, t, p)$
21:     Place $\hat{h}$ on appropriate stack $s$
22:     if possible then
23:         Recombine hypothesis $\hat{h}$ with other
            hypotheses on stack $s$
24:     if stack $s$ too big then
25:         Prune stack $s$
```

Figure 3: Two-pass stack decoding

As already mentioned, the number of forward steps for the NMT network per stack is equal to the greatest phrase length among all expansions. The total number of GPU queries increases therefore linearly with respect to the sentence length. Branching factors or stack sizes affect the matrix sizes, not the number of steps.[5]

For this method we do not provide results due to a lack of time. We confirmed for other experiments that improvements are smaller than for the next method. A comparison will be provided in an extended version of this work.

5.3 Stack rescoring

The previous approach cannot be used with lazy decoding algorithms — like cube pruning —

[5]Large matrix sizes, however, do slow-down translation speed significantly.

```
procedure StackRescoring
    Place empty hypothesis $h_0$ into stack $S_0$
    for stack $S$ in stacks do
        $L \leftarrow \emptyset$
        for hypothesis $h$ in $S$ do
            Extract predecessors $(\bar{h}, \bar{t})$ from $h$
            $L \leftarrow L \cup \{(\bar{h}, \bar{t})\}$
        $C \leftarrow$ ScoreBatch($L$, NMT)
        for hypothesis $h$ in $S$ do
            Extract predecessors $(\bar{h}, \bar{t})$ from $h$
            Look-up $p$ for $(\bar{h}, \bar{t})$ in $C$
            Recalculate score of $h$ using $p$
        Create cache $C_0$ with 0-probabilities
        ProcessStack($S$, Expand$\{C_0\}$)
```

Figure 4: Stack decoding with stack rescoring

which has also been implemented in Moses. Apart from that, due to the large number of expansions even small stack sizes of around 30 or 50 quickly result in large matrices in the middle steps of BatchScore where the prefix trees have the greatest number of edges at the same depth level. In the worst case, matrix size will increase by a factor b^d, where b is the branching factor and d is the current depth. In practice, however, the maximum is reached at the third or fourth step, as only few target phrases contain five or more words.

To address both shortcomings we propose a second algorithm: stack rescoring. This algorithm (Figure 4) relies on two ideas:

1. During hypothesis expansion the NMT feature is being ignored, only probabilities of 0 are assigned for this feature to all newly created hypotheses. Hypothesis recombination and pruning take place without NMT scores for the current expansions (NMT scores for all previous expansions are included). Any stack-based decoding algorithm, also cube-pruning, can be used in this step.

2. The BatchScore procedure is applied to all direct predecessors of hypotheses on the currently expanded stack. Predecessors consist of the parent hypothesis and the expansion that resulted in the current hypothesis. The previously assigned 0-probabilities are replaced with the actual NMT scores.

This procedure results in a number of changes when compared to standard stack decoding approaches and the previous method:

- The maximum matrix row count is equal to the stack size, and often much smaller due to prefix collapsing. Branching factors are irrelevant and stack sizes of 2,000 or greater are possible. By contrast, for two-pass stack decoding stack sizes of around 50 could already result in row counts of 7,000 and more.

- With cube pruning, by setting cube pruning pop-limits much larger than the stack size many more hypotheses can be scored with all remaining feature functions before the survivors are passed to BatchScore.

- Scoring with the NMT-feature is delayed until the next stack is processed. This may result in missing good translations due to recombination. However, the much larger stack sizes may counter this effect.

- N-best list extraction is more difficult, as hypotheses that have been recombined do not display correct cumulative sums for the NMT-feature scores. The one-best translation is always correctly scored as it has never been discarded during recombination, so there is no problem at test time. For tuning, where a correctly scored n-best list is required, we simply rescore the final n-best list with the same neural feature functions as during decoding. The resulting scores are the same as if they were produced at decode-time. Final n-best list rescoring can thus be seen as an integral part of stack-rescoring.

6 Experiments and results

For decoding, we use the cube-pruning algorithm with stack size of 1,000 and cube-pruning pop limit of 2,000 during tuning. At test time, a stack-size of 1,000 is kept, but the cube-pruning pop limit is increased to 5,000. We set a distortion limit of 12. We run 10 iterations of Batch-Mira (Cherry and Foster, 2012) and choose the best set of weights based on the development set. Our development set is a subset of 2,000 sentences from the newstest-2014 test set. Sentences have been selected to be shorter than 40 words to avoid GPU-memory problems. Our GPUs are three Nvidia GeForce GTX-970 cards with 4GB RAM each.

In this paper, similar as Alkhouli et al. (2015), we ignore the implications of the infinite neural state and hypothesis recombination in the face of

System	2015	2016
Phrase-Based (PB)	23.7	22.8
Pure neural:		
NMT-2	26.4	25.3
NMT-4 (Sennrich et al., 2016)	27.0	26.0
Stack rescoring:		
PB+NMT-2 (subm.)	—	**25.3**
Follow-up:		
NMT-4-Avg	26.7	25.5
PB+NMT-4-Avg	27.3	25.9

(a) BLEU scores English-Russian

System	2015	2016
Phrase-Based (PB)	27.4	27.5
Pure neural:		
NMT-3	28.3	27.8
NMT-4 (Sennrich et al., 2016)	28.3	28.0
Stack rescoring:		
PB+NMT-3 (subm.)	**29.5**	**29.1**
Follow-up:		
NMT-10-Avg	28.3	28.1
PB+NMT-10-Avg	**30.2**	**29.9**

(b) BLEU scores Russian-English

Table 1: Systems marked with **subm.** are our final WMT 2016 submissions.

