Transformers

Generated by Stable Diffusion 2.1
prompt: "purple and orange transformer toy, comic book style"

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Reminders

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

- A **Linear** layer is a regression: y = xW +b
- **Softmax** turns vectors into probability distribution (softmax(x_i) = exp(x_i) / sum(exp(x)))
- **Normalization** takes a vector and subtracts mean, then divide by var to get mean 0, var 1 version of vector

$$\text{Norm}(x_i) = \frac{x_i - \mu}{\sigma^2}$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\sigma^2 = \text{var}(x)$$

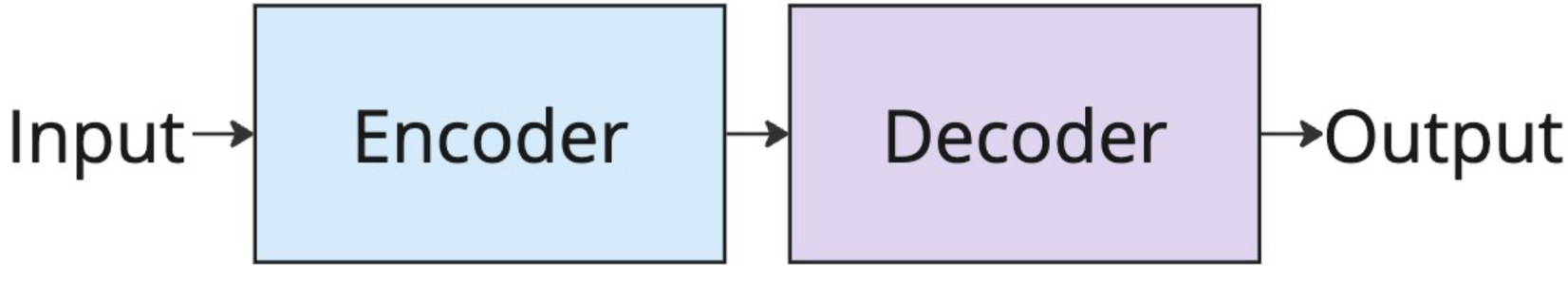
NLP Concepts

- Natural language processing (NLP) is how ML practitioners analyze text
- 3 main waves:
    - 'classical' methods: naive bayes, bag of words, n-gram
    - Recursive neural networks: auto-regressive network that keeps hidden state to store context
    - Transformer-based: current state of the art -- our focus today
- Neural networks like numbers, text is not numbers
- Definitions:
    - Token: small set of possible values (e.g. characters, words)
    - Vocabulary: fixed set of possible tokens for analysis
    - Tokenize: convert text into sequence of integers (index into vocab)

Attention

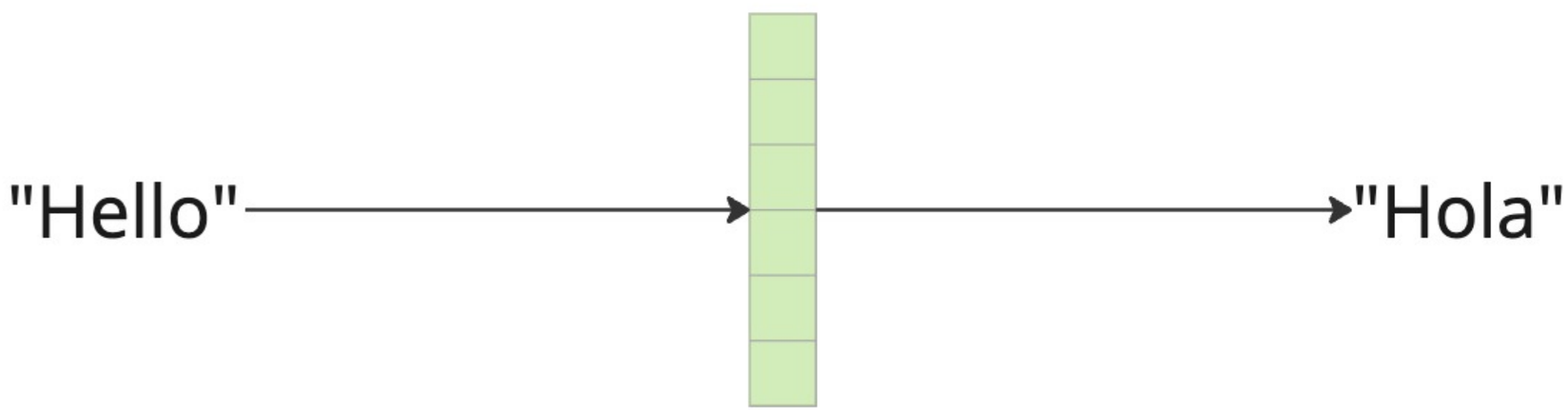
- Current state of the art in NLP is transformer
- Based on key idea called "Attention" (or self-attention)
- Allows network to selectively focus on other tokens (words or characters) depending on context
    - Quite literally answers "what do I pay attention to when processing this token"
- Applied by taking weighted averages
- Key in transformer is constructing the weights for these averages

