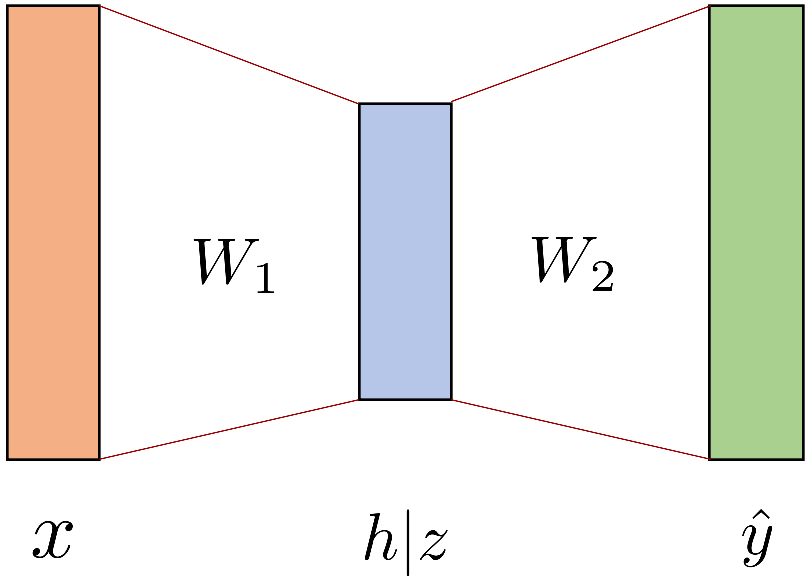
Memory Analysis in Transformers

Kun Yuan

Center for Machine Learning Research @ Peking University

Warmup: Linear neural network



dims: d (p,d) p (q,p) q

Forward

$$h = W_1 x$$

$$z = \sigma(h)$$

$$\hat{y} = W_2 z$$

$$f = L(\hat{y})$$

Backward

$$\frac{\partial f}{\partial W_1} = \frac{\partial f}{\partial h} x^T$$

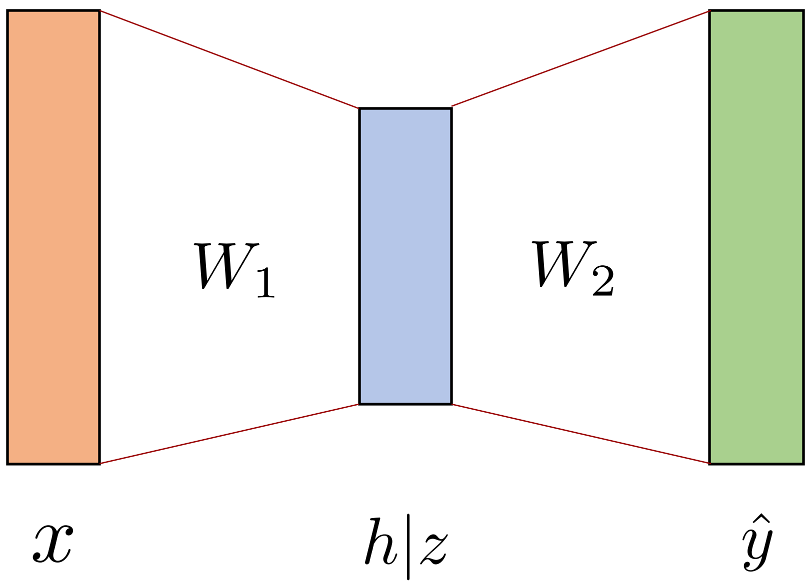
$$\frac{\partial f}{\partial h} = \frac{\partial f}{\partial z} \odot \nabla \sigma(h)$$

$$\frac{\partial f}{\partial W_2} = \frac{\partial L}{\partial \hat{y}} z^T, \quad \frac{\partial f}{\partial z} = W_2 \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial f}{\partial \hat{y}} = \nabla L(\hat{y})$$

Store h, z and $\hat{y}$ **Store $\nabla_{W_1} f(W_1)$ and $\nabla_{W_2} f(W_2)$**

