ADVANCED LEARNING FOR TEXT AND GRAPH DATA

Lab session 2: Neural Machine Translation

Lecture: Prof. Michalis Vazirgiannis
Lab: Moussa Kamal Eddine and Hadi Abdine

Tuesday, November 24, 2020

This handout includes theoretical introductions, coding tasks and questions. Before the deadline, you should submit here a **.zip** file (max 10MB in size) containing a `/code/` folder (itself containing your scripts with the gaps filled) and an answer sheet named `firstname_lastname.pdf`, following the template available here, and containing your answers to the questions. Your answers should be well constructed and well justified. They should not repeat the question or generalities in the handout. When relevant, you are welcome to include figures, equations and tables derived from your own computations, theoretical proofs or qualitative explanations. **One submission is required for each student. The deadline for this lab is November 30, 2020 11:59 PM**. No extension will be granted. Late policy is as follows: $]0, 24]$ hours late $\rightarrow$ -5 pts; $]24, 48]$ hours late $\rightarrow$ -10 pts; > 48 hours late $\rightarrow$ not graded (zero).

---

1 Learning objective

In this lab, you will learn about sequence to sequence (seq2seq) architectures. More precisely, we will implement the Neural Machine Translation (NMT) model described in [4] using Python 3.6 and PyTorch 1.3 (the latest version). The only difference is that we will be using non-stacked RNNs, whereas [4] uses stacked RNNs.

We will train our model on the task of English to French translation, using a set of sentence pairs from `http://www.manythings.org/anki/`, originally extracted from the Tatoeba project: `https://tatoeba.org/eng/`.

Our dataset features 136,521 pairs for training and 34,130 pairs for testing, which is quite small, but enough for the purpose of this lab. The average size of a source sentence is 7.6 while the average size of a target sentence is 8.3.

Note: the pairs have already been preprocessed. Each sentence was turned into a list of integers starting from 4. The integers correspond to indexes in the source and target vocabularies, that have been constructed from the training set, and in which the most frequent words have index 4. 0, 1, 2 and 3 are reserved respectively for the padding, out-of-vocabulary, start of sentence, and end of sentence special tokens.

2 Recurrent Neural Networks

2.1 Overview

While CNNs are good at dealing with grids, RNNs were specifically developed to be used with sequences. As shown in Fig. 1, a RNN can be viewed as a chain of simple neural layers that share the same parameters. From a high level, a RNN is fed an ordered list of input vectors $\{x_1, ..., x_T\}$ as well as an initial hidden state h_0 initialized to all zeros, and returns an ordered list of hidden states $\{h_1, ..., h_T\}$, as well as an ordered list of output vectors $\{y_1, ..., y_T\}$. The hidden states may serve as input to the RNN units above in the case of a stacked architecture, or directly be used as they are (e.g., by the attention mechanism). The hidden states correspond more or less to the "short-term" memory of the network.

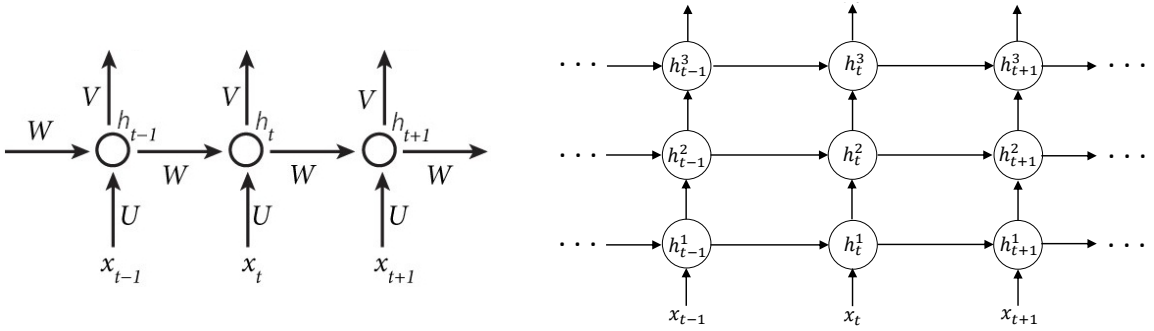

Figure 1: Left: 3 steps of an unrolled RNN (*adapted from Denny Britz' blog*). Right: 3 steps of an unrolled stacked RNN. The hidden states at a given position flow vertically through the RNN layers. On both sides, each circle represents a RNN unit.

3 Sequence-to-sequence architecture

Our input and output are sequences of words, respectively $x = (x_1, \ldots, x_{T_x})$ and $y = (y_1, \ldots, y_{T_y})$. x and y are usually referred to as the *source* and *target* sentences.

3.1 Encoder

Our encoder is a non-stacked unidirectional RNN with GRU units[1].

> **Task 1**
> Fill the gaps in the `forward` function of the `Encoder` class (in the `model.py` script).

3.1.1 Decoder

Our decoder is a non-stacked unidirectional RNN. It is a neural language model conditioned not only on the previously generated target words but also on the source sentence. More precisely, it generates the target sentence $y = (y_1, \ldots, y_{T_y})$ one word y_t at a time based on the distribution:

$$P\big[y_t|\{y_1, ..., y_{t-1}\}, c_t\big] = \text{softmax}\big(W_s\tilde{h}_t\big) \tag{1}$$

where $\tilde{h}_t$, the *attentional* hidden state, is computed as (biases are not shown for simplicity):

$$\tilde{h}_t = \tanh\big(W_c\big[c_t; h_t\big]\big) \tag{2}$$

[1] see the appendix for details about the GRU.

h_t is the t^{th} hidden state of the decoder, c_t is the source context vector, and $[;]$ denotes concatenation. W_s and W_c are matrices of trainable parameters.
Note: while all the inputs of the encoder (i.e., all the words of the input sentence) are known at encoding time, the decoder generates one target word at a time, and uses as input at time t its prediction from time $t-1$.

