CSCI-GA 2572 Deep Learning
Homework 4: Transformer

Chuanyang Jin

April 2, 2023

1.1 Attention (13pts)

This question tests your intuitive understanding of attention and its property.

(a) (1pts) Given queries $\boldsymbol{Q} \in \mathbb{R}^{d\times n}$, $\boldsymbol{K} \in \mathbb{R}^{d\times m}$ and $\boldsymbol{V} \in \mathbb{R}^{t\times m}$, what is the output $\boldsymbol{H}$ of the standard dot-product attention? (You can use the softargmax$_\beta$ function directly. It is applied to the column of each matrix).

Solution.

$$\boldsymbol{H} = \text{softargmax}_\beta(\boldsymbol{Q}^T\boldsymbol{K})\boldsymbol{V}^T$$

□

(b) (2pts) Explain how the scale β influence the output of the attention? And what β is conveniently to use?

Solution.

The parameter β controls the degree of "softness". A higher value of β will make the softargmax function more similar to an argmax function, where only a few input values will have significant impact on the output, while the rest will have negligible contributions. This behavior can lead to a more focused or selective attention mechanism, as the model will rely on fewer input values.

On the other hand, a lower value of β will make the softargmax function smoother, where the output will be influenced by a larger number of input values, leading to a more global attention mechanism.

In practice, it is common to set β equal to the inverse of the square root of the dimensionality of the key/query vectors, i.e., $\beta = 1/\sqrt{d_k}$. This scaling factor has been empirically found to work well in various applications. □

(c) (3pts) One advantage of the attention operation is that it is really easy to preserve a value vector $\boldsymbol{v}$ to the output $\boldsymbol{h}$. Explain in what situation, the outputs preserves the value vectors. Also, what should the scale β be if we just want the attention operation to preserve value vectors. Which of the four types of attention we are referring to? How can this be done when using fully connected architectures?

Solution.

When the attention map is highly peaked with a value of 1 at one position and 0 everywhere else, it implies that one value vector will dominate the output vector. In this situation, the output is a spread version of the dominating value vector. To achieve this effect, the scale parameter β should be set to a large value. Among the four types of attention (self/cross, hard/soft), this type is known as hard cross-attention.

In a fully connected architecture, we can obtain a similar output by setting the weights associated with the desired input vector to 1 and those associated with the other input vectors to 0. This can be accomplished by modifying the weights of the linear layers or using activation and pooling layers. For example, max pooling can be used to select the output vector that corresponds to the desired input vector, while softmax or softmin with a large β can be used to achieve the spread effect. □

(d) (3pts) On the other hand, the attention operation can also dilute different value vectors $\boldsymbol{v}$ to generate new output $\boldsymbol{h}$. Explain in what situation the outputs is spread version of the value vectors. Also, what should the scale β be if we just want the attention operation to diffuse as much as possible. Which of the four types of attention we are referring to? How can this be done when using fully connected architectures?

Solution.

When the attention map is diffuse, the attention operation dilutes many value vectors $\boldsymbol{v}$ to generate a spread version of the input vectors in the output $\boldsymbol{h}$. This effect is achieved by setting the scale parameter β to a small value or approaching zero, which makes the attention map similar to the non-weighted average of the input vectors. In fact, when β is exactly zero, we obtain the non-weighted average of $\boldsymbol{v}$. This type of attention is known as soft cross attention.

In a fully connected architecture, we can obtain a similar output by setting the weights associated with each input vector and the output to be equal. We can also use activation and pooling layers such as average pooling, which averages over many output vectors, or softmax (or softmin) with a small β, which distributes the weights more evenly among the input vectors. □

(e) (2pts) If we have a small perturbation to one of the $\boldsymbol{k}$ (you could assume the perturbation is a zero-mean Gaussian with small variance, so the new $\hat{\boldsymbol{k}} = \boldsymbol{k} + \boldsymbol{\epsilon}$), how will the output of the $\boldsymbol{H}$ change?

Solution.

