Self-Attention Mechanism in Transformer Networks

This is the first part of the Transformer Series. Here, I present an intuitive understanding of the self-attention mechanism in transformer networks.

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

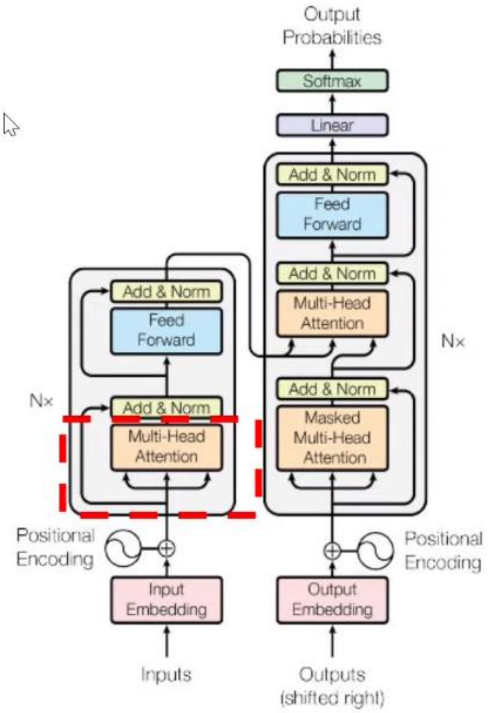
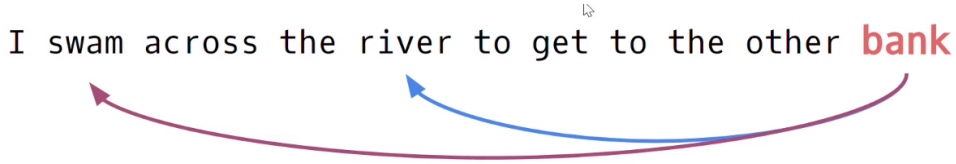


Figure 1: The Transformer - model architecture.

Attention is All You Need [Vaswani et al.]

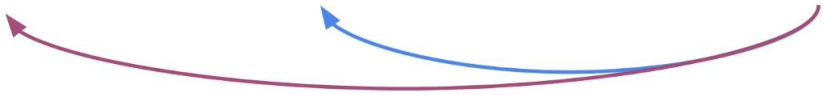


~~Bank == financial institution~~

Bank == sloping raised land

To understand this sentence, we need to identify the surrounding words around the word **bank**, the words that augment its meaning

I drove across the road to get to the other **bank**



Bank == financial institution
~~Bank == sloping raised land~~

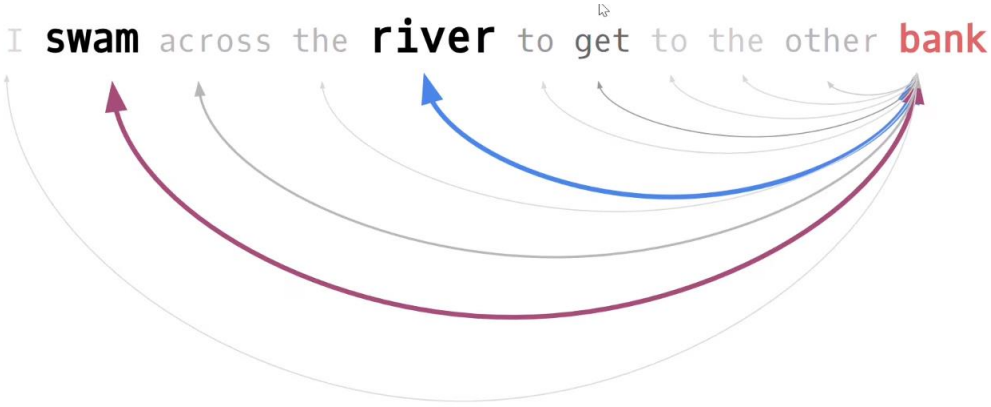
If you change the sentence a little then the meaning changes depending on the context in other to define a words meaning in a sentence

I swam across the river to get to the other **bank**

Context matters!

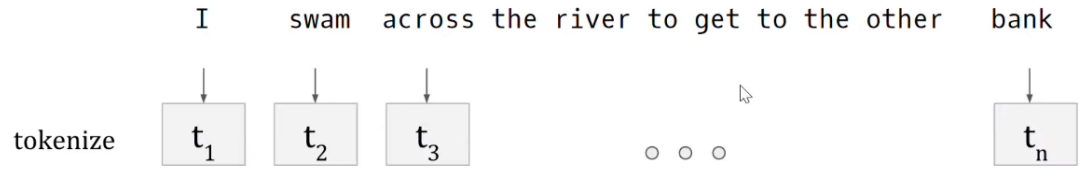
What would you think of **I swam across the river to get to the other bank** when the sentence is shown along with this picture?

Can we build a **mechanism** that **weighs** neighboring words to enhance the meaning of the word of interest?

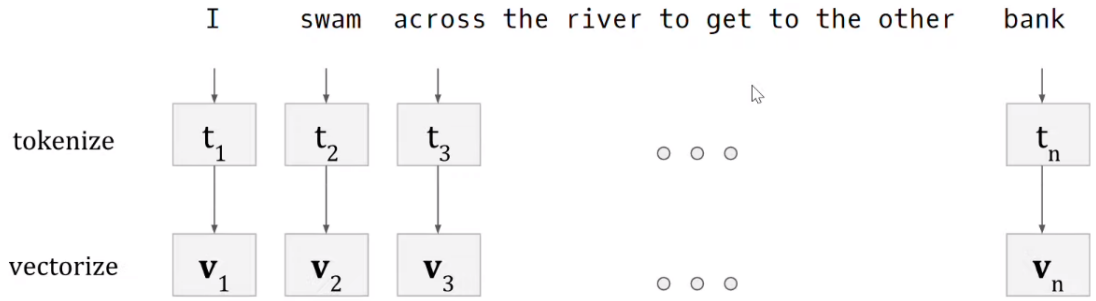


We want to augment the meaning of the word **bank** with two words **swam** and **river**, the other words might not be that important.

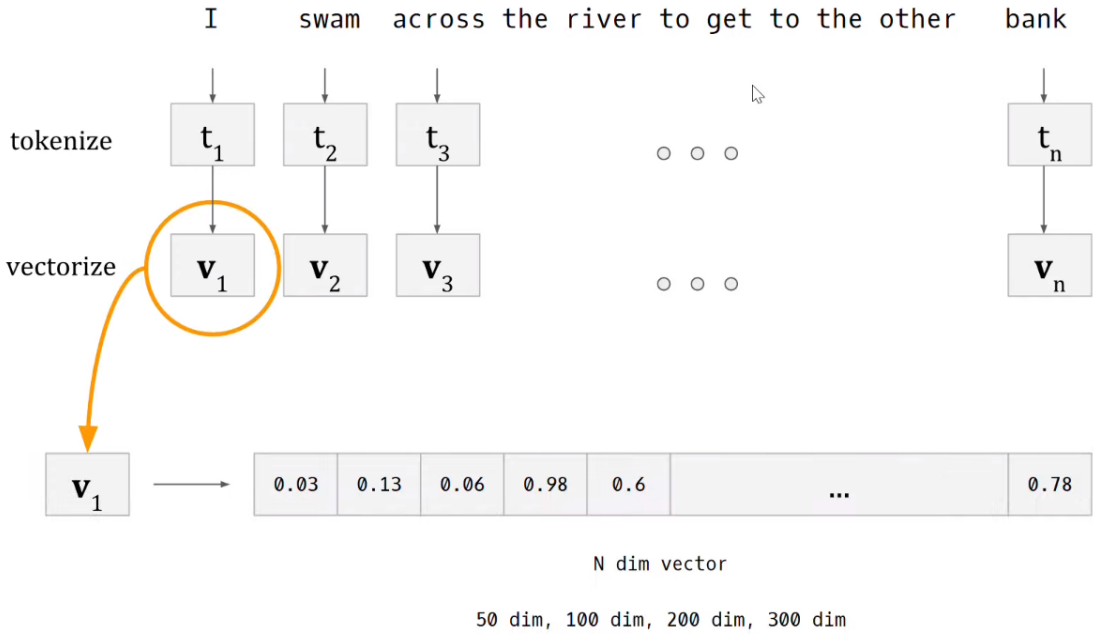
The main purpose of self-attention mechanism is to add **contextual information** to words in the sentence.



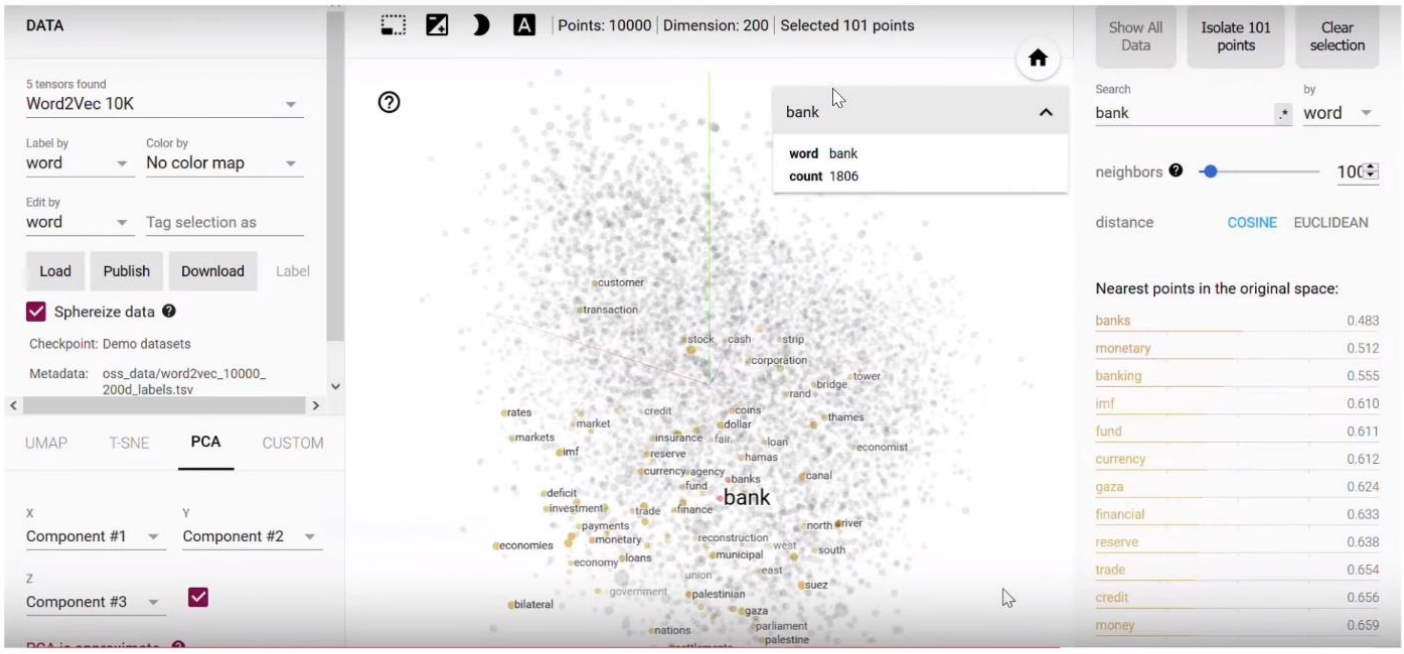
Our first step is to tokenize the sentence into different words



An NN does not understand English so we have to convert the words into word vectors of N dimensions as below

