The RWTH Aachen University Filtering System for the WMT 2018 Parallel Corpus Filtering Task

Nick Rossenbach, Jan Rosendahl, Yunsu Kim,
Miguel Graça, Aman Gokrani, Hermann Ney
Human Language Technology and Pattern Recognition Group
Computer Science Department
RWTH Aachen University
D-52056 Aachen, Germany
<surname>@i6.informatik.rwth-aachen.de

Abstract

This paper describes the submission of RWTH Aachen University for the De→En parallel corpus filtering task of the *EMNLP 2018 Third Conference on Machine Translation* (WMT 2018). We use several rule-based, heuristic methods to preselect sentence pairs. These sentence pairs are scored with count-based and neural systems as language and translation models. In addition to single sentence-pair scoring, we further implement a simple redundancy removing heuristic. Our best performing corpus filtering system relies on recurrent neural language models and translation models based on the transformer architecture. A model trained on 10M randomly sampled tokens reaches a performance of 9.2% BLEU on newstest2018. Using our filtering and ranking techniques we achieve 34.8% BLEU.

1 Introduction

In this work we describe the corpus filtering system of the RWTH Aachen University for the WMT 2018 parallel corpus filtering task.

We decided to rank the data using a two-stage process. During the first stage, we reduce the number of parallel sentences by applying basic rule-based heuristics each of whom can reject a sentence as described in Section 3. Afterward, we apply a variety of models on the remaining sentences to assign a score to each sentence pair. The details of those models, namely language models and translation models, can be found in Section 4.

Our final submission consists of three different systems on top of rule-based filtering: Two of them are based on scoring each sentence pair independently using either only count-based models or only neural models. The third submission extends on the neural network-based submission by removing redundancies before ranking the sentences.

We compare the behavior of neural network based models to count-based models and find that the performance differs by more than 1.0 % BLEU on average across all test sets. In total our best system reaches a performance of 34.8 % BLEU compared to 9.2 % BLEU using random sampling on newstest2018 of the news translation task with 10M token subsampled training data. We report our findings and results in detail in Section 6.

2 Preprocessing

As a first step, we normalize the data by removing soft-hyphen and zero-width space symbols. Furthermore, we replace all hash symbols (#) because we use them as separation symbol. A language specific tokenizer from Moses (Koehn et al., 2007) is applied to both sides of the corpus. We later found out that this language specific splitting can cause some issues if equal patterns are not split the same way on source and target side (see Section 3.6).

After tokenization, we search for and replace any escaped characters with the corresponding symbol and squeeze repeating whitespaces. We true-case our words by applying a frequent casing model from the Jane toolkit (Vilar et al., 2010) based on the parallel corpora.

If data is used to train the count-based models or if we apply the count-based models on a sentence pair, numbers are replaced by a category symbol. For the neural models, we generate joint BPE merge operations on the parallel training data with 20k merge operations (Sennrich et al., 2016). For the training of the neural scoring models, we create BPE vocabularies based on the clean parallel data of the WMT news task[1], and use a vocabulary threshold of 50. For evaluation of the subsampled data, we did not use a vocabulary threshold.

[1] CommonCrawl, Europarl, NewsCommentary, Rapid

Proceedings of the Third Conference on Machine Translation (WMT), Volume 2: Shared Task Papers, pages 959–967

https://doi.org/10.18653/v1/W18-6488

3 Rule-based Filtering

As the scoring of 104 million sentence pairs is hardly feasible with computationally expensive models like a transformer model (Vaswani et al., 2017), we need to preselect a smaller subset of the data. We do this by applying several rule-based heuristics as a first stage of our data cleaning pipeline. A sentence pair is removed from the corpus if its source side or target side fail to obey any of these rules. Note that none of these rules is language specific to either German or English and that they place only very mild assumptions on what 'good data' should look like.

Besides reducing the amount of data, some of the heuristic filtering methods can deal with aspects that can not be captured with language models and are hard to cover by translation models (see Subsection 6.4).

Table 1 shows the amount of remaining sentence pairs and tokens after applying each heuristic in sequential order.