When a small perturbation is added to one of the key vectors, $\boldsymbol{k}$, resulting in $\hat{\boldsymbol{k}} = \boldsymbol{k} + \boldsymbol{\epsilon}$, the output of $\boldsymbol{H}$ will also change slightly. Since the softargmax function is differentiable, the change in $\boldsymbol{k}$ will cause a small change in the attention scores, which in turn affects the weighted sum of the value vectors in $\boldsymbol{H}$. However, if the perturbation is small enough, the overall impact on the output will also be relatively small, and the model's performance should not be drastically affected. □

(f) (2pts) If we have a large perturbation that it elongates one key so the $\hat{\boldsymbol{k}} = \alpha\boldsymbol{k}$ for $\alpha > 1$, how will the output of the $\boldsymbol{H}$ change?

Solution.

If a large perturbation elongates one key, resulting in $\hat{\boldsymbol{k}} = \alpha\boldsymbol{k}$ for $\alpha > 1$, the output of $\boldsymbol{H}$ will be more significantly affected. The attention scores calculated from the dot product $\boldsymbol{Q}^T\hat{\boldsymbol{K}}$ will be influenced by the change in scale of the key vector, which can cause the softargmax function to assign higher attention weights to the corresponding value vector. This leads to a more prominent impact on the weighted sum of value vectors in $\boldsymbol{H}$, potentially changing the output significantly and affecting the model's performance. □

1.2 Multi-headed Attention (3pts)

This question tests your intuitive understanding of Multi-headed Attention and its property.

(a) (1pts) Given queries $\boldsymbol{Q} \in \mathbb{R}^{d \times n}$, $\boldsymbol{K} \in \mathbb{R}^{d \times m}$ and $\boldsymbol{V} \in \mathbb{R}^{t \times m}$, what is the output $\boldsymbol{H}$ of the standard multi-headed scaled dot-product attention? Assume we have h heads.

Solution.

We first split the queries, keys, and values into h separate "heads":

$$\boldsymbol{Q}_1, ..., \boldsymbol{Q}_h = \text{split}(\boldsymbol{Q}), \boldsymbol{K}_1, ..., \boldsymbol{K}_h = \text{split}(\boldsymbol{K}), \boldsymbol{V}_1, ..., \boldsymbol{V}_h = \text{split}(\boldsymbol{V})$$

For each head i, compute the scaled dot-product attention:

$$\boldsymbol{H}_i = \text{softargmax}_\beta(\boldsymbol{Q}_i^T \boldsymbol{K}_i)\boldsymbol{V}_i^T$$

Concatenate the results from all the heads and apply a linear transformation:

$$\boldsymbol{H} = \text{concat}(\boldsymbol{H}_1, ..., \boldsymbol{H}_h)\boldsymbol{W}^O$$

where $\boldsymbol{W}^O \in \mathbb{R}^{hd \times d}$ is a learnable weight matrix. □

(b) (2pts) Is there anything similar to multi-headed attention for convolutional networks? Explain why do you think they are similar. (Hint: read the conv1d document from PyTorch: link)

Solution.

Yes, there is a concept similar to multi-headed attention for convolutional networks: using multiple kernels or filters in a convolutional layer. Just like multi-headed attention, multiple kernels in a CNN capture different aspects of the input data by applying different transformations. The outputs of these kernels are stacked together to form multiple channels in the output feature maps, which is similar to how multi-headed attention concatenates the output of each head. This parallel approach in both architectures allows them to learn diverse and complementary features from the input data, thereby enhancing their representational capacity. □

1.3 Self Attention (11pts)

This question tests your intuitive understanding of Self Attention and its property.

(a) (2pts) Given an input $\boldsymbol{C} \in \mathbb{R}^{e \times n}$, what is the queries $\boldsymbol{Q}$, the keys $\boldsymbol{K}$ and the values $\boldsymbol{V}$ and the output $\boldsymbol{H}$ of the standard multi-headed scaled dot-product self-attention? Assume we have h heads. (You can name and define the weight matrices by yourself)

Solution.