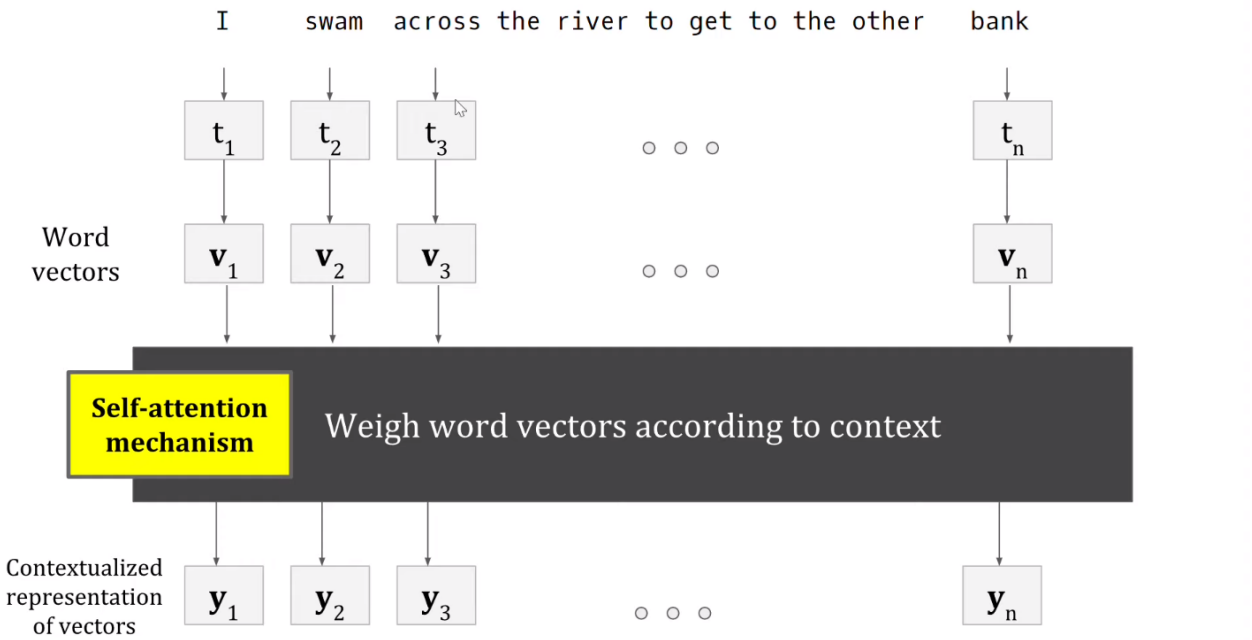


Words that have similar meanings will tend to cluster together as tensors

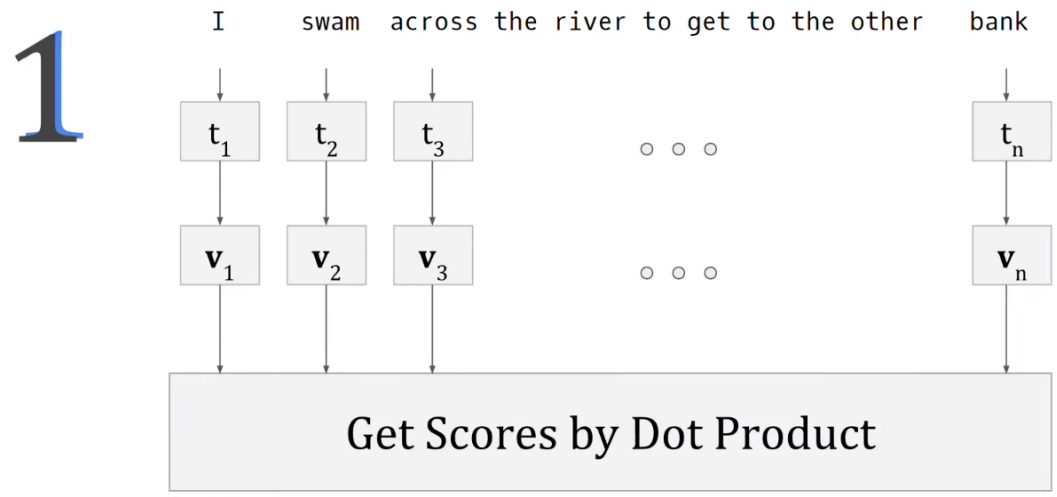


https://projector.tensorflow.org/

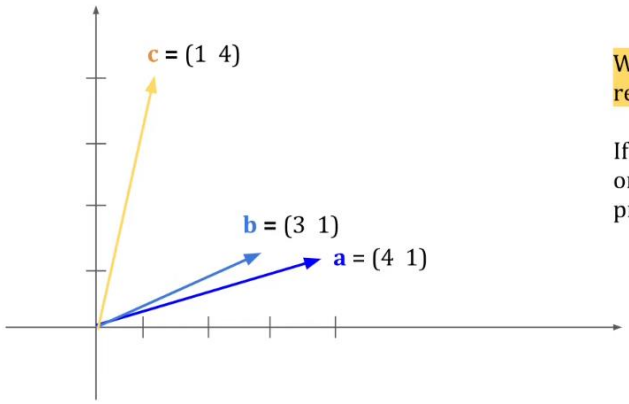
@ark_aung



The inputs to the self-attention mechanism are the word vectors, it will weight them to understand the contextualized representations



Vector dot product

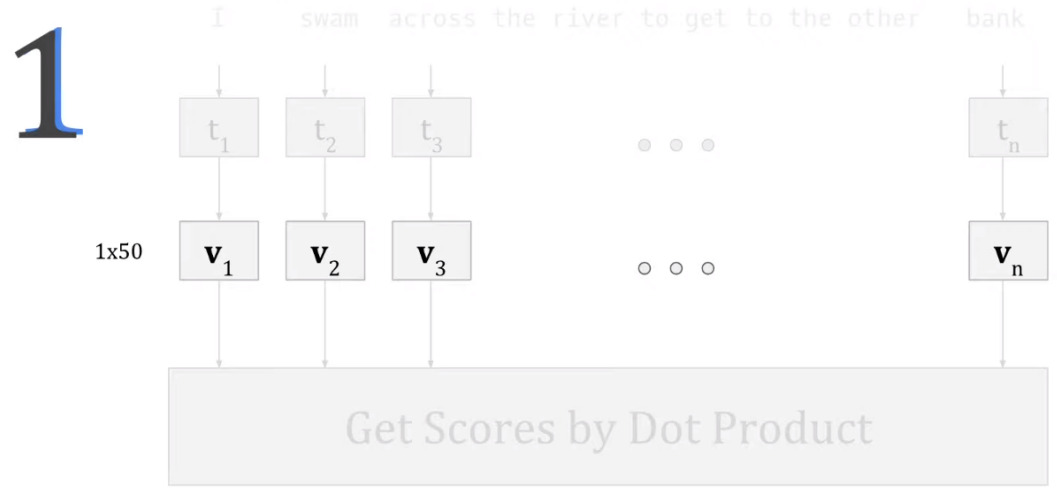


What I want you to remember:

If two vectors are close to one another, their dot product is going to be big

a . **b** = 13 **a** . **c** = 8

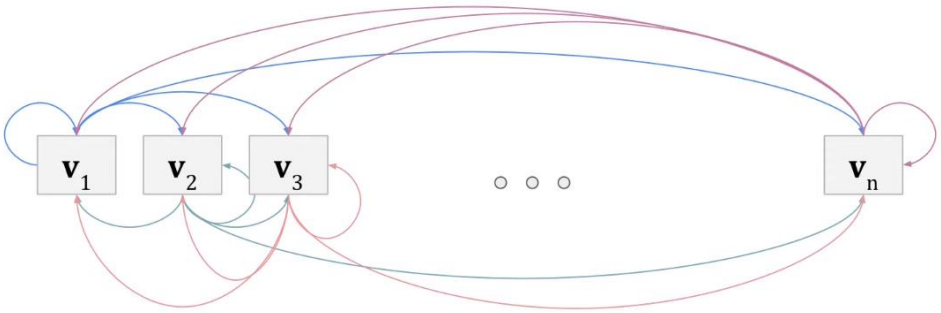
Vectors **a** and **b** are closer and have a larger dot product than vectors **a** and **c**. *Dot product* of perpendicular vectors is **0**.



Assume our word vectors are dimension 1 by 50 column, we can do the vector dot products as below

$$s_{11} = v_1 v_1^T$$

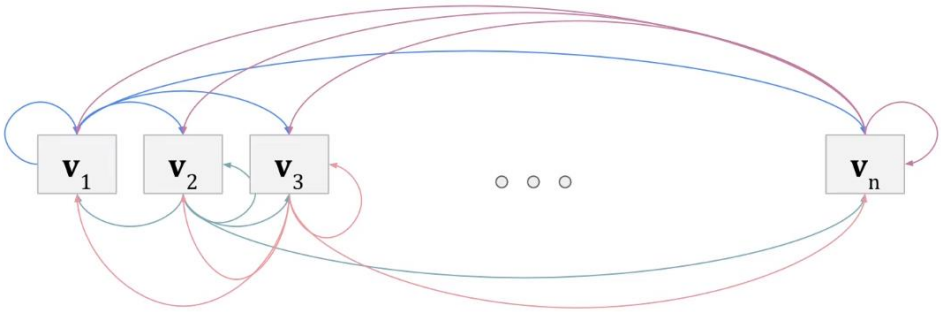
1x50 50x1

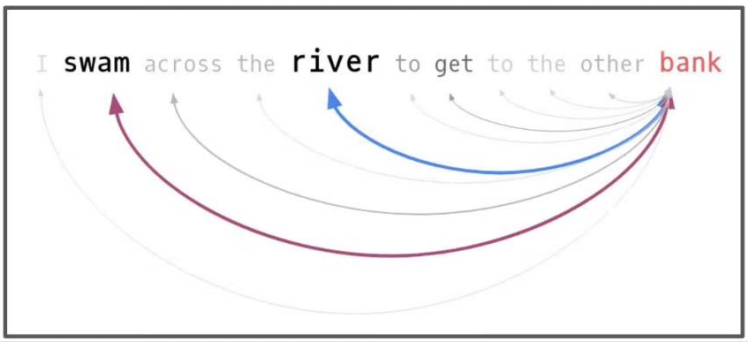


$$s_{11} = v_1 v_1^T$$
$$s_{12} = v_1 v_2^T$$
$$s_{13} = v_1 v_3^T$$
0
0
0
$$s_{1n} = v_1 v_n^T$$

$$s_{n1} = v_n v_1^T$$
$$s_{n2} = v_n v_2^T$$
$$s_{n3} = v_n v_3^T$$
0
0
0
$$s_{nn} = v_n v_n^T$$

We will calculate the dot products for every word vector pairs we have



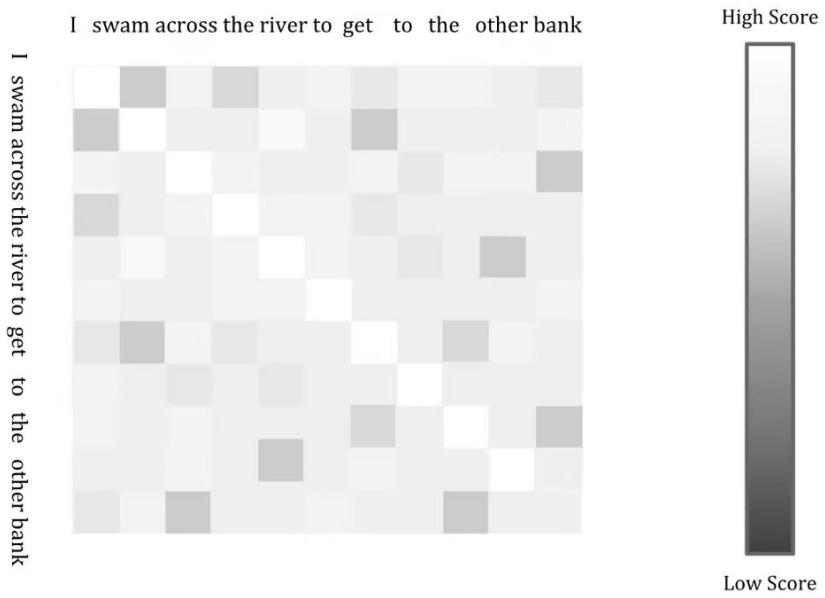


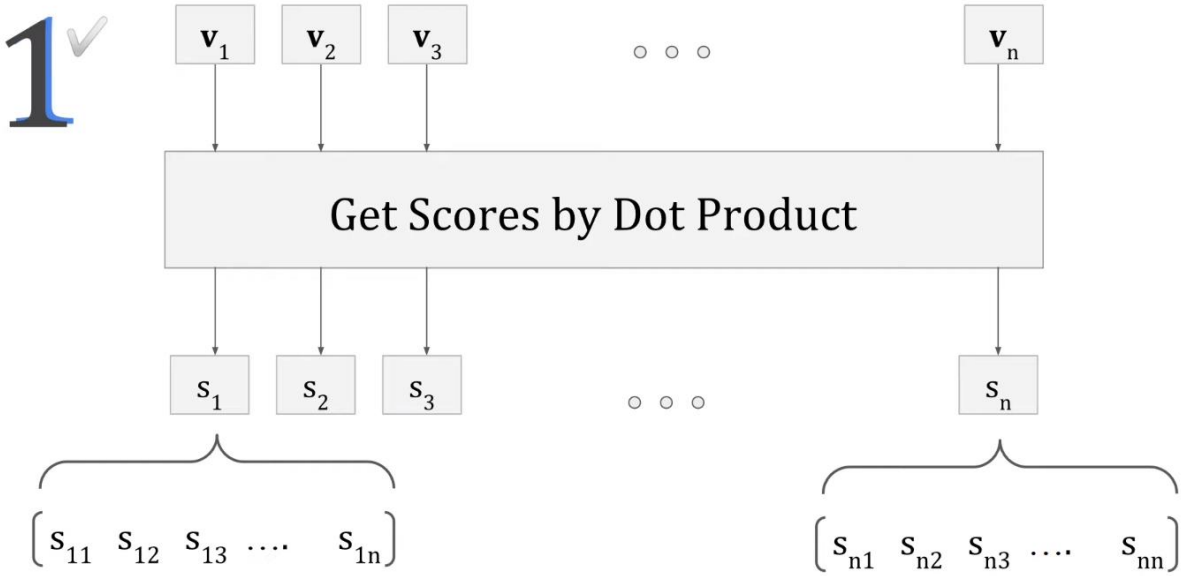
$$s_{n1} = v_n v_1^T$$
$$s_{n2} = v_n v_2^T$$
$$s_{n3} = v_n v_3^T$$

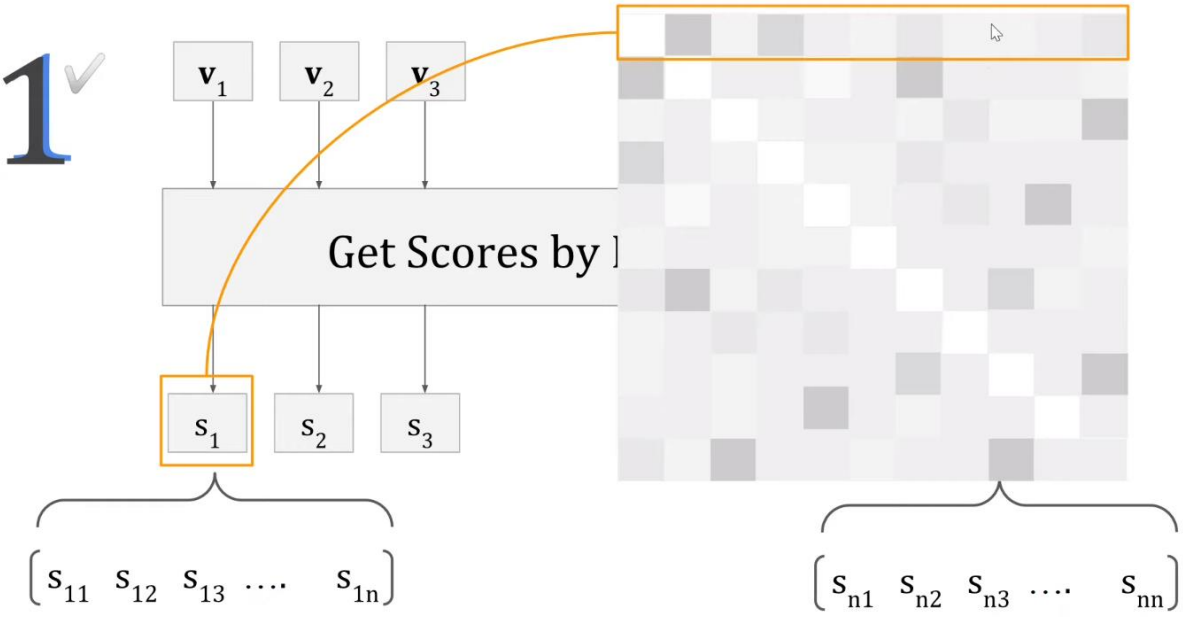
$$s_{nn} = v_n v_n^T$$

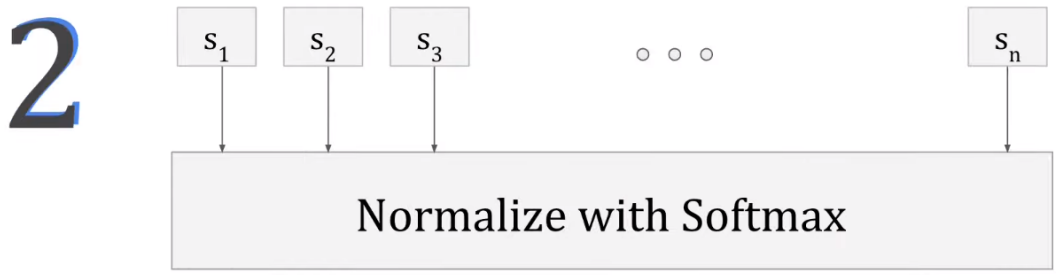
@ark_aung

The dot product will be higher for word vectors that are closer and smaller for far away word vectors.