Warmup: Linear neural network with batch size



dims: d (p,d) p (q,p) q

Forward

$$h_b = W_1 x_b$$

$$z_b = \sigma(h_b)$$

$$\hat{y}_b = W_2 z_b$$

$$f = \frac{1}{B}\sum_{b=1}^{B} L(\hat{y}_b)$$

Backward

$$\frac{\partial f}{\partial W_1} = \frac{1}{B}\sum_{b=1}^{B} \frac{\partial f}{\partial h_b} x_b^T,$$

$$\frac{\partial f}{\partial h_b} = \frac{\partial f}{\partial z_b} \odot \nabla\sigma(h_b)$$

$$\frac{\partial f}{\partial W_2} = \frac{1}{B}\sum_{b=1}^{B} \frac{\partial L}{\partial \hat{y}_b} z_b^T, \quad \frac{\partial f}{\partial z_b} = W_2 \frac{\partial f}{\partial \hat{y}_b}$$

$$\frac{\partial f}{\partial \hat{y}_b} = \frac{\partial L}{\partial \hat{y}_b}$$

Store $\{\mathbf{h_b}, \mathbf{z_b}, \hat{y}_b\}_{b=1}^{B}$

Store $\nabla_{W_1} f(W_1)$ and $\nabla_{W_2} f(W_2)$

Memory decomposition

Memory = Model + Gradient + Optimizer states + Activations

- Given a model with P parameters, gradient will consume P parameters, and Optimizer states will consume 2P parameters; **4P parameters in total.**

- When using FP32 to store parameters, each parameter takes **4** Bytes

- When using FP16 or BF16 to store parameters, each parameter takes **2** Bytes

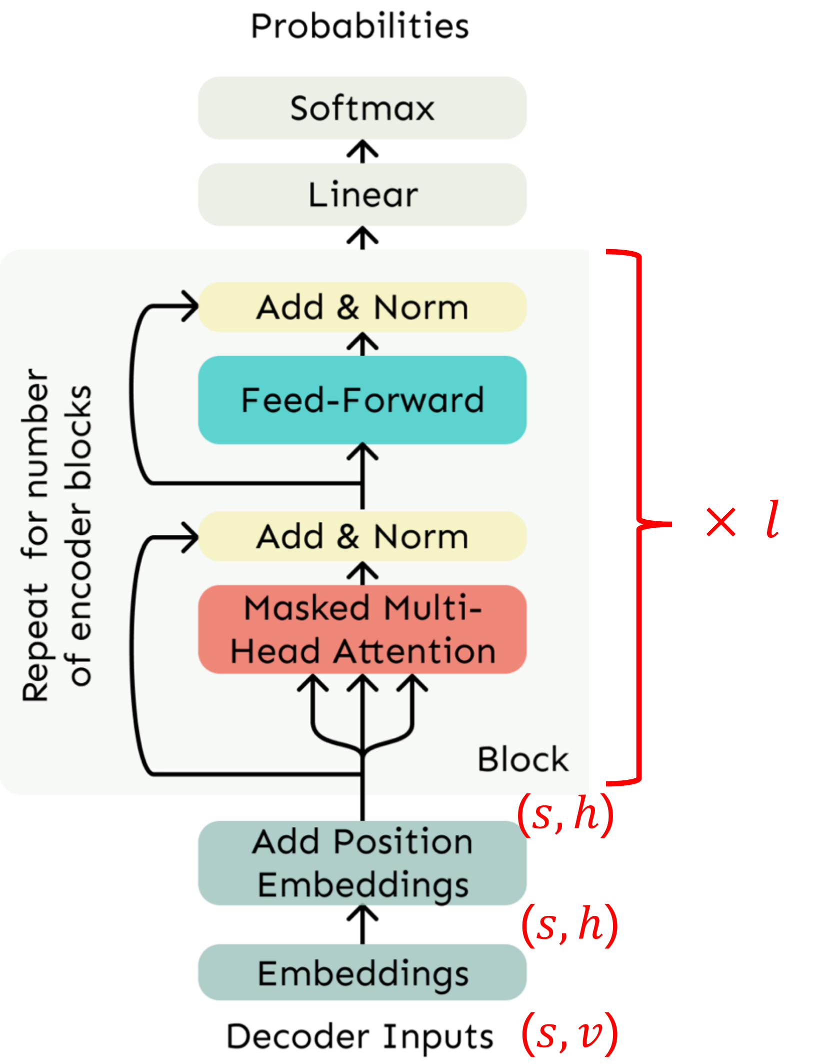
Decoder-only Transformers

- Number of the transformer layers: l
- Sequence length: s
- Vocabulary size: v
- Embedding representation dims: h

