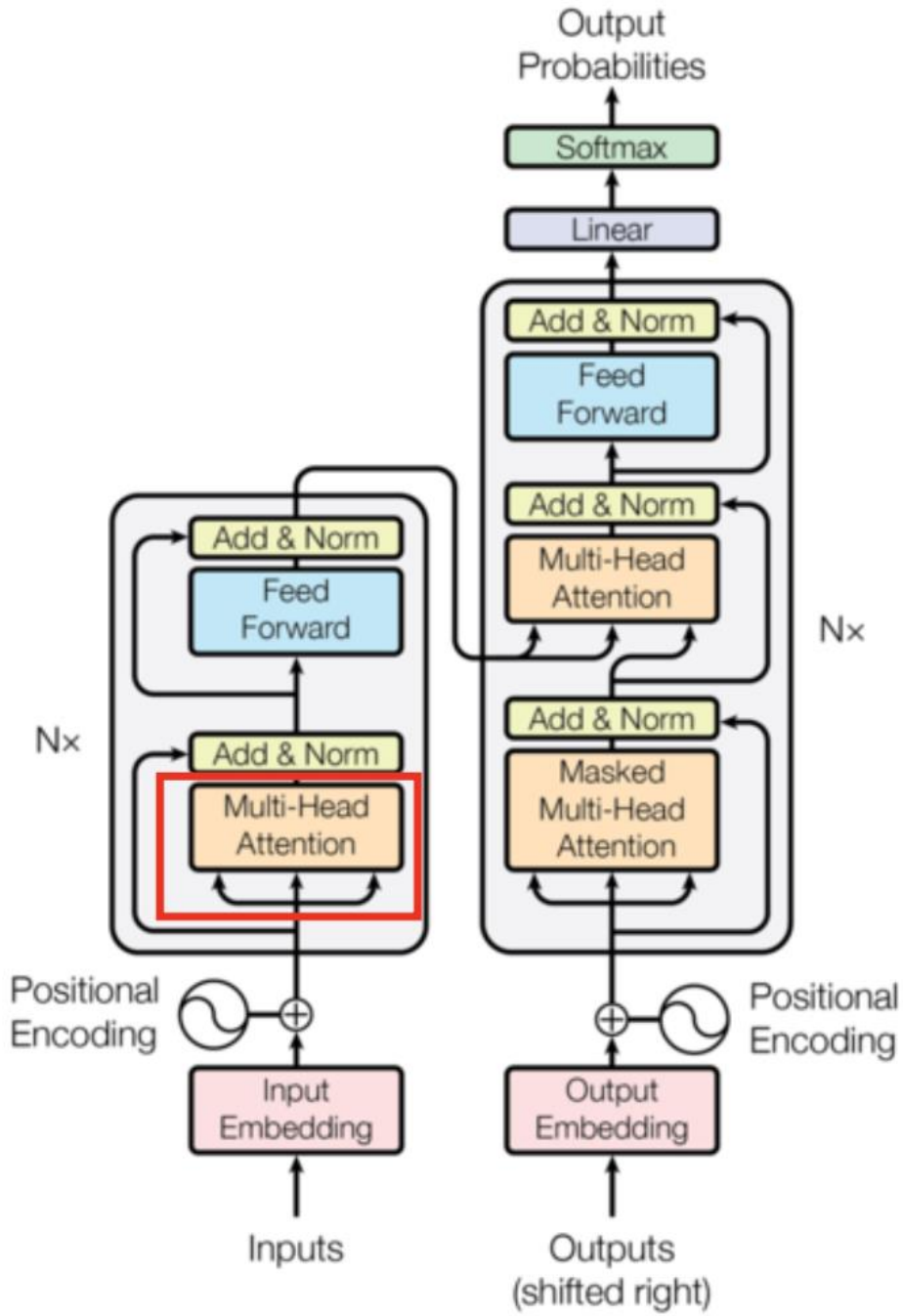
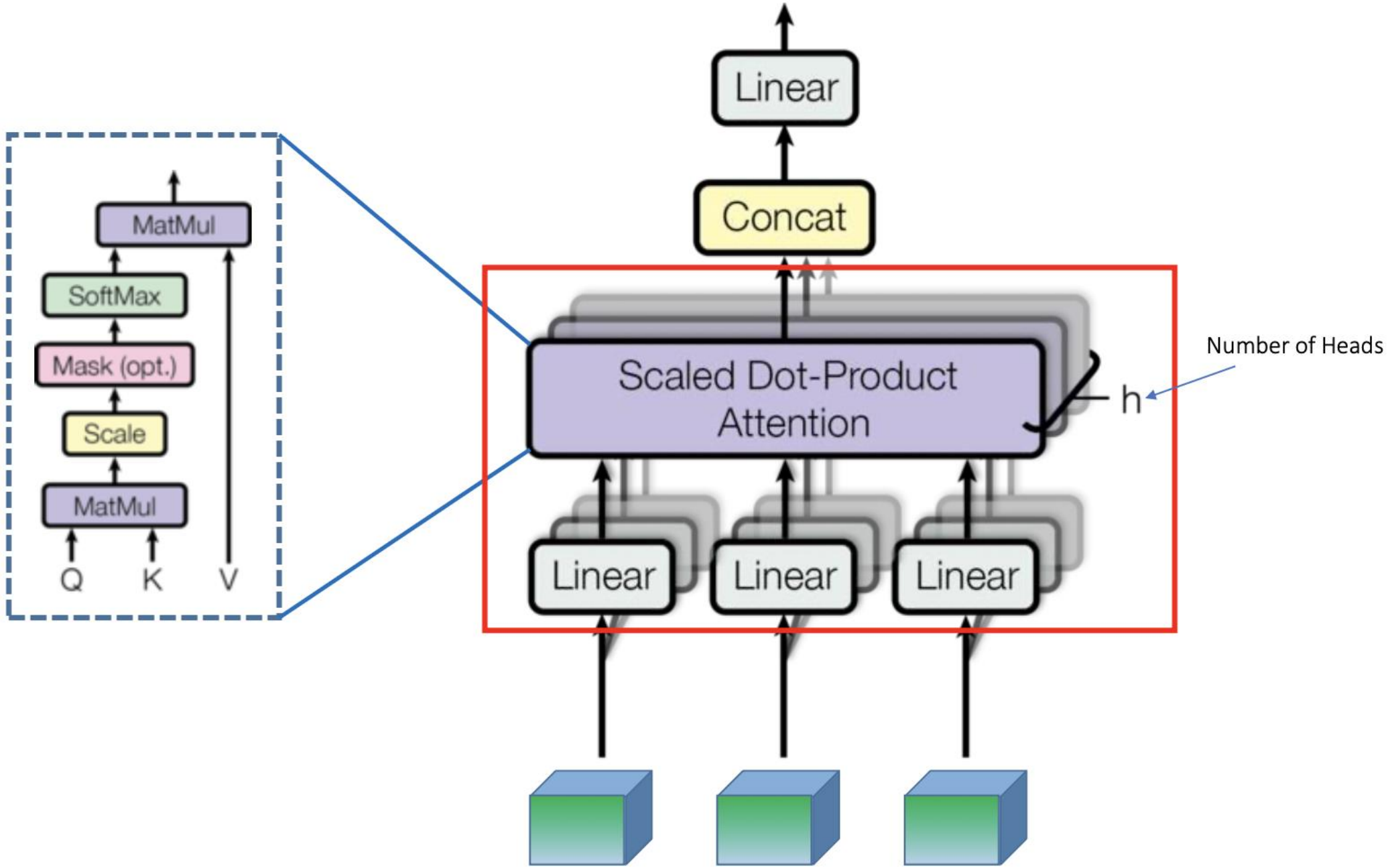
Transformer II

Yuxuan Du

EE 5993 AI Practicum

03/24/2025


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)


Linear
Concat
Scaled Dot-Product Attention
h
Number of Heads
Linear
Linear
Linear
MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q
K
V

Equations for Multi-Head Attention

As we said, we have more than one attention (multi-head attention).

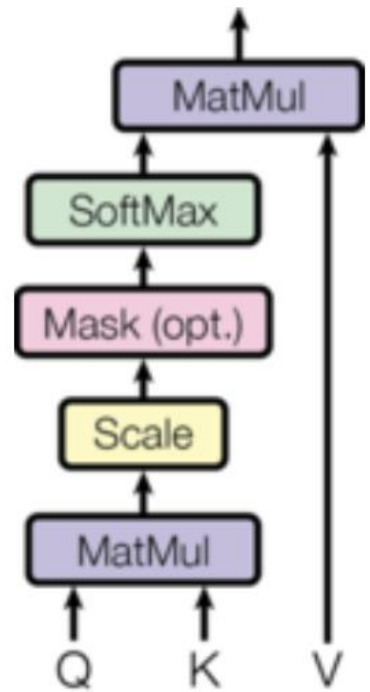
For each head i:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

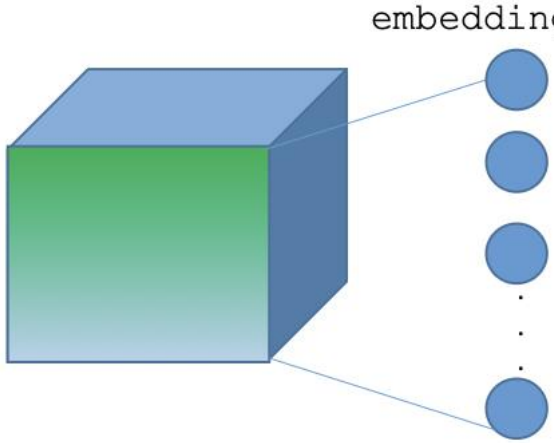
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

After that, concatenate them back and run them through a linear layer:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

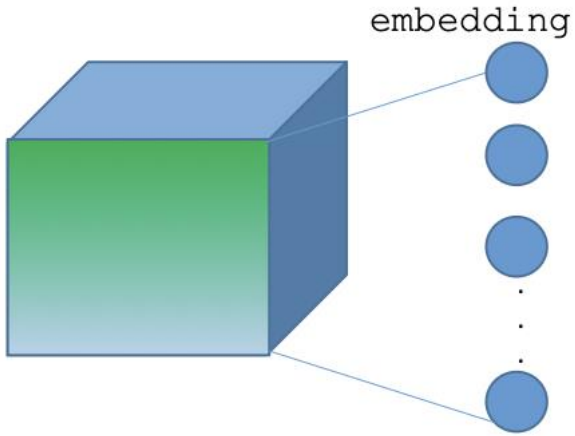


Now Let's go in detail into each one of them…..



