Homework 4: Transformer

CSCI-GA 2572 Deep Learning

Anoushka Gupta ag8733@nyu.edu

1 Theory (50 pts)

1.1 Attention (13 pts)

(a) (1 pts) 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).

ANSWER:

$\mathbf{a} = softargmax_\beta(\mathbf{K}^T.\mathbf{q}) \in \mathbb{R}^{\mathbb{m}}$

$\mathbf{h} = \mathbf{V}.\mathbf{a} \in \mathbb{R}^{\mathbb{t}}$

$\mathbf{A} \in \mathbb{R}^{\mathbb{m}\times\mathbb{n}}$

$\mathbf{H} = \mathbf{V}.\mathbf{A} \in \mathbb{R}^{\mathbb{t}\times\mathbb{n}}$

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

ANSWER:

$\mathbf{a} = softargmax_\beta(\mathbf{X}^T.\mathbf{x})$

OR
$\mathbf{a} = softargmax_\beta(\mathbf{K}^T.\mathbf{q})$

As β changes the sharpness of the attention weight changes.A smaller β results in smoother attention weights. When β is very cold the softargmax function gives a one hot encoding of x. When β is cold we preserve the vectors.
When β is very hot we get $\frac{1}{t}$ where $\mathbf{X} \in \mathbb{n} \times \mathbb{t}$ since $(\mathbf{X}^T.\mathbf{x})$ is effectively comparing a matrix which has t items with one item. This gives us a score in form of a vector having t values. When β is hot we dilute the vectors.

The β convenient to use is $\frac{1}{d^{0.5}}$ where d is the dimension of the query item $\mathbf{q}$ and $\mathbf{q} \in \mathbb{R}^d$

(c) (3 pts) 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:

The output preserve the value vectors when attention $\mathbf{A}$ is the identity matrix since $\mathbf{H} = \mathbf{V.A}$ where V is the value matrix and $\mathbf{H}$ is the hidden representation.
Since $\mathbf{a} = softargmax_\beta(\mathbf{K}^T.\mathbf{q})$
When β is set to very cold or a very small value the vector $\mathbf{a}$ becomes very close to 1 and is a one hot encoding. Due to this $\mathbf{A}$ acts like an identity matrix and value vector is preserved.
This type of attention is called hard attention. In a fully connected architecture this can be implemented by using a linear transformation with no bias where the weights of the linear layer is the identity matrix and the input is the value vector.

(d) (3 pts) 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:

The output is a spread version of value vector when the most attention weights are close to 0 and some are close to 1 such that the sum of all the weights is 1. i.e $||a||_1 = 1$. This will dilute the different value vectors $\mathbf{v}$.
The value of β should be very big or it should be hot. This type of attention is called soft attention.
In fully connected aarchitecture this can be done by taking the value vector a input and using he attention weights as weights in a linear transformation without any bias.

(e) (2 pts) 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?

ANSWER:

The output of $\mathbf{H}$ changes linearly wrt the perturbation ϵ. Let $\mathbf{x} \in R^n$ and $\mathbf{X} \in R^{n\times t}$.
Let the change happen in $\mathbf{k}_1$. Then $\hat{k_1} = k_1 + \epsilon$. The key matrix $\mathbf{K}$ is:
$\mathbf{K} = \begin{bmatrix}\mathbf{k}_1\mathbf{k}_2....\mathbf{k}_t\end{bmatrix}$
With the change key matrix $\hat{\mathbf{K}}$ is :
$\hat{\mathbf{K}} = \begin{bmatrix}\hat{\mathbf{k}}_1\mathbf{k}_2....\mathbf{k}_t\end{bmatrix} = \begin{bmatrix}\mathbf{k}_1 + \epsilon \;\; k_2 \; .... \; k_t\end{bmatrix}$
We know $\mathbf{a} = softargmax_\beta(\mathbf{K}^T.\mathbf{q})$
$\mathbf{a}_1 = softargmax_\beta(\mathbf{K}^T\mathbf{q}_1)$

$$= softargmax_\beta\left(\begin{bmatrix}(\mathbf{k}_1)\mathbf{q}_1\\ \mathbf{k}_2\mathbf{q}_1\\ ..\\ ..\\ \mathbf{k}_t.\mathbf{q}_1\end{bmatrix}\right)$$

