Data C182 Designing, Visualizing & Understanding DNN
Fall 2024 Eric Kim, Naveen Ashish Discussion 06

This discussion covers self-attention mechanism, multi-head attention, encoder architecture, attention vs CNN, and Cross attention.

1. Attention Mechanisms

For many NLP and visual tasks we train our deep models on, features appear on the input text/visual data often contributes unevenly to the output task. For example, in a translation task, not the entirety of the input sentence will be useful (and may even be confusing) for the model to generate a certain output word, or not the entirety of the image contributes to a certain sentence generated in the caption.

While some RNN architectures we previously covered possess the capability to maintain a memory of the previous inputs/outputs, to compute output and to modify the memory accordingly, these memory states need to encompass information of many previous states, which can be difficult especially when performing tasks with long-term dependencies.

Attention mechanisms were developed to improve the network's capability of orienting perception onto parts of the data, and to allow random access to the memory of processing previous inputs. In the context of RNNs, attention mechanisms allow networks to not only utilize the current hidden state, but also the hidden states of the network computed in previous time steps as shown in the figure below.

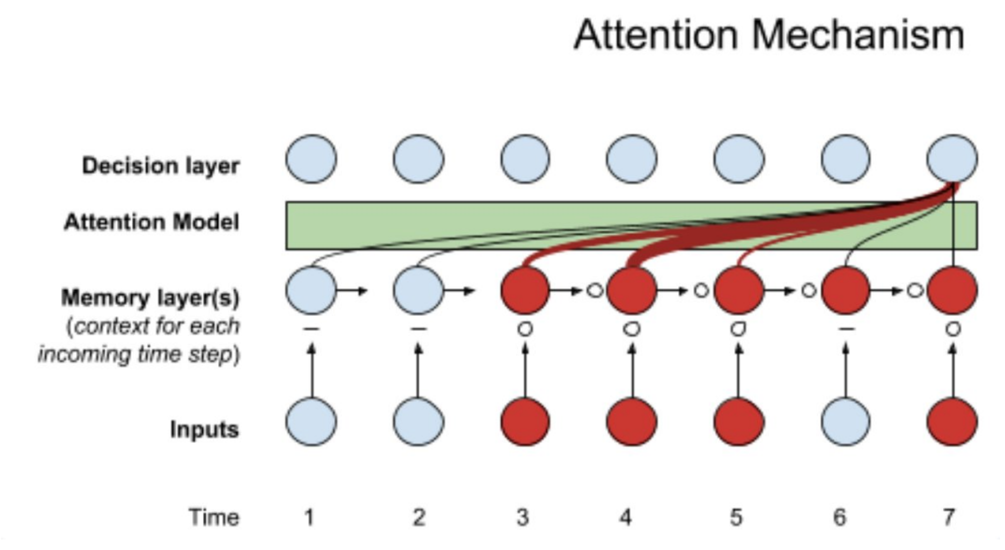

Figure 1: Attention mechanism

0.1 Self-Attention in Transformer Networks

Self attention is an attention mechanism introduced in the Transformer architecture. The first step of the attention is to compute Q, K, V using different transformations from the original input embedding as shown in the figure below.

