EECS 182 Deep Neural Networks
Spring 2023 Anant Sahai

Final Review: Transformers

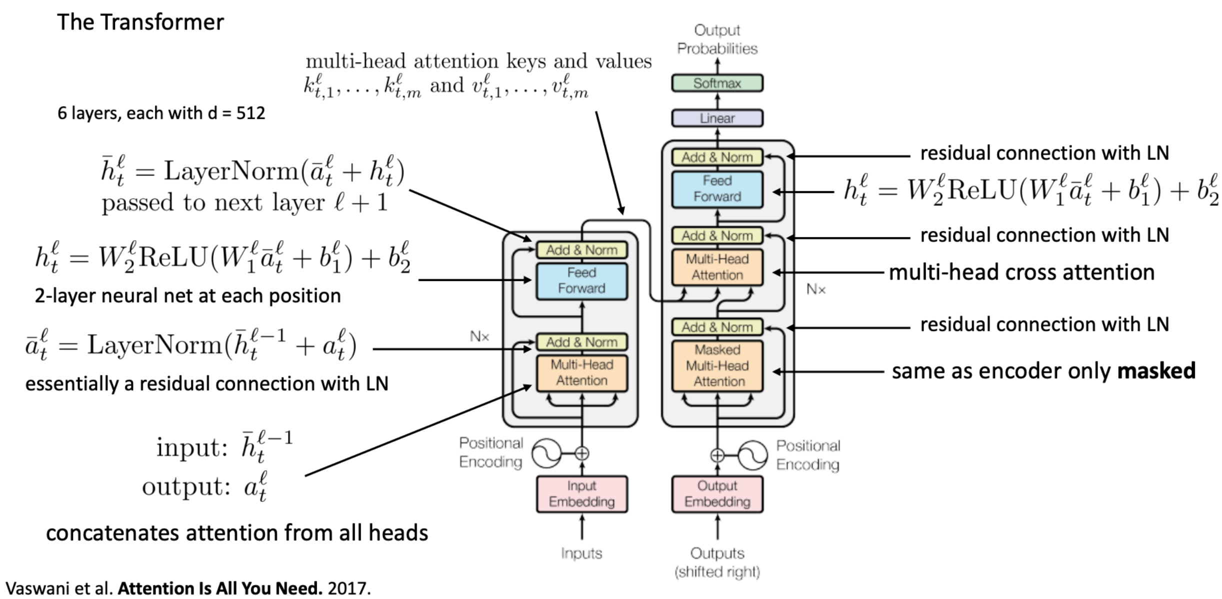

Figure 1: The diagram of the Transformer architecture.

Figure 1 shows the diagram of the Transformer architecture introduced in *Attention is All You Need*.

1. Scaled Dot-Product Attention

```
1  def scaled_dot_product_attention(q, k, v,
2          key_padding_mask=None, causal=False):
3      d_head = q.size(-1)
4      s = (einops.einsum(q, k, "n tl dh, n sl dh -> n tl sl")
5          / d_head ** 0.5)
6      if key_padding_mask is not None:
7          s = s.masked_fill(
8              key_padding_mask.unsqueeze(1).to(torch.bool),
9              float("-inf"),
10         )
11     if causal:
12         attn_mask = future_mask[: s.size(1), : s.size(2)].to(s)
13         s += attn_mask.unsqueeze(0)
14     a = F.softmax(s, dim=-1, dtype=torch.float32).type_as(s)
15     return einops.einsum(a, v, "n tl sl, n sl dh -> n tl dh")
```

(a) In scaled-dot product attention, **why do we divide pre-softmax attention scores by $\sqrt{d_{\textbf{head}}}$** (line 5), and **what would be the consequence of not doing so**. **Prove your arguments mathematically**, assuming the input tensor elements are i.i.d. and have a mean of 0 and standard deviation of 1?

Solution: Let consider a single query vector $\mathbf{q} \in \mathbb{R}^{d_{\text{head}}}$ and n key vectors $\mathbf{k}_1, \ldots, \mathbf{k}_n \in \mathbb{R}^{d_{\text{head}}}$. Let pre-softmax attention score $s_i = \mathbf{q}^T \mathbf{k}_i$. Compute the mean and variance of each s_i:

$$\mathbb{E}(s_i) = \sum_{j=1}^{d_{\text{head}}} \mathbb{E}(q_i k_{i,j}) = \sum_{j=1}^{d_{\text{head}}} \mathbb{E}(q_i)\mathbb{E}(k_{i,j}) = 0$$

$$\text{Var}(s_i) = \mathbb{E}(S_i^2) - \mathbb{E}(S_i)^2 = \sum_{j=1}^{d_{\text{head}}} \mathbb{E}(q_i^2 k_{i,j}^2) - 0 = \sum_{j=1}^{d_{\text{head}}} \mathbb{E}(q_i^2)\mathbb{E}(k_{i,j}^2) = d_{\text{head}}$$

