Homework 4: Transformer

Deep Learning

Fall 2024

1 Theory (50pt)

1.1 Attention (13pts)

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

(a) (1pts)

- Dot product between queries and keys:

$$S = K^\top Q \quad \text{where } S \in \mathbb{R}^{m \times n} \text{ because } Q \in \mathbb{R}^{d \times n} \text{ and } K \in \mathbb{R}^{d \times m}$$

- Attention scores calculation,

$$\alpha = \text{softargmax}_\beta(S) \text{ where } \alpha \in \mathbb{R}^{m \times n}$$

- Final output

$$H = V\alpha \quad \text{where } H \in \mathbb{R}^{t \times n} \text{ because } \alpha \in \mathbb{R}^{m \times n} \text{ and } V \in \mathbb{R}^{t \times m}$$

Output dimension: The output H has dimensions $\mathbb{R}^{n \times t}$.

(b) (2pts)

The scale β controls the sharpness of the attention distribution:

- **Large** β: Leads to a sharper, more peaked attention distribution, focusing on specific keys.
- **Small** β: Results in a flatter, more uniform attention distribution over keys.

Convenient value for β: Using $\beta = \frac{1}{\sqrt{d}}$ (where d is the dimension of the key vectors) balances the scale of the dot products and prevents extremely large or small gradients during training as suggested by the paper.

(c) (2pts)

Preserving a value vector v in the output h:

- When the attention weights become one-hot vectors (i.e., the model attends entirely to one key-value pair).
- Setting β to a large value ($\beta \to \infty$) makes the softargmax function approach the argmax, leading to hard attention.
- This refers to hard attention, where the model deterministically selects a single key-value pair.
- Incorporate mechanisms like gating or residual connections that allow the network to preserve specific value vectors directly through the layers.

(d) (2pts)

Diluting value vectors to generate new output $\boldsymbol{h}$:

- When the attention weights are spread out uniformly, effectively averaging the value vectors.
- Setting β to a small value ($\beta \to 0$) flattens the attention distribution.
- This refers to soft attention, where the model distributes focus over multiple key-value pairs.
- Designing the network so that it produces similar attention scores for all keys or includes mechanisms that promote uniform weighting, such as regularization techniques.

(e) (2pts)

A small perturbation ϵ to k_i affects the attention scores between k_i and all queries $\boldsymbol{Q}$:

- Slightly alters $S_{ji} = (k_i + \epsilon)^\top q_j$.
- Leads to minor changes in the attention weights α_{ji} for all queries q_j.
- The perturbation affects all columns of H corresponding to all queries, but the effect is localized to the contributions from the perturbed key k_j.

(f) (2pts)

Perturbing q_i influences only the computations for that specific query:

- Alters $S_{ij} = (q_i + \epsilon)^\top k_j$ for all keys K.
- Affects only α_{ij}, the attention weights for query q_i.
- Only the output h_i corresponding to q_i changes, as it adjusts how q_i attends to the values V.

- Unlike perturbing k_i, which affects whole output H, perturbing q_i affects only its own output.

(g) (2pts)

- Scaling k_i by α amplifies the dot products $(\alpha k_i)^\top q_j$, boosting the attention scores involving k.
- After applying softargmax, the attention weights for k become significantly larger.
- As α becomes large, A_{ji} approaches 1 for key k_i, and A_{jk} for $k \neq i$ approach 0.
- Scaling one key k_i by a large factor α causes the attention mechanism to focus almost entirely on k_i.
- The output H changes significantly, with all outputs h_j being heavily influenced by the corresponding value v_i.

1.2 Multi-headed Attention (3pts)

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

(a) (1pts)

$Q \in \mathbb{R}^{d \times n}$ $K \in \mathbb{R}^{d \times m}$ $V \in \mathbb{R}^{t \times m}$ Number of heads h

(a) Linear projections for each head:

For each head i (where $i = 1, 2, \ldots, h$), we project the queries, keys, and values using learned linear transformations:

- $W_Q^i \in \mathbb{R}^{d_k \times d}$
- $W_K^i \in \mathbb{R}^{d_k \times d}$
- $W_V^i \in \mathbb{R}^{d_v \times t}$

