2021

DATE:

Q1) Perceptron is a simple neural network with no hidden layers. We cannot linearly seperate XOR that means when we use XOR we show the value 0 if both the bits are same and 1 if both the bits are different. It is difficult to classify these by a single line. It is not possible to introduce non-linearity while using perceptron.

Multi-layer perceptrons have hidden layers and can learn complex patterns easily. With optimazation techinques like gradient descent, updating parameters can become more easier. It can use non-linearity by using functions by ReLU, softmax etc.

2021

DATE:

(Q2) When linear layers are stacked on linear layers, they transform into one linear layer all together.

Suppose $y = a_1 x_1 + b_1$

Now use another layer $a_2 x_2 + b_2$ in x_1

$$y = a_1 (a_2 x_2 + b_2) + b_1$$

$$y = a_1 a_2 x_2 + (a_1 b_2 + b_1)$$

$$y = a' x_2 + b'$$

Gradients are calculated through the chain rule. When calculating the derivative of the activation functions, the derivative leads to be less than 1. Due to back propagation, the value gets multiplied many times which leads to the total gradient to nearly zero.

Sigmoid function gets saturated for very large values and the derivatives value are very small compared to ReLU derivatives which causes sigmoid to vanish the gradient.

2021

DATE:

Q3) When we send the sentence in the encoder, first it tokenizes and converts them all into vectors so that we could send them parallely to the encoder but the problem arises when we permutate the words in the sentence, this changes the meaning of the sentence due to the position of the words. Positional encoding will help us to rectify this problem by assigning positions to the word and adding them to the original vectors.

Sinsoidal PE	Absolute PE
↳ Uses sine and cosine functions	↳ Trainable lookup tables
↳ The stability for long context becomes weak.	↳ Encodes only absolute position
↳ It tells a model where a token is but not how far from others	↳ Difficulty with longer sequences

2021

DATE:

RoPE uses rotation of the vectors which preserves the vector length and geometry. It also captures the meaning of the sentence even when the context size is long. It generalizes to longer sequences.

Q4) Query : It is essentially asking if there is anything useful for the token.

Key : It actually reveals whatever the token can offer.

Value : It is the actual information that the token represents.

Attention Scores are essentially the scores that tell how much is the key is relevant to Query. → $Q \cdot K^T$ → attention score matrix

2021

DATE:

We scale attention scores by $\sqrt{d_k}$ because when we use softmax, it converts the values to between 0 and 1. So when the scores are too large it assigns the highest value to that score and make the other scores nearly equal to zero. When the scores are very small or, it makes the values to be very less and similar which will affect the training of the model in understanding the relevance of key for the query. To solve this problem, we scale the values of $Q \cdot K^T$ by $\sqrt{d_k}$ so that it normalizes and doesn not cause any problem with softmax.

The Query and key are from the same token. So the dot product will be highest compared to other scores. So the diagonal values become highest.

2021

DATE:

(Q5) We split Attention into multiple heads to capture more relationships between the tokens. One head may find for the Semantic meaning, another one for patterns, another one for any long-range dependencies.

d_{model} → main embedding size of the model

h → number of attention heads

d_{head} → dimension of the attention heads for the Query, Key and Value used for the attention mechanism.

2021

DATE:

Q1)

(a) $d\_head = \frac{d\_model}{h} = \frac{768}{12} = 64$

(b) Projection matrices for Q, K, V are of the size ~~(dmodel, dmodel)~~ (dmodel, d model)

Q, K, V → size → (768 × 768)

Total parameters = $3 \times 768 \times 768 = 17,69,472$

Q2) Softmax = $\frac{e^i}{\sum_{i=1}^{n} e^i}$

↓

[2.0 1.0 0.0]

↓

$\frac{e^2}{e^2+e+e^0}$ $\frac{e^1}{e^2+e+e^0}$ $+\frac{e^0}{e^2+e+e^0}$

↓

[0.665 0.245 0.090]