

# Natural Language Processing

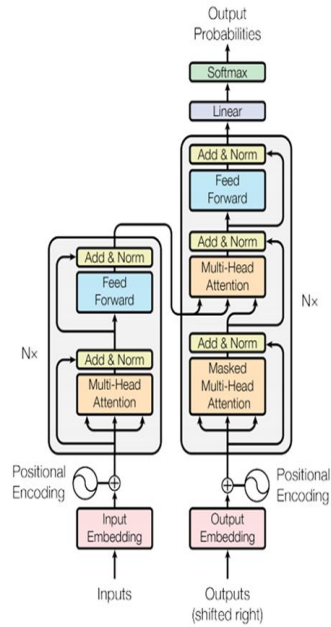
## Module 5: Transformers

# Contents

- Introduction
- Self-attention
- Multi-head attention
- Residual and Layer Normalisation
- Position Wise Feedforward Network
- Transformer Encoder
- Masked Multi-head attention (Transformer Decoder)
- Transformer Decoder
- Training Transformers

# Introduction

- Introduced in 2017
- Now underpins almost all SoTA models in ML
- Effectively scalable (e.g. GPT3)
- Non-recurrent model - purely based on attention



3

## SPEAK SLOWLY

- Ask audience:
  - 1. Has anyone heard of a Transformer
  - 2. Has anyone read up about how it works?
  - 3. ???

# Some things Transformers can do

## Transformer Encoder

### Tasks:

- Classification
- Masked language modelling
- Named entity recognition

### Popular models:

- BERT



- RoBERTa



- DeBERTa






## Some things Transformers can do

### Transformer Encoder-decoder

#### Tasks:

- Translation
- Summarization
- Free-form QA

#### Popular models:

- Original Transformer 
- T5 
- BART 

## Some things Transformers can do

### **Transformer Decoder**

#### *Tasks:*

- Causal language modelling
- Text generation

#### *Popular models:*

- GPT (1-3)



# Some things Transformers can do

## Vision Transformers

### Tasks:

- Visual classification
- Object detection
- Image generation

### Popular models:

- ViT



- DINO



- DETR



# Some things Transformers can do

## Multimodal Transformers

### Tasks:

- Image captioning/descriptions
- Language-based image querying
- Generating images from text descriptions

### Popular models:

- ViLBERT
- CLIP
- DALL-E





# Some things Transformers can do

## Other Uses

### Tasks:
















- Predicting protein structures
- Basic reasoning tasks

### Popular models:

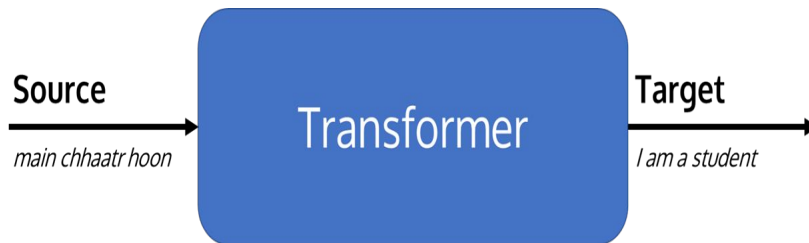
- AlphaFold
- Maths word problems



## Some things Transformers can do

| Encoder             |   | Encoder-decoder         |   | Decoder             |   |
|---------------------|---|-------------------------|---|---------------------|---|
| BERT                |  | Original Transformer    |  | GPT                 |  |
| RoBERTa             |  | T5                      |  |                     |   |
| DeBERTa             |  | BART                    |  |                     |   |
| Vision Transformers |   | Multimodal Transformers |   | Other               |   |
| ViT                 |  | ViLBERT                 |  | AlphaFold           |  |
| DINO                |  | CLIP                    |  | Maths word problems |  |
| DETR                |  | DALL-E                  |  |                     |   |

## Holistically

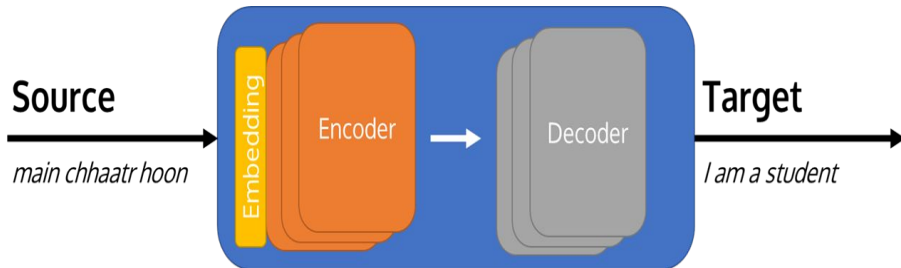


11

### SPEAK SLOW

- Like with other translation models we've looked at previously, the Transformer takes in a source sentence, and generates a target

## Holistically

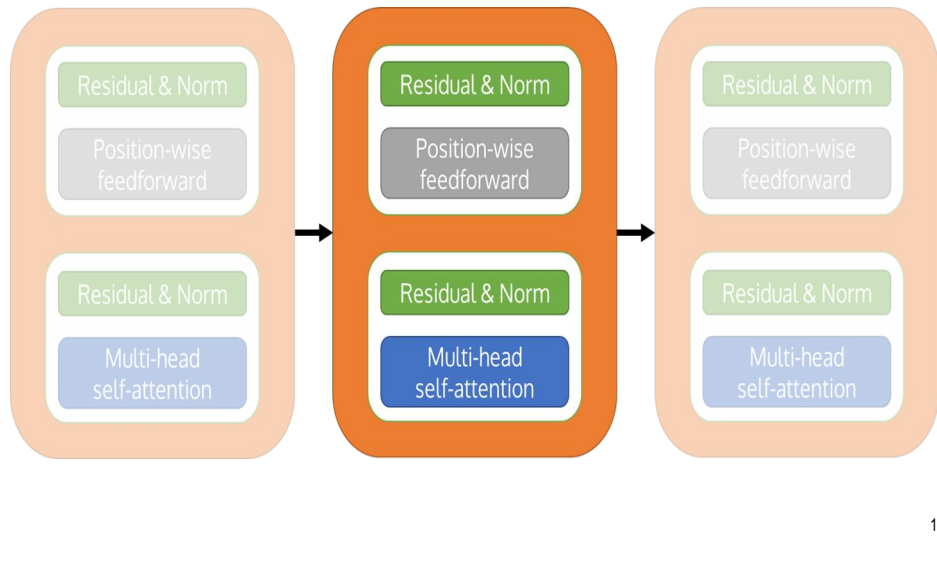


12

### SPEAK SLOW

- Inside the transformer, we have a stack of encoder layers.
- We also have a stack of decoder layers
- At some point during decoding, we will use the encoded output to help us generate the target
- Note that the output of one encoder layer is sent as input to the next encoder layer. Similarly, the output of one decoder layer is sent to the next decoder layer as input

## Holistically (Encoder)

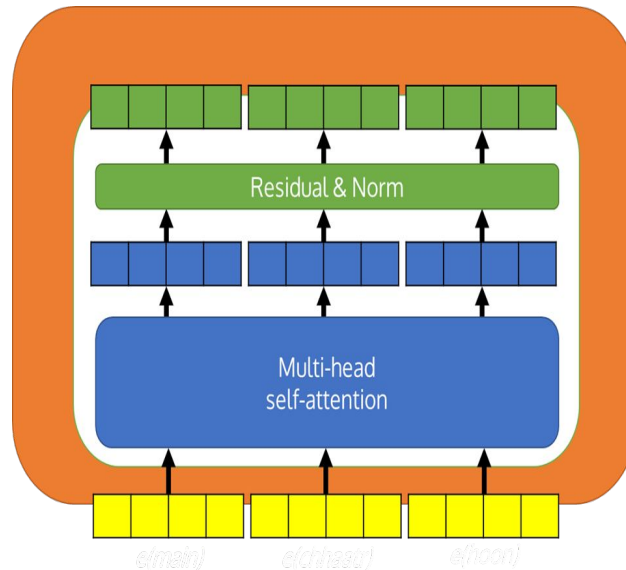


13

### SPEAK SLOW

- We're going to take a look at the Encoder first.
- Each encoder layer consists of 2 sublayers. The first sublayer contains a multi-head self-attention module, and the second contains a position-wise feedforward network.
- A common theme in Transformers is that each sublayer will have a residual connection and layer normalization
- As I mentioned, the output of one encoder layer is going to be the input to the next encoder layer
  - I.e. the output of the "Residual & Norm" after the Position-wise feedforward will form the input to the Multi-head self-attention in the next encoder layer
-

## Holistically (First sublayer)

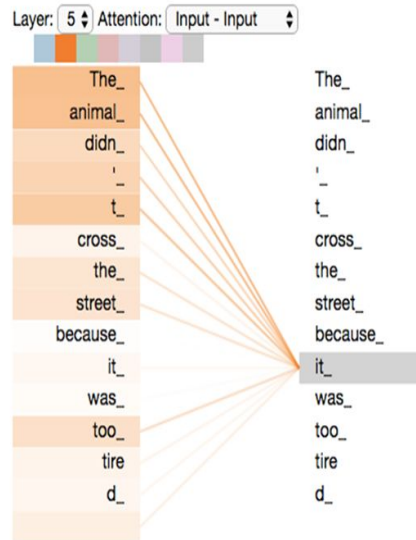


14

- SPEAK SLOW
- So let's take a deeper look at the first sublayer in an encoder: The one which contains the multi-head self-attention.
- We're going to receive some input - represented here in yellow. The input will be an encoding of each of our words. So here, we have 3 words represented with 4 dimensions. More generally, you would have  $S$  words in the input sequence, each represented with  $D$  dimensionality.
- The MHA module processes the input and outputs another set of  $S \times D$  encodings.
- These encodings get sent to the Residual & Norm, which outputs another set of  $S \times D$  encodings.
- Conceptually this should make transformers easy to work with - mostly everything in the encoder stays as  $S \times D$

## Self Attention

Mechanism that allows **each input** in a sequence to look at the **whole sequence** to compute a **representation of the sequence**.



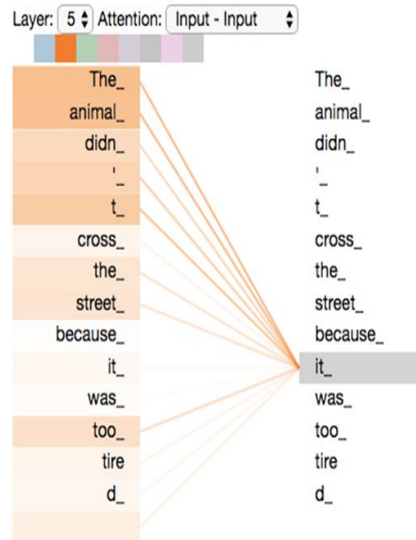
15

- SPEAK SLOW
- The definition is weird the first time you come across it. Let's break it down.
- Each input = each word. So it's saying that each word looks at every other word (including itself).
  - Each word therefore gets a representation based on all the other words in the input sequence.
  - The representation of the sequence is  $S \times D$ . So we have  $S$  words (e.g. The animal didn etc.). Each  $s$  in  $S$  has looked at every other word to compute its  $D$ -dimensional representation

## Self Attention

Mechanism that allows **each input** in a sequence to look at the **whole sequence** to compute a **representation of the sequence**.

- “the animal didn’t cross the street because **it** was too tired.”
  - What does **it** refer to?



16

### SPEAK SLOW

- As an example, the sentence we've been given is “the animal didn’t cross the street because **it** was too tired”.
- For humans, understanding that “it” refers to the animal is natural. Not so much for computers.
- Think back to RNNs. If we were processing this sequence, we would have some hidden state representation for the word “it”.
  - But in this RNN case, the word “it” hasn’t been explicitly forced to look at other words in the sequence to align itself with them.
  - In other words, the importance of “animal” to the word “it” would be the same as “because”
- Transformers are designed such that the hidden state representation of the word “it” would be influenced more by “animal” than “because”.
  - Look at the definition of self-attention: each input (e.g. consider “it”) looks at the whole sequence (e.g. the, animal, didn etc) and then computes a representation of that word, based on how important each of the other words are to it.



How?

The diagram shows the Attention function formula:  $\text{Attention}(Q, K, V) = \sigma \left( \frac{QK^T}{\sqrt{d_h}} \right) V$ . Arrows point from the labels 'Queries', 'Keys', and 'Values' to the variables  $Q$ ,  $K$ , and  $V$  respectively. An arrow points from the label 'Softmax' to the  $\sigma$  symbol.