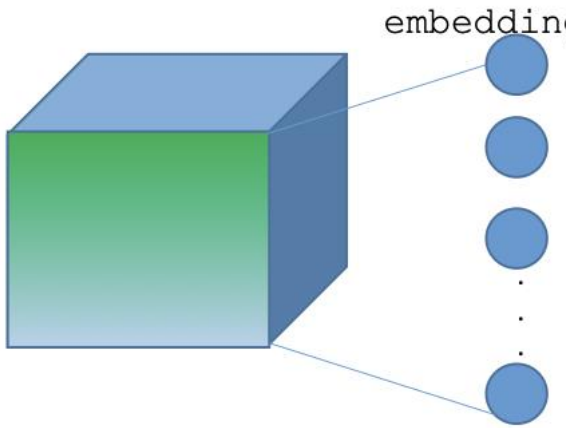
Query

Divide into *h* heads



Key

Divide into *h* heads

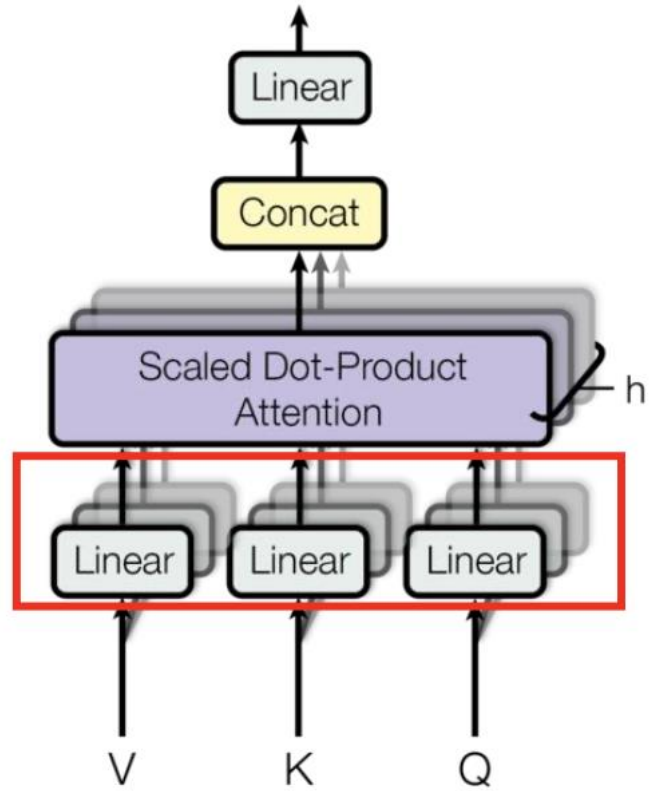


Value

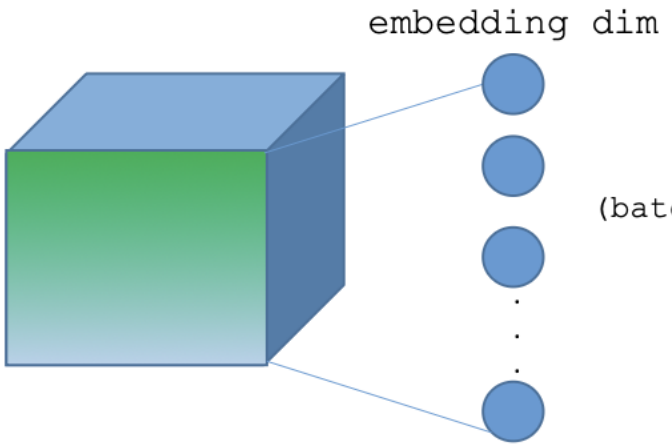
Divide into *h* heads

The dimensionality of each head d_k is `embedding_dim / h`

Example: If we use 8 heads, then each head would have a dimension of: `512 / 8 = 64`

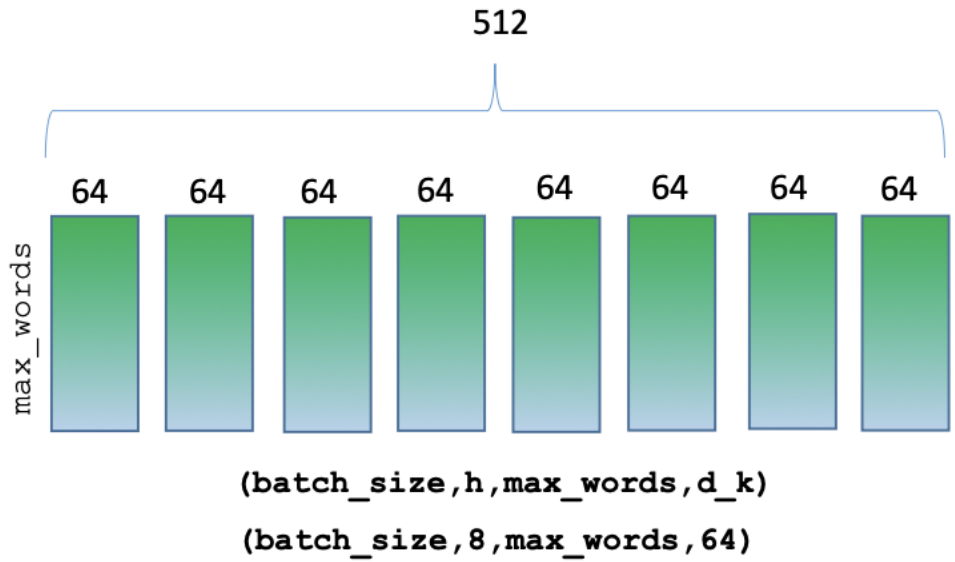


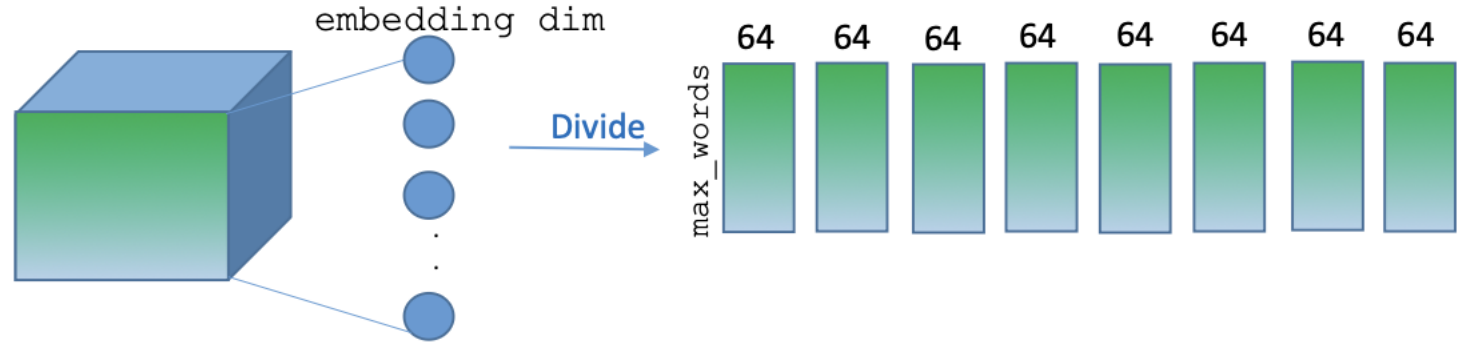
Let's choose:
embedding dimension = 512
Number of heads = 8



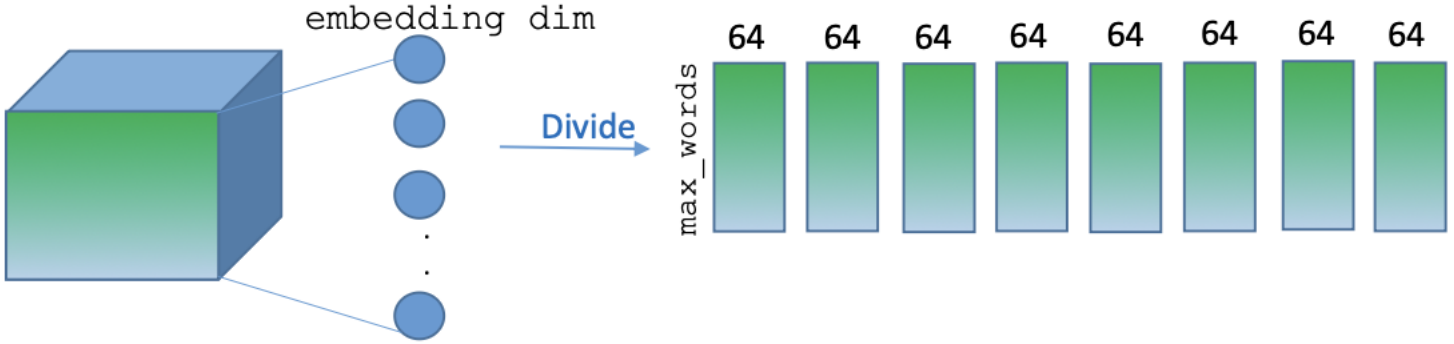
(batch_size,max_words,embedding_dim)

Divide

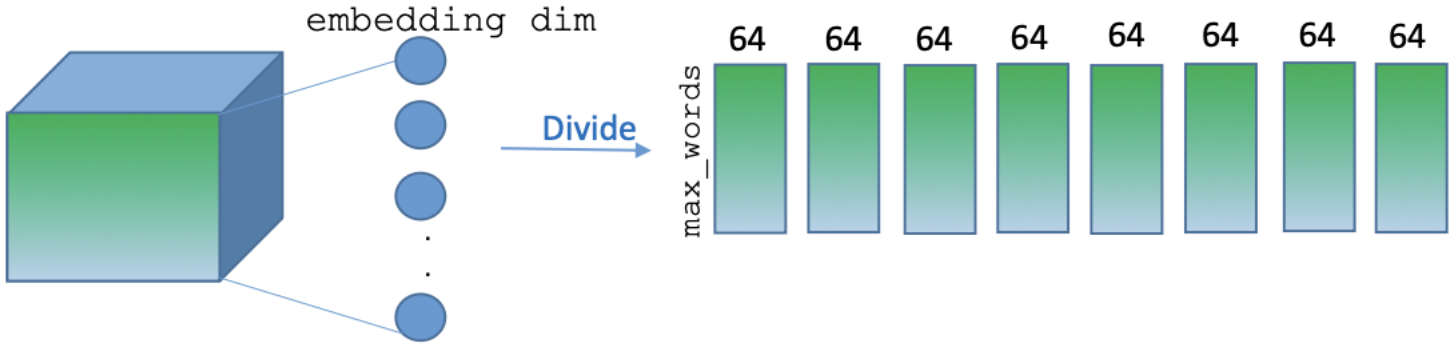




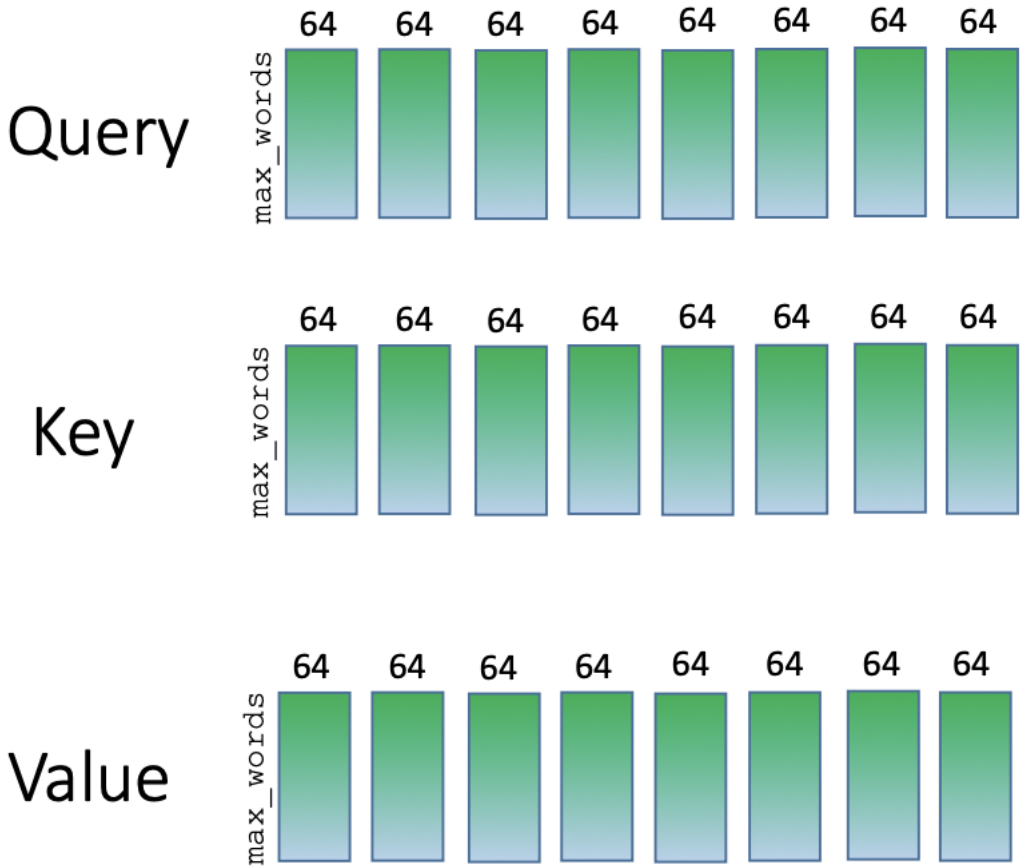
Query



Key



Value



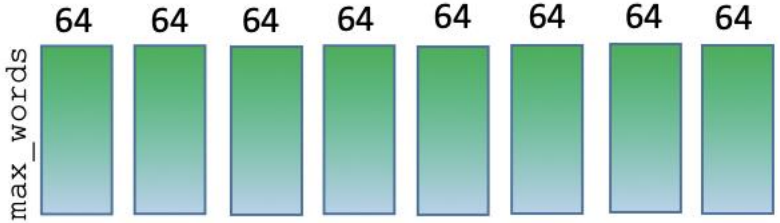
(batch_size, h, max_len, d_k) $\odot$ (batch_size, h, d_k, max_len)

