Neural Machine Translation with attention

Fabio Curi and Mirthe van Diepen
fabio.curi.paixao@student.uva.nl, mmvandiepen@gmail.com

Abstract

This project applies Neural Machine Translation in a parallel corpora. The model makes use of a sequence-to-sequence positional embedding with different encoders and decoders, namely: linear, Long Short-Term Memory and Gated Recurrent Unit. An attention mechanism was implemented in the decoder through dot and bilinear product in order to focus on specific areas of the sentences. Finally, the output sentences from the model were evaluated through four different scores.

1 Introduction

Neural Machine Translation (NMT) has been applied to perform sequence-to-sequence modeling and translation of bilingual corpus. Applying Recurrent Neural Networks (RNN) to encode and decode the source and target languages, respectively, have been implemented alongside new mechanisms to optimize the training of the model. As presented by Neubig (2017), attention can be applied in order to express input sentences in a much more efficient way, avoiding the problems of inefficient representations for simple encoder-decoders. In this paper the translations are built from French to English.

This paper is divided as follows: first, the general model is explained, as well as the procedures done for encoding and decoding the sentences. A short introduction to the attention mechanisms applied is carried out and the implementation of GRU and LSTM encoders is shortly explained. Then, the evaluation metrics are introduced, and the experiments' setup is detailed. Following, a quick explanation on the preprocessing of the used data is done. At last, we present the results and the conclusion.

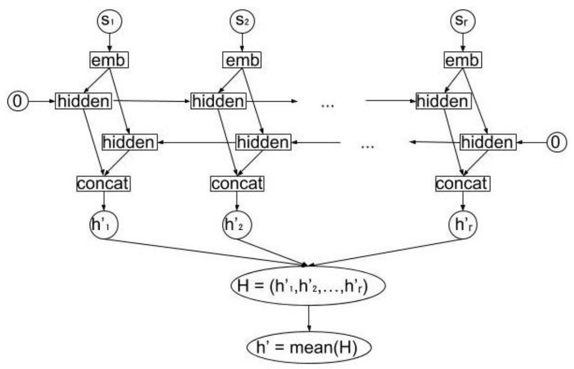

Figure 1: The process of encoding the source sentence.

2 Model

The overall NMT model built for this project follows the general scope introduced by Neubig (2017). This model contains an encoder and a decoder which will be trained jointly. Online-learning is used to train the model.

2.1 Encoder

The encoding process is visualized in Figure 1. For all words s_j ($j = 1, \dots, r$) of the source sentence, the embedding is computed. The embedding goes through hidden layers in two different directions. The hidden layer can be linear, Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU). This results into two hidden outputs, which are concatenated, resulting in a bidirectional representation of the word. Placing these vectors in a matrix, we obtain the bidirectional matrix representation H of the sentence which is the output of the encoder. Also, the mean $\boldsymbol{h}'$ of all these vectors has been computed since the mean is required for the decoder.

2.2 Decoder with attention

The decoding process is recursive and is shown in Figure 2. The contexts vector $\boldsymbol{c}_i$ and target word

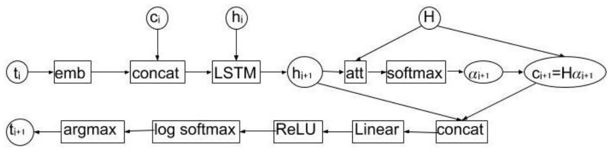

Figure 2: The process of decoding the encoded source sentence.