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

17

## SPEAK SLOW

- We're gonna break this down in the upcoming slides, but there are some key things to note.
- Attention is a function of 3 variables: Q, K and V. Or Queries, Keys and Values.
- We also have a softmax function, and this sqrt of  $d_h$ . Again, we'll break this down over the upcoming slides

## The intuition

1. Define a Q
2. Calculate similarity of Q against the Ks
3. Output is the weighted sum of Vs

| Colour (K) | RGB (V)         |
|------------|-----------------|
| Red        | (255, 0, 0)     |
| Green      | (0, 255, 0)     |
| Blue       | (0, 0, 255)     |
| Yellow     | (255, 255, 0)   |
| Black      | (0, 0, 0)       |
| White      | (255, 255, 255) |

18

### SPEAK SLOW

- The easiest way to start building an understanding of self-attention is to think of the process of looking up a value in a dictionary
- Let's say we had a dictionary of colours. Our keys would be the name of the colour, and our values would be the RGB values of the colour
- Our query would be the colour we want to find.

## The intuition

1. Define a Q
2. Calculate similarity of Q against the Ks
3. Output is the weighted sum of Vs

### **Q = Yellow**

- Exact match with K = Yellow
- Therefore value = (255, 255, 0)

| Colour (K) | RGB (V)         |
|------------|-----------------|
| Red        | (255, 0, 0)     |
| Green      | (0, 255, 0)     |
| Blue       | (0, 0, 255)     |
| Yellow     | (255, 255, 0)   |
| Black      | (0, 0, 0)       |
| White      | (255, 255, 255) |

19

## SPEAK SLOW

- So, in the case that we wanted to find the value of the colour yellow, we would do a lookup of our dictionary with Q=Yellow
- We have an exact match with K = Yellow, and therefore we know the value should be (255, 255, 0)

## The intuition

1. Define a Q
2. Calculate similarity of Q against the Ks
3. Output is the weighted sum of Vs

### Q = Yellow

- Exact match with K = Yellow
- Therefore value = (255, 255, 0)

### Q = Orange

- 50% K = Red, 50% K = Yellow
- Value =  $0.5(255, 0, 0) + 0.5(255, 255, 0)$   
= (255, 128, 0)

| Colour (K) | RGB (V)         |
|------------|-----------------|
| Red        | (255, 0, 0)     |
| Green      | (0, 255, 0)     |
| Blue       | (0, 0, 255)     |
| Yellow     | (255, 255, 0)   |
| Black      | (0, 0, 0)       |
| White      | (255, 255, 255) |

20

## SPEAK SLOW

- In the case that our Query is orange... well, we know that orange is a combination of Red and Yellow.
- Our keys don't actually contain orange. But they do contain red and yellow, and we know that we can create orange with 50% Red and 50% Yellow.
  - Right now we're just running through it intuitively.. So assume for a second that this weighting information has been given to us
- We then apply this weighting over the values for the relevant keys. So 50% red = (255, 0, 0), and 50% yellow = (255, 255, 0)
- Orange is then the combination of these weighted values

## Questions so far?

- What is the definition of self-attention?
- What are the 3 main variables required to compute an attention representation?
- Can you describe the responsibility of each of those variables?

SPEAK SLOW

## Now with vectors

**Q = [0, 10, 0]**

- Exact match with [0, 10, 0]
- V = [10, 0, 2]

| Key        | Value        |
|------------|--------------|
| [10, 0, 0] | [1, 0, 1]    |
| [0, 10, 0] | [10, 0, 2]   |
| [0, 0, 10] | [100, 5, 0]  |
| [0, 0, 10] | [1000, 6, 0] |

SPEAK SLOW

## Now with vectors

**Q = [0, 0, 10]**

- Matches the two [0, 0, 10]'s
- $V = 0.5*[100, 5, 0] + 0.5*[1000, 6, 0]$   
= [550, 5.5, 0]

| Key        | Value        |
|------------|--------------|
| [10, 0, 0] | [1, 0, 1]    |
| [0, 10, 0] | [10, 0, 2]   |
| [0, 0, 10] | [100, 5, 0]  |
| [0, 0, 10] | [1000, 6, 0] |

23

### SPEAK SLOW

- Here we're starting to build a stronger intuition of how the attention mechanism will actually work in practise
- We have Query = [0, 0, 10]
- If we work out a similarity score for all our keys, we will have a 100% match with both of the [0, 0, 10].
- We will therefore consider each of these values with equal importance
- The output will therefore be 50% weighting to [100, 5, 0] and 50% weighting to [1000, 6, 0]

## Now with vectors

**Q = [10, 10, 0]**

- Matches [10, 0, 0] and [0, 10, 0]
- $V = 0.5 * [1, 0, 1] + 0.5 * [10, 0, 2]$   
= [5.5, 0, 1.5]

| Key        | Value        |
|------------|--------------|
| [10, 0, 0] | [1, 0, 1]    |
| [0, 10, 0] | [10, 0, 2]   |
| [0, 0, 10] | [100, 5, 0]  |
| [0, 0, 10] | [1000, 6, 0] |

24

### SPEAK SLOW

- A similar phenomenon occurs for our Query vector [10, 10, 0]
- The keypoint is that we're working out a similarity score between our Q and all of our Ks
- Here, we're 50% similar to K=[10, 0, 0], and 50% similar to K=[0, 10, 0]
- The output would again be a weighted combination of the values of the matched keys



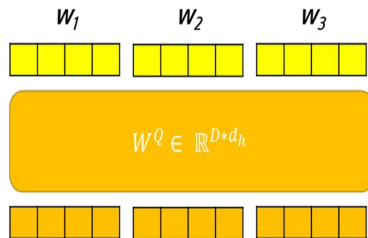
Questions so far?

25

SPEAK SLOW

# Self Attention

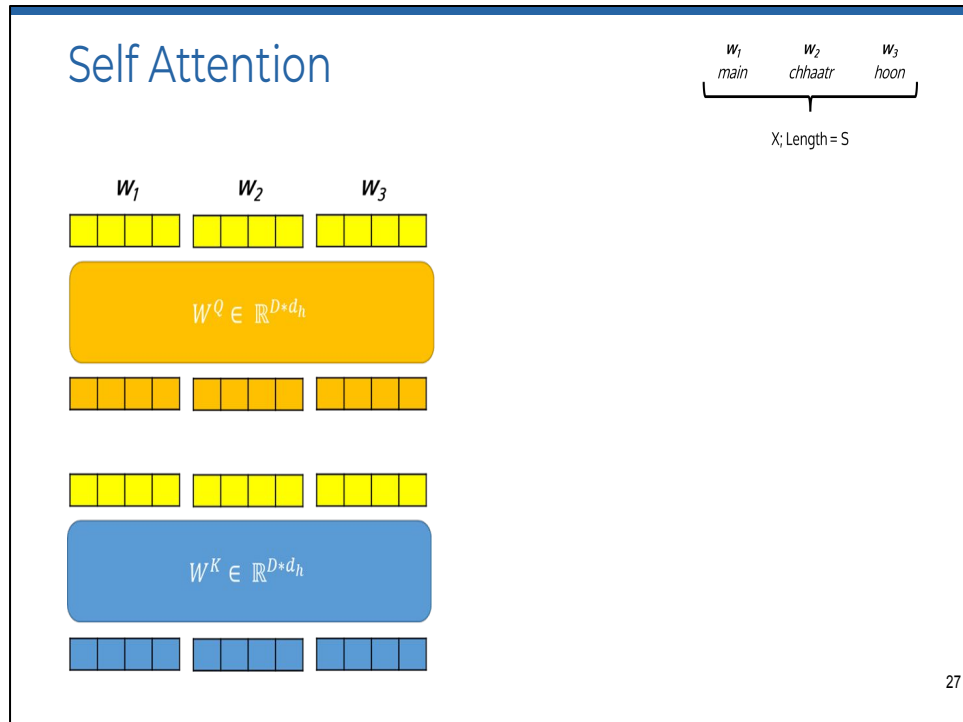
$w_1$     $w_2$     $w_3$   
 main   chhaatr   hoon  
 X; Length = S



26

## SPEAK SLOW

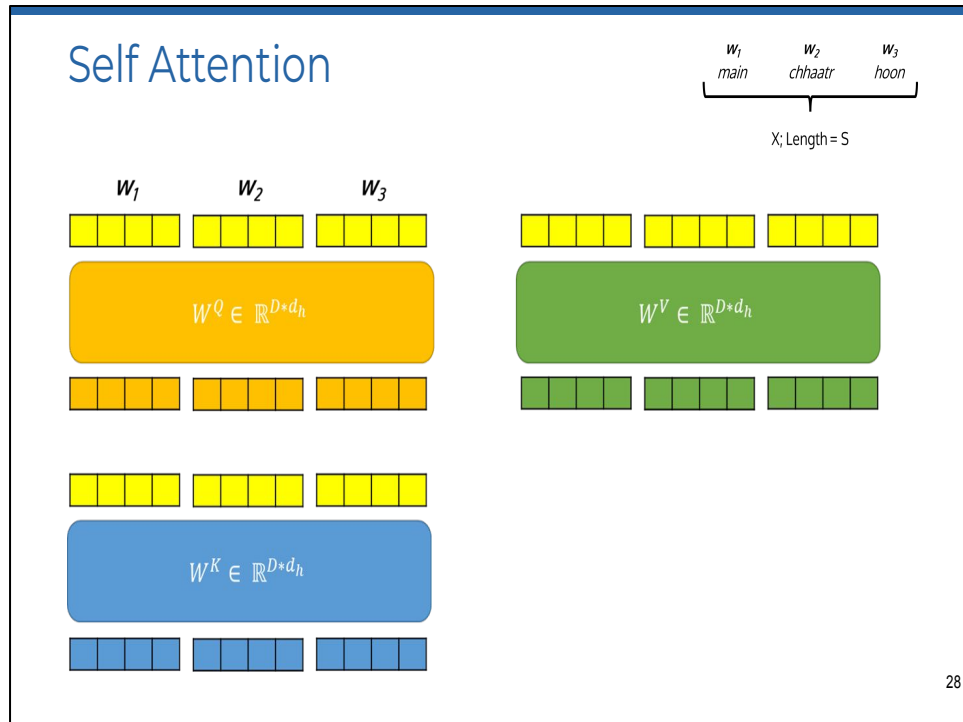
- The point of the previous slides was to show you that we have our Qs, Ks and Vs as vectors that we're comparing against each other.
- Now how do we obtain our Qs, Ks and Vs?
- Well... surprise surprise... they're learned
- Before we take a look at the general matrix form, let's take a look at how our model will work in vector form
- So, we have each one of our word encodings...  $w_1, w_2, w_3$ . We will call our input sequence X, and the length is S
- We will obtain a query vector for each one of our inputs. So  $q_1 = w_1 \cdot W^Q$ ;  $q_2 = w_2 \cdot W^Q$  etc
- Right now, pretend that  $d_h = D$ . We will come back to this later.



27

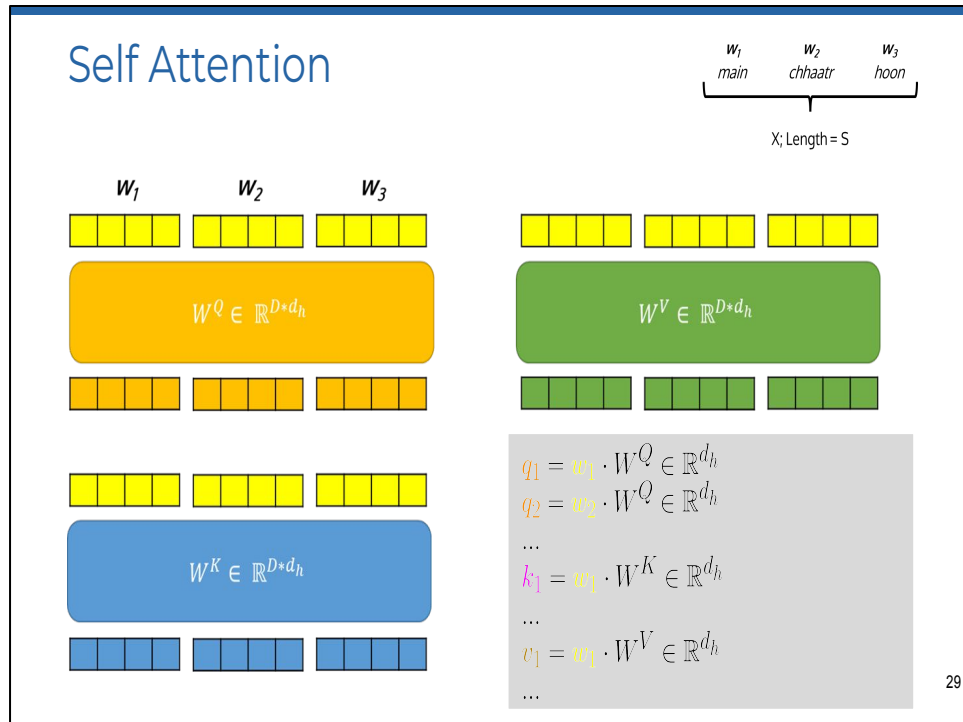
## SPEAK SLOW

- The same thing will happen for our Ks. Each one of our encoded inputs will be sent through WK to produce an S x dh representation. (Again, D = dh here)
- So we have k1, k2 etc. All the way up to K\_S.



