SNGULAR

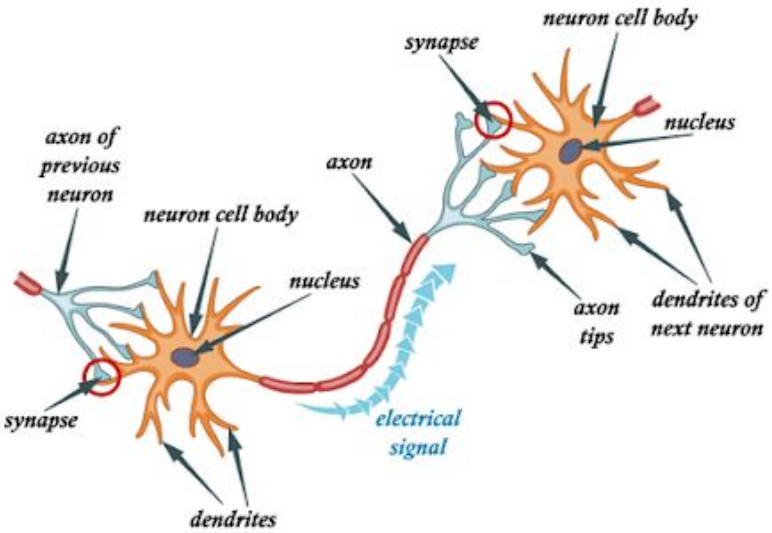
Transformer Model And Distillation.
A Theoretical Perspective.

Neuronal network from theory to garden model deployment.

SNGULAR

Contents:

SNGULAR


neuron cell body
synapse
nucleus
axon of
previous
neuron
axon
neuron cell body
nucleus
axon
tips
dendrites of
next neuron
synapse
electrical
signal
dendrites

SNGULAR

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:

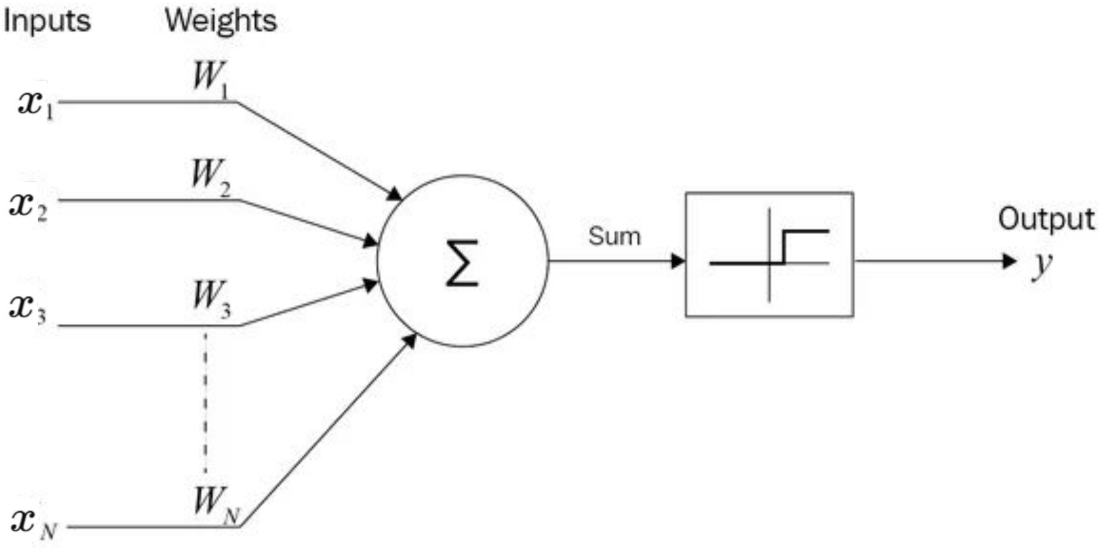
$$z = \mathbf{w}^T \mathbf{x} = \sum_{i=1}^{n} w_i x_i$$

