ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prof. Ryan Cotterell

Assignment 6: Attention and Transformers

08/01/2023 - 13:03h

The first half of the assignment is a series of theoretical questions about the material. When you have answered the questions to your satisfaction, you should upload a PDF file with your solutions (preferably written in LaTeX) to Moodle. The second question is a coding task that is to be solved in the released Google Colab notebook. You have to copy the notebook to your own drive in order to edit it. When submitting your solution, please execute all cells in your copied notebook, then save it and include a link to your notebook in your PDF file submission. We will run software on your submission to *test for plagiarism*.

Important: Please ensure, that all cells in the notebook are already **executed** and that the notebook is accessible via the shared link in **Editor mode**! The notebook must also be executable from top to bottom without throwing errors.

Question 1: Attention and Convolution (60 pts)

We'll start off this section by reviewing briefly the formulation of self-attention and convolutions. Both methods have been used, separately and jointly, in various NLP tasks. Our goal with this question is to reason about what similarities exist between them.

We define a Self-Attention layer of a neural network as a layer that takes as input the input matrix $\mathbf{Z} \in \mathbb{R}^{N \times D_i}$, representing N input tokens, and outputs an output matrix $\mathbf{O} \in \mathbb{R}^{N \times D_o}$ according to the following operation:

$$\mathbf{O} = \text{Self-Attention}(\mathbf{Z}) := \text{softmax}\left(\mathbf{Z}\mathbf{W}_q\mathbf{W}_k^\top\mathbf{Z}^\top\right)\mathbf{Z}\mathbf{W}_v \tag{1}$$

where $\mathbf{W}_q \in \mathbb{R}^{(D_i \times D_{hid})}, \mathbf{W}_k \in \mathbb{R}^{(D_i \times D_{hid})}, \mathbf{W}_v \in \mathbb{R}^{(D_i \times D_o)}$ are learnable parameter matrices for the query, key and value respectively. This particular form of attention is called *self*-attention, as all the weight matrices are multiplied with the same input matrix $\mathbf{Z}$. For clarity, we define the attention matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ as

$$\mathbf{A} := \mathbf{Z}\mathbf{W}_q\mathbf{W}_k^\top\mathbf{Z}^\top \tag{2}$$

The elements of this matrix are called *attention scores* and indicate how relevant each token is to every other token. We denote as *attention probabilities* the elements of $\text{softmax}(\mathbf{A})$.

It has been empirically proven that it is beneficial to perform the attention mechanism over *multiple heads*, that is, to perform self-attention many times in parallel in a single layer, each time with different weight matrices. The outputs of the attention heads are then combined as follows:

$$\text{MultiHead-Self-Attention}(\mathbf{Z})_{t,:} := \left(\underset{h \in [N_h]}{\text{concat}}[\text{Self-Attention}_h(\mathbf{Z})]\mathbf{W}_{out}\right)_{t,:} + \mathbf{b}_{out} \tag{3}$$

where $\mathbf{W}_{out} \in \mathbb{R}^{(N_h \cdot D_o) \times D_{out}}$ and $\mathbf{b}_{out} \in \mathbb{R}^{1 \times D_{out}}$ are again learnable parameters and N_h is the number of heads.

The rows of the input matrix $\mathbf{Z}$ are usually token embeddings representing a word token in a vector space. Lately, however, attention layers have also been used with image patches as input, obtaining good results in Computer Vision tasks such as Image Classification.

In this case, the input image will be encoded with a tensor $\mathbf{Z} \in \mathbb{R}^{(W \times H \times D_i)}$, where W and H are the sizes of the input image, and D_i is the number of channels. We want to adapt the formulation of self-attention in Equation 1 to work with pixels $\mathbf{p} = (i, j)$ instead of word tokens. We simplify notation by indexing matrices with $\mathbf{Z}_{\mathbf{p},:}$ to mean $\mathbf{Z}_{i,j,:}$.

(a.1) How many learnable parameters are contained in a multi-head self-attention layer with N_h heads, such as the one described by Equation 3? **(2.5 pts)**

(a.2) Rewrite Equation 1 and Equation 3 for a single query pixel $\mathbf{q}$, obtaining Self-Attention$(\boldsymbol{Z})_{\mathbf{q},:} \in \mathbb{R}^{(1 \times D_o)}$ and MultiHead-Self-Attention$(\mathbf{Z})_{\mathbf{q},:} \in \mathbb{R}^{(1 \times D_{out})}$, respectively. **(5 pts)**