3.1 Minimum Words

Our first heuristic filter ensures that every sentence contains at least a certain number of words. To do so, we count the number of tokens (i.e. character sequence between two spaces) that contain at least one letter from the alphabet of the language. Thus numbers or punctuation symbols are not counted as words according to this definition. A sentence is only valid if this number reaches a certain threshold (which we set to 3 for all our experiments).

3.2 Average Word Length

In the next step of the filtering process, we remove long chains of characters and sequences where only single characters appear. This aims in particular for lines which consist mainly of a single, very long URL. Although we did not expect a lot of sentence pairs to have an average word length lower than 2 or bigger than 20 characters, we removed about 1% of the sentence pairs with this procedure.

3.3 Length Ratio

Judging sentence pairs by the ratio of source sentence vs. target sentence length is a very simple but effective criterion. We limited this length ratio to be not greater than 1.7. Because of tokenization, all punctuation symbols are counted as single words. To smooth the ratio for shorter sentences, we always add 1 to the token count, i.e. we reject the sentence if:

$$\frac{J+1}{I+1} > 1.7 \vee \frac{I+1}{J+1} > 1.7$$

where I is the target and J the source sequence length.

3.4 Maximum Sentence Length

Because many translation systems have an upper bound for the sentence length during training and to reduce the computational cost of our scoring models, we limited the maximum number of tokens to 50.

3.5 Maximum Subword-Token Length

As scoring with Sockeye (Hieber et al., 2017) transformer model requires a maximum sequence length as fixed parameter, we enforce a limit on the number of subword units. The subword merge operations are computed on the parallel WMT 2018 news training data, excluding the filtered ParaCrawl data. We limited each sentence to consist of a maximum of 100 subword tokens.

3.6 Levenshtein Distance

In our experiments we observe that the transformer model tends to assign a very high score to sentence pairs in which source and target share a great number of words. This happens even if neither the given source nor the given target sentence are in the correct language. It seems that the model regards copying as a valid form of translation. To detect sentences where source and target are too similar, we compute the word-level Levenshtein distance D (Levenshtein, 1966) between the lowercased sentences . We also take into account a length normalized Levenshtein distance $\bar{D} = \frac{D}{I+J}$. A sentence is rejected if:

$$D \leq 1 \vee \bar{D} \leq 0.15$$

These values were determined by visually looking at 100k random examples ranked by $\bar{D}$ and ensuring that no valid looking sentence gets removed. The language specific tokenizers sometimes split the same sequence differently depending on the language, which increases the distance e.g.:

the do ' s and don ' ts of the audience .
the do 's and don 'ts of the audience .

Method	pairs	de tok.	en tok.	del.%	del. total
Original Data	104.0M	1,520M	1,562M		
Min. Words (3.1)	61.9M	1,276M	1,313M	40.45%	42.0M
Avg. Word Length (3.2)	61.3M	1,262M	1,298M	0.94%	0.6M
Length Ratio (3.3)	50.6M	1,072M	1,092M	17.60%	10.8M
Max. Seq. Length (3.4)	46.0M	625M	642M	8.91%	4.5M
Max. Seq. Length (BPE) (3.5)	46.0M	622M	638M	0.20%	0.1M
Levenshtein (3.6)	36.6M	512M	528M	20.26%	9.3M
Word Token Ratio (3.7)	28.1M	398M	412M	23.38%	8.6M
Redundancy (3.8)	13.0M	227M	236M	53.84%	15.1M

Table 1: Sizes of datasets after applying the heuristic filtering methods. Sizes are given in sentence pairs, tokens on German side and tokens on English side. Every heuristic is applied on top of the preceding heuristic. The last two columns show the percentage (with respect to its input not the original corpus) respectively the absolute number of lines removed by a heuristic.

Thus, the Levenshtein heuristic sometimes misses some sentence pairs that should have been removed.

3.7 Word Token Ratio

We extend the idea of minimum word filtering from Section 3.1 to scale with sentence length. We count the number of tokens that contain at least one character that is a standard alphabet letter. If this count is less than 60% of the total sentence length, we reject the sentence. This can be helpful to remove sentences from languages with different alphabets or lines which simply consist of a time and date. Also, sentences with more than 60% numbers and punctuation symbols are removed.