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

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

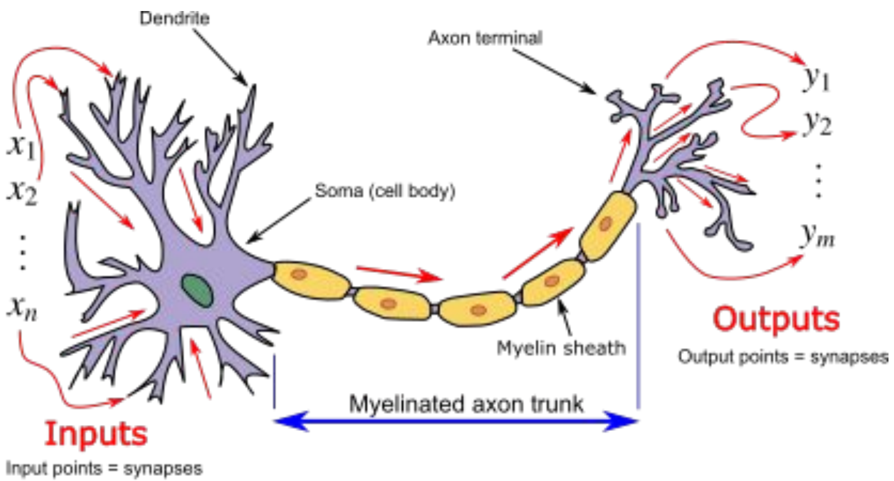
SNGULAR

Mcculloch & Pitts Neuron (1943)



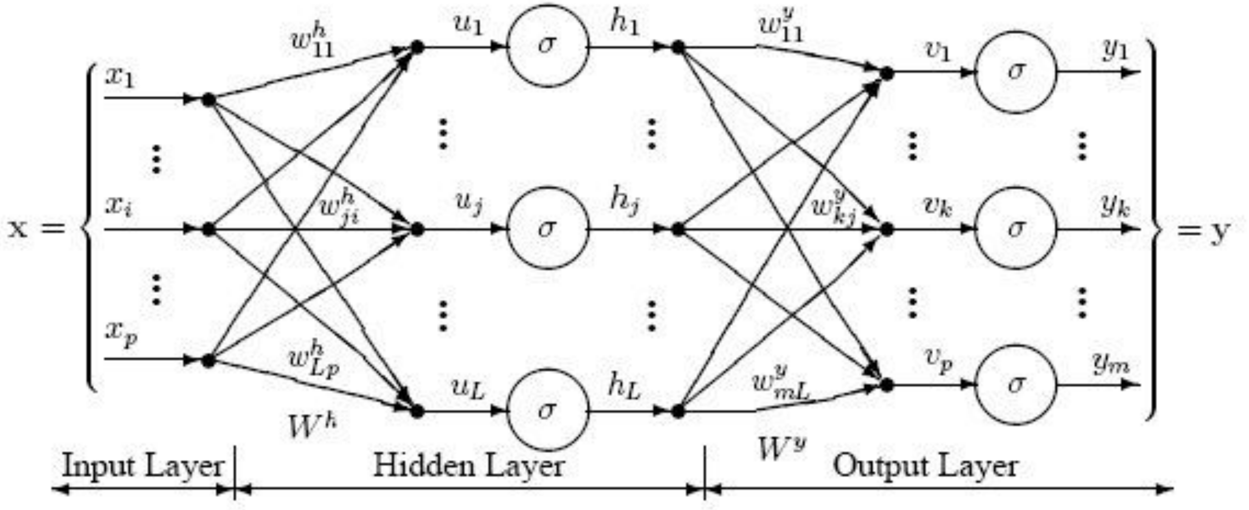
SNGULAR

Biological neuron v/s Artificial neuron



SNGULAR

Multilayer Perceptron - Frank Rosenblatt (1958)



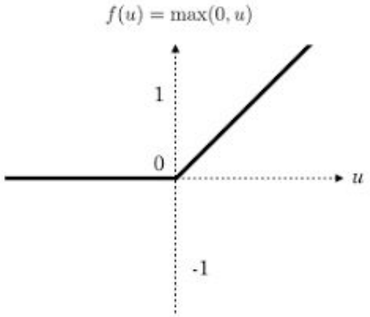
SNGULAR

Input Layer & Hidden Layer

$$x = \begin{bmatrix}0.5 & 0.1 & 0.8\end{bmatrix}$$

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

$$b^h = \begin{bmatrix}0.1 & 0.2 & 0.15 & 0.25\end{bmatrix}$$



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

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

$$u = \begin{bmatrix}(0.5 \cdot 0.2 + 0.1 \cdot 0.7 + 0.8 \cdot 0.5) & (0.5 \cdot 0.4 + 0.1 \cdot 0.3 + 0.8 \cdot 0.6) & \ldots\end{bmatrix} + \begin{bmatrix}0.1 & 0.2 & 0.15 & 0.25\end{bmatrix}$$