With the change,
$\hat{a_1} = softargmax_\beta(\hat{\mathbf{K}}^T.\mathbf{q}_1)$

$$= softargmax_\beta\left(\begin{bmatrix}(\mathbf{k}_1 + \epsilon)\mathbf{q}_1\\ \mathbf{k}_2.\mathbf{q}_1\\ ..\\ ..\\ \mathbf{k}_t\mathbf{q}_1\end{bmatrix}\right)$$

Similarly we can calculate vector $\hat{\mathbf{a}_2}, \hat{\mathbf{a}_3}...\hat{\mathbf{a}}_t$ where the perturbation happens and $\mathbf{a}_2, \mathbf{a}_3...\mathbf{a}_t$ without perturbation.
We know $\mathbf{h} = \mathbf{V}.\mathbf{a}$

$$\mathbf{h}_1 = \mathbf{V}\mathbf{a}_1 = \mathbf{V}.softargmax_\beta\left(\begin{bmatrix}(\mathbf{k}_1)\mathbf{q}_1\\ \mathbf{k}_2\mathbf{q}_1\\ ..\\ ..\\ \mathbf{k}_t\mathbf{q}_1\end{bmatrix}\right) = \begin{bmatrix}\mathbf{V}.(softargmax_\beta(\mathbf{k}_1.\mathbf{q}_1))\\ \mathbf{V}.(softargmax_\beta(\mathbf{k}_2.\mathbf{q}_1))\\ ..\\ ..\\ \mathbf{V}.(softargmax_\beta(\mathbf{k}_t.\mathbf{q}_1))\end{bmatrix}$$

Similarly with the perturbation,

$$\hat{h_1} = V.\hat{a_1} = V.softargmax_\beta\left(\begin{bmatrix}(k_1 + \epsilon).q_1)\\ k_2q_1\\ ..\\ ..\\ k_tq_1\end{bmatrix}\right) = \begin{bmatrix}V.(softargmax_\beta(k_1.q_1) + V.(softargmax_\beta(\epsilon.q_1)\\ V.(softargmax_\beta(k_2.q_1))\\ ..\\ ..\\ V.(softargmax_\beta(k_t.q_1))\end{bmatrix}$$

Similarly we can calculate vector $\hat{\mathbf{h}_2}, \hat{\mathbf{h}_3}...\hat{\mathbf{h}_t}$ where the perturbation happens and $\mathbf{h}_2, \mathbf{h}_3...\mathbf{h}_t$ without perturbation.
$H = \begin{bmatrix}\mathbf{h}_1 \; \mathbf{h}_2...\mathbf{h}_t\end{bmatrix}$

$$= \begin{bmatrix}\mathbf{V}.(softargmax_\beta(\mathbf{k}_1.\mathbf{q}_1)) & \mathbf{V}.(softargmax_\beta(\mathbf{k}_1.\mathbf{q}_2)) & .. & \mathbf{V}.(softargmax_\beta(\mathbf{k}_1.\mathbf{q}_t))\\ \mathbf{V}.(softargmax_\beta(\mathbf{k}_2.\mathbf{q}_1)) & \mathbf{V}.(softargmax_\beta(\mathbf{k}_2.\mathbf{q}_2)) & .. & \mathbf{V}.(softargmax_\beta(\mathbf{k}_2.\mathbf{q}_t))\\ .. & .. & .. & ..\\ \mathbf{V}.(softargmax_\beta(\mathbf{k}_t.\mathbf{q}_1)) & \mathbf{V}.(softargmax_\beta(\mathbf{k}_t.\mathbf{q}_2)) & .. & \mathbf{V}.(softargmax_\beta(\mathbf{k}_t.\mathbf{q}_t))\end{bmatrix}$$

With the perturbation,

$$\hat{\mathbf{H}} = \begin{bmatrix}\hat{\mathbf{h}_1}\ \hat{\mathbf{h}_2}...\hat{\mathbf{h}_t}\end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{V}.(softargmax_\beta(\mathbf{k}_1+\epsilon).\mathbf{q}_1) & .. & \mathbf{V}.(softargmax_\beta(\mathbf{k}_1+\epsilon).\mathbf{q}_t) \\ \mathbf{V}.(softargmax_\beta(\mathbf{k}_2.\mathbf{q}_1)) & .. & \mathbf{V}.(softargmax_\beta(\mathbf{k}_2.\mathbf{q}_t)) \\ .. & .. & .. \\ \mathbf{V}.(softargmax_\beta(\mathbf{k}_t.\mathbf{q}_1)) & .. & \mathbf{V}.(softargmax_\beta(\mathbf{k}_t.\mathbf{q}_t)) \end{bmatrix}$$

(f) (2 pts) 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?