t_i are initialized as follows:

$$\boldsymbol{c}_0 := 0, \qquad t_0 := \text{‘SOS’},$$

respectively, where ‘SOS’ stands for ‘start of sentence’. The hidden layer for the LSTM layer is initialized by $\boldsymbol{h}_0 := \boldsymbol{h}'$. In this process, first the embedding of t_0 is computed. To avoid overfitting, the embeddings have been regularized using dropout with values 0.1, 0.2, 0.3 and 0.4 (these values are inspired from Pereyra et al. (2017)). Thereafter, the embedding and current context vector are concatenated and used as input for the LSTM layer. The LSTM then outputs the next hidden layer $\boldsymbol{h}_{i+1}$. Using this hidden layer and H, the attention score can be computed. This last has been implemented using the dot product or bilinear function:

$$\boldsymbol{a}_{i+1} = \begin{cases} H^T \boldsymbol{h}_{i+1} & \text{if dot product;} \\ H^T W_a \boldsymbol{h}_{i+1} & \text{if bilinear function,} \end{cases}$$

where W_a is the attention matrix. According to Neubig (2017) the attention score is used to weight the bidirectional representation matrix H. Also, the bilinear function has as advantage that H and $\boldsymbol{h}_i$ does not necessarily need to have the same sizes. Now, the softmax over the attention score is computed to normalize the output, and we call the outcome $\boldsymbol{\alpha}_{i+1}$. Thereafter, the next context vector can be computed as follows

$$\boldsymbol{c}_{i+1} = H^T \boldsymbol{\alpha}_{i+1}.$$

The next hidden vector and the context vector are concatenated and used as input for a linear layer, which maps the vector into an output vector of the size of the target vocabulary. A log-softmax over the non-negative values pushes the values into a distribution. This last can be seen as a distribution over the target vocabulary: the index (corresponding to a word) with the highest value is the most probable translation. Hence, the argmax has been used to compute the next word.

For training the model, the target sentence has been used to push the model into the right direction. Since this is an optional argument in the implementation, the model can generate a new sentence given a source sentence. In the case that the model does not know the target length, the model will stop generating words as soon as the word ‘EOS’ (end of sentence) string has been generated.

3 GRU vs LSTM

An extra research question here suggested is to compare the to-be-computed metrics after the implementation of GRU and LSTM encoders in the NMT system. Enough information about their application is present in the literature. The work done by Irie et al. (2016) mentions that “LSTM seems to be a better default choice for language modeling than the GRU”. On the other hand, Khandelwal et al. (2017) suggests that there is no clear difference in encoding the data with either of the methods. We will investigate through the implementation of RNN with fine-tuned drop-out values which method renders the highest scores in the four metrics to be used (see Section 4).

4 Evaluation

Four different scores were used to evaluate the outputs of the model: BLEU1, BLEU4, Meteor and TER. All of them were measured comparing the outputted translated words to the golden standards. The two first scores were calculated through the work done by Moses, while the two last were measured thanks to the works done by Denkowski and Lavie (2014) and Snover et al. (2006), respectively.

5 Experiments’ setup

All experiments were run using the Torcs library on the CPU. Three different main models were run: RNN with linear, GRU and LSTM encoders of the hidden layers. For each one of these models, attention is implemented with either the dot product, or with the bilinear function. For the linear experiments, a fine-tuning of the drop-out rate was done for values ranging between 0.1 and 0.4. The original idea was to decide on the number of epochs based on the evolution of the metrics for the train and validation sets, however it was made impossible due to insufficient RAM memory in the processors. Thus, a fixed number of 5 epochs was set for each model. The learning rate was set to 10^{-2} for all models.

5.1 Data

Three sets of data were provided: train, validation and test. The model was trained on the train data and the fine-tuning of the drop-out value of the embedded words is done analyzing the scores of the validation set.

5.2 Preprocessing

The MOSES algorithm was used to preprocess the data through two steps, namely tokenization and lower-casing (see Moses for code). Then, a byte-pair encoding (BPE) was also applied to detect patterns in words and segment text into sub-word units (see Sennrich for code). This last is a simple universal text compression scheme based on pattern-substitution and also copes with the issue of having unknown and infrequent words in the corpus.