Let $\mathbf{a} = \text{softmax}(\mathbf{s})$ represent the post-softmax attention scores. The output distribution of softmax becomes sharper with increasing input scale. Given that $\text{Var}(s_i)$ is proportional to d_{head}, for any $\epsilon > 0$, a sufficiently large d_{head} can be found such that $a_{i_{\text{max}}} > 1 - \epsilon$ for the entry i_{max} with the highest pre-softmax score, while all other elements i satisfy $a_i < \epsilon$.

We know that the Jacobian of softmax is:

$$\frac{\partial \mathbf{a}^T}{\partial \mathbf{s}} = \text{diag}(\mathbf{a}) - \mathbf{a}\mathbf{a}^T$$

Its squared Frobenius norm is:

$$\begin{aligned}
\|\frac{\partial \mathbf{a}^T}{\partial \mathbf{s}}\|_F^2 &= \|\text{diag}(\mathbf{a}) - \mathbf{a}\mathbf{a}^T\|_F^2 \\
&= \sum_{i=1}^{n} a_i^2 (1 - a_i)^2 + 2 \sum_{1 \le i < j \le n} a_i^2 a_j^2 \\
&\le (1 - a_{i_{\text{max}}})^2 \cdot 1^2 + \sum_{i \ne i_{\text{max}}} a_i^2 \cdot 1^2 + 2 \sum_{1 \le i < j \le n} \min\{a_i, a_j\}^2 \cdot 1^2 \\
&< \epsilon^2 + (n-1)\epsilon^2 + 2\frac{n(n-1)}{2}\epsilon^2 \\
&\le n^2 \epsilon^2
\end{aligned}$$

It means that the Jacobian matrix will also go infinitely small, causing the *vanishing gradient* gradient.

2. Multi-head Attention

```python
def forward(self, q, k, v, key_padding_mask=None, causal=False):
    q = self.q_proj(q)
    k = self.k_proj(k)
    v = self.v_proj(v)
    q = einops.rearrange(q, "b tl (nh dh) -> (b nh) tl dh",
        nh=self.n_heads)
    k = einops.rearrange(k, "b sl (nh dh) -> (b nh) sl dh",
        nh=self.n_heads)
    v = einops.rearrange(v, "b sl (nh dh) -> (b nh) sl dh",
        nh=self.n_heads)
    if key_padding_mask is not None:
        key_padding_mask = einops.repeat(
```



```
13              key_padding_mask, "b sl -> (b nh) sl",
14                  nh=self.n_heads)
15      o = scaled_dot_product_attention(q, k, v, key_padding_mask, causal)
16      o = einops.rearrange(o, "(b nh) tl dh -> b tl (nh dh)",
17          nh=self.n_heads)
18      return self.o_proj(o)
```

(a) Let's review the rationale behind multi-head attention. Given that softmax typically exhibits unimodal behavior, it can be approximated by argmax attention. **Determine the receptive field size of a node at layer n for the following scenarios:**

(i) With a single head.

(ii) With two heads.

(iii) With k heads.

Solution: With only a single head, we only have attention with one other time step (ie. the key vector), so with the residual connection in the transformer block, a branching factor of 2 at each level. Hence total size is 2^n.

With two heads, each hidden state can pay attention to itself and two other hidden states, so we have a branching factor of 3. Total size of receptive field is 3^n.

Similarly, with k heads, size of the receptive field is $(k+1)^n$

(b) In NLP, a batch of sentences typically contains sequences of varying lengths, requiring padding to match the longest sentence. To prevent these pad tokens from affecting computation, we apply *key padding masks* and *causal masks* to attentions. **Describe how these masks are applied in each of the following scenarios** (applied to which multi-head attention modules in which Transformer stack):

(i) Transformer encoder (e.g., BERT) tarined for text classification.

(ii) Transformer decoder (e.g., GPT-3) trained for sequence generation.

(iii) Transformer encoder-decoder (e.g., T5) trained for machine translation.

Solution:

(i) In Transformer encoder (e.g., BERT) tarined for text classification. Only key padding mask is applied to encoder self-attention.

(ii) In Transformer decoder (e.g., GPT-3) trained for sequence generation. Only causal mask is applied to self-attention. Note that if we do padding on the right (which is the usual case), key padding mask is not needed when there is causal mask.