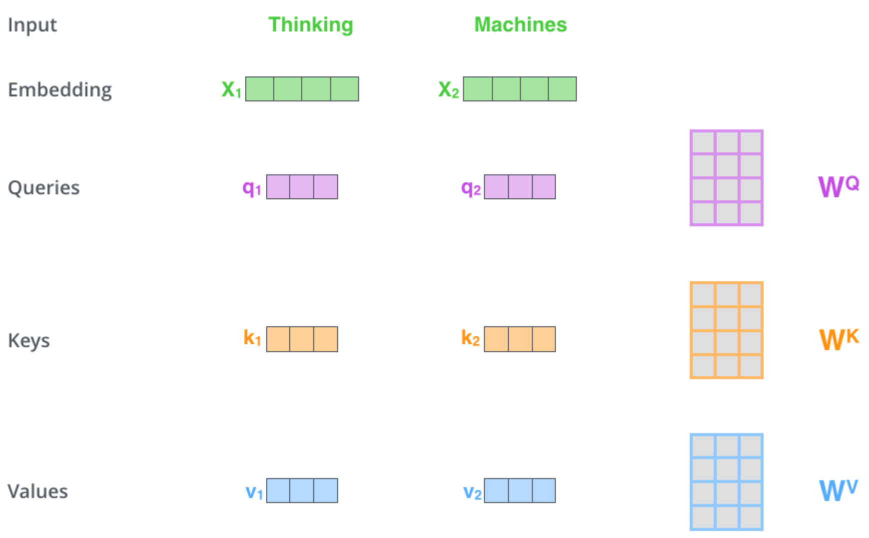

Figure 2: Computing K, Q, V from input embeddings in a Transformer Network.

Then, using Q and K, we can compute a dot product as the 'score' of K for Q as shown in the figure below. Intuitively, Q is the querying term that you would like to find. Its relations for each corresponding K and V pairs (key-value) pairs, can be computed using the key. Note that this dot product is computed across various time steps by matrix multiplication. So we get a score for each K for each Q. We then use a Softmax function to get our attention weights.

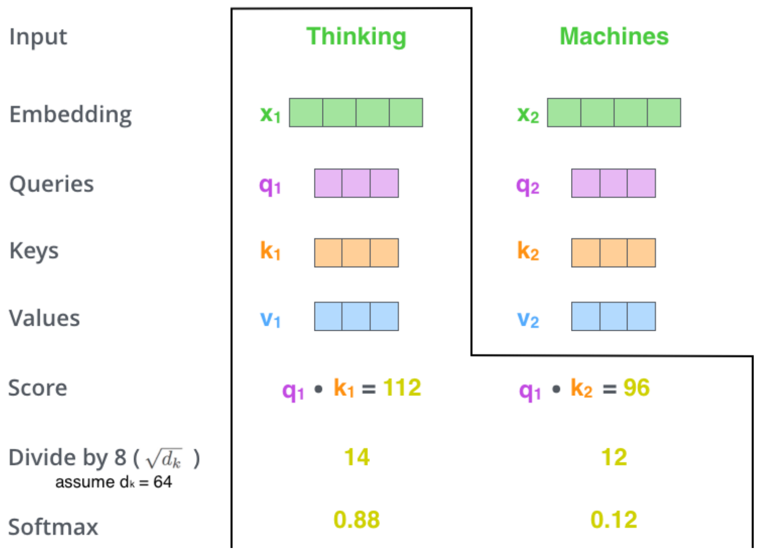

Figure 3: Computing Attention Scores from K, Q, V

Finally, using these weights, we can compute our weighted sum by multiplying the weights with the values. You may find the follow figure helpful.