> **Task 2**
> Fill the gaps in the `forward` function of the `Decoder` class.

3.2 Global attention mechanism

The context vector c_t is computed as a weighted sum of the encoder's hidden states $\bar{h}_i$. The vector of weights α_t is obtained by applying a softmax to the output of an *alignment* operation (`score()`) between the current target hidden state h_t and all source hidden states $\bar{h}_i$'s. α_t indicates which words in the source sentence are the most likely to help in predicting the next word. `score()` can in theory be any comparison function. In our implementation, we will use the `concat` attention formulation of [4] (see section 3.1 of the paper). An overview is provided in Fig. 2.

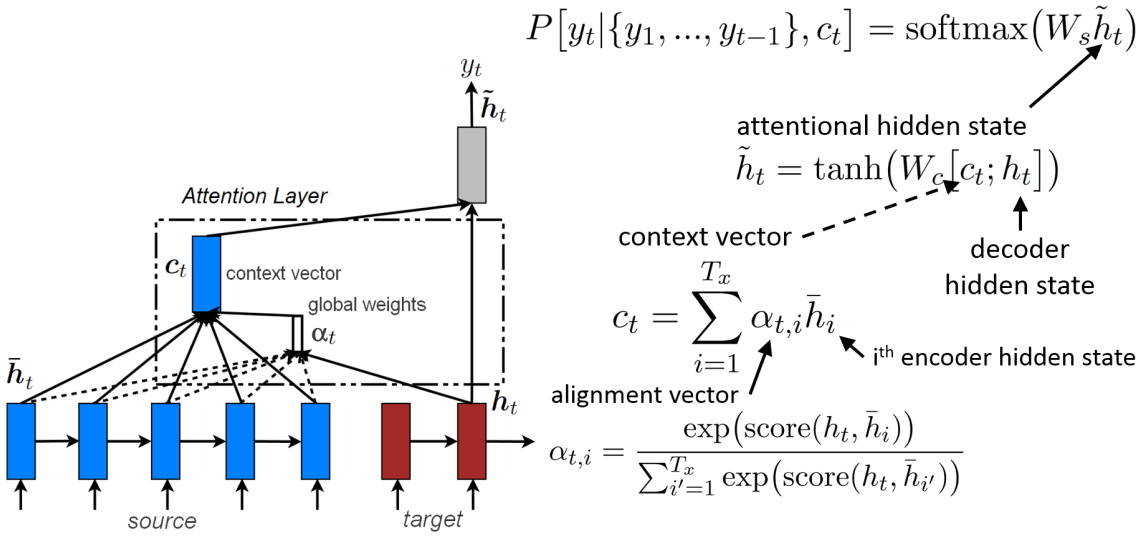

Figure 2: Summary of the *global attention* mechanism [4].

> **Task 3**
> Fill the gaps in the `forward` function of the `seq2seqAtt` class.

4 Training and evaluation

> **Task 4**
> Fill the gaps in the `forward` function of the `seq2seqModel` class.

> **Task 5**
> Check that your implementation is correct by running `main.py` with `is_prod=False` for a few epochs and verifying that the loss decreases.

> **Task 6**
> Run `main.py` with `is_prod=True`, using the pre-trained weights.

5 Questions

Question 1 (5 points)
What do you think about our greedy decoding strategy? Base your answer on slides 87-95 from this presentation (taken from this ACL tutorial).

Question 2 (5 points)
What major problem do you observe with our translations? How could we remediate this issue? You may find inspiration from reading [4, 6].

Question 3 (5 points)
Write some code to visualize source/target alignments in the style of Fig. 3 in [1] or Fig. 7 in [4] . Provide and interpret your figures for some relevant examples (e.g. to illustrate adjective-noun inversion).

Question 4 (5 points)
What do you observe in the translations of the sentences below? What properties of language models does that illustrate? Read [3, 5] to get some ideas.

- `I did not mean to hurt you`
- `She is so mean`

6 Appendix

6.1 GRU unit

As shown in Fig. 3, the GRU unit [2] is a simple RNN unit with two gates (reset and update):

$$\text{reset gate: } r_t = \sigma\big(U_r x_t + W_r h_{t-1} + b_r\big) \tag{3}$$

$$\text{update gate: } z_t = \sigma\big(U_z x_t + W_z h_{t-1} + b_z\big) \tag{4}$$

The candidate hidden state is computed as:

$$\hat{h}_t = \tanh\big(U_h x_t + W_h(r_t \circ h_{t-1}) + b_h\big) \tag{5}$$

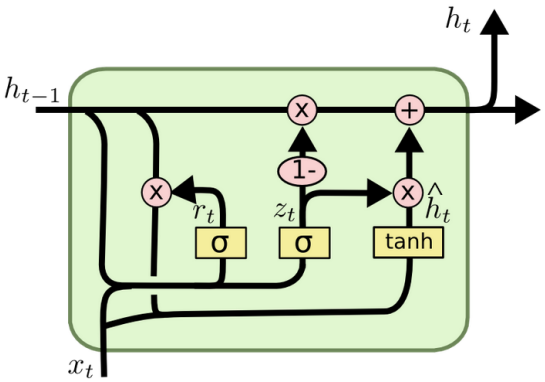

Figure 3: GRU unit. Taken from *Chris Olah's blog*.

The reset gate determines how much of the information from the previous time steps (stored in h_{t-1}) should be discarded. The new hidden state is finally obtained by linearly interpolating between the previous hidden state and the candidate one:

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \hat{h}_t \tag{6}$$

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[5] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[6] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. Modeling coverage for neural machine translation. *arXiv preprint arXiv:1601.04811*, 2016.