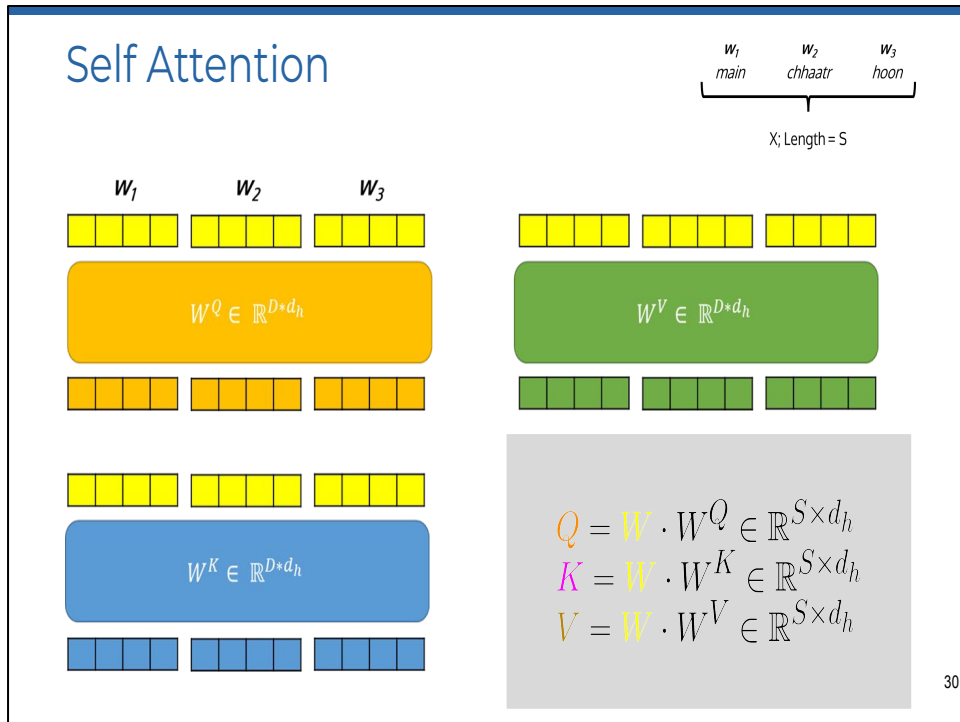
SPEAK SLOW

- And same thing for our Vs



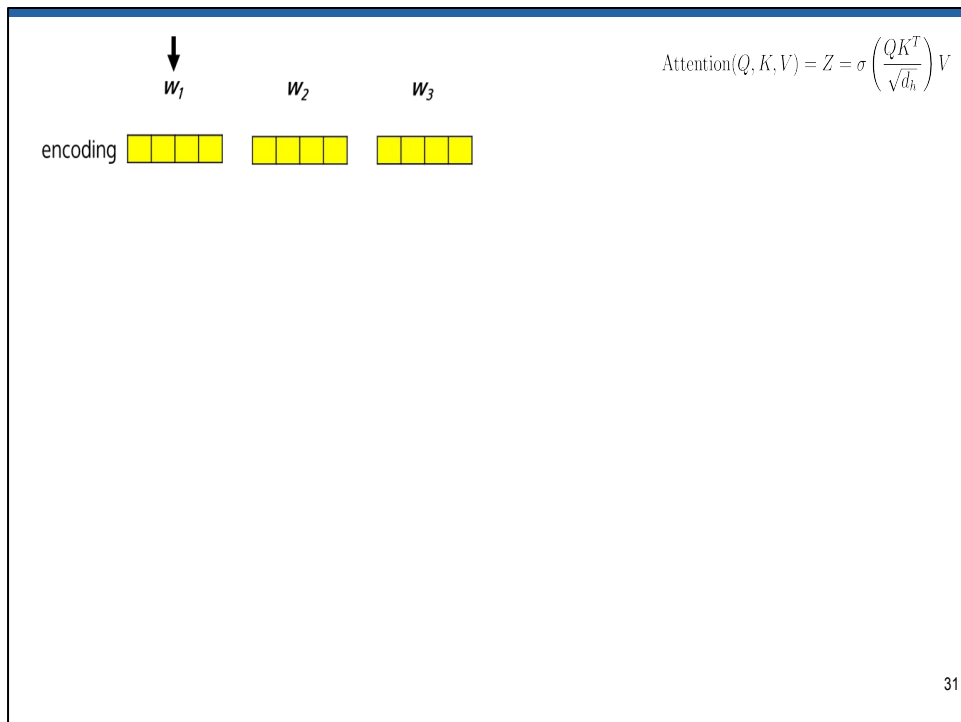
## SPEAK SLOW

- If we were to encode each one of our word encodings separately, we would have an encoding for each one of our 3 variables for each word we have. These vectors would be  $d_h$  dimensional



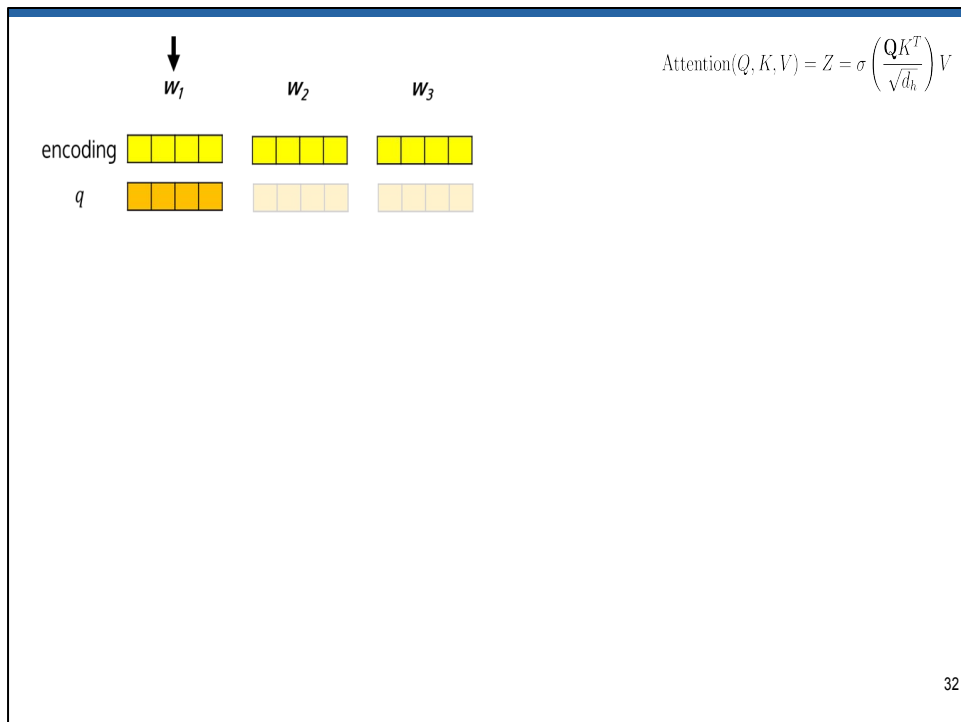
## SPEAK SLOW

- But we could apply these weight matrices directly over the  $S \times D$  matrix input of our word encodings
- Our Qs, Ks and Vs would then be an  $S \times d_h$  matrix. Again, for now, set  $d_h = D$
- To recap: We have a 3 different weights matrices responsible for learning our Qs, Ks and Vs.
- To actually obtain our Q, K and V matrix, we project our encodings thus far through each one of these weight matrices.
- Each weight matrix will transform the D-dimensional representation to  $d_h$ . Currently,  $d_h = D$ .
- So, the output of each weights matrix will be an  $S \times d_h$  matrix.



## SPEAK SLOW

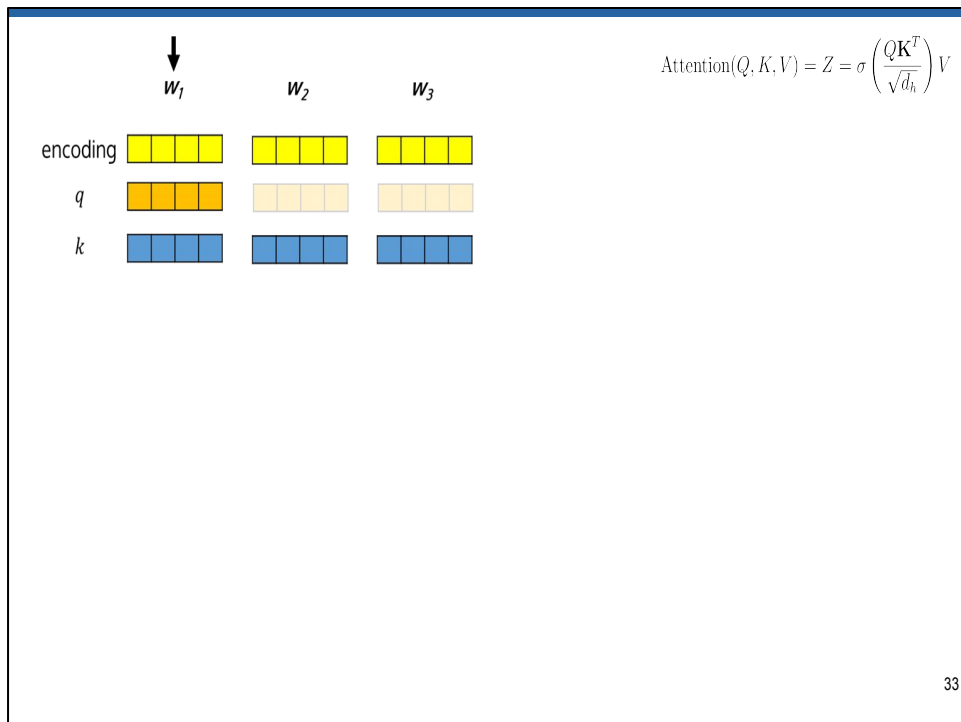
- Ok. So here's what we're going to do: Looking at attention in vector form first, we're going to work out our attention-ized representation for each word. Let's start with  $w_1$



## SPEAK SLOW

- Due to the projections I outlined on the previous slides, we have query vectors for all of our words.
- But right now, we only need the query vector for  $w_1$

