CS 577 — Deep Learning — Homework 5

Read these instructions carefully:

- In the LaTeX source code, type your answer in between "`%%% BEGIN ANSWER`" and "`%%% END ANSWER`". For advanced LaTeXusers, you can use your custom macros if you wish by placing them between "`%%% BEGIN MACROS`" and "`%%% END MACROS`" in the header. Do not modify anything else.
- Turn in both your `.tex` file and the generated `.pdf` file.

1 The transformer module

In homework 3 and 4, we investigated computation for a simple attention model, where each data point $\mathbf{X}^{(i)}$ is itself a *collection* of C vectors (for concreteness, take $C = 32$). Namely, we write $\mathbf{X}^{(i)}$ as a matrix whose c-th column is $\mathbf{x}^{(i,c)}$, i.e.,

$$\mathbf{X}^{(i)} = \begin{bmatrix} \mathbf{x}^{(i,1)} & \cdots & \mathbf{x}^{(i,C)} \end{bmatrix} \in \mathbb{R}^{d \times C} \quad \text{is a } d \times C \text{ matrix}$$

You can think of the $\mathbf{x}^{(i,j)}$'s for each $j = 1, \ldots, C$ as the "word embedding" for some sentence of length C. We say that $\mathbf{X}^{(i)}$ is a "sequence".

In the code, we will name

- C as `n_context`
- d as `n_features`
- n as `n_samples`, which denotes the mini-batch size $\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(n)}$.

The transformer module that we consider in this homework is composed of a *self-attention* submodule followed by a *multilayer perceptron* submodule with 1-hidden layer. For simplicity, we focus on these two submodules, leaving aside for now dropout, layer norm, etc. Both of these submodules will have parameters. We will refer to these parameters as $\theta^{(\texttt{att})}$ and $\theta^{(\texttt{mlp})}$ respectively.

The self-attention submodule has parameters

$$\theta^{(\texttt{att})} = [\mathbf{W}^{(Q)}, \mathbf{W}^{(K)}, \mathbf{W}^{(V)}] \qquad (\text{"}Q\text{, }K\text{, }V\text{" stands for "query", "key", "value", respectively})$$

where

$$\mathbf{W}^{(Q)} \text{ and } \mathbf{W}^{(K)} \text{ and } \mathbf{W}^{(V)} \in \mathbb{R}^{d \times d}.$$

Here are the main changes compared to the previous homeworks

- We assume $q = d$ for simplicity. In this homework, there is no separate `n_reduced`.
- In previous homeworks, we had $\mathbf{W}^{(1)}$ instead of $\mathbf{W}^{(Q)}$, and had $\mathbf{W}^{(2)}$ instead of $\mathbf{W}^{(K)}$.
- In previous homeworks, we had $\mathbf{w}^{(3)}$ which was a vector. In this homework, we have that $\mathbf{W}^{(V)}$ is a $d \times d$ matrix.

In this homework, our self-attention submodule is "**seq-2-seq**", i.e., it maps the sequence $\mathbf{X}^{(i)}$ to another sequence $\texttt{attention}(\mathbf{X}^{(i)};\theta)$:

$$\texttt{attention}(\mathbf{X}^{(i)};\theta) := \mathbf{W}^{(V)\top}\mathbf{X}^{(i)}\text{softmax}\left(\mathbf{X}^{(i)\top}\mathbf{W}^{(K)\top}\mathbf{W}^{(Q)}\mathbf{X}^{(i)}\right) \in \mathbb{R}^{d\times C}.$$

Note that

- $\text{softmax}\left(\mathbf{X}^{(i)\top}\mathbf{W}^{(K)\top}\mathbf{W}^{(Q)}\mathbf{X}^{(i)}\right)$ is a C-by-C matrix.
- $\texttt{Attention}(\mathbf{X}^{(i)};\theta)$ has the same shape as $\mathbf{X}^{(i)}$

The **MLP submodule** has a single hidden layer and the following parameters:

$$\theta^{(\texttt{mlp})} = [\mathbf{W}^{(H)}, \mathbf{b}^{(H)}, \mathbf{W}^{(\text{O})}, \mathbf{b}^{(\text{O})}] \qquad (\text{"}H\text{, } O\text{" stands for "hidden" and "output", respectively})$$

where

$$\mathbf{W}^{(H)} \in \mathbb{R}^{d\times h}, \quad \mathbf{b}^{(H)} \in \mathbb{R}^{h}, \quad \mathbf{W}^{(\text{O})} \in \mathbb{R}^{h\times d}, \quad \text{and} \quad \mathbf{b}^{(\text{O})} \in \mathbb{R}^{d}.$$