ANSWER:

The output of $\mathbf{H}$ changes wrt the perturbation α. Let $x \in R^n$ and $X \in R^{n\times t}$.
Let the change happen in k_1. Then $\hat{k_1} = \alpha k_1$. The key matrix K is:
$K = \begin{bmatrix}k_1 k_2 .... k_t\end{bmatrix}$
With the change key matrix $\hat{K}$ is :
$\hat{K} = \begin{bmatrix}\hat{k_1} k_2 .... k_t\end{bmatrix} = \begin{bmatrix}\alpha.k_1\ \ k_2\ ....\ k_t\end{bmatrix}$
We know $a = softargmax_\beta(K^T q)$
$a_1 = softargmax_\beta(K^T q_1)$
$$= softargmax_\beta\left(\begin{bmatrix}(k_1)q_1 \\ k_2 q_1 \\ .. \\ .. \\ k_t q_1\end{bmatrix}\right)$$
With the change,
$\hat{a_1} = softargmax_\beta(\hat{K}^T q_1)$
$$= softargmax_\beta\left(\begin{bmatrix}(\alpha k_1)q_1 \\ k_2 q_1 \\ .. \\ .. \\ k_t q_1\end{bmatrix}\right)$$
Similarly we can calculate vector $\hat{a_2}, \hat{a_3} ... \hat{a_t}$ where the perturbation happens and $a_2, a_3 ... a_t$ without perturbation.
We know $h = Va$
$$h_1 = Va_1 = V.softargmax_\beta\left(\begin{bmatrix}(k_1)q_1 \\ k_2 q_1 \\ .. \\ .. \\ k_t q_1\end{bmatrix}\right) = \begin{bmatrix}V.(softargmax_\beta(k_1.q_1)) \\ V.(softargmax_\beta(k_2.q_1)) \\ .. \\ .. \\ V.(softargmax_\beta(k_t.q_1))\end{bmatrix}$$
Similarly with the perturbation,

$$\hat{h_1} = V.\hat{a_1} = V.softargmax_\beta\left(\begin{bmatrix}(\alpha k_1).q_1)\\ k_2 q_1\\ ..\\ ..\\ k_t q_1\end{bmatrix}\right) = \begin{bmatrix}V.(softargmax_\beta(\alpha k_1.q_1)\\ V.(softargmax_\beta(k_2.q_1))\\ ..\\ ..\\ V.(softargmax_\beta(k_t.q_1))\end{bmatrix}$$

Similarly we can calculate vector $\hat{h_2}, \hat{h_3}...\hat{h_t}$ where the perturbation happens and $h_2, h_3...h_t$ without perturbation.

$H = \begin{bmatrix}h_1 \ h_2 ... h_t\end{bmatrix}$

$$= \begin{bmatrix}V.(softargmax_\beta(k_1.q_1)) & V.(softargmax_\beta(k_1.q_2)) & .. & V.(softargmax_\beta(k_1.q_t))\\ V.(softargmax_\beta(k_2.q_1)) & V.(softargmax_\beta(k_2.q_2)) & .. & V.(softargmax_\beta(k_2.q_t))\\ .. & .. & .. & ..\\ V.(softargmax_\beta(k_t.q_1)) & V.(softargmax_\beta(k_t.q_2)) & .. & V.(softargmax_\beta(k_t.q_t))\end{bmatrix}$$

With the perturbation,

$\hat{H} = \begin{bmatrix}\hat{h_1} \ \hat{h_2} ... \hat{h_t}\end{bmatrix}$

$$= \begin{bmatrix}V.(softargmax_\beta(\alpha k_1).q_1) & .. & V.(softargmax_\beta(\alpha k_1).q_t)\\ V.(softargmax_\beta(k_2.q_1)) & .. & V.(softargmax_\beta(k_2.q_t))\\ .. & .. & ..\\ V.(softargmax_\beta(k_t.q_1)) & .. & V.(softargmax_\beta(k_t.q_t))\end{bmatrix}$$

1.2 Multi-headed Attention (3 pts)

(a) (1 pts) 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.