3.8 Redundancy

To increase the amount of information in the subsampled data, we wanted to remove redundant information. Checking the redundancy of a sentence in the context of a big corpus is challenging, as trivial algorithms need to do $\frac{C^2}{2}$ comparisons for corpus size C which is not feasible for large datasets. One simple solution for removing identical sentences in linear time is to compute a hash value[2] for each sentence, and check for existing hashes in a set. We extended this approach to detect 'similar' sentences by removing each word individually and store the hash of the remaining sentence. By doing this we also remove sentences that have a word edit distance of one compared to any previously added sentence. We do not distinguish between source or target side sentences, both are stored in the same set. A simple pseudo-code description is shown in Algorithm 1.

Algorithm 1: Duplicate checking

$hm \leftarrow empty\_hashmap()$
for *each sentence* s_1^N **do**
  $sent\_hm \leftarrow empty\_hashmap()$
  for *each position* $i \in [1, N]$ **do**
    $h \leftarrow hash([s_1^{i-1}, s_{i+1}^N])$
    if $h \in hm$ **then**
      reject s_1^N
      break
    else
      $sent\_hm$.add(h)
  if s_1^N **not** $rejected$ **then**
    hm.add($sent\_hm$)

4 Model-based Scoring

In the second stage of our filtering pipeline we score each sentence using different kinds of language and translation models. Every model assigns a probability to each sentence. These scores are used afterward to rank the corpus and select the top sentences.

4.1 Count-Based Language Model

To score the remaining sentences, we start by applying count based language models on each side of the parallel sentences. The language models used are 5-gram KenLM (Heafield et al., 2013) models with singleton tri-gram pruning and trained with modified interpolated kneser-ney smoothing (Chen and Goodman, 1996). They

[2]We use the python3 default hash() function

are trained on the NewsCrawl 2016, Europarl, NewsCommentary and Rapid corpora from the WMT 2018 German→ English task. Adding NewsCrawl 2012-2015 as further monolingual training sets does not achieve better results.

We apply the preprocessing mentioned in Section 2 and we remove any sentence from the training data that contains token repetitions of length three or more. This is done to get rid of phenomenons like chains of exclamation marks. For more details about the data selection see Section 6.2.

4.2 IBM1 Dictionary Model

IBM1 models are a simple approach to model the dependency $p(e_1^I|f_1^J)$, as they assume a uniform alignment. We train the model with the GIZA++ toolkit (Och and Ney, 2003) on the parallel data to create an IBM1 table. IBM model 1 scores are computed as in (Brown et al., 1993):

$$p(e_1^I|f_1^J) = \frac{1}{(J+1)^I} \prod_{i=1}^{I} \sum_{j=0}^{J} p(e_i|f_j) \quad (1)$$

where I and J are the length of the target respectively source sentence, and f_0 is a null token. We train IBM1 models for both directions (s2t and t2s) using the bilingual data from the WMT 2018 German↔English task namely the Europarl, CommonCrawl, NewsCommentary and Rapid corpus.

4.3 Neural Network Language Model

We modified the RWTH Aachen translation system as described in (Peter et al., 2017) based on the Blocks framework (van Merriënboer et al., 2015) and Theano (Theano Development Team, 2016) to also work as a recurrent language model. The training data is chosen to be equivalent to the one used in the training of the count-based models. The language model has an embedding size of 250 and two LSTM layers (Hochreiter and Schmidhuber, 1997) with a hidden size of 1000. As it is default in Blocks, it also includes a maxout layer of factor 2 (Goodfellow et al., 2013) between the second LSTM and the output softmax. The system was trained using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 0.001 for 300k iterations with a batch size of 100 sentences and a dropout (Srivastava et al., 2014) of 0.2.