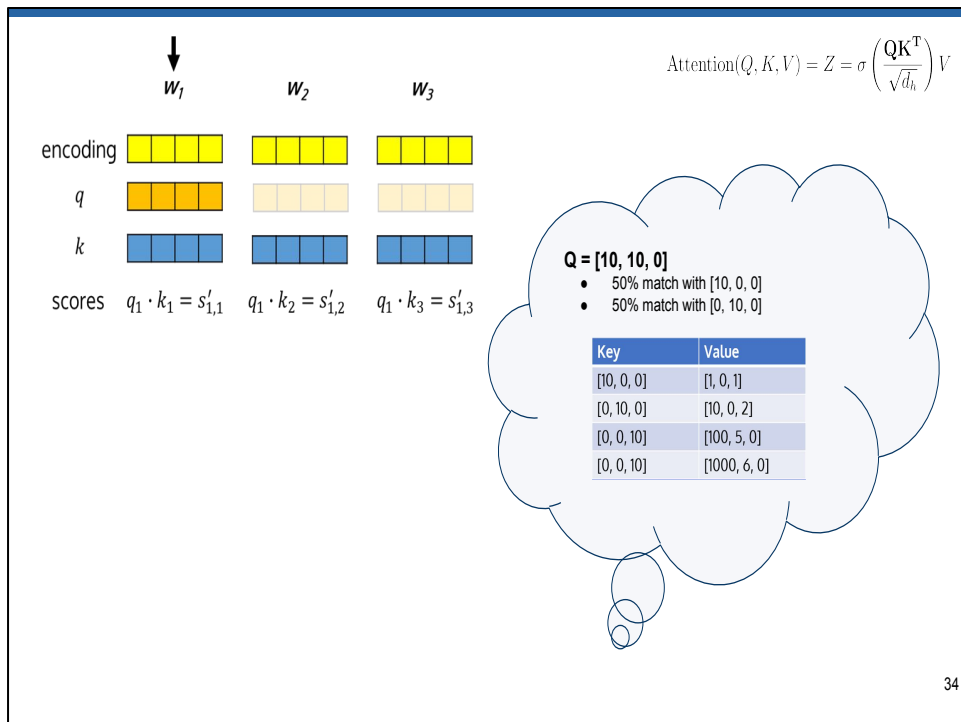




33

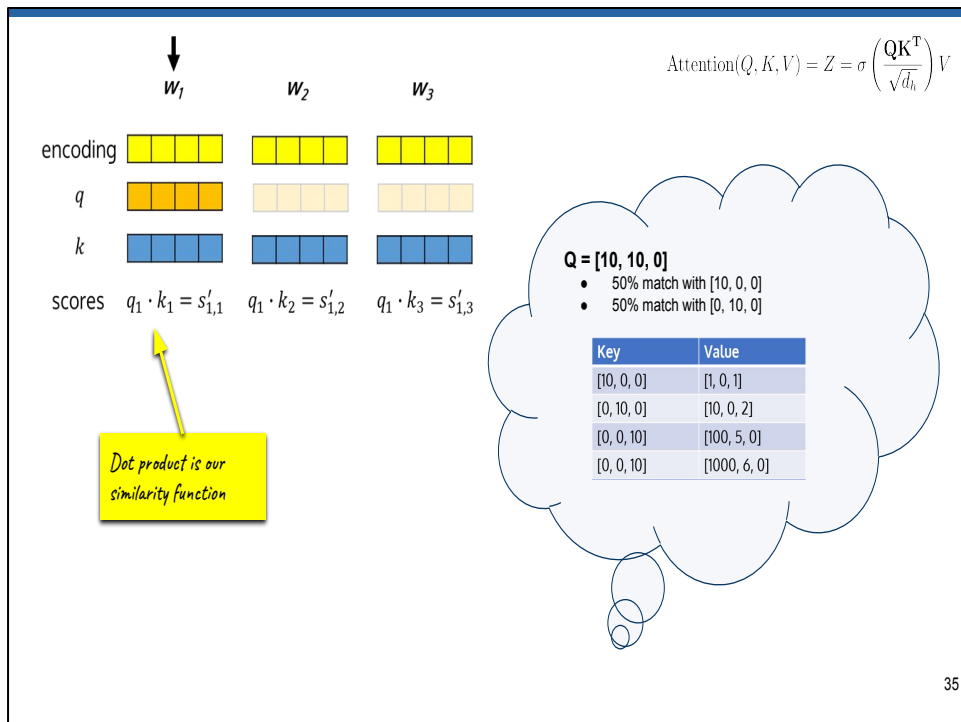
## SPEAK SLOW

- However, we're going to require all the key vectors.
- Why? (Ask audience)
- Well... think back to our "intuition examples". We were comparing the query to all the keys that we had. This allowed us to get an output which represented the weighted value of that query.
- So here, we want to get a representation for  $w_1$ . We will get this representation by comparing our query vector to all our keys.



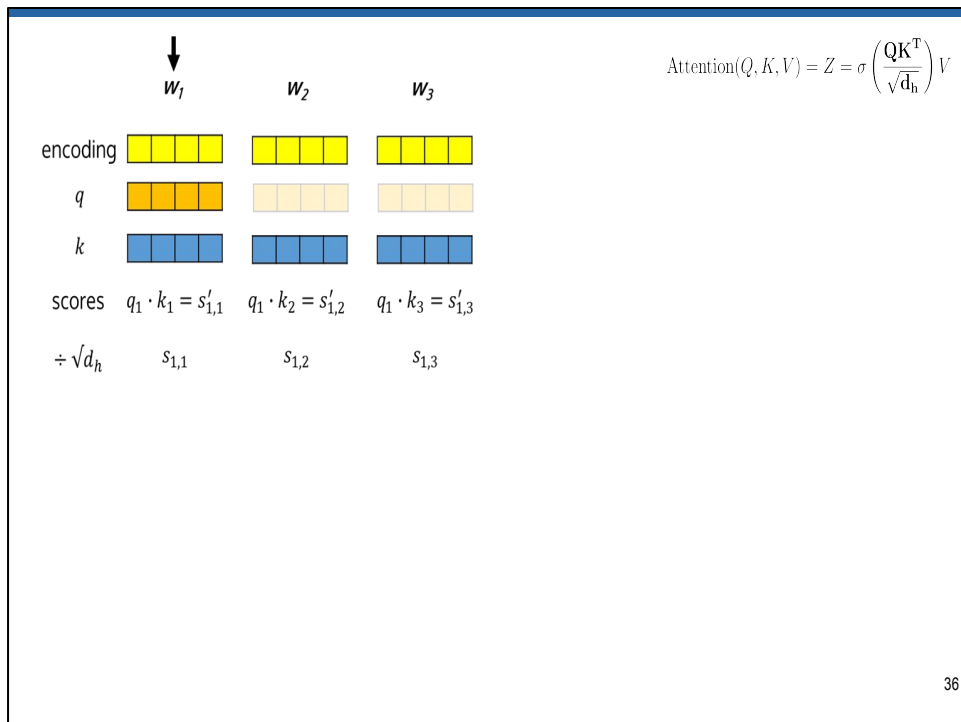
## SPEAK SLOW

- Ok. Now we're going to work out the similarity between the query vector we have, and ALL the key vectors.
- Think back to the intuition examples. We took our query vector and calculated the similarity of the vector to all the key vectors. We then obtained a weighting for how similar each of the key vectors were to the query vector
- The score here is NOT the weighting. However, we need it in our calculation of the weighting



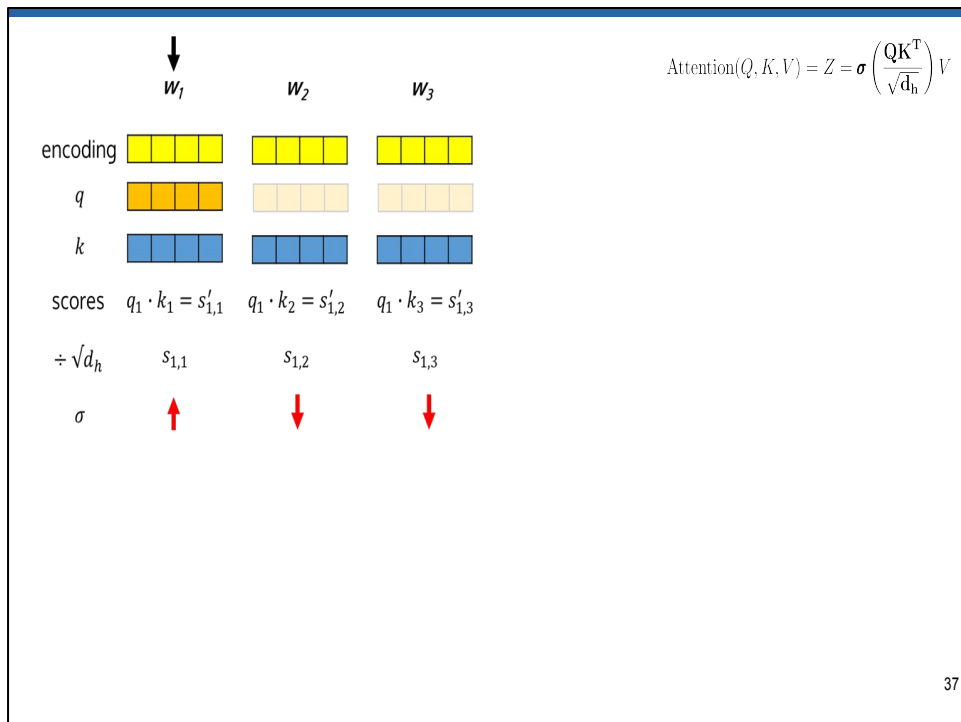
## SPEAK SLOW

- Our similarity function is the dot product
- So,  $s'_{1,1}$ ,  $s'_{1,2}$  etc. is the un-normalized similarity between our given query vector ( $q_1$ ), and key vectors ( $k_1$ ,  $k_2$  etc..)



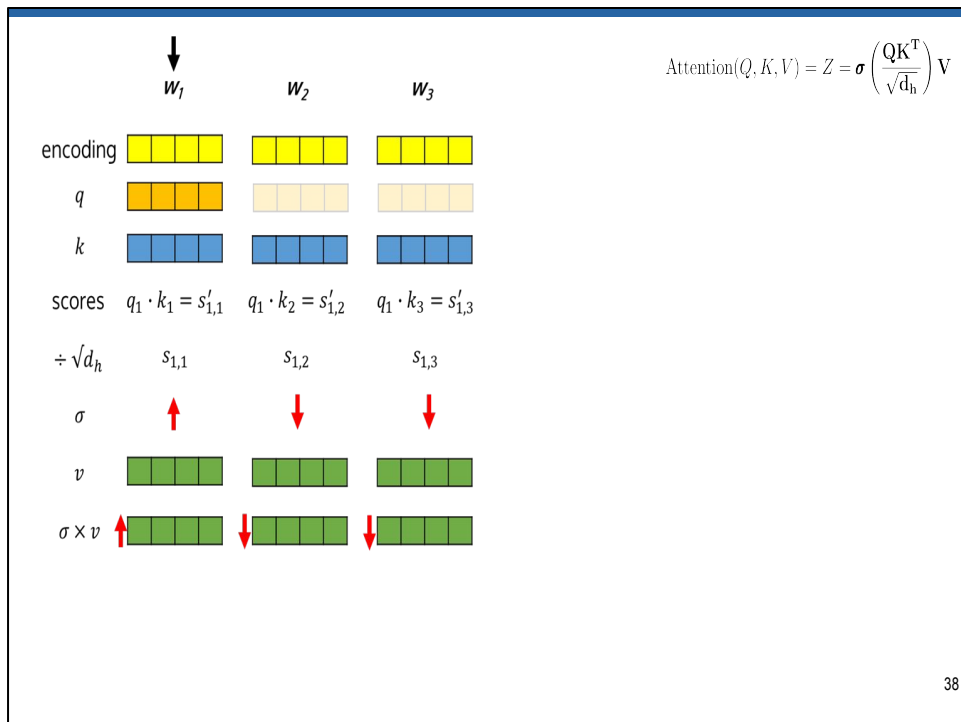
## SPEAK SLOW

- In the next step, we're going to apply a softmax which will give us our weights/normalized similarity values.
- However, before we do this, we're going to divide by the sqrt of  $d_h$ .
- There isn't actually anything special about the division value  $\sqrt{d_h}$ .
- However, before we feed our scores into the softmax, what we're doing is dividing our  $s'$  by some value  $> 1$
- ASK: What effect does this have on our softmax operation?



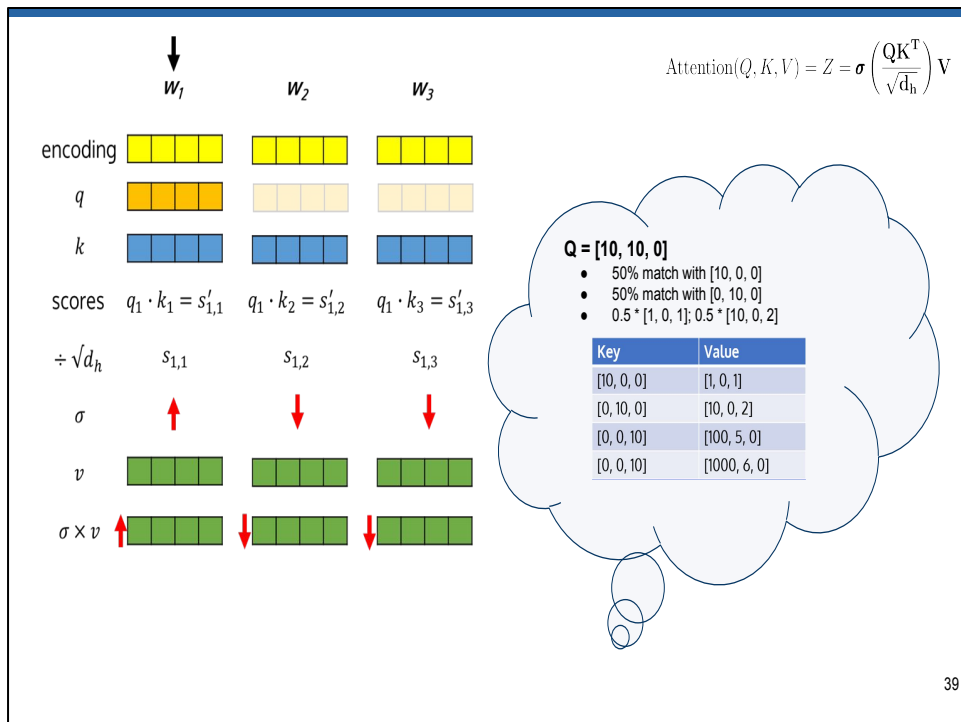
## SPEAK SLOW

- Ans: Post-division values are now closer to 0. This makes the softmax operation less peaky. That means that the outputs of the softmax operation will ideally not have an individual value which dominates the weightings.
- N.B. think back to temperature sampling from RNN lecture. They both use a division to ensure that the softmax operation isn't peaky. Here the division we use is  $\sqrt{d_h}$
- Anyway, after our softmax operation we have our normalized weighting scores
- Here, I've represented high values (values closer to 1) with an up arrow, and lower values with a down arrow



