Ono Gantsog, Anton Rozhkov
4/23/19

FINAL PROJECT

Sequence to Sequence Model for Translation

Introduction - Problem statement

Today, sequence-to-sequence (seq2seq) models are becoming very widespread and provide very high quality and almost excellent performance in a wide variety of tasks. The main structure for all these models is usually a deep neural network comprising an encoder and a decoder. In another words, we can say that seq2seq models constitute a common framework for solving sequential problems, like Google Translate, online chatbots, text summarization, speech to text conversion and speech recognition in voice-enabled devices.

During this semester in one of the assignments we were asked to create a seq2seq model for translation in general and specifically to train two models: first, the seq2seq model for a language pair where the output is English and the input is a language of our choice (we used Russian language as an input, because there are enough data for it and both of us is able to evaluate the quality of translation), second model was for the reverse pair (i.e., from English to Russian). In this model, we uses the GloVe 100 dimensional embedding while training, this pretrained data were able to increase the accuracy of the model.

Using Tab-delimited Bilingual Sentence Pairs data (https://www.manythings.org/anki/) , we found that there are more than 70 language pairs available, where some of them with the most comprehensive vocabulary, somewhere with somewhat level of details, but also we found that there are no English-Mongolian pair in this dictionary. We recognize that on the general website of the project (https://tatoeba.org/) where there are more than 330 different language pair, including Mongolian, but the data for this language is too small - around 550 sentences and the performance of the translation is so bad. Since that time we decided to focus on this language pair and work on seq2seq model.

Language

Before going to the model we firstly went deeper to the field of linguistics. In terms of linguistic typology and classification of languages according to their structural and functional features, languages are divided into two main parts. Languages may be classified according to the dominant sequence of these elements in unmarked sentences [6].

SVO = Subject-Verb-Object type of languages are languages like English, Russian, French, etc. Sentence starts with subject, followed by a verb and details in the object part. For example: “I go to school”. SVO languages almost always place relative clauses after the nouns which they modify and adverbial subordinators before the clause modified.

SOV = Subject-Object-Verb type of languages are languages like Mongolian, Turkish, Japanese, etc. Sentence starts with subject, followed by details in object part and verb comes at the end. For example: “I school to go”.

Therefore, as far as we did not have training data for Mongolian translation, we wrote the translation network code using Turkish training data thinking it might help with the overall network structure. Another interesting fact is that Russian and Mongolian languages use the same alphabet - Cyrillic and Turkish uses Latin alphabet like in English language.

Data collection

Since we could not find training data for English to Mongolian translation, we asked friends to translate the sentences. We got sentences from Russian training data of Tatoeba Project and we selected the sentences with eight or less words. From those 16,000 sentences, we tried to translate longer sentences (more than 4 words) for training purposes.

Finally, we got about 1,700 sentences translated. We checked each of them for a proper and correct translation. From that point we are able to move forward to the model.

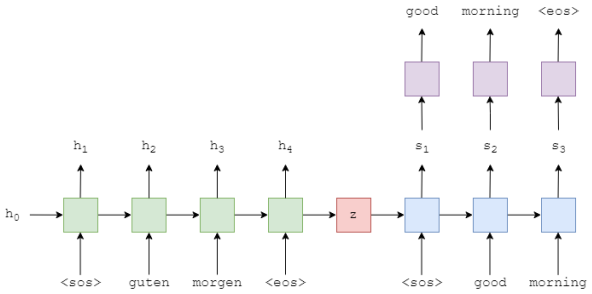
Neural network structure

From a probabilistic perspective, translation is equivalent to finding a target sentence **y** that maximizes the conditional probability of **y** given a source sentence **x**, i.e., arg max**y** p(**y|x**). In neural machine translation, we fit a parameterized model to maximize the conditional probability of sentence pairs using a parallel training corpus. Once the conditional distribution is learned by a translation model, given a source sentence a corresponding translation can be generated by searching for the sentence that maximizes the conditional probability (Bahdanau, et al., 2014).