$$u = \begin{bmatrix}0.57 & 0.71 & 0.45 & 1.03\end{bmatrix} + \begin{bmatrix}0.1 & 0.2 & 0.15 & 0.25\end{bmatrix}$$

$$u = \begin{bmatrix}0.67 & 0.91 & 0.60 & 1.28\end{bmatrix}$$

$$h = \sigma(u) = \mathrm{ReLU}(\begin{bmatrix}0.67 & 0.91 & 0.60 & 1.28\end{bmatrix})$$
$$h = \begin{bmatrix}0.67 & 0.91 & 0.60 & 1.28\end{bmatrix}$$

SNGULAR

Output Layer

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

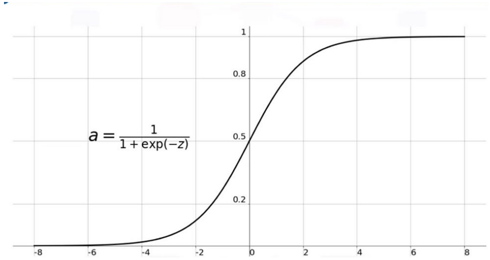
$$v = \begin{bmatrix} 0.67 & 0.91 & 0.60 & 1.28 \end{bmatrix} \cdot \begin{bmatrix} 0.6 & 0.1 \\ 0.5 & 0.3 \\ 0.2 & 0.8 \\ 0.9 & 0.4 \end{bmatrix} + \begin{bmatrix} 0.3 & 0.1 \end{bmatrix}$$

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

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

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

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



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

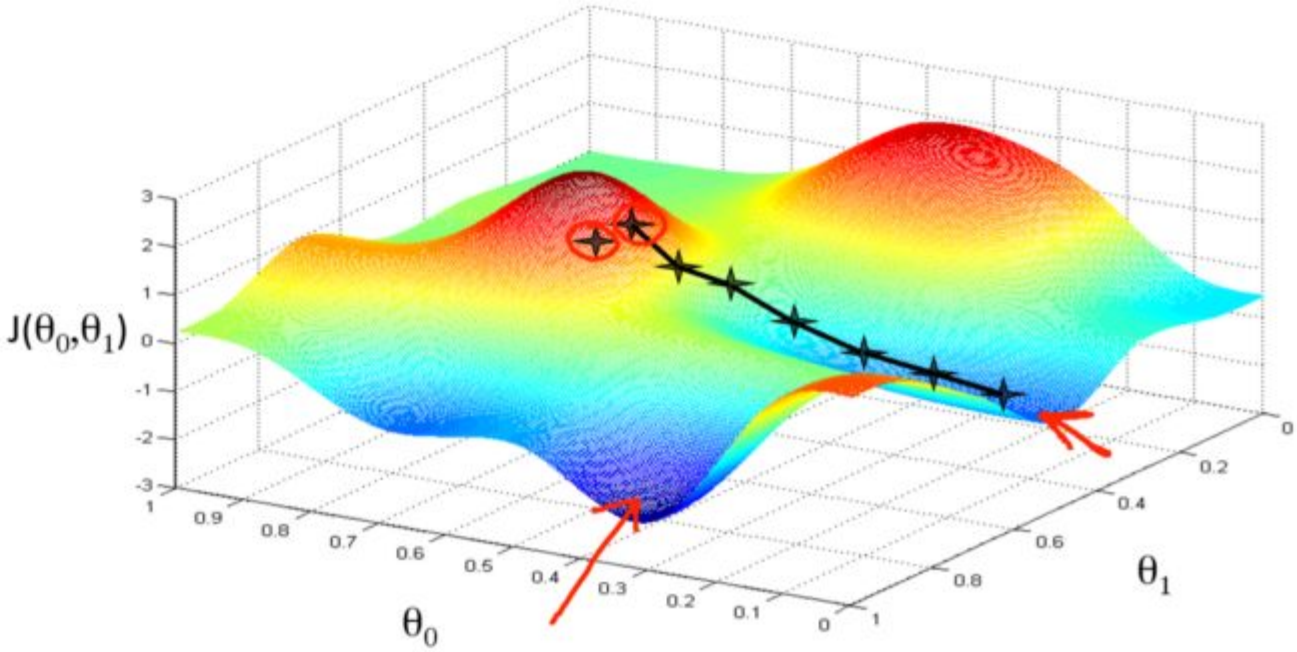
$$y = \begin{bmatrix} \frac{1}{1+e^{-2.427}} & \frac{1}{1+e^{-1.432}} \end{bmatrix}$$