(batch_size, h, max_len, max_len)

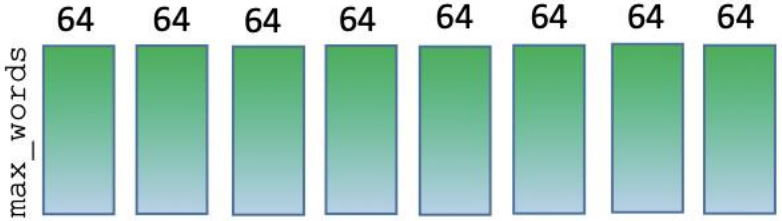
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

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

Query



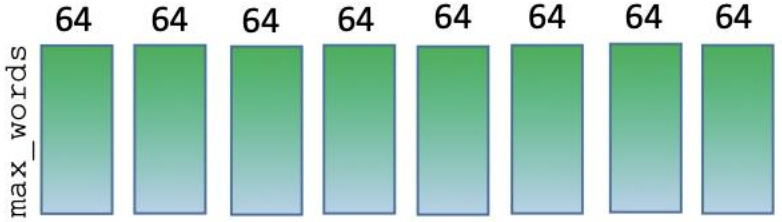
Key

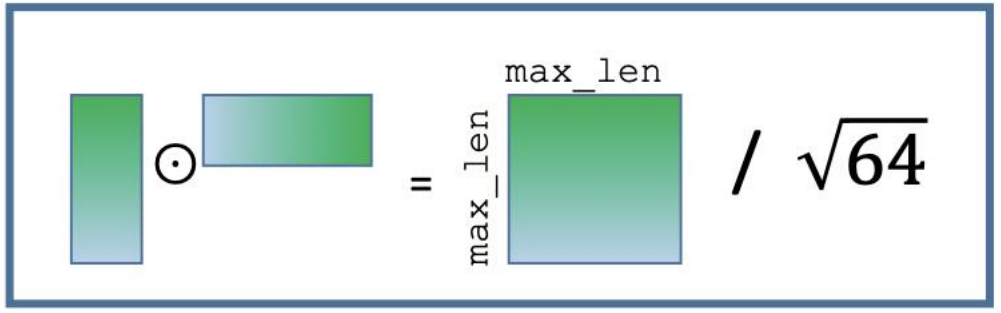


For each

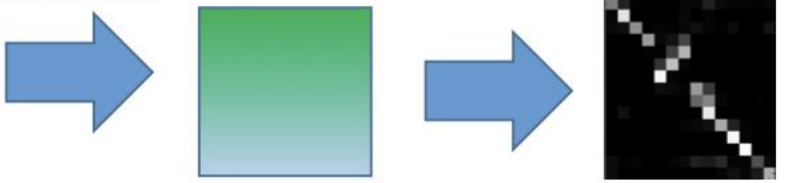
Dot Product and Scale

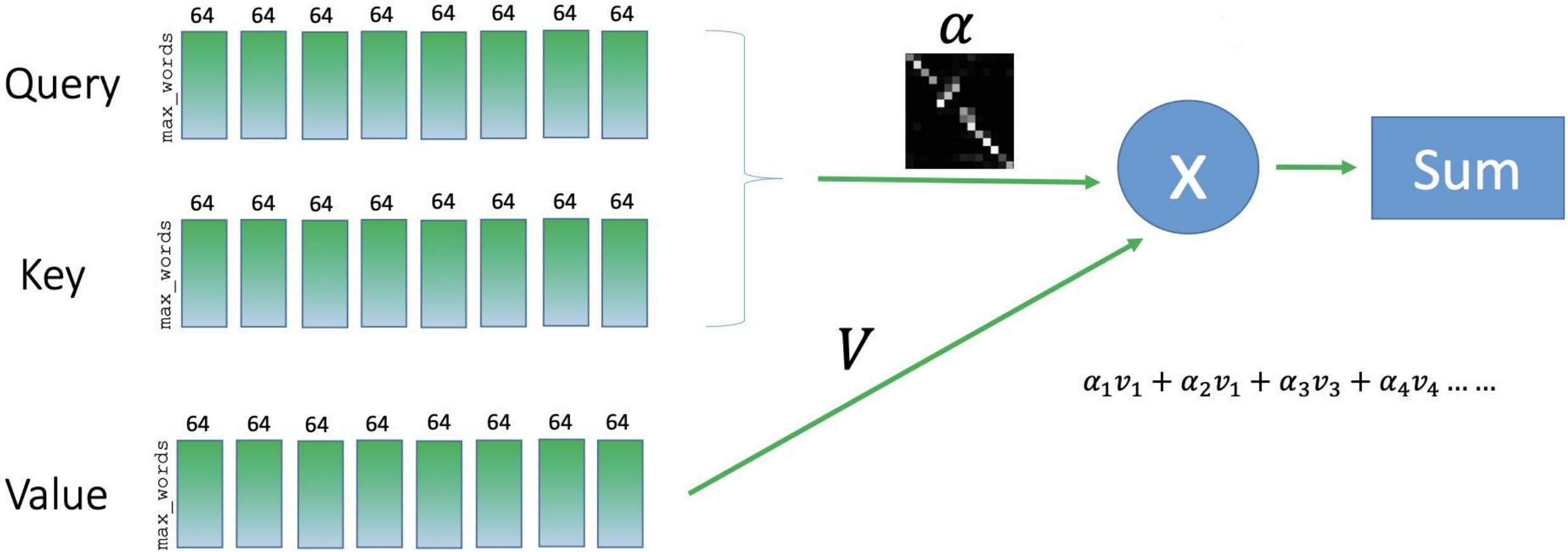
Value





Softmax





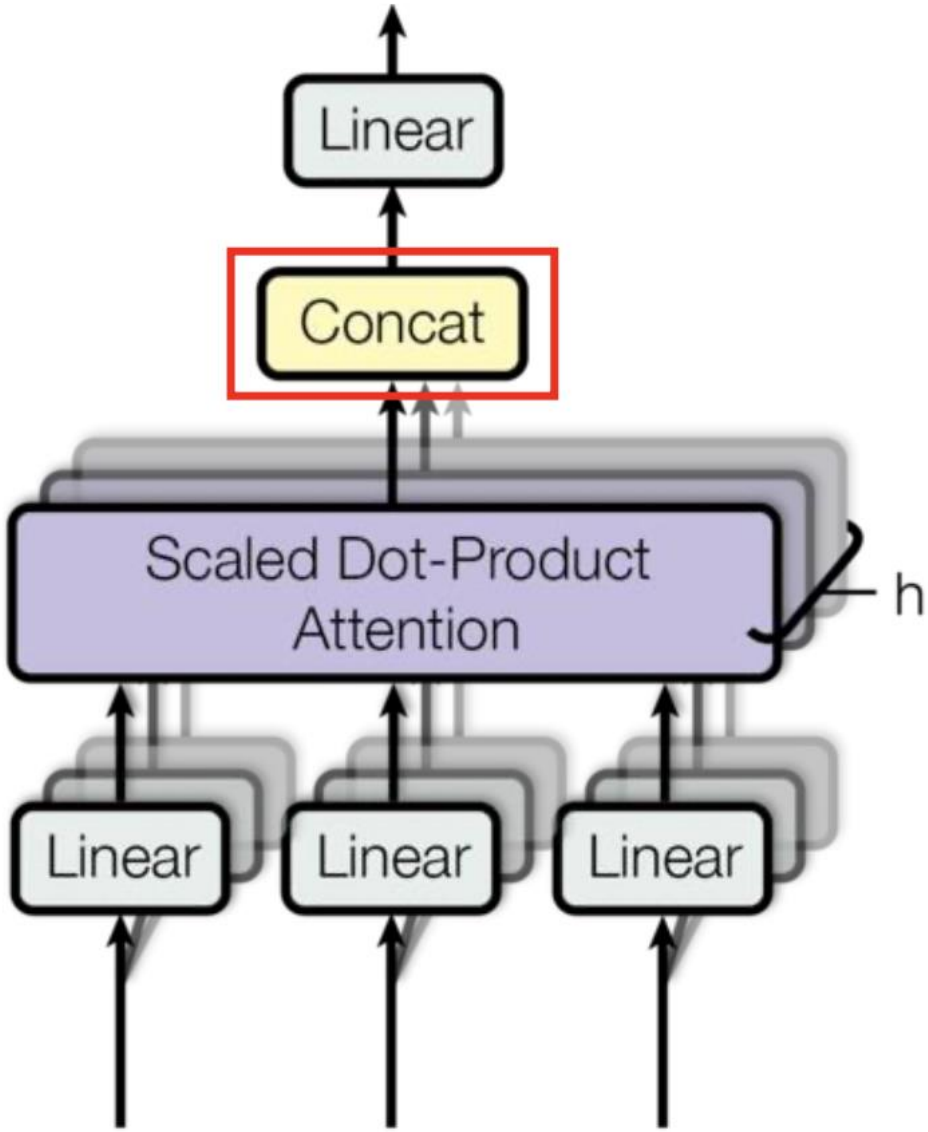
(batch_size, h, max_len, max_len) ⊙ (batch_size, h, max_len, d_k)

(batch_size, h, max_len, d_k)

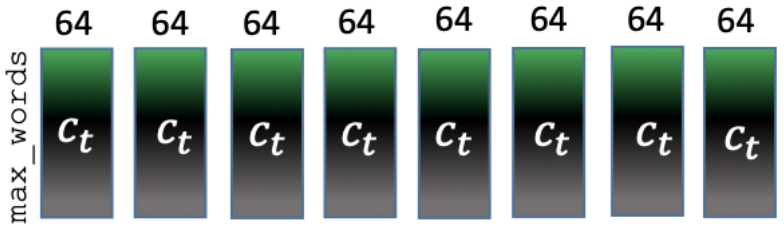
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

For each head i:

$'_1 \top u_2 v_1 \top u_3 v_3 \top u_4 v_4 \ldots\ldots$


Linear
Concat
Scaled Dot-Product
Attention
h
Linear
Linear
Linear

Attention Result

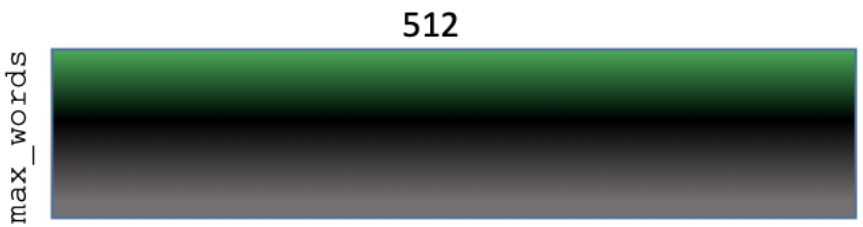


`(batch_size, h, max_len, d_k)`

So now we have 8 different context vectors

Now concatenate them back again to the original shape:

Final Attention Result



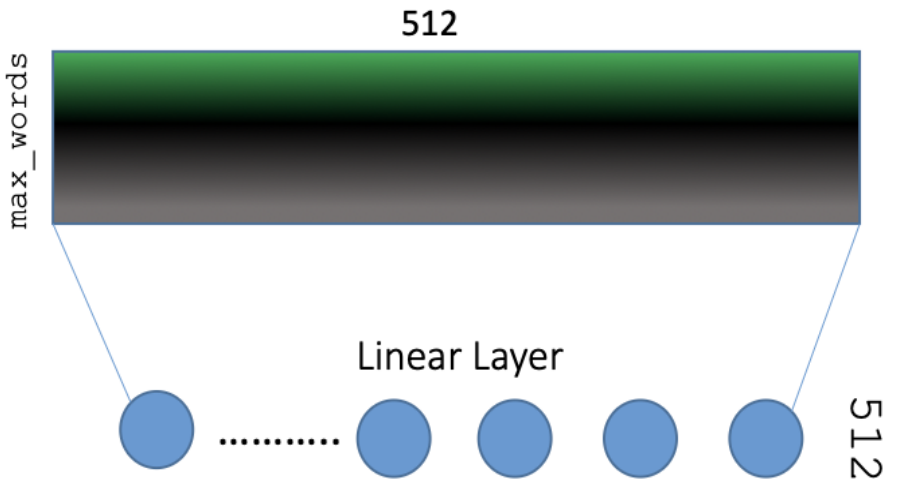
`(batch_size, max_len, h * d_k)`

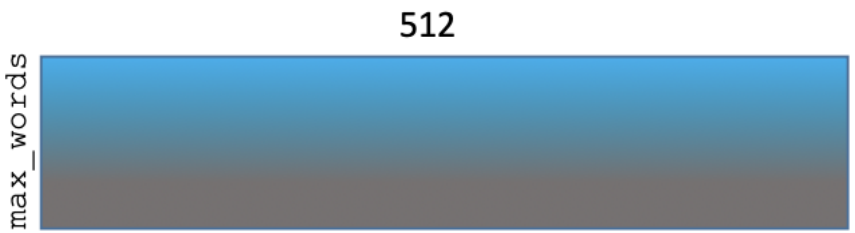
`(batch_size, max_len, 8 * 64)`

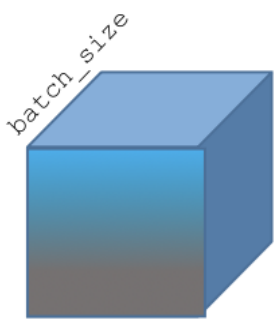
`(batch_size, max_len, 512)`

(batch_size, max_len, 512)

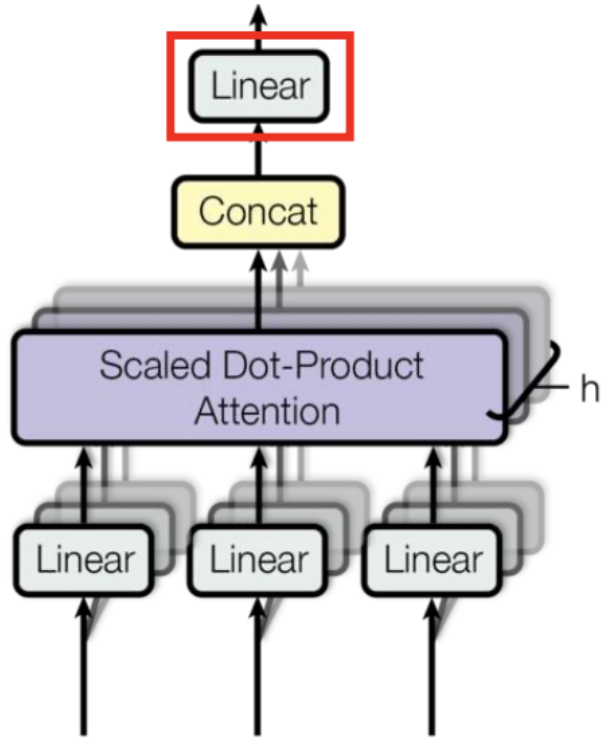
Final Attention Result





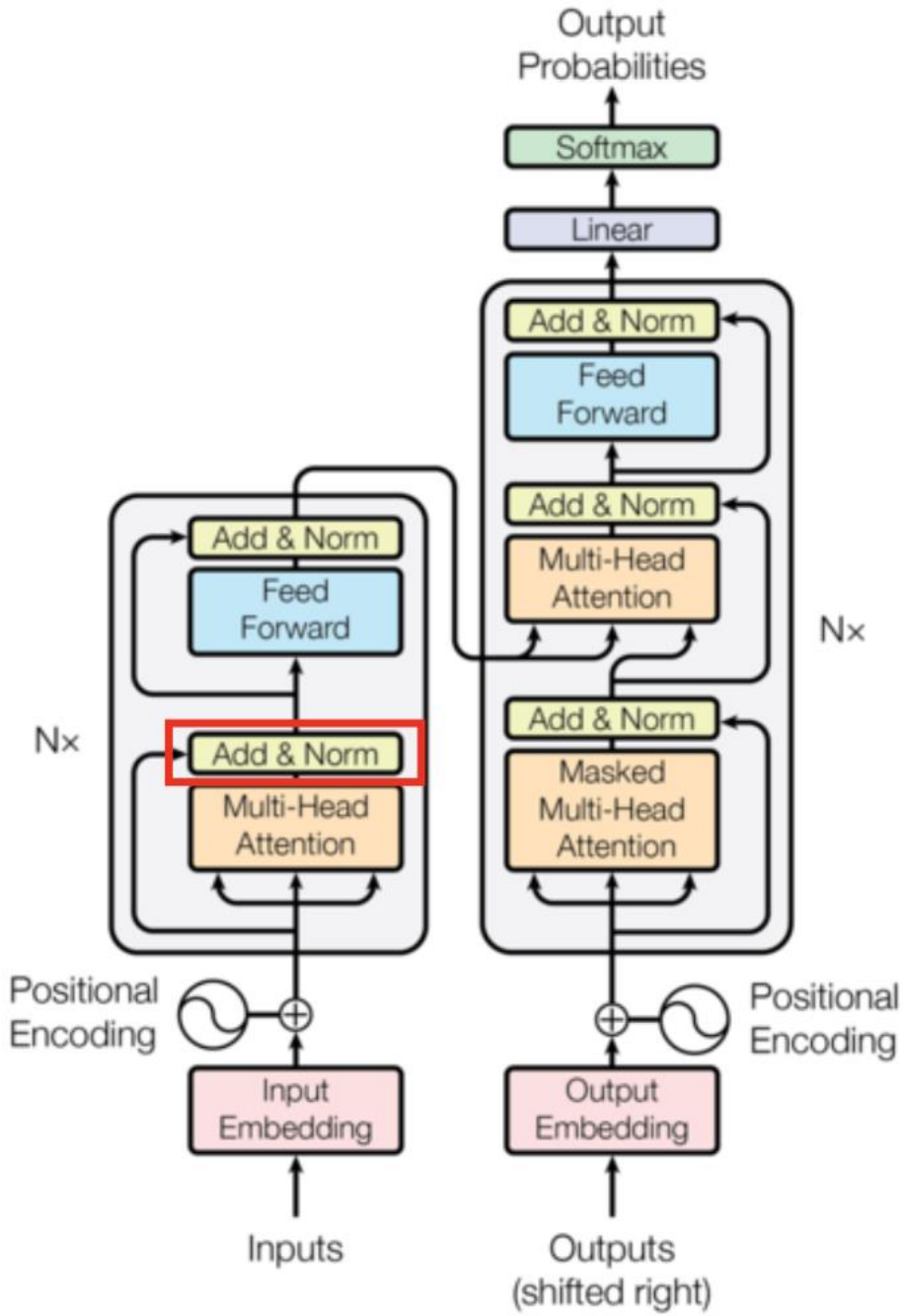


(batch_size, max_len, 512)

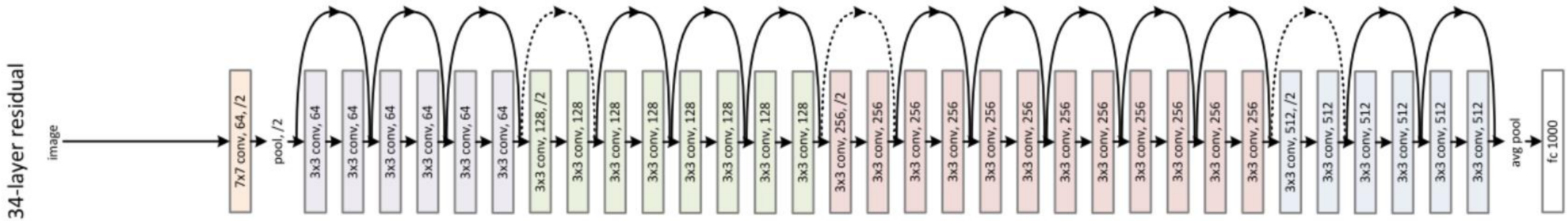


Why do we need a Linear Layer?

We computed 8 different attention results. After we concatenate them, we want them to interact with each other. So we run the concatenated result through a single feed forward (linear) layer.