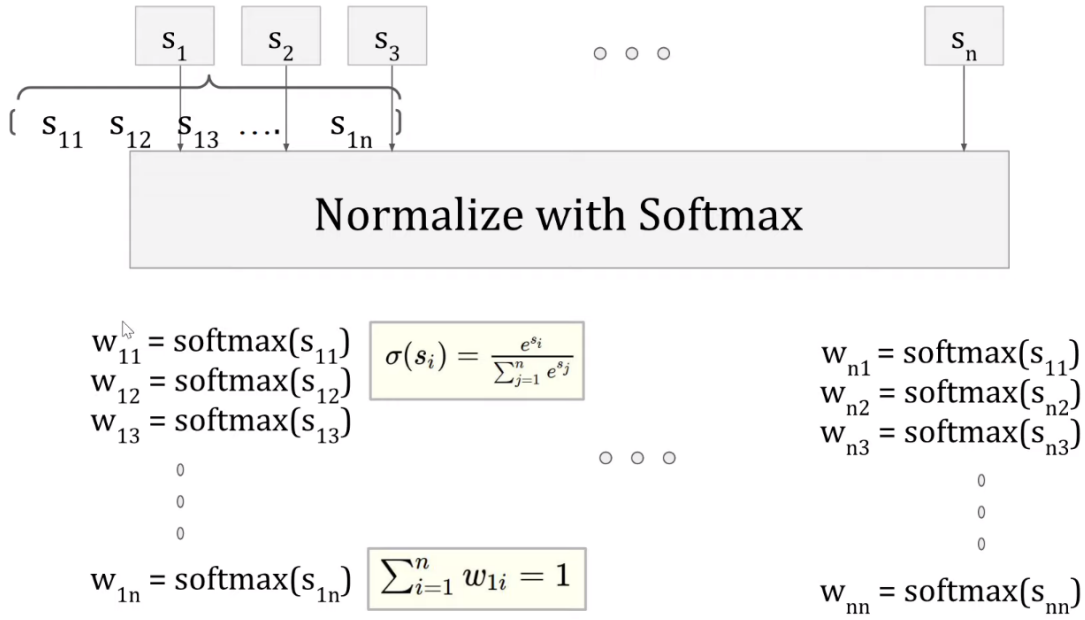

I swam across the river to get to the other bank
I swam across the river to get to the other bank
High Score
Low Score

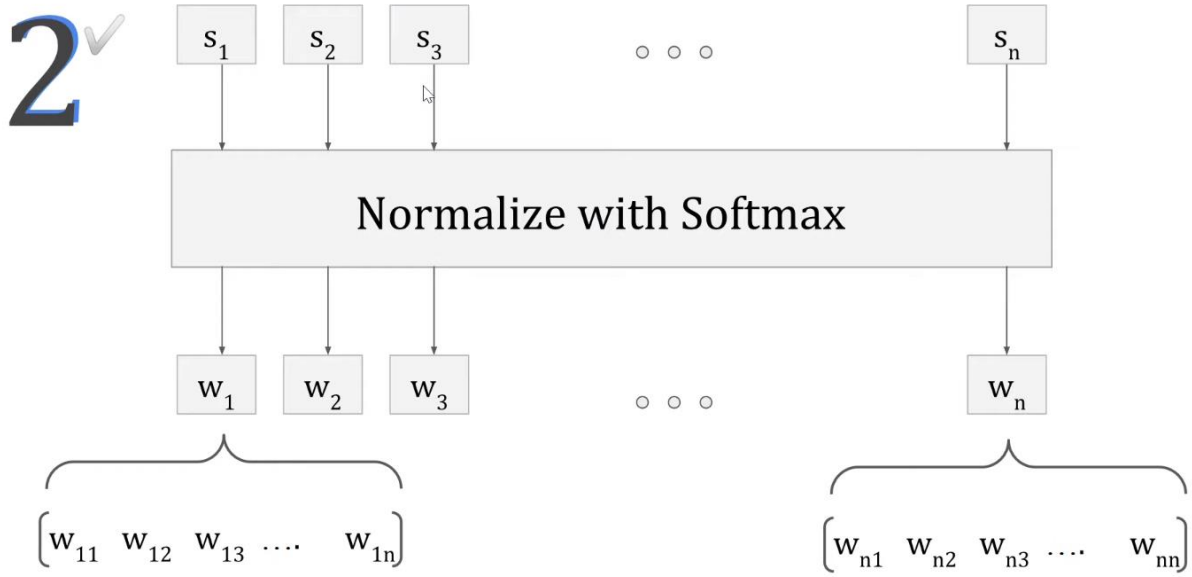

v1
v2
v3
vn
Get Scores by Dot Product
s1
s2
s3
sn
s11 s12 s13 .... s1n
sn1 sn2 sn3 .... snn

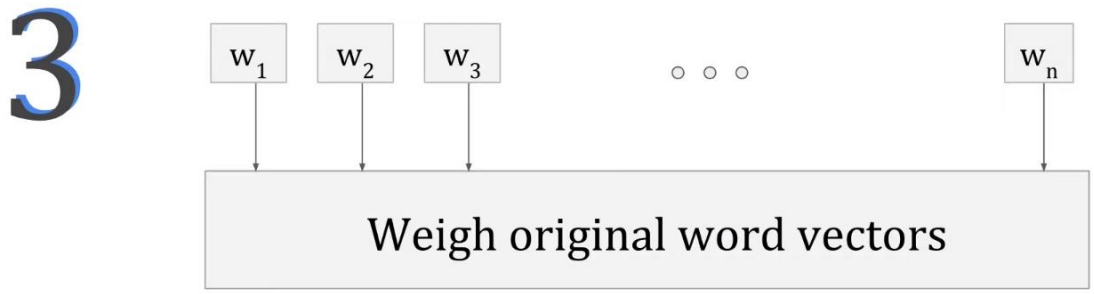

v1
v2
v3
Get Scores by
s1
s2
s3
s11 s12 s13 .... s1n
sn1 sn2 sn3 .... snn



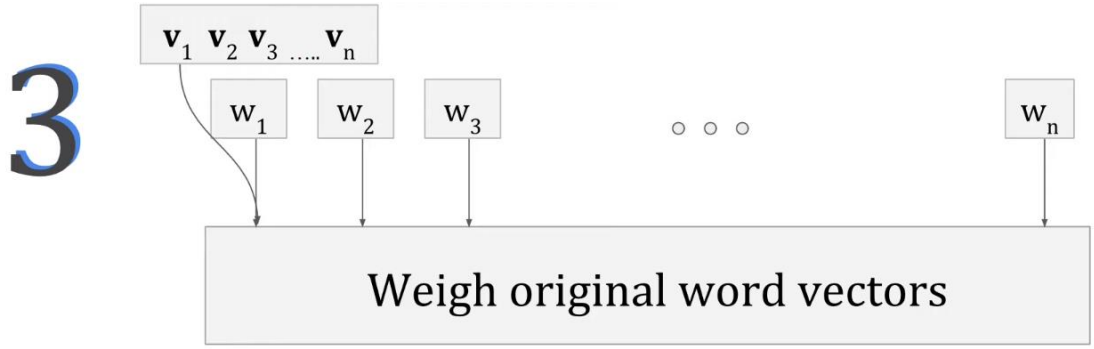
After doing dot products between word vectors, we will get the scores that will be across a wide range from very small to very large values. Using SoftMax, we need to normalize the scores (get the weight values $\mathbf{W_{ii}}$!) such that when we add up each entry in the $\mathbf{S_{ij}}$ row vectors, we will get the value 1.





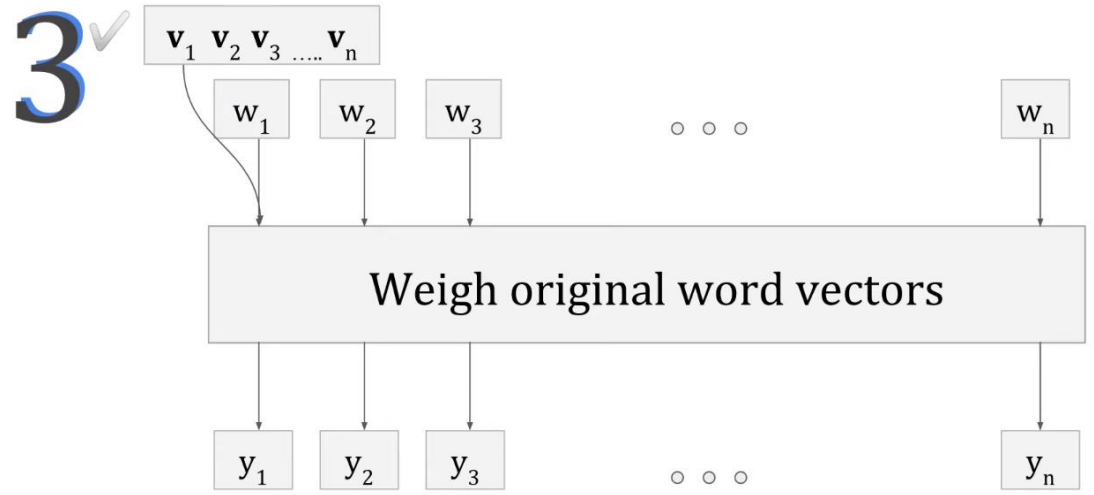


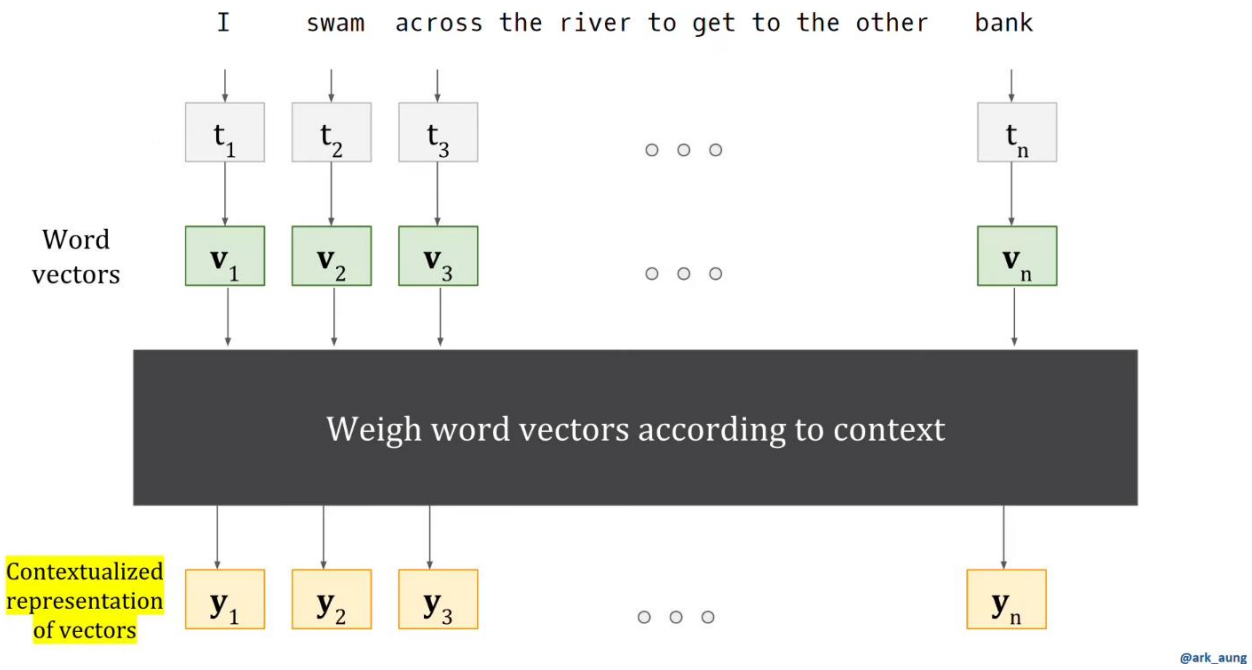
We can now put the corresponding weight values $\mathbf{W_i}$ to weigh each of the original word vectors $\mathbf{V_i}$.