ANSWER:

Multihead attention can be represented as-

$$\begin{bmatrix} \mathbf{q}^1 \\ \mathbf{q}^2 \\ . \\ . \\ \mathbf{q}^h \\ \mathbf{k}^1 \\ \mathbf{k}^2 \\ . \\ . \\ \mathbf{k}^h \\ \mathbf{v}^1 \\ \mathbf{v}^2 \\ . \\ . \\ \mathbf{v}^h \end{bmatrix} = \begin{bmatrix} \mathbf{W_q}^1 \\ \mathbf{W_q}^2 \\ . \\ . \\ \mathbf{W_q}^h \\ \mathbf{W_k}^1 \\ \mathbf{W_k}^2 \\ . \\ . \\ \mathbf{W_k}^h \\ \mathbf{W_v}^1 \\ \mathbf{W_v}^2 \\ . \\ . \\ \mathbf{W_v}^h \end{bmatrix} \mathbf{x} \in R^{3hd}$$

$\{\mathbf{q_i}\}_{i=1}^{t}{}^{(1)}, \{\mathbf{k_i}\}_{i=1}^{t}{}^{(1)}, \{\mathbf{v_i}\}_{i=1}^{t}{}^{(1)} \rightsquigarrow \mathbf{Q}^{(1)}, \mathbf{K}^{(1)}, \mathbf{V}^{(1)}$

$\{\mathbf{q_i}\}_{i=1}^{t}{}^{(2)}, \{\mathbf{k_i}\}_{i=1}^{t}{}^{(2)}, \{\mathbf{v_i}\}_{i=1}^{t}{}^{(2)} \rightsquigarrow \mathbf{Q}^{(2)}, \mathbf{K}^{(2)}, \mathbf{V}^{(2)}$

...
$\{\mathbf{q_i}\}_{i=1}^{t}{}^{(h)}, \{\mathbf{k_i}\}_{i=1}^{t}{}^{(h)}, \{\mathbf{v_i}\}_{i=1}^{t}{}^{(h)} \rightsquigarrow \mathbf{Q}^{(h)}, \mathbf{K}^{(h)}, \mathbf{V}^{(h)}$

$\mathbf{a}^{(1)} = softargmax_{\beta}(\mathbf{K}^{(1)^T}\mathbf{q}^{(1)})$
$\{\mathbf{a_i}\}_{i=1}^{t}{}^{(1)} \rightsquigarrow \mathbf{A}^{(1)}$

$\mathbf{a}^{(2)} = softargmax_{\beta}(\mathbf{K}^{(2)^T}\mathbf{q}^{(2)})$
$\{\mathbf{a_i}\}_{i=1}^{t}{}^{(2)} \rightsquigarrow \mathbf{A}^{(2)}$

....

$\mathbf{a}^{(h)} = softargmax_{\beta}(\mathbf{K}^{(h)^T}\mathbf{q}^{(h)})$
$\{\mathbf{a_i}\}_{i=1}^{t}{}^{(h)} \rightsquigarrow \mathbf{A}^{(h)}$

$\mathbf{h}^{(1)} = \mathbf{V}^{(1)}.\mathbf{a}^{(1)} \rightsquigarrow \mathbf{H}^{(1)} = \mathbf{V}^{(1)}.\mathbf{A}^{(1)}$

...

$\mathbf{h}^{(h)} = \mathbf{V}^{(h)}.\mathbf{a}^{(h)} \rightsquigarrow \mathbf{H}^{(h)} = \mathbf{V}^{(h)}.\mathbf{A}^{(h)}$

$$H = \begin{bmatrix} \mathbf{H}^{(1)} \\ \mathbf{H}^{(2)} \\ .. \\ \mathbf{H}^{(h)} \end{bmatrix}$$

(b) (2 pts) 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)

ANSWER:

Yes, there is something similar to multi-headed attention in CNN. This can be done by setting the *groups* parameter in the Conv1D layer to the number of $in_channels$. When the *group* parameter is set to this, each input is convolved with its own set of filters of size $\frac{out_channels}{in_channels}$

This allows the CNN model to capture the relationship between the input features just like multi-headed attention helps capture the relationship between the elements of the input sequence. This helps learning more complex representations of the input.