Here $d_k = \frac{d}{h}$ and $d_v = \frac{t}{h}$

Then the projections becomes,

$$Q^i = W_Q^i Q \quad (\in \mathbb{R}^{d_k \times n})$$
$$K^i = W_K^i K \quad (\in \mathbb{R}^{d_k \times m})$$
$$V^i = W_V^i V \quad (\in \mathbb{R}^{d_v \times m})$$

(b) Dot-product attention for each head:

For each head i:

- Attention scores:

$$S^i = (K^i)^\top Q^i \quad (\in \mathbb{R}^{m \times n})$$

- Softargmax function:

$$\alpha^i = \text{softargmax}_\beta(S^i) \quad (\in \mathbb{R}^{m \times n})$$

- Head's output:

$$H^i = V^i \alpha^i \quad (\in \mathbb{R}^{d_v \times n})$$

(c) Concatenate the outputs of all heads:

$$H_{\text{concat}} = [H^1, H^2, \ldots, H^h] \quad (\in \mathbb{R}^{hd_v \times n})$$

- The outputs H^i are concatenated along the feature dimension.

(d) Final linear projection:

Apply a linear transformation to combine the information from all heads:

$$H = W_O H_{\text{concat}} \quad (\in \mathbb{R}^{t \times n})$$

- $W_O \in \mathbb{R}^{t \times hd_v}$
- The final output H has dimensions $\mathbb{R}^{t \times n}$.

(b) (2pts)

(a) Parallel Processing of Features:

- Multi-headed Attention: Each head in multi-headed attention processes the input data in parallel but focuses on different aspects or representations of the data. Each head learns to attend to different positions or features in the input sequences.
- CNNs with Multiple Filters: In CNNs, multiple convolutional filters (kernels) operate in parallel across the input data. Each filter detects specific features or patterns (like edges, textures) in different spatial locations.

(b) Learning Diverse Representations:

- Multi-headed Attention: By using multiple heads, the model can capture diverse relationships and interactions between elements in the input sequences, enhancing the richness of the learned representations.
- CNNs: Multiple filters allow the network to learn a variety of feature detectors, enabling the model to represent complex patterns in the data.

(c) Weight Sharing :

- Multi-headed Attention: Attention mechanisms share weights across positions in the sense that the same projection matrices are applied to all positions in the sequence.

- CNNs: Convolutional layers use weight sharing (the same filter is applied across different spatial locations), exploiting the local spatial coherence in images.

(d) Aggregation of Information:

- Multi-headed Attention: Aggregates information from different positions weighted by attention scores, allowing the model to focus on relevant parts of the input when generating each output element.
- CNNs: Pooling layers aggregate information from local neighborhoods, summarizing features in spatial regions to create a higher-level representation.

1.3 Self Attention (11pts)

(a) (2pts)

Assuming e to be feature dimension and n to ne sequence length.

- Input $C \in \mathbb{R}^{e \times n}$.
- Number of heads h.

For each head i (where $i = 1, 2, \ldots, h$), we have projection matrices:

- $W_Q^i \in \mathbb{R}^{d_k \times e}$ for queries.
- $W_K^i \in \mathbb{R}^{d_k \times e}$ for keys.
- $W_V^i \in \mathbb{R}^{d_v \times e}$ for values.

$d_k = d_v = \frac{e}{h}$

For each head i:

- $Q^i = W_Q^i C \quad (\in \mathbb{R}^{d_k \times n})$
- $K^i = W_K^i C \quad (\in \mathbb{R}^{d_k \times n})$
- $V^i = W_V^i C \quad (\in \mathbb{R}^{d_v \times n})$

For each head i:

- Attention scores:
$$S^i = (K^i)^\top Q^i \quad (\in \mathbb{R}^{n \times n})$$
- Softargmax function:
$$\alpha^i = \text{softargmax}_\beta(S^i) \quad (\in \mathbb{R}^{n \times n})$$
- Head's output:
$$H^i = (V^i)\alpha^i \quad (\in \mathbb{R}^{d_v \times n})$$

Concatenate the outputs of all heads:

$$H_{\text{concat}} = [H^1, H^2, \ldots, H^h] \quad (\in \mathbb{R}^{hd_v \times n})$$

- The outputs H^i are concatenated along the feature dimension.

