

# SNGULAR

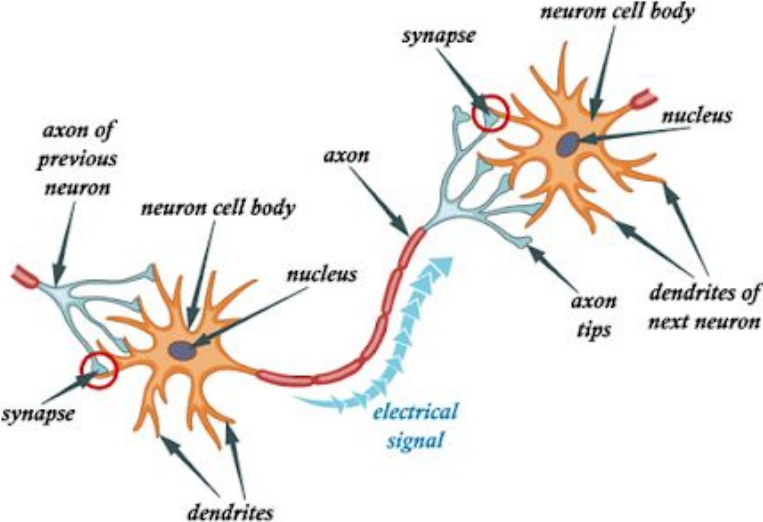
Transformer Model And Distillation.  
A Theoretical Perspective.

Neuronal network from theory to garden model deployment.

## Contents:

---

- Multilayer Perceptron
- Transformer Model:
  - Input Embedding and Positional Encoding
  - Multi-Head Self-Attention
  - Add & Layer Norm
  - Position-wise Feedforward Network
  - Stacking Layers
  - Output Layer
  - Loss Function
- LLM Distillation
- Hugging Face.
- Vertex AI



## Mcculloch & Pitts Neuron (1943)

Let's define:

- **Inputs:**  $x_1, x_2, \dots, x_n \in \{0, 1\}$  (binary)
- **Weights:**  $w_1, w_2, \dots, w_n \in \mathbb{R}$  (real numbers)
- **Threshold:**  $\theta \in \mathbb{R}$
- **Output:**  $y \in \{0, 1\}$

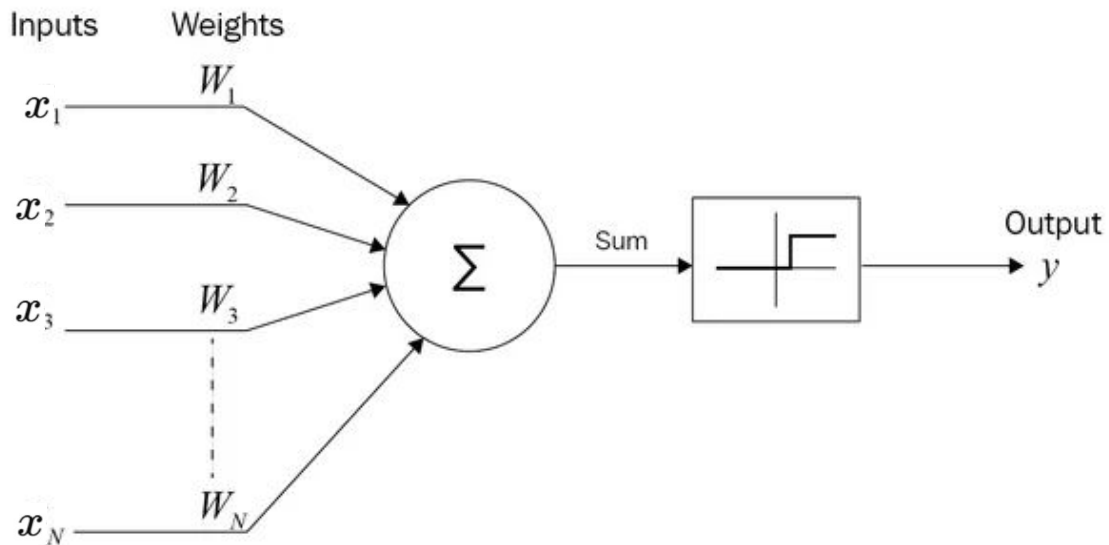
The neuron computes the weighted sum of inputs:

$$z = \mathbf{w}^T \mathbf{x} = \sum_{i=1}^n w_i x_i$$

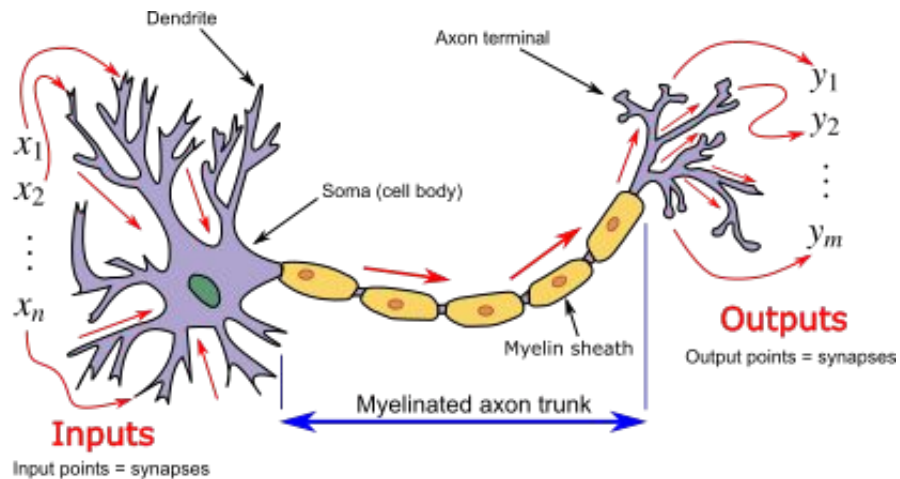
Then it applies a **step activation function**:

$$y = \begin{cases} 1 & \text{if } z \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

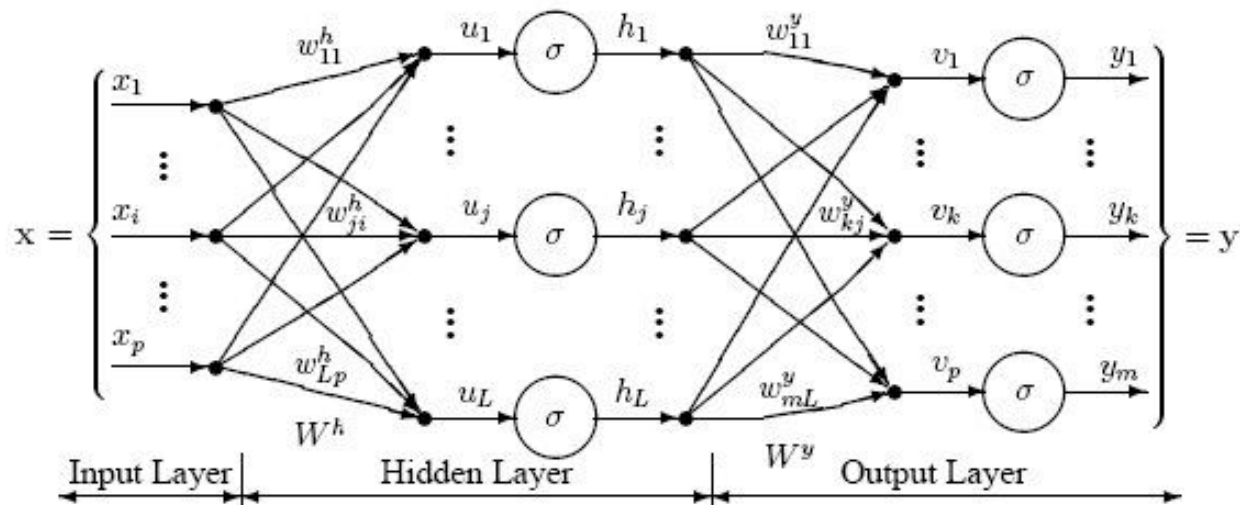
## Mcculloch & Pitts Neuron (1943)



## Biological neuron v/s Artificial neuron



# Multilayer Perceptron - Frank Rosenblatt (1958)

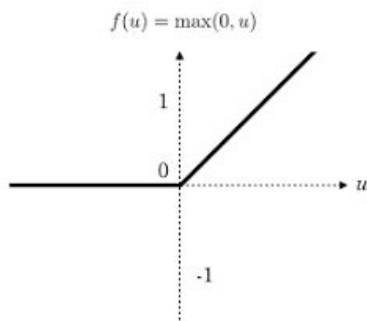


# Input Layer & Hidden Layer

$$x = [0.5 \quad 0.1 \quad 0.8]$$

$$W^h = \begin{bmatrix} w_{11}^h & w_{12}^h & w_{13}^h & w_{14}^h \\ w_{21}^h & w_{22}^h & w_{23}^h & w_{24}^h \\ w_{31}^h & w_{32}^h & w_{33}^h & w_{34}^h \end{bmatrix} = \begin{bmatrix} 0.2 & 0.4 & 0.1 & 0.9 \\ 0.7 & 0.3 & 0.8 & 0.2 \\ 0.5 & 0.6 & 0.4 & 0.7 \end{bmatrix}$$

$$b^h = [0.1 \quad 0.2 \quad 0.15 \quad 0.25]$$



$$u = x \cdot W^h + b^h$$

$$u = [0.5 \quad 0.1 \quad 0.8] \cdot \begin{bmatrix} 0.2 & 0.4 & 0.1 & 0.9 \\ 0.7 & 0.3 & 0.8 & 0.2 \\ 0.5 & 0.6 & 0.4 & 0.7 \end{bmatrix} + [0.1 \quad 0.2 \quad 0.15 \quad 0.25]$$