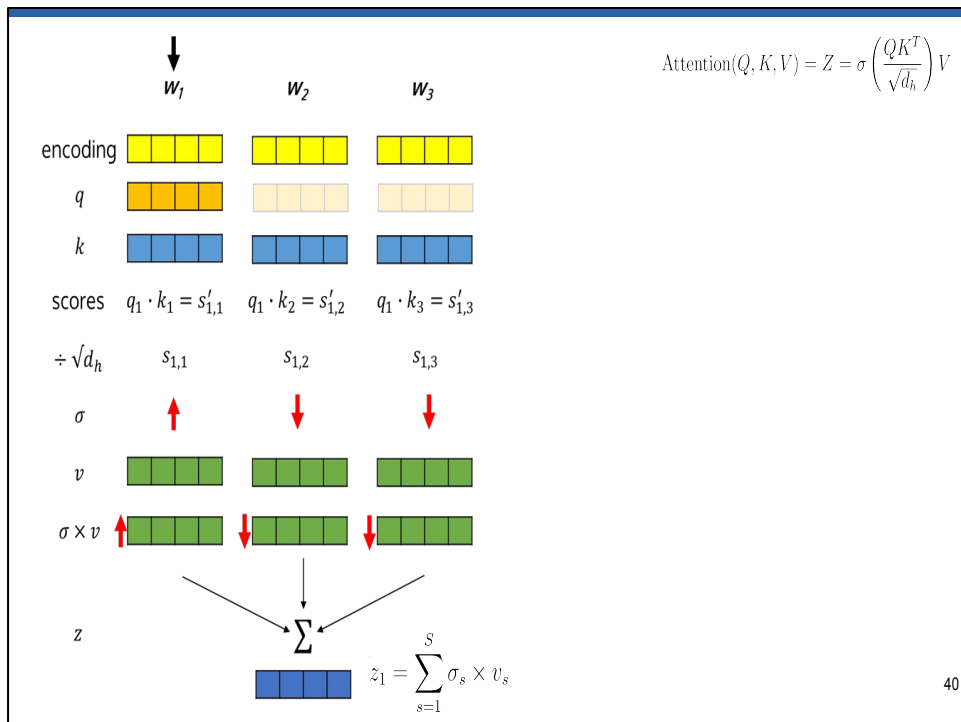
## SPEAK SLOW

- I've put 2 steps in one go here. The first step is simply retrieving the value vectors we obtained previously.
- The second step is us actually performing the weighting on each of our value vectors.
- So our value vectors retain more information if the softmax value for that index is high, and retain less information if the softmax value for that index is low



## SPEAK SLOW

- I've put 2 steps in one go here. The first step is simply retrieving the value vectors we obtained previously.
- The second step is us actually performing the weighting on each of our value vectors.
- So our value vectors retain more information if the softmax value for that index is high, and retain less information if the softmax value for that index is low

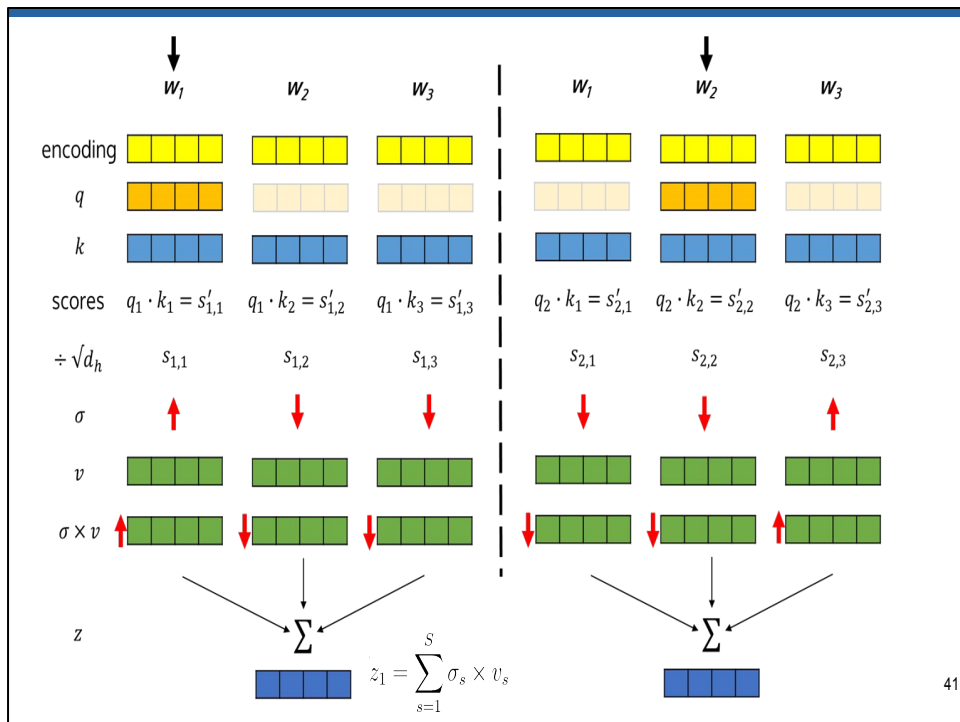


40

## SPEAK SLOW

- Finally, we obtain our contextual representation for the first word. This is the sum over our weighted value vectors. We've called it  $z$  here.





41

## SPEAK SLOW

- We can repeat this process for every word in our input sequence
- Our overall representation for our input sequence would then be all of our  $z$  vectors stacked together.
- Each  $z$ -vector is  $D$ -dimensional, and obviously we have  $S$  amounts of them

## Questions so far?

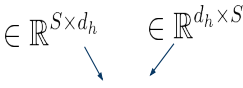
- Why do we divide by  $\sqrt{d_h}$  ?
- What is our similarity function?
- How do we normalize the similarity scores?

SPEAK SLOW

## Matrix form

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

$\in \mathbb{R}^{S \times d_h}$        $\in \mathbb{R}^{d_h \times S}$



43

### SPEAK SLOWLY

- Let's take a look at the shapes in the matrix version of the attention mechanism.
- Both Q and K are S x d<sub>h</sub>
- To multiply them together (and thus calculate the similarity scores), we need to transpose K.
  - K is thus d<sub>h</sub> x S

## Matrix form

$$\text{Attention}(Q, K, V) = Z = \sigma \left( \overbrace{\frac{QK^T}{\sqrt{d_h}}}^{\in \mathbb{R}^{S \times S}} \right) V$$

44

SPEAK SLOWLY

$$- \quad S \times d_h \times d_h \times S = S \times S$$

## Matrix form

$$\text{Attention}(Q, K, V) = Z = \sigma \left( \overbrace{\frac{QK^T}{\sqrt{d_h}}}^{\in \mathbb{R}^{S \times S}} \right) V$$

Diagram illustrating matrix multiplication:

Matrix  $Q$  (rows  $q_1, q_2, q_3$ ) is multiplied by Matrix  $K$  (columns  $k_1, k_2, k_3$ ) to produce Matrix  $Z$ .

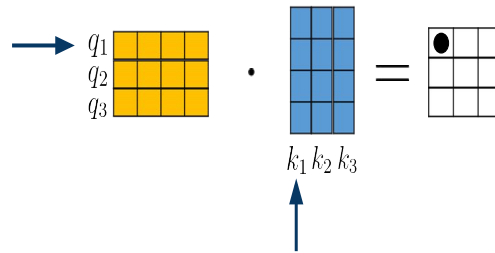
### SPEAK SLOWLY

- So that we can link the vector run-through we did previously with the matrix version, let's look at the Q K matrix multiplication that happens.
- The output of this operation gives us our unnormalized attention matrix

## Matrix form

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

$\in \mathbb{R}^{S \times S}$

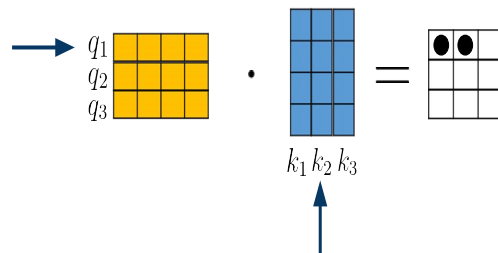


### SPEAK SLOWLY

- So that we can link the vector run-through we did previously with the matrix version, let's look at the  $Q K$  matrix multiplication that happens.
- The output of this operation gives us our unnormalized attention matrix

## Matrix form

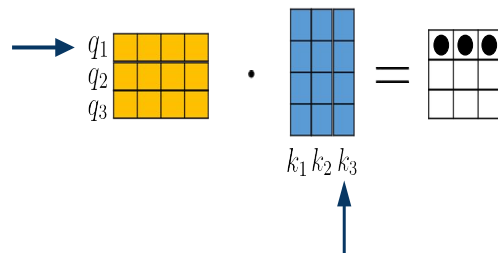
$$\text{Attention}(Q, K, V) = Z = \sigma \left( \overbrace{\frac{QK^T}{\sqrt{d_h}}}^{\in \mathbb{R}^{S \times S}} \right) V$$



SPEAK SLOWLY

## Matrix form

$$\text{Attention}(Q, K, V) = Z = \sigma \left( \overbrace{\frac{QK^T}{\sqrt{d_h}}}^{\in \mathbb{R}^{S \times S}} \right) V$$

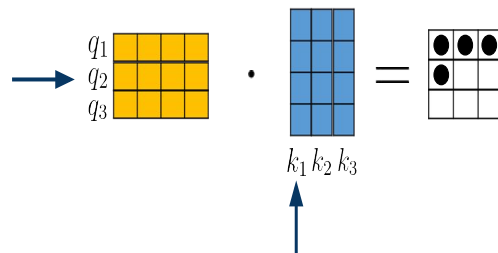


SPEAK SLOWLY



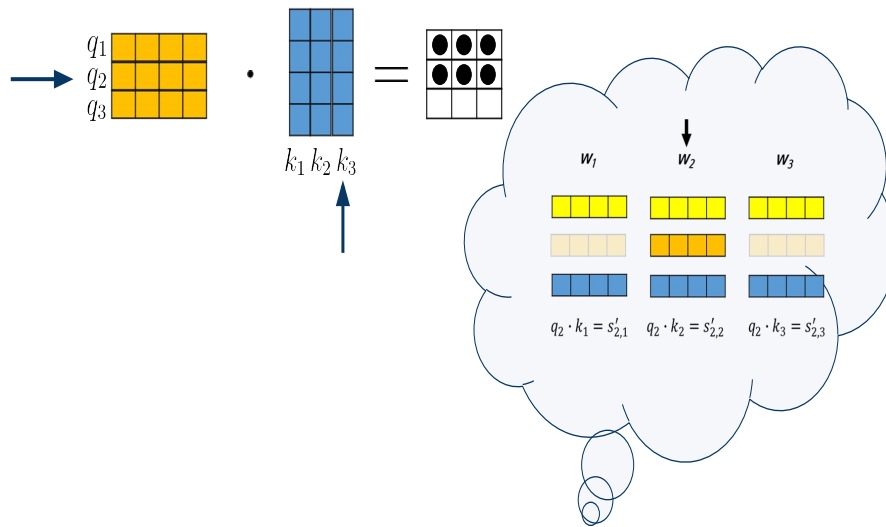
## Matrix form

$$\text{Attention}(Q, K, V) = Z = \sigma \left( \overbrace{\frac{QK^T}{\sqrt{d_h}}}^{\in \mathbb{R}^{S \times S}} \right) V$$



SPEAK SLOWLY

It's the same!




SPEAK SLOWLY

## Revisiting the formula

$$\text{Attention}(Q, K, V) = Z = \sigma \left( \underbrace{\frac{QK^T}{\sqrt{d_h}}}_{\in \mathbb{R}^{S \times S}} \right) V$$

Attention  
matrix



51

### SPEAK SLOWLY

- `sqrt(d_h)` or `softmax` doesn't change the shape of the QK operation.
- This output of this is known as our attention matrix.
  - It contains weightings for how relevant each word is to each other word

## Revisiting the formula

$$\text{Attention}(Q, K, V) = Z = \underbrace{\sigma \left( \frac{QK^T}{\sqrt{d_h}} \right)}_{\in \mathbb{R}^{S \times S}} \overset{\substack{\in \mathbb{R}^{S \times d_h} \\ \downarrow}}{V}$$

52

SPEAK SLOWLY

- V is  $S \times d_h$

## Revisiting the formula

$$\text{Attention}(Q, K, V) = Z = \underbrace{\sigma \left( \frac{QK^T}{\sqrt{d_h}} \right)}_{\in \mathbb{R}^{S \times d_h}} V$$

53

SPEAK SLOWLY

- Therefore our attention output (called Z here), is going to be S x d\_h

## Questions so far?

- What shape is our self-attention matrix?
- What shape is our attention output?

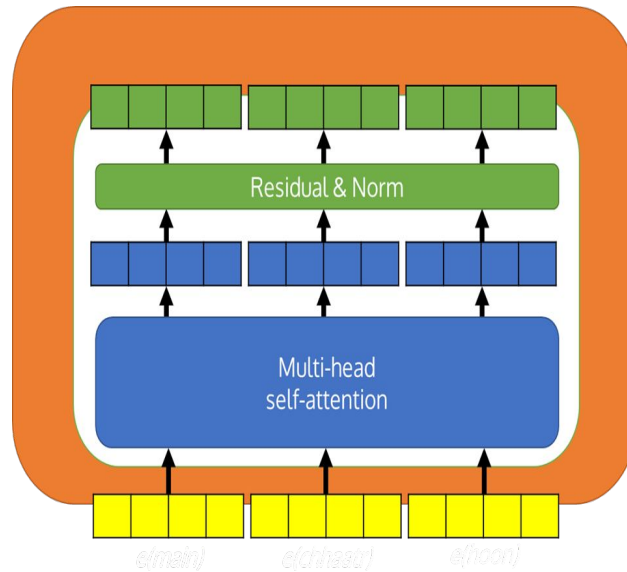
SPEAK SLOW

In code

55

SPEAK SLOW

Didn't the diagram say "multi-head"?