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

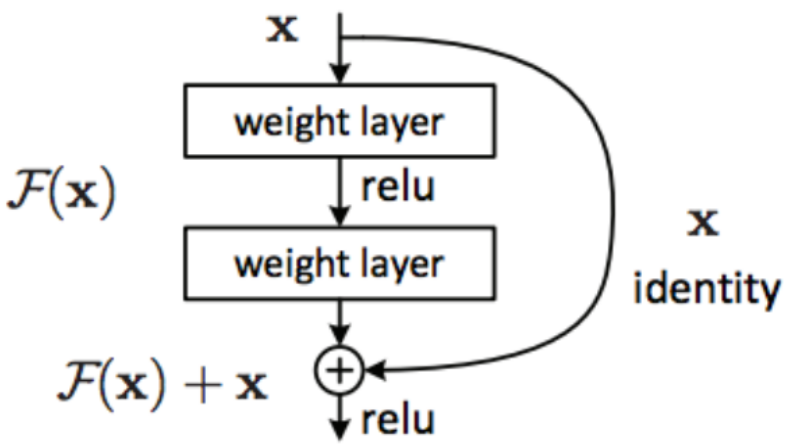
Residual Networks

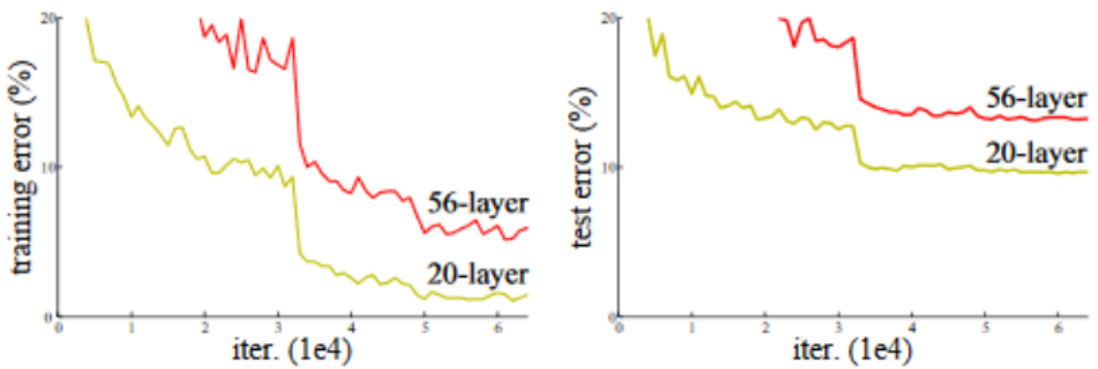


The idea behind Residual Learning

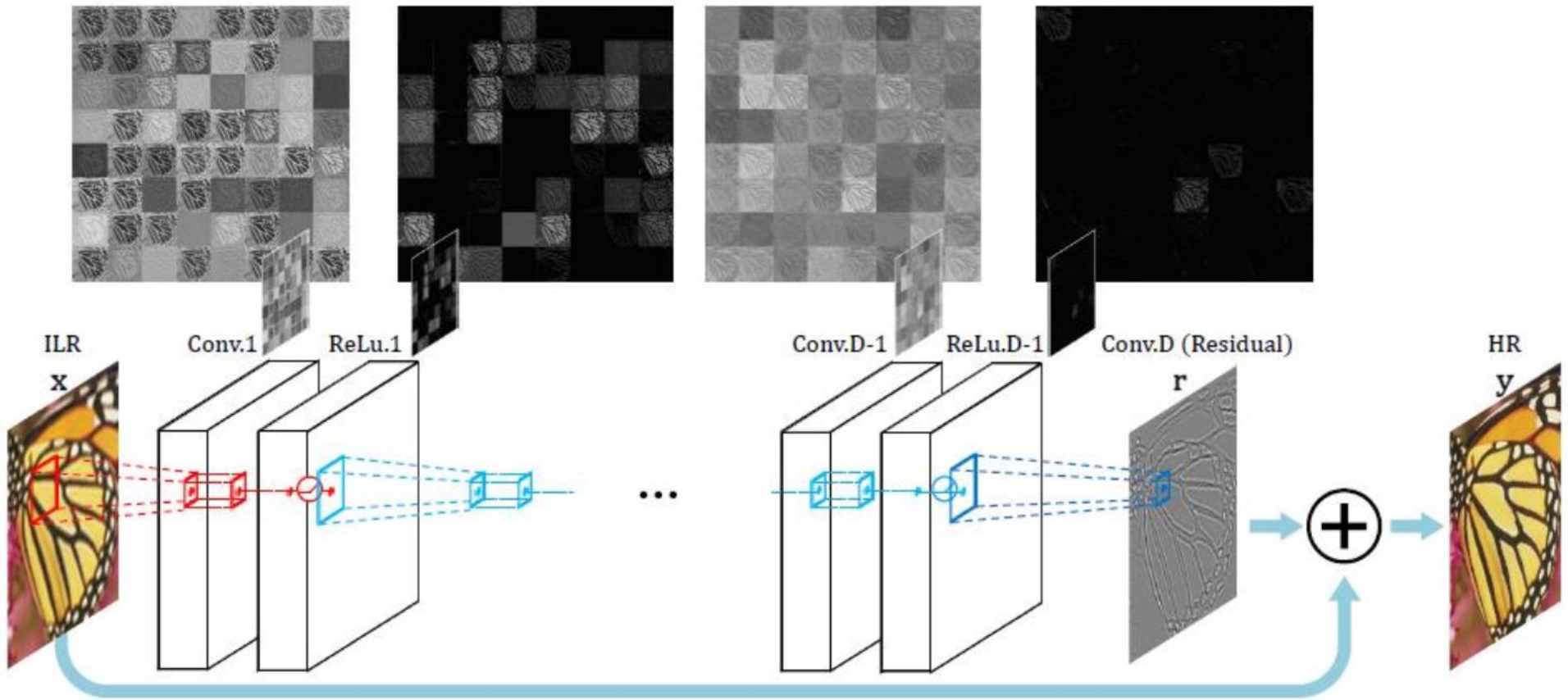
We are learning what's left of (residual), without learning a new representation. You are learning the "remaining" only. If the block doesn't learn anything, then your F(x) would be 0, and that it what makes the training go much faster, since learning a completely new representation is omitted. Therefore, the model can default to using the identity function if the layer is not beneficial.

→ Either learn something useful, or don't learn anything!





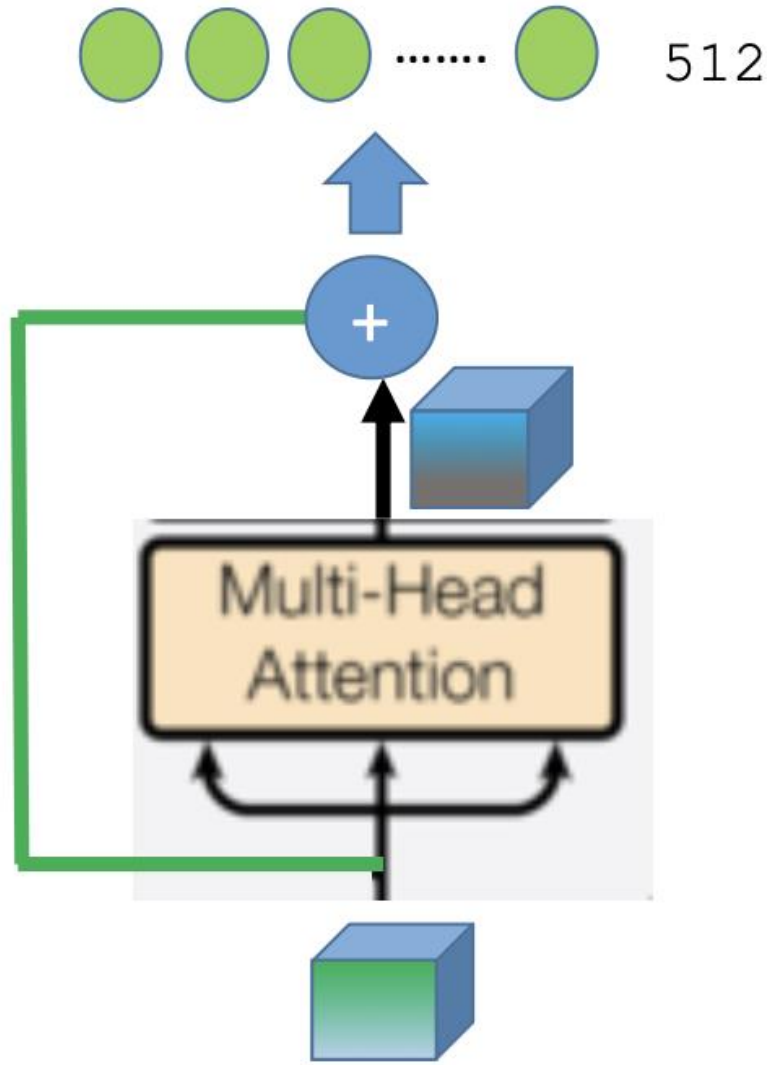
A nice way to understand the concept of Residual Learning is through an application



Low Resolution Image

The network focuses on learning to construct the high Frequency Information (Image Details) such as edges without learning to re-construct the low frequency information.

High Resolution Image


512
+
Multi-Head
Attention

Layer Normalization

Normalize the outputs of a layer

Why do we need Layer Normalization?
In order to prevent the values of the outputs from becoming bigger. We have performed a lot of operations which may cause the values of the layer output to become bigger. So we use Layer Norm to normalize them back again.

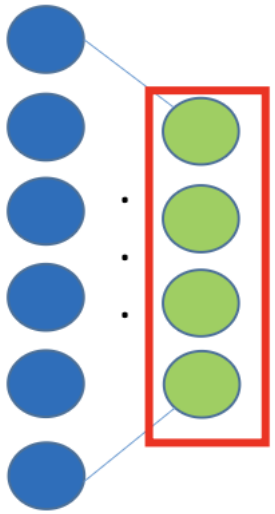
At the output of a feedforward layer:

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

Mean over all neurons in the layer

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

Variance over all neurons in the layer



For each neuron in the output layer:

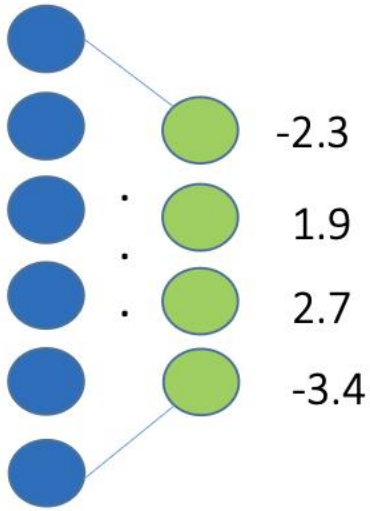
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

Normalize: subtract mean and divide by standard deviation

$$y_i \leftarrow \gamma \hat{x}_i + \beta$$

Scale by a learnable value and shift by a learnable value

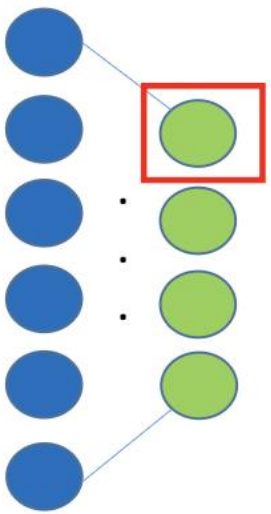
Learnable Parameters



$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \frac{-2.3+1.9+2.7-3.4}{4} = -0.275$$

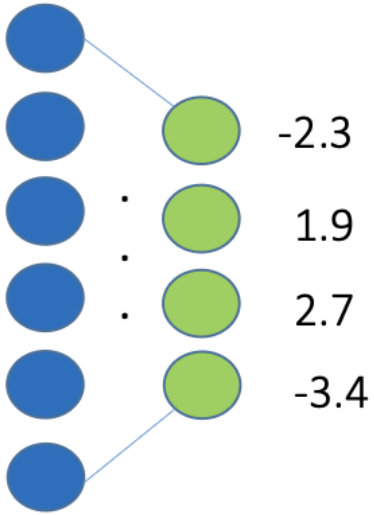
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i-\mu_{\mathcal{B}})^2$$

$$\frac{(-2.3+0.275)^2+(1.9+0.275)^2+(2.7+0.275)^2+(-3.4+0.275)^2}{4} = 6.861$$



$$\hat{x}_i \leftarrow \frac{x_i-\mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}} \qquad \frac{-2.3+0.275}{\sqrt{6.861+10^{-9}}} = -0.77$$

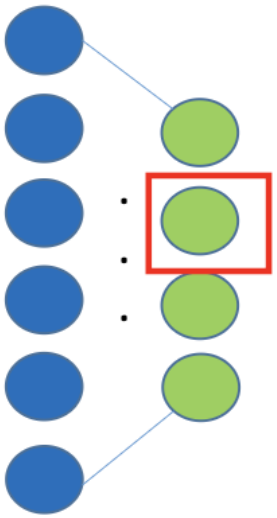
$$y_i \leftarrow \gamma\hat{x}_i+\beta \qquad -0.77\theta_1+\theta_2$$



$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \frac{-2.3 + 1.9 + 2.7 - 3.4}{4} = -0.275$$

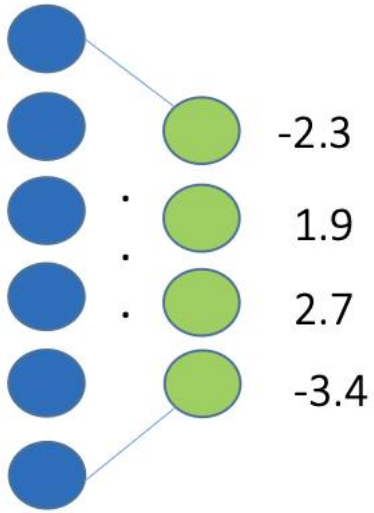
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

$$\frac{(-2.3 + 0.275)^2 + (1.9 + 0.275)^2 + (2.7 + 0.275)^2 + (-3.4 + 0.275)^2}{4} = 6.861$$