$$u = [(0.5 \cdot 0.2 + 0.1 \cdot 0.7 + 0.8 \cdot 0.5) \quad (0.5 \cdot 0.4 + 0.1 \cdot 0.3 + 0.8 \cdot 0.6) \quad \dots] + [0.1 \quad 0.2 \quad 0.15 \quad 0.25]$$

$$u = [0.57 \quad 0.71 \quad 0.45 \quad 1.03] + [0.1 \quad 0.2 \quad 0.15 \quad 0.25]$$

$$u = [0.67 \quad 0.91 \quad 0.60 \quad 1.28]$$

$$h = \sigma(u) = \text{ReLU}([0.67 \quad 0.91 \quad 0.60 \quad 1.28])$$

$$h = [0.67 \quad 0.91 \quad 0.60 \quad 1.28]$$



## Output Layer

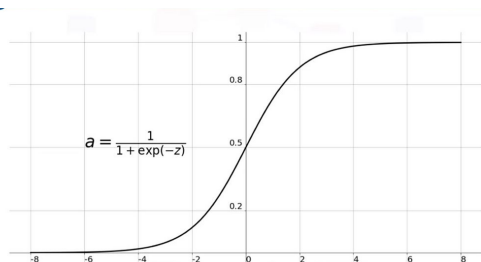
$$v = h \cdot W^y + b^y$$

$$v = \begin{bmatrix} 0.67 & 0.91 & 0.60 & 1.28 \end{bmatrix} \cdot \begin{bmatrix} 0.6 & 0.1 \\ 0.5 & 0.3 \\ 0.2 & 0.8 \\ 0.9 & 0.4 \end{bmatrix} + \begin{bmatrix} 0.3 & 0.1 \end{bmatrix}$$

$$v = [(0.67 \cdot 0.6 + \dots) \quad (0.67 \cdot 0.1 + \dots)] + \begin{bmatrix} 0.3 & 0.1 \end{bmatrix}$$

$$v = \begin{bmatrix} 2.127 & 1.332 \end{bmatrix} + \begin{bmatrix} 0.3 & 0.1 \end{bmatrix}$$

$$v = \begin{bmatrix} 2.427 & 1.432 \end{bmatrix}$$



$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

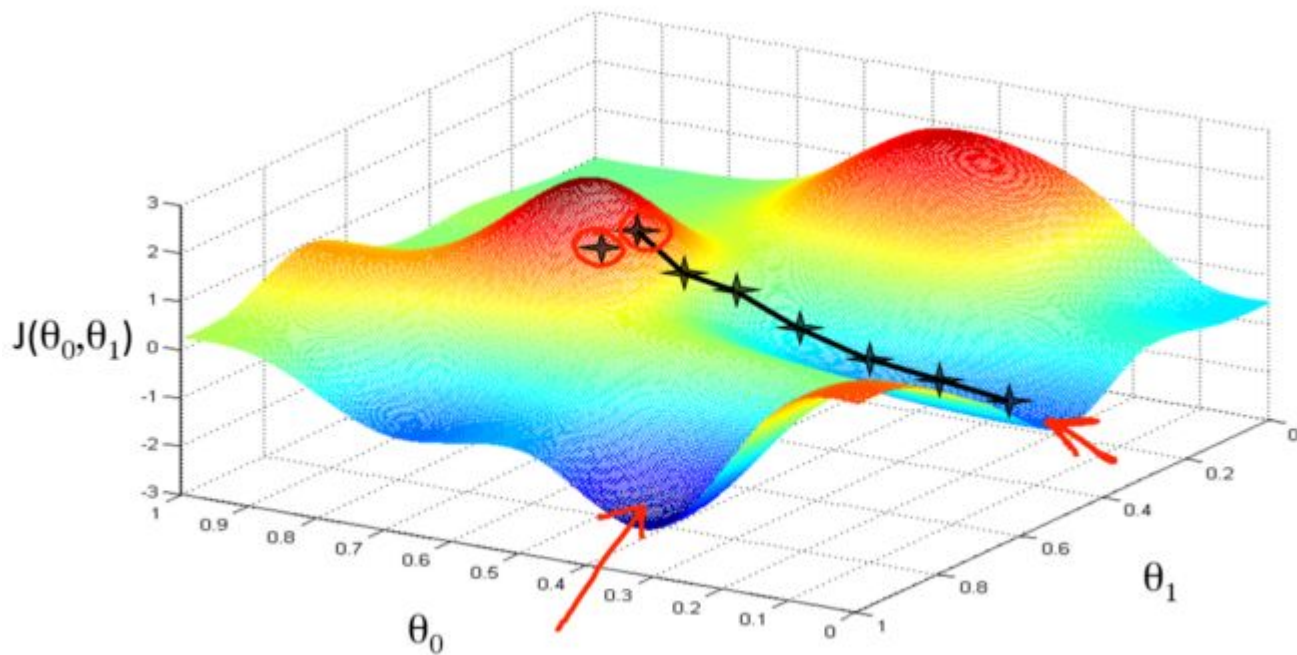
$$y = \sigma(v) = \text{sigmoid}(\begin{bmatrix} 2.427 & 1.432 \end{bmatrix})$$