56

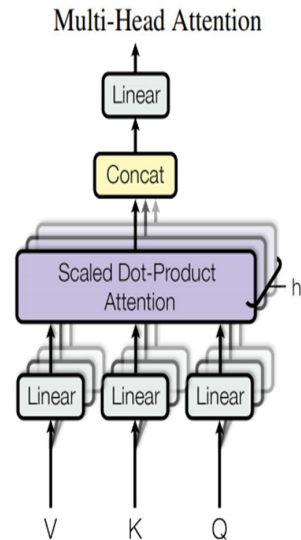
SPEAK SLOW

- We just looked at "self-attention"
- But the diagram says "multi-head" self-attention
- Let's take a look at what multi-head is.



## Multi-head attention

- Multi-head self-attention is basically just self-attention
  - But it performs self-attention head amount of times



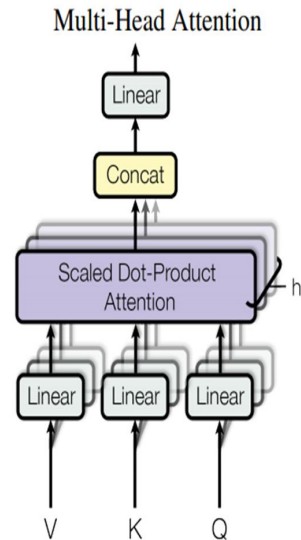
57

### SPEAK SLOWLY

- Why? Intuitively multiple attention heads allows for attending to parts of the sequence differently (e.g. some heads are responsible for longer-term dependencies, others for shorter-term dependencies)

## Multi-head attention

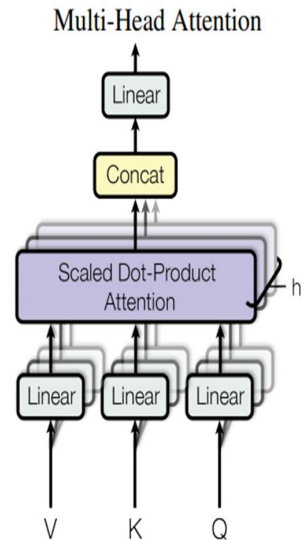
- Multi-head self-attention is basically just self-attention
  - But it performs self-attention head amount of times
- The original Transformer used 8 heads
  - This means we run self-attention 8 times over 8 *different* Q, K, Vs



SPEAK SLOWLY

## Multi-head attention

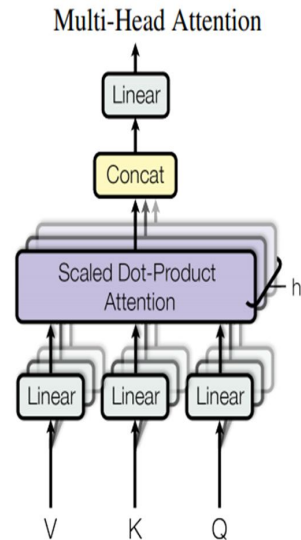
- Multi-head self-attention is basically just self-attention
  - But it performs self-attention head amount of times
- The original Transformer used 8 heads
  - This means we run self-attention 8 times over 8 *different* Q, K, Vs
- Our final representation concatenates and projects the self-attention outputs



SPEAK SLOWLY

## Multi-head attention

- Multi-head self-attention is basically just self-attention
  - But it performs self-attention head amount of times
- The original Transformer used 8 heads
  - This means we run self-attention 8 times over 8 *different* Q, K, Vs
- Our final representation concatenates and projects the self-attention outputs
- Each head has dimensionality  $D/\text{heads}$



60

### SPEAK SLOWLY

- The  $D/\text{heads}$  is the  $d_h$  as we saw previously

## Questions so far?

- What is a reason for us using multi-head self attention instead of single head?
- Let  $D = 64$ , and  $\text{\#heads} = 4$ . What is the dimension of each head?

61

SPEAK SLOW

In code

62

SPEAK SLOW

## Normalization

- Data fed to a neural network should be normalized:  $\hat{x} = \frac{x - \mu}{\sigma}$
- There are also normalisation techniques within the activations or layers of a network:
  - Layer normalization, batch normalization, group normalization etc.

63

### SPEAK SLOWLY

- Ask audience if anyone has come across any of the mentioned normalization techniques before
-

## Layer Normalization

$$\hat{x} = \frac{x - \mu}{\sigma}$$

- Layer normalization normalizes the features of **one sample in one batch**

- $B = \{x_1, x_2, \dots, x_N\} \quad x_n \in \mathbb{R}^d$

SPEAK SLOWLY

-



## Layer Normalization

$$\hat{x} = \frac{x - \mu}{\sigma}$$

- Layer normalization normalizes the features of **one sample in one batch**

- $B = \{x_1, x_2, \dots, x_N\} \quad x_n \in \mathbb{R}^d$

- Two key steps:

- 1.  $\hat{x}_n = \text{normalize}(x_n)$
  - 2. Transform  $\hat{x}_n$  with learned parameters  $\gamma, \beta$

SPEAK SLOWLY

-

## Layer Normalization

$$\hat{x} = \frac{x - \mu}{\sigma}$$

- Layer normalization normalizes the features of **one sample** in **one batch**

- $B = \{x_1, x_2, \dots, x_N\} \quad x_n \in \mathbb{R}^d$

- Two key steps:

- 1.  $\hat{x}_n = \text{normalize}(x_n)$
  - 2. Transform  $\hat{x}_n$  with learned parameters  $\gamma, \beta$

$$LN(\hat{x}_n) = \gamma \hat{x}_n + \beta$$

$$\gamma \in \mathbb{R}^d$$

$$\beta \in \mathbb{R}^d$$

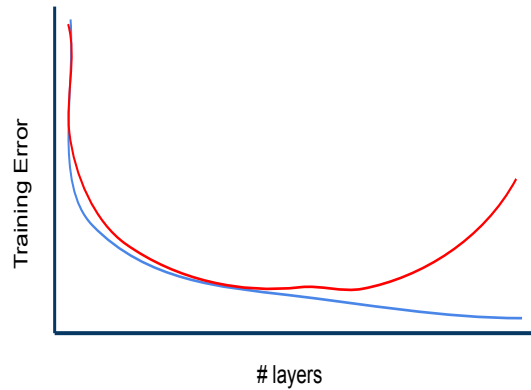
66

### SPEAK SLOWLY

- The purpose of gamma and beta is to allow the neural network to learn the optimal scale and shift for each feature after normalization.
- By applying these learned parameters to the normalized feature vectors, the neural network can adjust the range and mean of each feature to better fit the task at hand.

# Residual Connections

- **Theory:** If we stack layers, we should get a lower training error
- **Practical:** lol no



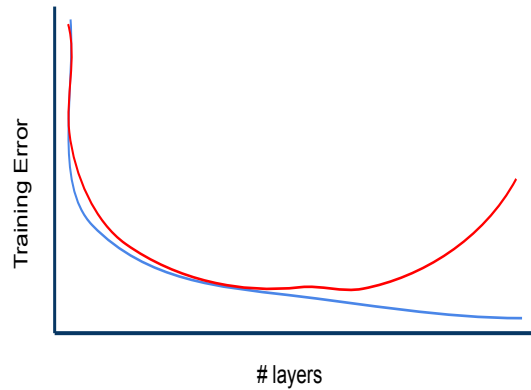
67

SPEAK SLOWLY

-

# Residual Connections

- **Theory**: If we stack layers, we should get a lower training error
- **Practical**: lol no
- **Residual connections**: hold my activations



68

SPEAK SLOWLY

-

## Residual Connections

- Residual Connections help mitigate the vanishing gradient problem
  - Vanishing gradients = tiny weight changes

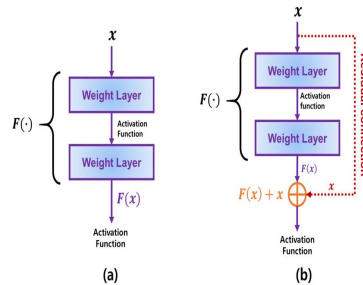
69

SPEAK SLOWLY

-

# Residual Connections

- Residual Connections help mitigate the vanishing gradient problem
  - Vanishing gradients = tiny weight changes
- Residual connections provide a shortcut for information to flow to later layers of a network
  - The output of an earlier layer is added directly to the output of a later layer



70

## SPEAK SLOWLY

- By adding the previous layer's output directly to the current layer's output, a residual connection allows the current layer to focus on learning the difference between the two outputs, rather than learning an entirely new transformation.

## Questions so far?

- We ran through layer norm with an input tensor of (batch\_size, n\_features). Describe what the differences are if we had an input tensor of (batch\_size, sequence\_length, n\_features)?
- Why are gamma and beta  $\in \mathbb{R}^d$  instead of a scalar?

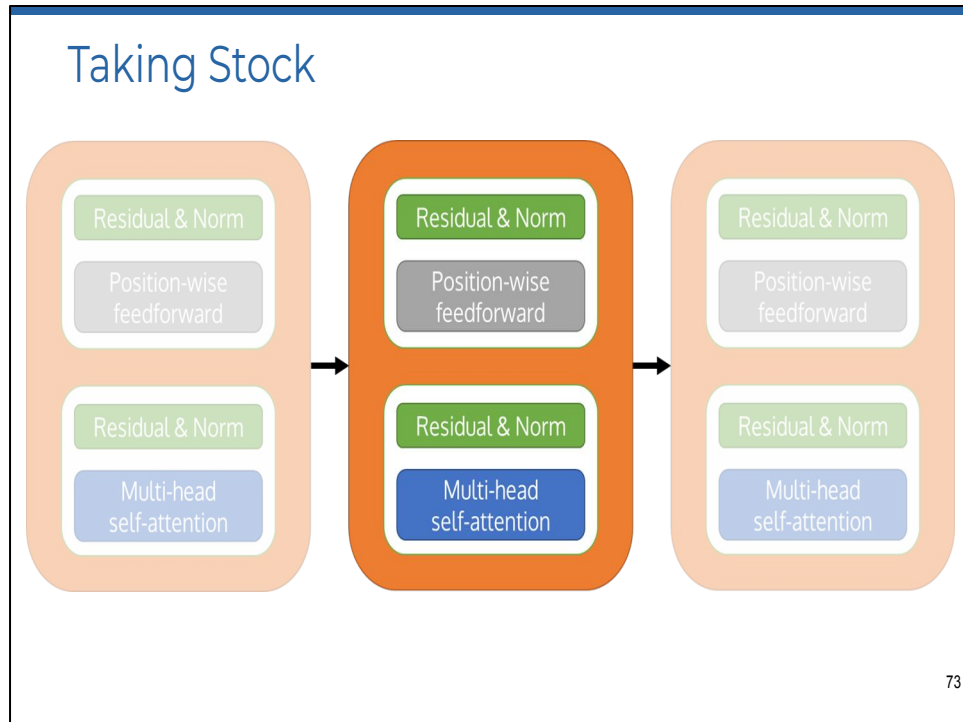
SPEAK SLOW

In code

72

SPEAK SLOW





### SPEAK SLOWLY

- We've finished the first sublayer and covered most of the hard stuff (i.e. attention)
- We've almost finished covering the encoder. Just the position-wise feedforward layer and positional encodings left.

## Position-wise Feedforward Network

- A position-wise feedforward network is an MLP
- *Position-wise* just means it is applying the same transformation to every element in the sequence
  - i.e. same weights applied to all tokens in the sequence

74

SPEAK SLOWLY

-

## Position-wise Feedforward Network

- A position-wise feedforward network is an MLP
- *Position-wise* just means it is applying the same transformation to every element in the sequence
  - i.e. same weights applied to all tokens in the sequence

$$\text{FNN}(x) = \max(0, xW_1 + b)W_2 + b_2$$
$$W_1 \in \mathbb{R}^{D \times d_{ff}} \quad W_2 \in \mathbb{R}^{d_{ff} \times D} \quad d_{ff} = 2048$$

75

SPEAK SLOWLY

- Q: How many layers in this network?
- Q: What is  $\max(0, X)$ ?

