Practical 3: Neural Machine Translation

Deep Natural Language Processing Class, 2022

Report due date - 11.04.2023

This practical was greatly inspired by the CS224 class from Stanford University. The practical report (pdf + code) should be sent via Moodle/Pegaz with a standard due dates policy.

1 Neural Machine Translation with Attention

In Machine Translation, our goal is to convert a sentence from the source language (e.g. English) to the target language (e.g. Polish). In this assignment, we will implement a sequence-to-sequence (Seq2Seq) network with attention, to build a Neural Machine Translation (NMT) system. In this section, we describe the training procedure for the proposed NMT system, which uses a Bidirectional LSTM Encoder and a Unidirectional LSTM Decoder. If you have troubles remembering the mechanism behind LSTM, have a look at this handy visual introduction here.

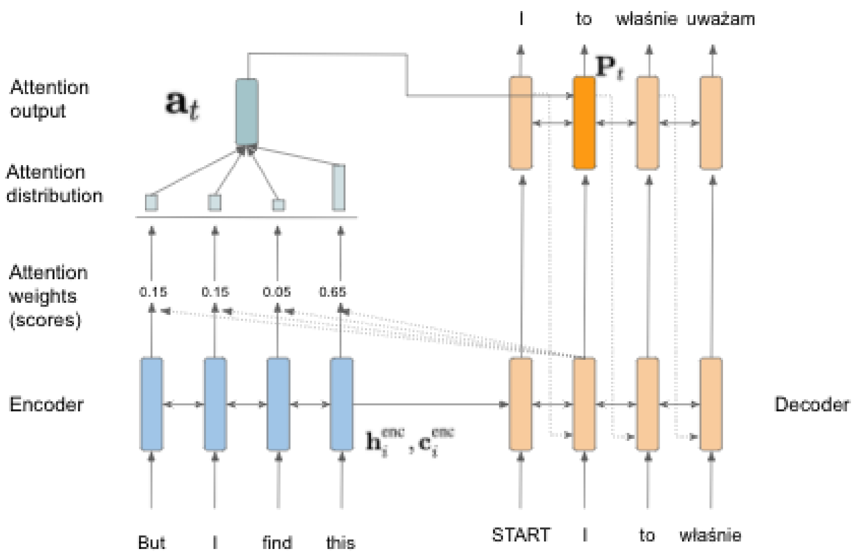

Figure 1: The example of the NMT model we are building in this practical.

Given a sentence in the source language, we look up the word embeddings from an embeddings matrix, yielding $\boldsymbol{x}_1, ..., \boldsymbol{x}_m$, $\boldsymbol{x}_i \in \mathbb{R}^{e\times 1}$, where m is the length of the source sentence and e is the embedding size. We feed these embeddings to the bidirectional Encoder, yielding hidden states and cell states for both the forwards and backwards LSTMs. The forwards and backwards versions are concatenated to give hidden states $\boldsymbol{h}_i^{\text{enc}}$ and cell states $\boldsymbol{c}_i^{\text{enc}}$:

$$\begin{aligned}
\boldsymbol{h}_i^{enc} &= [\overleftarrow{\boldsymbol{h}_i^{enc}}; \overrightarrow{\boldsymbol{h}_i^{enc}}], \text{ where } \boldsymbol{h}_i^{enc} \in \mathbb{R}^{2h\times 1}, \overleftarrow{\boldsymbol{h}_i^{enc}}, \overrightarrow{\boldsymbol{h}_i^{enc}} \in \mathbb{R}^{h\times 1}, 1 \le i \le m, \\
\boldsymbol{c}_i^{enc} &= [\overleftarrow{\boldsymbol{c}_i^{enc}}; \overrightarrow{\boldsymbol{c}_i^{enc}}], \text{ where } \boldsymbol{c}_i^{enc} \in \mathbb{R}^{2h\times 1}, \overleftarrow{\boldsymbol{c}_i^{enc}}, \overrightarrow{\boldsymbol{c}_i^{enc}} \in \mathbb{R}^{h\times 1}, 1 \le i \le m.
\end{aligned}$$