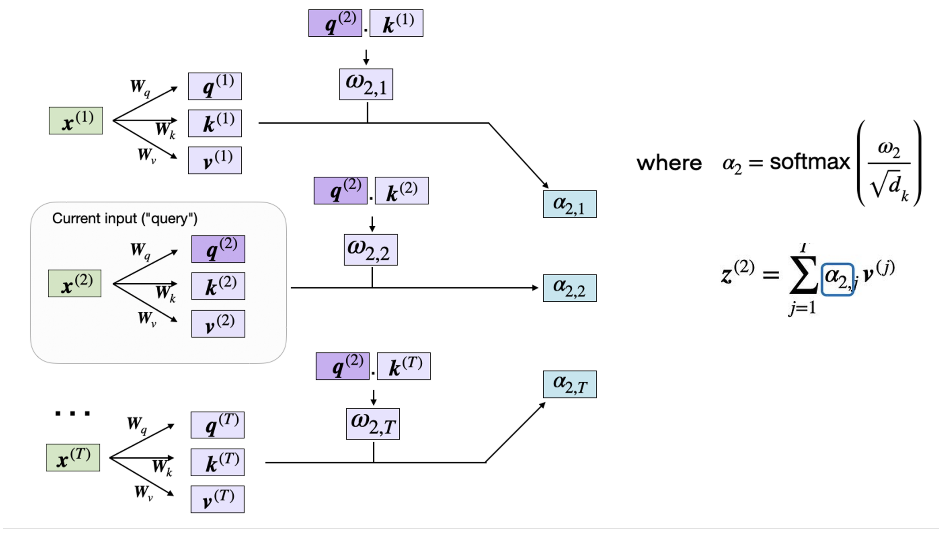

Figure 4: Attention operation

(a) **Problem: Generalized Attention in Matrix Form**

Consider a form of attention that matches query q to keys $k_1, \dots, k_t$ in order to attend over associated values $v_1, \dots, v_t$.
If we have multiple queries $q_1, \dots, q_t$, how can we write this version of attention in matrix notation?

Solution:

Solution: Generalized Attention in Matrix Form

Stack queries into a matrix Q, keys into K and values V. Then,

$$a(Q, K, V) = \text{Softmax}(QK^\top)V$$

where Softmax is applied row-wise. Note that here, Q, K, V are all matrices of shape [t, d], where t is the number of tokens in the sequence, and d is the hidden dimension.

Problem: Justifying Scaled Self-Attention

Given a well-initialized neural network, activations should be on the order of $O(1)$. Consequently, entries in the query (q), key (k), and value (v) vectors are also $O(1)$. A well-initialized network avoids collapsing its representations into a low-dimensional subspace. Thus, the entries of q, k, and v can be considered approximately independent random variables, perhaps drawn from normal distributions.
We estimate the order of magnitude of $Var[q \cdot k]$, assuming entries of q and k are independent samples from a normal distribution $\mathcal{N}(\mu, \sigma)$, where $\mu \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}^+$

(b) i. Asymptotically, how does $Var(q^\top k)$ scale with d? Use big-O notation. Why might this be problematic?

ii. Suppose we scale the dot product by $\frac{1}{s}$, i.e. $Var(q^\top k/s)$. What value of s should we choose to address the issue from 3.?

iii. (optional) Calculate the exact value of $\mathbb{E}[q^\top k]$ in terms of μ, σ, d

iv. (optional) Calculate the exact value of $Var(q^\top k)$ in terms of μ, σ, d

Solution:

Solution: Justifying Scaled Self-Attention

i. Since q_i and k_i are independent, each term $q_i k_i$ has the same mean and variance. Because the dot product $q \cdot k$ is a sum of d such independent terms, the variance is additive:

$Var[q \cdot k] = Var[\sum_{i=1}^d q_i k_i] = \sum_{i=1}^d Var[q_i k_i] = d \times Var[q_1 k_1]$.

If q_1 and k_1 are independent and normally distributed with mean μ and standard deviation σ, then $Var[q_1 k_1] = \sigma^2(\mu^2 + \sigma^2)$. Assuming μ and σ are $O(1)$, $Var[q_1 k_1]$ is also $O(1)$. Therefore, $Var[q \cdot k] = d \times O(1) = O(d)$.

Since $Var(q^\top k) = O(d)$, as d grows larger, the variance increases. Given that this is immediately the input to a softmax, which is known to have issues with vanishing gradients given large inputs, this is problematic.

ii. With the scaling term, the variance becomes $\frac{1}{s^2}(2d\sigma^2\mu^\top\mu + d\sigma^4)$. We would like $s = \sqrt{d}$ for the variance to no longer be linear in d.

iii. While these derivations may be somewhat out of scope for Data C182 (Fall 2024), we do want you to understand the argument why $Var()$ being a function of d is bad, and how dividing by $\sqrt{d}$ fixes it.