4.4 Transformer Translation Model

As neural network-based translation model we use the transformer architecture (Vaswani et al., 2017) implemented in the Sockeye toolkit (Hieber et al., 2017) which is build on top of MXNet (Chen et al., 2015). Encoder and decoder each consist of 6 layers. The hidden and embedding size is set to be 512 and the feed forward layer size is 2048. The number of attention heads is 8. A dropout of 0.1 is applied, except for the embedding layer. We use an initial learning rate of 0.0002. We save checkpoints every 20k iterations, and reduce the learning rate by factor 0.7 after each non-improving checkpoint (measured by means of perplexity on newstest2015). The network is trained on the bilingual data from the WMT 2018 German↔English task namely the Europarl, CommonCrawl, NewsCommentary and Rapid corpus.

5 Evaluation Model

To check the quality of a filtering approach, we train a transformer model on the top 10M respectively top 100M subwords of the scored training data. We mainly focus on the 10M-subsampling results, as this scenario shows clearer differences in performance between different methods. Like in Section 4.4 we use the Sockeye implementation of the transformer architecture but we train smaller models for evaluation purposes. The decision to use small transformer networks was made as they give strong results in a much shorter amount of time (1 day compared to 5 days). To verify the generality of the approach we cross-checked several experiments using recurrent neural network-based (RNN-based) translation systems from the Marian framework (Junczys-Dowmunt et al., 2018) and found their training behavior to be correlated.

We apply the provided subsampling script on the filtered data to extract training data. Due to an error in our filtering setup the input data is tokenized and subworded. Since both procedures increase the number of tokens per sentence, we extract less sentence pairs than intended.

Note that because of these two effects (transformer and lower subsampling rate) BLEU scores reported in this submission can vary in comparison to other submissions, if RNN-based systems are used.

For the transformer model used for evaluation,

we use only 3 layers in encoder and decoder and increase the batch size to 8,000 words. We train for 100k updates evaluating a checkpoint every 10k updates for the 10M word experiments. In the case of 100M words of training data, 200k update steps are performed with a checkpoint being written every 20k updates. The best checkpoint is selected by computing BLEU (Papineni et al., 2002) on newstest2015. The beam-size for translation is 12. We report BLEU scores using mteval from the Moses toolkit (Koehn et al., 2007) and TER scores (Snover et al., 2006) using TERcom.

During system building most of our design decisions are based on results for the 10M-word-version of the task, however, we observe very similar trends for the 100M-word subsampling. For brevity we report most results only on the smaller subsamples.

6 Experimental Evaluation

In this section we report the results of our filtering experiments. We use newstest2015 and newstest2017 as development sets and report the results on newstest2018. For brevity, we shorten the names of newstestX to tstX in the header of several tables.

All BLEU and TER scores reported in this sections are obtained by using the system under consideration as filtering system and training the transformer system described in Section 5 on the resulting training data. All processing steps and experiments are organized with Sisyphus (Peter et al., 2018) as workflow manager.

6.1 Rule-based Filtering

The purpose of the rule-based heuristics is not to select perfect training data, but rather to reduce the original 104M lines of the ParaCrawl corpus down to an amount that can be handled by stronger, computationally more complex, methods. Table 2 and Table 3 show the evaluation results for different levels of heuristic cleaning for 10M subsampling and 100M subsampling respectively. Since there is no score-based ranking yet, we sample the desired amount of data randomly from the filtered corpus. Although a big part of the corpus is removed (58M sentences or 60% of the original corpus), the first 5 heuristic steps (3.1)-(3.5) have nearly no impact on the data quality. Applying the Levenshtein distance heuristic (see Section 3.6) resulted in a strong increase of data quality to

Filtering	tst15		tst17		tst18	
10M	BLEU	TER	BLEU	TER	BLEU	TER
Unfiltered	8.3	87	8.3	87.3	9.2	85
(3.1) - (3.5)	8.4	82.1	8.7	82.1	10.2	79.4
(3.1) - (3.6)	18.7	64.4	19.1	65.1	22.8	59.0
(3.1) - (3.7)	20.1	61.5	20.0	62.7	25.0	56.0
(3.1) - (3.8)	23.3	56.7	23.5	57.3	28.9	50.6

Table 2: Model evaluation of 10M random sampling from the datasets created by rule-based heuristic filtering.