$$y = \left[ \frac{1}{1 + e^{-2.427}} \quad \frac{1}{1 + e^{-1.432}} \right]$$

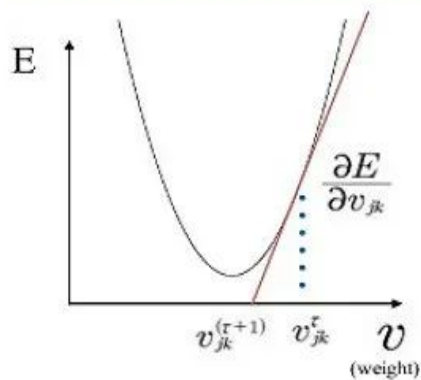
$$y = \begin{bmatrix} 0.919 & 0.807 \end{bmatrix}$$

## Back Propagation - David Rumelhart (1986)

$$E = \sum_{k=1}^m (y_{k,\text{true}} - y_{k,\text{pred}})^2$$



## Back Propagation - Loss Function



$$v_{jk}^{(\tau+1)} = v_{jk}^{\tau} + \Delta v_{jk}$$

$$\Delta v_{jk} = -\eta \frac{\partial E}{\partial v_{jk}}$$

$v_{jk}^{(\tau+1)}$  new weight

$v_{jk}^{\tau}$  current weight

$\eta$  learning rate

$E$  Error Function

# Transformer

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

is all you need

## Input Embedding & Positional Encoding

Let:

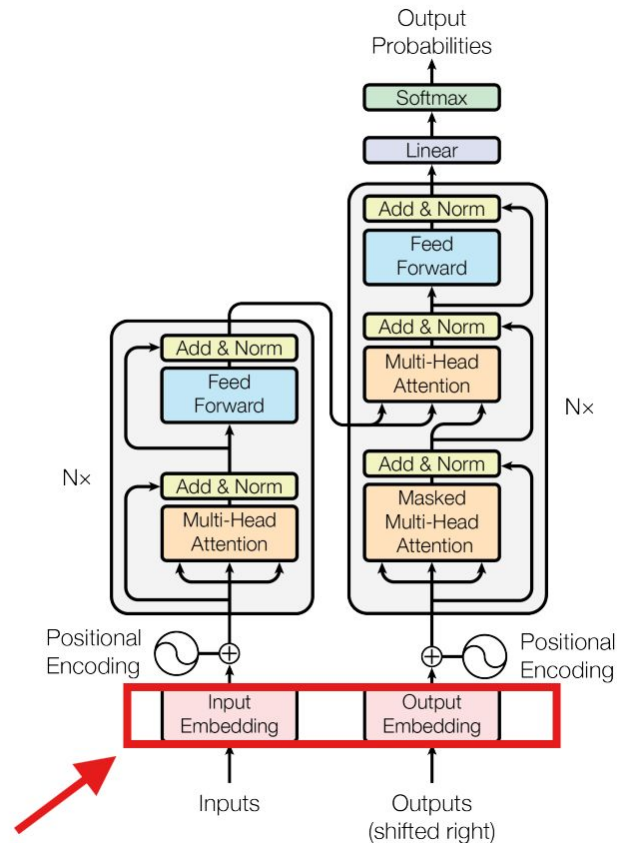
- Vocabulary size =  $V$
- Embedding dimension =  $d_{\text{model}}$
- Input sequence =  $x = [x_1, x_2, \dots, x_n]$

Then:

$$\text{Embedding}(x_i) = E[x_i] \in \mathbb{R}^{d_{\text{model}}}$$

Where  $E \in \mathbb{R}^{V \times d_{\text{model}}}$  is the embedding matrix.

"I"	→	[0.1, 0.2, 0.3, 0.4]
"am"	→	[0.5, 0.6, 0.7, 0.8]
"GPT"	→	[0.9, 1.0, 1.1, 1.2]