Encoder
Decoder
Networks

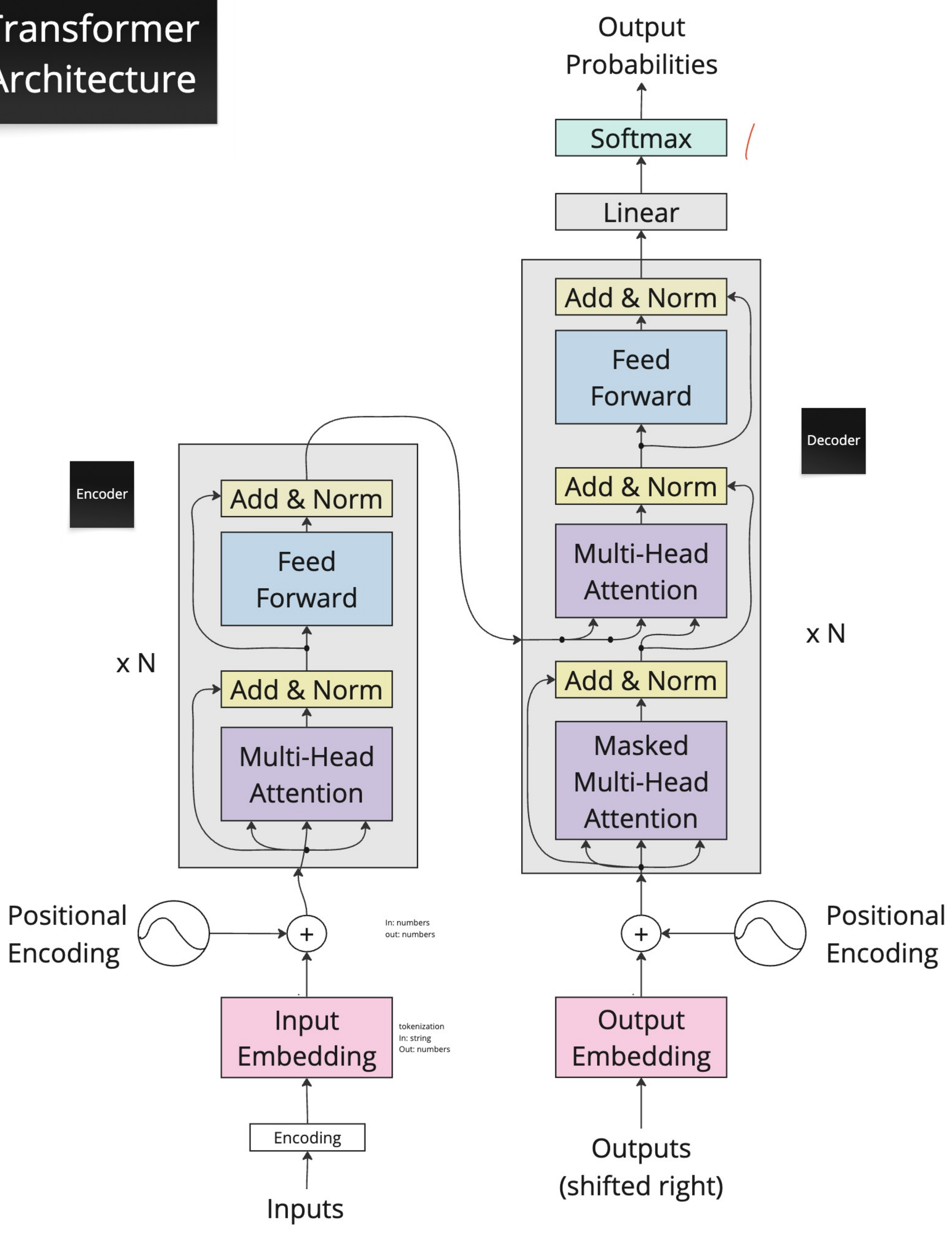


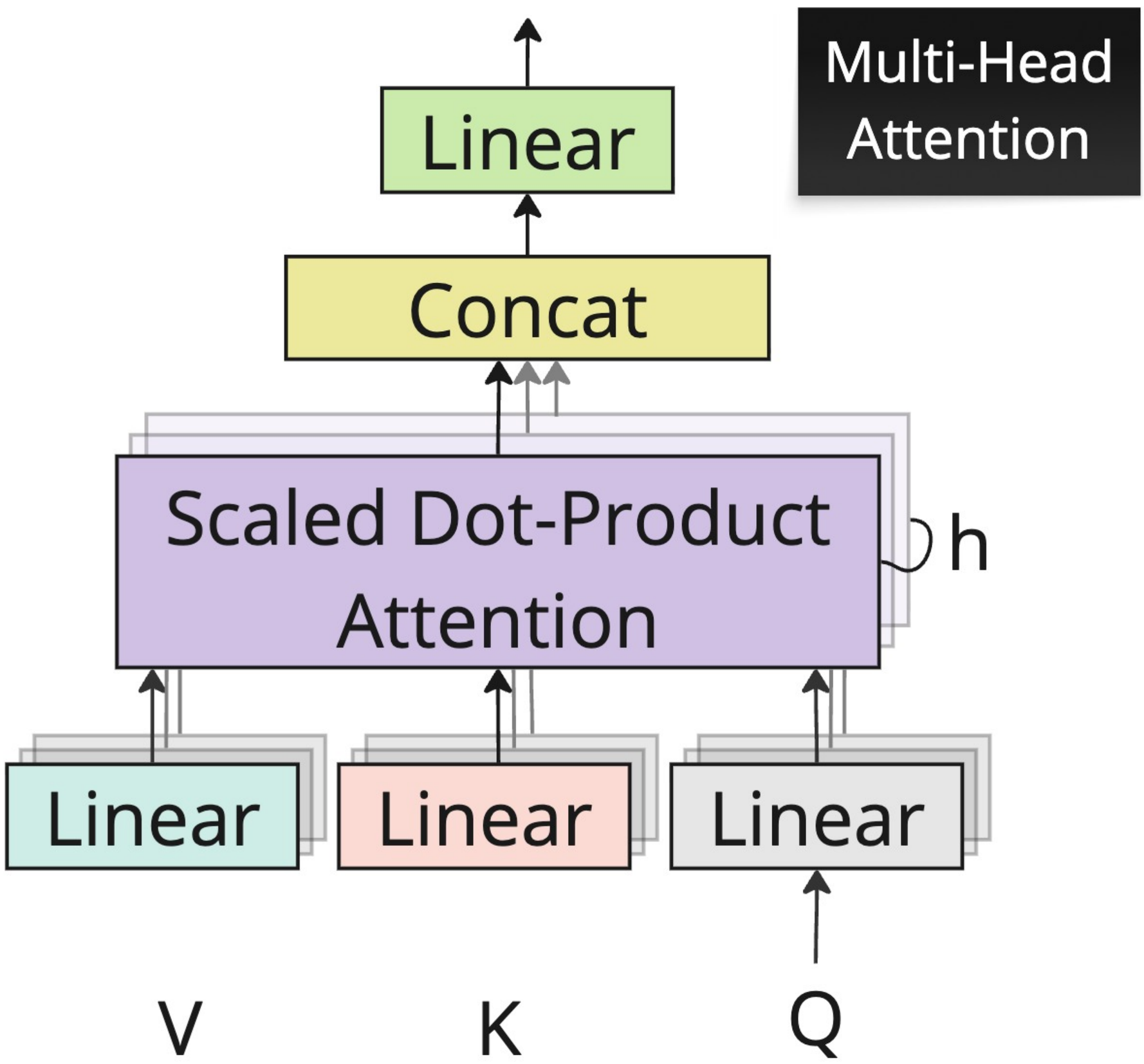