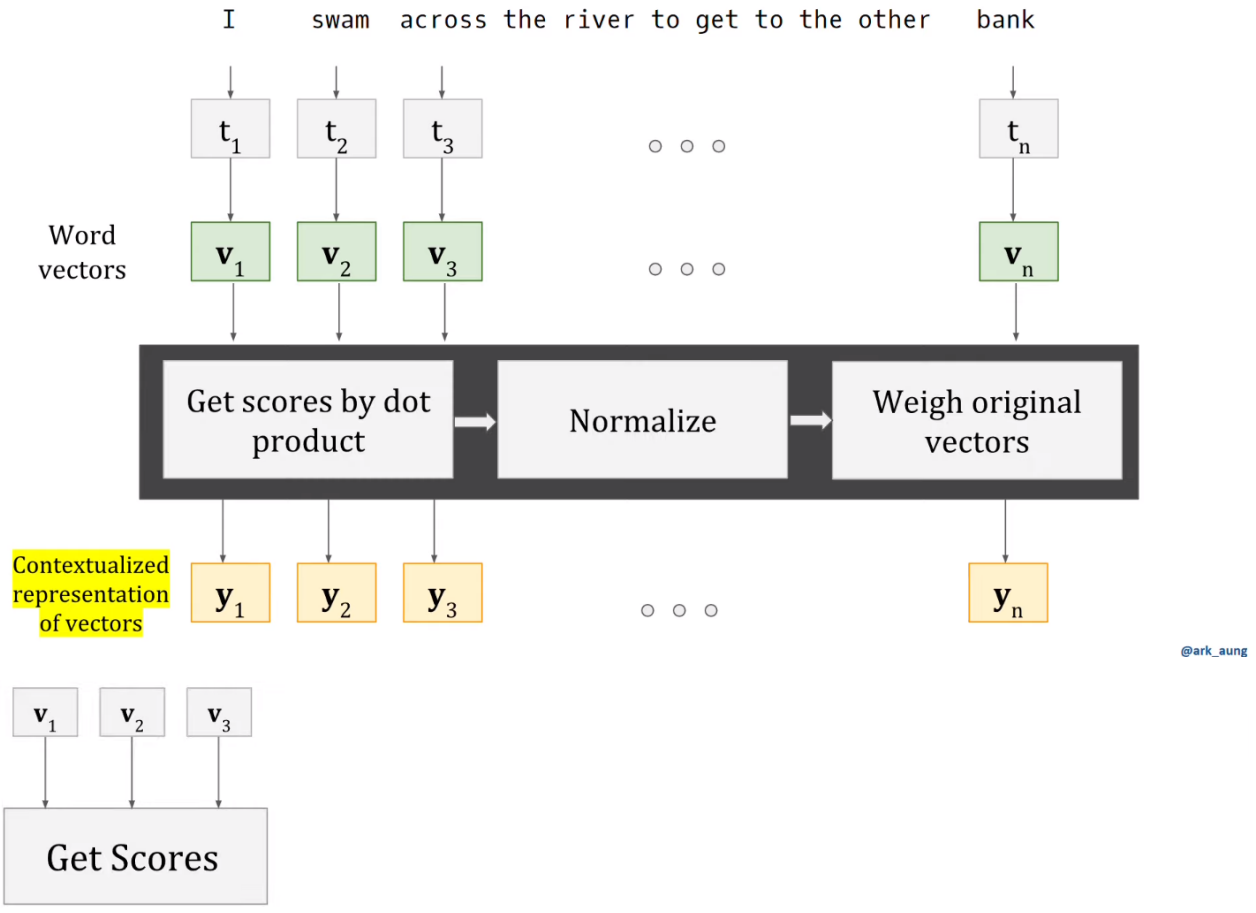
$$\mathbf{y}_1 = w_{11}\mathbf{v}_1 + w_{12}\mathbf{v}_2 + w_{13}\mathbf{v}_3 + \ldots + w_{1n}\mathbf{v}_n$$
$$\mathbf{y}_2 = w_{21}\mathbf{v}_1 + w_{22}\mathbf{v}_2 + w_{23}\mathbf{v}_3 + \ldots + w_{2n}\mathbf{v}_n$$
$$\mathbf{y}_3 = w_{31}\mathbf{v}_1 + w_{32}\mathbf{v}_2 + w_{33}\mathbf{v}_3 + \ldots + w_{3n}\mathbf{v}_n$$
$$\vdots$$
$$\mathbf{y}_n = w_{n1}\mathbf{v}_1 + w_{n2}\mathbf{v}_2 + w_{n3}\mathbf{v}_3 + \ldots + w_{nn}\mathbf{v}_n$$

This results of the $\mathbf{Y_i}$ vectors from the dot product of the weights $\mathbf{W_i}$ and the original word vectors $\mathbf{V_i}$ ***signifies that our original word vectors are now pointing more towards the direction influenced by its neighbors***.

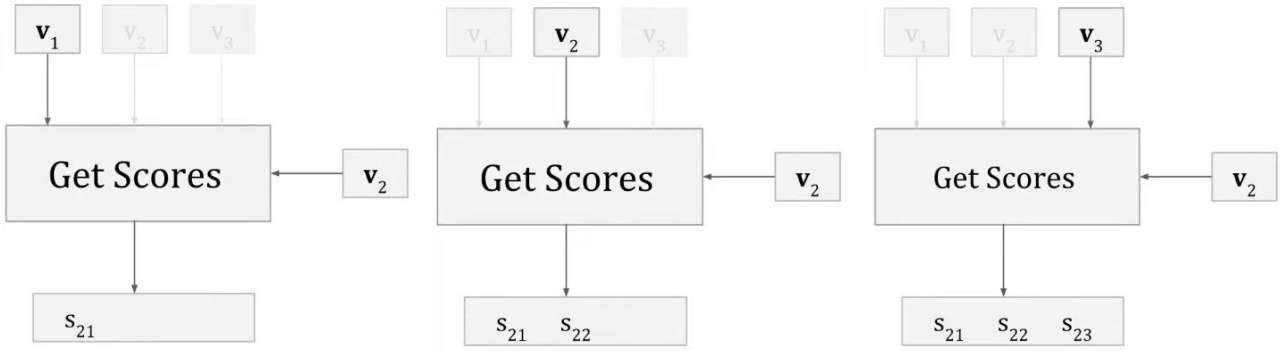


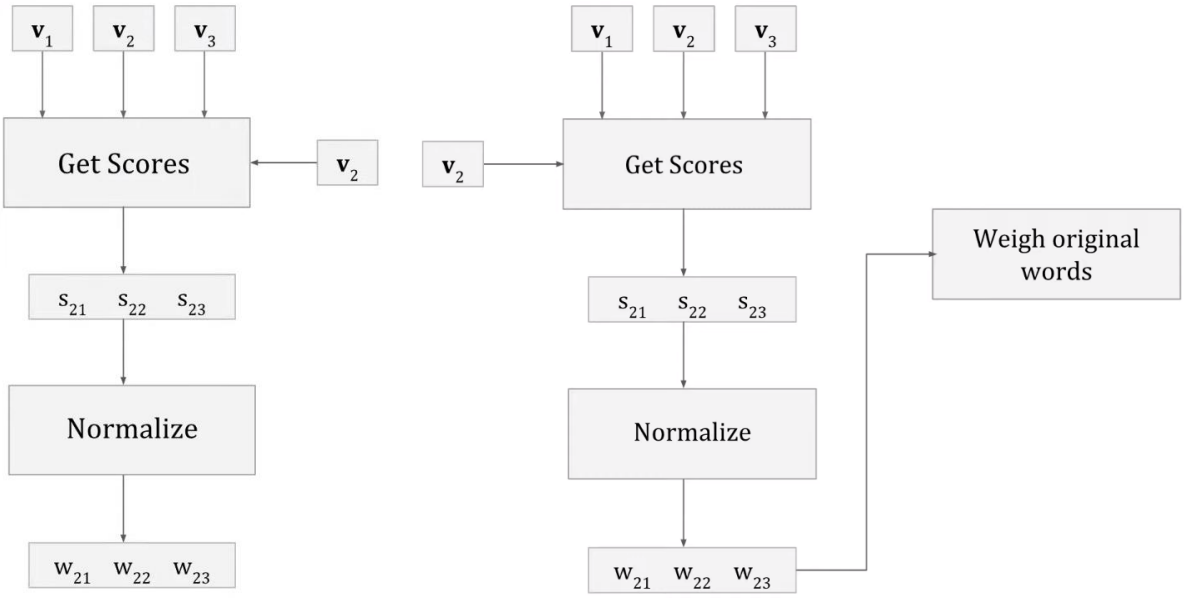


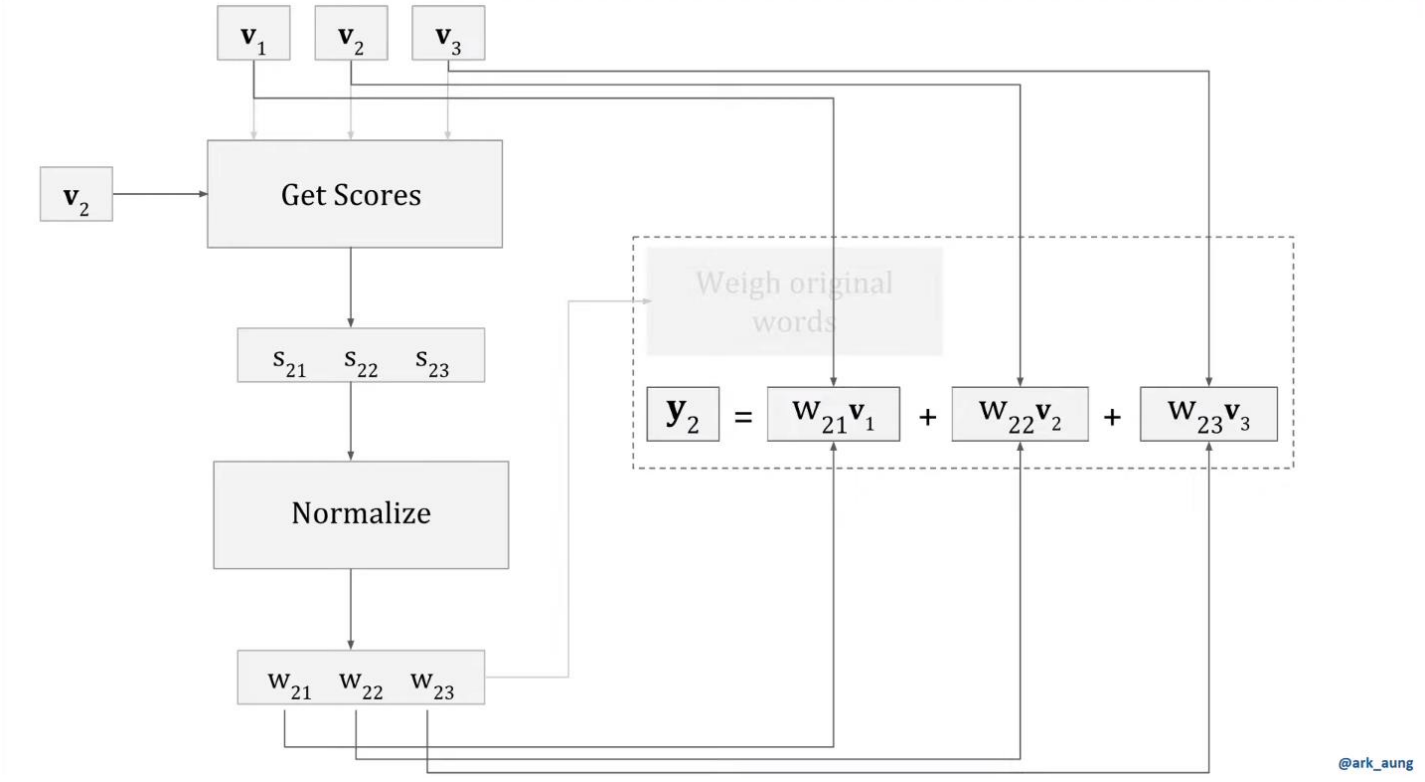
This completes the three steps for a self-attention mechanism, $\mathbf{Y}_i$ is the contextualized word vector pointing in the direction that is influenced by its word vector neighbors.



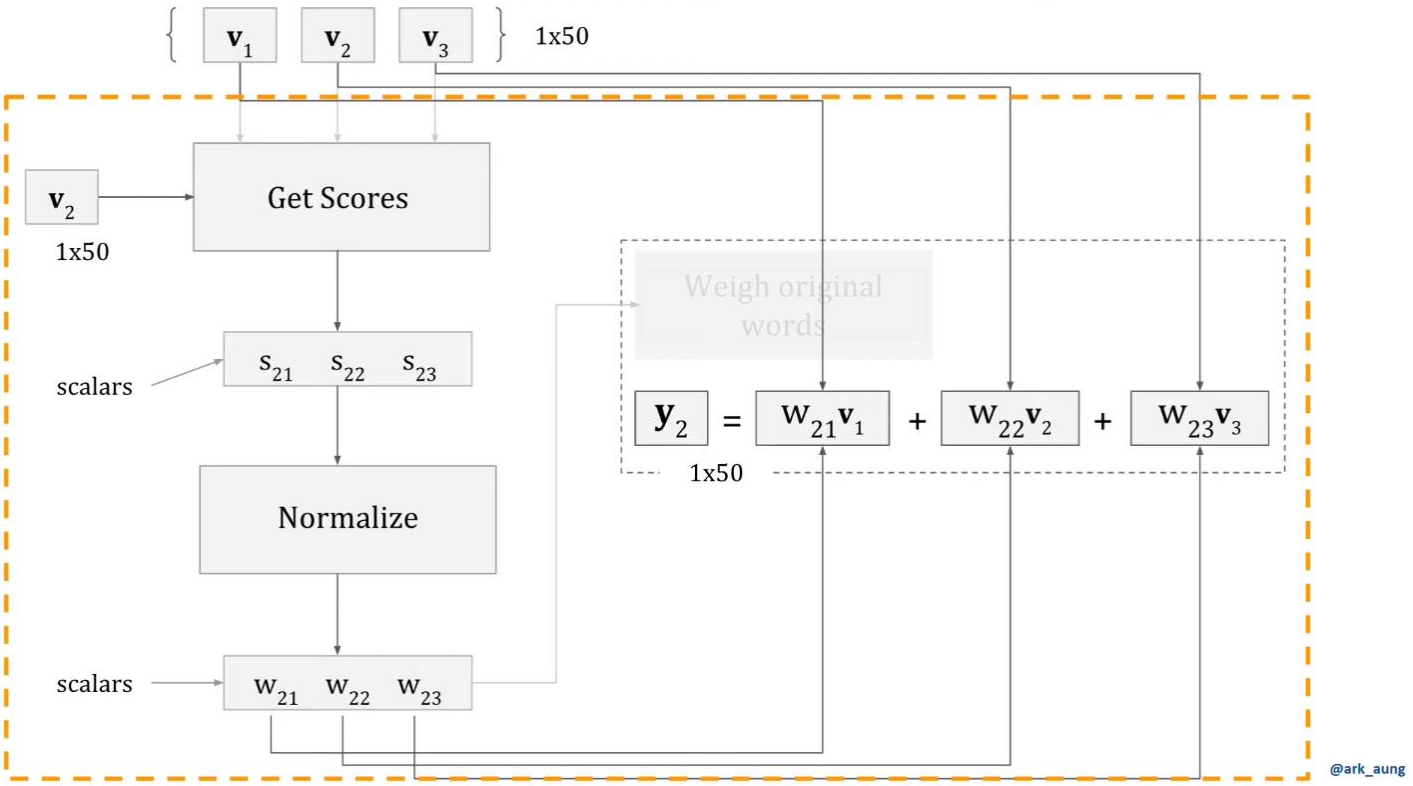
Assume we have a sentence with only three words being represented by their individual word vectors as above. Assume our task is to enrich the word vector $\mathbf{V}_2$ by finding its contextualized representation.





