KU Leuven - NLP 2021-2022
7 December 2021

Exercise Session 5
Machine Translation

For questions contact:
Toledo Forum for Questions or
nlp@ls.kuleuven.be

In Machine Translation we will see many types of algorithms and model designs for a second time. First we will practice with designing and describing a model using mathematical notations. We will also think about making design changes to the model needed for a different application.
Next we will look into back-propagation to update our weights. We will investigate what is needed to update the weights in a RNN.

1 Design a Translation Model

In the lecture you have seen many different architectures and options. You have seen drawing of the models and some simplified computations.
In this question you will combine everything you have seen. Define an entire encoder-decoder model. We will make use of a bi-LSTM as the encoder, and a LSTM for the decoder. We also add cross-attention to our model. We combine the bi-LSTM hidden states through concatenation.
For our source language, we use pre-trained word2vec embeddings of size 300.
Our target language vocabulary consists of 3000 words. For the target-language we also have pre-trained embeddings of size 300.
For predicting the target words, use a two-layer network with a non-linearity after the first layer.

1.A: Designing the model

1. First we need to update the target vocabulary (assume the source vocabulary is already updated). Use what you have learned in previous lectures and exercise sessions to correctly update the vocabulary. Describe your updates and why they are needed. What is the updated size?
2. Write down all the equations in the model. Make sure to be consistent in the letters and symbols used, so it is clear how layers connect to each other.
   You don't have to write down all the equations of every LSTM, simply write
   $$\text{outputs} = LSTM_{\text{name}}(\text{inputs})$$
   for every LSTM used.
3. How many weights matrices and bias vectors are there? Don't forget to count those in the LSTM.
4. Assume that we want the hidden state, that is the output for the encoder and the input for the decoder, to have a dimension of 128.
   What are the sizes of the matrices and the biases?

1.B: Design mistake! We need live translation!

Oh no, we designed our model while not considering the application! This model will be used for a live translation in the European Union. It will used for the German representative. He likes to speak in his native language. When he speaks, it will be translated directly into all the languages of the other members in the Union. Let us update the model so this is possible.
We will do this in two steps:

1. The first step is to make sure we can do live translations. What do we have to change about the model to make this possible? Think about the speed of predictions, design choices, quality of predictions, ...
2. The second step is to make sure we can translate to multiple languages.
   What are the changes to the model so this is possible?
   How would you train this model, so that it can work optimally for all the languages?

2 Training the RNN Decoder

In this question, we design a simple model so we can practice and experience back-propagation. We will only look at the decoder for now, otherwise the number of computations becomes too big.
Assume we have the following RNN Decoder:

$$\boldsymbol{h}^{(t+1)} = \boldsymbol{W}_{in}^T \boldsymbol{y}^{(t)} + \boldsymbol{W}_h^T \boldsymbol{h}^{(t)} + \boldsymbol{b}$$

with $\boldsymbol{W}_{in} \in \mathbb{R}^{3\times 2}$ the input to hidden matrix, $\boldsymbol{W}_h \in \mathbb{R}^{2\times 2}$ the hidden to hidden matrix, $\boldsymbol{b} \in \mathbb{R}^2$ the bias, and $\boldsymbol{h}^t \in \mathbb{R}^2$ the hidden state at time step t. $\boldsymbol{y}^{(t)} \in \mathbb{R}^3$ is the embedding for the previous word in the target sentence.
For predicting the output token we have the following function:

$$\boldsymbol{s}^{(t)} = \boldsymbol{W}_{score}^T \boldsymbol{h}^{(t)} + \boldsymbol{b}_{score}$$

with $\boldsymbol{W}_{score} \in \mathbb{R}^{2\times 4}$ the weight matrix, $\boldsymbol{b}_{score} \in \mathbb{R}^4$ the bias, and $\boldsymbol{s}^t \in \mathbb{R}^4$ the prediction scores for the word from the vocabulary at timestep t.
We have the following vocabulary:

```
[sos, word1, word2, word3]
```

And we predicted the following sentence:

```
word1 word2 word3
```

Assume we have a loss function $\mathcal{L}$.

2.A Draw the model

Start by drawing the entire unrolled RNN model, thus clearly showing the output at each timestep. Also, make sure that you show clearly all the weights on the arrows.

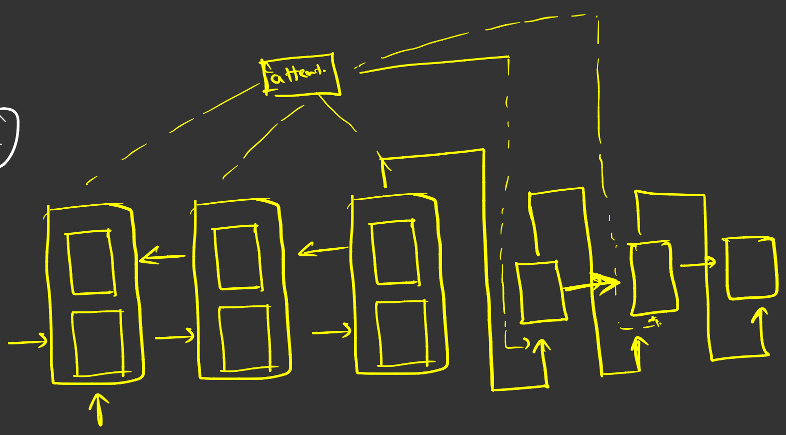
2.B Back propagation through time.