The first step was to build a list of codes that will split strings into sub-strings. This was done for both parallel corpora simultaneously, so patterns in both languages are taken into account together. Different numbers of operations were tested to calculate the BPE of the provided data: 1.000, 10.000, 20.000 and 30.000. For a number of 1.000, there was a considerably higher number of word segmentation. For higher number of operations, the outputs seem more coherent with what it expected from this task: the algorithm took longer to learn what patterns should be split, and what segmentation should take place. For instance, words such as 'uniformly' became 'uni@@ form@@ ly', which is an eminent step in segmenting words such as adverbs into separate elements as prefixes and suffixes. No major differences were seen for higher numbers than 20.000, thus this number was kept as the number of iterations to perform BPE for computation cost purposes. After these three preprocessing steps, there was a considerable reduction of the vocabulary size in both corpora. Overall, there were sharply 4.9% more French words in the vocabulary than English ones. Finally, the word embedding of the source and target-language was done in a way that all sentences were converted to tensors, with each word being represented by its index in its respective language vocabulary. We set the word 'EOS' with index of 1.

6 Results

The present section presents the results for all models tested. It is important to know that all scores were calculated after the merging of the sub-words generated from the BPE step in the preprocessing. These scores were measured for the validation data, and parameter choices are made so as to build a proper configuration for all models.

6.1 Linear

The first experiment was to explore how different drop-out values affected the scores, for a linear encoding of the hidden layers. This is an important step of building the algorithms, given that a fixed value for the drop-out is needed for whichever upcoming model to be tested afterwards. The scores for the linear encoding with dot product are shown in Table 1.

	0.1	**0.2**	**0.3**	**0.4**
BLEU1	31.20	**38.70**	36.50	33.50
BLEU4	1.30	**1.70**	0.80	0.80
Meteor	0.12	**0.13**	0.10	0.09
TER	1.02	0.82	**0.81**	0.84

Table 1: Metrics for the linear encoder with dot product and 5 epochs. Different drop-out values are tested.

The values in Table 1 suggest that a drop-out value of 0.2 renders the highest three first scores. Its TER value is rather close to the one obtained by a drop-out value of 0.3, which is the minimal one. Note that the lower the TER values, the less work a translator should do to a perfect translation of the source sentence. A finer-grained choice for drop-out values could have been explored to see what happens for drop-out values ranging between 0.1 and 0.3, however this would be computationally too expensive to explore. Finally, all the following models carry a drop-out of 0.2, which is analogical to the choice made by Nguyen and Chiang (2017).

Now, a different attention mechanism is implemented using the bilinear function. The results are shown in Table 2. There was a considerable decrease in all scores using the bilinear function. Perhaps this is due to the vulnerability of the results in respect to the choice of the values in the W_a matrix, which affect the way the context vector is computed. A possible future analysis could be analyzing how the elements in this matrix affect the metrics.

BLEU1	29.20
BLEU4	0.70
Meteor	0.09
TER	0.95

Table 2: Metrics for the linear encoder with bilinear attention, drop-out of 0.2 and 5 epochs.

6.2 GRU vs LSTM

The same experiments are run, but now with two different encoders of the hidden layers: LSTM and GRU. The interest is to know which one of these models performs best, and how different epochs affect the scores. In Table 3, the scores for both encoders are measured using the dot product and the bilinear function.

	GRU	**GRU$_{bi}$**	**LSTM**	**LSTM$_{bi}$**
BLEU1	54.50	**57.10**	58.60	**59.60**
BLEU4	9.60	**11.50**	11.20	**11.40**
Meteor	0.23	**0.26**	**0.24**	**0.24**
TER	0.62	**0.59**	**0.56**	**0.56**

Table 3: Metrics for the GRU and LSTM encoders with both dot product and bilinear attention, drop-out of 0.2 and 5 epochs.