## Position-wise Feedforward Network

- A position-wise feedforward network is an MLP
- *Position-wise* just means it is applying the same transformation to every element in the sequence
  - i.e. same weights applied to all tokens in the sequence

$$\text{FNN}(x) = \max(0, xW_1 + b)W_2 + b_2$$
$$W_1 \in \mathbb{R}^{D \times d_{ff}} \quad W_2 \in \mathbb{R}^{d_{ff} \times D} \quad d_{ff} = 2048$$

- Two layered network, with a ReLU activation function

76

SPEAK SLOWLY

- Q: How many layers in this network?
- Q: What is  $\max(0, X)$ ?

Questions so far?

77

SPEAK SLOW

In code

78

SPEAK SLOW

## Encoder Layer

- That's everything we need to code up an encoder layer
- Then we just have positional encoding and the decoder to deal with
  - We're about 70% done

SPEAK SLOWLY

In code

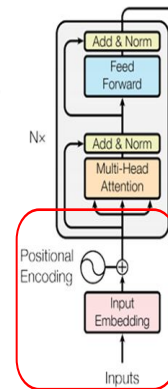
80

SPEAK SLOW



# Positional Encodings

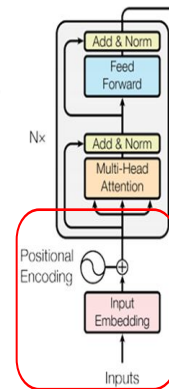
- Our diagram indicates that the output of one layer forms the input to the next
  - But what about the first layer? What is this thing called positional encoding?



SPEAK SLOWLY

# Positional Encodings

- Our diagram indicates that the output of one layer forms the input to the next
  - But what about the first layer? What is this thing called positional encoding?
- **Transformers are position invariant by default**
  - The cat sat = sat the cat
- Differs from RNNs, which are inherently sequential
- Positional encodings are a way to inject position information into the embeddings



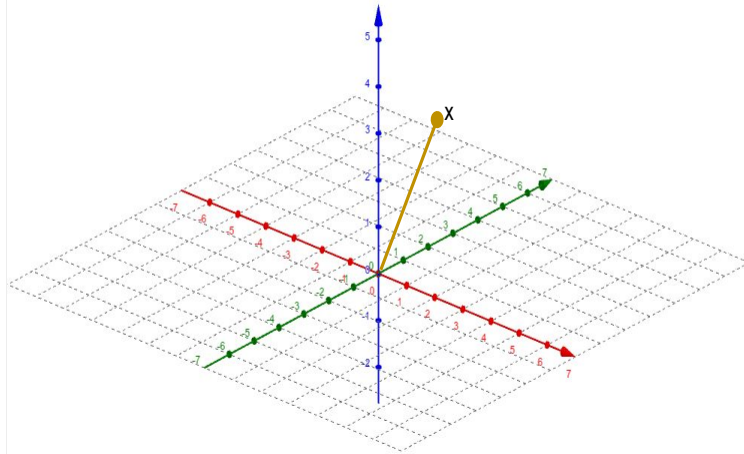
82

## SPEAK SLOWLY

- Position invariant. What this means is they have no inherent understanding of sequence order.
  - By default, [The cat sat] = [sat the cat]

# Positional Encodings

$\text{input\_to\_first\_layer} = \text{embedding}(x) + \text{get\_positional\_encoding\_vector}(s)$



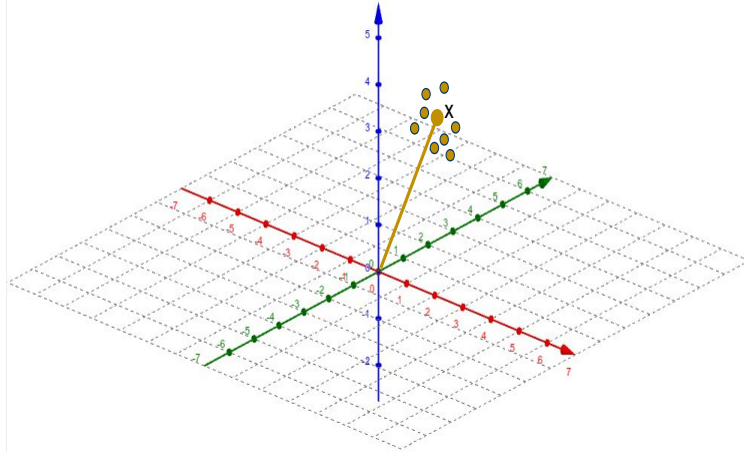
83

SPEAK SLOWLY

- The slide shows an example embedding for an input word x

# Positional Encodings

$\text{input\_to\_first\_layer} = \text{embedding}(x) + \text{get\_positional\_encoding\_vector}(s)$



84

## SPEAK SLOWLY

- The positional encoding vector is a small vector which is added to the embedding of  $x$ .
- The resulting vector will be in the same general space as the original embedding
- The function which gets the positional encoding is purely a function of the timestep of  $x$ .
  - Again, it only depends on which index  $x$  appeared in the input sequence (e.g. index 1, 2, 3;  $s$  generally).
  - More of this in a second
- Visually, the other green dots may represent the vector point once the positional encoding vectors have been added to the embedding vector

## Positional Encodings

- Multiple ways of incorporating positional embeddings
  - Authors propose using sinusoids
  - You can also learn the positional embeddings (more in BERT)

85

SPEAK SLOWLY

-

## Positional Encodings

- Multiple ways of incorporating positional embeddings
  - Authors propose using sinusoids
  - You can also learn the positional embeddings (more in BERT)
- Applying positional encodings depends ONLY on the index the word appears in

86

### SPEAK SLOWLY

- Positional Encoding does not depend on the features of any given word. Only the position that it appears in. e.g. index 1, index 2, index 3 etc

## Positional Encodings

$$PE_{pos,2i} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

$$PE_{pos,2i+1} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

Position of  
the word      Index of d

87

### SPEAK SLOWLY

- PE is a function of 2 inputs: position of a word, and the model dimensionality
- Notice what is happening with these arguments. The  $2i$  and  $2i+1$  imply that we're going to be looping over the indexes in range of  $d$ .
  - E.g. Consider we had a vector of 512 dimensions. We would be looping over it. At an even index (e.g. 0), we'd apply the sin variant of PE. At an odd index (e.g. 1), we would apply the cosine variant.
- Now observe that the only difference between the functions is whether we use a sin or a cosine
- Apart from that, the functions divide the position argument by  $10000^{(2i/d)}$ 
  - Note that when we're early on in looping over  $d$ ,  $i$  will be small. The resulting exponent would be therefore be small. This makes the denominator small. Thus the overall value of  $pos/10000^{(2i/d)}$  would be larger than when we're later in our loop over  $d$
  - You can play around with the implications of how this affects the output sin and cosine in your own time (just thought it was worth mentioning)

## Positional Encodings (pseudo-code)

```
PE_matrix = zeros(maxT, d)
for pos in maxT:
    for i in range(d):
        if i % 2 == 0:
            PE_matrix[pos, i] = sin(pos/10000^(i/d))
        if i % 2 == 1:
            PE_matrix[pos, i] = cos(pos/10000^(i/d))
```

88

### SPEAK SLOWLY

- In implementation, we have a PE matrix in [maxT, d]
- maxT is the maximum sequence length we ever want to support (e.g. 5000)
- d is our 'model dimensionality'
- 
-



## Positional Encodings (pseudo-code)

```
PE_matrix = zeros(maxT, d)
for pos in maxT:
    for i in range(d):
        if i % 2 == 0:
            PE_matrix[pos, i] = sin(pos/10000^(i/d))
        if i % 2 == 1:
            PE_matrix[pos, i] = cos(pos/10000^(i/d))
```

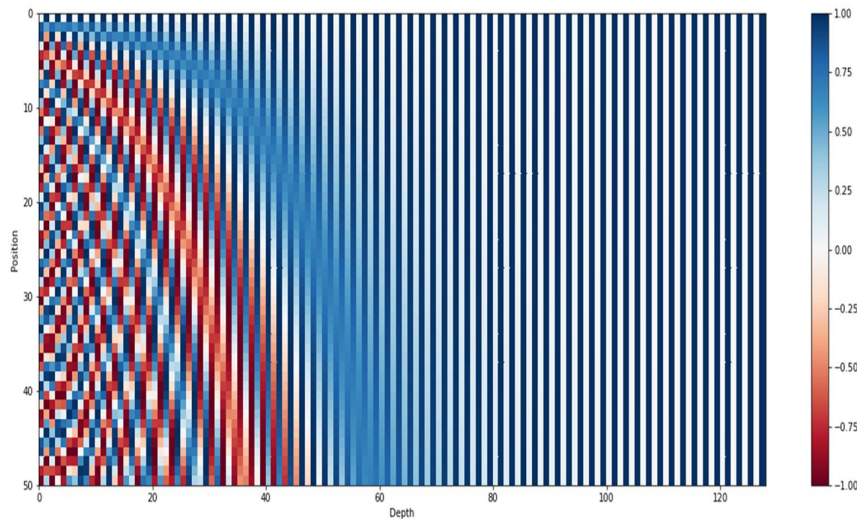
- Then, given a word embedding and its position in the sequence (e.g. given: pos = 2, embedding = [1.0, 1.1, 1.2, 1.3]), add PE\_matrix[pos] to the embedding

89

### SPEAK SLOWLY

- In implementation, we have a PE matrix in [maxT, d]
- maxT is the maximum sequence length we ever want to support (e.g. 5000)
- d is our 'model dimensionality'
- 
-

## Positional Encodings (pseudo-code)



90

### SPEAK SLOWLY

- Consider position 0 (i.e. the first word in the sequence)
  - Positional encoding function means no manipulation is done to the vector
- Consider position 1
  - Positional encoding function shows that indexes up to 10 have values close to 1 added to them
- Consider position 22
  - Positional encoding function shows that the first few indexes are affected heavily by positional encodings. Some indexes have values close to -1 added to them, followed shortly by a value +1 added to them

## Questions so far?

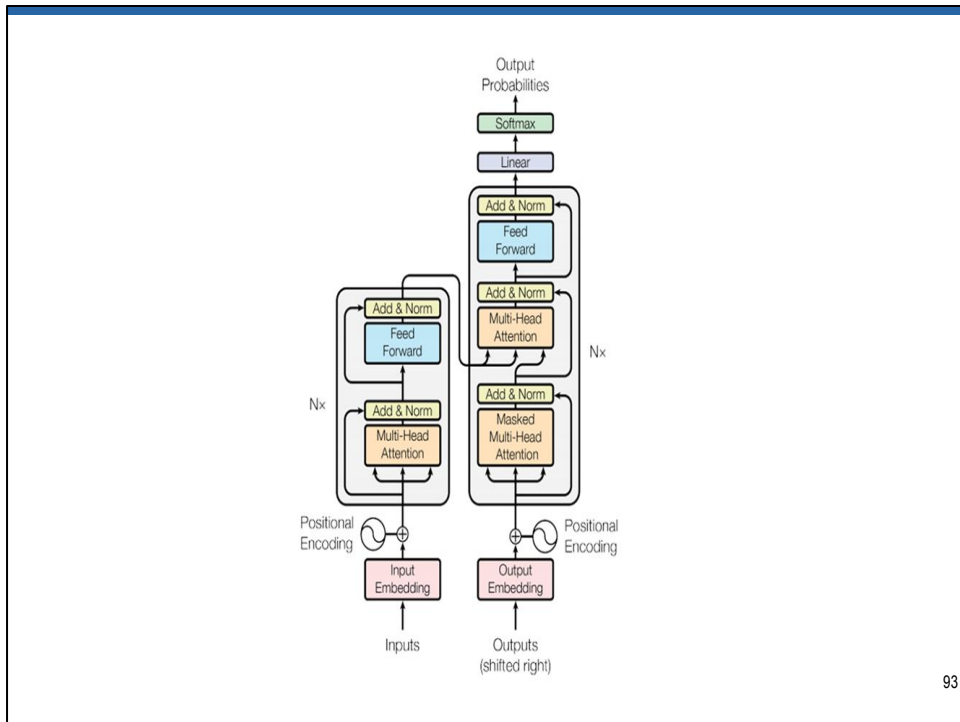
- Why do we need positional encodings?
- Intuitively, how does a positional encoding vector affect our word embedding?

SPEAK SLOW

In code

92

SPEAK SLOW



93

## SPEAK SLOWLY

- Ask audience:
  - 1. Has anyone heard of a Transformer
  - 2. Has anyone read up about how it works?
  - 3. ???