$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \frac{1.9 + 0.275}{\sqrt{6.861 + 10^{-9}}} = 0.83$$

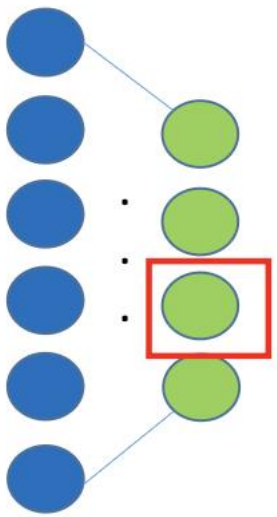
$$y_i \leftarrow \gamma\hat{x}_i + \beta \qquad 0.83\theta_1 + \theta_2$$



$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \frac{-2.3 + 1.9 + 2.7 - 3.4}{4} = -0.275$$

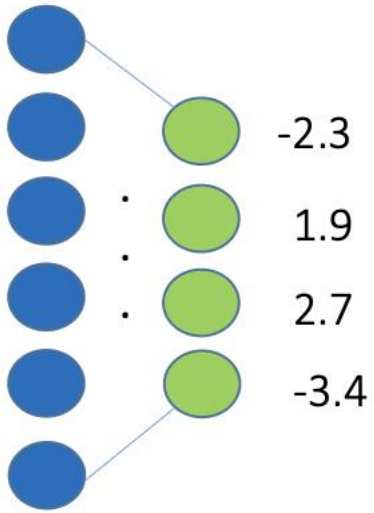
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

$$\frac{(-2.3 + 0.275)^2 + (1.9 + 0.275)^2 + (2.7 + 0.275)^2 + (-3.4 + 0.275)^2}{4} = 6.861$$



$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \frac{2.7 + 0.275}{\sqrt{6.861 + 10^{-9}}} = 1.13$$

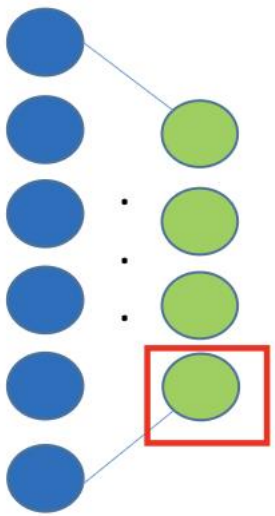
$$y_i \leftarrow \gamma\hat{x}_i + \beta \qquad 1.13\theta_1 + \theta_2$$



$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \frac{-2.3+1.9+2.7-3.4}{4} = -0.275$$

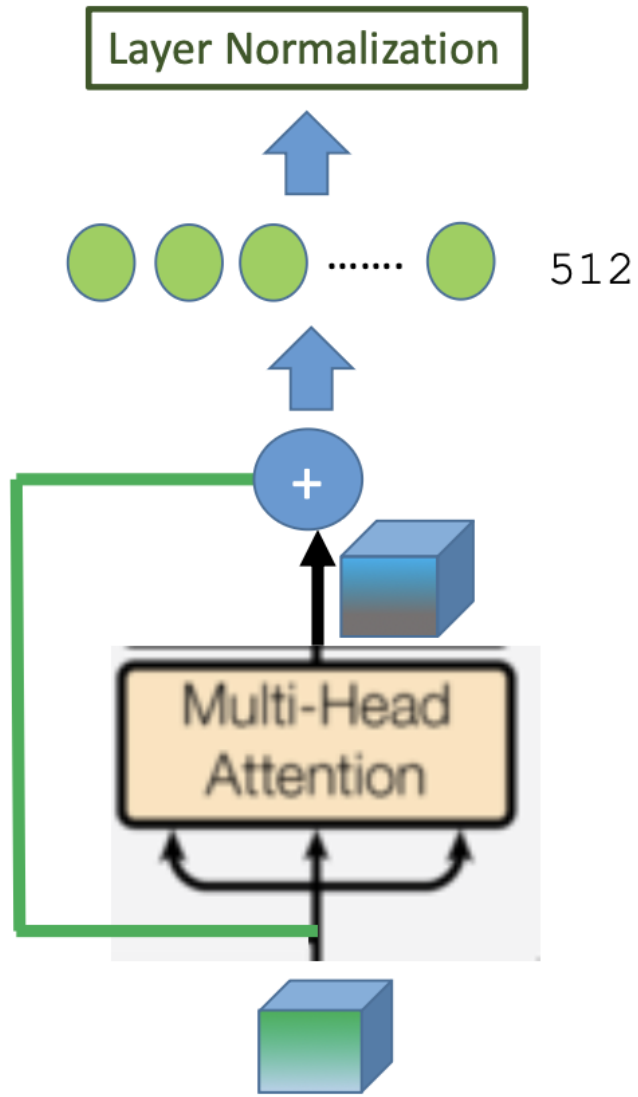
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i-\mu_{\mathcal{B}})^2$$

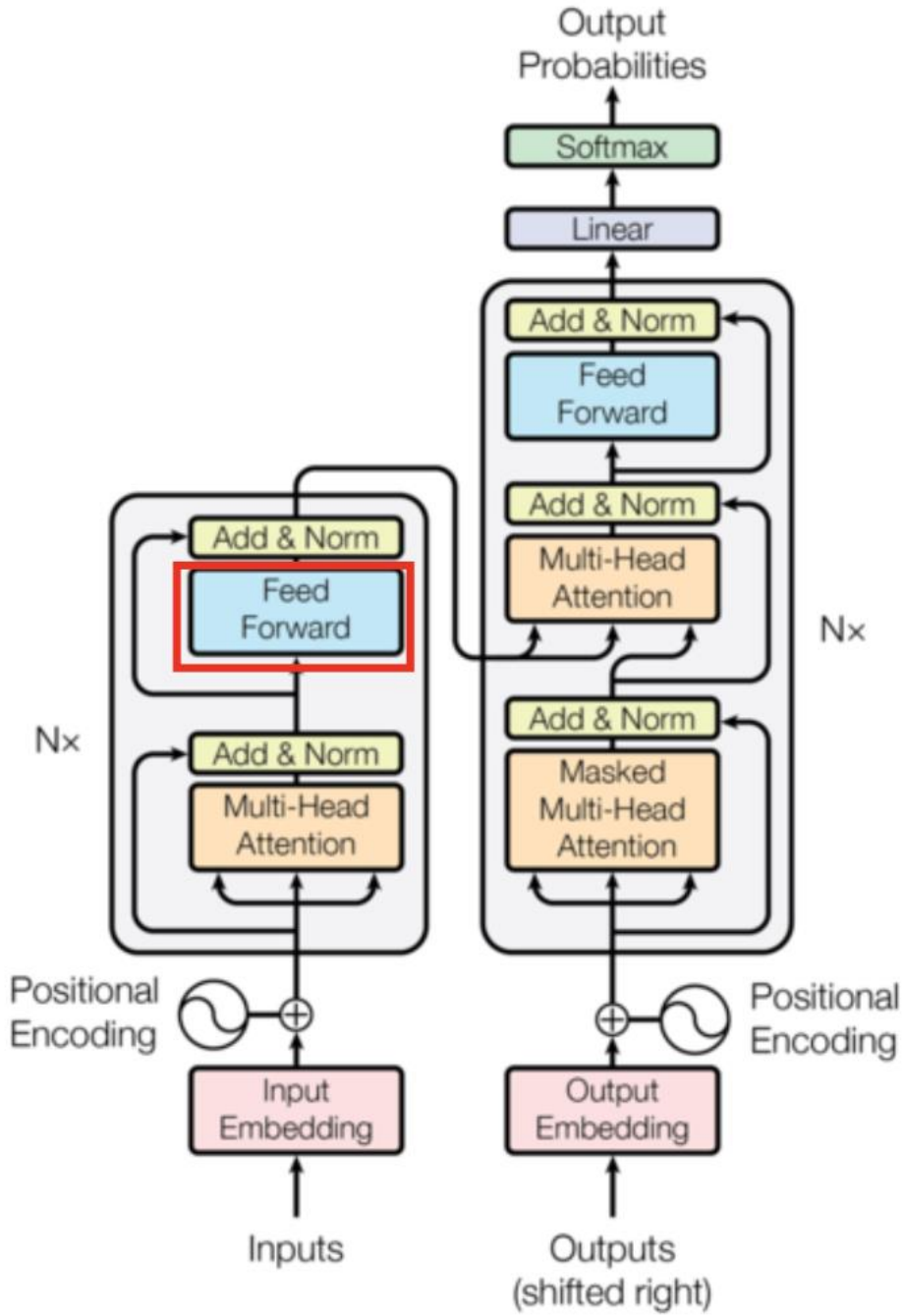
$$\frac{(-2.3+0.275)^2+(1.9+0.275)^2+(2.7+0.275)^2+(-3.4+0.275)^2}{4} = 6.861$$



$$\hat{x}_i \leftarrow \frac{x_i-\mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}} \qquad \frac{-3.4+0.275}{\sqrt{6.861+10^{-9}}} = -1.19$$

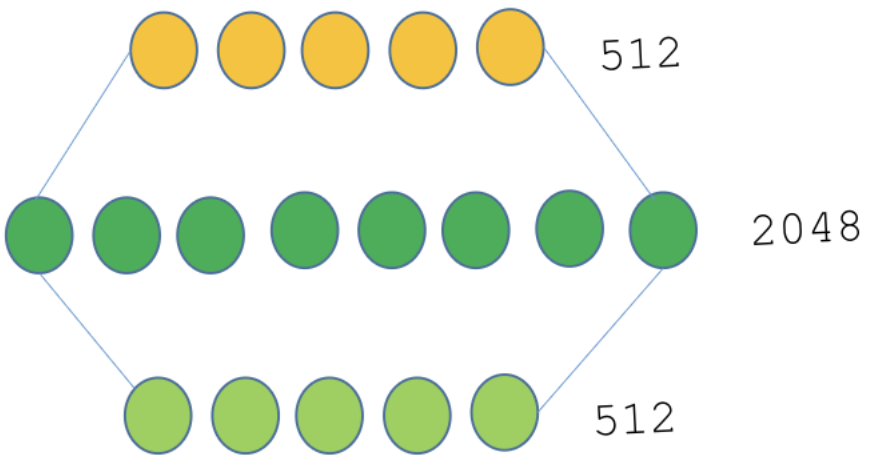
$$y_i \leftarrow \gamma\hat{x}_i+\beta \qquad -1.19\theta_1+\theta_2$$