Filtering	tst15		tst17		tst18	
100M	BLEU	TER	BLEU	TER	BLEU	TER
Unfiltered	9.1	84.7	8.6	85.2	10.6	80.9
(3.1) - (3.5)	10.8	78.2	10.3	79.5	12.6	75.5
(3.1) - (3.6)	23.2	59.0	23.2	60.8	29.1	52.4
(3.1) - (3.7)	23.8	57.9	23.8	59.4	30.0	50.8
(3.1) - (3.8)	27.2	53.1	27.3	53.5	33.8	45.8

Table 3: Model evaluation of 100M random sampling from the datasets created by rule-based heuristic filtering.

an average of 20.2% BLEU. This increase occures despite removing only 20% of the sentence pairs. Applying the word-token-ratio-heuristic (see Section 3.7) has a lesser impact, but still increases the evaluation scores by about 1.0% BLEU for 10M and about 0.6% BLEU for 100M subsampled data. Checking for redundant sentences increases the scores by up to 3.9% BLEU. This is not suprising, because as more than 50% of the sentences are removed, we replace 50% of the random selected data by potentially more informative examples.

6.2 Model-based Scoring

While the heuristics alone already result in quite satisfying cleaning results, the scoring models are used to create a ranking of the remaining sentences.

We use the corpus cleaned by the heuristics (3.1)-(3.6) as starting point for the following experiments.

In our first experiments we test the behavior of the models presented in Section 4 in isolation. Note that all our language model experiments always rely on a source and a target side language model each scoring the corresponding part of the sentence pair. All experiments with IBM1 or transformer models use a combination of a source-to-target and a target-to-source model. We average the log probabilities of the models to get a single

	System (10M)	newstest15		newstest17		newstest18	
		BLEU	TER	BLEU	TER	BLEU	TER
#1	(3.1)-(3.6) random sampling	18.7	64.4	19.1	65.1	22.8	59.0
#2	KenLM	21.3	62.1	21.2	63.6	25.8	56.5
#3	BlocksLM	23.3	59.6	23.2	60.9	28.1	54.6
#4	IBM	24.7	55.2	25.2	55.3	31.3	47.7
#5	Transformer	24.2	55.8	24.2	56.3	30.2	48.7
#6	KenLM + IBM	26.8	53.8	26.9	54.3	33.0	46.7
#7	+ Word Token Ratio (3.7)	26.6	53.9	27.0	54.0	33.1	46.2
#8	+ Redundancy (3.8)	27.2	53.5	27.1	53.9	33.4	46.2
#9	+ **IBM retraining**[1]	27.2	53.3	27.6	53.4	33.4	46.1
#10	BlocksLM + IBM	27.2	53.6	27.4	53.7	33.5	46.1
#11	BlocksLM + Transformer	28.1	52.4	28.4	52.4	34.6	45.0
#12	+ **Word Token Ratio (3.7)**[2]	28.0	52.6	28.3	52.6	34.4	45.1
#13	+ **Redundancy (3.8)**[3]	28.1	52.3	28.3	52.3	34.8	44.8
#14	KenLM + IBM + BlocksLM + Trans.	27.5	53.0	27.8	53.7	33.5	46.0

Table 4: Results for 10M word subsampling when applying different scoring models on already filtered data. All models are scoring the data that was filtered with methods described in Section 3.1 to 3.6. For model-based filtering, both source and target sides are scored.
[1]: Submission 1 with name **rwth-count**
[2]: Submission 2 with name **rwth-nn**
[3]: Submission 3 with name **rwth-nn-redundant**

score, where 0 is the best and all other scores are negative. For our submission, we added a score of -1000 for rejected sentence pairs.

From Table 4 we can see that all 4 trained models improve the heuristic filtering by more than 2.0% BLEU. Note that BlocksLM achieves better filtering results than the count-based KenLM system. However neural systems provide weaker cleaning when it comes to translation models. We are not sure why transformer performs up to 1.1% BLEU and 1.0% TER worse than IBM1 models in standalone comparison. A possible explanation is that the transformer model prefers very short sentences when not combined with a language model. For 10M subsampling, the IBM1 model ranks sentences with an average sentence length of 20 as best, while for the transformer model it is only 10.6. Combined with a langauage model, this value increases to 17.6. As can be seen from Table 4 Row #10 vs #11 this effect disappears when both systems are extended with the same language model. In this case the purely neural-network-based system has a consistent lead of roughly 1.0 % BLEU.