We then initialize the Decoder's first hidden state $\boldsymbol{h}_0^{enc}$ and cell state $\boldsymbol{c}_0^{enc}$ with a linear projection of the Encoder's final hidden state and final cell state:

$$\begin{aligned}
\boldsymbol{h}_0^{dec} &= \boldsymbol{W}_h[\overleftarrow{\boldsymbol{h}_1^{enc}}; \overrightarrow{\boldsymbol{h}_m^{enc}}], \text{ where } \boldsymbol{h}_0^{dec} \in \mathbb{R}^{h\times 1}, \boldsymbol{W}_h \in \mathbb{R}^{h\times 2h}, \\
\boldsymbol{c}_0^{dec} &= \boldsymbol{W}_c[\overleftarrow{\boldsymbol{c}_1^{enc}}; \overrightarrow{\boldsymbol{c}_m^{enc}}], \text{ where } \boldsymbol{c}_0^{dec} \in \mathbb{R}^{h\times 1}, \boldsymbol{W}_c \in \mathbb{R}^{h\times 2h}.
\end{aligned}$$

With the Decoder initialized, we must now feed it a matching sentence in the target language. On the t-th step, we look up the embedding for the t-th word, $\mathbf{y}_t \in \mathbb{R}^{e\times 1}$. We then concatenate $\mathbf{y}_t$ with the combined-output vector $\mathbf{o}_{t-1} \in \mathbb{R}^{h\times 1}$ from the previous timestep (we will explain what this is later down this page!) to produce $\hat{\mathbf{y}}_t \in \mathbb{R}^{(e+h)\times 1}$. Note that for the first target word (i.e. the start token) $\mathbf{o}_0$ is a zero-vector. We then feed $\hat{\mathbf{y}}_t$ as input to the Decoder LSTM.

We then use $\boldsymbol{h}_t^{dec}$ to compute multiplicative attention over $\boldsymbol{h}_1^{enc}, ..., \boldsymbol{h}_m^{enc}$:

$$\begin{aligned}
\boldsymbol{e}_{t,i} &= (\boldsymbol{h}_t^{dec})^T \mathbf{W}_{attProj} \boldsymbol{h}_i^{enc}, \quad \text{where } \boldsymbol{e}_t \in \mathbb{R}^{m\times 1}, \mathbf{W}_{attProj} \in \mathbb{R}^{h\times 2h} \\
\alpha_t &= \text{softmax}(\boldsymbol{e}_t), \quad \text{where } \alpha_t \in \mathbb{R}^{m\times 1}, \\
\boldsymbol{a}_t &= \sum_{i=1}^{m} \alpha_{t,1} \boldsymbol{h}_i^{enc}, \quad \text{where } \boldsymbol{a}_t \in \mathbb{R}^{2h\times 1},
\end{aligned}$$

for $i \in \{1, ..., m\}$. We now concatenate the attention output $\mathbf{a}_t$ with the decoder hidden state $\mathbf{h}_t^{dec}$ and pass this through a linear layer, tanh, and dropout to attain the combined-output vector $\boldsymbol{o}_t$.

$$\begin{aligned}
\boldsymbol{u}_t &= [\boldsymbol{a}_t; \boldsymbol{h}_t^{dec}], \text{where } \boldsymbol{u}_t \in \mathbb{R}^{3h\times 1} \\
\boldsymbol{v}_t &= \boldsymbol{W}_u \boldsymbol{u}_t, \text{where } \boldsymbol{v}_t \in \mathbb{R}^{h\times 1}, \boldsymbol{W}_u \in \mathbb{R}^{h\times 3h} \\
\boldsymbol{o}_t &= \text{dropout}(\tanh(\boldsymbol{v}_t)), \text{where } \boldsymbol{o}_t \in \mathbb{R}^{h\times 1}
\end{aligned}$$