We first apply learnable weight matrices $\boldsymbol{W}^Q, \boldsymbol{W}^K, \boldsymbol{W}^V \in \mathbb{R}^{hd \times e}$ respectively to the input $\boldsymbol{C}$ and splitting the results into h heads:

$$\boldsymbol{Q}_1, ..., \boldsymbol{Q}_h = \text{split}(\boldsymbol{W}^Q\boldsymbol{C}), \boldsymbol{K}_1, ..., \boldsymbol{K}_h = \text{split}(\boldsymbol{W}^K\boldsymbol{C}), \boldsymbol{V}_1, ..., \boldsymbol{V}_h = \text{split}(\boldsymbol{W}^V\boldsymbol{C})$$

For each head i, compute the scaled dot-product attention:

$$\boldsymbol{H}_i = \text{softargmax}_\beta(\boldsymbol{Q}_i^T \boldsymbol{K}_i)\boldsymbol{V}_i^T$$

Concatenate the results from all the heads and apply a linear transformation:

$$\boldsymbol{H} = \text{concat}(\boldsymbol{H}_1, ..., \boldsymbol{H}_h)\boldsymbol{W}^O$$

where $\boldsymbol{W}^O \in \mathbb{R}^{hd \times d}$ is a learnable weight matrix. □

(b) (2pts) Explain when we need the positional encoding for self-attention and when we don't. (You can read about it at link)

Solution.

We need positional encoding for self-attention when the order of the input sequence matters, such as in natural language processing tasks, to help the model differentiate between different word orders. We may not need positional encoding when the order of inputs doesn't significantly impact the task, such as processing high-level features of images extracted by CNNs. However, it's generally recommended to include positional embeddings as they can still provide some useful information. □

(c) (2pts) Show us one situation that the self attention layer behaves like an identity layer or permutation layer.

Solution.

One situation where the self-attention layer behaves like an identity or permutation layer is when the attention weights are set up in such a way that they either maintain the input order or rearrange it. For example, if the attention weights are set such that each token exclusively attends to itself (i.e., the diagonal elements of the attention matrix are 1 and the off-diagonal elements are 0), the self-attention layer would effectively act as an identity layer, passing the input sequence unchanged. Similarly, if the attention weights are set up to focus on specific positions in the input sequence, creating a rearranged output, the self-attention layer would then act as a permutation layer. □

(d) (2pts) Show us one situation that the self attention layer behaves like and "running" linear layer (it applies the linear projection to each location). What is the proper name for a "running" linear layer.

Solution.

One situation where the self-attention layer behaves like a "running" linear layer, applying linear projections to each location, occurs when the attention scores have equal weights for all tokens in the sequence. In this case, the self-attention layer essentially applies a shared weight matrix to all tokens. This behavior is similar to that of a 1D convolutional layer with a kernel size of the sequence length, so a proper name may be a global convolutional layer. □

(e) (3pts) Show us one situation that the self attention layer behaves like an convolution layer with a kernel larger than 1. You can assume we use positional encoding.

Solution.

A self-attention layer can behave like a convolution layer with a kernel larger than 1 when the attention mechanism is set up to focus on a fixed-size local neighborhood around each token. For example, assume we have a sequence of five tokens, and a self-attention mechanism that attends to the immediate neighboring tokens (i.e., a kernel size of 3). The attention matrix would look like:

$$A = [[1, a, 0, 0, 0]\,,\ [b, 1, c, 0, 0]\,,\ [0, d, 1, e, 0]\,,\ [0, 0, f, 1, g]\,,\ [0, 0, 0, h, 1]]$$

Here, a, b, c, d, e, f, g, and h represent attention weights that are non-zero only within the kernel window. This captures local dependencies similarly to a convolution layer. □

1.4 Transformer (15pts)

Read the original paper on the Transformer model: "Attention is All You Need" by Vaswani et al. (2017).

(a) (3pts) Explain the primary differences between the Transformer architecture and previous sequence-to-sequence models (such as RNNs and LSTMs).

Solution.

The primary differences between the Transformer architecture and previous sequence-to-sequence models like RNNs and LSTMs are:

(a) Parallelization: Transformers use self-attention mechanisms, which allow for parallel processing of input tokens, whereas RNNs and LSTMs process tokens sequentially.

(b) Long-range dependencies: Transformers can capture long-range dependencies more effectively due to their global receptive field in self-attention, while RNNs and LSTMs may struggle with long sequences due to vanishing or exploding gradients.

(c) Positional encoding: Transformers require positional encoding to capture the order of tokens in a sequence, as they lack inherent sequential processing. In contrast, RNNs and LSTMs naturally encode sequence order due to their recurrent structure.

□

(b) (3pts) Explain the concept of self-attention and its importance in the Transformer model.

Solution.

Self-attention is a mechanism that enables a model to weigh and relate different tokens in an input sequence, effectively capturing dependencies and contextual information among them.

