- Model is trying to predict the next character (not phrase/word) in this case

Tokenization

- each characters is getting tokenized
- 90% - Train, 10% - Valid

Batches / Chunks

- Batches is for parallel data processing
- Chunks is the size of characters being sent at once
  - In code the chunks are size +1 b/c there are x amount of possibilities/examples in x+1 chunk

ex: (18, 47, 56, 57, 58, 1, 15, 47, 58) chunk = 8

input: (18) output: (47)
input: (18, 47) output: (56)
input: (18, 47, 56) output: (57)
input: (18, 47, 56, 57) output: (58)
.
.
.

8 examples or possibilities

Bigram Language Model / loss / generation

- Bigram LM is a model that predicts the next word in a sequence based on the previous word
- The loss function we are using is the negative log-likelihood or cross entropy
  - Used to measure the quality of predictions
  - The mathematical meaning of logs in calculating the likelihood of something is numerical stability and simplification of computations

numerical stability: The likelihood of an event involves multiplying probabilities which can lead to numerical underflow, logs can help solve this problem b/c $\log(a \cdot b) = \log(a) + \log(b)$ which makes the values more manageable

Simplifications of Computations: Logs can be used to convert • to + which is more computationally efficient

numerical underflow: a phenomenon that occurs when working with very small floats that fall under the precision limits of float representations and the float is just rounded down to 0

- Cross Entropy: $H(P,Q) = \sum P(x) \log(Q(x))$
  P = target values, Q = output prob for each class
- The lower the Cross Entropy value the closer the predicted prob to true prob

- Tensors are n-dimensional matricies
- The generate function will first generate garbage since there is no self-attention layer / history yet
- Forward function represents the forward propagation step where the input data flows through the network layers produces the final output

Building the Self-Attention

- 1st method, take the average of the channels (groups or divisions) of previous tokens
- The mathematical trick in self-attention is to allow a token to only communicate with previous tokens with matrix multiplication (2nd method)

Triangular matrix

Mask → $\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 2 & 7 \\ 6 & 4 \\ 6 & 6 \end{pmatrix} = \begin{pmatrix} 2 & 7 \\ 8 & 11 \\ 14 & 16 \end{pmatrix}$

2+6=8, 7+4=11, 8+6=14, 11+5=16

Now we can apply this to find mean

$\begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 0 \\ 1/3 & 1/3 & 1/3 \end{pmatrix} \times \begin{pmatrix} 2 & 7 \\ 6 & 4 \\ 6 & 6 \end{pmatrix} = \begin{pmatrix} 2 & 7 \\ 4 & 5.5 \\ 4.667 & 5.333 \end{pmatrix}$

$\frac{1}{2}(2+6)=4$, $\frac{1}{2}(7+4)=5.5$

$\frac{1}{2}(2+6+6)$, $\frac{1}{2}(7+4+5)$

- 3rd method is to use Softmax to create the mask
- Every token has two vectors; a query and a key
  - key vec is what is contained
  - Query vec is what you are looking for (transposed)
- The query of a token is dot producted with the keys of all the other keys of tokens in the sequence
- If the query and key are aligned the dot product will have a higher res/weight
- The mask actually used will be dependent on the input and its tokens, so won't be exactly like mask above
  - Softmax will help normalize the weights of the tokens

↑ All of this is the decoder block

- The decoder only allows for seen nodes to communicate with past nodes (this is done with the mask matrix)
- A encoder attention block allows for attention between all tokens (this is done by removing the mask)

Batch dimension is the same as dimension size

Scaled dot-product Attention

Weight

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

From Attention is all you need

$Q \cdot K^T$ from the decoder above

d_k is the head size

Multiplying V is same as aggregating (sending information)

The Reason Why $\sqrt{d_k}$

- When apply softmax to weights with very positive and negative values the result will converge
- If you multiply the softmax result the values will sharpen towards the max (leans to the max more)

Softmax(0.1, -0.2, 0.3, -0.2, 0.5)

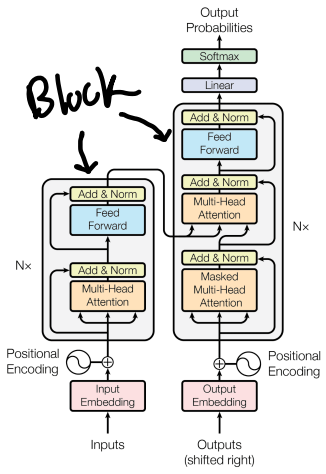
res (0.1925, 0.1426, 0.2351, 0.1426, 0.2872)