$$\begin{aligned} \mathbb{E}[q^\top k] &= \mathbb{E}\left[\sum_{i=1}^d q_i k_i\right] \\ &= \sum_{i=1}^d \mathbb{E}[q_i k_i] \end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=1}^{d} \mu_i^2 \\
&= \mu^\top \mu
\end{aligned}
$$

iv. First, notice that if random variables are uncorrelated, then, we have

$$
Var\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} Var(X_i)
$$

Then,

$$
\begin{aligned}
Var(q^\top k) &= \mathbb{E}[(q^\top k)^2] - \mathbb{E}[q^\top k]^2 \\
&= \mathbb{E}[q^\top k k^\top q] - (\mu^\top \mu)^2 \\
&= \mathbb{E}[Tr(q q^\top k k^\top)] - (\mu^\top \mu)^2 \\
&= Tr(\mathbb{E}[q q^\top]\mathbb{E}[k k^\top]) - (\mu^\top \mu)^2 \\
&= Tr\left((\mathbb{E}[q]\mathbb{E}[q^\top] + \sigma^2 I)(\mathbb{E}[k]\mathbb{E}[k^\top] + \sigma^2 I)\right) - (\mu^\top \mu)^2 \\
&= Tr\left((\mu\mu^\top + \sigma^2 I)(\mu\mu^\top + \sigma^2 I)\right) - (\mu^\top \mu)^2 \\
&= Tr(\mu\mu^\top\mu\mu^\top) + 2\sigma^2 Tr(\mu\mu^\top) + Tr(\sigma^4 I) - (\mu^\top \mu)^2 \\
&= \mu^\top\mu\mu^\top\mu + 2\sigma^2\mu^\top\mu + d\sigma^4 - (\mu^\top \mu)^2 \\
&= 2d\sigma^2\mu^\top\mu + d\sigma^4
\end{aligned}
$$

Problem: Non-local Means Interpretation of Attention

Given a dataset of $(x, y)_i$, a non-parametric method for estimating the value for arbitrary x is via "averaging" the y of the nearest data points to x.

$$
y = \sum_i \frac{K(x, x_i)}{\sum_j K(x, x_j)} y_i
$$

Where $K(x, x_i)$ is the "similarity" between x and x_i, and y is computed as the weighted average. How does this relate to transformer, specifically, the q, k, v matrix?

Solution:

Solution: Justifying Scaled Self-Attention

We can rewrite the attention formulation as

$$
\begin{aligned}
&\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \\
&\text{out}_i = a_{ij} v_j
\end{aligned}
$$

$$a_{ij} = \text{softmax}(\frac{q_i k_j^\top}{\sqrt{d_k}}) = \frac{\exp\left(q_i k_j^\top / \sqrt{d_k}\right)}{\sum_r \exp\left(q_i, k_r^\top / \sqrt{d_k}\right)}$$

Here, $\exp\left(q_i k_j^\top\right)$ captures the "similarity" between query element i and key element j.

0.2 Multi-Headed Attention

The multi-head self-attention module is a key component in Transformer. Rather than only computing the attention once, the multi-head mechanism splits the inputs into smaller chunks and then computes the scaled dot-product attention over each subspace in parallel. The independent attention outputs are simply concatenated and linearly transformed into expected dimensions.

$$\begin{aligned}
\mathbf{X}_0 &= \mathbf{X}[:, : d_k] \\
\mathbf{X}_1 &= \mathbf{X}[:, d_k : 2d_k] \\
\mathbf{X}_2 &= \mathbf{X}[:, 2d_k : 3d_k] \\
&\ldots\ldots \\
\text{head}_i &= \text{Attention}(\mathbf{X}_i \mathbf{W}_i^q, \mathbf{X}_i \mathbf{W}_i^k, \mathbf{X}_i \mathbf{W}_i^v) \\
\text{MHA}(\mathbf{X}, \mathbf{W}^q, \mathbf{W}^k, \mathbf{W}^v) &= \text{Concat}[\text{head}_1; \ldots; \text{head}_h]\mathbf{W}^o
\end{aligned}$$

Here X_i is the i-th split of size d_k along the embed dimension where $d_k = \frac{d}{h}$.