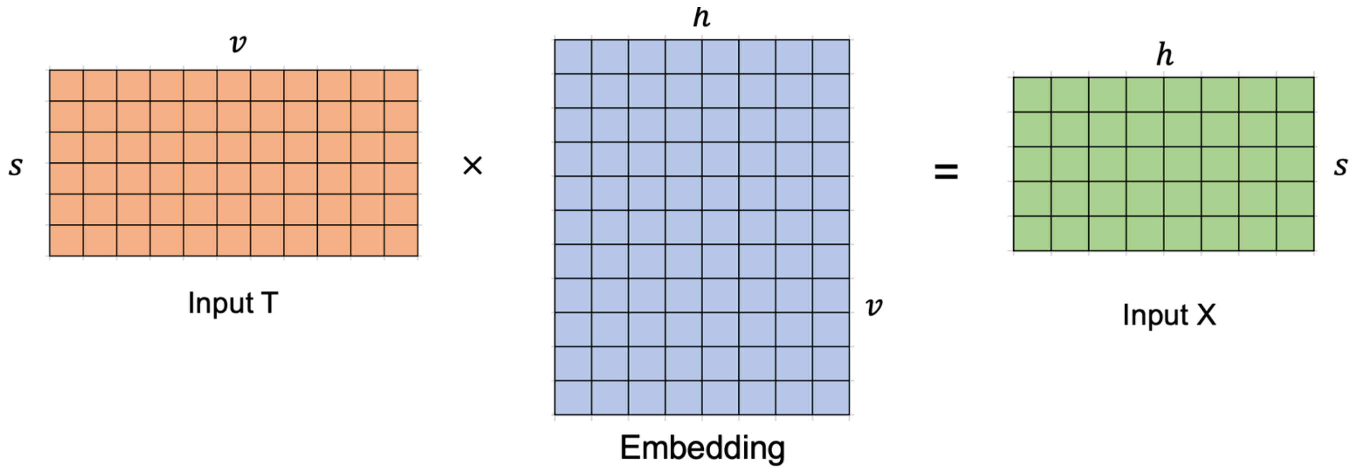
Parameters $P = 12\ell h^2 + 2vh$

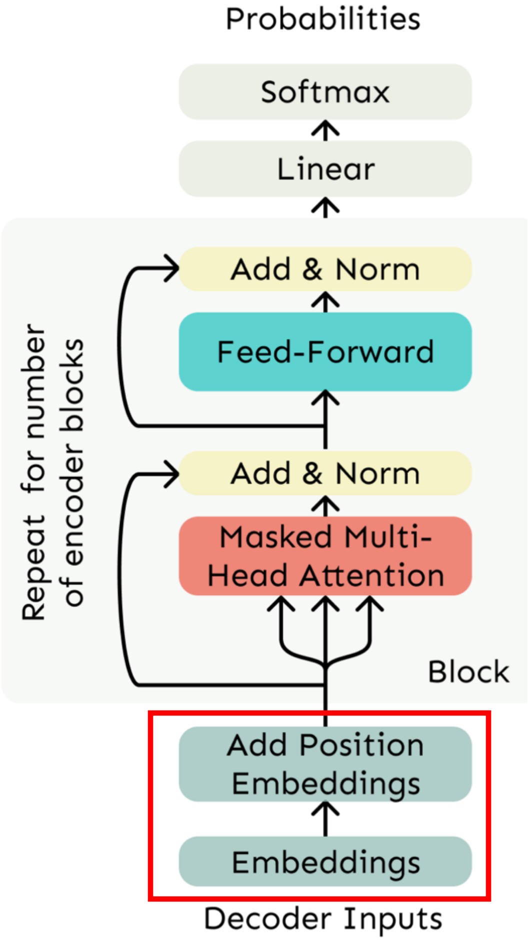
Model + Gradient + Optimizer states = 4P * 4 (Bytes)

