ML and Optim Lec 26

- Transformer Seq2Seq Architecture
- Skip-connections
- Layer-normalization
Skip connections are connections between layers $l$ and layers higher than $l+1$, e.g., DenseNet, where all layers are connected.

An efficient & popular type of skip-connection is a residual connection.

\[
o_{l+1} = \sigma(w_{l+1}o_l+b_{l+1})
\]

\[
o_{l+1} = \sigma(w_{l+1}o_l+b_{l+1}) + o_l
\]
Layer Normalization

Like batch normalization, this helps stabilize training by ensuring the distribution of the outputs of layer $l$ doesn't vary much over time, so layer $l+1$ doesn't have to waste optimization.

$$\text{LayerNorm}(x) = x \cdot \frac{0 - \bar{o}}{\sigma} + \beta$$

where $x \in \mathbb{R}^{n_x}$ and $\beta \in \mathbb{R}^{n_x}$ are the parameters of the layer normalization layer, and

$$\bar{o} = \frac{1}{n_x} \sum_{i=1}^{n_x} o_i$$
$$\sigma^2 = \frac{1}{n_x} \sum_{i=1}^{n_x} (o_i - \bar{o})^2$$
Batch normalization

- Batch normalization computes per neuron means and stds over the minibatch.
- Index into minibatch.
- Output neuron index.
- \( \bar{y} \cdot \left( \frac{y_i - \mu}{\sigma} \right) + \beta \)
- Layer normalization operates on a single output.
- \( \sigma = \text{std of } y \)
- \( \bar{y} = \text{mean of } y \)
- The layer normalization of \( y \) is given by:
  \[ \gamma \cdot \left( \frac{y_i - \mu}{\sigma} \right) + \beta \]
Expressions for Attention

Self-attention:

$\alpha_i^T = \begin{bmatrix} q_i^T k_1 & \cdots & q_i^T k_m \end{bmatrix}$

is the logits vector (row vector) for the $i$th attention vector

$\alpha_i^T = \text{softmax} (\tilde{\alpha}_i^T)$

is my attention vector

then our contextual embedding for the $i$th token is

$c_i^T = \sum_{j=1}^{m} (\alpha_i)^j v_j^T$

let $X = \begin{bmatrix} x_1^T & \cdots & x_m^T \end{bmatrix} \in \mathbb{R}^{m \times d}$

and $C = \begin{bmatrix} c_1^T & \cdots & c_m^T \end{bmatrix} \in \mathbb{R}^{m \times d_v}$
Then
\[ C = \text{Att}(X_T X) \]

where the attention layer has parameters \( \omega_Q, \omega_K, \omega_V \).

To compute \( C \), write

\[ K = X \omega_K = \begin{bmatrix} x_1^T \omega_K \\ \vdots \\ x_m^T \omega_K \end{bmatrix} \]

\[ Q = X \omega_Q = \begin{bmatrix} x_1^T \omega_Q \\ \vdots \\ x_m^T \omega_Q \end{bmatrix} \]

\[ V = X \omega_V = \begin{bmatrix} x_1^T \omega_V \\ \vdots \\ x_m^T \omega_V \end{bmatrix} \]

then note that

\[ C = \begin{bmatrix} c_1^T \\ \vdots \\ c_m^T \end{bmatrix} = \text{softmax} \left( \frac{Q K^T}{\sqrt{d_k}} \right) V \]
Similarly, a general attention layer takes two sequences $X \in \mathbb{R}^{m \times d}$ and $X' \in \mathbb{R}^{T \times d}$, so a general attention layer computes

$$C = \text{softmax} \left( \frac{Q K^T}{\sqrt{d_k}} \right) V$$

where

$$K = X W_K$$
$$V = X' W_V$$
$$Q = X' W_Q$$
Transformer Architecture

Encoder:
- use multiple layers of multi-head self-attention with residual connections, layer normalization, and dense nonlinear transforms

Decoder:
- use multiple layers of multi-head self-attention, multi-head general attention, with residual connections, layer normalizations, and dense nonlinear transforms.
Encoder:

$[c_1, \ldots, c_m]$ contextual representations

Block $N$

Block $1$

$x_1, \ldots, x_M$
token embeddings

where each embedding block has its own parameters and computes:

Given input $X = \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix} \in \mathbb{R}^{m \times d}$,

1) $y = \text{LayerNorm} \left( X + \text{Multi-head Att}(X, X) \right)$

recall $\text{Multi-head Att}(X, X) = \left[ \text{Att}_1(X, X) | \cdots | \text{Att}_h(X, X) \right] \odot$

concatenates in different attention outputs together then takes linear combination to learn a final $d$-dim representation

2) $C = \text{LayerNorm} \left( y + \text{FF}(y) \right)$

where FF applies the same feedforward dense + ReLU layer to each row of $y$
During training, use teacher forcing

each block has its own parameters
and computes, given inputs $X' = \begin{bmatrix} x'_1 \\ \vdots \\ x'_T \end{bmatrix}$
and the final representations from
the encoder of the input sequence $C = \begin{bmatrix} c_1' \\ \vdots \\ c_m' \end{bmatrix}$:

1. $Y = \text{LayerNorm} \left( X' + \text{MultiheadAtt}(X', X') \right)$
2. $Z = \text{LayerNorm} \left( Y + \text{MultiheadAtt}(C, Y) \right)$
3. $O = \text{LayerNorm} \left( Z + \text{FF}(Z) \right)$

In the self-attention layers, the attention is masked: we set
$(x'_i)_j = 0$ for $j > i$ to ensure we only predict using historical info.
During inference time, we autoregressively use the decoder to predict the next token, add it onto our input sequence, and pass it back through the decoder to get the next token. Repeat until we either get an <EOS> token or we’ve reached the maximum sequence length we set.
Positional Encodings

Issue w/ Transformers as is: they don’t properly learn positional relations (e.g. that “the cat ate its food” should have a different contextual rep from “the food ate its cat”)

Soln: replace \( X \in \mathbb{R}^{m \times d} \) with \( X + P \)

where \( P \in \mathbb{R}^{m \times d} \) encodes the positions 1 through \( m \)

\[
P_{i,j} = \begin{cases} 
\sin(i \omega j/d) & \text{if } j \text{ even} \\
\cos(i \omega (j-1)/d) & \text{if } j \text{ odd}
\end{cases}
\]

(refer to preceding diagram to see where positional embeddings are added)