Final linear projection:

- Apply a final linear transformation to integrate the information from all heads:

$$H = W_O H_{\text{concat}} \quad (\in \mathbb{R}^{e \times n})$$

- $W_O \in \mathbb{R}^{e \times (h \cdot d_v)}$ is the output projection matrix.
- Final output has dimensions of $\mathbb{R}^{e \times n}$.

(b) (2pts)

Positional Encoding:

- Positional encoding is a technique used to inject information about the position of elements in a sequence into the model.
- Since the self-attention mechanism is permutation-invariant (it doesn't inherently encode the order of elements), positional encoding ensures the model can utilize the sequence order.

Absolute Positional Encoding:

- Assigns a unique encoding to each position in the sequence based on its absolute position.
- Common methods include sinusoidal functions.
- The positional encodings are added to the input embeddings before passing them to the model.
- When the absolute position of elements is crucial, such as in tasks where the position conveys specific meaning (e.g., language modeling, where word order matters).

Relative Positional Encoding:

- Represents the relative distances between sequence elements rather than their absolute positions.
- The model learns or computes embeddings based on the relative position between pairs of elements.
- When the relationship between elements depends on their relative positions, such as when learning for image classification we care about the relative position among the patches.

(c) (2pts)

Identity Layer Behavior:

- If the self-attention mechanism assigns all the attention weight to the element itself at each position.
- The queries and keys are designed such that $q_i^\top k_j$ is a large value when $i = j$ and very negative when $i \neq j$.
- This results in the softargmax producing attention weights that are one-hot vectors, where each element attends only to itself.
- The output at each position is effectively the value at that position:

$$H = V$$

- Since V is a linear projection of C, if the projection matrices are identity matrices ($W_V^i = I$), then:

$$H = C$$

- The self-attention layer behaves like an identity layer, passing the input directly to the output.

Permutation Layer Behavior:

- If the attention weights represent a permutation matrix, where each row contains a single 1 and zeros elsewhere, but not necessarily on the diagonal.
- The model learns attention patterns that rearrange the sequence elements.
- The queries and keys are such that each position attends to a specific different position.
- The output sequence is a permuted version of the input sequence.
- This can be useful for tasks that require reordering or shuffling elements based on learned patterns.

(d) (3pts)

We desire the self-attention mechanism which is configured to attend primarily to neighboring positions within a fixed window size, similar to a convolutional kernel.

- Positional Encoding:
  - Incorporate positional encodings that emphasize local neighborhoods.
  - For example, use relative positional encoding to bias the attention towards nearby positions.
- Masking Mechanism:

- Apply a mask to the attention scores to limit the attention to a specific range around each position.
- The mask sets the attention scores outside the desired window to a large negative value before the softmax, effectively zeroing out those positions.

- Each output position h_i becomes a weighted sum of the values within its local neighborhood:

$$h_i = \sum_{j=i-k}^{i+k} v_j \alpha_{ij}$$

where k is the window size (kernel size), and α_{ij} are the attention weights.

- This operation is analogous to a convolution with a kernel size of $2k + 1$, capturing local patterns and dependencies.

(e) (2pts)

Special Considerations for the Attention Mechanism:

- Causal (Unidirectional) Attention:
  - In real-time ASR, the model should not use future information to predict the current output, as it processes streaming input data.
  - The attention mechanism must be modified so that each position can only attend to previous positions, not future ones.

- Causal Mask:
  - Introduce a mask to the attention scores that prevents the model from attending to future positions.
  - The mask sets the attention scores s_{ij} to a large negative value (e.g., $-\infty$) when $j > i$, ensuring the softmax assigns zero weight to those positions.

(a) Attention scores:

$$s_{ij} = \frac{k_j^\top q_i}{\sqrt{d_k}}$$

(b) Apply the causal mask:

$$s_{ij} = \begin{cases} s_{ij} & \text{if } j \le i \\ -\infty & \text{if } j > i \end{cases}$$

1.4 Transformer (15pts)

(a) (3pts)

- Transformers remove the need for recurrent connections. While RNNs and LSTMs process sequences sequentially, the Transformer processes the entire sequence simultaneously using attention. This allows for greater parallelization during training and inference.
- Transformers rely entirely on attention to model dependencies between input and output sequences. This contrasts with RNNs and LSTMs, which use hidden states to capture sequential information.
- RNNs and LSTMs can struggle with long-range dependencies due to issues like vanishing or exploding gradients. The Transformer's self-attention mechanism can directly connect distant positions in the sequence, making it more effective at capturing global dependencies.
- Because Transformers can process all positions in a sequence in parallel, resulting in significantly faster training times compared to sequential models like RNNs and LSTMs.
- Since Transformers lack the sequential nature of RNNs, they incorporate positional encoding to retain information about the order of the sequence. This is done through adding sinusoidal positional encodings to the input embeddings.