Then, we produce a probability distribution $\mathbf{P}_t$ over target words at the t-th timestep:

$$\mathbf{P}_t = \text{softmax}(\mathbf{W}_{\text{vocab}} \mathbf{o}_t),$$

where $\mathbf{P}_t \in \mathbb{R}^{V_t \times 1}, \mathbf{W}_{\text{vocab}} \in \mathbb{R}^{V_t \times h}$. Here, V_t is the size of the target vocabulary. Finally, to train the network we then compute the softmax cross entropy loss between $\mathbf{P}_t$ and $\mathbf{g}_t$, where $\mathbf{g}_t$ is the 1-hot vector of the target word at timestep t:

$$J_t(\theta) = CE(\mathbf{P}_t, \mathbf{g}_t).$$

Here, θ represents all the parameters of the model and $J_t(\theta)$ is the loss on step t of the decoder.

Now let's try to implement the above model for English to Polish translation! We will be using the data provided for the PolEval 2019 Task 4 - Machine Translation. Using any technology, with limited textual resources you are asked to train the best possible machine translation system. The competition was run for 2 language pairs, more popular English-Polish (into Polish direction) and pair that can be called low resourced Russian-Polish (in both directions). We will focus only on the former pair. Look at the data in the `en_pl_data` folder.

a) (1.5 point)

In order to apply tensor operations, we must ensure that the sentences in a given batch are of the same length. Thus, we must identify the longest sentence in a batch and pad others to be the same length. Implement the `pad_sents` function in `utils.py`, which shall produce these padded sentences.

b) (2 points)

Implement the `__init__` function in `model_embeddings.py` to initialize the necessary source and target embeddings.

c) (3 points)

Implement the `__init__` function in `nmt_model.py` to initialize the necessary model embeddings (using the ModelEmbeddings class from `model_embeddings.py`) and layers (LSTM, projection, and dropout) for the NMT system.

d) (5 points)

Implement the encode function in `nmt_model.py`. This function converts the padded source sentences into the tensor $\mathbf{X}$, generates $\boldsymbol{h}_1^{enc}, ..., \boldsymbol{h}_m^{enc}$, and computes the initial state $\boldsymbol{h}_0^{dec}$ and $\boldsymbol{c}_0^{dec}$ for the Decoder. You can run a non-comprehensive sanity check by executing: `python sanity_check.py 1d`

e) (6 points)

Implement the decode function in `nmt_model.py`. This function constructs $\overline{\boldsymbol{y}}$ and runs the `step` function over every timestep for the input. You can run a non-comprehensive sanity check by executing: `python sanity_check.py 1e`

f) (7 points)

Implement the `step` function in `nmt_model.py`. This function applies the Decoder's LSTM cell for a single timestep, computing the encoding of the target word $\boldsymbol{h}_t^{\text{dec}}$, the attention scores $\boldsymbol{e}_t$, attention distribution α_t, the attention output $\boldsymbol{a}_t$, and finally the combined output $\boldsymbol{o}_t$. You can run a non-comprehensive sanity check by executing: `python sanity_check.py 1f`

g) (2 points)

The generate `sent_masks` function in `nmt_model.py` produces a tensor called `enc_masks`. It has shape (batch size, max source sentence length) and contains 1s in positions corresponding to 'pad' tokens in the input, and 0s for non-pad tokens. Look at how the masks are used during the attention computation in the `step` function (lines 295-296). First explain (in around three sentences) what effect the masks have on the entire attention computation. Then explain (in one or two sentences) why it is necessary to use the masks in this way.

h)

Run the following command to generate the necessary vocabulary file: `sh run.sh vocab`. Try to train the model on your local machine now: `sh run.sh train_local` Once you establish that training works locally, run the training on the machine with GPU by: `sh run.sh train`

i) (3 points)

Once your model is done training (this should take about 4 hours with one GPU), execute the following command to test the model: `sh run.sh test`

Please report the model's corpus BLEU Score. It should be larger than 9.

j) (2 points)

We learned about dot product attention, multiplicative attention, and additive attention. Please provide one possible advantage and disadvantage of each attention mechanism, with respect to either of the other two attention mechanisms. We define dot product attention is $\boldsymbol{e}_{t,i} = \boldsymbol{s}^T \boldsymbol{h}_i$, multiplicative attention is $\boldsymbol{e}_{t,i} = \boldsymbol{s}_t^T \boldsymbol{W} \boldsymbol{h}_i$, and additive attention is $\boldsymbol{e}_{t,i} = \boldsymbol{v}^T \tanh(\boldsymbol{W}_1 \boldsymbol{h}_i + \boldsymbol{W}_2 \boldsymbol{s}_t)$.