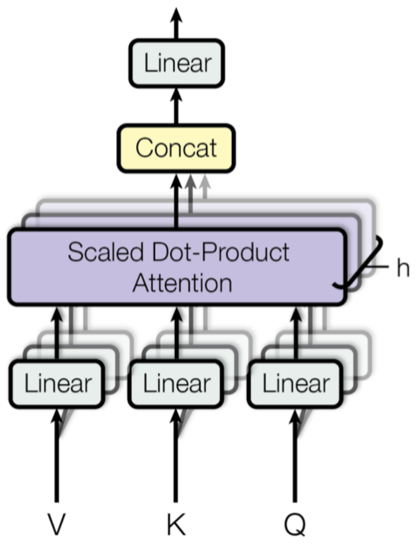

Figure 5: Multi-headed attention

Problem: Properties of Multi-head attention (MHA)

(c) i. We want to change single-head attention to multi-head attention. The original single-head attention has hidden dimension of d, d_k keys, d_q queries and d_v values. What is the relationship between d_k and d_q?

ii. Suppose the sequential input that the user feeds in is of length n What is the total computational cost of self attention operation? Only consider matrix multiplications. Assume the computation is the product of the number of elements in the two matrices. You may find it helpful to walk through the following diagram.

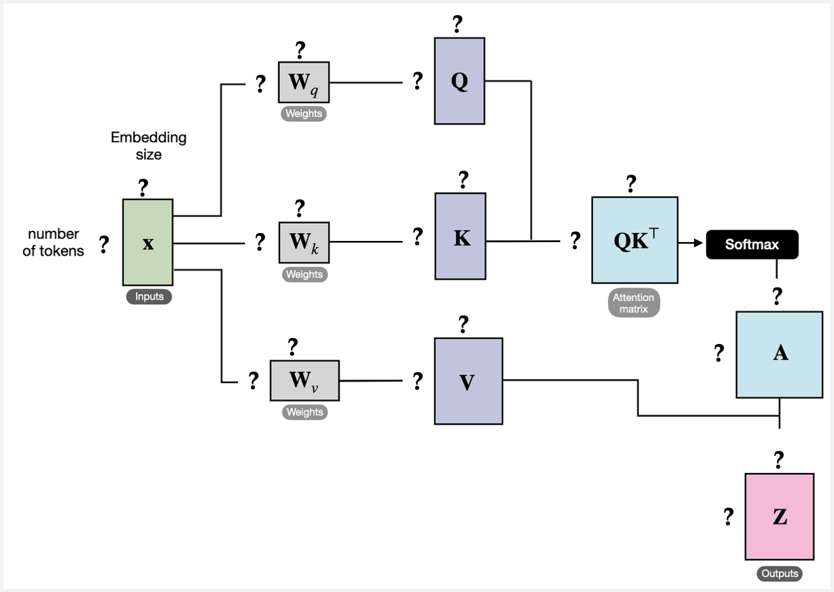

Figure 6: Self attention

iii. Now we want to maintain the same number of parameters, and use h heads. How many keys, queries and values will there be? What is the computational advantage of using MultiheadAttention?

Solution:

Problem: Properties of Multi-head attention

i. Due to the multiplication between keys and queries, they need to match in their number. $d_k = d_q$.

ii. The computation is $O(nd(2d_q + d_v)) + O(d_q n^2) + O(n^2 d_v)$. The computational bottle neck is n^2, which is quadratic to the number of elements in the input, which limits the capability of the attention to scale to very long sequences. In theory, you can design an MHA block where $d_v \neq d_q$. But in practice, we usually set $d_v = d_q$ so that it results in (1) a simpler architecture, and (2) we can enable certain computation optimizations, see pytorch's torch.nn.MultiheadAttention docstring.