From Table 4 we observe that language models generally perform worse in cleaning than translation models. This could be due to the fact that many kinds of noise, which can be detected by only looking at either the source or the target sentence, are already removed by the heuristics.

Combining KenLM with an IBM1 model improves the BLEU score by 1.8% on average over IBM1 models and by 6.1% BLEU over KenLM. Adding the word to token ratio (3.7) does not affect the system performance. Note that word to token ratio was quite effective when only heuristic filtering is used (Table 2). This underlines the assumption that our heuristics remove sentence pairs, which would be sorted out by trained models anyhow. To close the gap between the count-based and neural-network-based filtering, we retrain the IBM model using its original training data plus the top 500k sentences selected from the to-be-cleaned ParaCrawl corpus, which was filtered using transformer. This improves the system by up to 0.6% BLEU but the results are still more than 0.7% BLEU behind a similar neural-network based filtering system (see Table 4 Row #9 vs. Row #12).

We achieve the best performance by combining the BlocksLM with the transformer translation systems plus the word token ratio and redundancy

System	perplexity		tst17 eval	
	de	en	BLEU	TER
KenLM	282.2	150.5	26.9*	54.3*
+ CommonCrawl	277.6	146.6	25.8*	55.6*
BlocksLM	111.07	120.62	27.4*	53.7*
+ CommonCrawl	110.55	117.7	26.4*	54.5*

Table 5: Comparison of language model perplexity with its performance as data cleaning system as well as the effect of CommonCrawl on LMs.
* For KenLM filtering results we combine the corresponding LM scores of source, target and two fixed IBM1 scores.

heuristic. The resulting system uses heuristics to filter a corpus of 104M lines down to 13M sentence pairs without the need to apply any complex model. This part of the pipeline is cheap and fast, and already gives a performance of 23.3 % BLEU on newstest2015 (see Table 2). Applying strong translation and language models yields an additional improvement of 4.8% BLEU as is shown in Table 4.

6.3 Noisy Data Effect

To investigate the effect of noisy training data for the scoring models, we add the CommonCrawl corpus to the language model training data. Although the perplexity on the dev set improves slightly for both model architectures (see Table 5), the evaluation results for the subsampled data drop by about 1.0 % BLEU. This indicates that the models are required to not only recognize good sentences well, but also to give low scores to bad sentences. If the training data contains more noisy data, a model will give higher scores to bad sentences. While this is usually a smaller problem for translation models, in terms of sentence ranking it is an important issue.

6.4 Levenshtein Distance

Table 6 shows the effect of the Levenshtein heuristic on count-based and neural scoring models. While removing sentence pairs with similar source and target does not change the performance when ranking with count-based models, it increases the performance of neural models by up to 1.0% BLEU. This confirms the assumption from Section 3.6 that transformer-based models assign high scores when copying sentences. We regard Levenshtein-based filtering as a crucial heuristic when ranking sentence pairs with neural models.

6.5 Submission Results

Table 7 shows the official evaluation results of our submitted rankings compared to the best submission from Microsoft. While slightly exceeding on the SMT 10M evaluation, we are 0.8% BLEU behind the leading submission on NMT 100M. For NMT 10M, we have the best results on newstest2018, iwslt2017 and Acquis, but perform a lot weaker on KDE, thus being worse on average. This might be due to some unavoidable domain adaptation when training language models with mono-lingual news data.

7 Conclusion

This paper describes the RWTH Aachen University data-filtering and ranking methods for the WMT 2018 parallel corpus filtering task. We describe various rule-based heuristic filtering methods to reduce the amount of data to be scored, and to tackle some of the weak spots of neural language and translation models. We describe 4 different ranking models, two language model architectures and 2 translation models, count-based and neural. Our results indicate that even without ranking the sentence pairs with model scores, a high quality subset can be extracted.

