Submission for Homework 4: Transformer

Jay Nitin Kaoshik
jnk7726@nyu.edu
N13143537

CSCI-GA 2572 Deep Learning

Fall 2024

1 Theory (50pt)

1.1 Attention (13pts)

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

(a) (1pts) Given queries $\boldsymbol{Q} \in \mathbb{R}^{d \times n}$, keys $\boldsymbol{K} \in \mathbb{R}^{d \times m}$ and values $\boldsymbol{V} \in \mathbb{R}^{t \times m}$, describe the operations needed to calculate the output $\boldsymbol{H}$ of the standard dot-product attention. What is the output dimension? (You can use the softargmax_β function directly. It is applied to the column of each matrix).

Answer for 1.1 (a):

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

$$H \in \mathbb{R}^{t \times n} \text{ and } (K^T Q) \in \mathbb{R}^{m \times n} \text{ (Attention Matrix)}$$

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

Answer for 1.1 (b):

The parameter β in the attention mechanism is crucial because it controls the "sharpness" or "softness" of the attention distribution. Specifically, β affects how much each element in the softargmax contributes to the final weighted sum. We consider two scenarios below:

- **High β Value**: When β is large, the softargmax function behaves more like an argmax operation, meaning that it will produce a distribution where only a few elements receive significant weight. This leads to a more selective attention mechanism where only the most relevant inputs (i.e., those

with the highest similarity scores) contribute meaningfully to the output. Consequently, the model focuses on a small subset of inputs.

- **Low β Value**: On the other hand, a lower β value smoothens the distribution of the softargmax. This causes the attention mechanism to assign relatively even weights across more elements, creating a more distributed focus. In this case, the model "attends" to a broader set of inputs, leading to a more global attention mechanism.

A convenient value for β is $\frac{1}{\sqrt{d_k}}$, where d_k is dimensionality of key vectors.

(c) (2pts) 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?

Answer for 1.1 (c):

When the attention mechanism's output is intended to preserve a specific value vector v as the output vector h, it generally requires the attention weights to be highly peaked, with a weight of 1 assigned to the desired value vector and 0 to all other values. This situation creates a focused or selective attention effect, allowing only the desired value vector to dominate the output. To achieve this effect, we can set the scaling parameter β to a **large value**. A large β sharpens the softargmax function, making it approach an argmax behavior, where only one value is highly weighted, while others are nearly ignored. This type of attention is known as **hard attention** because it strongly focuses on a single element.

In a fully connected network, a similar effect can be achieved by carefully setting the weights. For example, the **weight** associated with the target input vector can be **set to 1**, while weights for other input vectors are set to 0. This configuration allows the network to preserve the selected input vector as the output. Alternatively, using **activation** functions and **pooling** layers can simulate this effect. For instance, max pooling can isolate the desired vector by selecting the maximum value in each position, while a high β with softmax can similarly create a sharp focus on one vector.

(d) (2pts) 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?

Answer for 1.1 (d):

When the attention output is diffused, the attention mechanism produces a spread-out effect, where multiple value vectors v contribute evenly to the

output h. This effect happens when the scale parameter β is set to a **small value** or approaches zero, which **reduces the "sharpness"** of the attention weights. In this scenario, the attention distribution becomes more uniform, similar to computing a non-weighted average of the input values. If β is exactly zero, the output essentially represents an equal blend of all input vectors v. This type of attention is referred to as **soft attention** because it diffuses focus across multiple elements, rather than concentrating on one.

In a fully connected architecture, a similar diffused output can be achieved by setting **equal weights** across all input vectors. Another approach is to use **activation** and **pooling** layers that **promote averaging**, such as **average pooling**. These operations inherently distribute the weights evenly among inputs. Applying **softargmax** with a **small** β can create this spread-out effect, producing a uniform focus across multiple input vectors.

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

Answer for 1.1 (e):

Assuming the original key vector is k_i, and it undergoes a small perturbation represented as a zero-mean Gaussian noise ϵ, resulting in the modified key $\hat{k}_i = k_i + \epsilon$. After this perturbation, the output H_{new} of the attention mechanism is calculated based on the updated key vector. Since the noise ϵ has a small variance, the impact on the output H_{new} can be thought of as a slight deviation from the original H.

$$H_{new} = \text{softmax}_\beta\left([k_1^T Q, \ldots, (k_i+\epsilon)^T Q, \ldots, k_m^T Q]\right) V$$

$$H_{new} = H + \frac{\partial}{\partial k_i}\left(\text{softmax}_\beta(K^T Q)\right) V \epsilon^T$$

Because the mean of ϵ is zero, this small perturbation in k_i does not significantly alter the overall output H, assuming the noise remains minimal. Hence, the output H will experience only a slight, nearly negligible change due to this small perturbation.