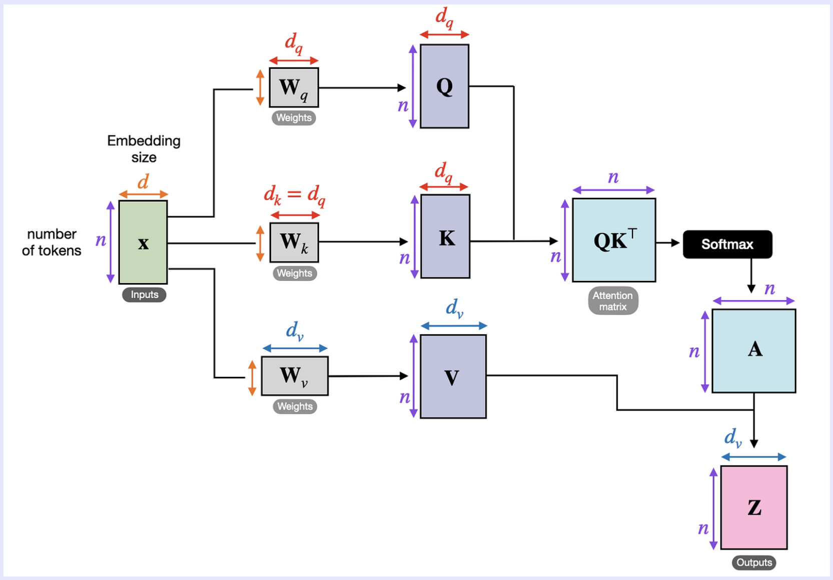

Figure 7: Self attention

iii. We effectively reduce the size $d_k \to \frac{d_k}{h}$, and $d_q \to \frac{d_q}{h}$, $d_v \to \frac{d_v}{h}$. Assume matmul between two matrices with shape $[N, D], [D, M]$ is $O(NDM)$.

The computational cost of attention is dominated by the attention score calculation and weighted value aggregation. Given sequence length n and key, query, and value dimensions d_k, d_q, d_v, the cost is $O(nd_kn) + O(n^2d_v)$. Assuming $n \geq d$ and $d_q = d_k$, this simplifies to $O(n^2(d_k + d_v))$. Dividing and multiplying by a constant h doesn't change this asymptotic complexity. Therefore there is no computational advantage to using multi-head attention in terms of computational complexity.

However, multi-head attention allows for parallelization of the attention computation across the heads, which can be beneficial for running on GPUs. Some research have also shown that the heads can specialize in different aspects of the input, which can improve the model's performance. For example see *Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned*.

A similar phenomenon can be observed in mixture-of-experts models, where different experts specialize in different aspects of the input data. See for example *St-moe: Designing stable and transferable sparse expert models*, Table 13: *We find experts that specialize in punctuation, conjunctions and articles, verbs, visual descriptions, proper names, counting and numbers.*

2. Encoder Architecture

Among the famous transformer-based architectures, encoder-only transformers are popular for image classification tasks. Below are from the famous BERT and ViT paper