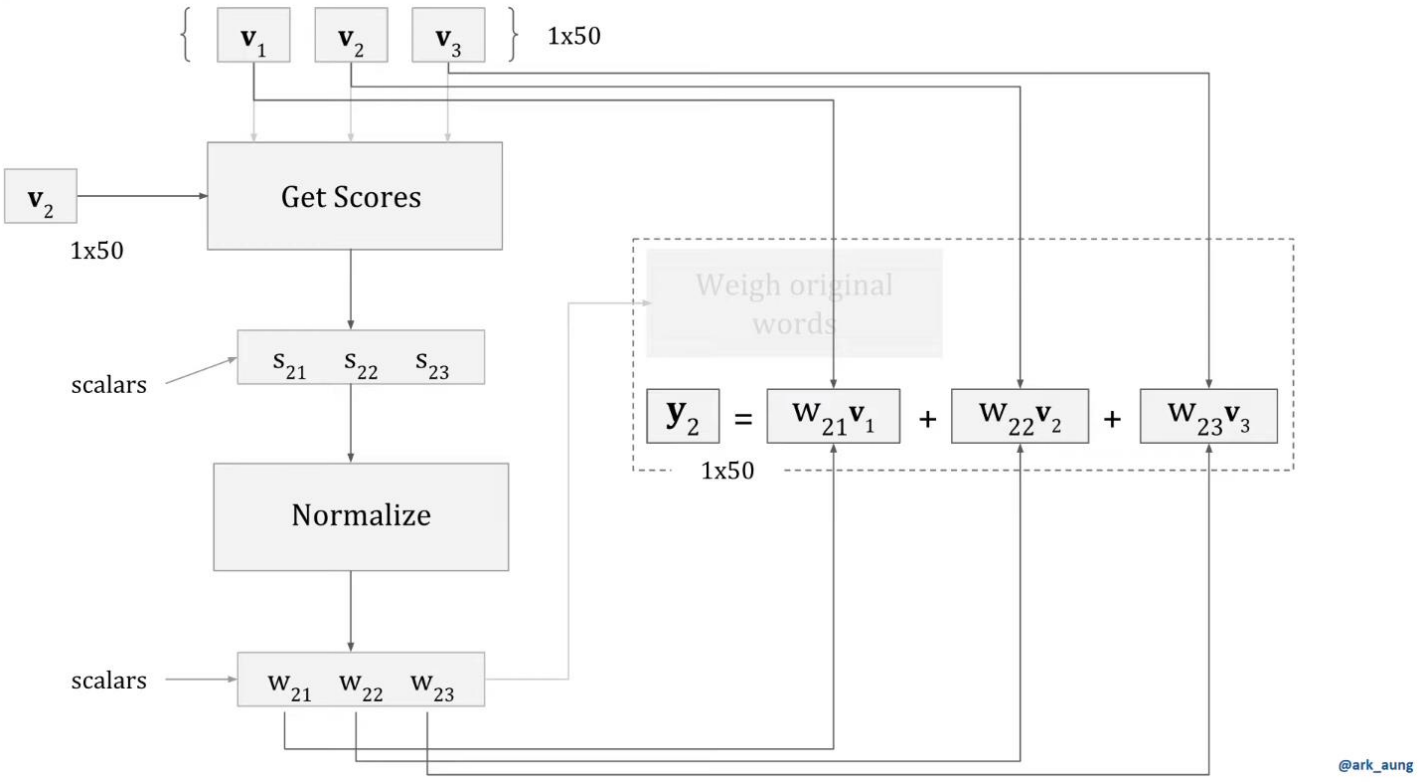


Y_2 is now the contextualized representation of our original word vector **V_2**.

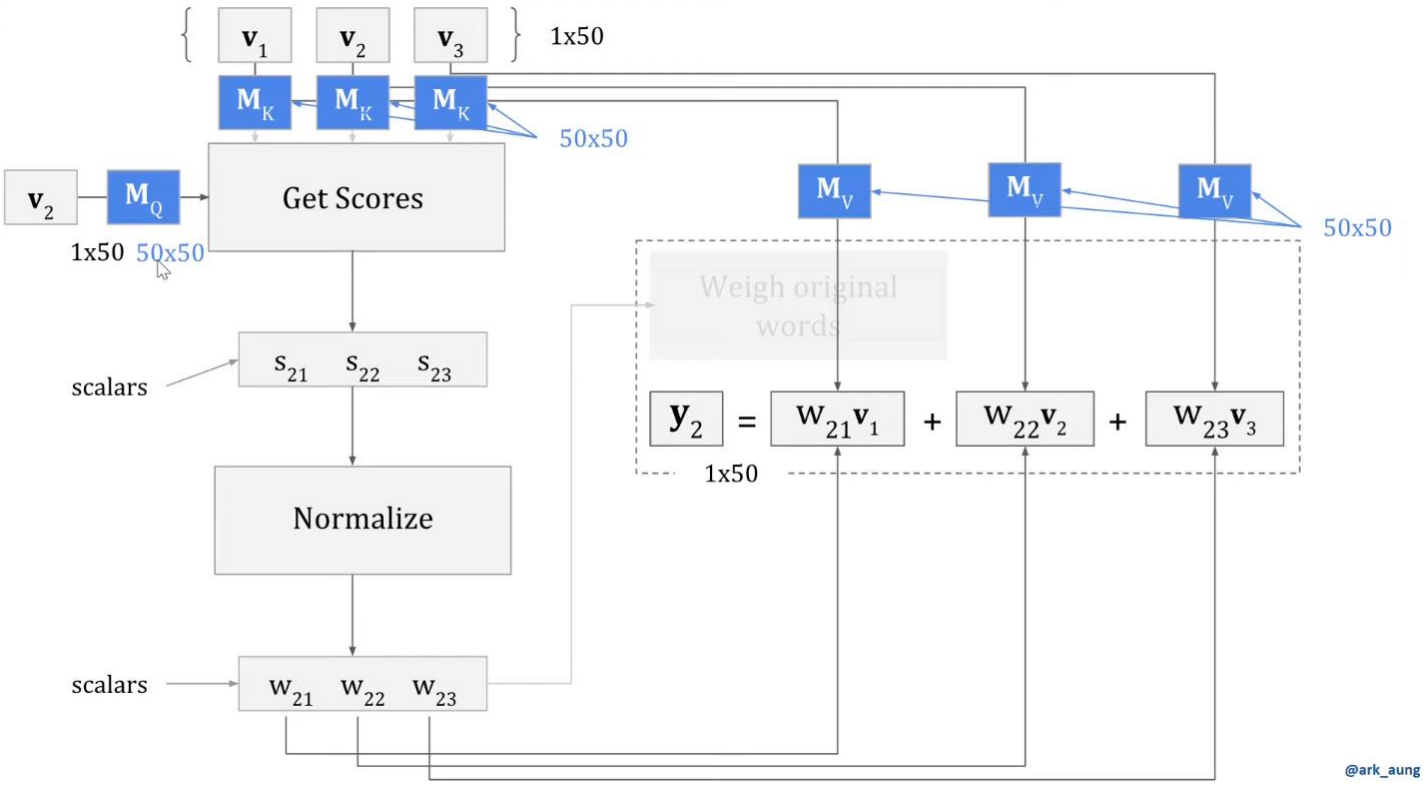


The yellow box is an attention mechanism or a self-attention mechanism.

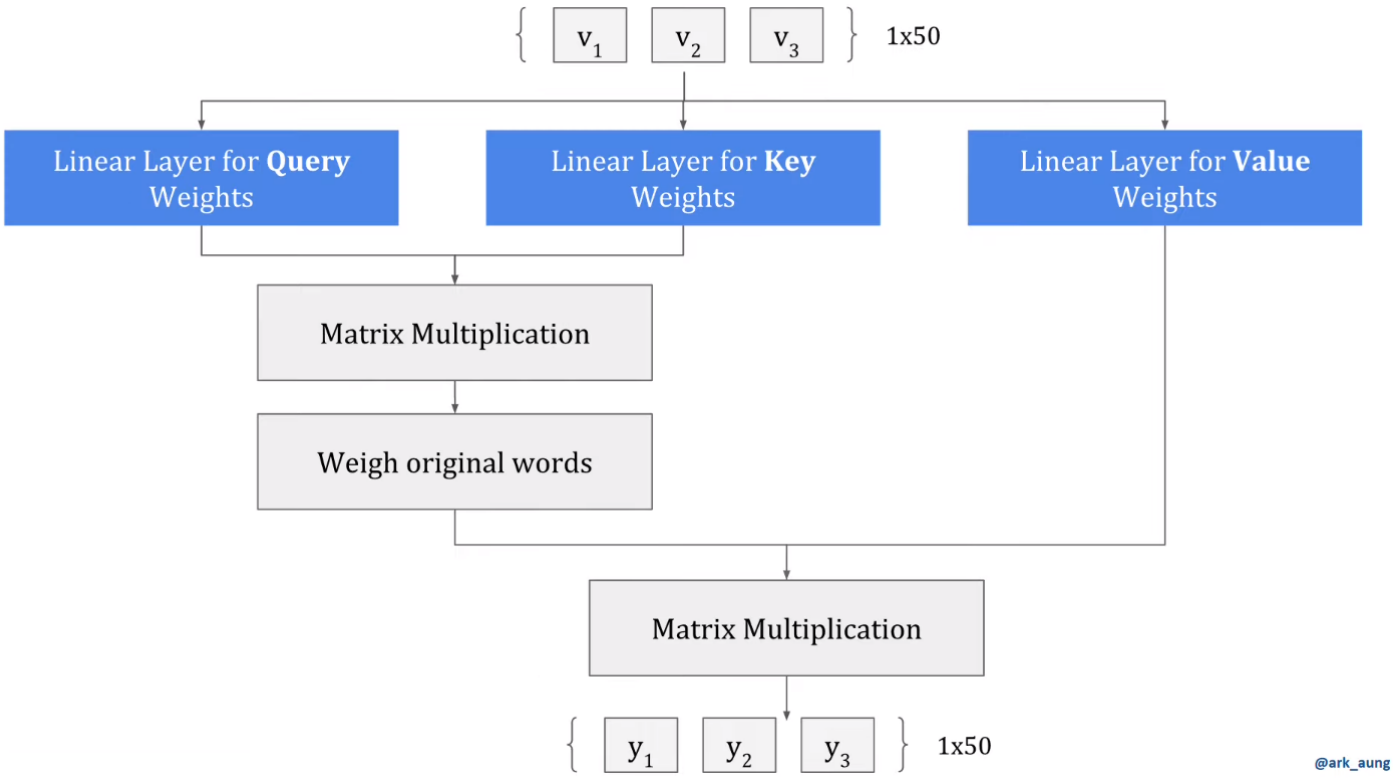
Where are all the learnable weights?



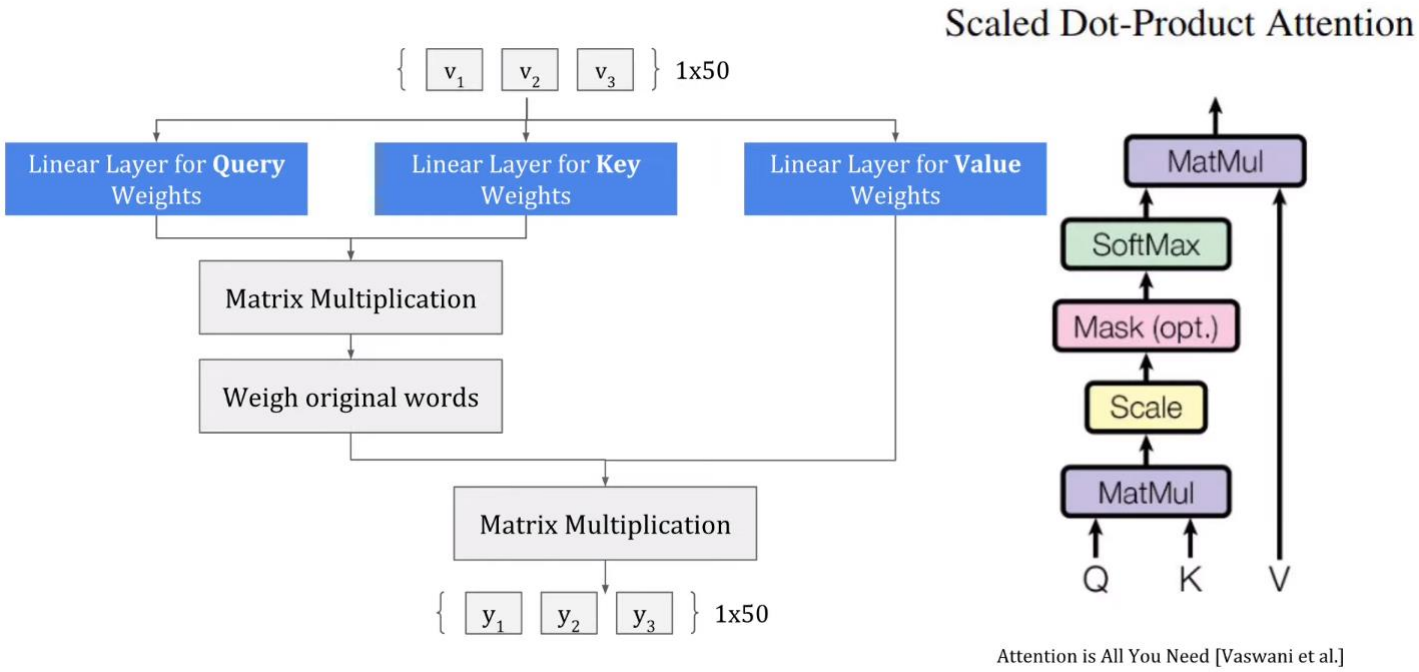
We now inject something used repeatedly in self-attention mechanisms, which word is closer to our attention word $\mathbf{V_2}$? Note that the word vectors $\mathbf{V_i}$ are like **keys** in a DB, and we have a $\mathbf{v_i}$ **query** about which words are closer to some other word. We need to find the **value** that our **key** points to for each **query**. The $\mathbf{S_{ij}}$ are like a dot product of our **key.query**.
Note that the self-attention mechanism is simply using the same word vectors for its Keys, Queries and Values!