1.3 Self Attention (11 pts)

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

(a) (2 pts) Given an input $\boldsymbol{C} \in \mathbb{R}^{e\times n}$, what are 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:

Given $\mathbf{C} \in R^{e\times n}$
i.e $\mathbf{c} \in R^{e}$ and $\{x_i\}_{i=1}^{n} = \{x_1, x_2.....x_n\}$
Let the weights be-
$(\mathbf{W}_q{}^{(i)})_{i=1}^{h} \in R^{d'\times e}$
$(\mathbf{W}_k{}^{(i)})_{i=1}^{h} \in R^{d'\times e}$
$(\mathbf{W}_v{}^{(i)})_{i=1}^{h} \in R^{d"\times e}$

$\mathbf{q}^{(1)} = \mathbf{W_q}^{(1)} c \in R^{d'}$
$\mathbf{k}^{(1)} = \mathbf{W_k}^{(1)} c \in R^{d'}$
$\mathbf{v}^{(1)} = \mathbf{W_v}^{(1)} c \in R^{d"}$
$\{\mathbf{q_i}\}_{i=1}^{t}{}^{(1)}, \{\mathbf{k_i}\}_{i=1}^{t}{}^{(1)}, \{\mathbf{v_i}\}_{i=1}^{t}{}^{(1)} \rightsquigarrow \mathbf{Q}^{(1)}, \mathbf{K}^{(1)} \in R^{d'\times n}, \mathbf{V}^{(1)} \in R^{d"\times n}$

Similarly,
$\{\mathbf{q_i}\}_{i=1}^{t}{}^{(2)}, \{\mathbf{k_i}\}_{i=1}^{t}{}^{(2)}, \{\mathbf{v_i}\}_{i=1}^{t}{}^{(2)} \rightsquigarrow \mathbf{Q}^{(2)}, \mathbf{K}^{(2)} \in R^{d' \times n}, \mathbf{V}^{(2)} \in R^{d" \times n}$
....

$\{\mathbf{q_i}\}_{i=1}^{t}{}^{(h)}, \{\mathbf{k_i}\}_{i=1}^{t}{}^{(h)}, \{\mathbf{v_i}\}_{i=1}^{t}{}^{(h)} \rightsquigarrow \mathbf{Q}^{(h)}, \mathbf{K}^{(h)} \in R^{d' \times n}, \mathbf{V}^{(h)} \in R^{d" \times n}$

$\mathbf{a}^{(1)} = softargmax_{\beta}(\mathbf{K}^{(1)^T}\mathbf{q}^{(1)}) \in R^n$
$\{\mathbf{a_i}\}_{i=1}^{t}{}^{(1)} \rightsquigarrow \mathbf{A}^{(1)} \in R^{n \times n}$

$\mathbf{a}^{(2)} = softargmax_{\beta}(\mathbf{K}^{(2)^T}\mathbf{q}^{(2)}) \in R^n$
$\{\mathbf{a_i}\}_{i=1}^{t}{}^{(2)} \rightsquigarrow \mathbf{A}^{(2)} \in R^{n \times n}$

....

$\mathbf{a}^{(h)} = softargmax_{\beta}(\mathbf{K}^{(h)^T}\mathbf{q}^{(h)}) \in R^n$
$\{\mathbf{a_i}\}_{i=1}^{t}{}^{(h)} \rightsquigarrow \mathbf{A}^{(h)} \in R^{n \times n}$

$\mathbf{h}^{(1)} = \mathbf{V}^{(1)}.\mathbf{a}^{(1)} \in R^{d"}$
$\{\mathbf{h_i}\}_{i=1}^{t}{}^{(1)} \rightsquigarrow \mathbf{H}^{(1)} = \mathbf{V}^{(1)}.\mathbf{A}^{(1)} \in R^{d" \times n}$

...

$\mathbf{h}^{(h)} = \mathbf{V}^{(h)}.\mathbf{a}^{(h)} \in R^{d"}$
$\{\mathbf{h_i}\}_{i=1}^{t}{}^{(h)} \rightsquigarrow \mathbf{H}^{(h)} = \mathbf{V}^{(h)}.\mathbf{A}^{(h)} \in R^{d" \times n}$