(f) (2pts) If we have a small perturbation to one of the queries $\boldsymbol{q_i}$, how will the output of the $\boldsymbol{H}$ change? How would this differ from the previous case?

Answer for 1.1 (f):

If a small perturbation is added to one of the query vectors q_i, we can denote the modified query as $\hat{q}_i = q_i + \epsilon$, where ϵ is a zero-mean Gaussian noise with small variance. When a query q_i is perturbed, it changes the similarity scores only for that specific query q_i with all keys, which modifies the attention weights associated with that query alone. Consequently, the impact of the perturbation is **localized** to the row in the attention matrix

that corresponds to q_i, affecting only the weighted sum for that particular position in H. This results in a small shift in the values for that particular output position, leaving the other positions relatively unaffected. In the previous case, perturbing a key vector k_i affects all positions in the attention output H because each query interacts with every key. Thus, the perturbation in a key can influence multiple rows in H but in our case a perturbation in query only affects the corresponding position in the output.

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

Answer for 1.1 (g):

If a large perturbation scales one of the key vectors, resulting in $\hat{k} = \alpha k$ for $\alpha > 1$, the output H will be **significantly impacted**. This scaling changes the magnitude of the dot product $Q^T \hat{K}$, which affects the attention scores. With a larger value for $\hat{k}$, the softargmax function will assign a **higher weight** to the corresponding value vector in H. As a result, this key-value pair will have a more prominent influence on the final weighted sum of value vectors in H, potentially altering the output substantially. Such a change can also affect the overall performance of the model, as it disrupts the usual balance of attention across all keys and values.

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 $\boldsymbol{V} \in \mathbb{R}^{t \times m}$, describe the operations for calculating the output $\boldsymbol{H}$ of the standard multi-headed scaled dot-product attention? Assume we have h heads.

Answer for 1.2 (a):

To compute multi-headed scaled dot-product attention with h heads, we first project the input queries Q, keys K, and values V into h distinct subspaces. For each head i, these projections are defined as:

$$Q_i = W_Q^i Q, \quad K_i = W_K^i K, \quad V_i = W_V^i V$$

where W_Q^i, W_K^i, and W_V^i are learned projection matrices specific to each head, reducing the dimensions to $\frac{d}{h}$ for queries and keys, and $\frac{t}{h}$ for values. For each head, we calculate the attention output H_i as follows:

$$H_i = \text{softmax}_\beta \left(K_i^T Q_i \right) V_i$$

This produces h separate outputs $H_1, H_2, \ldots, H_h$, each corresponding to an independent attention head. These outputs are then concatenated along

the depth dimension and transformed back to the original space via a final projection matrix W^0 as follows:

$$H = [H_1, H_2, \ldots, H_h]W^0$$

(b) (2pts) Is there anything similar to multi-headed attention for convolutional networks? Explain why do you think they are similar.

Answer for 1.2 (b):

Yes, there is a similarity between multi-headed attention in transformers and the use of multiple kernels in convolutional neural networks (CNNs). In both cases, the model captures diverse features from the input. In CNNs, multiple kernels (or filters) are applied to the input to detect various spatial patterns, such as edges, textures, and shapes, across different parts of the image. Each kernel extracts a specific type of feature, and the combination of these features gives a comprehensive representation of the input. Similarly, in multi-headed attention, each attention head learns to focus on different aspects of the input by using distinct linear projections for queries, keys, and values. Each head captures unique relationships within the input sequence, and the combined output gives a richer understanding.

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)

Answer for 1.3 (a):

Given the input $C \in \mathbb{R}^{e \times n}$, we compute the queries Q, keys K, and values V for multi-headed scaled dot-product self-attention as follows. With h heads, each head i uses its own projection matrices W_Q^i, W_K^i, and W_V^i to map the input into distinct subspaces:

$$Q_i = CW_Q^i, \quad K_i = CW_K^i, \quad V_i = CW_V^i$$

Each projection maps the input C into a smaller dimensional space. For each head i, the attention output H_i is calculated by applying scaled dot-product attention as follows:

$$H_i = \text{softmax}_\beta\left(K_i^T Q_i\right) V_i$$

Thus, we obtain individual outputs $H_1, H_2, \ldots, H_h$ for each head. We concatenate these outputs along the depth dimension to form a unified output and apply a final projection W^0:

$$H = [H_1, H_2, \ldots, H_h]W^0$$

Thus, H is the multi-headed self-attention output for the given input C.