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

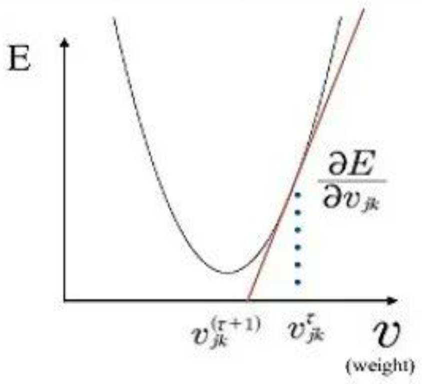
SNGULAR

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}^{τ}	current weight
η	learning rate
E	Error Function

SNGULAR

Transformer

$$\mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

is all you need

https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Input Embedding & Positional Encoding

Let:

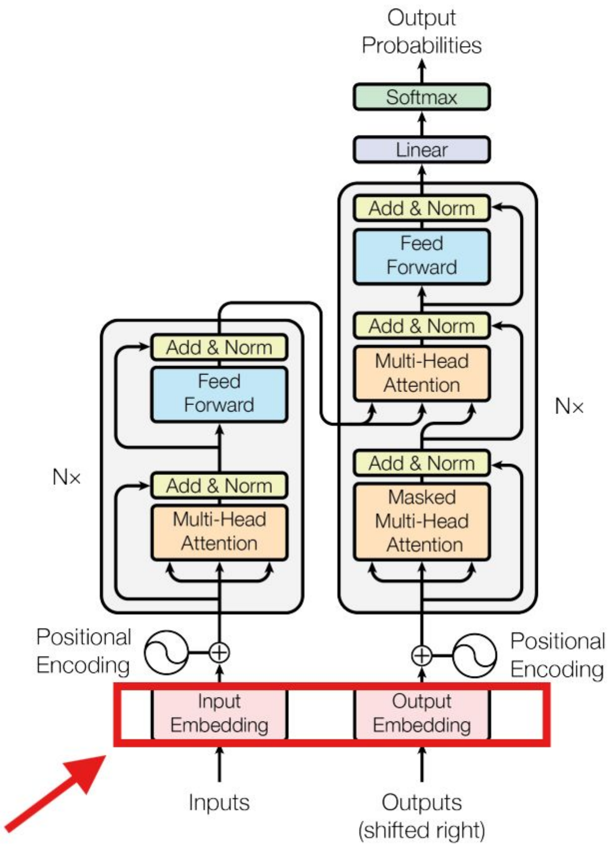
- Vocabulary size = V
- Embedding dimension = d_{model}
- Input sequence = $x = [x_1, x_2, \ldots, x_n]$