Maps inputs to vectors

Converts vectors to outputs



Transformer Architecture





Code

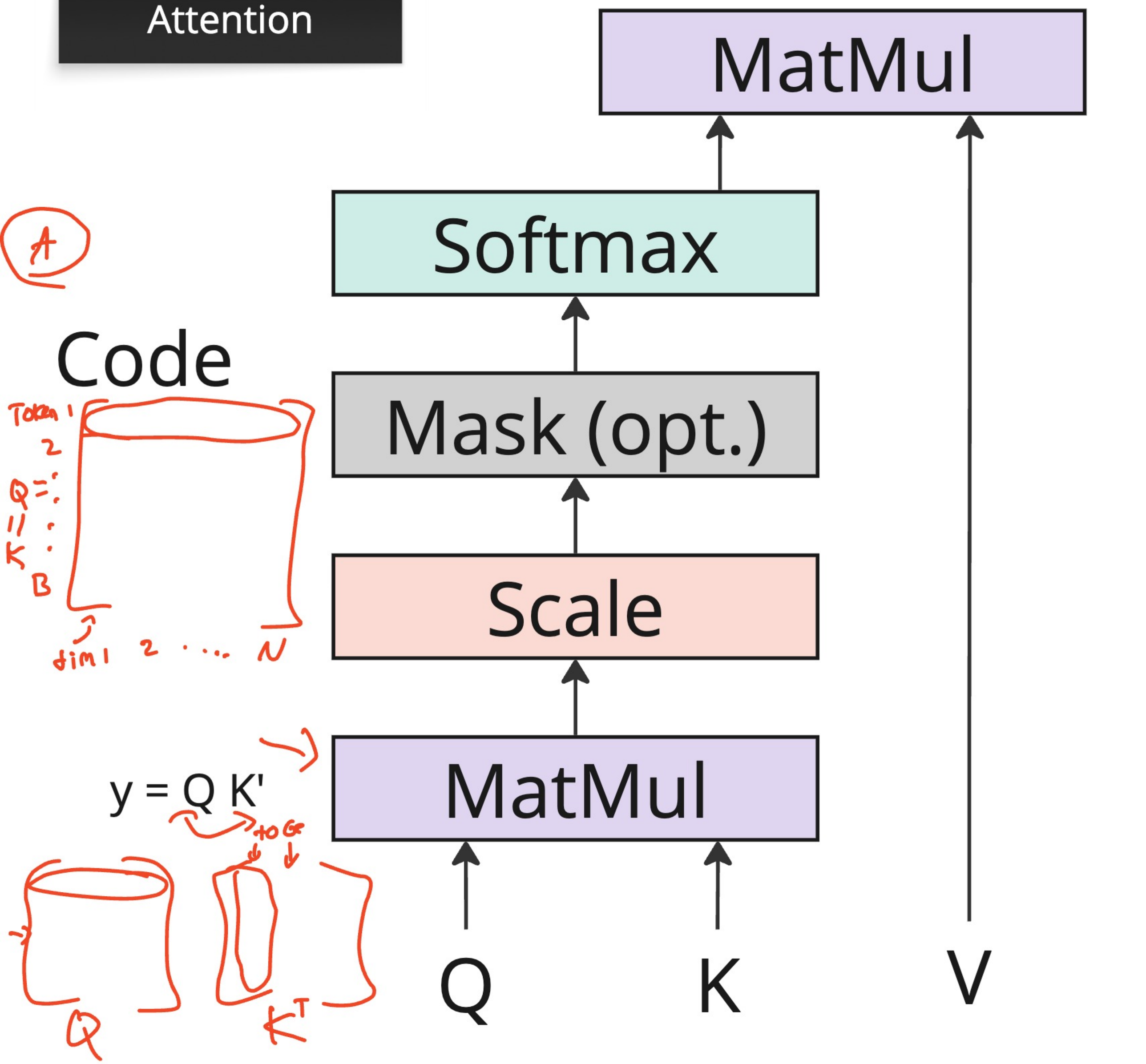
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