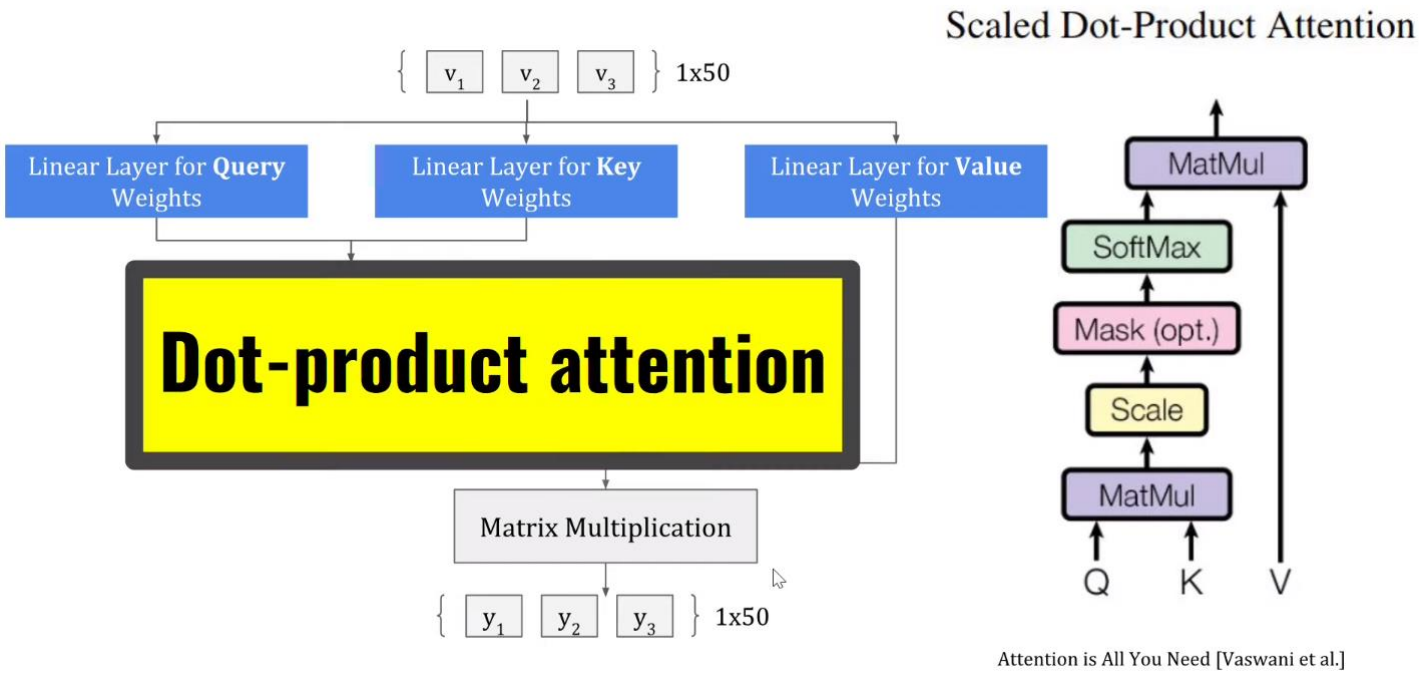
We can inject a weight vectors, the query weights $\mathbf{M_Q}$, key weights $\mathbf{M_K}$, and value weights $\mathbf{M_v}$ that are optimizable during back-propagation but still preserving the dimensions.



We can now put things into a NN architecture/module as linear layers that are dense layers for the Query weights, Key weights and the Value weights. We are essentially replacing the several dot product multiplications with better matrix multiplications, we can implement this in PyTorch or TensorFlow.

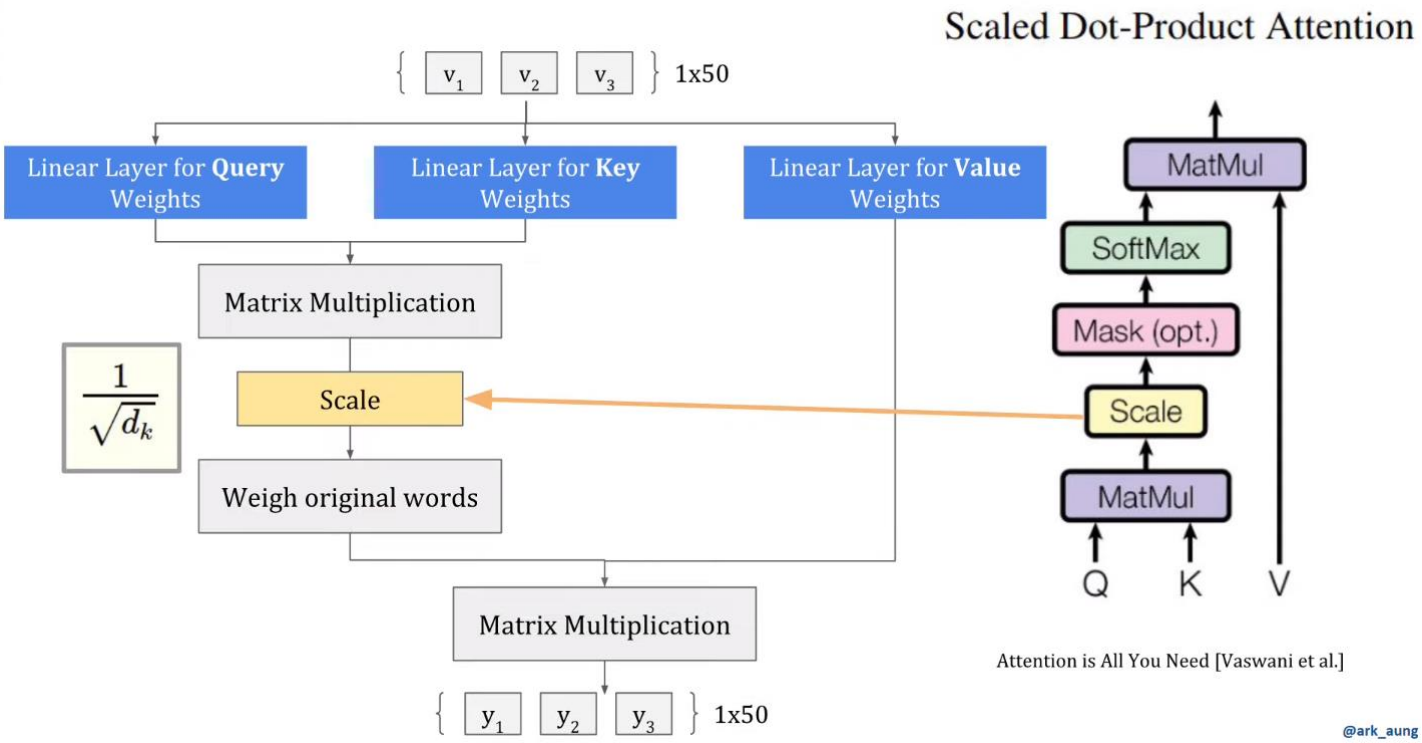


Attention is All You Need [Vaswani et al.]

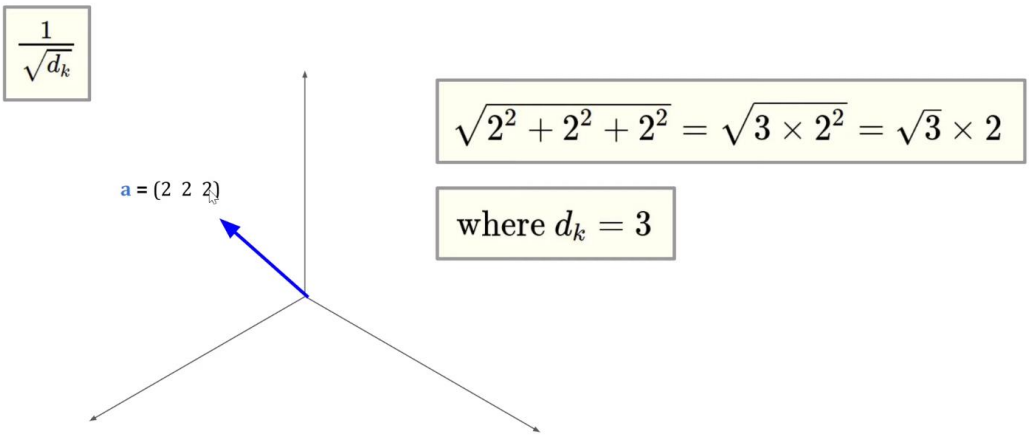


Attention is All You Need [Vaswani et al.]

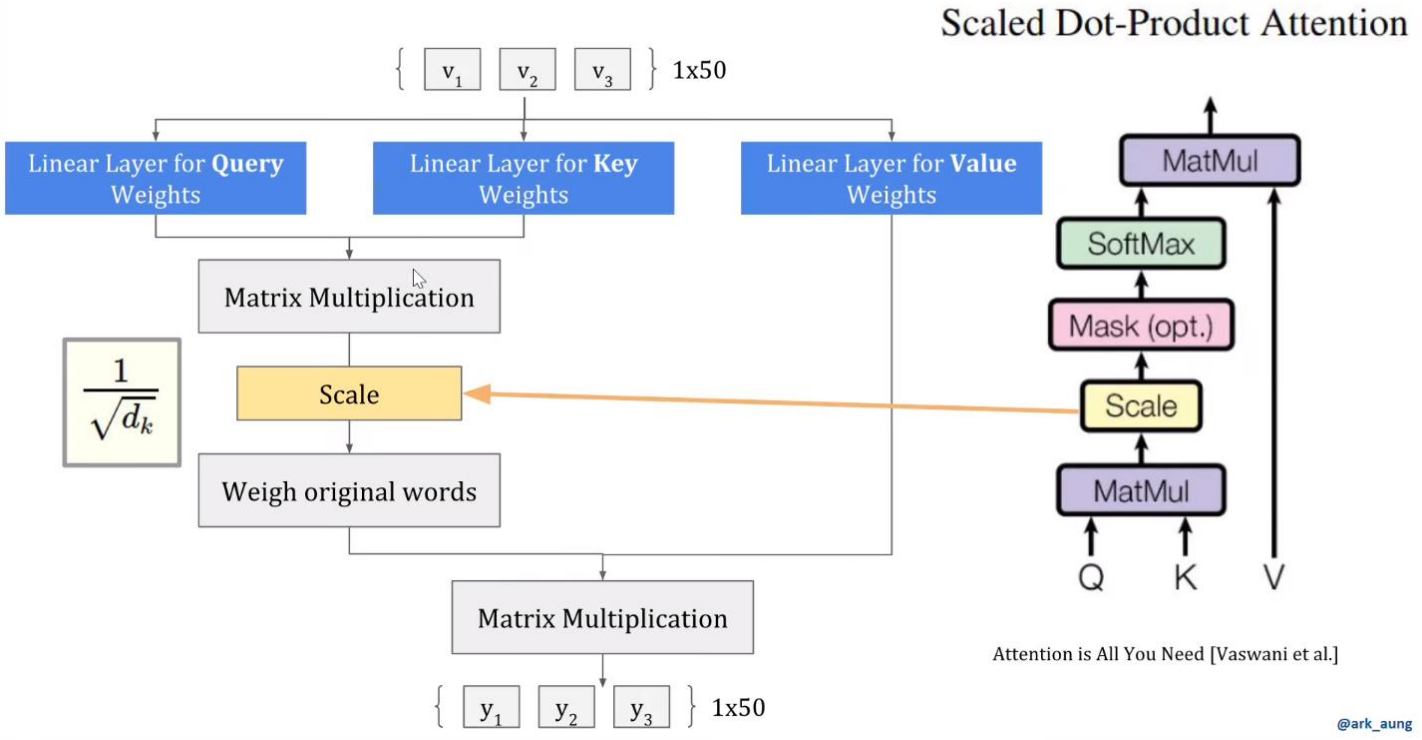
Remember that we have skipped the Scale part for now to implement just the **Dot Product Attention**!

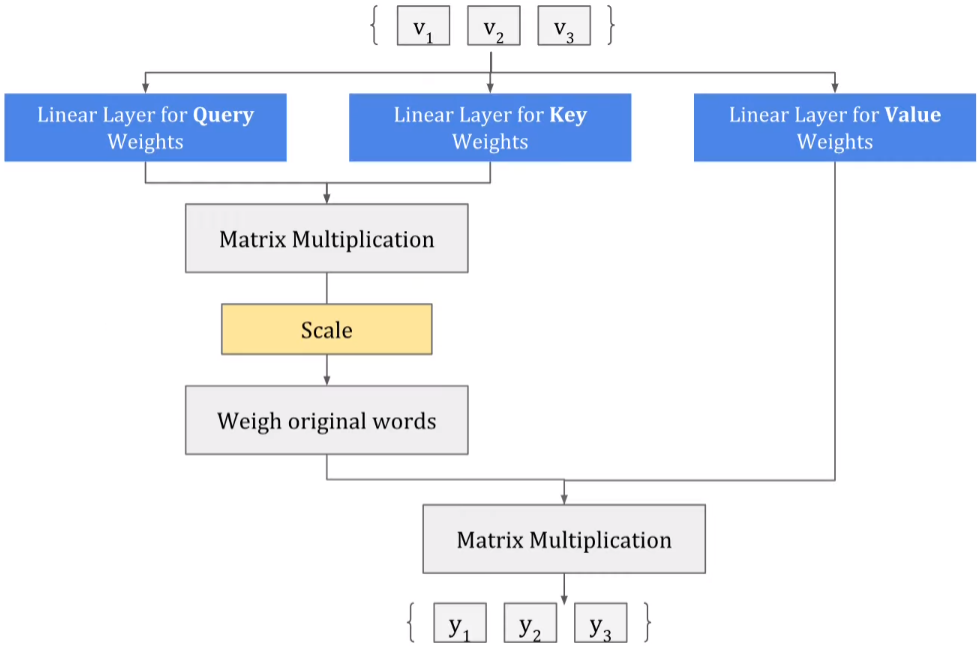


The Scale is doing an element-wise multiplication with the boxed value, where $\mathbf{d_k}$ is related to the dimension of our input vectors.