(b) (3pts)

Self-attention, also known as intra-attention, is a mechanism that allows each position in a sequence to attend to all other positions in the sequence:

- In self-attention, for each position in the input sequence, the model computes a weighted sum of the representations of all positions, where the weights are determined by the similarity between the representations.
- Importance in Transformers:
  - Self-attention enables the model to capture dependencies between words regardless of their distance in the sequence, which is particularly useful for modeling long-range relationships.
  - Since self-attention does not rely on previous time steps (unlike recurrence), it allows for parallel computation over all positions in the sequence, leading to more efficient training.
  - It provides a flexible way to model context, as the attention weights can focus on relevant parts of the sequence dynamically.
- Role in Transformer Architecture: Self-attention is the core component of the Transformer's encoder and decoder layers. It replaces the need for recurrent connections by allowing the model to directly attend to relevant information at any position in the input or output sequences.

(c) (3pts)

Multi-head attention extends the self-attention mechanism by allowing the model to focus on different positions and representation subspaces simultaneously:

- Mechanism:
  - Multiple Attention Heads: Instead of performing a single attention function, the model computes multiple attention functions (heads) in parallel.
  - Learned Linear Projections: For each head, the queries, keys, and values are linearly projected into different subspaces.
  - Parallel Attention Computation: Each head performs attention independently, capturing different aspects of the input.
  - Concatenation and Projection: The outputs of all heads are concatenated and projected to form the final output.
- Benefits:
  - By attending to information from different representation subspaces, multi-head attention allows the model to capture a variety of relationships and patterns in the data.
  - It enhances the model's ability to focus on relevant parts of the input for different tasks, leading to better overall performance.
  - Multiple heads prevent the attention mechanism from averaging out important details, which can happen with a single attention head.

(d) (3pts)

In the Transformer architecture, positison-wise feed-forward neural networks are applied after the self-attention mechanisms in both the encoder and decoder layers:

- Mechanism:
  - Each feed-forward network consists of two linear transformations with a ReLU activation function in between.
  - The same feed-forward network is applied independently and identically to each position in the sequence.
- Purpose:
  - The feed-forward networks introduce non-linearities and allow for complex transformations of the input representations, enabling the model to learn more intricate patterns.
  - They help in extracting higher-level features from the attention outputs, contributing to the model's depth and capacity.
  - Since they are applied to each position separately, they maintain the model's ability to process sequences in parallel.

(e) (3pts)

The Transformer model employs certain techniques to address the issues of exploding and vanishing gradients, enhancing training stability:

(a) Residual Connections:

- Residual connections involve adding the input of a layer to its output (`LayerNorm(x + Sublayer(x))`).
- Why they work:
  - They facilitate the flow of gradients through the network by providing shortcut paths, which helps in mitigating the vanishing gradient problem.
  - By allowing the layers to learn modifications to the identity function, they make it easier to optimize deeper networks.

(b) Layer Normalization:

- Layer normalization normalizes the inputs across the features for each training case, stabilizing the distributions of layer inputs.
- Why they work:
  - It reduces the internal covariate shift, which stabilizes the training process and allows for higher learning rates.
  - By normalizing activations, it helps maintain consistent gradient magnitudes, addressing both exploding and vanishing gradients.

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)

The primary difference between the Vision Transformer (ViT) and traditional Convolutional Neural Networks (CNNs) lies in how they process input images:

- Patch-Based Input Processing:
  - ViT: Divides the input image into a sequence of fixed-size patches, flattens them, and linearly projects each patch to create embeddings. These embeddings are then treated as input tokens for the Transformer model.
  - CNNs: Process the entire image using convolutional layers that apply filters across spatial dimensions to capture local patterns.
- Lack of Convolutional Layers:

  - ViT: Does not use convolutional layers in its pure form. Instead, it relies entirely on the Transformer architecture, which is based on self-attention mechanisms, to model relationships between patches.
  - CNNs: Utilize convolutional layers throughout the architecture to exploit spatial locality and translation invariance inherent in images.
- Inductive Biases:
  - ViT: Has fewer image-specific inductive biases (like locality and translation equivariance), relying on large datasets to learn these properties.
  - CNNs: Built-in inductive biases help them generalize well even on smaller datasets.