A characteristic of self-attention as defined in Equation 1 is that it is permutation invariant. However, both when we work with text sequences and with images, the position of each token or pixel carries important information that we would like to preserve. To do so, we use a (learned or fixed) positional matrix $\mathbf{P} \in \mathbb{R}^{N \times D_i}$ changing the form of the attention matrix $\mathbf{A}$ in Equation 2 as follows:

$$\mathbf{A} := (\mathbf{Z} + \mathbf{P})\mathbf{W}_q\mathbf{W}_k^\top(\mathbf{Z} + \mathbf{P})^\top \tag{4}$$

Positional encodings can be either *absolute* or *relative*. In absolute encodings, a position vector for an input pixel $\mathbf{p}$, $\mathbf{P}_{\mathbf{p},:}$, depends only on the absolute position of $\mathbf{p}$ in the input. Therefore, we can expand Equation 4, for query pixel $\mathbf{q}$ and key pixel $\mathbf{k}$, obtaining:

$$\begin{aligned} \mathbf{A}^{absolute}_{\mathbf{q},\mathbf{k}} &= \left(\mathbf{Z}_{\mathbf{q},:} + \mathbf{P}_{\mathbf{q},:}\right)\mathbf{W}_q\mathbf{W}_k^\top\left(\mathbf{Z}_{\mathbf{k},:} + \mathbf{P}_{\mathbf{k},:}\right)^\top = \\ &= \mathbf{Z}_{\mathbf{q},:}\mathbf{W}_q\mathbf{W}_k^\top\mathbf{Z}_{\mathbf{k},:}^\top + \mathbf{Z}_{\mathbf{q},:}\mathbf{W}_q\mathbf{W}_k^\top\mathbf{P}_{\mathbf{k},:}^\top + \mathbf{P}_{\mathbf{q},:}\mathbf{W}_q\mathbf{W}_k^\top\mathbf{Z}_{\mathbf{k},:} + \mathbf{P}_{\mathbf{q},:}\mathbf{W}_q\mathbf{W}_k^\top\mathbf{P}_{\mathbf{k},:} \end{aligned} \tag{5}$$

On the other hand, in relative encoding, the position vector for an query pixel depends only on the relative position difference $\boldsymbol{\delta} := \mathbf{q} - \mathbf{k} = (\boldsymbol{\delta}_1, \boldsymbol{\delta}_2) \in \mathbb{Z}^2$ between itself and a key pixel.

$$\mathbf{A}^{relative}_{\mathbf{q},\mathbf{k}} := \mathbf{Z}_{\mathbf{q},:}\mathbf{W}_q\mathbf{W}_k^\top\mathbf{Z}_{\mathbf{k},:}^\top + \mathbf{Z}_{\mathbf{q},:}\mathbf{W}_q\widetilde{\mathbf{W}}_k\mathbf{r}_{\boldsymbol{\delta}} + \mathbf{u}^\top\mathbf{W}_k\mathbf{Z}_{\mathbf{k},:} + \mathbf{v}^\top\widetilde{\mathbf{W}}_k\mathbf{r}_{\boldsymbol{\delta}} \tag{6}$$

Here, $\mathbf{u}$ and $\mathbf{v}$ are learnable vector parameters, and we introduce a new parameter matrix $\widetilde{\mathbf{W}}_k \neq \mathbf{W}_k$. While there are many types of relative positional encodings, in this question we are going to focus on a specific form of relative encoding denominated *gaussian encoding*, which obeys the following equations:

$$\mathbf{v}^{(h)} := -\alpha^{(h)}\begin{pmatrix} 1 \\ -2\boldsymbol{\Delta}_1^{(h)} \\ -2\boldsymbol{\Delta}_2^{(h)} \end{pmatrix}; \mathbf{r}_\delta := \begin{pmatrix} \|\boldsymbol{\delta}\|^2 \\ \boldsymbol{\delta}_1 \\ \boldsymbol{\delta}_2 \end{pmatrix}; \mathbf{W}_q = \mathbf{W}_k := \mathbf{0}; \widetilde{\mathbf{W}_k} := \boldsymbol{I} \tag{7}$$

We denote as D_p the dimension of both $\mathbf{v}^{(h)}$ and $\mathbf{r}_\delta$. In this case we have $D_p = 3$. $\mathbf{\Delta}_1^{(h)}$ and $\mathbf{\Delta}_2^{(h)}$ are also learnable parameters.

