# Assignment - 2

Date: / / Page No.:

1. A perceptron computes a linear decision boundary of the form -

$$y = \text{sign}(w_1 x_1 + w_2 x_2 + b)$$

thus it can seperate data that is linearly seperable
Now,

Truth table of XOR -

| $x_1$ | $x_2$ | XOR |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

XOR is not linearly seperable, and a single perceptron can only learn linear boundaries

Hence, a perceptron fails on XOR.

Now,
When we introduced multi-layer perceptions, hidden layers with non-linear activation functions were added. These layers transform the input into a space where XOR becomes linearly seperable, allowing the network to learn non-linear decision boundaries.

Date: / / Page No.:

2 A linear layer computes

$$y = Wx + b$$

stack two linear layers:

this is still of form:

$$y = W_2(W_1 x + b_1) + b_2 = (W_2 W_1)x + (W_2 b_1 + b_2)$$

$$y = Wx + b$$

thus linear layers stacked on linear layers remain linear

- During backpropagation, gradients are computed using the chain rule:

$$\frac{\partial L}{\partial w} = \prod (\text{activation derivatives})$$

If activation derivatives are less than 1 (eg - sigmoid) then ⇒ multiplying many small numbers → exponentially smaller gradients

⇒ Earlier layers receive almost zero updates

- Sigmoid activation:

$$\sigma'(x) = \sigma(x)\left(1 - \sigma(x)\right) \leq 0.25$$

⇒ derivative is always < 1

⇒ saturates at 0 or 1 → gradients ≈ 0

Date: / / Page No.:

Now, ReLU activation:

$$f(x) = \max(0, x), \quad f'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

thus, derivative = 1 for active neuron and no saturation for positive inputs

Hence gradients flow better through deep networks with ReLU.

3) Transformers use self-attention which is permutation-invariant. Without positional information, the model cannot distinguish: ex - "dog bites man" vs "man bites dog".

thus positional ~~and~~ encoding is needed to inject order of tokens. And let attention know which token comes before or after which

- Aspect. Sinusoidal PE Absolute PE

| | Aspect | Sinusoidal PE | Absolute PE |
|---|---|---|---|
| 1. | Type | fixed, deterministic | Trainable parameters |
| 2. | Formula | Uses sin & cos of different frequencies | Lookup table of embeddings |
| 3. | Generalisation | Can extrapolate to longer sequences | Limited to trained length |
| 4. | Parameters | No extra parameters | Add parameters |
| 5. | Relative info | Implicit | Weak |

Date: / / Page No.:

RoPE rotates query and key vectors based on positions:

$$q' = R(\theta_{pos})\, q \quad , \quad k' = R(\theta_{pos})\, k.$$

this gives:

1) Relative positional encoding naturally
2) Better extrapolation to long sequences
3) Stable dot products

4. Query (Q): what the current token is looking for
Key (K): what each token offers
Value (V): the info. to be passed if a key matches the query

Attention computes:

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

Now,

- Dot products $Q \cdot K$ grows in magnitude as dimension $d_K$ increases
- Large values → softmax becomes too sharp
- Gradients become very small

Scaling by $\sqrt{d_K}$:

$\frac{QK^T}{\sqrt{d_K}}$ keeps scores in a stable range, preventing saturation and improving training

Now,
- Diagonal entries correspond to a token attending to itself
- A token's query and key are derived from the same embedding
- So, similarity $Q_i \cdot K_i$ is usually maximal

5. Multi-head attention allows the model to look at the same sequence from different perspectives at the same time.
Each head can learn to focus on:
- syntax (eg - subject - verb relations)
- Semantics (meaning - related tokens)
- Long-range vs short-range dependencies

This increases expressive power without increasing total computation too much.

- $d_{model}$ - Total embedding dimension of the model
$h$ - number of attention heads
$d_{head}$ - Dimension of each head

$$d_{head} = \frac{d_{model}}{h}$$

6. Greedy decoding selects the most probable token at each step:

$$y_t = \arg\max P(y_t \mid y_{<t}, x)$$

It makes locally optimal choices, ignores future consequences and cannot revise earlier decisions

So it may miss the globally most probable sequence

- Beam seach keeps the top-K (beam width) partial sequences, Expands all candidates at each step. Also, selects the sequences with highest total log-probability

example where greedy decoding fails

Assume probabilities

Time step 1

- "I" : 0.6
- "You" : 0.4

Greedy picks "I"

Time step 2

After "I" → "am" : 0.4
After "I" → "like" : 0.6

Greedy picks "like"

sequence prob : $0.6 \times 0.6 = 0.36$

But

"You" → "are" : 0.9

sequence prob : $0.4 \times 0.9 = 0.36$ (tie here) more clearly:

"I" → "like": 0.6 × 0.5 = 0.30

"You" → "are": 0.4 × 0.9 = 0.36

Beam search (k=2) keeps both prefixes and selects "You are", which greedy misses