# HW3 - Theory

hhp9256 - Hoang Pham

## 1 Theory

### 1.1 Attention

(a) The attention matrix is calculated as $\mathbf{H} = \mathbf{VA} = \mathbf{V}[\text{soft}]\text{argmax}_\beta(\mathbf{K}^\top \mathbf{Q})$, where $[\text{soft}]\text{argmax}_\beta$ is applied column-wise. This matrix attention is in $\mathbb{R}^{t\times n}$.

(b) Without the scaling factor $\beta = \frac{1}{\sqrt{d_k}}$ where $d_k$ is the dimension of the keys (and queries), this is done to keep the temperature constant across different choice of the model dimension $d_k$.

(c) The attention function is $\mathbf{h} = \mathbf{Va} = \mathbf{V}[\text{soft}]\text{argmax}_\beta(\mathbf{K}^\top \mathbf{q})$, therefore $\mathbf{h}$ preserves some of the value of the column vector $\mathbf{v}$ in the matrix $\mathbf{V}$ when $[\text{soft}]\text{argmax}_\beta(\mathbf{K}^\top \mathbf{q})$ is a one-hot vector. This means

- either we are using the $\arg\max$ function instead of softargmax, or
- the temperature $\beta$ is small, making the softargmax output looks like a one hot vector.

The attention we are using is called hard attention.

For a fully connected architecture, simply $\mathbf{H} = \mathbf{VI}$ where $\mathbf{I}$ is the identity matrix. This would preserve $\mathbf{v}$ in $\mathbf{h}$.

(d) Unlike part (c) above, instead of returning a $\arg\max$, softargmax returns a probability distribution of best possible value due to $\mathbf{H} = \mathbf{V}\text{softargmax}_\beta(\mathbf{K}^\top \mathbf{q})$. The value $\beta$ should be big (i.e. high temperature). This is called a soft attention.

For a fully connected layer, given an input $\mathbf{Q}$, the model is as follow:

- Fully connected layer with weight $\mathbf{K}^\top$, and no bias.
- Apply softargmax action on the first layer output.
- Another fully connected layer with weight $\mathbf{V}$ with no bias.

(e) We have

$$\hat{\mathbf{K}} = \begin{bmatrix} | & \cdots & | & \cdots & | \\ \mathbf{k}_1 & \cdots & \hat{\mathbf{k}}_i & \cdots & \mathbf{k}_m \\ | & \cdots & | & \cdots & | \end{bmatrix} = \begin{bmatrix} | & \cdots & | & \cdots & | \\ \mathbf{k}_1 & \cdots & \mathbf{k}_i + \boldsymbol{\epsilon} & \cdots & \mathbf{k}_m \\ | & \cdots & | & \cdots & | \end{bmatrix}$$

and

$$\hat{\mathbf{K}}^\top \mathbf{q} = \begin{bmatrix} - & \mathbf{k}_1 & - \\ & \vdots & \\ - & \mathbf{k}_i + \epsilon & - \\ & \vdots & \\ - & \mathbf{k}_m & - \end{bmatrix} \mathbf{q} = \left( \mathbf{K}^\top + \begin{bmatrix} - & \mathbf{0} & - \\ & \vdots & \\ - & \boldsymbol{\epsilon} & - \\ & \vdots & \\ - & \mathbf{0} & - \end{bmatrix} \right) \mathbf{q} = \mathbf{K}^\top \mathbf{q} + \boldsymbol{\epsilon}^\top \mathbf{q}$$

which yields $\mathbf{a} = \text{softargmax}(\hat{\mathbf{K}}^\top \mathbf{q}) = \text{softargmax}(\mathbf{K}^T q + \boldsymbol{\epsilon}^\top \mathbf{q})$. Since $\epsilon$ has mean 0, this new value will still has the same mean of $\mathbf{K}^\top \mathbf{q}$, and small variance. Therefore with large amount of data, by law of large number, the value of $\mathbf{a}$ and the attention $\mathbf{h}$ won't be changing a lot. And this pertubation here only affects the attention corresponding to the key $\mathbf{k}_i$, not the entire attention matrix.