Softmax(0.1, -0.2, 0.3, -0.2, 0.5) · 8

res (0.0326, 0.0030, 0.1615, 0.0030, 0.8000) much better

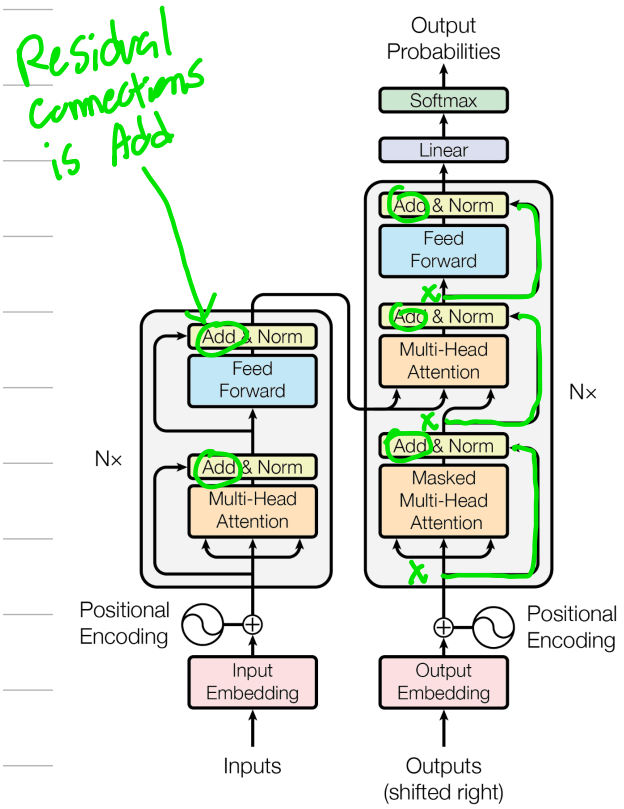
Building the Transformer

- Multihead-Attention is used much more and runs multiple heads of self attention in parallel
- Logits are raw unnormalized predictions of a model
- The Feed-Forward Layer is used to process the logits from multi-head attention layer
  - "The model needs to think and process the logits"

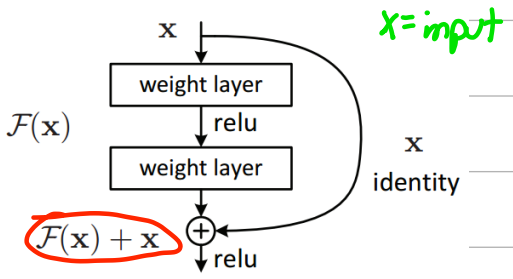


- The block handles the communication followed by the computation
- Communication is done with multihead-attention
- Computation is done with feedforward

- Our head size is 8 and the number of heads is 4



- green arrows are called skip connections / residual connections



- Residual connections is meant to solve "vanishing gradient problem" which happens with deeper NN

Vanishing gradient problem: When the gradient of the loss function with respect to the weights become extremely small during back-propagation (calc/updating weights starting from outputs) as earlier layers receive smaller and smaller updates

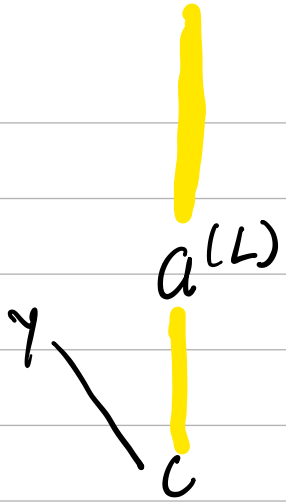
Math explanation

Back propagation multiplies the gradients using the chain rule

- 1st we want to calc how much C ("cost") changes

ex: $W^{(L-1)}$ $a^{(L-1)}$ $b^{(L-1)}$ → $Z^{(L)}$

W: weights
a: activation
b: bias
Z: weighted sum
Y: result/output



- Trying to find how much C changes if W changes

So we want: $\frac{\partial C}{\partial W^{(L-1)}}$

$$\frac{\partial C}{\partial W^{(L-1)}} = \frac{\partial z^{(L)}}{\partial W^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}}$$

↑ Chain Rule

Now repeat with bias then activation

$$\text{2nd: } \frac{\partial C}{\partial b^{(L-1)}} = \frac{\partial z^{(L)}}{\partial b^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}}$$

$$\text{3rd: } \frac{\partial C}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}}$$

- So models with more layers will have a longer chain rule which shrinks the earlier derivatives

Now how does residual connections solve this?