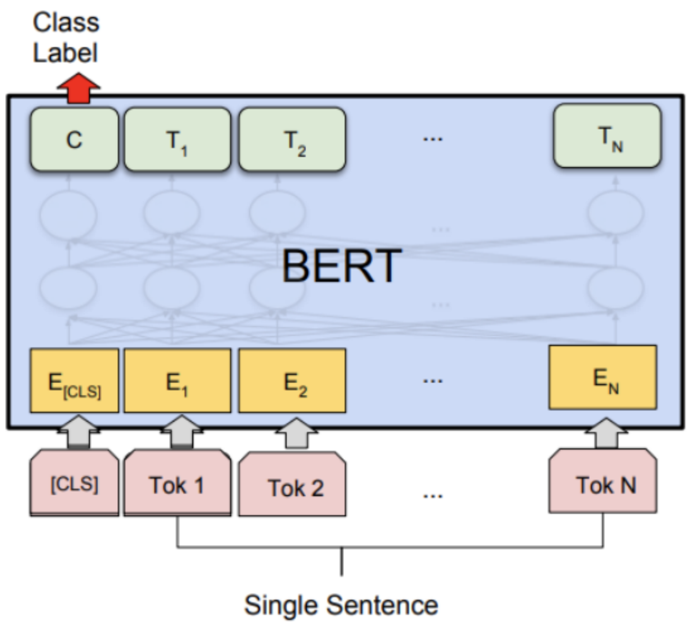

Figure 8: Encoder: BERT

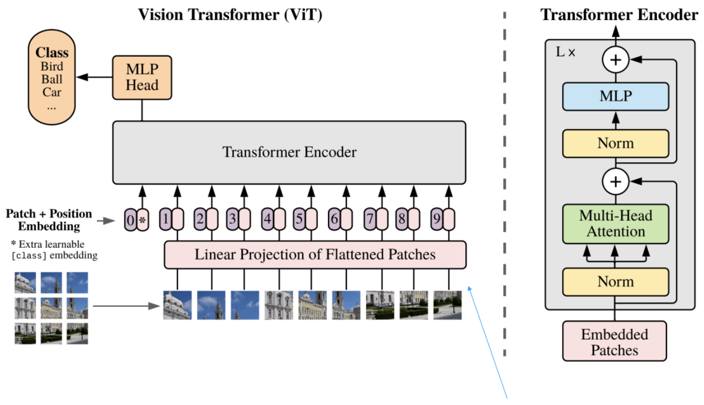

Figure 9: Encoder: ViT

Encoder Architecture

3. (a) Explain how classification works for encoder-only architectures

(b) How does transformer-based encoder compare to CNN?

Solution:

Encoder Architecture

(a) The transformers, by default are sequence to sequence networks. As there is no decoder layer in ViT, then the length of input sequence (number of patches) equals the length of output sequence. So If the goal is classification, there is two choices:

i. Either apply a fully connected layer on top of the transformer (which is not a good idea because then we have to fix the number of patches–which translates to input image resolution)

ii. Or apply the classification layer on one items of the output sequence, but which one?! The best answer here is none of them! We don't want to be biased toward any of the patches. One idea here is to add a dummy input, call it class token and apply the classification layer on the corresponding output item!

Since then, this method has been obsoleted by better methods. One method is Global Average Pooling (GAP), which just takes the average of all the output vectors, then apply the classification layer. Another method is Multihead Attention Pooling (MAP), which is essentially a small cross-attention module. We will implement a ViT using MAP in homework 4.

(b) CNN models are traditionally known for their compact size and efficient memory utilization, making them suitable for resource-constrained environments. Due to their inductive bias, they perform better than ViT using fewer number of training images. On the other hand, Vision Transformers offer a powerful approach to capture global dependencies and contextual understanding in images. The downside of vision transformer is that due to having less inductive bias, they require bigger models and much larger datasets to train.

For instance, the original ViT paper used JFT-300M, a dataset with 300 million images, which is 20x bigger than ImageNet. The authors of the ViT paper state that:

When trained on mid-sized datasets such as ImageNet without strong regularization, these models yield modest accuracies of a few percentage points below ResNets of comparable size. This seemingly discouraging outcome may be expected: Transformers lack some of the inductive biases inherent to CNNs, such as translation equivariance and locality, and therefore do not generalize well when trained on insufficient amounts of data. However, the picture changes if the models are trained on larger datasets (14M-300M images). We find that large scale training trumps inductive bias. Our Vision Transformer (ViT) attains excellent results when pre-trained at sufficient scale and transferred to tasks with fewer datapoints.

4. Cross Attention

What is cross-attention, and how does it differ from self-attention?

In self-attention, we work with the same input sequence. In cross-attention, we mix or combine two different input sequences. Note that in cross-attention, the two input sequences x_1 and x_2 can have different numbers of elements. However, their embedding/hidden dimensions must match.