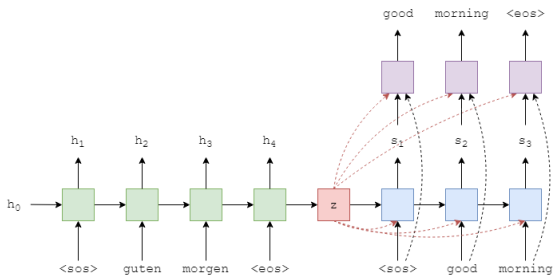
This neural machine translation approach typically consists of two components, the first of which **encodes** a source sentence x and the second **decodes** to a target sentence y. For instance, two recurrent neural networks (RNN) were used by Sutskever et al. (2014) to encode a variable-length source sentence into a fixed-length vector and to decode the vector into a variable-length target sentence.

Despite being a quite new approach, neural machine translation has already shown promising results. Sutskever et al. (2014) reported that the neural machine translation based on RNNs with long short-term memory (LSTM) units achieves close to the state-of-the-art performance of the conventional phrase-based machine translation system on an English-to-French translation task. Adding neural components to existing translation systems, for instance, to score the phrase pairs in the phrase table or to re-rank candidate translations (Sutskever et al., 2014), has allowed to surpass the previous state-of-the-art performance level.

In the general encoder-decoder model (Pic. 1), an encored reads the input sentence, a sequence of vectors **x** = (x1,..., xT) into a vector **c.** The decoder is often trained to predict the next word, given the context vector c and all the previously predicted words. In other words, the decoder defines a probability over the translation **y** by decomposing the joint probability into the ordered conditionals.



Pic. 1 - General encoder-decoder model



Pic. 2 - Encoder-decoder model, based on the [4] study

Based on the some research in the topic already done [4], the architecture of the model was set-up in a way to reduce "information compression" by explicitly passing the context vector, **z**, to the decoder at every time-step and by passing both the context vector and input word, **y**t, along with the hidden state, **s**t, to the linear layer, **f**, to make a prediction. Even though they have reduced some of this compression, our context vector still needs to contain all of the information about the source sentence.

In this study we are using attention to avoid this compression by allowing the decoder to look at the entire source sentence (via its hidden states) at each decoding step (Pic. 2). Attention works by first, calculating an attention vector, **a**, that is the length of the source sentence. The attention vector has the property that each element is between 0 and 1, and the entire vector sums to 1. We then calculate a weighted sum of our source sentence hidden states, **H**, to get a weighted source vector, **w** [4].

First step is building the **encoder**. Similarly to the general model, we use a single layer GRU, but also we are using a bidirectional RNN. It connects two hidden layers of opposite directions to the same output. With this form of generative deep learning, the output layer can get information from past (backwards) and future (forward) states simultaneously. Simply saying, it's just putting two independent RNNs together, so we have two RNNs in each layer. A forward RNN going over the sentence from left to right, and a backward RNN going over the sentence from right to left. All we need to do in code is set *bidirectional* = *True* and then pass the embedded sentence to the RNN as before.

Next is the **attention** layer. The layer will output an attention vector, **a**t, that is the length of the source sentence, each element is between 0 and 1 and the entire vector sums to 1. Also, here we calculated the *energy* between the previous decoder hidden state and the encoder hidden states by concatenating them together and passing them through a linear layer (*attn*) and a *tanh* activation function. Finally, $E_t = \tanh(\text{attn}(s_{t-1}, H))$ we ensure the attention vector fits the constraints of having all elements between 0 and 1 and the vector summing to 1 by passing it through a *softmax* layer (not log_softmax as it was in a general model).

The **decoder** contains the attention layer, which takes the previous hidden state, all of the encoder hidden states, and returns the attention vector, **a**t**.** One of the differences between the current model and the general one is that in this model we are not using softmax in the decoder part.