The hidden-layer activation is computed as $\mathbf{H} = \text{relu}\left(\mathbf{W}^{(H)\top}\mathbf{X}^{(i)} + \mathbf{b}^{(H)}\right) \in \mathbb{R}^{h\times C}$ and the output

$$\texttt{MLP}(\mathbf{X}^{(i)};\theta^{(\texttt{mlp})}) = \mathbf{W}^{(O)\top}\mathbf{H} + \mathbf{b}^{(O)} \in \mathbb{R}^{d\times C}$$

Putting both submodules together, we have the transformer block:

$$\texttt{TransformerBlock}(\mathbf{X}^{(i)}\,;\,\theta^{(\texttt{att})},\theta^{(\texttt{mlp})}) := \texttt{MLP}(\texttt{Attention}(\mathbf{X}^{(i)};\theta^{(\texttt{att})})\,;\,\theta^{(\texttt{mlp})})$$

The above can be implemented (vectorizing over a batch $\mathbf{X}^{(1)},\ldots,\mathbf{X}^{(n)}$) as follows:

```
# IMPORTANT: the batch data tensor has shape (n_samples, n_context, n_features)
#
class SingleHeadAttention:
    def __init__(self, n_features):
        self.Wq = ag.Tensor(np.random.randn(n_features, n_features), label="Wq")
        self.Wk = ag.Tensor(np.random.randn(n_features, n_features), label="Wk")
        self.Wv = ag.Tensor(np.random.randn(n_features, n_features), label="Wv")
    def __call__(self, Xin):
        Queries = Xin @ self.Wq
        Keys = Xin @ self.Wk
        KQ = (Keys @ ag.moveaxis(Queries, 1,2))
        expKQ = ag.exp(KQ)
        softmaxKQ = expKQ / ag.sum(expKQ, axis=1, keepdims=True)
        Xout = ag.moveaxis(ag.moveaxis(Xin,1,2) @ softmaxKQ, 1,2) @ self.Wv
        return Xout

class MLP:
    def __init__(self, n_features, n_hidden):
        self.Wh = ag.Tensor(np.random.randn(n_features, n_hidden), label="Wh")
        self.bh = ag.Tensor(np.random.randn(n_hidden), label="bh")
        self.wo = ag.Tensor(np.random.randn(n_hidden, n_features), label="Wo")
        self.bo = ag.Tensor(np.random.randn(n_features), label="bo")

    def __call__(self, Xin):
        hidden = ag.relu((Xin @ self.Wh) + self.bh)
        return hidden @ self.wo + self.bo

class TransformerBlock:
    def __init__(self, n_features, n_hidden):
        self.att = SingleHeadAttention(n_features)
        self.mlp = MLP(n_features, n_hidden)
    def __call__(self, Xin):
        return self.mlp(self.att(Xin))
```

[5] point(s) — part a:
This is a programming exercise. See `hw5.ipynb`

---

[5] point(s) — part b:
Explain your answer from part a.

Answer:
Firstly, the code calculates the **binary cross-entropy loss between the model's output and the target** labels:

$$\text{loss} = \text{nn.BinaryCrossEntropyLoss}()(\text{att}(X)[:, 3, 0], y)$$

Here, the model's output is defined as $\text{att}(X)[:, 3, 0]$, and y represents the true label. After calculating the loss, the gradient of the loss with respect to the model's parameters is computed with loss.backward() to **facilitate backpropagation.**
I selected the last element of the 3rd row of the output tensor, $\text{att}(X)[:, 3, 0]$, as the model's output because the last row in each column represents an **accumulated summary** of information from all previous rows. Finally, the second and third column of output are zeros because Wv have zero values in these columns, then any projection along those columns will yield zero values.

```
[[[-0.633688    0.          0.        ]
  [ 0.71820254  0.          0.        ]
  [ 1.56586578  0.          0.        ]
  [ 1.6528977   0.          0.        ]]

 [[-0.30924845  0.          0.        ]
  [-0.91080457  0.          0.        ]
  [-0.84907579  0.          0.        ]
  [-0.50304305  0.          0.        ]]

 [[-1.50919189  0.          0.        ]
  [-1.53127116  0.          0.        ]
  [-1.53920107  0.          0.        ]
  [-1.54629904  0.          0.        ]]

 [[ 0.19011611  0.          0.        ]
  [-1.17665276  0.          0.        ]
  [-0.12830874  0.          0.        ]
  [-1.18000691  0.          0.        ]]

 [[-1.8337886   0.          0.        ]
  [ 0.14258813  0.          0.        ]
  [ 0.94361548  0.          0.        ]
  [ 0.74528666  0.          0.        ]]]
```

Figure 1: Output of the attention, in red the accumulated summary of information from all previous rows

---

[10] point(s) — part c:

Given that a single `matmul` between matrices of shape (a, b) and (b, c) requires $O(abc)$ floating point operations (FLOPs), compute the number of FLOPs required to compute one forward pass of the `TransformerBlock` on an input tensor of shape (n, C, d). Give your answer in terms of big-O notation involving the quantities n, C, d and h.