(f) Similar to the transformation above,

$$\hat{\mathbf{Q}} = \begin{bmatrix} | & \cdots & | & \cdots & | \\ \mathbf{q}_1 & \cdots & \hat{\mathbf{q}}_i & \cdots & \mathbf{q}_m \\ | & \cdots & | & \cdots & | \end{bmatrix} = \begin{bmatrix} | & \cdots & | & \cdots & | \\ \mathbf{q}_1 & \cdots & \mathbf{q}_i + \boldsymbol{\epsilon} & \cdots & \mathbf{q}_m \\ | & \cdots & | & \cdots & | \end{bmatrix} = \mathbf{Q} + \begin{bmatrix} | & \cdots & | & \cdots & | \\ \mathbf{0} & \cdots & \boldsymbol{\epsilon} & \cdots & \mathbf{0} \\ | & \cdots & | & \cdots & | \end{bmatrix}$$

and

$$\mathbf{K}^\top \hat{\mathbf{Q}} = \mathbf{K}^\top \left( \mathbf{Q} + \begin{bmatrix} | & \dots & | & \dots & | \\ \mathbf{0} & \dots & \boldsymbol{\epsilon} & \dots & \mathbf{0} \\ | & \dots & | & \dots & | \end{bmatrix} \right) = \mathbf{K}^\top \mathbf{Q} + \boldsymbol{\epsilon}^\top \mathbf{k}_i$$

which yields $\mathbf{H} = \mathbf{V}\text{softargmax}_\beta \left(\mathbf{K}^\top \mathbf{Q} + \epsilon^\top \mathbf{k}_i\right)$. Pertubation in one query affect in the entire attention matrix, while pertubation in one key might only affect a single attention with respect to that query. This pertubation is causing more significant change in the result compare to the previous pertubation in part (e).

(g) We can consider masking certain inputs queries/keys in the calculation. This technique is used in the Attention Is All You Need paper to only consider already seen words and not future words. This happens before feeding output of $\mathbf{K}^T\mathbf{q}$ to the softargmax function, by masking indexes that we want to ignore to $-\infty$, so that the output probability at that index is 0.

## 1.2 Multi-headed Attention

(a) Multi-headed attention can be obtained by concatenating each head. Based on the Attention Is All You Need Paper:

$$\mathbf{H}_{\text{multihead}} = \text{Concat}(\mathbf{H}^{(1)}, \dots, \mathbf{H}^{(h)})\mathbf{W}^O,$$

where

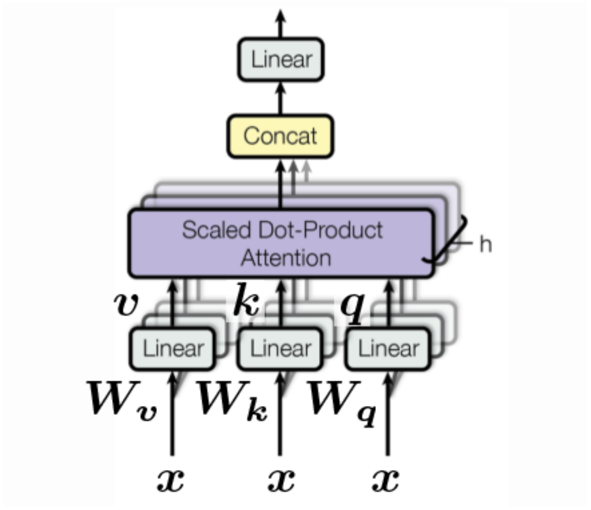
$$\begin{aligned} \mathbf{H}_i &= \text{Attention}(\mathbf{Q}\mathbf{W}_{\mathbf{q}}^{(i)}, \mathbf{K}\mathbf{W}_{\mathbf{k}}^{(i)}, \mathbf{V}\mathbf{W}_{\mathbf{v}}^{(i)}) \\ &= (\mathbf{V}\mathbf{W}_{\mathbf{v}}^{(i)})\text{softargmax}_\beta \left( (\mathbf{K}\mathbf{W}_{\mathbf{k}}^{(i)})^\top \mathbf{Q}\mathbf{W}_{\mathbf{q}}^{(i)} \right) \\ &= \mathbf{V}^{(i)}\text{softargmax}(\mathbf{K}^{(i)\top}\mathbf{Q}^{(i)}), \end{aligned}$$

with $\mathbf{V}^{(i)} = \mathbf{V}\mathbf{W}_{\mathbf{v}}^{(i)}, \mathbf{Q}^{(i)} = \mathbf{Q}\mathbf{W}_{\mathbf{q}}^{(i)}, \mathbf{K}^{(i)} = \mathbf{V}\mathbf{W}_{\mathbf{v}}^{(i)}$.

(b) In the context of ConvNet, multi-headed attention is similar to multi-filter ConvNet, where the number of output channels are similar to the number of heads. In ConvNet, each filter is applied on the input and then combined together to form a new multi-channel input. Similarly, each head generates an attention, and each attention is combined to form a final attention matrix.

## 1.3 Self Attention

(a) Self-attention is basically a regular attention model with the output dimension is the same as the input dimension. Specifically, The self-attention block accepts a set of inputs, from $x_1, \dots, x_t$, and outputs $z_1, \dots, z_t$ attention weighted values which are fed through the rest of the encoder.



There are some conceptual benefits to self-attention:

- Contextual attention: tokens can attend to every other token in the sequence.