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}^{(1)} \\ \mathbf{Q}^{(2)} \\ .. \\ \mathbf{Q}^{(h)} \end{bmatrix} \in R^{d' \times n \times h} \quad \mathbf{K} = \begin{bmatrix} \mathbf{K}^{(1)} \\ \mathbf{K}^{(2)} \\ .. \\ \mathbf{K}^{(h)} \end{bmatrix} \in R^{d' \times n \times h}$$

$$\mathbf{V} = \begin{bmatrix} \mathbf{V}^{(1)} \\ \mathbf{V}^{(2)} \\ .. \\ \mathbf{V}^{(h)} \end{bmatrix} \in R^{d" \times n \times h} \quad \mathbf{H} = \begin{bmatrix} \mathbf{H}^{(1)} \\ \mathbf{H}^{(2)} \\ .. \\ \mathbf{H}^{(h)} \end{bmatrix} \in R^{d" \times n \times h}$$

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

ANSWER:

Positional encoding is needed in self attention when the order of the input sequence matters. The positional encoding captures the long range dependencies between the various elements of the sequence.. For example: In NLP tasks, where input are sentences the order of the words matter to make sense of the grammar, the meaning of the language and the actual semantics of the sentence.
Positional encoding is not required when the order of the elements of input sequence don't matter. For example: Consider a sequence of numbers where the next number is not effected by the previous number. In such a input sequence positional encoding is not necessary and wouldn't effect the models performance.

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

ANSWER:
When the input, query, key nad value matrix are equal $X = Q = V = K$.
$a = softargmax(K^T Q)$.
If X and X^T are orthogonal matrices then $X.X^T = 1$
The attention matrix A will be a identity matrix and $H = VA$ will give back the input. Hence, when the matrices are equal self attention acts as a identity layer.
The self-attention layer behaves like a permutation layer when the attention scores are all zero. In this case, the weighted average of the input tokens will be zero, which means that the output sequence will be a permutation of the input sequence.

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

ANSWER: Self attention will act like a running linear layer when the self-attention layer behaves like a convolution layer of kernel size 1. The self attention layer acts as a convolutional layer with kernel size 1 when the bias of attention are all set to 1. In this case self attention is influenced by only immediate neighbours and itself.

(e) (3 pts) 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: When using positional encoding, the self-attention layer can behave like a convolution layer with a kernel larger than 1 when the input sequence has a certain pattern that is captured. Suppose the input sequence of text contains repeating patterns of phrases.The positional encoding can be designed to encode the position of each token relative to the start of its corresponding phrase or sentence. In this case, the self-attention layer can learn to attend to the entire phrase or sentence.This is similar to a convolutional layer with a kernel size that covers the entire length of the phrase or sentence.

1.4 Transformer (15 pts)

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

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

ANSWER:

- Transformers don't use recurrent connecions or convlutional layers to find dependencies between input and output. Transformers use self-attention and multi-head attention.
- The Transformer architecture can process the input sequence in parallel. This increases the training speed. This is not possible in recurrent networks.
- Transformers use positional encodings to incorporate the order of the input data an capture spatial dependencies. Recurrent models an LSTM's have this in the recurrent connections itself.Positional encodings is also used to hanle variable length input.
- The Transformer is encoder-decoder architecture, where the encoder processes the input sequence and the decoder generates the output sequence.

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

ANSWER:

Self attention is a mechanism relating different positions of a single sequence to compute a representation of the sequence. It computes a weighted representation of each element in the input sequence by taking into consideration all the other elements of the sequence.
Self attention is mapping a query and a set of key-value pairs to an output. Assuming the queries and key are of dimension d_k an values are dimensions d_v. The weights are obtained by applying a softargmax function to the dot product of all keys with queries. The output is the weighted sum of the values where weights are generated as described above.
IMPORTANCE

1) Self- attention layers are aster than recurrent layers in most cases (when sequence length $n <$ representation dimensionality d)

2) In transformers, by virtue of using self-attention the entire sequence can be fed instead of element by element that is one in RNN's. This helps in parallelization within training examples which is needed in longer sequence lengths. RNN's can't do this parrallelization.

3) Self attention makes it easy to have long range dependencies in its network because the maximum path lengths in self attention models are of constant complexity - O(1) compared to O(n) in RNN's. This makes transformers more effective and efficient in handling long range dependencies which are very essential in machine translation etc.

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

ANSWER

Multi-headed attention allows the model to jointly attend to information from different representation sub-spaces at different positions.In multi-head attention, the self-attention operation is performed multiple times in parallel, each time with a different set of learned linear projections. The outputs from each of these parallel self-attention operations are concatenated and passed through a linear layer to generate the final output.

BENEFITS

1) Multi-headed attention can help generate richer and more complex representations. By computing self-attention multiple times with different sets of query, key, and value matrices, the multi-head attention mechanism can capture different aspects of the input sequence
2) The multi-headed attention model is also less sensitive to noisy data.