In the Transformer model, self-attention provides a flexible and expressive way to model long-range dependencies within a sequence. Furthermore, it enables parallel processing of input tokens, resulting in faster computation and enhanced scalability. □

(c) (3pts) Describe the multi-head attention mechanism and its benefits.

Solution.

The multi-head attention mechanism is a component of the Transformer model that consists of several self-attention heads operating in parallel. Each head applies self-attention independently and learns different attention patterns. The outputs of these heads are then concatenated and linearly transformed.

The benefits of multi-head attention include enhanced expressiveness and richer contextual information, as it allows the model to capture diverse and complementary aspects of the input data, focus on different parts of the input, and better understand context and dependencies within the sequence. □

(d) (3pts) Explain the feed-forward neural networks used in the model and their purpose.

Solution.

In the Transformer model, feed-forward neural networks are used as a component within each encoder and decoder layer, following the multi-head attention mechanism. They consist of two linear layers with a non-linear activation function, such as ReLU or GELU, applied in between.

Their purpose is to provide additional non-linear transformations and model complex interactions between the input features. They act as a complementary component to the self-attention mechanism, enhancing the model's ability to learn and represent a wide range of patterns in the input data. □

(e) (3pts) Describe the layer normalization technique and its use in the Transformer architecture.

Solution.

Layer normalization is a technique used to normalize the activations of a neural network layer, by scaling and shifting them to have zero mean and unit variance. It is applied independently to each input feature across the batch, reducing the internal covariate shift and stabilizing the training process.

In the Transformer architecture, layer normalization is used after both the multi-head attention mechanism and the feed-forward neural networks in both encoder and decoder layers. It helps mitigate the vanishing and exploding gradient problems that can arise in deep networks, ensuring more stable and faster training. Additionally, layer normalization in combination with residual connections facilitates the flow of information through the layers, improving the overall performance of the model. □

1.5 Vision Transformer (8pts)

Read the paper on the Transformer model: "An Image is Worth 16 × 16 Words: Transformers for Image Recognition at Scale".

(a) (2pts) What is the key difference between the Vision Transformer (ViT) and traditional convolutional neural networks (CNNs) in terms of handling input images? Can you spot a convolution layer in the ViT architecture?

Solution.

The key difference between the Vision Transformer (ViT) and traditional convolutional neural networks (CNNs) in terms of handling input images lies in the way they process the input. ViT divides the input image into fixed-size non-overlapping patches, linearly embeds them, and then processes them using the Transformer architecture. In contrast, CNNs use convolutional layers to scan input images with overlapping receptive fields, capturing local features and spatial relationships.

In the ViT architecture, only the patch embedding process uses a convolution layer. □

(b) (2pts) Explain the differences between the Vision Transformer and the Transformer introduced in the original paper.

Solution.

The original Transformer is designed for natural language processing tasks, taking sequences of text tokens as input. In contrast, the Vision Transformer is designed for image recognition tasks, taking input images as its data source. The original Transformer consists of an encoder and a decoder, while the ViT only uses the encoder part of the Transformer architecture, as it focuses on image classification rather than sequence-to-sequence tasks. As a result, the ViT doesn't require masks, as the masking mechanism is used in the original Transformer to prevent the decoder from attending to future tokens. □

(c) (2pts) What is the role of positional embeddings in the Vision Transformer model, and how do they differ from positional encodings used in the original Transformer architecture?

Solution.

The role of positional embeddings in the Vision Transformer model is to provide information about the spatial location of each patch in the input image since the self-attention mechanism lacks inherent spatial awareness. They are added to the linearly embedded patches before being fed into the Transformer layers.

The main difference between positional embeddings in ViT and positional encodings in the original Transformer architecture is that positional embeddings in ViT are learnable parameters, whereas positional encodings in the original Transformer are fixed and based on sinusoidal functions. This allows the ViT to learn the positional information relevant to the specific image recognition tasks during training. □

(d) (2pts) How does the Vision Transformer model generate the final classification output? Describe the process and components involved in this step.

Solution.

A special classification token is added to the sequence of linearly embedded patches, and after passing through all Transformer layers, the updated token is extracted. We feed it into the classification head, consisting of a linear layer and a softmax activation, which generates class probabilities. The class with the highest probability is chosen as the final classification output. □