- Interpretable: due to contextual attention, we can understand better how one token attend to the other in the sequence.
- Easier to parallelize the model during training.

There are also some drawbacks:

- More hyper-parameter tuning to get the number of heads right.
- Lacks inherent sequential bias, leads to the use of positional encoding, which is also another hyper-parameter that requires tuning.
- Quadratic complexity with respect to the sequence length, due to scaled dot product attention.

(b) Positional encoding is a layer to embed position of the input token in the sequence, can be relative and absolute position. Positional encoding is needed so that the model can make use of the order in the sequence. The Attention/Transformer model is not recurrent or sequential, so it cannot understand position in a sequence without a positional encoding.

(c) Similar to what is showed in earlier parts, we can setup the self-attention model as follow to achieve a permutation/identity layer:

- Use $\arg\max$ or $\text{softargmax}_\beta$ with very small $\beta$, so that we achieve one-hot encoding output.
- As shown in part 1.1 (c), this hard self-attention preserves the value $\mathbf{v}$ in the attention $\mathbf{h}$, which makes the output of the attention an identity or a permutation layer.

(d) A convolutional filter resembles a self-attention model when the attention is skewed towards its neighbor tokens. The attention weight can be learned to constrain around its neighboring pixel (in the context of image processing), or token (in the context of language modeling). By restricting this, the model will force the self-attention layer to behave like a convolution layer with variable length $n > 1$.

## 1.4 Transformer

(a) There are several differences:

1. Information flow and sequential information: transformer architecture has the attention concept, allowing the model to focus to different parts of the input sequence, therefore letting the model attend to both local and long-range dependencies. Previous seq2seq models like LSTM or RNNS only process inputs sequentially, thus are able to capture local dependencies, but perform poorly on long-range dependencies.
2. Transformer model allows parallelization in training, making it more efficient to utilize underlying hardware. Traditional seq2seq model cannot utilize parallelization due to sequential token processing.
3. Layer complexity in transformer model however is higher, due to the self-attention complexity of $O(n^2)$ where $n$ is the input length. Classic seq2seq model doesn't have this layer complexity.

(b) There are several benefit of self-attention in Transformer model and machine translation task, similar to the previous part:

1. Allows the model to focus on both long-context attention and local attention in the sequence. This is especially useful for language model, as text is sequence based and require context.
2. Parallelization in training, unlike regular recurrent networks which process token by token sequentially.
3. Self-attention mechanism is more adaptive. The attention weights can be adjusted per input sequence and so it can be adapted to different input contexts. This means the transformer model can be scaled to longer sequence and more complicated inputs without significantly increasing computational complexity.
4. Self-attention makes the model more interpretable, by seeing how each token attend to others.

(c) The feed forward layer is used after attention layer to introduce non-linearity to the model, therefore allowing it to capture additional complexity and non-linear relationships within each position of the input sequence. This also allows the model to be more flexible in learning different representations per position, which makes the model more adaptive to different patterns.

(d) The techniques used to improve training stability regarding exploding/vanishing gradients are:

1. LayerNorm: unlike BatchNorm, which performs normalization for the entire batch, LayerNorm performs normalization on the each output. Therefore it doesn't introduce covariate shift among training batches.
2. Residual connection: this also happens with the batch normalization, which allows the gradient to flow directly through the network. This helps stabilize training and convergence, and allows for the training of very deep networks.

### 1.5 Vision Transformer

(a) ConvNet uses filters/kernels to detect local patterns and spatial hierarchies through weight sharing. ViT uses a ConvNet with size $P \times P$, where $P$ is the patch size, to tokenize the image and process it as a sequence, with positional encoding, to leverage the Transformer architecture.

(b) The main differences between ViT and the traditional Transformer model are:

- ViT operates on 2D input, therefore it needs to transform input into sequential tokens using convolutional patches.
- Positional encoding: ViT needs to encode position a little different, using positional embedding to encode 2D positions instead of sinusoidal positional encoding for 1D input in the regular Transformer model.

(c) As explained in previous parts, the ViT generates the final output based on the following steps:

- Patch embedding - tokenize the image as token sequence with positional encoding.
- Using the classical transformer model on the tokenize image input. This includes LayerNorm, Multi-head attention, another LayerNorm, and a fully connected Multi-Layer Perceptron layer, a residual layer, and repeat for $L$ times.
- Then the output is passed through a softargmax activation function to get the final output.

(d) On small datasets, as reported in both the ipynb and the paper, ConvNet outperforms ViT, due to ConvNet model has strong inductive, induced by the kernels that better learn local, sub-patterns within the images. ViT doesn't possess such property, but training on large scale data trumps inductive bias, as it requires more data for the model to recognize patterns in the tokenize sequences. With more data, ViT outperforms ConvNet due to its ability to capture complex and global relationships.