Next we estimate activations



Embedding



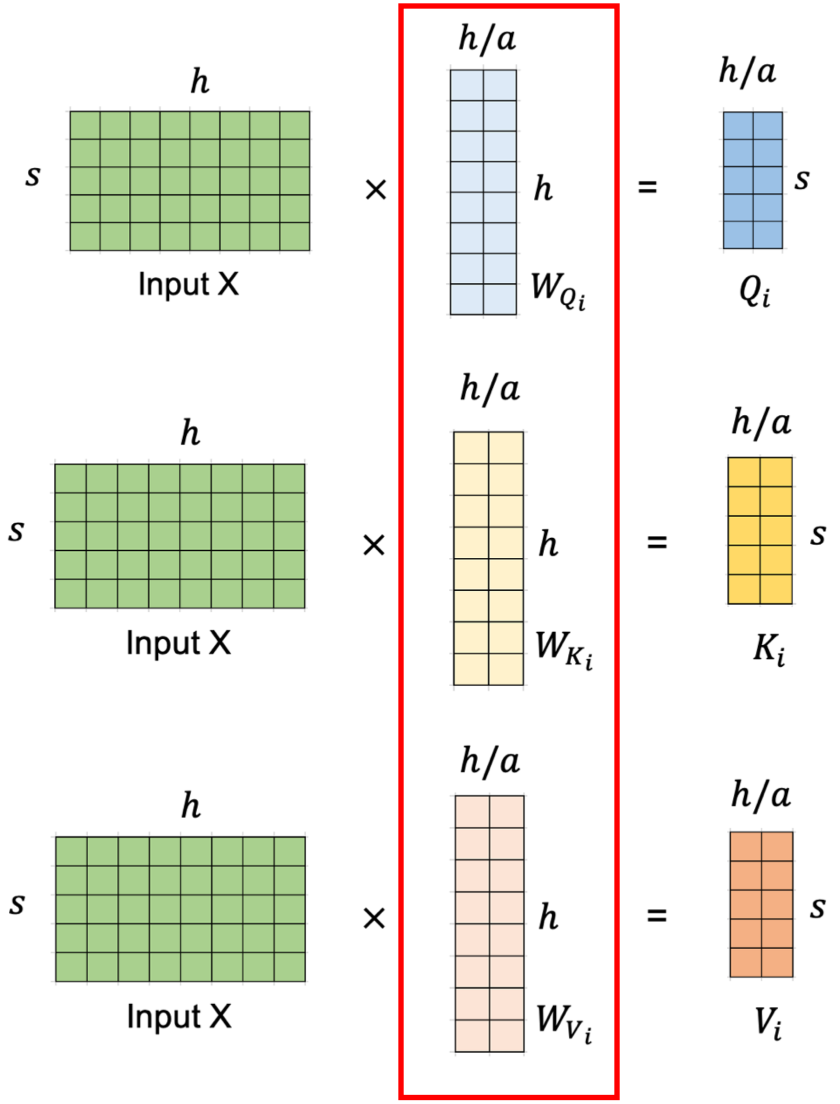


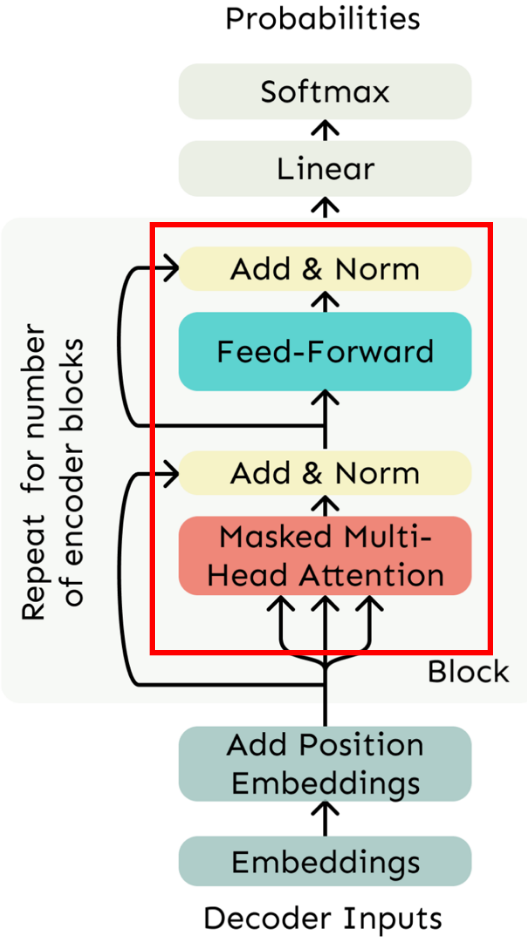
- We need to store the embedding activations with parameters sh
- Position embedding can be ignored when using RoPE and ALiBi

