Chapter 12: Transformers

Problem 12.1

Consider a self-attention mechanism that processes N inputs of length D to produce N outputs of the same size. How many weights and biases are used to compute the queries, keys, and values? How many attention weights a$[\bullet, \bullet]$ will there be? How many weights and biases would there be in a fully connected network relating all DN inputs to all DN outputs?

The self attention mechanism is defined in Equation (1).

$$\text{Sa}[\mathbf{X}] = \mathbf{V} \cdot \textbf{Softmax}[\mathbf{K}^T\mathbf{Q}] \tag{1}$$

Where the values $\mathbf{V}$, keys $\mathbf{K}$, and queries $\mathbf{Q}$ are computed as:

$$\mathbf{V}[\mathbf{X}] = \boldsymbol{\beta}_v\mathbf{1}^T + \boldsymbol{\Omega}_v\mathbf{X} \tag{2}$$
$$\mathbf{Q}[\mathbf{X}] = \boldsymbol{\beta}_q\mathbf{1}^T + \boldsymbol{\Omega}_q\mathbf{X} \tag{3}$$
$$\mathbf{K}[\mathbf{X}] = \boldsymbol{\beta}_k\mathbf{1}^T + \boldsymbol{\Omega}_k\mathbf{X} \tag{4}$$

$\mathbf{X}$ is a $D \times N$ matrix (i.e., N inputs of length D). Therefore the biases $\boldsymbol{\beta}_v$, $\boldsymbol{\beta}_q$, and $\boldsymbol{\beta}_k$ are of size $D \times 1$ and the weights $\boldsymbol{\Omega}_v$, $\boldsymbol{\Omega}_q$, and $\boldsymbol{\Omega}_k$ are of size $D \times D$. The number of weights and biases used to compute the queries, keys, and values are:

$$\text{Weights} = 3 \times D^2 \quad \text{Biases} = 3 \tag{5}$$

The dimensions of the values $\mathbf{V}[\mathbf{X}]$, queries $\mathbf{Q}[\mathbf{X}]$, and keys $\mathbf{K}[\mathbf{X}]$ are $D \times N$. Therefore, the attention weights a$[\bullet, \bullet]$ will be of size $N \times N$. The number of attention weights is:

$$\text{Attention Weights} = N^2 \tag{6}$$

Fully connected network: In a fully connected network relating all DN inputs to all DN outputs, the number of weights and biases would be:

$$\text{Weights} = D^2N^2 \quad \text{Biases} = N^2 \tag{7}$$

Problem 12.2

Why might we want to ensure that the input to the self-attention mechanism is the same size as the output?

Keeping the input and output dimensions the same allows for multiple layers of self-attention to be stacked without requiring additional transformation layers to match dimensions between them (e.g. in multiple head self-attention mechanisms). Additionally, keeping the input and output dimensions the same simplifies the network design. Each layer can be treated as a modular block that can be easily understood, implemented, and debugged

Problem 12.3

Show that the self-attention mechanism is equivariant to a permutation $\mathbf{XP}$ of the data $\mathbf{X}$, where $\mathbf{P}$ is a permutation matrix. In other words, show that:

$$\text{Sa}[\mathbf{XP}] = \text{Sa}[\mathbf{X}]\mathbf{P}$$

Since the linear projections of the keys, values, and queries are applied independently to each row of $\mathbf{X}$, permuting the rows of $\mathbf{X}$ will correspond to permuting the rows of $\mathbf{K}$, $\mathbf{V}$, and $\mathbf{Q}$:

$$\mathbf{K}[\mathbf{XP}] = \mathbf{KP}[\mathbf{X}] \tag{8}$$
$$\mathbf{V}[\mathbf{XP}] = \mathbf{VP}[\mathbf{X}] \tag{9}$$
$$\mathbf{Q}[\mathbf{XP}] = \mathbf{QP}[\mathbf{X}] \tag{10}$$

Substituting these into the self-attention mechanism in Equation (1):

$$\text{Sa}[\mathbf{XP}] = \mathbf{VP} \cdot \mathbf{Softmax}[\mathbf{KP}^T\mathbf{QP}] \tag{11}$$

Since $\mathbf{P}$ is a permutation matrix, its transpose is also its inverse, therefore $\mathbf{P}^T\mathbf{P} = \mathbf{I}$. Equation (11) can be rewritten as:

$$\text{Sa}[\mathbf{XP}] = \mathbf{VP} \cdot \mathbf{Softmax}[\mathbf{K}^T\mathbf{Q}] \tag{12}$$

Which is equivalent to $\text{Sa}[\mathbf{X}]\mathbf{P}$.