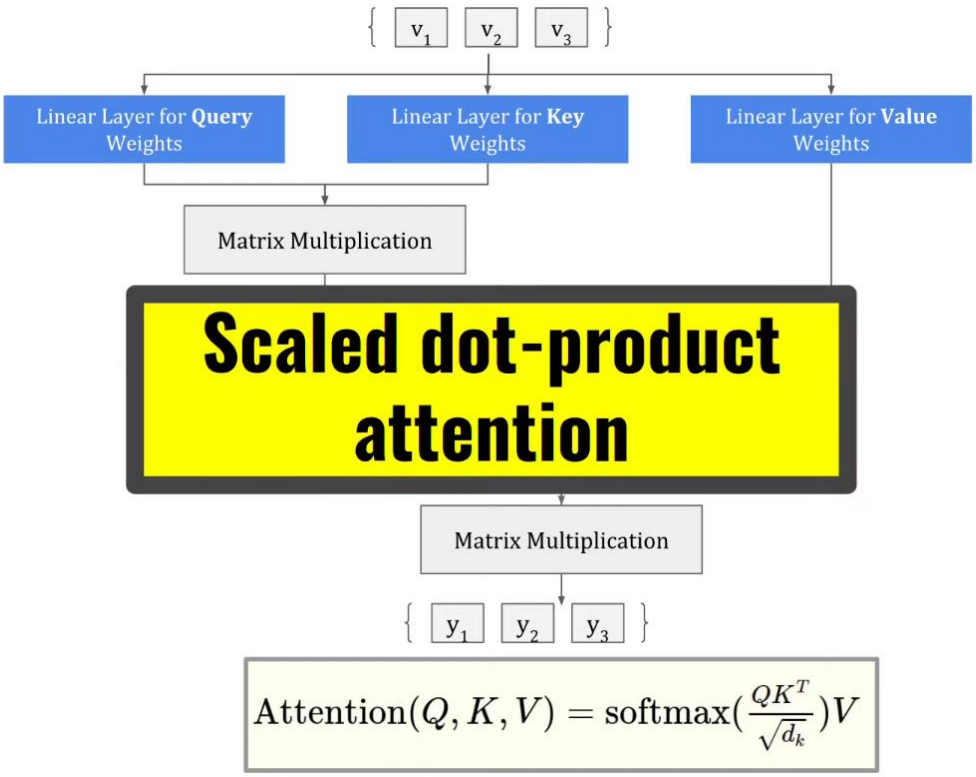
$\mathbf{d_k}$ for a vector is the average of the vector values in all its directions, scaling mean we are normalizing each vector to reduce the vectors effect on the SoftMax function values to not affect the overall gradient slope/speed.





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

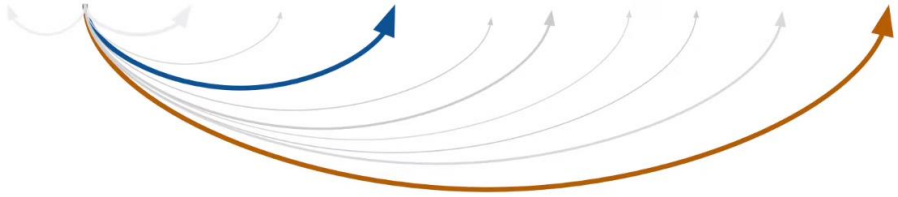
@ark_aung



@ark_aung

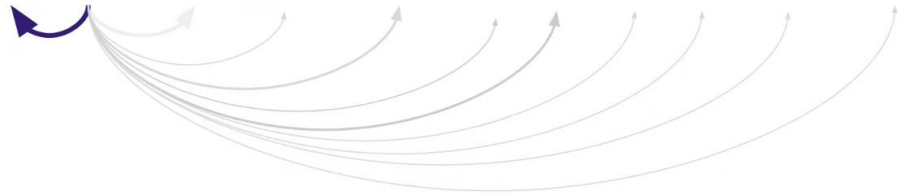
When you are looking at a word in a sentence, you are **giving attention to more than one thing**

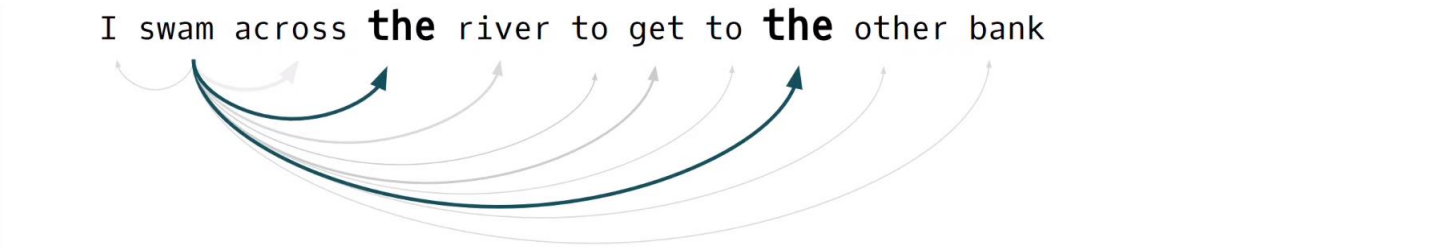
I swam across the **river** to get to the other **bank**



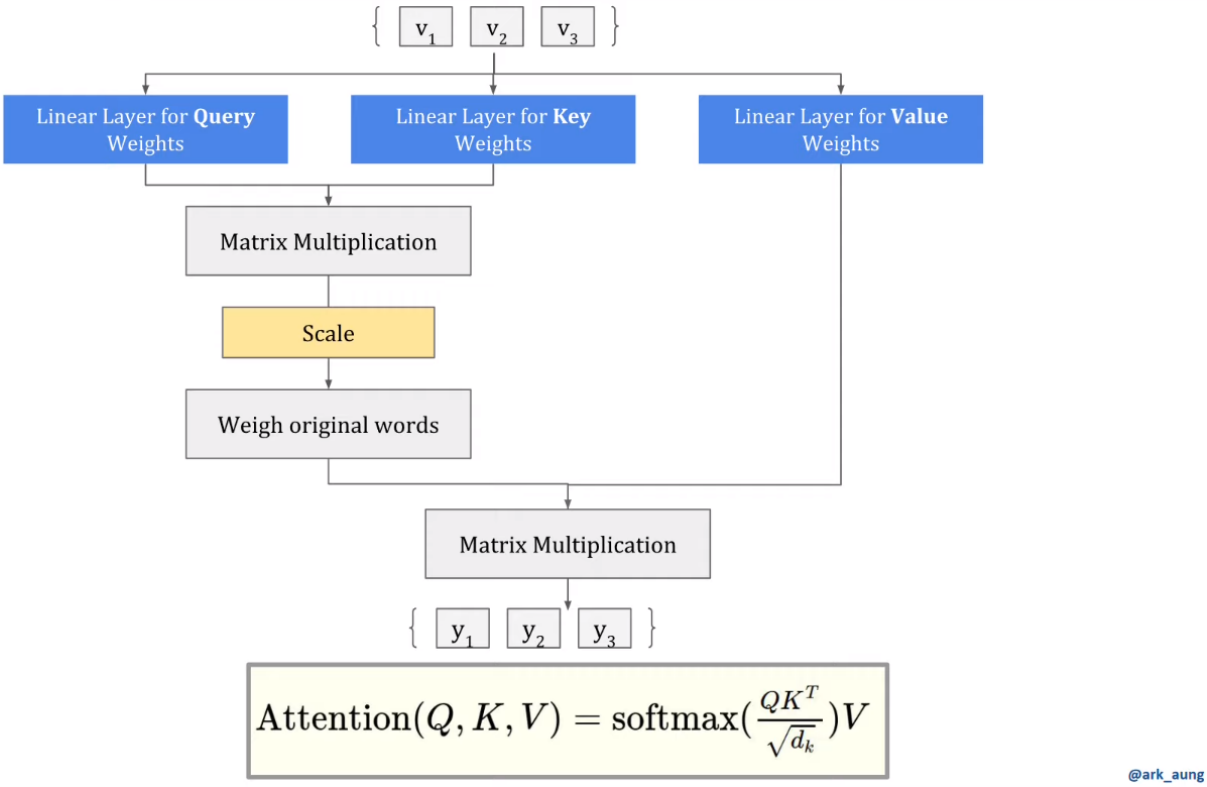
I swam across the river to get **to the other bank**

I swam across the river to get to the other bank

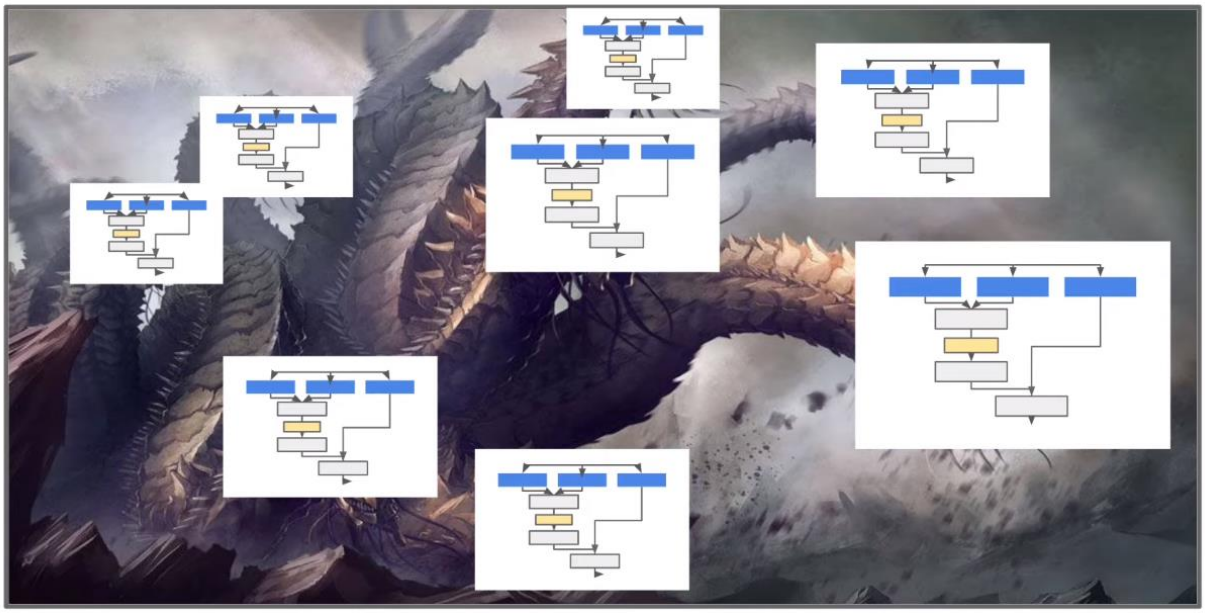




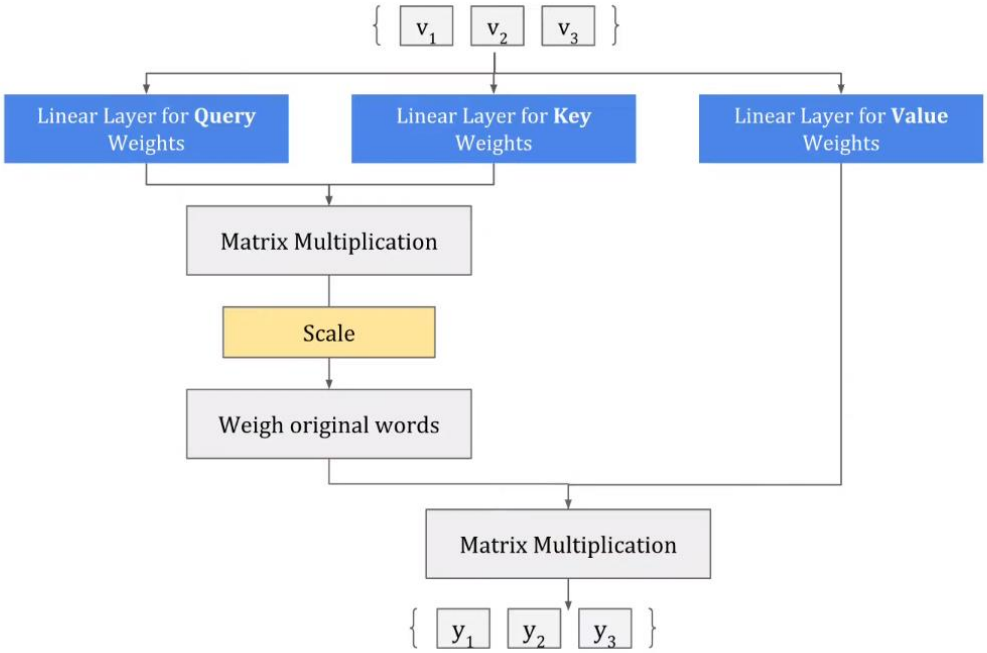
Using a single attention mechanism to capture multiple attentions can become problematic