- Residual connections allow layers to be skipped during forward and backward propagation so earlier layers can be influenced more

- After the residual connections we have to apply layer normalization : Layer norm (F(x)+x)

$$y = \left(\frac{x - E(x)}{\sqrt{Var[x]}}\right) \cdot \gamma + \beta = \left(\frac{x - E(x)}{\sigma}\right) \cdot \gamma + \beta$$

(Gamma, Beta)

↑ ↑ Learning Rates

$$x = f[W^T x + b]$$

ex: $\begin{bmatrix} 0.2 & 0.1 & 0.3 \\ 0.5 & 0.1 & 0.1 \end{bmatrix}$ 2 words, 3 dim

$E(x)_1 = \mu_1 = \frac{1}{3}(0.2 + 0.1 + 0.3) = 0.2$

$E(x)_2 = \mu_2 = \frac{1}{3}(0.5 + 0.1 + 0.1) = 0.233$

$\sigma^2 = E(X-\mu^2)$

$\sigma_1 = \sqrt{\frac{1}{3}[(0.2-0.2)^2 + (0.1-0.2)^2 + (0.3-0.2)^2]}$

$= 0.08164$

$\sigma_2 = \sqrt{\frac{1}{3}[(0.5-0.233)^2 + (0.1-0.233)^2 + (0.1-0.233)^2]}$

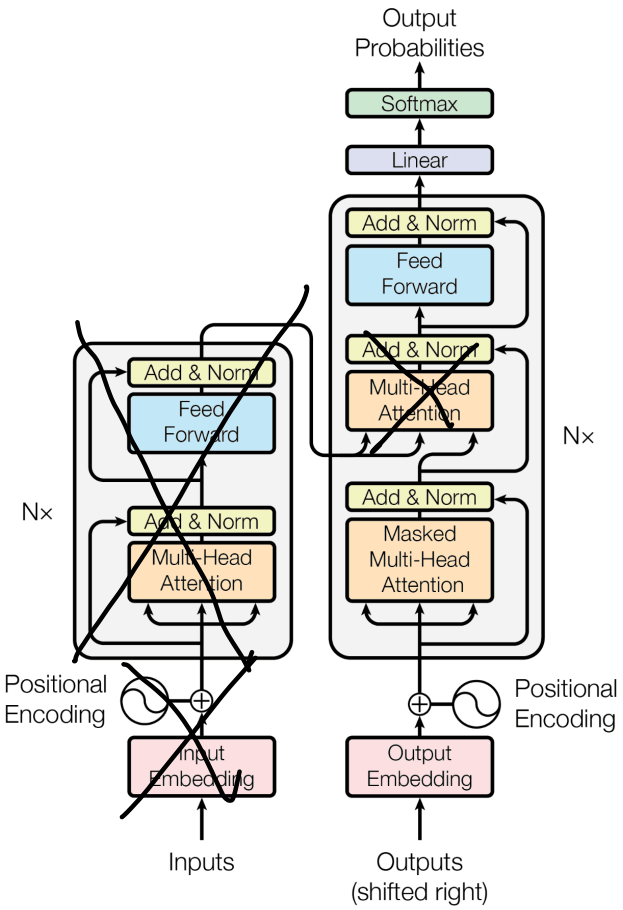
$= 0.1886$

$M = \begin{bmatrix} 0.2 \\ 0.233 \end{bmatrix} \qquad \sigma = \begin{bmatrix} 0.08164 \\ 0.1886 \end{bmatrix}$

$y = \frac{X-\mu}{\sigma} = \begin{bmatrix} 0 & -1.2248 & 1.2248 \\ 1.414 & -0.707 & -0.707 \end{bmatrix}$

output $= \gamma \cdot$ ↑ $+ \beta$

- γ and β will be updated during back propagation
- Slight update to the transformer model is to apply the layer norm before the transformation (multi-head attention)
- Final batch size increased to 64 and block size/content length increased to 256
- Dropout is also added for better optimization



- Our model only has a decoder block and no cross attention
- The model only generates text with no meaning
- The encoder would have no mask so all the tokens of the context can communicate freely
- With encoder the keys and values come from it into the decoder
- Our model has about 10 million parameters