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]
```



https://gptforwork.com/tools/tokenizer

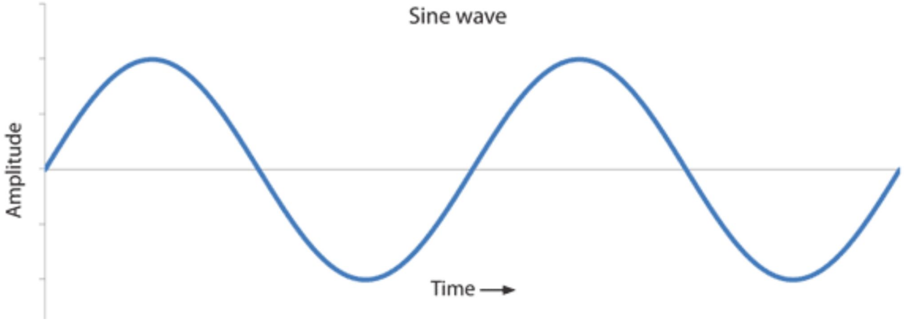
Positional Encoding(Sinusoidal)

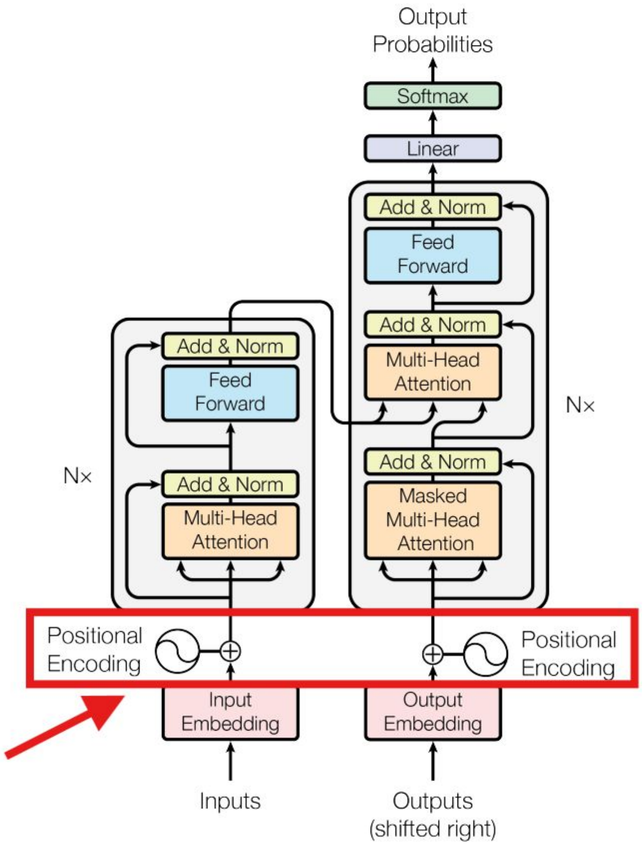
For position pos and dimension i:

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

Then, final input:

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





Multi-Head Self Attention

Scaled Dot-Product

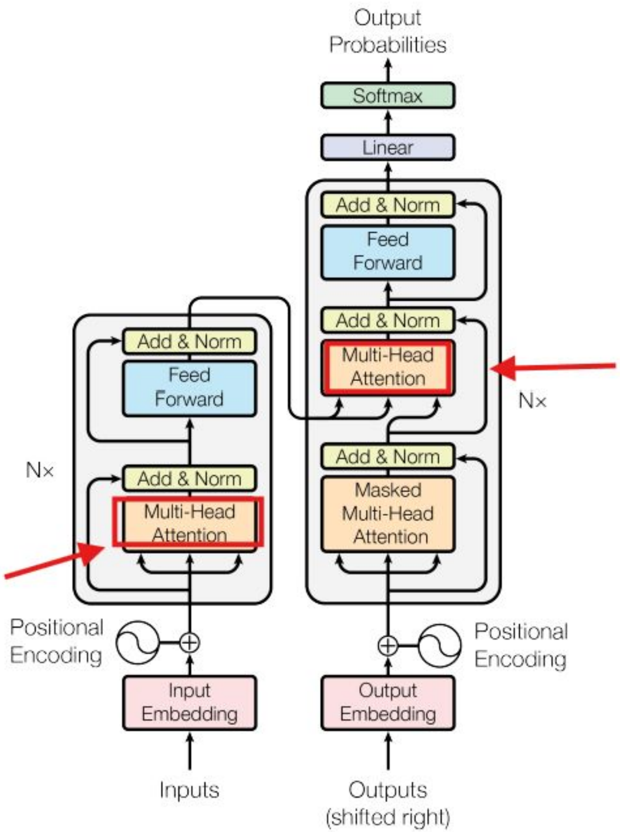
For a given input matrix $X \in \mathbb{R}^{n \times d_{\text{model}}}$, define:

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

Attention:

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

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



Multi-Head Self Attention

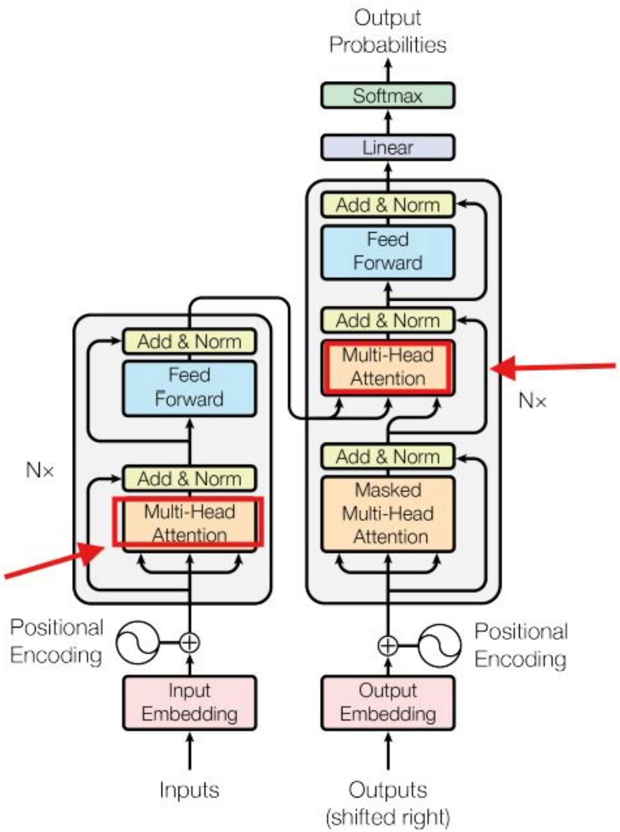
Multi-Head Attention