Multi-head attentions

- We need to store Q_i, K_i and V_i

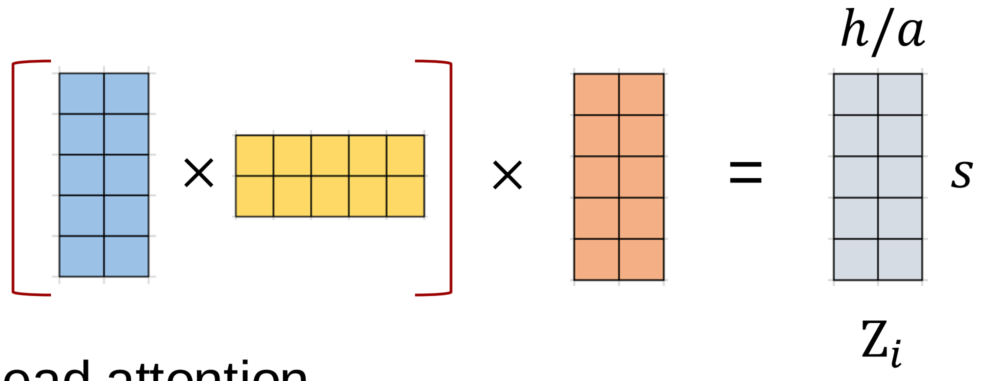
$$3(sh/a) \times a = 3sh$$





Self-attention

$$\mathrm{softmax}\left(\frac{Q_i K_i^T}{\sqrt{h/a}}\right)V_i \;=\; \text{softmax}$$

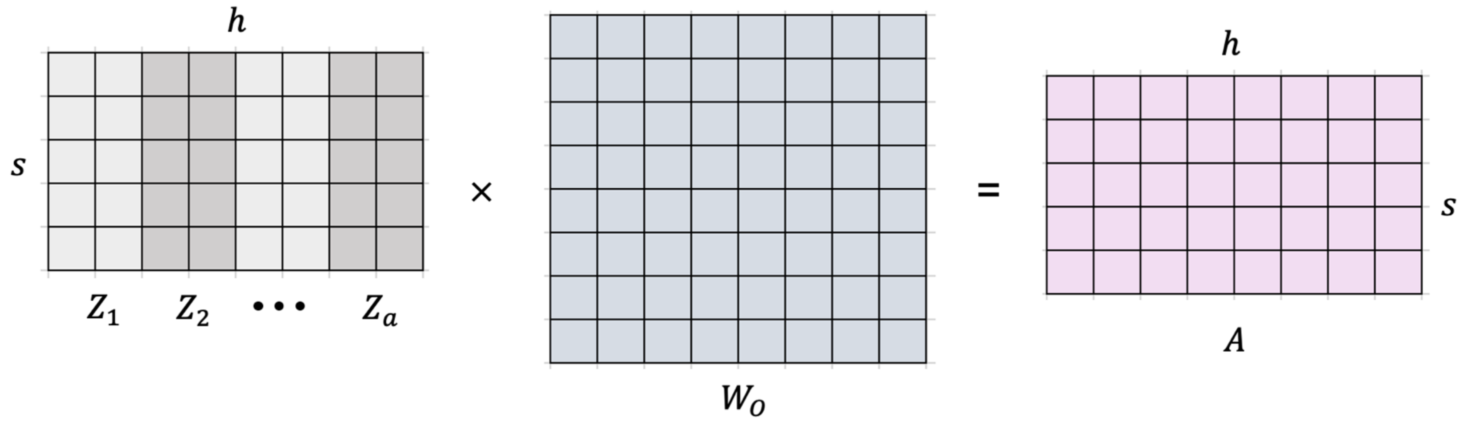


One-head attention

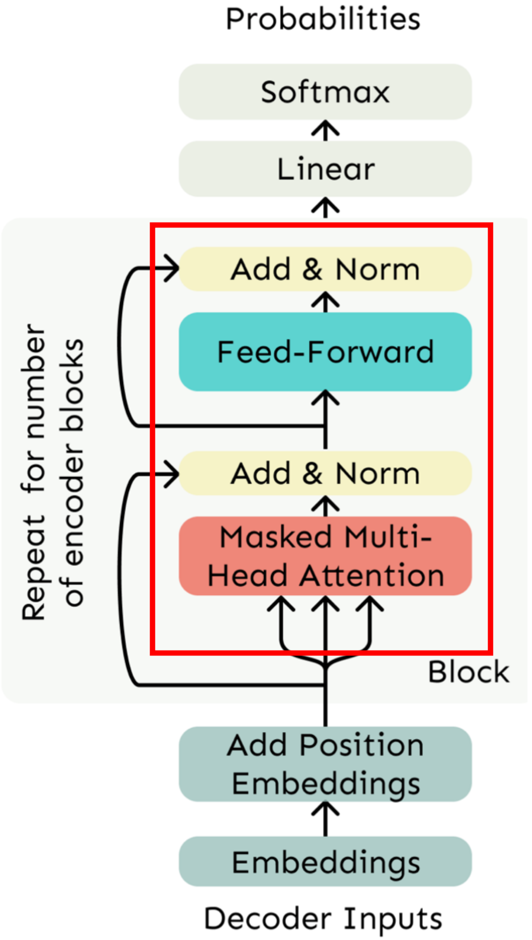
- Store $Q_i K_i^T$ with s^2 parameters;
- Store softmax($Q_i K_i^T$) with s^2 parameters
- Store $\mathrm{softmax}\left(\frac{Q_i K_i^T}{\sqrt{h/a}}\right)V_i$ with sh/a parameters