2 Analyzing NMT Trained Model

a) (3 points)

Now it is time to explore the outputs of the model that you have trained! The test-set translations your model produced in question 1 i) should be located in `outputs/test_outputs.txt`. Please identify 5 examples of errors that your model produced. For each example you should provide a reason why the model may have made the error (either due to a specific linguistic construct or specific model limitations) and describe one possible way we might alter the NMT system to fix the observed error.

b) (3 points)

Compare the results to the one from the PolEval track. We have slightly cheated with the evaluation of the model. We are calculating the BLEU score on the test sentences that are covered with our vocabulary.

Please run `sh run.sh test_full` and report the BLEU score. Check again `outputs/test_outputs.txt`. Why is the difference so large? What is the property of Polish language that is the main cause of the drop? What could be done to avoid this issue? Explain in less than 5 sentences.

c) (7 points)

BLEU Score is the most commonly used automatic evaluation metric for NMT systems. It is usually calculated across the entire test set, but here we will consider BLEU defined for a single example. Suppose we have a source sentence s, a set of k reference translations $r_1, ..., r_k$, and a candidate translation c. To compute the BLEU score of c, we first compute the modified n-gram precision p_n of c, for each of $n = 1, 2, 3, 4$:

$$p_n = \frac{\sum_{ngram \in c} \min\left(\max_{i=1,\dots k} \mathrm{Count}_{r_i}(ngram), \mathrm{Count}_c(ngram)\right)}{\sum_{ngram \in c} \mathrm{Count}_c(ngram)}$$

Here, for each of the n-grams that appear in the candidate translation c, we count the maximum number of times it appears in any one reference translation, capped by the number of times it appears in c (this is the numerator). We divide this by the number of n-grams in c (denominator).

Next, we compute the brevity penalty BP. Let c be the length of c and let r^* be the length of the reference translation that is closest to c (in the case of two equally-close reference translation lengths, choose r^* as the shorter one).

$$BP = \begin{cases} 1 & \text{if } c \geq r^* \\ \exp(1 - \frac{r^*}{c}) & \text{otherwise} \end{cases}$$

Lastly, the BLEU score for candidate c with respect to $r_1, ..., r_k$ is:

$$BLEU = BP \times \exp\left(\sum_{n=1}^{4} \lambda_n \log p_n\right),$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are weights that sum to 1.

Question 1: Please consider this example:

Source Sentence s: So this means a strict subset.
Reference Translation r_1: Czyli to oznacza podzbiór właściwy.
Reference Translation r_2: W takim razie to oznacza podzbiór właściwy.
NMT Translation c_1: Czyli to podzbiór właściwy.
NMT Translation c_2: W takim razie to oznacza jest zbiór właściwy.

Please compute the BLEU scores for c_1 and c_2. Let $\lambda_i = 0.5$ for $i \in \{1, 2\}$ and $\lambda_i = 0$ for $i \in \{3, 4\}$ (this means we ignore 3-grams and 4-grams, i.e., don't compute p_3 or p_4). Report your partial results to compute the BLEU scores (i.e. p_1, p_2, c, r^* and BP). Which of the two NMT translations is considered the better translation according to the BLEU score? Do you agree that it is the better translation?

Question 2: Our hard drive was corrupted and we lost Reference Translation r_2. Please recompute BLEU scores for c_1 and c_2, this time with respect to r_1 only. Which of the two NMT translations now receives the higher BLEU score? Do you agree that it is the better translation?

Question 3: Due to data availability, NMT systems are often evaluated with respect to only a single reference translation. Please explain (in a few sentences) why this may be problematic.

Question 4: List two advantages and two disadvantages of BLEU, compared to human evaluation, as an evaluation metric for Machine Translation.