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

ANSWER

The Transformer model uses feed-forward neural networks (FFNNs) in the encoder and decoder layers. These feed forward networks are applied after the self attention layers. The feed forward layers are applied separately and identically to each position of the input sequence. This feed forward network consists of two linear transformations and a ReLU activation. ReLU is applied to the first linear transformation which acts as the input to the second linear transformation. The linear transformations are same across different positions but they use different parameters/weights

in the 2 different layers.
$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$
The first linear transformation increases the dimensions of the input and the second linear layer reduces it back to original dimensions.

PURPOSE

1) Non-Linearity: The ReLU activation function introduces non-linearity into the model, allowing it to capture more complex relationships between the input elements
2) This FFN helps create representations that capture local patterns and dependencies between close elements in input sequence. This helps make more accurate predictions

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

ANSWER

The layer normalization is used in both the encoder and decoder parts of the transformer. The encoder has 6 layers and each layer has 2 sublayers. Residual connection is used in each sublayer and this is followed by the layer normalization. If x is the input of the sublayer then the output of the layer normalization is
$LayerNorm(x + Sublayer(x))$
Here the addition of x acts as the residual connection. The output of the layer normalization is fed into the subsequent layers.
The decoder has 3 sublayers in each layer and similar to the encoder the Layer normalization is applied before this output is fed to subsequent layers. A residual dropout is also applied to the output of the Sublayer before the addition of residual connection and layer normalization.
The layer normalization layers helps stablize and accelerate the training process.

1.5 Vision Transformer (8 pts)

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

(a) (2 pts) 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?

ANSWERR:

CNN layers extract various features of the input image which are then

pooled and combined to generate a high level representations of the image. The image acts as input to the CNN model. In ViT, patches of the input image are generated. The linear embeddings of these patches are then inputted into the Transformer.
CNN models have locality, a 2D neighbourhood structure and translation equivariance in each layer of the model while ViT model only the MLP layers are local and translationally equivariant. In ViT the self attention layers are global.
No, the ViT architecture doesn't have a convolution layer. The ViT uses a linear projection layer on the flattened patches to generate linear embeddings. The convolutions layers use local neighbourhoods are their inputs. Hence, ViT has no traditional convolutional layers.

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

ANSWER:

1) The transformer architecture takes as input 1D sequence data like text. The actual input to the ViT architecture are 2D images which are first converted to flattened patches before being fed into the transformer unit.
2) The transformer architecture has positional encodings to make use of the order of the input sequence. This adds information about the relative or absolute position of tokens in the sequence.Positional encoding is added to the inputs of the encoder and decoder of the transformer. ViT has position embeddings added to the linear projection of flattened patches before its fed to the transformer.The positional embedding is learned.
3) ViT has 86M,307M an 632M parameters in its base, large and huge variants while the Transformer model had 65M and 213M parameters in its base and big model variant respectively.
4) The original Transformer typically uses smaller embedding sizes, such as 512 or 1024 dimensions, whereas the ViT uses larger embedding sizes, such as 768 or 1024 dimensions, because images contain more information than text data sequences and requires more complex representations.
5) The ViT uses smaller feedforward networks compared to the original Transformer. This is because images already contain rich features and do not require as much processing as textual sequences.

(c) (2 pts) 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:

Positional embeddings in ViT are added to the linear projection of flattened patches before they are fed to the transformer. They are added to give spatial information to the model.The positional embedding tells the model the position or location of each patch inside the input image. For each patch, a positional embedding is added.
The positional encodings in the original Transformer architecture add information about the relative or absolute position of tokens in the sequence. The Transformer architecture uses positional encodings which are not trainable while the positional embeddings in ViT can be trained. The positional embedding is a learned positional encoding.

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

ANSWER:

- In the ViT model, similar to the BERT's [class] token, a learnable embedding is prepended to the sequence of embedded patches.
- In the output of the Transformer encoder, the state of this class token acts as the image representation y
- This classification head is implemented by A MLP with one hidden layer at pre-training and by a single linear layer at fine-tuning.
- The output of the transformer encoder is fed into a MLP which has *tanh* as the non linearity inside the single hidden layer.

2 Implementation (50 pts)

Please add your solutions to this notebook HW4-VIT-Student.ipynb. **Please use your NYU account to access the notebooks**. 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.