Store $2s^2a + sh$ activations

Multi-head attentions



- We need to store A with sh parameters



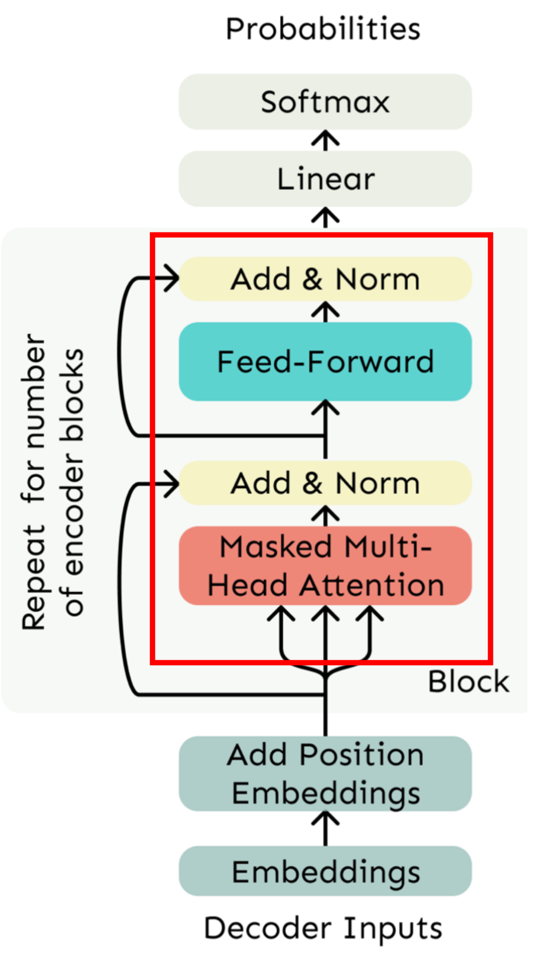
Layer normalization

- Then layer normalization computes:

$$\text{output} = \frac{x - \mu}{\sqrt{\sigma} + \epsilon} * \gamma + \beta$$

- Dims of γ and β: h

- We can ignore the layer normalization



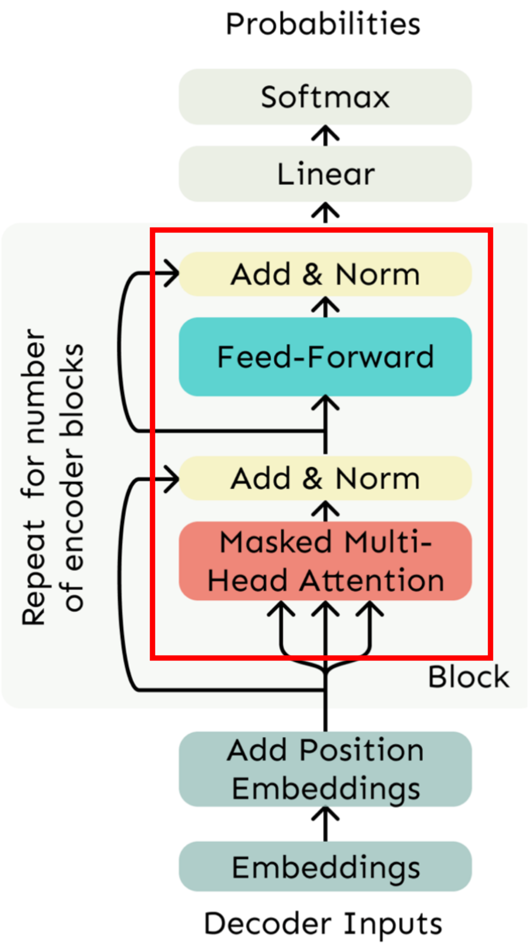
Feed-forward layers

$$X' = \mathrm{ReLU}(A \cdot W_1 + b_1) \cdot W_2 + b_2$$

- Store AW_1 with 4sh parameters
- Store ReLU(AW_1) with 4sh parameters
- Store ReLU(AW_1)$\cdot$ W_2 with sh parameters