Problem 12.4

Consider the softmax operation:

$$y_i = \text{softmax}_i[\mathbf{z}] = \frac{\exp(z_i)}{\sum_{j=1}^{5}\exp(z_j)}$$

in the case where there are five inputs with values $z_1 = -3$, $z_2 = 1$, $z_3 = 100$, $z_4 = 5$, and $z_5 = -1$. Compute the 25 derivatives $\partial y_i/\partial z_j$. for all $i, j \in \{1, 2, 3, 4, 5\}$. What do you conclude?

When $i = j$, the derivative $\partial y_i/\partial z_j$ is given by:

$$\frac{\partial y_i}{\partial z_i} = \frac{\exp(z_i)\sum_{j=1}^{5}\exp(z_j) - \exp(z_i)\exp(z_i)}{\left(\sum_{j=1}^{5}\exp(z_j)\right)^2} = y_i(1 - y_i) \tag{13}$$

When $i \neq j$, the derivative $\partial y_i/\partial z_j$ is given by:

$$\frac{\partial y_i}{\partial z_j} = -\frac{\exp(z_i)\exp(z_j)}{\left(\sum_{j=1}^{5}\exp(z_j)\right)^2} = -y_i y_j \tag{14}$$

The code for calculating these derivatives is shown in Code block 1.

Code Block 1: Calculate 25 derivates

```
inputs = [z1, z2, z3, z4, z5]

for i in range(5):
    for j in range(5):
        if i == j:
            di_dj = softmax_i(inputs[i], inputs) *
            (1 - softmax_i(inputs[i], inputs))
            print(f"dy_{i+1}/dz_{j+1} = {di_dj}")
        else:
            di_dj = -softmax_i(inputs[i], inputs) *
            softmax_i(inputs[j], inputs)
            print(f"dy_{i+1}/dz_{j+1} = {di_dj}")
```

All of the calculated derivatives are very close to zero. This is due to to $\exp(z_3)$ being much larger than the other values. The softmax function is therefore very sensitive to the input values, and small changes in the input can lead to large changes in the output. This problem highlights the case of the softmax function "saturating" where one term dominates and others are near zero.

Problem 12.5

Why is the implementation more efficient if the values, queries, and keys in each of the H heads each have dimensions D/H where D is the original dimension of the data?

In the multiple head self-attention mechanism, reducing the dimensionality of the values, keys, and queries from D to D/H where H is the number of heads, reduces both the computational cost of the mechanism and the memory required to store these matrices. he cost of computing the dot products in self-attention for each head is proportional to D/H rather than D, reducing the complexity per head from $O(D^2)$ to $O((D/H)^2)$. This reduction in complexity allows for more heads to be used without increasing the computational cost. The overall complexity across all heads remains $O((D)^2)$ but the reduction in per-head computation allows more efficient use of parallel architectures.

Additionally, the reduced dimensionality of the values, keys, and queries allows for more heads to be used without increasing the memory requirements, as total number of parameters across all heads remains the same ($H \times (D/H)^2 = D^2/H$), but spreading them across multiple heads means that the model can manage its memory use more flexibly and efficiently.

Problem 12.6

BERT was pre-trained using two tasks. The first task requires the system to predict missing (masked) words. The second task requires the system to classify pairs of sentences as being adjacent or not in the original text. Identify whether each of these tasks is generative or contrastive (see section 9.3.6). Why do you think they used two tasks? Propose two novel contrastive tasks that could be used to pre-train a language model.

BERT is an encoder model that uses a vocabulary of $30,000$ tokens. The first task during pre-training, which requires the model to predict the missing (masked) words is an example of a *generative* task:The model is required to generate the most likely token that has been masked out. The second task, which requires the model to classify pairs of sentences as being adjacent or not in the original text is an example of a *contrastive* task: The model is required to contrast the similarity between pairs of sentences.

Two examples of a contrastive task that could be used to pre-train a language model are:

1. **Inverted sentence detection**: Given two sentences, e.g. "The boy ran quickly" and "Quickly ran the boy", the model needs to identify if the second sentence is a syntactic inversion of the first.
2. **Logical inference verification**: Given two sentences, e.g. "John bought a new laptop" and "John has a new laptop", the model needs to determine if the second sentence is a logical inference of the first.

Pre-training language models on a combination of generative and contrastive tasks can lead to more comprehensive language understanding, preparing the model for complex language processing tasks that require both generation and fine-grained discrimination capabilities.