(b.1) Plug in the expressions given in Equation 7 to simplify $\mathbf{A}_{\mathbf{q},\mathbf{k}}^{relative}$ in Equation 6. **(2.5 pts)**

(b.2) What's the computational cost of a single-head attention layer using absolute encoding? Assume that $\mathbf{P}$ has already been computed beforehand. What would the computational cost be if the attention layer used gaussian encoding instead? Use big-$\mathcal{O}$ notation to compare the two with respect to $D_x := D_i = D_{hid} = D_o \gg D_p$ and N. **(10 pts)**

We now briefly recall the structure of a convolution. Given an image tensor $\mathbf{Z} \in \mathbb{R}^{W \times H \times C_{in}}$, a weight matrix $\mathbf{W} \in \mathbb{R}^{K \times K \times C_{in} \times C_{out}}$, and a bias vector $\mathbf{b} \in \mathbb{R}^{C_{out}}$, where K is the kernel size and C_{in} and C_{out} are the number of input and output channels respectively, the output of a convolutional layer for a single pixel $\mathbf{p} = (i, j)$ has the following form:

$$\text{Convolution}(\mathbf{Z})_{i,j,:} := \left(\sum_{(\delta_1,\delta_2) \in \Delta_K} \mathbf{Z}_{i+\delta_1, j+\delta_2, :} \mathbf{W}_{\delta_1,\delta_2,:,:} \right) + \mathbf{b} \tag{8}$$

where Δ_K is the set of all possible shifts allowed by the convolutional kernel of size K.

$$\Delta_K := \left[-\left\lfloor \frac{K}{2} \right\rfloor, \cdots, \left\lfloor \frac{K-1}{2} \right\rfloor \right] \times \left[-\left\lfloor \frac{K}{2} \right\rfloor, \cdots, \left\lfloor \frac{K-1}{2} \right\rfloor \right] \tag{9}$$

Finally, we are interested in proving the following theorem that highlights important similarities between the computational capabilities of self-attention layers and convolutional layers of a neural architecture.

Main Theorem: A multi-head self-attention layer operating on K^2 heads of dimension D_o and output dimension D_{out}, employing a relative positional encoding of dimension $D_p \geq 3$, can express any convolutional layer of kernel size $K \times K$ and D_{out} output channels.

We'll split the proof of this Main Theorem by separately proving two sub-theorems. Theorem 1 follows:

Theorem 1: Given a multi-head self-attention layer with $N_h = K^2$ heads, $D_o \geq D_{out}$, let $\mathbf{f} : [N_h] \to \Delta_K$ be a bijective map between heads and shifts. We also suppose that for every head we have:

$$\text{softmax}\left(\mathbf{A}_{\mathbf{q},:}^{(h)}\right)_{\mathbf{k}} = \begin{cases} 1 & \text{if } \mathbf{f}(h) = \mathbf{q} - \mathbf{k} \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

Then, for any convolutional layer with a $K \times K$ kernel and D_{out} output channels exists a corresponding $\{\mathbf{W}_v^{(h)}\}_{h \in N_h}$ so that MultiHead-Self-Attention$(\mathbf{Z})$ = Convolution$(\mathbf{Z})$ for any input tensor $\mathbf{Z} \in \mathbb{R}^{W \times H \times D_{in}}$.

(c.1) We can rewrite Equation 3 in the following form:

$$\text{MultiHead-Self-Attention}(\boldsymbol{Z}) = \sum_{h \in [N_h]} \text{softmax}\left(\mathbf{A}^{(h)}\right) \mathbf{Z}\mathbf{W}^{(h)} + \mathbf{b}_{out} \tag{11}$$

Express $\mathbf{W}^{(h)}$ in terms of $\mathbf{W}_v$ and $\mathbf{W}_{out}$. Then, use $\mathbf{W}^{(h)}$ to write an expression for MultiHead-Self-Attention$(\mathbf{Z})_{\mathbf{q},:} \in \mathbb{R}^{D_{out}}$ **(5 pts)**

(c.2) Use the assumptions of Theorem 1 to express MultiHead-Self-Attention$(\mathbf{Z})_{\mathbf{q},:}$ in a form that is equivalent to a convolutional layer as defined in Equation 8. **(5 pts)**

We now have to prove whether we can construct a particular positional encoding that would give us the same set of assumptions that we leveraged in Theorem 1. We, therefore, try to prove the following theorem:

Theorem 2: It is possible to construct a relative encoding scheme $\{\mathbf{r}_{\boldsymbol{\delta}} \in \mathbf{R}^{D_p}\}_{\boldsymbol{\delta} \in \mathbb{Z}^2}$, using parameters $\mathbf{W}_q$, $\mathbf{W}_k$, $\widetilde{\mathbf{W}}_k$ and $\mathbf{u}$ so that, for every shift $\boldsymbol{\Delta} \in \boldsymbol{\Delta}_K$, there exists a vector $\mathbf{v}$ that yields the mapping $\mathbf{f}$ of Equation 10.