# Positional Encoding(Sinusoidal)

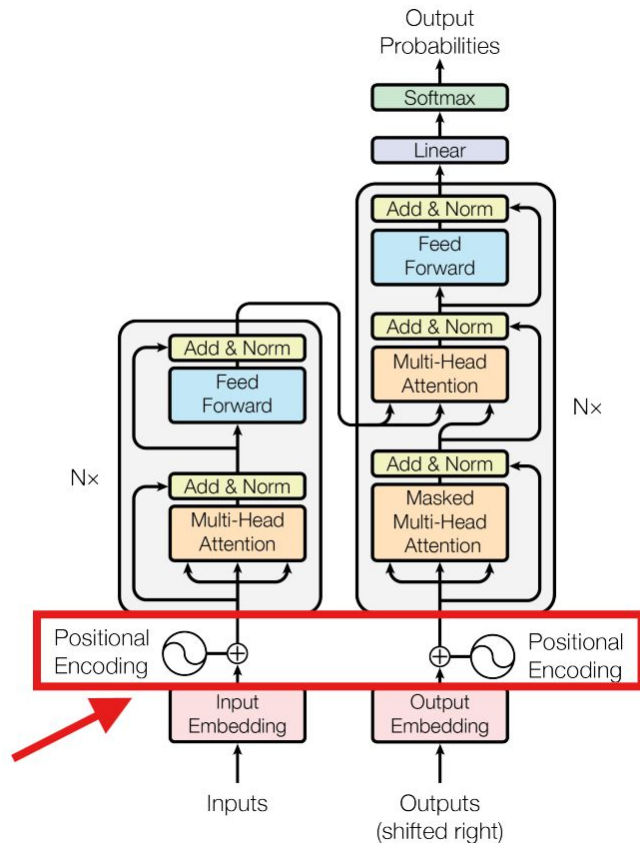
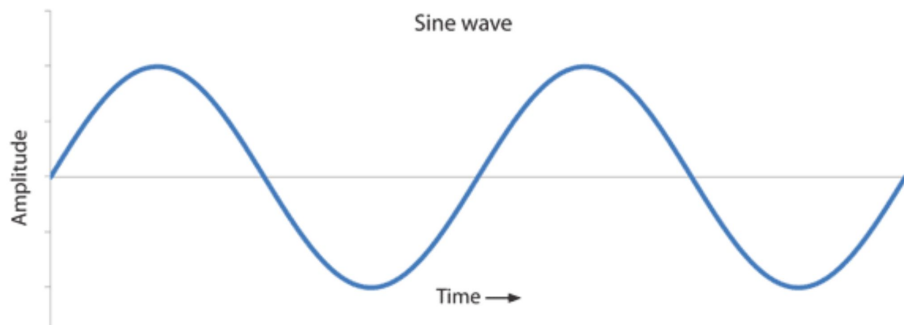
For position  $pos$  and dimension  $i$ :

$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$PE_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

Then, final input:

$$\text{Input} = \text{Embedding}(x) + \text{PositionalEncoding}$$



# Multi-Head Self Attention

## Scaled Dot-Product

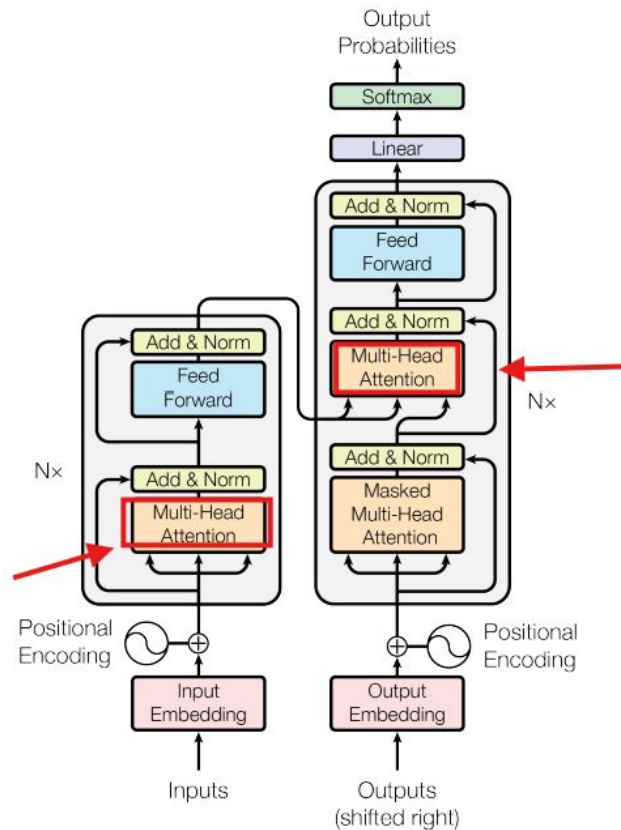
For a given input matrix  $X \in \mathbb{R}^{n \times d_{\text{model}}}$ , define:

- $Q = XW^Q, K = XW^K, V = XW^V$
- $W^Q, W^K, W^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$

Attention:

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

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}$$



# Multi-Head Self Attention

## Multi-Head Attention

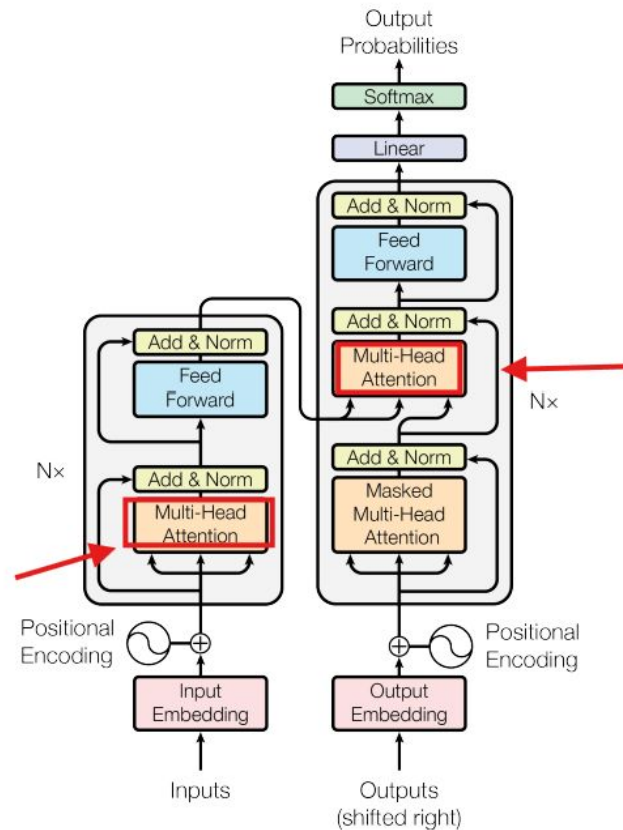
Instead of one attention, multiple heads (say  $h$ ) are used in parallel:

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i)$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

Where:

- $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$





# Masked Multi-Head Attention

The **raw attention scores** (scaled dot product):

$$A = \frac{QK^{\top}}{\sqrt{d_k}} \quad \text{where } A \in \mathbb{R}^{t \times t}$$

We apply a **mask matrix**  $M \in \mathbb{R}^{t \times t}$ , usually:

$$M_{ij} = \begin{cases} 0 & \text{if } j \leq i \quad (\text{can attend}) \\ -\infty & \text{if } j > i \quad (\text{mask future}) \end{cases}$$

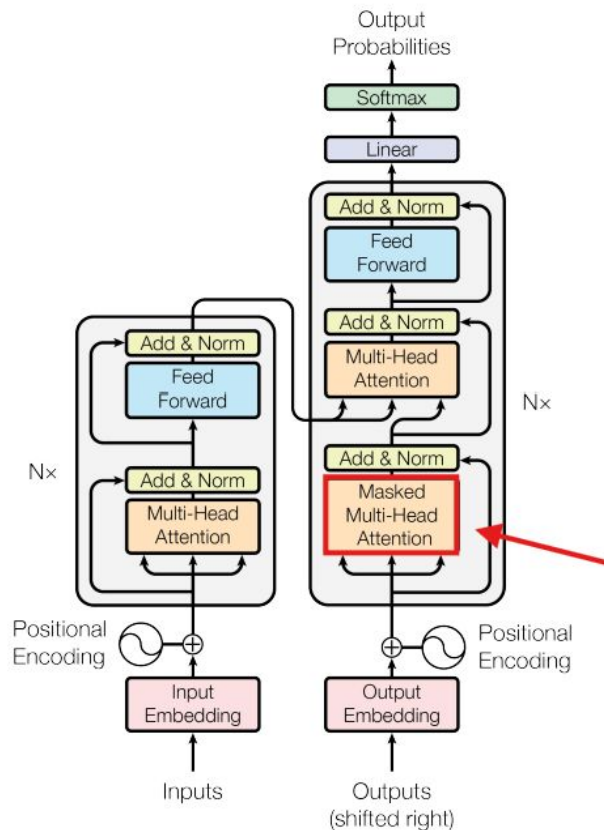
So we modify the scores as:

$$A' = A + M$$

$$\alpha = \text{softmax}(A') \in \mathbb{R}^{t \times t}$$

Then output of attention:

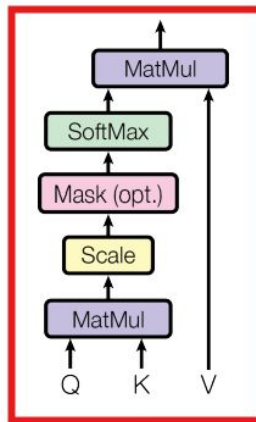
$$\text{Attention}(Q, K, V) = \alpha V$$



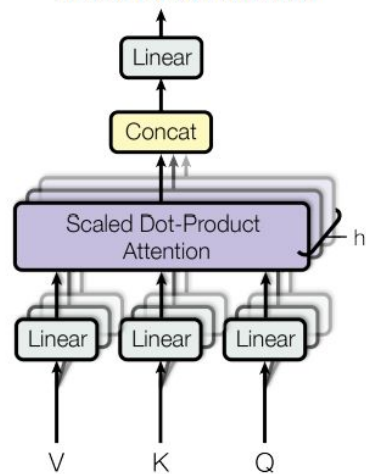
# Masked Multi-Head Attention

$$M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Scaled Dot-Product Attention



