ADVANCED LEARNING FOR TEXT AND GRAPH DATA

Lab session 7: Transformer

Lecture: Prof. Michalis Vazirgiannis
Lab: Jean-Baptiste Remy and Antoine Tixier

Tuesday, January 14, 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 January 20, 2020 11:59 PM**. No extension will be granted. Late policy is as follows: $]0, 24]$ hours late → -5 pts; $]24, 48]$ hours late → -10 pts; > 48 hours late → not graded (zero).

1 Learning objective

In this lab, you will learn about the Transformer architecture. We will implement, step by step, this Neural Machine Translation (NMT) model described in [8] using Python 3.6 and PyTorch 1.3 (the latest version).

We will train our model on the task of English to French translation, using the same dataset as for the fourth lab (19/11/19). We provide a brief description of this dataset again for your convenience: it is a set of sentence pairs from http://www.manythings.org/anki/, originally extracted from the Tatoeba project: https://tatoeba.org/eng/. It 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.

2 Attention is all you need

From the previous labs, we have seen 2 ways of representing texts in a neural network. Convolutional Neural Networks (CNNs) [6] represent documents as a collection of weighted n-grams. Convolutional layers are fast to compute as they fully leverage GPUs, but have no grammatical understanding of a document. On the other hand, Recurrent Neural Networks (RNNs) [3] are built to read text as a human would do and to take into account long term dependencies in text. However they are, by design, not adapted to GPU computing and are difficult to train consistently.

In the paper *Attention Is All You Need* [8], the authors propose to use only dense layers and self-attention layers to learn representations.

2.1 Scaled Dot-Product Attention

Here, we briefly remind the principle of attention and introduce the framework proposed by [8]. Attention was originally proposed by [2] as a mean to align text in the source and target languages and to relieve the encoder of the burden to encode the full sentence into a unique vector.

The Scaled Dot-Product attention computes attention over Values (V) based on the relation between Keys (K) and Queries (Q). Formally, $K \in \mathbb{R}^{T_1,d}$, $Q \in \mathbb{R}^{T_2,d}$ and $V \in \mathbb{R}^{T_2,d}$ are three sequences represented on d dimensions. The attention layer computes:

$$\text{Attention}(Q, K, V) = \text{softmax}\Big(\frac{QK^\top}{\sqrt{d}}\Big)V$$

Where T_1 and T_2 are the lengths of the sequences. Dot product scales with the dimension of Q and K thus the division by $\sqrt{d}$ that keeps the result away from low gradients region of the softmax function.

Question 1 (1 points)
Let's consider two sequences x and y. What sequences play the role of Q, K and V in the following situations:

- Attention of y over x, as in the classical encoder-decoder framework.
- Self-attention over x.

2.2 Self-attention

We talk about self-attention when a sentence attends to itself. In each layer of self-attention a word pulls the representation of the words he is in relation with.

The author compare self-attention against CNNs and RNNs on three criteria: Complexity per Layer, Sequential Operations, the number of operations that need to be done in sequential order (the lower the more parallelisable the layer), Maximum Path Length, and the number of intermediary neurons between the first and last words of a sentence (i.e. the capacity to take into account long term dependencies).

Question 2 (3 points)
Prove the identities of Table 1.

Based on the observations of Table 1, the author argue that self-attention is extremely efficient to encode and decode text. It is extremely cheap to compute, and highly parallelisable. Moreover, as we said earlier, stacking self-attention layers allows to builds complex relationships between words. Finally long term dependencies are taken into account just as much as short term ones.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(n/k)$

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions.

2.3 Multi-Head Attention

Instead of using one instance of the previous module at every layer, Multi-Head Attention uses in parallel multiple heads of smaller dimensions and then concatenates their outputs.

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

The projection matrices $W_1^Q, \dots, W_h^Q, W_1^K, \dots, W_h^K, W_1^V, \dots, W_h^V, W^O$ are the parameters of the network. For all i $W_i^Q \in \mathbb{R}^{d_{model}, d_k} W_i^K \in \mathbb{R}^{d_{model}, d_k}$ and $W_i^V \in \mathbb{R}^{d_{model}, d_v}$. The authors choose $d_v = d_k = d_{model}/h$. d_{model} is the dimensionality of text representation throughout the network.

Question 3 (2 points)
According to you, what is the point of having multiple heads of small dimension ?

2.4 Implementation

Task 1
Complete the `forward` function of the `MultiHeadAttention` class. Pay attention to the fact that the heads need to run in parallel. Although it would be easy to ensure this in TensorFlow, PyTorch requires some attention. You can use the `unsqueeze`, `squeeze` and `permute` methods of `torch.Tensor` and we strongly advise that you read carefully the documentation for `torch.matmul`. *For now, ignore the* `maskout` *option.*

3 Architecture

Now that we have introduced the core concept of Transformer, we will describe its overall architecture and modules. Figure 1 presents the overall architecture of the network. The transformer uses the Encoder-Decoder with attention framework that was introduced in lab 4.

Feed-Forward module In addition to the Multi-Head Attention, Transformer uses a Feed-Forward module. This module is a classical fully connected network with one hidden layer.

$$\mathrm{FFN}(x) = \mathrm{relu}(xW_1 + b_1)W_2 + b_2 \tag{1}$$

The author use a very high, 2048, hidden dimension for those modules.

Task 2
Complete the `forward` function of the `FeedForward` class.