(iii) In Transformer encoder-decoder (e.g., T5) trained for machine translation, key padding mask is applied to encoder self-attention and decoder-encoder cross-attention. As for decoder self-attention, causal mask is applied, and key padding mask is not needed as long as we are padding on the right.

(c) **Determine the asymptotic time complexity of multi-head attention** as a function of key/value length n_s, query length n_t, head dimension d_{head}, and the number of heads h. Ignore key padding masks and causal masks.

Solution: Let's go through the code line by line. Note that $d_{\text{model}} = d_{\text{head}} h$

Line 2: $\Theta(n_t d_{\text{model}}^2)$

Line 3, 4: $\Theta(n_s d_{\text{model}}^2)$

Line 5: $\Theta(n_t d_{\text{head}} h)$

Line 6, 7: $\Theta(n_s d_{\text{head}} h)$

Line 15: Let's step into `scaled_dot_product_attention`

- Line 4-5: $\Theta(h n_t n_s d_{\text{head}})$
- Line 14: $\Theta(h n_t n_s)$
- Line 15: $\Theta(h n_t n_s d_{\text{head}})$

Line 16-17: $\Theta(h n_t d_{\text{head}})$

Line 18: $\Theta(n_t d_{\text{model}}^2)$

So the total time complexity is $\Theta(n_t d_{\text{head}}^2 h^2 + n_s d_{\text{head}}^2 h^2 + n_t n_s d_{\text{head}} h)$

This also equals to $\Theta(n_t d_{\text{model}}^2 + n_s d_{\text{model}}^2 + n_t n_s d_{\text{model}})$

(d) Based on your analysis, **identify the computational efficiency bottleneck for the following scenarios**:

(i) When d_{model} is large but sequences are short.

(ii) When sequences are long but d_{model} is small.

Solution:

(i) When d_{model} is large but sequences are short, the bottleneck is line 2, 3, 4, 18 of multi-head attention, which is the query/key/value/output projections.

(ii) When sequences are long but d_{model} is small, the bottelneck is line 4, 15 of scaled dot-product attention: computing attention scores and linear combination of values according to attention scores, respectively.

3. Layer Normalization

Examine the Transformer diagram, which includes an "add and norm" layer after each multi-head attention or feed-forward module. The "add" represents a residual connection, inspired by ResNet. This question serves as a review of layer normalization.

(a) Consider an input tensor $\mathbf{X}$ of shape $[B, D]$, where B is the batch size and D is the hidden state dimension. Layer normalization is applied to obtain output tensor $\mathbf{Y}$ with the same shape. For an input element $x_{i,j} \in \mathbb{R}$ and its corresponding output $y_{i,j} \in \mathbb{R}$, **determine which the value of $y_{i,j}$ depends on (select all that apply)**:

(i) $x_{i,j}$

(ii) $x_{i',j}$ where $i \neq i'$

(iii) $x_{i,j'}$ where $j \neq j'$

(iv) $x_{i',j'}$ where $i \neq i'$ and $j \neq j'$

Repeat the same analysis for batch normalization.

Solution: Layer normalization: (i), (iii). Layer normalization is elementwise, meaning it is applied independently to each input vector within the batch.

Batch normalization: (i), (ii). Batch normalization is computed using the statistics of corresponding elements across different vectors in the batch.

For further clarification, refer to the formulas for layer normalization and batch normalization.

(b) **Prove the following:** Given an input vector $\mathbf{x} \in \mathbb{R}^d$ and applying layer normalization with scale γ, bias β, and $\epsilon = 0$, the output $\mathbf{y}$ satisfies

$$\|\mathbf{y} - \beta \mathbf{1}\|_2 = \gamma\sqrt{d}$$

Solution: The layer normalization can be expressed as:

$$\mathbf{z} = (\mathbf{x} - \mu\mathbf{1})/\sigma.$$

where $\mu = \frac{1}{d}\mathbf{x}^T\mathbf{1}$ and $\sigma = \sqrt{\frac{1}{d}\|\mathbf{x} - \mu\mathbf{1}\|_2^2}$.
and

$$\mathbf{y} = \gamma\mathbf{z} + \beta\mathbf{1}.$$

So $\mathbf{z}^T\mathbf{1} = 0$, $\|\mathbf{z}\|_2 = \sqrt{d}$
Therefore

$$\|\mathbf{y} - \beta\mathbf{1}\|_2 = \|\gamma\mathbf{z}\|_2 = \gamma\sqrt{d}$$