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

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i)$$
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

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:

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

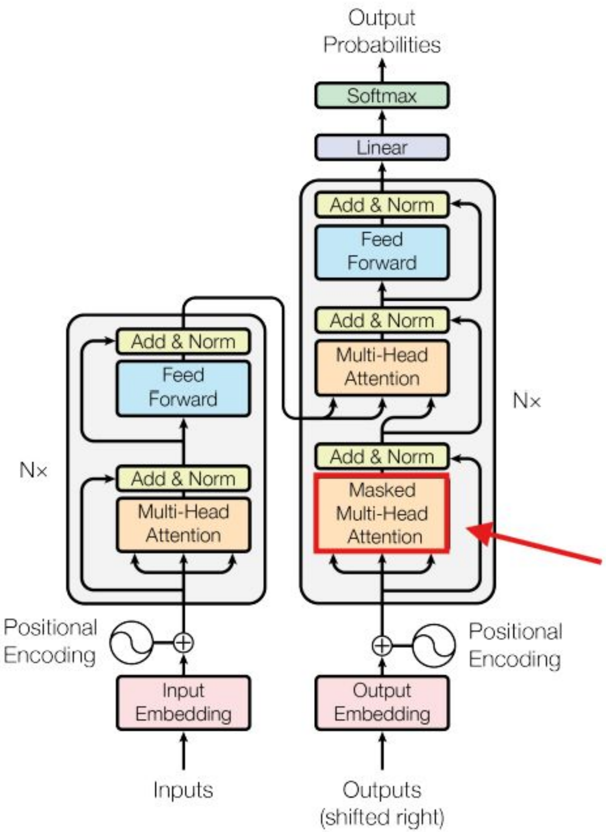
So we modify the scores as:

$$A' = A + M$$

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

Then output of attention:

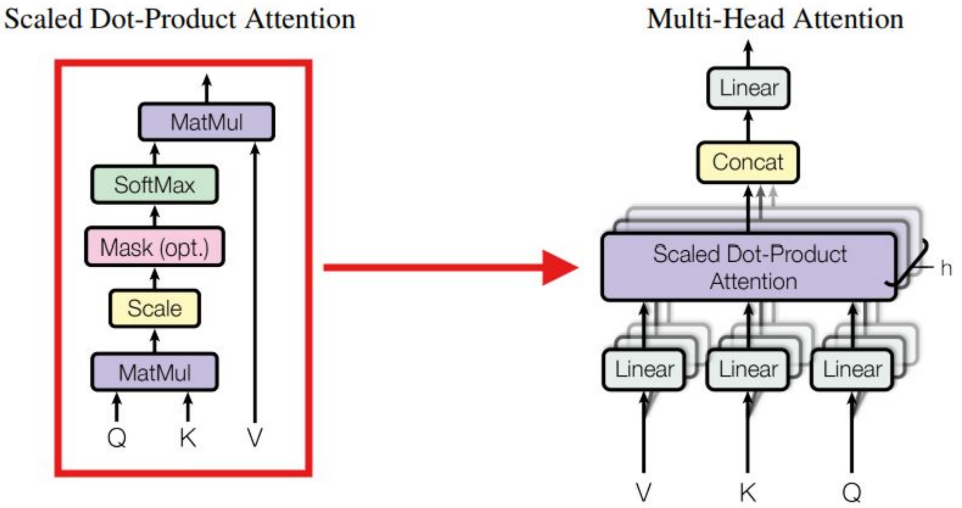
$$\text{Attention}(Q, K, V) = \alpha V$$



SNGULAR

Masked Multi-Head Attention

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



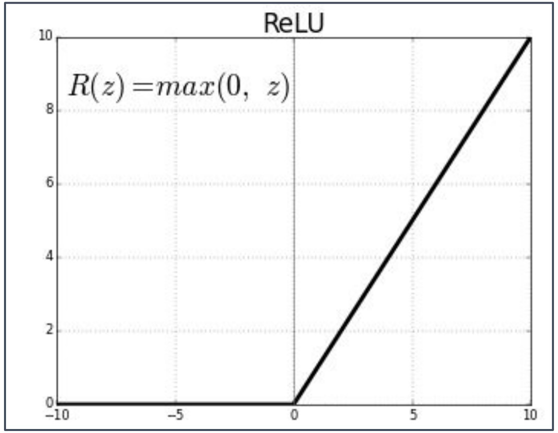
SNGULAR

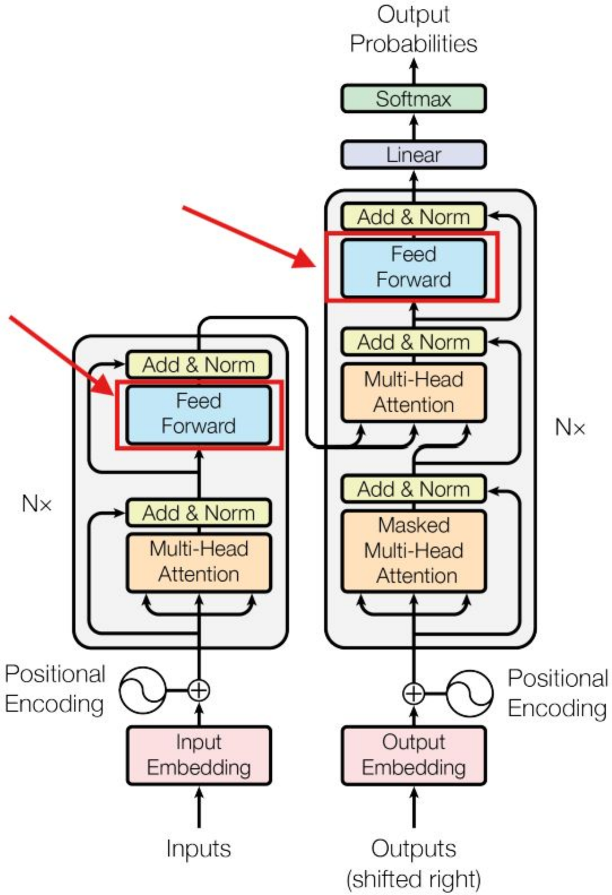
Masked Multi-Head Attention

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

Where:

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



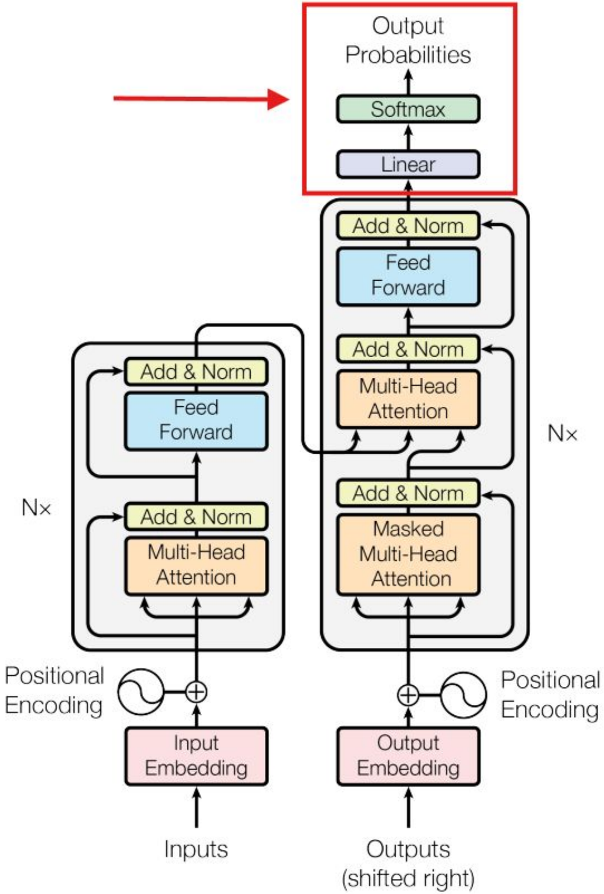


SNGULAR

Output Layer - Linear & Softmax

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

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



SNGULAR

Output Layer - Linear & Softmax

Hidden vector $\mathbf{h}$:

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

Output weight matrix W_{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%

SNGULAR

Learning Mechanism - Cross Entropy Loss Function

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

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

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

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

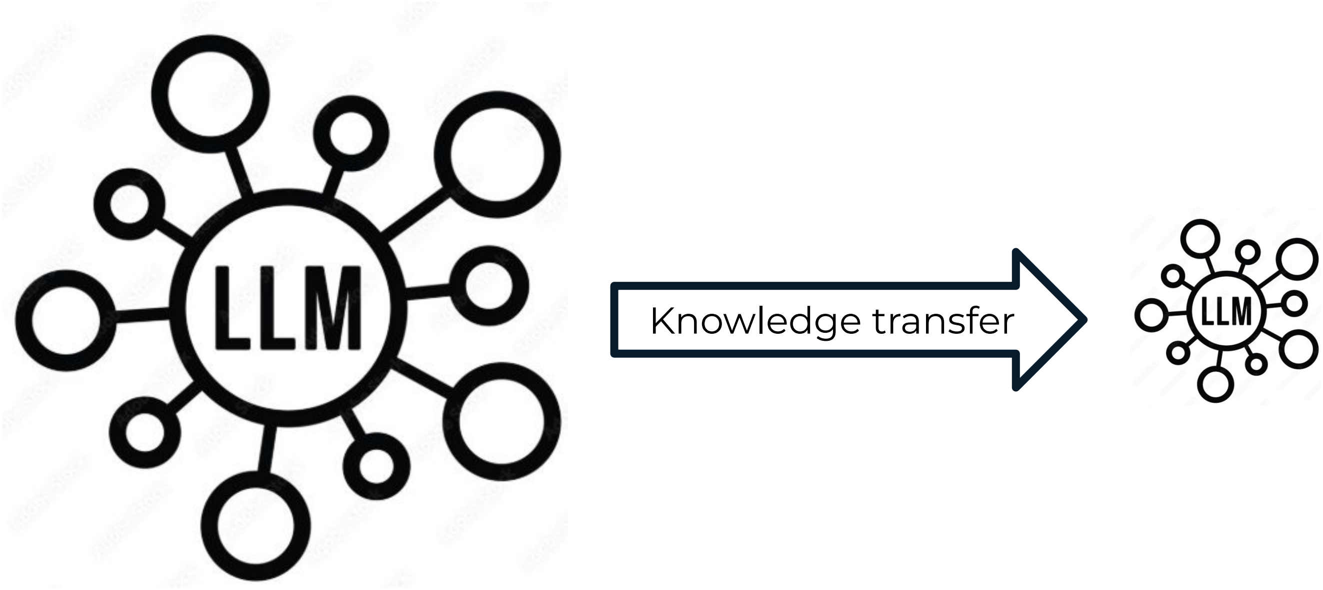
$$\text{Loss} = -\log(\hat{y}_2) = -\log(0.35)$$

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

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

$$\boxed{1.05}$$

LLM Distillation



Teacher Model (M_T)

Student Model (M_S)

SNGULAR

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**).

SNGULAR

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 **Ps**, 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.

SNGULAR

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_{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

SNGULAR

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 your own distilled LLM
- Fine Tuning types.
- Dataset preparation
  - **Vectorization**
  - **K-Means**
- Use case - Hugging Face model creation.
- Decoder Strategies for LLMs.
- Deployment & integration Use Case.

SNGULAR

Questions

Thanks for your attention!!!