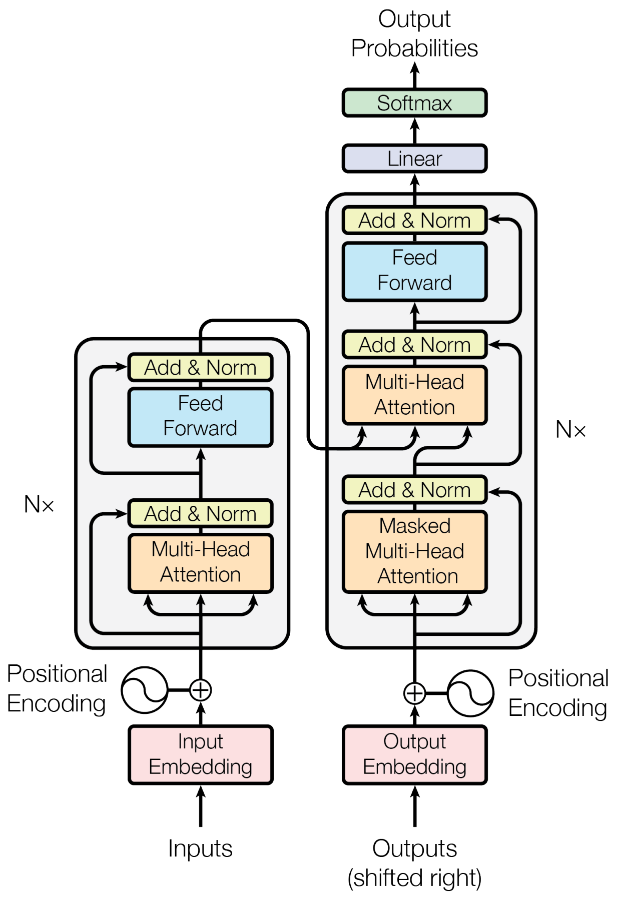

Figure 1: Overall architecture of the Transformer

Add & Norm In between every module of the network, there is a residual connection [5] and a layer normalization [1]. Those two mechanisms are classical in deep learning and mainly serve as optimization tricks. To implement them you can use a simple addition and the `torch.functional.layer_norm` function.

Positional Encoding With self-attention layers there is no comprehension of the order of the text, only relationships between words. To take into account the sequential aspect of the text, a Positional-Encoding is added to the inputs of the encoder and the decoder after the embedding layer.

$$PE_{(pos,2i)} = sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$
$$PE_{(pos,2i+1)} = cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

Where $PE_{(pos,j)}$ is added to the j^{th} dimension of the representation of the word in position pos in the sentence.

Question 4 (2 points)
The author argue that this encoding works because, for any k, PE_{pos+k} can be represented as a linear function of PE_{pos}. Prove it and explain why it makes it a suitable encoding.

3.1 Encoder

The encoder is a stacking of N stacks. One encoder stack consists of a Multi-Head self-Attention layer and a Feed-Forward Module.

Task 3
Complete the `forward` function of the `EncoderStack` class.

Each stack takes as input the output of the previous one, and the overall output of the encoder is the output of the last stack.

3.2 Decoder

The decoder is a bit more tricky. Since the model is sequence to sequence, it is auto-regressive. The input of the decoder is the sequence translated in the target language and shifted to the right so that every word is translated based on the translation of the previous words. At test time, we need to process word by word starting with a `<SOS>` token. During training, we use teacher forcing. Due to the construction of the network, if we know the expected output, we can process all the decoding in one pass. This is one of the main advantages of Transformer. During training, the input of the decoder is thus the expected translation with a `<SOS>` token added at the beginning.

Leftward Masking Of course, we can't use future information to translate one word, we thus need to cut left-ward connections in the decoder. This can be achieved by only masking out leftward connections in the self-attention layer of the decoder. Formally, we set every input of the softmax in this layer corresponding to an illegal connection to $-\infty$. Those connections correspond to the upper triangular part of the weight matrix.

Task 4
Add masking to the implementation of the `forward` function of the `MultiHeadAttention` class. This should be applied when the `maskout` option is true.

Question 5 (3 points)
Explain how this masking effectively prevents use of illegal information, especially during training.

Connection with the encoder The decoder is also a stacking of N identical stacks, each stack is connected to the output of the encoder. Meaning that a stack takes as input the output of the previous stack and the output of the encoder at the level of the second attention layer in the stack. For this second attention layer, the Queries are the output of the previous self attention layer and the Keys and Values are the output of the encoder.

Task 5
Complete the `forward` function of the `DecoderStack` class. Remember to activate the `maskout` option when necessary.

The last layer of the decoder is a fully connected layer followed by a softmax, to transform the output into a probability distribution over the target language vocabulary.

3.3 Implementation

Task 6
We now have all the elements we need. Complete the `forward` function of the `Transformer` class. Remember to differentiate decoding between training and testing.

4 Questions

Question 6 (2 points)
How many coefficients are there in the model described by the original paper? Do you think you could train it, even with a GPU?

Question 7 (3 points)
Train a small version of the model and display the logs and test translations you obtained.

Question 8 (4 points)
Recently, ELMo and BERT [4, 7] made a big impact on the NLP deep learning community. While the main idea of the papers rely on the training tasks, BERT leverages Transformer. By reading (at least) the ”Model Architecture” and ”Input/Output Representations” of BERT's paper [4], explain how the network represents a sentence. Based of your understanding of the Multi-Head self-Attention, is this representation efficient ?

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

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

[3] 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.

[4] 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.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[7] 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.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.