Multi-Head Attention

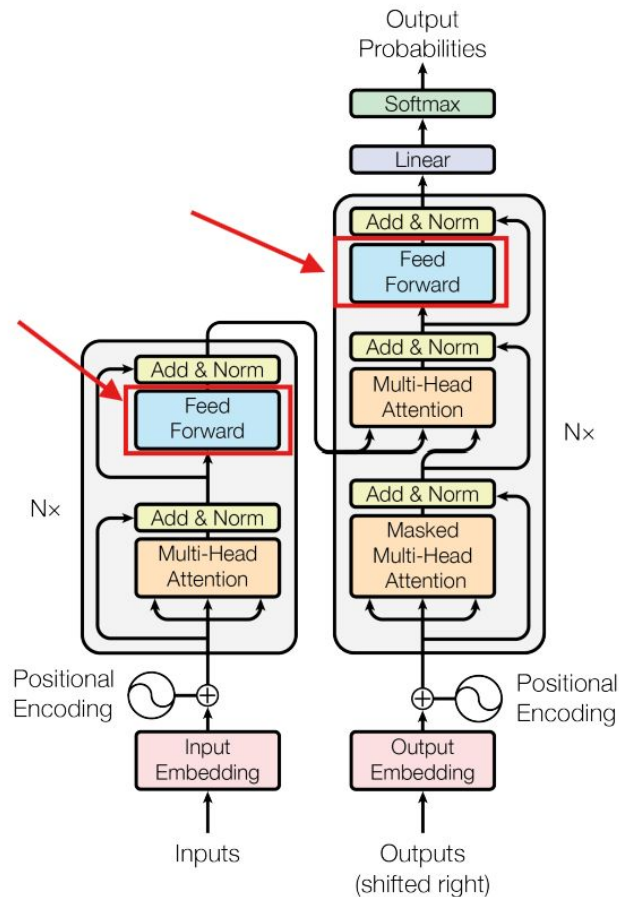
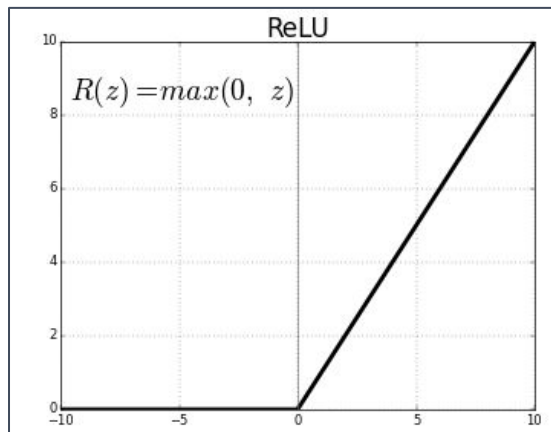


# Masked Multi-Head Attention

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

Where:

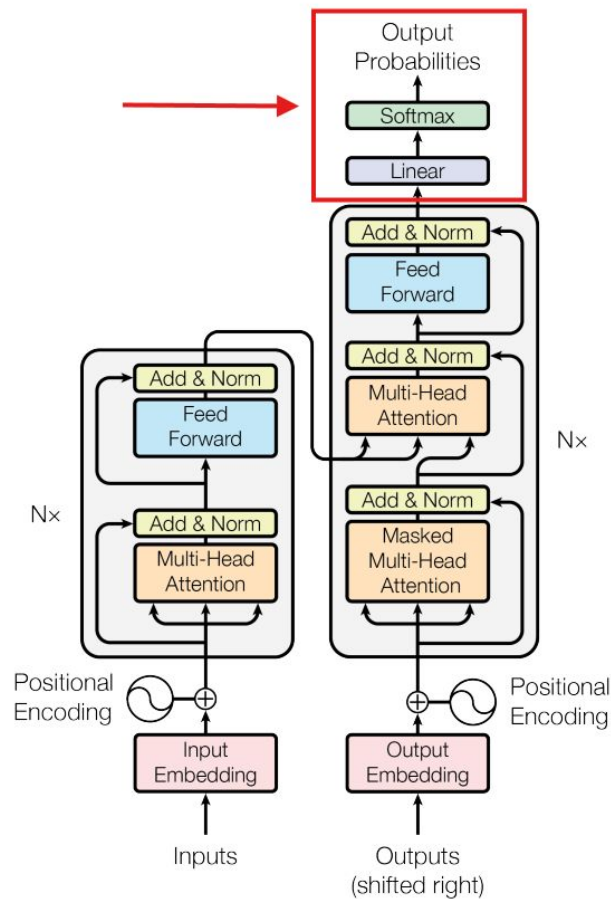
- $W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$
- $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$