The best scores for GRU are shown in blue, while the best ones for LSTM are in red. The bilinear function renders the best scores for both encoders. It appears that the GRU encoder is more sensitive to the choice of applying dot product or using the bilinear function, while LSTM scores do not seem to differ as much regarding the choice of either attention mechanisms. Finally, all these models were run for only 5 epochs, given that the average time per epoch was around 20 minutes. Figure 3 shows the loss for each one of these models. Cross-entropy was used as the technique to calculate the losses, and the average value for each epoch is calculated. From the results in Table 3 and Figure 3, it is not safe to state that either one of the two encoders perform better than the other, after comparison of the blue and red scores and the loss. During the training time, it is expected that the loss decreases with time, which is observed for all the models. In general, the bilinear attention seems to converge to lower values in comparison to the dot product; however, this is not true for the linear models. We reason that this model might be much more sensitive to the values present in the W_a matrix, given that the encoder is linear.

Finally, we run the bilinear models for 10 epochs to see if more sensitive results emerge for GRU and LSTM, see Table 4.

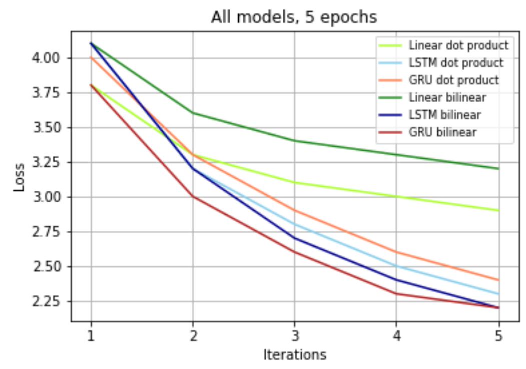

Figure 3: The average loss per epoch for all models run with 5 epochs.

	GRU$_{bi}$	**LSTM$_{bi}$**
BLEU1	60.6	**62.60**
BLEU4	11.6	**16.20**
Meteor	0.26	**0.29**
TER	0.60	**0.52**

Table 4: Metrics for the GRU and LSTM encoders with bilinear attention, drop-out of 0.2 and 10 epochs.

It appears that introducing 5 more epochs in the training is an eminent step towards obtaining better scores. The values in bold show that LSTM clearly outperforms GRU in all metrics, which makes our results to be in line what was discovered by Irie et al. (2016).

7 Conclusion

This study explored different ways of implementing a NMT model with different encoders of the hidden layers. A fine-tuning of the drop-out value was initially done, and the metrics were compared for all three different encoders and two attention mechanisms. Regardless of these last, GRU and LSTM clearly outperform linear encoding, and the implementation of the bilinear function to calculate the attention considerably improved the scores for the former two models. Results suggest that for enough epochs, LSTM seems to outperform GRU when the bilinear function is applied. Thus, a final model with LSTM encoding of the hidden layers alongside the application of such a function would be applied in the test set, rendering the best English translated sentences. Metrics for the test set were not measured as it turned out to be computationally expensive, thus we rely on the validation results for the choice of the best model found.

References

Michael Denkowski and Alon Lavie. 2014. Meteor Universal: Language Specific Translation Evaluation for Any Target Language. In *Proceedings of the EACL 2014 Workshop on Statistical Machine Translation*.

Kazuki Irie, Zoltán Tüske, Tamer Alkhouli, Ralf Schlüter, and Hermann Ney. 2016. LSTM, GRU, Highway and a Bit of Attention: An Empirical Overview for Language Modeling in Speech Recognition. In *INTERSPEECH*.

Shubham Khandelwal, Benjamin Lecouteux, and Laurent Besacier. 2017. Comparing GRU and LSTM for Automatic Speech Recognition .

Graham Neubig. 2017. Neural Machine Translation and Sequence-to-sequence Models: A Tutorial .

Toan Nguyen and David Chiang. 2017. Transfer Learning across Low-Resource, Related Languages for Neural Machine Translation .

Gabriel Pereyra, George Tucker, Jan Chorowski, Lukasz Kaiser, and Geoffrey Hinton. 2017. Regularizing Neural Networks by Penalizing Confident Output Distributions .

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A Study of Translation Edit Rate with Targeted Human Annotation .

Appendices

A Code

Please find the code at: Github.