When we want to train and update the weights of the weights and biases, we have to do back-propagation. In an RNN, this is called Back-Propagation Through Time (BPTT).
Indicate by which computations each weight matrix is influenced. You can use the drawing from 2.A for this.
You don't have to compute or define the derivatives here, just consider how they influence each matrix.

3 Implementing NMT models

We provided you with a jupyter notebook file with the ground works for a NMT model. You can load this on your own device, but the easiest solution will be to open it with google colab.
In this code, you will see most of the basic implementation for training and testing a model. We are going to compare the difference between using a simple RNN and an LSTM.
Your goal is to fill several missing lines in the model and in the training loop. We indicated clearly in comments where you have to add some code.
Fill in the code and compare the results using the RNN and the LSTM.

(1)



(i) Add SOS token, EOS token and UNK token.

(ii) $X \leftarrow$ sequence of words

$e \leftarrow$ embedding of words (source & target)

$h \leftarrow$ hidden states of LSTM.

$c \leftarrow$ cell states of LSTM.

$h \leftarrow$ hidden states of bi-LSTM.

$a_i^j \leftarrow$ attention score of encoder state i & decoder j.

$\alpha_i^j \leftarrow$ attention weight for " " " " decoder j.

$; \leftarrow$ for concatenation.

$\rightarrow \leftarrow$ for direction in LSTM.

Let $X = \{x_1, x_2, \ldots, x_N\}$

$\therefore \overrightarrow{h}_0^e = 0$

$\therefore \overleftarrow{h}_{N+1}^e = 0$

$\therefore \overrightarrow{c}_0^e = 0$

$\therefore \overleftarrow{c}_{N+1}^e = 0$

$\therefore e_i^e = \mathrm{Emb}(x_i)$

$$\therefore \overrightarrow{h}_i^e, \overrightarrow{c}_i^e = \mathrm{LSTM}_f^{enc}(\overrightarrow{h}_{i-1}^e, \overrightarrow{e}_i^e, \overleftarrow{c}_{i-1}^e)$$

$$\therefore \overleftarrow{h}_i^e, \overleftarrow{c}_i^e = \mathrm{LSTM}_b^{enc}(\overleftarrow{h}_{i+1}^e, \overleftarrow{e}_i^e, \overleftarrow{c}_i^e)$$

$$\therefore h_i^e = [\overleftarrow{h}_i^e ; \overrightarrow{h}_i^e]$$

$$\therefore c_i^e = [\overleftarrow{c}_i^e ; \overrightarrow{c}_i^e]$$

$$\therefore h_0^d = [\overleftarrow{h}_1^e ; \overrightarrow{h}_N^e] \quad (128)$$

$$\therefore c_0^d = [\overleftarrow{c}_1^e ; \overrightarrow{c}_N^e] \quad (128)$$

$$\therefore a_i^j = \omega_\alpha^d [h_{j-1}^d \cdot h_i^e] + b_\alpha$$

$$\therefore \alpha_i^j = \frac{\exp(a_i^j)}{\sum_{i=1}^{N} \exp(a_i^j)}$$

$$\therefore s_i^d = \sum_{i=1}^{N} \alpha_i^j h_i^e$$

$$\therefore e_0^d = \mathrm{Emb}(SOS)$$

$$\therefore e_t^d = \mathrm{Emb}(\mathrm{argmax}(\hat{y}))$$

$$\therefore c_i^d, h_i^d = \mathrm{LSTM}^{dec}([e_i^d ; s_i^d], h_{i-1}^d, c_{i-1}^d)$$

$$O_t = W_2^T(\tan(W_1^T h_t^d + b_1)) + b_2$$

$$\hat{y}_t = \mathrm{softmax}(O_t^d)$$

$X \in \mathbb{R}^n$ $3 \times (8$ weight matrices $+ 4$ bias$) + 2 + 2 + 2$

$= 42$

$\overrightarrow{h} \times \overleftarrow{h} : 64 \times 64$

$X : N \times 64$

$W_\alpha^d : 256 \times 1$

$b_\alpha^d : 1$

decoder input : (128×1)

weight matrices : (128×128)

biases : 128×1

$W_1 : 128 \times 128$

$b_1 : 128 \times 1$

$W_2^T : 3003 \times 128$

$b_2^T : 3003$

(2) $h^{t+1} = W_{in}^T y^{(t)} + W_h^T h^{(t)} + b$

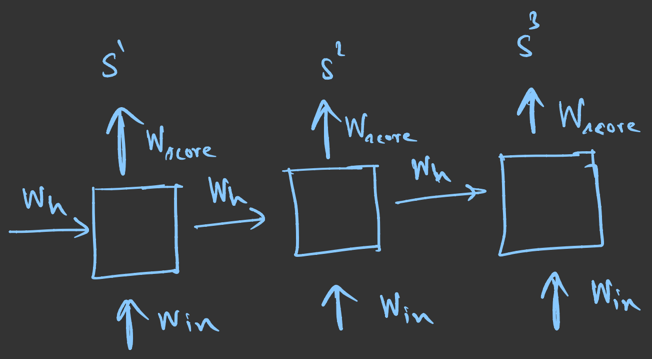
$W_{in} \in \mathbb{R}^{3 \times 2}$

$W_h \in \mathbb{R}^{2 \times 2}$

$b \in \mathbb{R}^2$

$h^t \in \mathbb{R}^2$

$y^{(t)} \in \mathbb{R}^3$


s^1
s^2
s^3
W_{score}
W_{score}
W_{score}
W_h
W_h
W_h
W_{in}
W_{in}
W_{in}