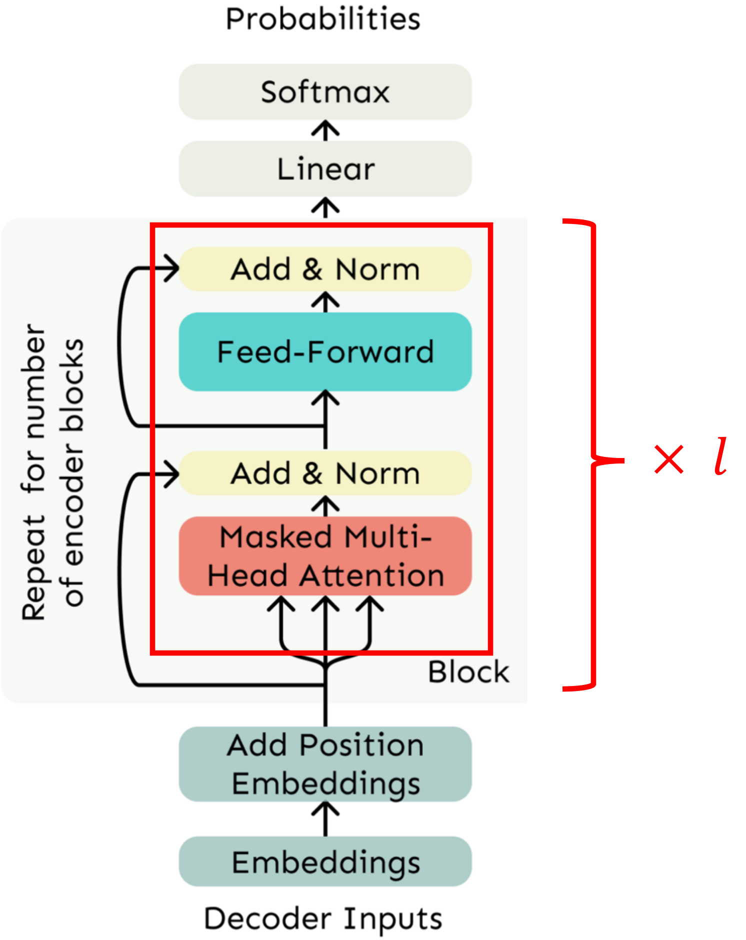
9sh

- The activations of b_1 and b_2 can be ignored



Transformer block

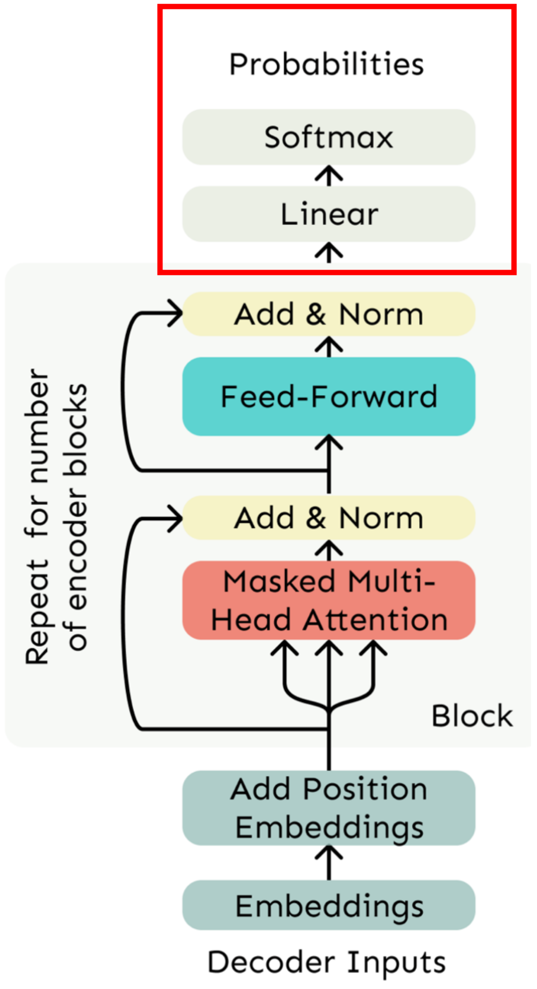
- Multi-head attentions: $5sh + 2s^2a$
- Feed-forward layers : $9sh$
- l layers of attentions : $(2s^2a + 14sh) \times l$