## Output Layer - Linear & Softmax

- A final linear layer  $W \in \mathbb{R}^{d_{\text{model}} \times V}$
- Softmax over vocabulary

$$P(y_t|x) = \text{softmax}(h_t W^T + b)$$




## Output Layer - Linear & Softmax

Hidden vector  $\mathbf{h}$ :

$$\mathbf{h} = [1.0, 2.0, -1.0, 0.5]$$

Output weight matrix  $W_{\text{out}}$ :


$$W_{\text{out}} = \begin{bmatrix} 0.1 & 0.3 & -0.2 \\ 0.4 & -0.5 & 0.6 \\ -0.3 & 0.2 & 0.1 \\ 0.7 & 0.0 & -0.4 \end{bmatrix}$$

Bias vector:

$$\mathbf{b} = [0.2, -0.1, 0.5]$$

$$\mathbf{h} \cdot W_{\text{out}} = [1.55, -0.9, 0.7]$$

$$\text{logits} = [1.55 + 0.2, -0.9 - 0.1, 0.7 + 0.5] = [1.75, -1.0, 1.2]$$

$$\text{softmax}([1.75, -1.0, 1.2]) = \left[ \frac{e^{1.75}}{Z}, \frac{e^{-1.0}}{Z}, \frac{e^{1.2}}{Z} \right] \approx [0.61, 0.04, 0.35]$$

- Token 0: **61%**
- Token 1: 4%
- Token 2: 35%

## Learning Mechanism - Cross Entropy Loss Function

$$\text{Loss} = - \sum_{i=1}^C y_i \log(\hat{y}_i)$$

$$\hat{y} = [0.61, 0.04, 0.35]$$

$$y = [0, 0, 1]$$

Since only one  $y_i = 1$  (at index 2), the formula simplifies to:

$$\text{Loss} = -\log(\hat{y}_2) = -\log(0.35)$$

$$\text{Loss} \approx -\log(0.35) \approx -(-1.05) = 1.05$$

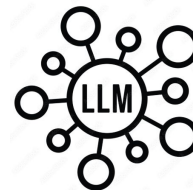
The **cross-entropy loss** for this prediction is:

1.05

# LLM Distillation



Teacher Model ( $M_T$ )



Student Model ( $M_S$ )

# Why Distilled Models?

## Faster Inference.

- Fewer parameters therefore **faster computations**.
- Lower latency and **better user experience**.

## Lower Cost.

- Reduce cloud compute **costs**.
- **Lower hardware** requirements.
- Especially **valuable** for **startups**

## When to Use Distilled Models?

- **Speed, cost, or deployment** constraints are a **concern**.
- You're building **production apps** (**chatbots, summarizers, QA systems**).



# LLM Distillation

$$P_S \ll P_T$$

Student's parameters

Teacher's parameters

$$\text{Compression Ratio} = \frac{P_T}{P_S}$$

**The choice of** the student model's architecture, and therefore its parameter count  **$P_S$** , is a critical design decision that balances **performance** with computational **efficiency**.

## LLM Distillation - Distillation Loss

### Kullback-Leibler Divergence

$$L_{\text{distill}} = D_{KL}(p_T || p_S) = \sum_i p_T(x_i) \log \left( \frac{p_T(x_i)}{p_S(x_i)} \right)$$

Where:

- $p_T$  is the probability distribution of the teacher's soft targets.
- $p_S$  is the probability distribution of the student's outputs.
- $x_i$  represents each possible output token.

## Calculation of Parameters

$$P_{\text{Total}} \approx \underbrace{V \cdot d}_{\text{Embedding Layer}} + N \cdot \left( \underbrace{4 \cdot d^2}_{\text{Attention Weights}} + \underbrace{8 \cdot d^2}_{\text{Feedforward}} \right) + \underbrace{d \cdot V}_{\text{Output Layer}}$$

Where:



- $V$ : Vocabulary size
- $d$ : Hidden size (a.k.a.  $d_{\text{model}}$ )
- $N$ : Number of layers (Transformer blocks)
- $4d^2$ : Self-attention (Q, K, V, output) projections
- $8d^2$ : FFN (typically with expansion factor 4)
- $d \cdot V$ : Output projection to vocab

# Use Case

**01** Select Hugging Face Distilled Model

**02** Navigate Through Model Garden

**03** Deploy Distilled Models

**04** Query Deployed Model



## Bibliography

- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.
- Lin, J. W. (2017). Artificial neural network related to biological neuron network: a review. *Advanced Studies in Medical Sciences*, 5(1), 55-62.
- Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.
- Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.
- Lin, Z., Feng, M., Santos, C. N. D., Yu, M., Xiang, B., Zhou, B., & Bengio, Y. (2017). A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*.

## Next Talk

---

- Train our own distilled LLM
- What fine tuning is and types:
  - **Full Fine-Tuning**
  - **LoRA**
  - **QLoRA**
- Dataset preparation
  - **Vectorization**
  - **K-Means**
- Use case - Hugging Face model creation

## Questions

---

Thanks for your attention!!!