	words/s
Alkhouli et al. (2015) (1 thread?)	0.19
Phrase-based PB (24 threads)	40.30
PB-NMT-10-Avg (3 GPUs)	4.83

Table 2: Translation speed for different configurations in words per second.

infinite state. We rely on the hypothesis recombination controlled by the states of the other feature functions. It is worth mentioning again that our phrase-based baseline features a 9-gram word-class language model which should be rather prohibitive of recombinations. If recombination was only allowed for hypotheses with the same partial translations, results were considerably worse.

6.1 Speed

Translation speed is difficult to compare across systems (Table 2). Even with three GPUs our system is ten times slower than than a pure PB-SMT system running with 24 CPU-threads. It is however unclear at this moment if the large stack sizes we use are really necessary. Significant speed-up might be achieved for smaller stacks.

6.2 Submitted results

Table 1 summarizes the results for our experiments. BLEU scores are reported for the newstest-2015 and newstest-2016 test sets.

Our baseline phrase-based systems (PB) are quite competitive when comparing to the best results of last year's WMT (24.4 and 27.9 for English-Russian and Russian-English, respectively). NMT-4 is the best pure neural ensemble from Sennrich et al. (2016) for both translation directions. Due to memory restrictions, we were not able to use all four models as separate feature functions and limit ourselves to the best two models for English-Russian and best three for Russian-English. The pure neural ensembles are NMT-2 (en-ru) and NMT-3 (ru-en), respectively.

For English-Russian, our results stay behind the pure-neural 4-ensemble NMT-4 in terms of BLEU. In a direct comparison between ensembles of 2 models (PB+NMT-2 and NMT-2), we actually reach similar BLEU scores. However, in the manual evaluation our system is best restricted system, tied with the neural system. Absolute TrueSkill scores are even slightly higher for our system.

For Russian-English the best-performing pure neural system NMT-4 and the phrase-based baseline are only 0.5% BLEU apart. Adding three NMT models as feature functions to Moses results in a 1.1% BLEU improvement over the neural model and 1.6% over the phrase-based system. The systems PB-NMT-2 (en-ru) and PB-NMT-3 (ru-en) are our submissions to the WMT-2016 news translation task. PB-NMT-3 scores the top BLEU results for Russian-English. In the manual evaluation, our system is the best restricted system in its own cluster.

6.3 Follow-up experiments

Frustrated by the limited memory of our GPU cards and against better knowledge[6], we computed

[6]The neural network lore seems to suggest that this should not work, as neural networks are non-linear models. We only found one paper with evidence to the contrary: Utans (1996)

the element-wise average of all model weights in the NMT ensembles and saved the resulting model. Interestingly, the performance of these new models (NMT-4-Avg) is not much worse than the actual ensemble (NMT-4), while being four times smaller and four times faster at decode-time. The average models outperforms any single model or the smaller 2-ensembles. All models taking part in the average are parameter dumps saved at different points in time during the same training run. This seem to be an interesting results for model compression and deployment settings. We can also average more models: for the Russian-English direction we experiment with the parameter-wise average of ten models (NMT-10-Avg) which even slightly outperforms the real four-model ensemble NMT-4.

With this smaller model it is easier to tune and deploy our feature function. The performance of our combined setup improves for both translation directions. For English-Russian, however, the pure NMT system (NMT-4) remains ahead of our WMT 2016 submission. For Russian-English we get another improvement of 0.8 BLEU, which sets the new state-of-the-art for this direction.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements 644333 (TraMOOC) and 688139 (SUMMA) and was partially funded by the Amazon Academic Research Awards programme.

References

Tamer Alkhouli, Felix Rietig, and Hermann Ney. 2015. Investigations on phrase-based decoding with recurrent neural network language and translation models. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 294–303, Lisbon, Portugal, September. Association for Computational Linguistics.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Colin Cherry and George Foster. 2012. Batch tuning strategies for statistical machine translation. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL HLT '12, pages 427–436, Stroudsburg, PA, USA. Association for Computational Linguistics.

Nadir Durrani, Alexander Fraser, Helmut Schmid, Hieu Hoang, and Philipp Koehn. 2013. Can Markov models over minimal translation units help phrase-based SMT? In *ACL*, pages 399–405. The Association for Computer Linguistics.

Nadir Durrani, Hassan Sajjad, Hieu Hoang, and Philipp Koehn. 2014. Integrating an unsupervised transliteration model into statistical machine translation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2014, April 26-30, 2014, Gothenburg, Sweden*, pages 148–153.

Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. 2013. Scalable modified Kneser-Ney language model estimation. In *Proceedings of the 51st Annual Meeting of the ACL*, pages 690–696.

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*, pages 177–180. ACL.

Philipp Koehn. 2010. *Statistical Machine Translation*. Cambridge University Press, New York, NY, USA, 1st edition.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, pages 1310–1318, , Atlanta, GA, USA.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015a. Improving Neural Machine Translation Models with Monolingual Data. *ArXiv e-prints*, November.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015b. Neural Machine Translation of Rare Words with Subword Units. *CoRR*, abs/1508.07909.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Edinburgh Neural Machine Translation Systems for WMT 16. In *Proc. of the Conference on Machine Translation (WMT)*, Berlin, Germany.

Joachim Utans. 1996. Weight averaging for neural networks and local resampling schemes. In *Proc. AAAI-96 Workshop on Integrating Multiple Learned Models*, pages 133–138. AAAI Press.

Matthew D. Zeiler. 2012. ADADELTA: An Adaptive Learning Rate Method. *CoRR*, abs/1212.5701.