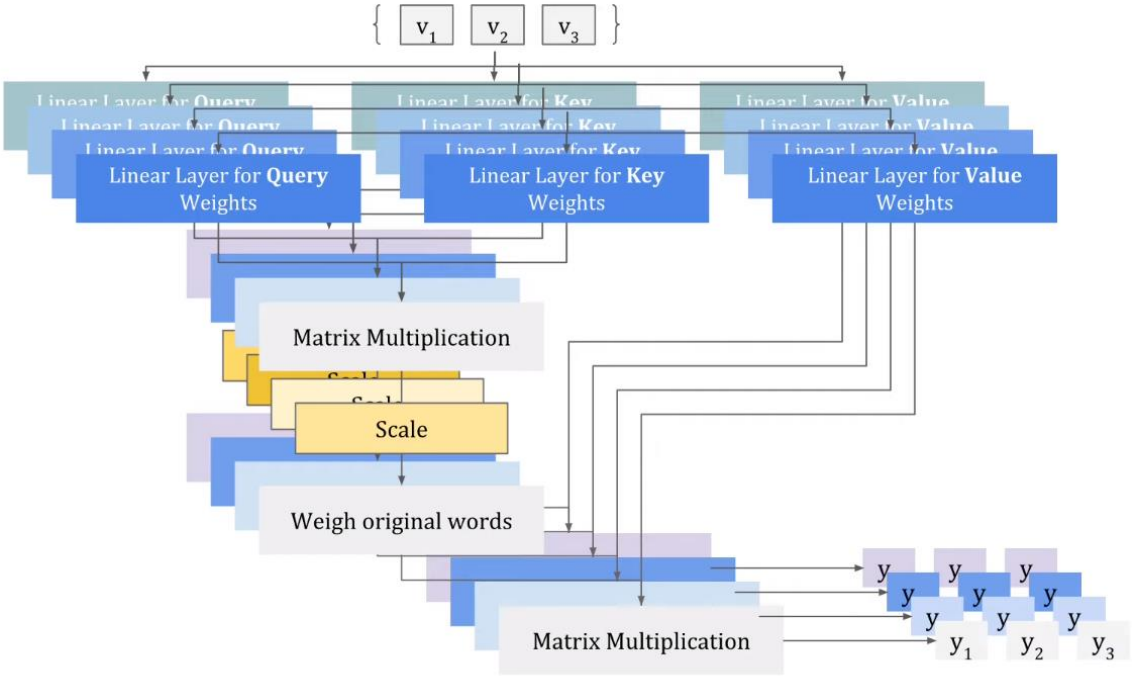
Multi-head attention



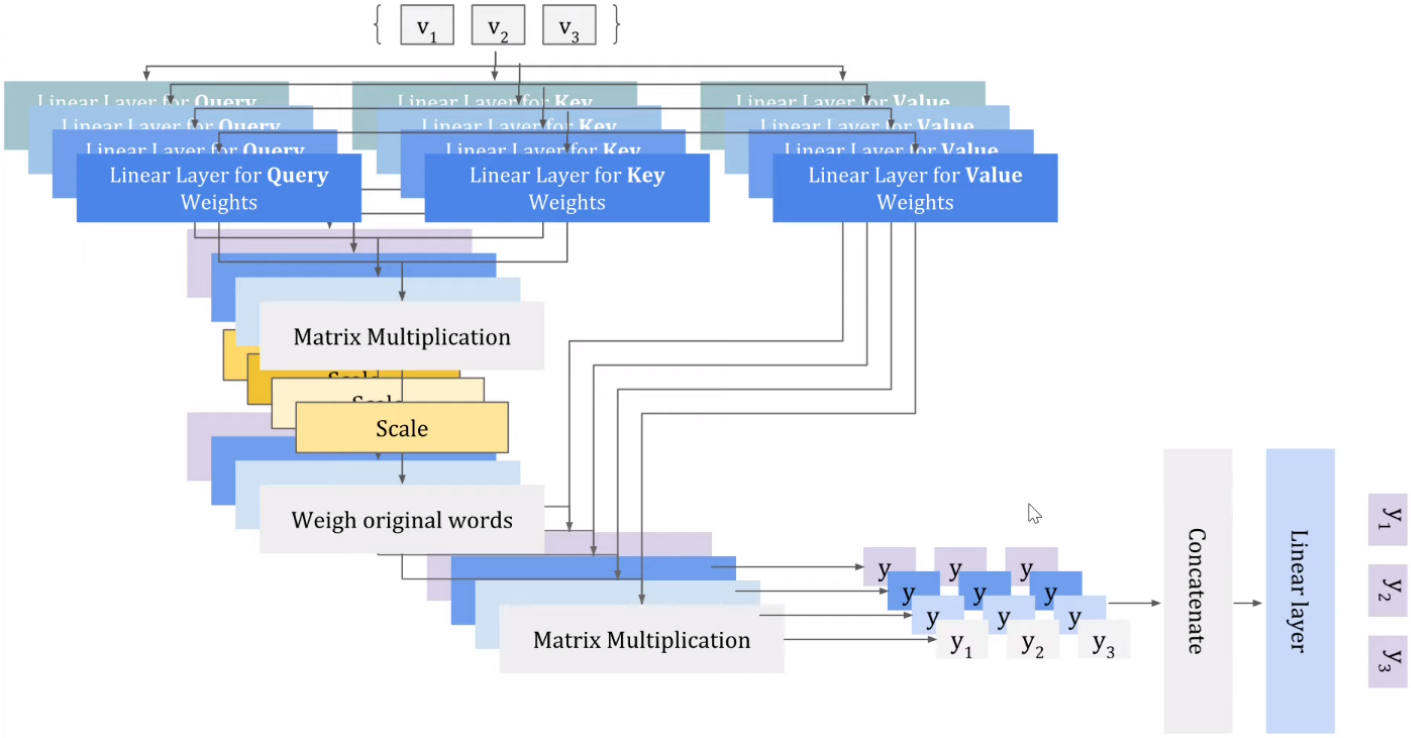
We can use above where each attention head pays attention to a different word



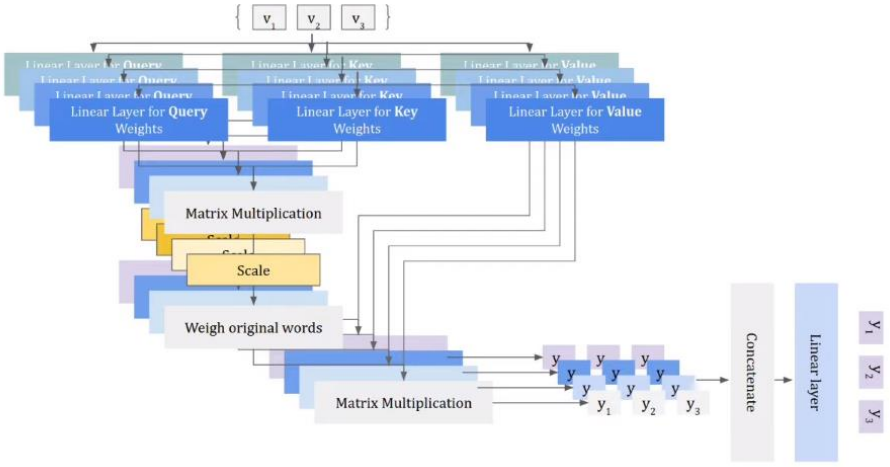
We thus need a model with a single attention head above to the multi-head attention head architecture below

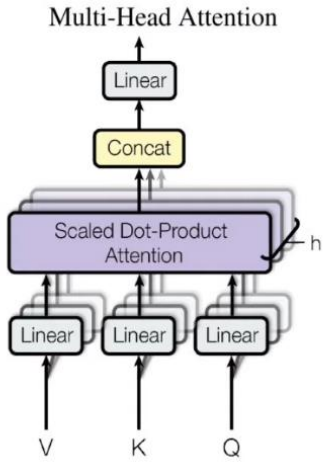


For the Multi-Head attention mechanism, we stack up the attention heads with multiple different Query, Key and Value weights so that they can learn different kinds of attentions based on the task that you are asking the transformer network to learn. For a **named-entity recognition task**, or **an autoregressive task** (like predicting the next word in a sentence!), or **a sentiment analysis task**, these weights may learn different kinds of attention at each attention head,

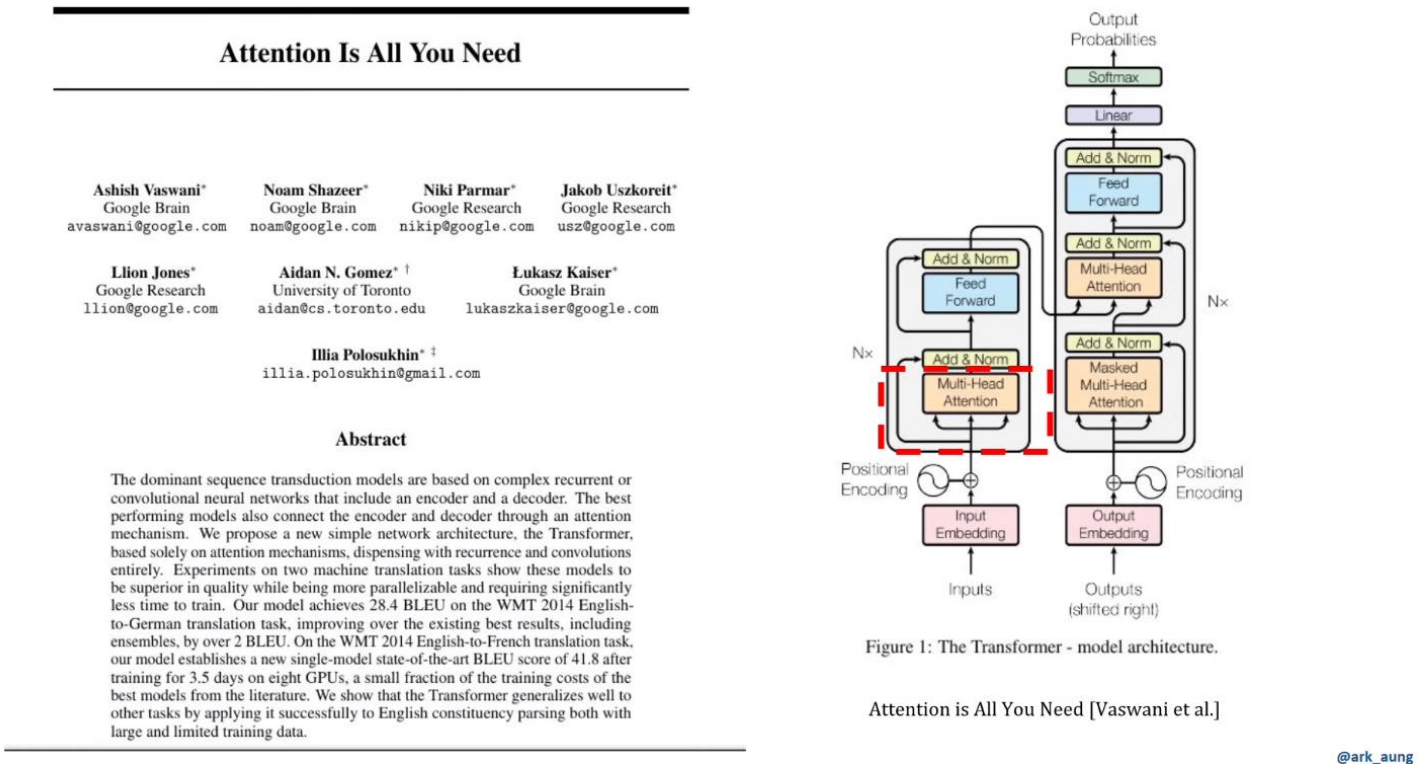


We then add linear layers to convert the outputs to the same dimensions as the input vectors dimensions.





Attention is All You Need [Vaswani et al.]



Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Figure 1: The Transformer - model architecture.

Attention is All You Need [Vaswani et al.]

@ark_aung

There are other parts of the transformer architecture around the **Multi-Head Attention mechanism** we saw here. The **Multi-Head Attention mechanism** are mostly used today in NLP tasks.

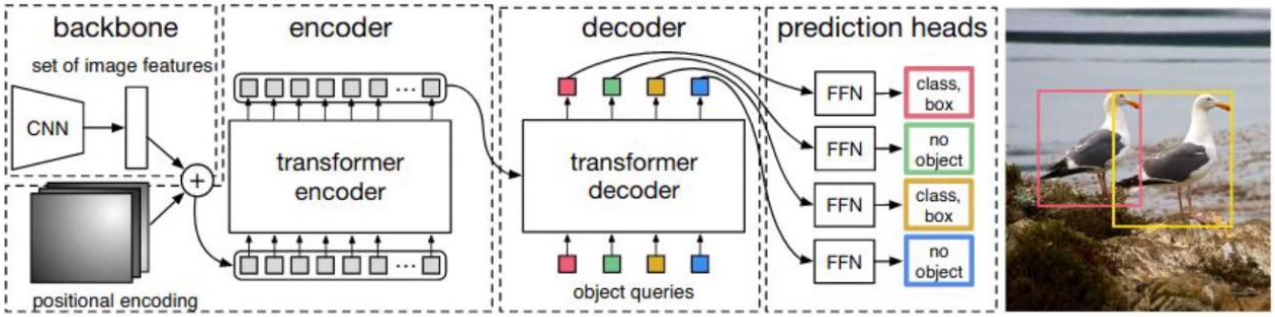
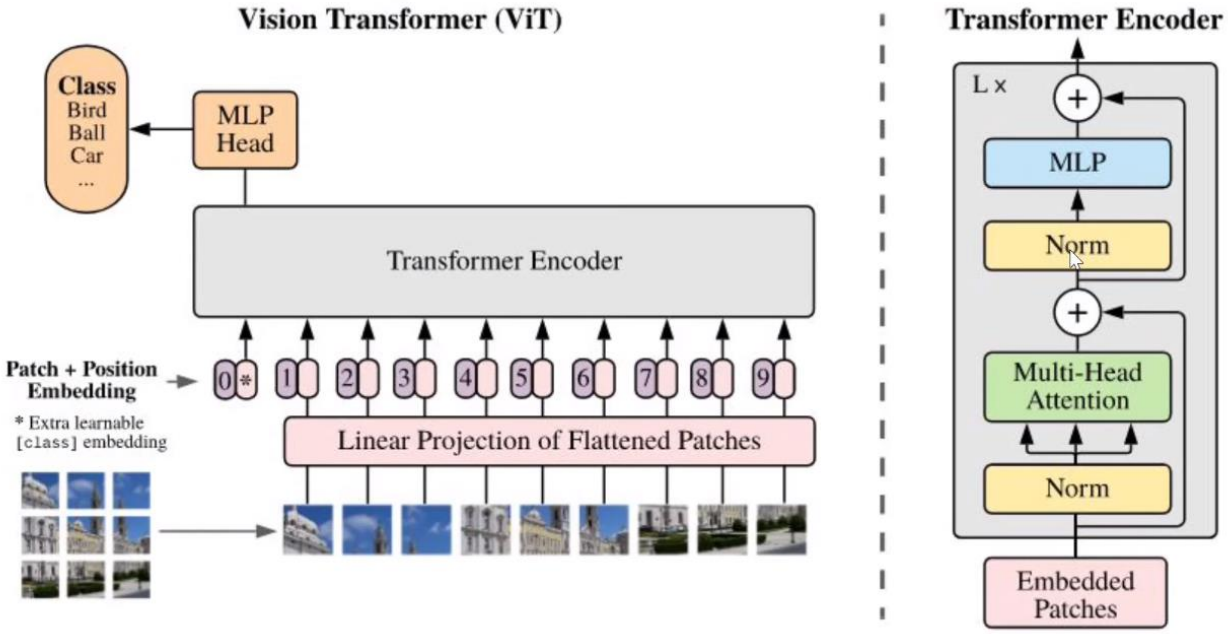


Fig. 2: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional embeddings, which we call *object queries*, and additionally attends to the encoder output. We pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a "`no object`" class.

End-to-End Object Detection with Transformers [Carion, Massa, et al.]

Even Vision related tasks are being done with transformers today by using it to replace attention mechanisms of vision encoders in object detection tasks.



An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale [Anonymous ICLR- under review]

@ark_aung

They are using transformers to completely replace CNNs for vision tasks