SELF ATTENTION IN TRANSFORMERS

RAISONI GROUP
— a vision beyond —

Consider two text statements:
1) money bank grows
2) river bank flows

Here, the meaning of word bank is different in both statements which the static word embedding won't be able to capture. So, we'll need contextual word embeddings.

So, we'll represent the two statements in this way-

money bank grows	river bank flows
money: 0.7 money + 0.2 bank + 0.1 grows	river: 0.8 river + 0.15 bank + 0.05 flows
bank: 0.25 money + 0.7 bank + 0.05 grows	bank: 0.2 river + 0.78 bank + 0.02 flows
grows: 0.1 money + 0.2 bank + 0.7 grows	flows: 0.4 river + 0.01 bank + 0.59 flows

Now here as we can see, each word of a statement is now depending on other words of their respective statement. This helps retain contextual information of the statements. But machine doesn't understand words so we need to represent each word as it's embedding form.

⇒

$e^{(new)}_{money} = 0.7\, e_{money} + 0.2\, e_{bank} + 0.1\, e_{grows}$ e_{money} : Embedding of word money

$e^{(new)}_{bank} = 0.25\, e_{money} + 0.7\, e_{bank} + 0.05\, e_{grows}$ e_{bank} : Embedding of word bank

$e^{(new)}_{grows} = 0.1\, e_{money} + 0.2\, e_{bank} + 0.7\, e_{grows}$ e_{grows} : Embedding of word grows

Embeddings are vector representation of words & can be n dimensional

The nos. before each word embedding represents the weight that describes the similarity between each word embedding.

i.e. $e^{(new)}_{money} = (e_{money} \cdot e^T_{money}) e_{money} + (e_{money} \cdot e^T_{bank}) e_{bank} + (e_{money} \cdot e^T_{grows}) e_{grows}$

$$e^{(new)}_{bank} = (e_{bank} \cdot e^T_{money}) e_{money} + (e_{bank} \cdot e^T_{bank}) e_{bank} + (e_{bank} \cdot e^T_{grows}) e_{grows}$$

$$e^{(new)}_{grows} = (e_{grows} \cdot e^T_{money}) e_{money} + (e_{grows} \cdot e^T_{bank}) e_{bank} + (e_{grows} \cdot e^T_{grows}) e_{grows}$$

* The similarity b/w each word embedding is calculated using simple dot product b/w them.

$\Rightarrow$ $e^{(new)}_{money} = S_{11} e_{money} + S_{12} e_{bank} + S_{13} e_{grows}$

$$e^{(new)}_{bank} = S_{21} e_{money} + S_{22} e_{bank} + S_{23} e_{grows}$$

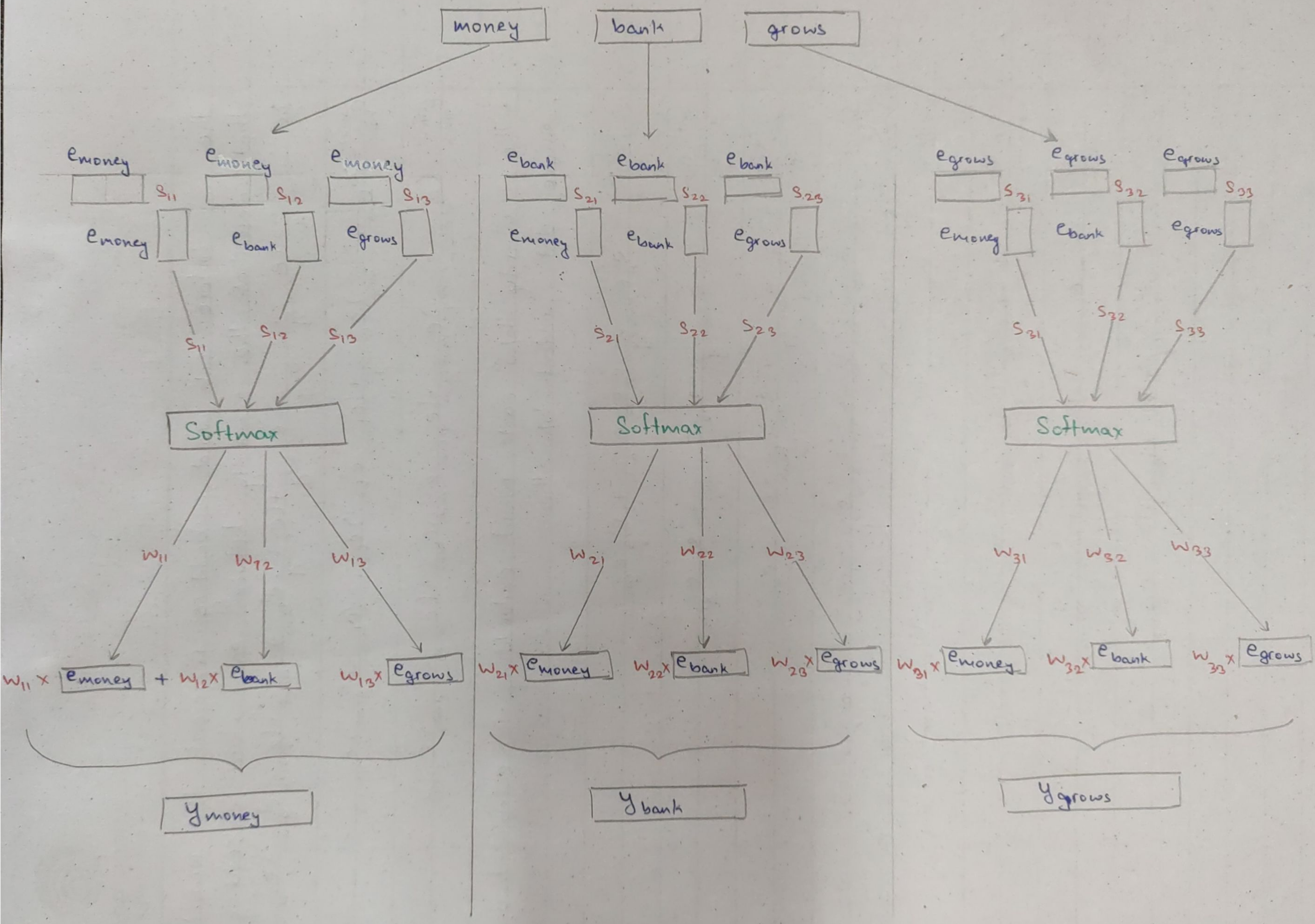
$$e^{(new)}_{grows} = S_{31} e_{money} + S_{32} e_{bank} + S_{33} e_{grows}$$