Answer:
To calculate the computational complexity for a forward pass through the `TransformerBlock`, given an input tensor of shape (n, C, d), where:

- n is the number of samples in the batch.
- C is the sequence length or context size.
- d is the input feature dimension.
- h is the hidden dimension size used in the `MLP` component.

The `TransformerBlock` consists of two main parts:

- A `SingleHeadAttention` layer.
- An `MLP` layer.

Therefore, the total FLOPs for the `TransformerBlock` can be calculated as the **sum of the FLOPs** for the `SingleHeadAttention` and `MLP` layers.

1. FLOPs for the SingleHeadAttention Layer

In the `SingleHeadAttention` layer:

- `Wq`, `Wk`, and `Wv` are weight matrices of shape (d, d).
- The input tensor `Xin` has a shape (n, C, d).

Steps and FLOPs in SingleHeadAttention:

- The input `Xin` is projected to Queries, Keys, and Values by multiplying with `Wq`, `Wk`, and `Wv`.
- Each of these multiplications has a complexity of $O(n \cdot C \cdot d^2)$, as the input (n, C, d) is multiplied by a weight matrix of (d, d).
- Since there are three projections (for `Wq`, `Wk`, and `Wv`), the total FLOPs for this step is:

$$3 \cdot O(n \cdot C \cdot d^2) = O(n \cdot C \cdot d^2)$$

Attention Score Calculation (Dot Product between Keys and Queries):

- After projecting Keys and Queries, they have shapes (n, C, d).
- To calculate the attention scores, the dot product of Keys and Queries (with one axis moved) results in a matrix of shape (n, C, C).
- This operation has a complexity of $O(n \cdot C^2 \cdot d)$, as we are performing $C \times C$ matrix multiplications for each sample and each feature dimension.

Softmax Normalization:

- The softmax normalization over the attention scores has a complexity of $O(n \cdot C^2)$, as it's applied across the sequence length for each sample.

Weighted Sum of Values:

- Finally, we compute the weighted sum of Values using the attention weights, resulting in an output shape of (n, C, d).
- This operation also has a complexity of $O(n \cdot C^2 \cdot d)$, as it involves matrix multiplications between the attention weights (n, C, C) and Values (n, C, d).

Total FLOPs for SingleHeadAttention: Adding all these terms together, we get:

$$O(n \cdot C \cdot d^2) + O(n \cdot C^2 \cdot d) + O(n \cdot C^2) + O(n \cdot C^2 \cdot d) = O(n \cdot C \cdot d^2 + n \cdot C^2 \cdot d)$$

2. FLOPs for the MLP Layer

In the `MLP` layer:

- `Wh` (the first weight matrix) has a shape of (d, h).
- `Wo` (the second weight matrix) has a shape of (h, d).
- The input to the `MLP` layer has shape (n, C, d).

First Linear Transformation:

- The input (n, C, d) is multiplied by `Wh` (of shape (d, h)) to produce a hidden representation of shape (n, C, h).
- This matrix multiplication has a complexity of $O(n \cdot C \cdot d \cdot h)$.

ReLU Activation:

- The ReLU activation function is applied element-wise on (n, C, h), with a complexity of $O(n \cdot C \cdot h)$.

Second Linear Transformation:

- The hidden representation (n, C, h) is multiplied by `Wo` (of shape (h, d)) to produce an output of shape (n, C, d).
- This matrix multiplication has a complexity of $O(n \cdot C \cdot h \cdot d)$.

Total FLOPs for MLP: Adding up the terms, we get:

$$O(n \cdot C \cdot d \cdot h)$$

Total FLOPs for the TransformerBlock

Now, combining the FLOPs from `SingleHeadAttention` and `MLP`, we get the total FLOPs for one forward pass through the `TransformerBlock`:

$$O(n \cdot C \cdot d^2) + O(n \cdot C^2 \cdot d) + O(n \cdot C \cdot d \cdot h)$$

$$O(n \cdot C \cdot d^2 + n \cdot C^2 \cdot d + n \cdot C \cdot d \cdot h)$$