Among the submissions our best models works very well for the small data condition, ranking first on the 10M-subsampled SMT translation and second on the 10M-subsampled NMT translation. Also with the 100M-subsampled data condition, we perform above average, with a gap of 0.7% average BLEU to the leading submission for NMT translation.

Acknowledgements

This work has received funding from the European Research Council (ERC) (under the European Union's Horizon 2020 research and innovation programme, grant agreement No 694537, project "SEQCLAS") and the Deutsche Forschungsgemeinschaft (DFG; grant agreement NE 572/8-1, project "CoreTec"). The GPU computing cluster was supported by DFG (Deutsche Forschungsgemeinschaft) under grant INST 222/1168-1 FUGG.
The work reflects only the authors' views and

System (10M)	newstest15		newstest17		newstest18	
	BLEU	TER	BLEU	TER	BLEU	TER
(3.1-5) + KenLM + IBM	26.8	53.9	27.0	54.3	32.5	46.9
(3.1-5) + KenLM + IBM + Lev.Sht.	26.8	53.8	26.9	54.3	33.0	46.7
(3.1-5) + BlocksLM + Transformer	27.3	53.2	27.4	53.5	33.6	46.1
(3.1-5) + BlocksLM + Transformer + Lev.Sht.	28.1	52.4	28.4	52.4	34.6	45.0

Table 6: Effect of using the Levenshtein distance heuristic (3.6) on count-based and neural scoring.

Submission System	SMT 10M	SMT 100M	NMT 10M	NMT 100M
(3.1)-(3.7) + KenLM + retrained IBM1[1]	23.85	25.91	26.65	31.05
(3.1)-(3.7) + BlocksLM + Transfomer[2]	24.53	26.18	28.00	31.20
+ Redundancy Heuristic[3]	24.58	26.21	28.01	31.29
Microsoft	24.45	26.50	28.62	32.06

Table 7: Official submission result for each evaluation method. The scores report the average BLEU % across all 6 test sets.
[1]: Submission 1 with name **rwth-count**
[2]: Submission 2 with name **rwth-nn**
[3]: Submission 3 with name **rwth-nn-redundant**

none of the funding agencies is responsible for any use that may be made of the information it contains.

References

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311.

Stanley F. Chen and Joshua Goodman. 1996. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, ACL '96, pages 310–318, Stroudsburg, PA, USA. Association for Computational Linguistics.

Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*. Version 1.

Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. 2013. Maxout networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1319–III–1327. JMLR.org.

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 Association for Computational Linguistics*, pages 690–696, Sofia, Bulgaria.

Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton, and Matt Post. 2017. Sockeye: A Toolkit for Neural Machine Translation. *arXiv preprint arXiv:1712.05690*. Version 2.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. 2018. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia. Association for Computational Linguistics.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. Version 9.

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 Constantine, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *acl07-poster*, pages 177–180, Prague, Czech Republic.

Vladimir Iosifovich Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

Bart van Merriënboer, Dzmitry Bahdanau, Vincent Dumoulin, Dmitriy Serdyuk, David Warde-Farley, Jan Chorowski, and Yoshua Bengio. 2015. Blocks and Fuel: Frameworks for deep learning. *CoRR*, abs/1506.00619.

Franz Josef Och and Hermann Ney. 2003. A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics*, 29(1):19–51.

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.

Jan-Thorsten Peter, Eugen Beck, and Hermann Ney. 2018. Sisyphus, a workflow manager designed for machine translation and automatic speech recognition. In *2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium. (To appear).

Jan-Thorsten Peter, Andreas Guta, Tamer Alkhouli, Parnia Bahar, Jan Rosendahl, Nick Rossenbach, Miguel Graça, and Ney Hermann. 2017. The RWTH Aachen University English-German and German-English machine translation system for WMT 2017. In *EMNLP 2017 Second Conference on Machine Translation*, Copenhagen, Denmark.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1715–1725.

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.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688. Version 1.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

David Vilar, Daniel Stein, Matthias Huck, and Hermann Ney. 2010. Jane: Open source hierarchical translation, extended with reordering and lexicon models. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, pages 262–270. Association for Computational Linguistics.