$S_{11}, S_{12}, S_{13}, S_{21}, S_{22}, S_{23}, S_{31}, S_{32}, S_{33}$ are not normalized & are normalized using a softmax function resulting in normalized weights of $W_{11}, W_{12}, W_{13}, W_{21}, W_{22}, W_{23}, W_{31}, W_{32}, W_{33}$

$\Rightarrow$ $e^{(new)}_{money} = W_{11} e_{money} + W_{12} e_{bank} + W_{13} e_{grows}$

$$e^{(new)}_{bank} = W_{21} e_{money} + W_{22} e_{bank} + W_{23} e_{grows}$$

$$e^{(new)}_{grows} = W_{31} e_{money} + W_{32} e_{bank} + W_{33} e_{grows}$$



money
bank
grows
e money
e money
e money
s11
s12
s13
e money
e bank
e grows
s11
s12
s13
Softmax
w11
w12
w13
w11 x e money + w12 x e bank
w13 x e grows
y money
e bank
e bank
e bank
s21
s22
s23
e money
e bank
e grows
s21
s22
s23
Softmax
w21
w22
w23
w21 x e money
w22 x e bank
w23 x e grows
y bank
e grows
e grows
e grows
s31
s32
s33
e money
e bank
e grows
s31
s32
s33
Softmax
w31
w32
w33
w31 x e money
w32 x e bank
w33 x e grows
y grows

Points to consider -

1) This operation is a parallel operation, i.e. calculation of new word embeddings can be done parallely thus increasing processing speed & reducing time. But this parallel processing comes at a cost of losing sequential information which is crucial in sequential data.

2) There are no learnable parameters involved, i.e. there are no learnable weights & biases during the learning process. Considering the example of machine translation the self-attention model will not focus of the dataset for training but only the current sentence. For eg, for translating from English to Hindi the sentence "Piece of cake" the model will return "केक का टुकड़ा" whereas it is possible that in the dataset it was "बहुत आसान काम". This implies that the model is generating general contextual embeddings instead of task specific contextual embeddings which is more required. To solve this we need to introduce weights in our overall process so that the model can learn.

$$\underset{3\times n}{\begin{bmatrix} e_{money} \\ e_{bank} \\ e_{grows} \end{bmatrix}} \; \underset{n\times 3}{\begin{bmatrix} e_{money} & e_{bank} & e_{grows} \end{bmatrix}} \xrightarrow{\text{dot}} \underset{3\times 3}{\begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{bmatrix}} \xrightarrow{\text{Softmax}} \underset{3\times 3}{\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}} \; \underset{3\times n}{\begin{bmatrix} e_{money} \\ e_{bank} \\ e_{grows} \end{bmatrix}} \xrightarrow{\text{dot}} \underset{3\times n}{\begin{bmatrix} y_{money} \\ y_{bank} \\ y_{grows} \end{bmatrix}}$$

This overall process is a three step process & we need to introduce learnable parameters in this process.

This whole process consists of three steps, 1 softmax operation & 2 dot product operations. Learnable parameters can be introduced in dot product operations.

Observing the diagram we can see that each word embedding vector is performing different roles at each step.

For word embedding vector e_{money} at first operation of dot product it is querying for the similarity ~~of~~ b/w it & all word embedding vectors & also returning the similarity during the dot product. In the last dot product it is being used as value to output ~~final~~ contextual word embedding vector for the word money. Same goes with other word embedding vectors.

