Self Attention

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

Calculates similarities for each word between each word and itself and all of the other words.

Each word in the prompt is converted to word embeddings. We then add positional encodings to the word embeddings. In the transformer paper each word embedding has a dimension of 512. We then stack these embeddings and multiply it by respective weights.

Consider the following sentence.

Hello world, I live in Pune.

Above sentence has 6 words, embedding dimension after adding positional embedding would be

6 * 512. We then create 3 copies of the resultant embedding and this forms Querry, Key and Value matrix. Each of these matrices have separate weights of dimension of weights would be 512 * 512. Multiplying them would result in 6* 512 dimension. The weight matrix is trainable and is initialized with glorot uniform.

Q.K^T

(6*512) *(512*6)=> (6*6)

We scale down the value to math.sqrt(512) because it improves the performance.

Finally we take the SoftMax of each row in the resultant matrix .

The matrix represents how much each word is similar to the other word explained by the value of the prob. We then multiply this matrix with Value matrix. (6*512)

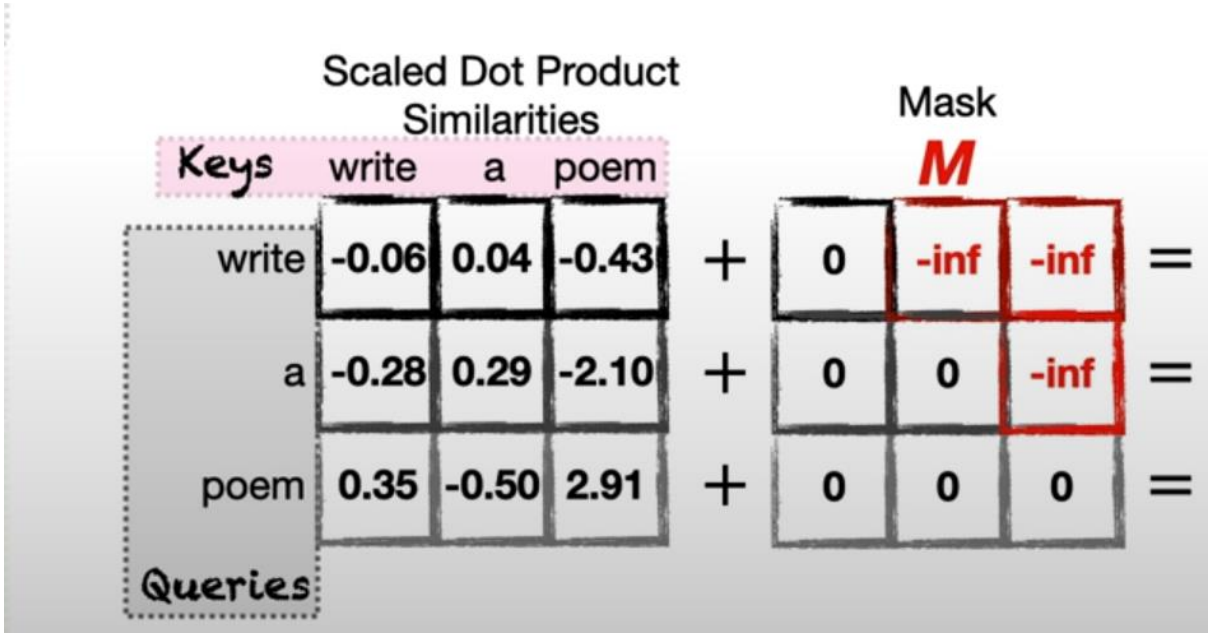
Res-> (6*6) * (6*512) => (6*512)

The percentage that comes out of softmax tells us how much influence each word have on the final encoding for any given word.

Masked Self Attention

$$MaskedAttention(Q, K, V, M) = SoftMax\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V$$

Masked Attention is usually used with decoder only models where the next i/p sequence after the words is hidden from the model and the model has to predict the next word. These are helpful in generating words.



Note: e**(-inf)=0

Creating mask

```
mask=torch.tril(torch.ones(3,3))
mask=mask==0
print(mask)
```

```
tensor([[False,  True,  True],
        [False, False,  True],
        [False, False, False]])
```