While ViT does not use convolution layer for learning relation between patches, but it is used for projecting raw image patches into path embeddings.

(b) (2pts)

- Since the Transformer architecture lacks inherent mechanisms to capture the order or spatial relationships between input tokens, positional embeddings are added to the patch embeddings to retain information about the position of each patch within the image.
- In ViT, positional embeddings are learnable vectors that are added to the corresponding patch embeddings before feeding them into the Transformer encoder.

Differences

- ViT Positional Embeddings:
  - Learnable 1D Embeddings: ViT uses learnable positional embeddings that are optimized during training.
  - Image-Specific Considerations: While the embeddings are 1D, they represent 2D spatial positions when images are flattened into sequences.
- Original Transformer Positional Encodings:
  - Fixed Sinusoidal Functions: The original Transformer uses sinusoidal positional encodings that are not learned but are fixed functions of the token positions.
  - Mathematical Formulation: These encodings use sine and cosine functions of different frequencies to provide a unique positional representation.

(c) (2pts)

(a) Input Preparation:

- Patch Embeddings: The input image is divided into fixed-size patches, each of which is flattened and linearly projected to form patch embeddings.
- `[CLS]` Token: A special learnable classification token (`[CLS]`) is prepended to the sequence of patch embeddings. This token does not correspond to any image patch but serves as an aggregate representation.

(b) Positional Embeddings:

- Addition of Positional Information: Learnable positional embeddings are added to the patch embeddings and the `[CLS]` token embedding to retain positional information.

(c) Transformer Encoder:

- Processing the Sequence: The combined sequence is fed into the Transformer encoder, which consists of multiple layers of multi-head self-attention and feed-forward neural networks.
- Self-Attention Mechanism: Allows the model to compute contextualized representations of each token (including the `[CLS]` token) by attending to all tokens in the sequence.

(d) Extracting the Image Representation:

- `[CLS]` Token Output: After passing through the Transformer encoder, the output corresponding to the `[CLS]` token ($z_L^{[0]}$) contains the aggregated information of the entire image.
- Normalization: Layer normalization is often applied to the `[CLS]` token output.

(e) Classification Head:

- MLP Head: The normalized `[CLS]` token representation is fed into a classification head, typically a Multi-Layer Perceptron (MLP), which may consist of one or more fully connected layers.
- Final Output: The classification head outputs the logits for each class, which are then used to compute probabilities (e.g., via soft-argmax) and make the final prediction.

(d) (2pts)

- Small Datasets:
  - CNNs Outperform ViT: When trained on smaller datasets (e.g., ImageNet with 1.3 million images), CNNs generally perform better than ViT models.
  - CNNs have strong inductive biases like locality and translation invariance, which help them generalize well from limited data.
- Large Datasets:

  - ViT Outperforms CNNs: When trained on large-scale datasets (e.g., ImageNet-21k with 14 million images or JFT-300M with 300 million images), ViT models outperform CNNs.
  - With sufficient data, ViT can learn the necessary spatial relationships and patterns without relying on the inductive biases inherent in CNNs.

- Inductive Biases:
  - CNNs: Built-in inductive biases allow CNNs to capture local patterns and spatial hierarchies effectively, leading to better performance on smaller datasets.
  - ViT: Lacks these biases and relies on learning spatial relationships from data, which requires larger datasets to achieve comparable or superior performance.
- Capacity to Learn Representations:
  - ViT: With large datasets, ViT can learn rich representations that capture global context, leading to better generalization and higher accuracy.
  - CNNs: May saturate in performance even with more data, as their inductive biases can limit the diversity of patterns they can learn.

2 Implementation (50pt)

Please add your solutions to this notebook `HW3-VIT-Student.ipynb` . **Plase use your NYU account to access the notebook.** The notebook contains parts marked as `TODO`, where you should put your code or explanations. The notebook is a Google Colab notebook, you should copy it to your drive, add your solutions, and then download and submit it to NYU Classes. You're also free to run it on any other machine, as long as the version you send us can be run on Google Colab.