∴, we can see that e_{money}, & others, are ~~being~~ playing 3 roles in the whole process - query, key, & value. All 3 roles are being performed by the same word embedding, e_{money} for instance, which won't be able to do it properly. Instead, it should 3 separate vectors for performing those 3 roles, q_{money} for query, k_{money} for key, & v_{money} for value.

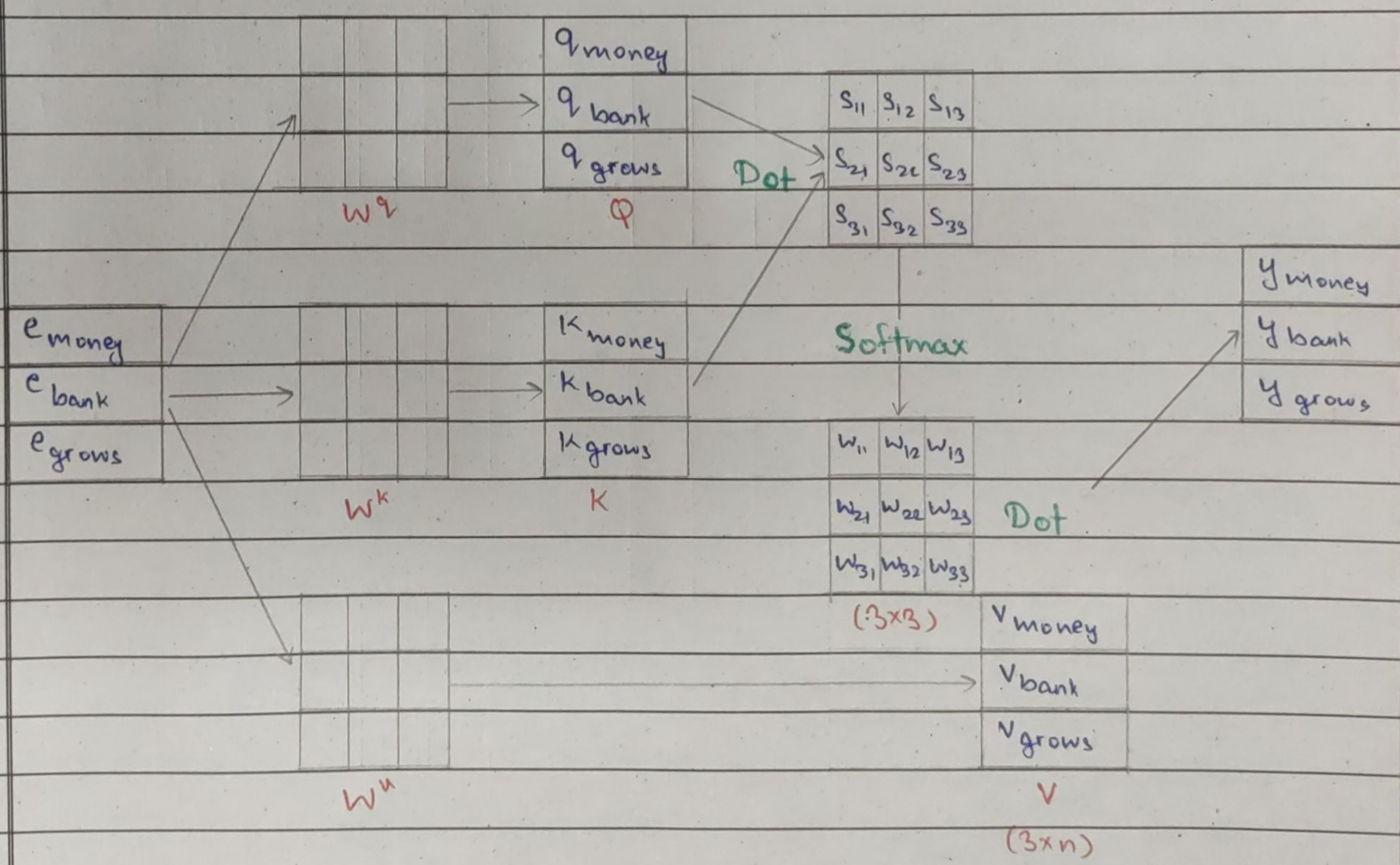
This is called separation of concerns as each concerned role has now it's specific vector for it which will improve performance.

Now, to generate 3 new vectors from another vector we can use linear transformation where we can multiply the word embedding vector with 3 different weight matrices, namely W^q for query, W^k for key, & W^v for values respectively.

The values of the matrices can be randomly initialized which will get updated to best values while training & backpropagation. And we will do this for all word embedding vectors.

So now, our model architecture can also work parallely & has learnable parameters to generate Task specific Contextual Word Embeddings.

So now, our final model architecture looks like this -



* The value of W^q, W^k, W^v will be same for all word embeddings