Project

For the task of Neural Machine Translation on seq2seq model we used python, PyTorch library and for GPU we used Google Colab notebook. Google Collaboratory has interface like Jupyter notebook and works in Google drive. We used our school account to get access. Till getting the training data, we experimented with our last homework seq2seq model and turkish training data. We picked the architecture because it was mentioned in the paper (Bahdanau, et al., 2016) that for longer sentences it was performing well. In order to translate from SVO language to SOV language, the model needs to carry information until the end of the sentence.

Result

In the table below, we see the performance on different data. Performance is improved when hidden dimension is increased. Performance is decreasing when dropout is increased. Although not shown in the table, increasing Embedding size also improves the performance. Training time on CPU is about 14 min and 25 sec on GPU.

Training data	Num of sente	Hidden dim	Dropout	Training loss	Val loss
Mongolian - cirillyc	1040	128	0.2	4.4-1.4	4.1-1.6
Mongolian - cirillyc	1040	256	0.3	2.5 - 0.1	1.4-0.2
Mongolian - latin	400	256	0.5	1.8-0.1	0.3-0.1
Turkish	16000	128	0.1	5.5-4.6	
Turkish	16000	256	0.1	5.4-2.3	
Turkish	16000	256	0.1	5.4-3.8	
Turkish	16000	256	0.2	5.4-3.9	

```
> he s very interested in japanese .
= тэр япон хэлэнд маш сонирхолтой .
< япон маш их сонирхолтой . EOS
```

The translation above is interesting. It captured the meaning of words japan, very and interesting, but totally ignored he and language part.

Our best model has following configuration.

ENC_EMB_DIM = 128

DEC_EMB_DIM = 128

ENC_HID_DIM = 512

DEC_HID_DIM = 512

ENC_DROPOUT = 0.2

DEC_DROPOUT = 0.2

Challenges and future work

- **Cyrillic alphabet**

We prepared all our data in spreadsheet file and finally when we combine all the translated sentences from friends into one csv, all the letters became unrecognizable. We tried with tsv format files. Finally, we kept the file as xlsx and imported data from there. Also, some

letters specific to mongolian was not recognized. When we added those in the regular expression, those letters were fixed.

- **Training data**

In the Tatoeba project, there are 400,000 pair of sentences in french-english training data. But we could only translate 1,700 pair of sentences in a month. The selected sentences for translation have mostly 4-8 words in english. For us, the hardest part of the project was to ask friends to translate the sentences and fix the translations if they missed certain phrases and converting to cyrillic.

- **More data (for sure)**

To improve translation model we definitely need to have more data. We think, translation model performance will be reasonable at about 10,000 pairs of sentence.

- **Word embeddings**

We could implement mongolian word embedding at first and then use it in the translation model. Like in english, GloVe has 100 dimension embeddings for each word. Unfortunately there is no ready data for mongolian language in popular datasets like spaCy and GloVe.

- **Transfer learning**

We found a paper “Transfer Learning for Low-Resource Neural Machine Translation.” (Zoph, et al., 2016). Like in case with images, neural machine translation can also use transfer learning. In the paper, it mentions that 5.6 BLEU on average in row-resource languages.

References

1. Sutskever I., Vinyals O., and Q. V. Le, “Sequence to sequence learning with neural networks,” in NeurIPS, 2014, pp. 3104–3112.
2. Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate”, 2014, arXiv:1409.0473, 2014
3. Barret Zoph, Deniz Yuret, Jonathan May, Kevin Knight, “Transfer Learning for Resource Neural Machine Translation”, 2016, https://arxiv.org/pdf/1604.02201v1.pdf
4. Tatoeba Collection, https://tatoeba.org
5. “Neural Machine Translation by Jointly Learning to Align and Translate”, https://github.com/bentrevett/pytorch-seq2seq/blob/master/3%20-%20Neural%20Machine%20Translation%20by%20Jointly%20Learning%20to%20Align%20and%20Translate.ipynb
6. https://en.wikipedia.org/wiki/Subject–verb–object
7. https://deepai.org/machine-learning-glossary-and-terms/bidirectional-recurrent-neural-networks
8. PyTorch, https://pytorch.org/