(b) (2pts) Explain what is positional encoding. What is the difference between absolute and relative positional encoding. When is it appropriate to use absolute positional encoding? When is it more appropriate to use relative encoding?

Answer for 1.3 (b):

Positional Encoding is used to incorporate information about the order or position of tokens in sequence data within a self-attention model like the Transformer. Since self-attention layers don't inherently encode any order information, positional encoding is essential to provide a notion of sequence to the model. **Absolute Positional Encoding** assigns fixed, precomputed values (using sine and cosine functions in original paper by Vaswani et al.) to each position in a sequence, allowing the model to differentiate positions in a straightforward way. In contrast, **Relative Positional Encoding** provides position information in terms of the relationships between positions, allowing the model to compute the same dynamically. Absolute encoding works well for tasks where sequence order matters globally, while relative encoding is more flexible for tasks needing a focus on local relationships.

Suited Tasks for Absolute: Machine Translation, Language Modeling
Suited Tasks for Relative: Document Summarization, Question Answering

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

Answer for 1.3 (c):

If the query matrix Q is equal to the key matrix K and Q is orthogonal, then the attention operation behaves like an identity layer. This occurs because the dot product $K^T Q$ results in the identity matrix I when Q is orthogonal:

$$K^T Q = Q^{-1} Q = I.$$

In this scenario, each token attends only to itself as the diagonal elements of the attention matrix are 1, while all off-diagonal elements are 0. Consequently, the self-attention mechanism outputs the input sequence unchanged, effectively acting as an identity layer.

Additionally, when the attention weights are configured such that they rearrange the input sequence, the self-attention layer behaves as a permutation layer. For example, setting the attention matrix to reflect specific positions within the input sequence can reorder the tokens accordingly.

(d) (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.

Answer for 1.3 (d):

A self-attention layer can act like a convolution layer with a kernel size larger than 1 when it is configured to focus on a fixed-size local neighboring region around each token. This behavior is achieved when the attention mechanism is set to distribute weights across adjacent tokens in the sequence. If the resulting attention matrix from $\text{softargmax}_\beta(K^T Q)$ is structured as a diagonal or **patterned matrix**, it can perform localized projections, akin to a 1D convolution. For example, if the attention matrix has **non-zero entries** arranged to attend to **nearby tokens**, it applies shared weights over these positions. This results in a behavior similar to a convolution operation with overlapping receptive fields, focusing on **local context** and learning dependencies within a fixed window size.

(e) (2pts) Suppose we are training a transformer architecture for real time automatic speech recognition. Do we need to do anything special to the attention mechanism? How do we achieve this?

Answer for 1.3 (e):

To adapt a transformer architecture for real-time automatic speech recognition (ASR), modifications to the attention mechanism are necessary to handle long, noisy input sequences efficiently. One approach is to incorporate a content-based attention mechanism that can score elements based on their contextual relevance, helping to distinguish similar-sounding speech fragments. Additionally, integrating a location-based attention component may be beneficial, where the model learns to align current states with previous outputs and adjust weights accordingly, as seen in handwriting synthesis tasks. These changes ensure that the transformer processes speech inputs effectively and maintains accuracy in generating output transcriptions.
Inspiration: https://proceedings.neurips.cc/paper_files/paper/2015/file/1068c6e4c8051cfd4e9ea8072e3189e2-Paper.pdf

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).

Answer for 1.4 (a):

The Transformer architecture differs from traditional sequence-to-sequence models like RNNs and LSTMs in several key ways. First, Transform-

ers use self-attention mechanisms, allowing for parallel processing of input tokens, while RNNs and LSTMs process inputs sequentially, leading to slower training times. Second, Transformers are adept at capturing long-range dependencies due to their global receptive field provided by self-attention, whereas RNNs and LSTMs often face challenges with vanishing or exploding gradients in long sequences. Lastly, while RNNs and LSTMs inherently encode the order of tokens through their recurrent nature, Transformers rely on different types of positional encodings to incorporate sequence order, enabling them to efficiently handle larger inputs while maintaining token position awareness at all times during training.

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

Answer for 1.4 (b):

Self-attention is an attention mechanism that enables a model to relate different positions within an input sequence to create a representation that captures dependencies among tokens. Unlike recurrent architectures, self-attention can process all input tokens simultaneously, which allows for parallelized computation. This parallelism significantly reduces computational complexity and leads to faster processing compared to sequential models like RNNs and LSTMs. Additionally, self-attention helps in effectively capturing long-range dependencies by enabling the model to focus on relevant tokens across the sequence, enhancing the learning of contextual relationships. This characteristic is particularly crucial for scaling to large datasets and complex tasks in NLP and other related domains.