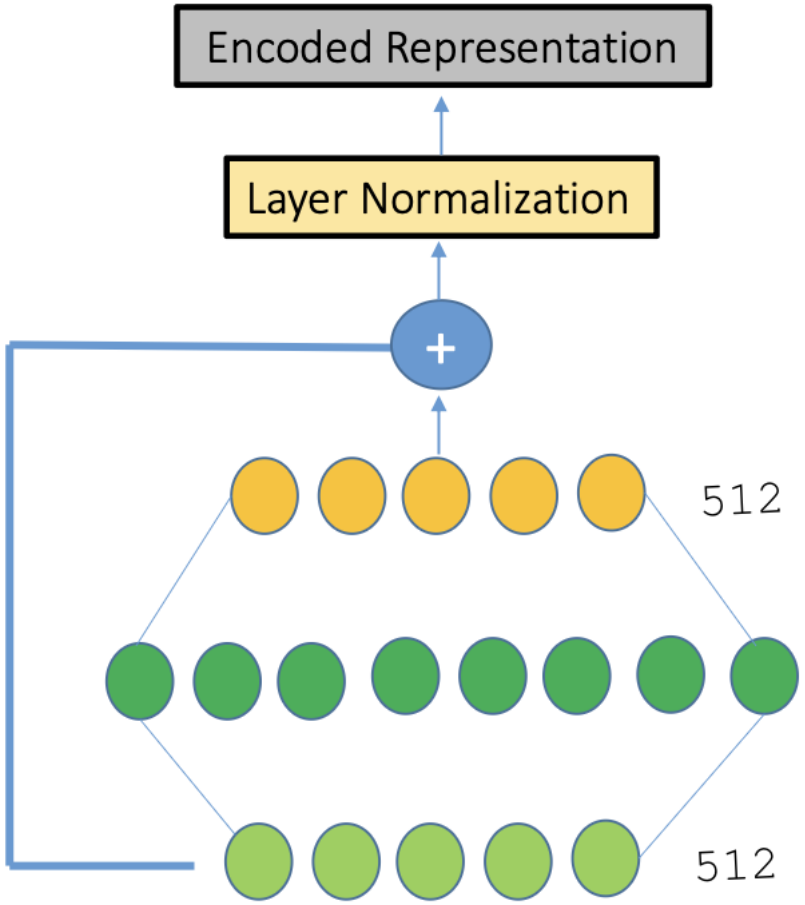

Layer Normalization
512
+
Multi-Head
Attention

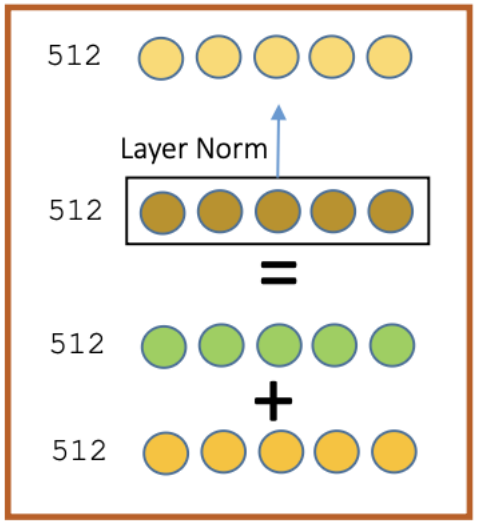

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

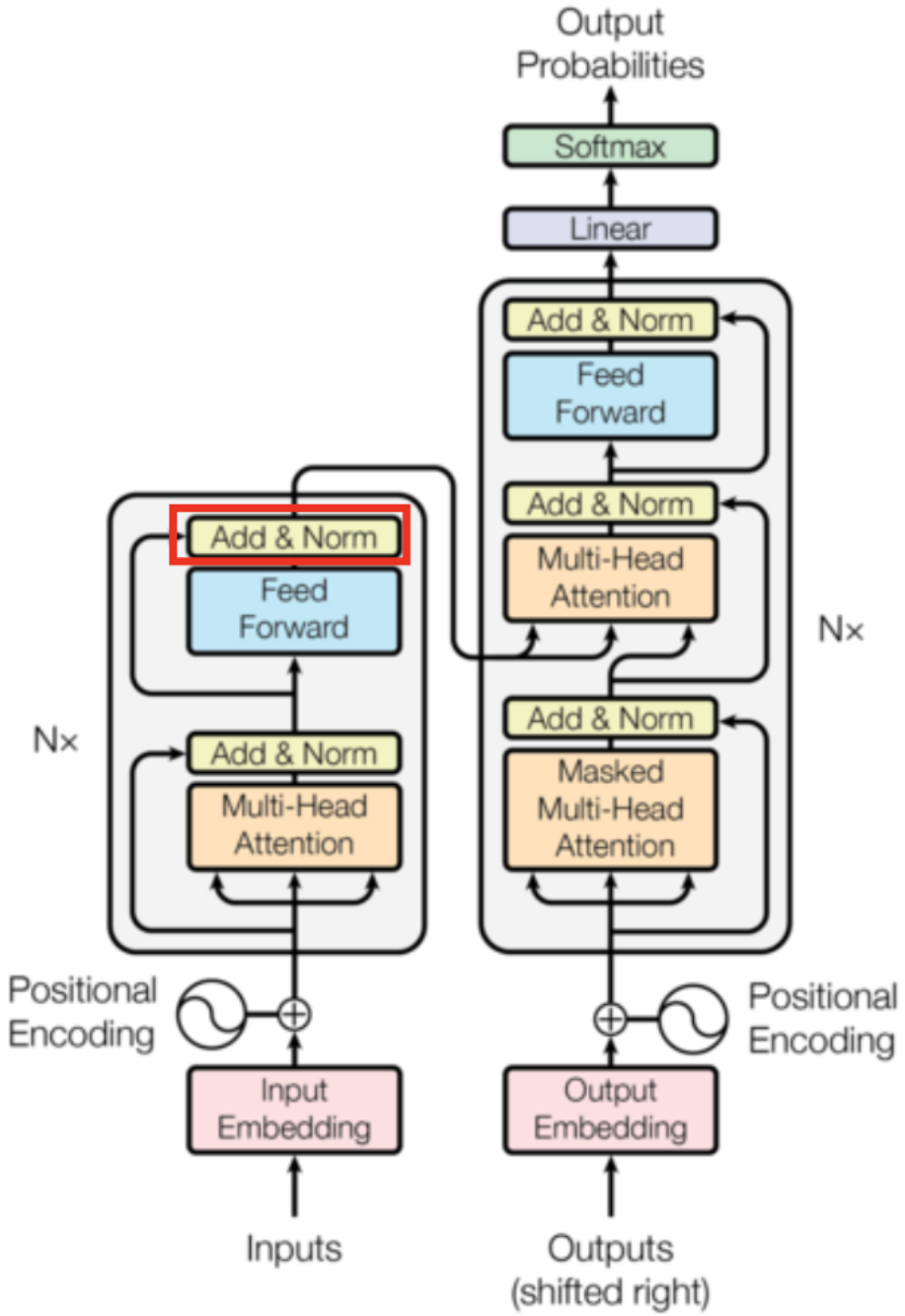
Feed Forward

- 2 linear (feed forward) layers to learn more encoding information.
- The dimensions set in the paper are 512 → 2048 for the first layer, and 2048 → 512 for the second layer.

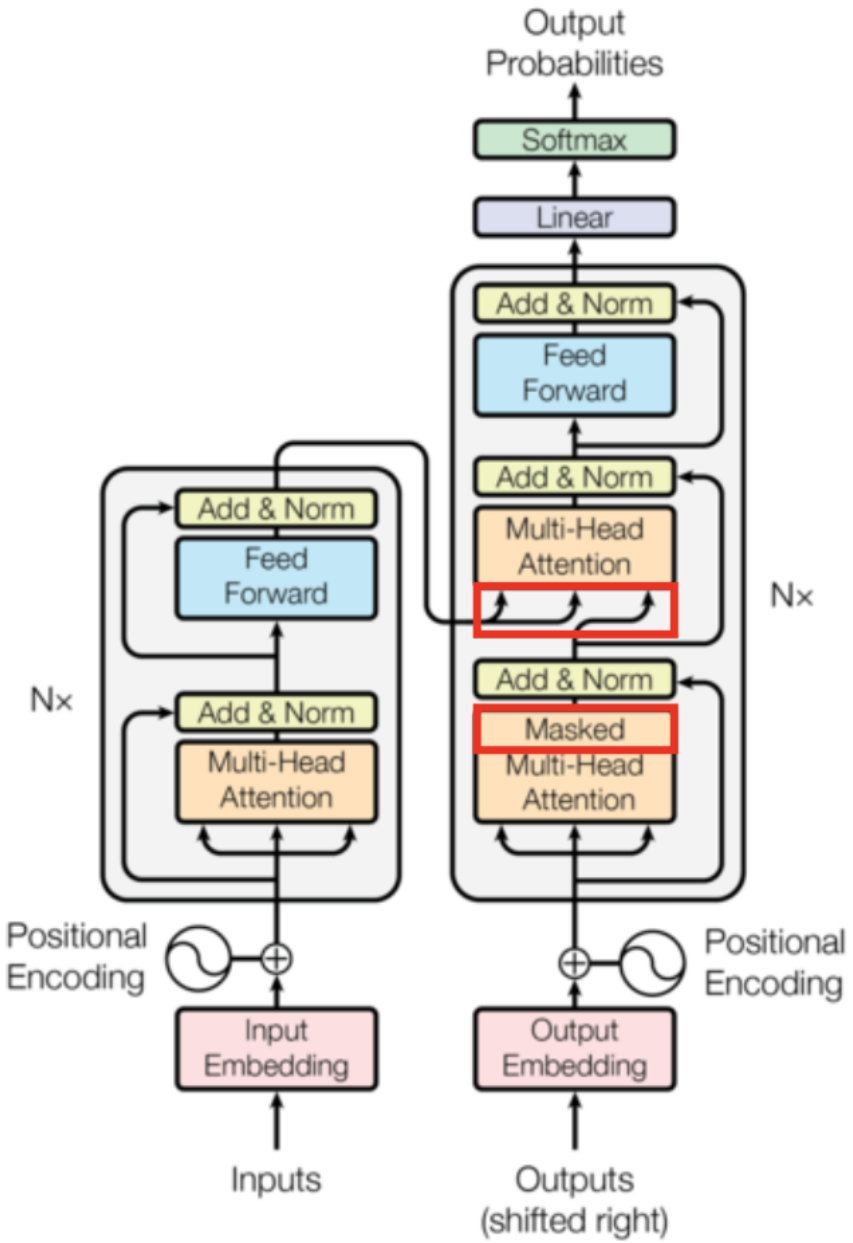



Encoded Representation
Layer Normalization
+
512
2048
512


512
Layer Norm
512
=
512
+
512


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Decoder: Same as encoder with 2 slight differences



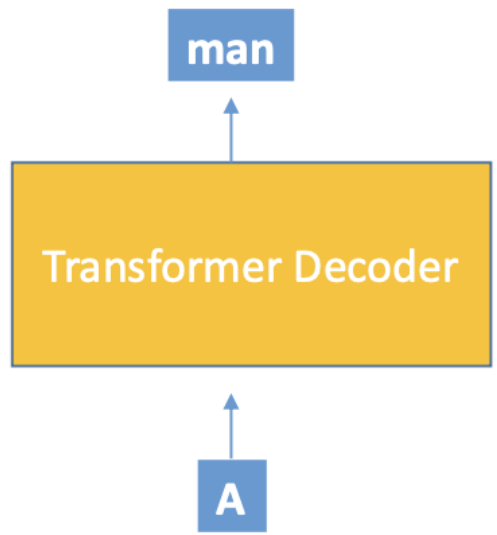
Masked Multi-Head Attention: *Subsequent Mask*

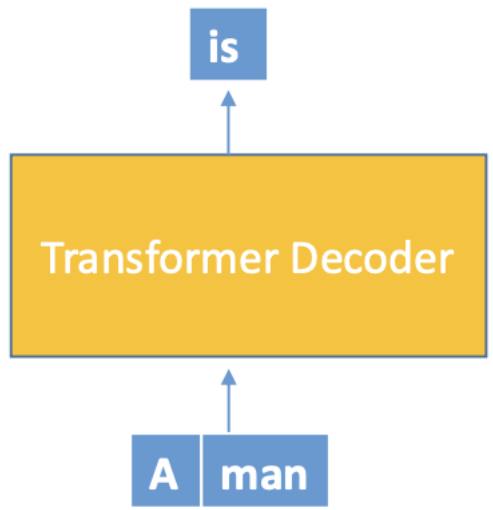
- For self-attention, we don't want our decoder to attend to future word. Otherwise, the model will cheat and learn to look at future words. At testing time, we don't have future words! We are predicting one word at a time (autoregression) (running the decoder for a number of timesteps, just like an LSTM at testing time). So this will be incompatible during testing (inference). Therefore, the decoder is only allowed to attend to earlier positions. During testing time, it can only attend to what has been generated so far. So we need to resemble the testing time scenario during training as well.