We will leverage the Gaussian Encoding defined in Equation 7 and assume that the following equation holds

$$\mathbf{A}_{\mathbf{q},\mathbf{k}} = -\alpha(||\boldsymbol{\delta} - \boldsymbol{\Delta}||^2 + c) \tag{12}$$

where c is a constant value.

(d.1) Solve the following limit for both $\boldsymbol{\delta} = \boldsymbol{\Delta}$ and $\boldsymbol{\delta} \neq \boldsymbol{\Delta}$ and explain why it satisfies Equation 10. **(10 pts)**

$$\lim_{\alpha \to +\infty} \text{softmax}(\mathbf{A}_{\mathbf{q},:})_{\mathbf{k}} \tag{13}$$

(d.2) Use the definition of Gaussian Encoding and determine for which constant value c the assumption in Equation 12 holds. **(5 pts)**

Putting together Theorem 1 and Theorem 2, we have proved the Main Theorem, demonstrating that, under certain constraints, an attention layer can learn to behave like a convolutional layer.

Finally, we are interested in determining whether this result holds for different hyperparameters that are usually tweaked when working with convolutional networks. If you are not familiar with strides, paddings, or dilations, here (click on "here" for the link!) you can find a visual representation of how they work.

(e.1) For which padding sizes and values is the output of a multi-head self-attention layer equivalent to the one of a convolutional layer? **(5 pts)**

(e.2) Can a multi-head self-attention layer express an arbitrarily-dilated convolution, up to structural limits concerning the input image? Explain your reasoning. **(5 pts)**

(e.3) Can a multi-head self-attention layer learn to simulate a strided convolution? If so, why? Otherwise, why is it not possible? **(5 pts)**

Question 2: Coding - Attention and Convolutions (40 pts)

In this question, you'll be asked to complete some coding tasks with the end goal of empirically testing whether what you have demonstrated theoretically in the first section of this assignment can also be observed in practice.

We provide a Google Colab notebook at
`https://colab.research.google.com/drive/1-Xj3afsltEK6LNqsD_dBxbp37WTWQsZ_?usp=sharing`
that contains the implementation of a BERT-like model that is to be trained on downscaled images of the CIFAR-10 dataset to assess whether self-attention layers can learn to perform like convolutions when using Gaussian Encoding. However, some parts of the implementation are missing. You'll be asked to complete the missing cells in order to make the code run properly.

There are a several cells, marked with "(DO NOT CHANGE)", that you are not allowed to modify! To provide your answers, you should only fill in the marked cells in the notebook by adding your own code. Do not change the signatures of existing functions and classes. Also, you are not allowed to import any libraries other than the ones that are being automatically imported in the cell "Library Imports".

Important: Please ensure that all cells in the notebook are already **executed** and that the notebook is accessible via the shared link in **Editor mode**! The notebook must also be executable from top to bottom without throwing errors.

(a) For this subquestion, you'll work with the cell marked `Question a)`. Fill in the missing code for the `AbsolutePositionalEmbedding` class and the `RelativePositionalEmbedding` class. Then, fill in the missing code for the `visualize_abs_positional_embedding` and the `visualize_rel_positional_embedding` methods. Finally, use these classes and methods to write an additional method `verify_limit` that visually proves the limit results in Equation 13. Refer to the comments included in the code skeleton of the cell for more details. **(15 pts)**

(b) One of the differences between a standard Transformer Encoder and a BERT Encoder layer is the use of the GELU activation functions in place of the ReLU activations. Follow the directions in the `Question b)` cell to implement two versions of the GELU activation function and visualize their differences. **(5 pts)**

(c) Layer normalization is heavily used in the Transformer architecture. Use the code skeleton provided in the `Question c)` cell to implement the layer normalization layer that will be used by the BERT model. **(5 pts)**

(d) A Transformer Encoder can be divided into two halves. The first, implementing self-attention (with Gaussian Encoding), a residual connection, and a layer norm layer, is already implemented. The second half, however, is still missing. Use the code skeleton in the `Question d)` cell to complete the `BertFFN` class. **(5 pts)**

(e) If you have answered questions (b) to (d) by writing the requested code in the corresponding cells, you should be able to run the `Question e)` cell without errors and start the training procedure for the BERT model. The training script will use Tensorboard to plot the position of the attention heads for each encoder layer over time.

Run the training script for a few dozen of epochs and observe the placement change of the attention heads between the beginning and the end of the training procedure. What movements do you observe? What would you expect to observe to support the statement of the Main Theorem proven in the theory section of this sheet? **(10 pts)**

Note: Training the model for a few dozen epochs is sufficient to answer this question, since fully finishing the training loop takes many hours and is therefore not required.