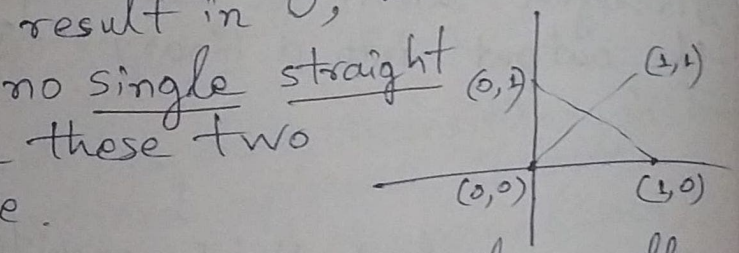
Architecting Intelligence

Assignment-2

Q.1) A single-layer perceptron is a linear classifier. It can only solve problems where the classes are linearly separable. In the XOR problem the inputs (0,0) and (1,1) result in 0, while (0,1) and (1,0) result in 1. There is no single straight line that can separate these two groups on a 2D-plane.



(diagonally opposite class)

By adding a hidden layer with non-linear activation functions (multi-layer perceptron (MLP)), the network can warp the input space. This allows the model to create a non-linear decision boundary, effectively "folding" the space so that classes become separable.

Q.2) Stacked Linear Layers:- Mathematically, the composition of two linear function is itself a linear function. If $f(x) = W_1 x$ and $g(x) = W_2 x$, then $g(f(x)) = W_2(W_1 x)$ $= (W_2 W_1)x$. This collapses into a single matrix W_{new}, meaning no matter how many layers you add, the model remains a simple linear transformation unless a non-linear activation (like ReLU) is placed between them.

Vanishing Gradients:- In deep networks, gradients are calculated via the chain rule. If the derivative of the activation function is small (e.g.:- sigmoid's max derivative is 0.25), multiplying many of these small numbers together causes the gradient to "shrink" or vanish as it flows back to early layers, preventing learning.

ReLU v/s Sigmoid:- ReLU solves this because its derivative is either 0 (for $x<0$) or 1 (for $x>0$). A gradient of 1 does not shrink when multiplied through many layers, allowing the signal to reach the earliest layers of the network.

Q.3) Necessity:- Transformers use self-attention, which is "permutation invariant". Without PE, the model would treat the sentence "The dog bit the man" exactly the same as "The man bit the dog". PE injects info about the order of tokens.

Sinusoidal v/s Absolute PE:- Absolute PE learns a unique vector for each specific position (POS 1, POS 2 etc)

Sinusoidal PE uses fixed sine and cosine functions. It allows the model to attend to relative positions more easily and can potentially generalize to sequence lengths longer than those seen during training.

RoPE (Rotary Embeddings):- RoPE encodes position by rotating the Query and Key vectors in a complex space. It naturally captures the relative distance between tokens, which is more robust for very long contexts compared to absolute positions.

Q.4) Query (Q) :- What I am looking for.

Key (K) :- What I have to offer.

Value (V) :- The actual information content.

Scaling by $\sqrt{d_k}$:- For large dimensions, the dot product of Q and K can grow very large in magnitude. This pushes the softmax function into regions where the gradient is extremely small (the "saturation" region). Scaling keeps the variance of the dot products near 1, ensuring stable gradients.

Diagonal Values :- Diagonal values represent a token attending to itself. Naturally, a token's Query is most similar to its own key, leading to the highest attention scores.

Q.5) MHA allows the model to attend to different types of relationships simultaneously. One head might focus on grammar, another on semantic meaning, and another on historical references within the text.

→ d_model :- The total dimensionality of hidden states (eg :- 512)

→ h :- The number of attention heads (e.g. - 8)

→ d_head :- The dimension of each individual head. Calculated as $d\_head = \frac{d\_model}{h}$.

Q.6) Greedy Decoding :- Picks the single most likely token at every step. It is suboptimal because a high-probability word now might lead to a dead-end or a very low-probability sequence later. It lacks "foresight."

Beam Search :- Keeps track of the top-k most likely sequences (beams) at each step. This allows the model to sacrifice a high probability word in the short term to achieve a higher total probability for the entire sentence.

Example of Greedy Failure :-

→ Input :- "The weather is---"

→ Greedy :- Might pick "nice" (high prob), leading to "The weather is nice today."

→ Beam search :- Might pick "going" (lower prob than 'nice'), eventually leading to the much more complex/accurate "The weather is going to be stormy later this evening."

SOLVING & CODING

Q.1) $d\_model = 768$; $h = 12$

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

(b) Calculation:- $768 \times 768 = 589,824$ parameters per matrix.
Total Q, K, V: Since we have three distinct matrices:
$3 \times (768 \times 768) = 3 \times 589,824 = 1,769,472$ parameters.

Q.2) To compute the Softmax for the scores $[2.0, 1.0, 0.0]$ we follow three steps:

i) Compute the exponentials:-
$e^{2.0} \approx 7.389$; $e^{1.0} \approx 2.718$; $e^{0.0} = 1.000$

ii) Compute the sum of exponentials:-
$\Sigma = 7.389 + 2.718 + 1.000 = 11.107$

iii) Divide each exponential by the sum:-
$P_1 = \frac{7.389}{11.107} \approx 0.665$; $P_2 = \frac{2.718}{11.107} \approx 0.245$;
$P_3 = \frac{1.000}{11.107} \approx 0.090$
Final Softmax vector: $[0.665, 0.245, 0.090]$