Probability predictions

$$p = \mathrm{Softmax}(X \cdot W_v + b_v)$$

- Store XW_v with sv parameters
- Store Softmax(XW_v) with sv parameters



Total activations

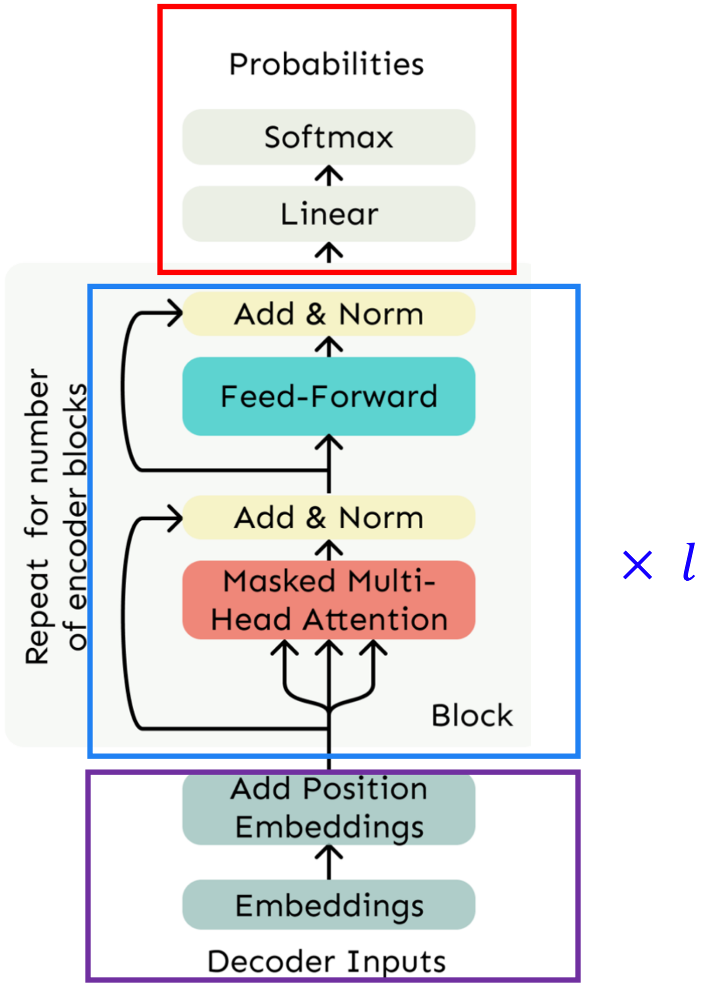
- Embedding activations: sh
- Self-attention activations: $(2s^2a + 14sh) \times l$
- Probability activations: $2sv$

Total activation parameters with batch-size 1:

$$(2s^2a + 14sh) \times l + 2sv + sh$$

Ignoring 2sv and sh and using batch-size b:

$$(2s^2a + 14sh) \times l \times b$$



Total memory

Memory = **Model + Gradient + Optimizer states** + **Activations**

$$\left(48\, l\, h^2 + bl(2s^2a + 14sh)\right) \times 4 \text{ Bytes}$$

- When hidden state h is large, the model parameters dominate the memory
- When batch-size b or sequence length s is large, the activation dominates the memory

Examples

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

GPT3 has 175B parameters; its model consumes $4 \times 175 \times 10^9$ Bytes = 700 GB

Its gradient takes 700 GB parameters; Optimizer states take 1.4 TB

GPT has sequence length s = 2048. When b=1, its activation takes 444 GB, 63% of the model

When b=128, its activation is 81 times of the model size

Summary

Memory = **Model + Gradient + Optimizer states** + **Activations**

$$\left(48\, l\, h^2 + bl(2s^2a + 14sh)\right) \times 4 \text{ Bytes}$$

- When hidden state h is large, the model parameters dominate the memory
- When batch-size b or sequence length s is large, the activation dominates the memory
- The activation-incurred memory **cannot be ignored**