## Encoder

- We have everything we need to code up the encoder
- Embedding + PE layer, followed by a for loop over the encoder layers

SPEAK SLOWLY

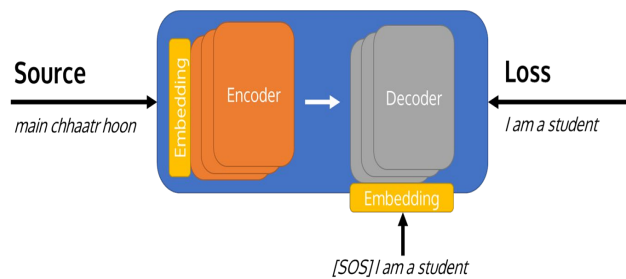
In code

95

SPEAK SLOW

## Decoder

- Encoder is DONE
- Only a few differences between encoder and decoder
  - Differences in train vs test time
    - Masked multi-head self-attention
  - Cross attention



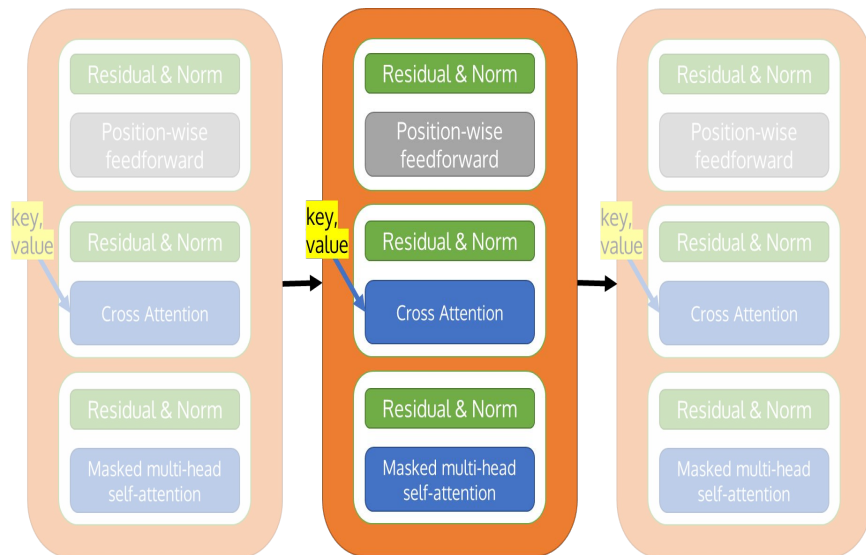
96

### SPEAK SLOWLY

- Ask if there are any questions about the encoder
-



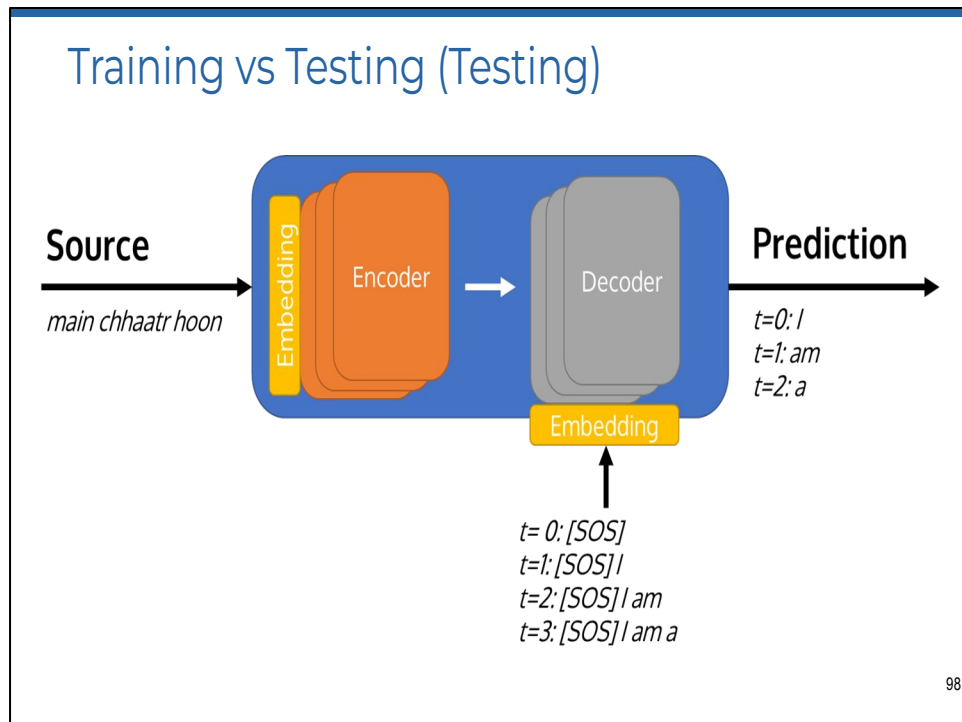
## Decoder



97

### SPEAK SLOWLY

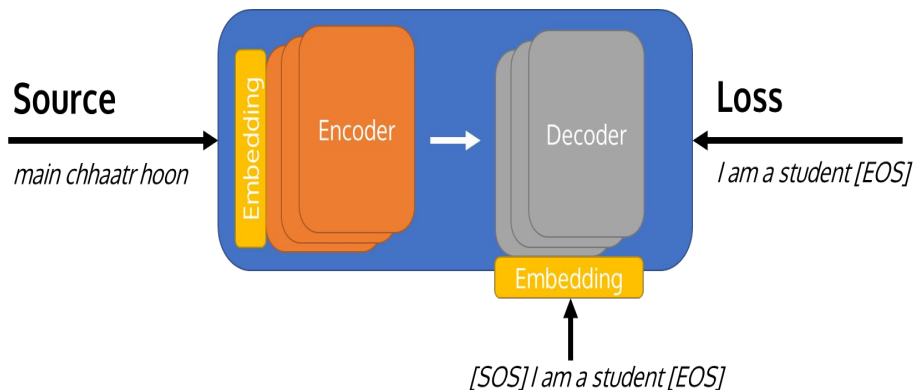
- Two of the differences I mentioned in the previous slide are present here in this diagram:
  - Masked multi-head self-attention
  - Cross attention. Note that cross attention is also receiving a key and value tensor. These key and value tensors come from the encoder. We'll look at this more in a future slide



### SPEAK SLOWLY

- I'll start by covering how Transformers work for inference, since it's a bit more intuitive and something that you guys should be familiar with
- Namely, we perform auto-regressive generation
- 1. Our source sentence gets encoded via our encoder
- 2. We feed an SOS token to our decoder. It then predicts the first word (i.e. I)
- 3. We append the prediction to our SOS token, and then use this to predict the next word (i.e. am)
- 4. And so forth. At some point our decoder will predict an EOS token (not shown here), and we'll know that this is the end of the generation.

## Training vs Testing (Training)



99

### SPEAK SLOWLY

- However, during training, we won't get the model to generate auto-regressively.
- We're actually going to feed the whole target sequence as input to the decoder and decode the sentence in one go
  - This means we can run the decoder stack once.
  - As opposed to running it for as many tokens as we have in the target sequence (which is what we do when performing inference)
- If we feed in the whole sequence to the decoder, we need a way to tell the decoder not to look at future tokens when computing attention for one of the tokens.
  - E.g. when trying to compute the attention-based representation for "am", we should not be allowed to look at "a" or "student", since these tokens are in the future
  - We enforce this unidirectionality with masking

## Masked Multi-head Self-attention

- We've looked at attention in the encoder
  - Bidirectional: each word looked at every other word

100

### SPEAK SLOWLY

- Attention allowed us to get a representation for every word in the input sequence. We got this representation by each word looking at every other word in the input sequence
- Because each word can look at every other word, including those in future positions, it is considered bidirectional

## Masked Multi-head Self-attention

- We've looked at attention in the encoder
  - Bidirectional: each token looked at every other token
- There's a difference when decoding... we shouldn't have access to future tokens
- Masked MHA is a strategy to tell the model, during training, not to look at future tokens

101

### SPEAK SLOWLY

- ...this is what we covered in the previous slide
- Note that we're not using a predicted value as an input to the transformer during training. An arbitrary token in the decoder input *is* the ground truth  $t$ . In this sense, the transformer uses 100% teacher forcing

## Masked Multi-head Self-attention

- Our mask is going to be  $\in \mathbb{R}^{T \times T}$ 
  - i.e. a square matrix
  - Here,  $T$  is the target sequence length
- For every token, an illegal location will be all future tokens we're not meant to have access to.
  - So at  $t=1$ , we don't have access to any words  $t>1$ ,
  - At  $t=2$ , we don't have access to any words  $t>2$

102

### SPEAK SLOWLY

- We're going to enforce unidirectionality in the self-attention mechanism by using a mask.
- Here, unidirectionality means that we should only be looking at words to the left.
  - Words to the right are future tokens which would be illegal to access

## Masked Multi-head Self-attention

- Our mask is going to be  $\in \mathbb{R}^{T \times T}$ 
  - i.e. a square matrix
  - Here,  $T$  is the target sequence length
- For every token, an illegal location will be all future tokens we're not meant to have access to.
  - So at  $t=1$ , we don't have access to any words  $t>1$ ,
  - At  $t=2$ , we don't have access to any words  $t>2$

|         | I | am | a | student |
|---------|---|----|---|---------|
| I       | ✓ | ✗  | ✗ | ✗       |
| am      | ✓ | ✓  | ✗ | ✗       |
| a       | ✓ | ✓  | ✓ | ✗       |
| student | ✓ | ✓  | ✓ | ✓       |

103

### SPEAK SLOWLY

- We're going to enforce unidirectionality in the self-attention mechanism by using a mask.
- Here, unidirectionality means that we should only be looking at words to the left.
  - Words to the right are future tokens which would be illegal to access
- The mask will be a  $T \times T$  matrix
- We will set values in the upper triangular to be  $-\infty$  (indicating that future tokens should not be accessed)

## Questions so far?

- What are the differences between training and testing in a Transformer?
- How does masked self-attention work?
- What teacher forcing ratio do we use in Transformers?

104

SPEAK SLOW



In code

105

SPEAK SLOW

## Cross Attention

- The decoder needs to know the encoded representation
- Every time we're decoding a token, we need to know which encoded words we should look at to decode the token
  - Achieved via cross attention

106

### SPEAK SLOWLY

- Achieved via cross attention: Think back to attention in RNNs - the decoder looked at all the encoded words to find out which source words were most important to the current step of decoding
  - Cross attention is doing a similar thing. As we decode a token, we look at all encoded tokens to find out which are more important for the current decoding step

## Cross Attention

- The decoder needs to know the encoded representation
- Every time we're decoding a token, we need to know which encoded words we should look at to decode the token
  - Achieved via cross attention
- We use the **key, value** tensors from the **last encoder layer**, and send them to all the decoder layers.
  - So, **query** comes from the **current decoder layer**

107

### SPEAK SLOWLY

- Think back to our dictionary intuition about the attention mechanism.
- We have a query vector (in target-language embedding space). We're then looking to find the similar key vectors from the encoder. In other words, which encoded words are most similar to this current decoding word
- Then, using the softmax weighted similarities, we obtain some aggregated value indicating which words are most important

## Cross Attention

- The decoder needs to know the encoded representation
- Every time we're decoding a token, we need to know which encoded words we should look at to decode the token
  - Achieved via cross attention
- We use the **key, value** tensors from the **last encoder layer**, and send them to all the decoder layers.
  - So, **query** comes from the **current decoder layer**
- Cross attention matrix shape =  $T \times S$

108

### SPEAK SLOWLY

- Think back to our dictionary intuition about the attention mechanism.
- We have a query vector (in target-language embedding space). We're then looking to find the similar key vectors from the encoder. In other words, which encoded words are most similar to this current decoding word
- Then, using the softmax weighted similarities, we obtain some aggregated value indicating which words are most important

## Decoder

- Our overall decoder is similar to our encoder layer in construction
- We create a decoder and decoder layer class.
- The decoder layer class contains one decoder layer
- The decoder class contains Embeddings + PE, and a for loop over the desired number of decoder layers

SPEAK SLOWLY

-

## Questions so far?

- What is cross attention doing?
- What would the shape of the cross attention matrix be?

SPEAK SLOW

In code

111

SPEAK SLOW

## Transformers

- Now we just plug the encoder and decoder together.
- Our transformer class does 4 things every training loop:
  - Create a source and target mask
  - Run the encoder
  - Run the decoder
  - Output logits for token prediction

112

SPEAK SLOWLY

-

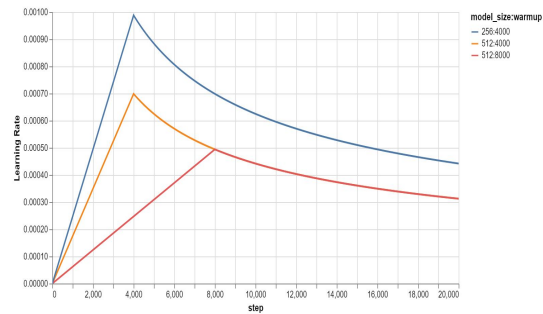


## Other tricks

- Authors use weight tying in the decoder
  - Embedding matrix and output projection matrix are shared
- Authors use a decaying learning rate

$$lr = \sqrt{\frac{1}{d}} \times \min \left( \sqrt{\frac{1}{d}}, i \times \text{warmup}^{-1.5} \right)$$

$i$  = current global step  
 $\text{warmup}$  = hyperparameter (default: 4000)  
 $d$  = model dimensionality



113

SPEAK SLOWLY

-

Questions so far?

114

SPEAK SLOW

In code

115

SPEAK SLOW