Cross attention is useful for fusing together different sources of information, for instance, combining text and image data, or in autonomous driving, combining vehicle information with the roadgraph information. In machine translation, one can also combine sentences of two different language with cross attention.

Cross Attention

(a) i. Fill in the dimension in the diagram.

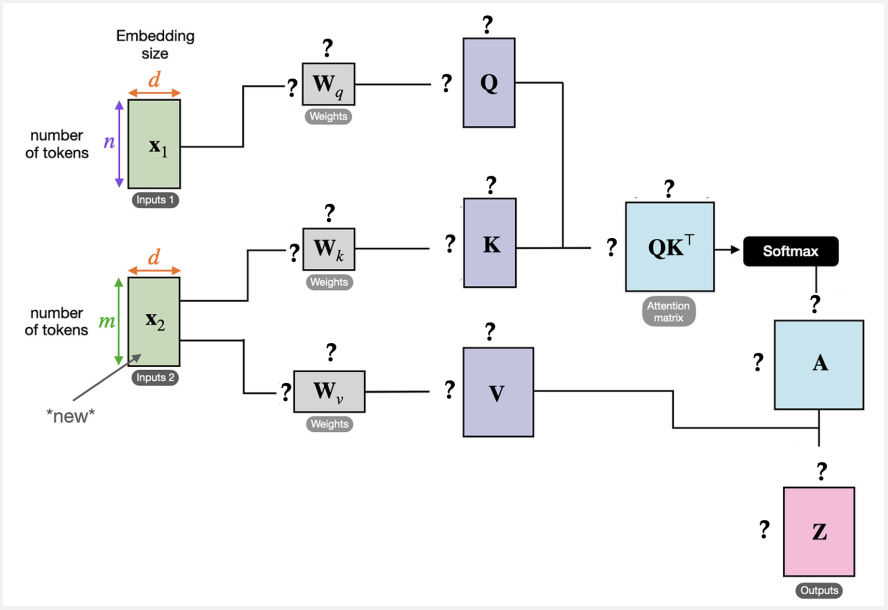

Figure 10: Cross Attention mechanism

Solution:

Cross Attention

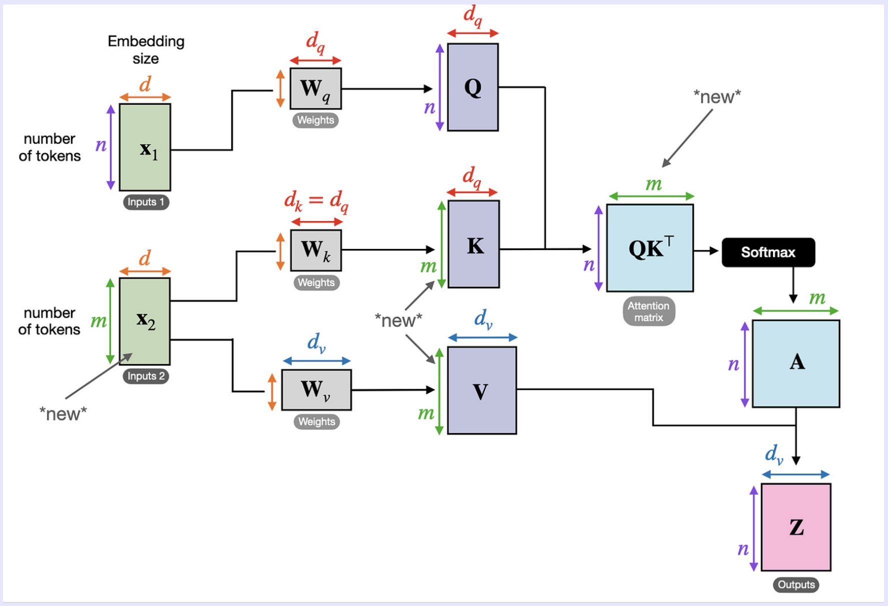

Figure 11: Cross Attention mechanism