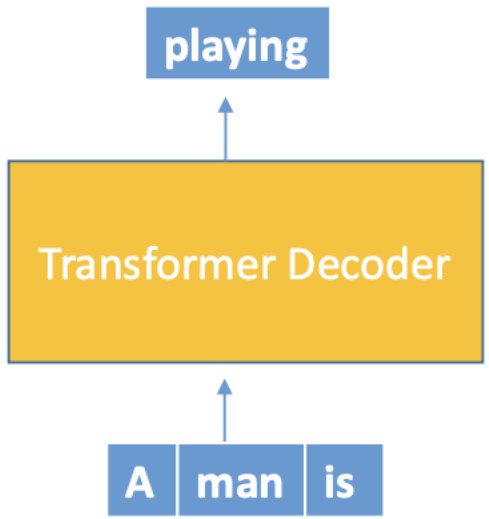
5 words
(5,5)

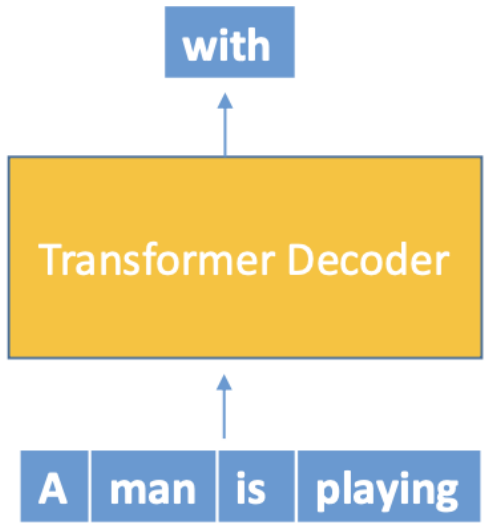
Word 1 `[[[1, 0, 0, 0, 0],`
Word 2 `[1, 1, 0, 0, 0],`
Word 3 `[1, 1, 1, 0, 0],`
Word 4 `[1, 1, 1, 1, 0],`
Word 5 `[1, 1, 1, 1, 1]]]`

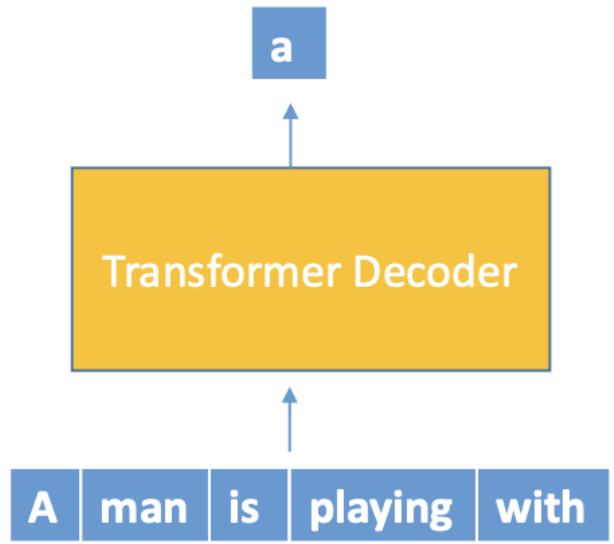
At testing time

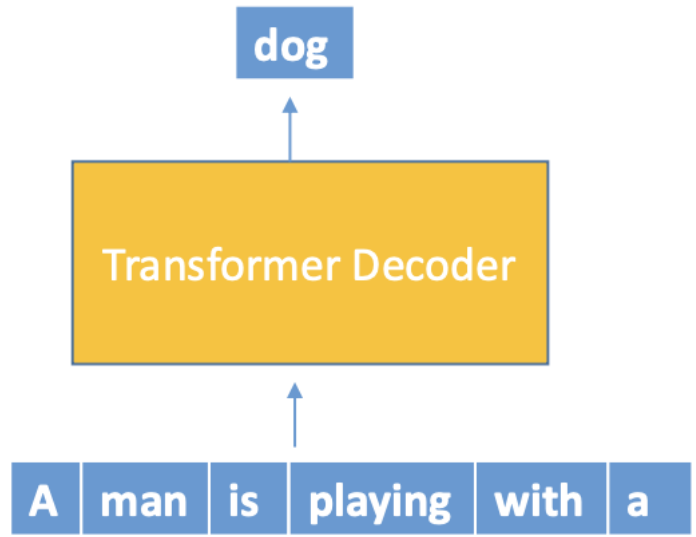


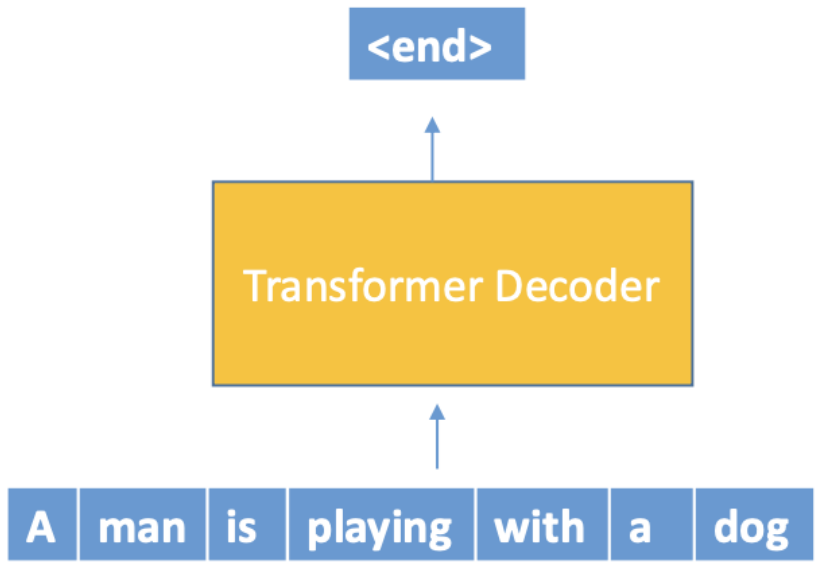

is
Transformer Decoder
A
man


playing
Transformer Decoder
A
man
is


with
Transformer Decoder
A
man
is
playing


a
Transformer Decoder
A
man
is
playing
with


dog
Transformer Decoder
A
man
is
playing
with
a

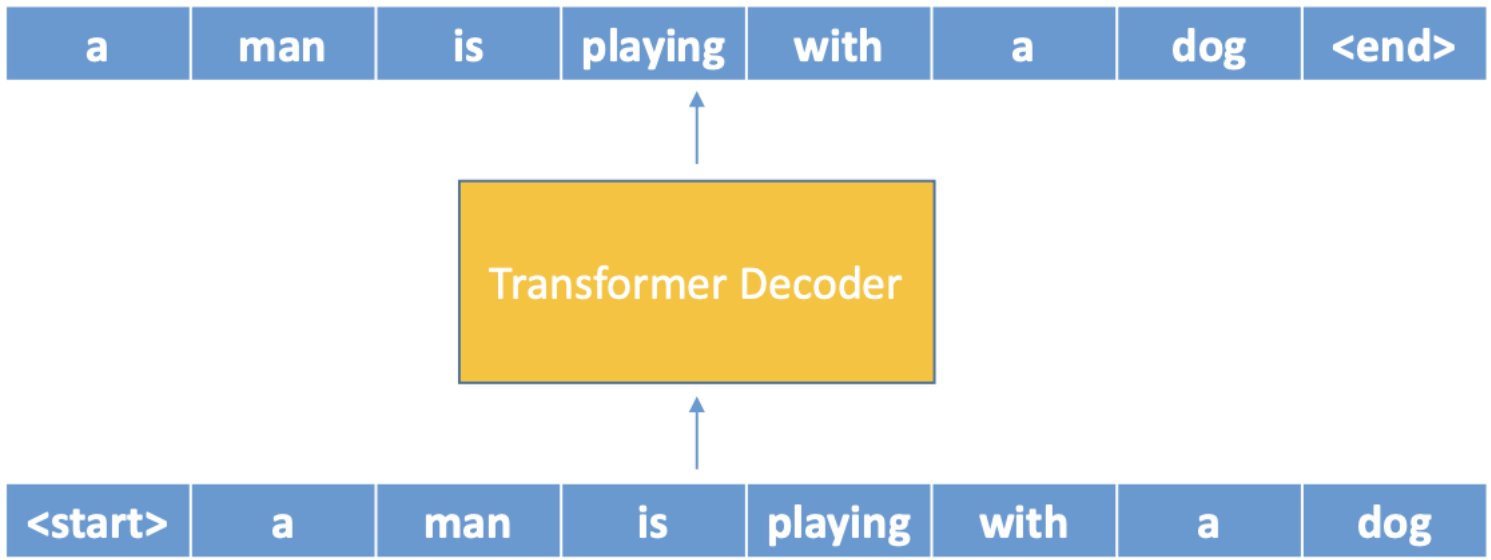

<end>
Transformer Decoder
A
man
is
playing
with
a
dog

At training time

Sentence: <start> A man is playing with a dog <end>

Input: <start> A man is playing with a dog

Label: A man is playing with a dog <end>

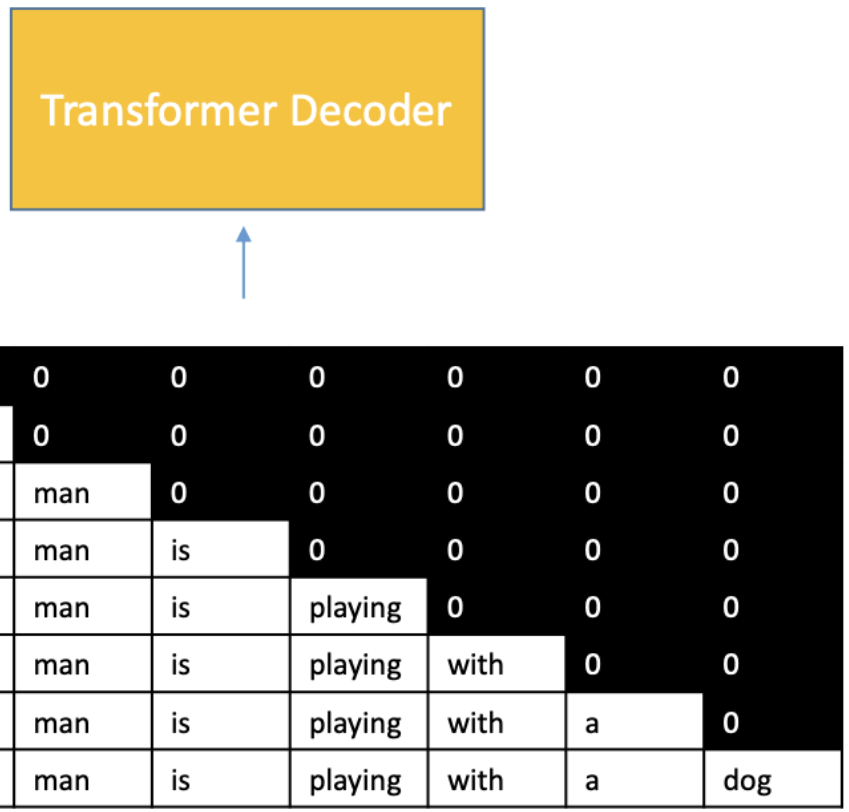


At training time

Sentence: <start> A man is playing with a dog <end>

Input: <start> A man is playing with a dog

Label: A man is playing with a dog <end>



Decoder: Queries, Keys and Values of Multi-Head Attention

The Key and Values are the encoded representation we got from the encoder.

The Query is the result we got from self-attention.

This is similar to LSTM attention, except that in LSTM attention, the query comes from decoder LSTM output rather than the decoder self-attention output in a transformer.

The Keys and Values come from the encoder (they are the encoded representation), exactly like in LSTM attention.