Problem 12.7

Consider adding a new token to a precomputed masked self-attention mechanism with N tokens. Describe the extra computation that must be done to incorporate this new token.

In the self-attention mechanism, three matrices are computed for every input token: the keys, queries, and values. Therefore, adding a new token would require computing the keys, queries, and values for the new token. Although the existing weights can be used for the computation of these new matrices.

The attention weights would then be computed as dot products of the new token's keys and all other tokens' queries, i.e., The size of the attention matrix increases from $N \times N$ to $(N+1) \times (N+1)$.

In conclusion, adding a new token requires extending the self-attention calculations to account for interactions between this new token and all existing tokens, updating the softmax-normalised attention weights, and recalculating the output vectors.

Problem 12.8

Computation in vision transformers expands quadratically with the number of patches. Devise two methods to reduce the computation using the principles from figure 12.15.

1. **Local attention mechanism**: Instead of computing attention between all pairs of patches (i.e., figure 12.15a), only compute attention between patches that are close to each other. This can be achieved by introducing a convolutional structure similar to what's described for the encoder case in figure 12.15c: Instead of allowing every token to interact with every other token, we can limit the interactions to a local neighborhood around each patch. By doing so, we reduce the quadratic complexity associated with the number of patches.

2. **Global Tokens and hierarchical attention**: Similar to figure 12.5h, introduce global tokens which interact with all image patches to capture the global context of the image. By using global tokens and a hierarchical attention mechanism, each patch initially attends only to the global tokens, and then attends to other patches based on the information from the global tokens. This reduces the number of pairwise interactions between patches, reducing the overall computation.

Problem 12.9

Consider representing an image with a grid of 16×16 patches, each represented by a patch embedding of length 512. Compare the amount of computation required in the DaViT transformer to perform attention (i) between the patches, using all of the channels, and (ii) between the channels, using all of the patches.

Dual attention vision transformers (DaViT) alternates two types of transformers. In the first, image patches attend to one another, and the self-attention computation uses all the channels. In the second, the channels attend to one another, and the self-attention computation uses all the image patches.

The amount of computation required in the DaViT transformer to perform attention between all the patches and using all of the channels can be calculated as follows:

- Number of computations per patch: $16 \times 16 \times 512 = 131,072$
- Total number of computations across all patches: $16 \times 16 \times 131,072 = 33,554,432$

The amount of computation required too perform attention between the channels, using all of the patches can be calculated as follows:

- Number of computations per channel: $16 \times 16 \times 512 = 131,072$
- Total number of computations across all channels: $512 \times 131,072 = 67,108,864$

The amount of computation required for attention between channels using all patches is approximately double the computation required for attention between patches using all channels.

Problem 12.10

Attention weights are usually computed as:

$$\text{a}[\mathbf{x}_m, \mathbf{x}_n] = \text{softmax}_m\left[\mathbf{k}_\bullet^T \mathbf{q}_n\right] = \frac{\exp[\mathbf{k}_m^T \mathbf{q}_n]}{\sum_{m'=1}^N \exp[\mathbf{k}_{m'}^T \mathbf{q}_n]}$$

Consider replacing $\exp[\mathbf{k}_m^T \mathbf{q}_n]$ with the dot product $\mathbf{g}[\mathbf{k}_m]^T \mathbf{g}[\mathbf{q}_n]$ where $\mathbf{g}[\bullet]$ is a nonlinear transformation. Show how this makes the computation of the attention weights more efficient.

$$\text{a}[\mathbf{x}_m, \mathbf{x}_n] = \frac{\mathbf{g}[\mathbf{k}_m]^T \mathbf{g}[\mathbf{q}_n]}{\sum_{m'=1}^N \mathbf{g}[\mathbf{k}_{m'}]^T \mathbf{g}[\mathbf{q}_n]} = \frac{\mathbf{g}[\mathbf{k}_m]^T \mathbf{g}[\mathbf{q}_n]}{\left(\sum_{m'=1}^N \mathbf{g}[\mathbf{k}_{m'}]^T\right) \mathbf{g}[\mathbf{q}_n]}$$

Using this formulation, the computation of the attention weights is more efficient because the summation can be computed before the dot product $\mathbf{g}[\mathbf{k}_m]^T\mathbf{g}[\mathbf{q}_n]$. In practice, this means that the computation of the attention is reduced from $\mathcal{O}(N^2D)$ to $\mathcal{O}(ND)$, making the computation more efficient.