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

Answer for 1.4 (c):

Multi-head attention is a mechanism used in the Transformer model that involves applying self-attention multiple times in parallel, with each instance referred to as a head. Each head independently learns unique attention patterns by projecting the input into different subspaces. After processing, the outputs of all the heads are concatenated and passed through a final weighted linear transformation. This allows the model to capture diverse contextual relationships in the data (similar to multiple kernels in CNNs!), enhance its expressiveness, and represent different parts of the input sequence simultaneously, thus enriching the overall learning process.

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

Answer for 1.4 (d):

Feed-forward networks in the Transformer architecture are position-wise and operate independently at each input location. Each feed-forward network consists of two linear transformations separated by a non-linear activation function such as ReLU/GELU. This structure allows the model to

apply unique transformations at each position and utilize different parameters across the layers at the same position. The purpose of these networks is to add non-linear transformations and capture complex patterns, enriching the model's representational power and complementing the self-attention mechanism for better learning and diverse pattern recognition.

(e) (3pts) Name two techniques used in the paper to improve training stability of the transformer model, in particular regards to the issue of exploding / vanishing gradients. And briefly explain how they do so.

Answer for 1.4 (e):

- **Layer Normalization**: The paper employs layer normalization after the addition of the residual connection in each sub-layer of the Transformer. This helps to stabilize the training by normalizing the outputs of the layers, thus maintaining mean and variance. Layer normalization reduces the risk of exploding or vanishing gradients by ensuring that the network maintains a stable distribution of activations and reduced internal covariate shift, which is crucial for deep networks.
- **Residual Connections**: The use of residual connections (or skip connections) helps to combat vanishing gradients by allowing gradients to flow directly through the network layers without being diminished. This technique combined with Layer Normalization helps the model maintain gradient flow even as the depth of the network increases, improving the stability and convergence rate of training.

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?

Answer for 1.5 (a):

The main distinction between Vision Transformers (ViT) and traditional CNNs lies in their approach to handling input images. ViT divides images into **fixed-size patches**, which are then linearly embedded and processed using **self-attention** mechanisms akin to those used in NLP tasks. This approach allows ViT to treat image patches as individual tokens for attention-based analysis. In contrast, CNNs leverage convolutional layers to progressively scan images with overlapping receptive fields, effectively capturing spatial hierarchies and local dependencies. While CNNs excel at capturing local spatial features, ViTs excel at modeling **global relationships** across the image. In the ViT architecture, a convolutional layer is typically employed during the patch embedding phase (also done in Lab).

(b) (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?

Answer for 1.5 (b):

The positional embeddings in the Vision Transformer (ViT) play a critical role in encoding the spatial positions of image patches to retain information about their locations, as the self-attention mechanism itself lacks any inherent understanding of spatial context. These embeddings are added to the input patch representations before they are processed by the Transformer layers, enabling the model to interpret the spatial structure of images effectively. The main distinction between the positional embeddings used in ViT and those in the original Transformer model is that ViT embeddings are learnable, allowing them to adapt to image-specific tasks. In contrast, the original Transformer architecture typically employs fixed, sinusoidal-based positional encodings that remain static. This learnable nature in ViT facilitates better spatial feature learning during training.

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

Answer for 1.5 (c):

In the Vision Transformer (ViT) model, a special classification token is appended to the sequence of linearly embedded image patches. This classification token, along with the embedded patches, is passed through the Transformer layers. The updated classification token, which contains the output information from all the self-attention operations, is then extracted and sent to a classification head. The classification head typically comprises of a linear layer followed by a softmax activation, which calculates class probabilities. Like any other task, we take the max probability class.

(d) (2pts) How does ViT compare with CNN in terms of performance across different data regimes? What explains this trend?

Answer for 1.5 (d):

The Vision Transformer (ViT) generally outperforms Convolutional Neural Networks (CNNs) when trained on **large-scale datasets** due to its ability to model **long-range dependencies** more effectively using the self-attention mechanism. This capacity allows ViT to capture **global context in the input** data, which becomes beneficial with ample training data to learn meaningful representations.

However, in **data regimes** with **limited samples**, CNNs often demonstrate superior performance compared to ViT. This trend can be attributed to CNNs' **inherent inductive biases**, such as locality and translation invariance, which help them generalize better with fewer data. In contrast, ViT lacks these biases, making it more reliant on larger training datasets or pre-training to perform at its best.

2 Implementation (50pt)

Best Validation Accuracy Achieved: 67.96%
Epoch Index for Best Accuracy: 90