xWQ, x Wk, x wV

reference: https://arxiv.org/abs/1706.03762

Scaled Dot-Product Attention



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

Scale

$$x \rightarrow \frac{x}{\sqrt{d_k}}$$

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

- Scale just before softmax
- Without scale, "spikey" distribution
- Scaling flattens, makes distribution diffuse

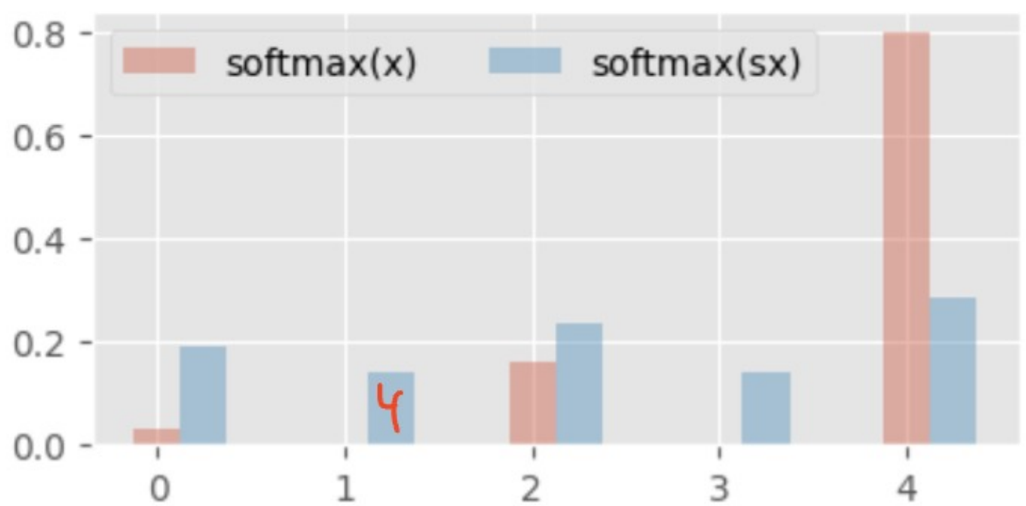
```
[30]: sx = torch.tensor([0.1, -0.2, 0.3, -0.2, 0.5])
      x = sx * 8

[31]: torch.softmax(x, dim=-1)

[31]: tensor([0.0326, 0.0030, 0.1615, 0.0030, 0.8000])

[32]: torch.softmax(sx, dim=-1)

[32]: tensor([0.1925, 0.1426, 0.2351, 0.1426, 0.2872])
```



Mask

KQT

exp(-∞) = 0

Hello	1.76	2.25	-3.78	0.65
I	-0.84	-0.19	0.25	-3.89
am	0.93	2.21	2.35	8.07
Spencer	0.55	10.96	3.33	-4.8

+

mask

0	-inf	-inf	-inf
0	0	-inf	-inf
0	0	0	-inf
0	0	0	0

=

masked scores

1.76	-inf	-inf	-inf
-0.84	-0.19	-inf	-inf
0.93	2.21	2.35	-inf
0.55	10.96	3.33	-4.8

Apply Softmax...

$$x_i \rightarrow \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

mask

1	0	0	0
0.34	0.66	0	0
0.1	0.35	0.55	0
0.0	0.99	0.01	0.0

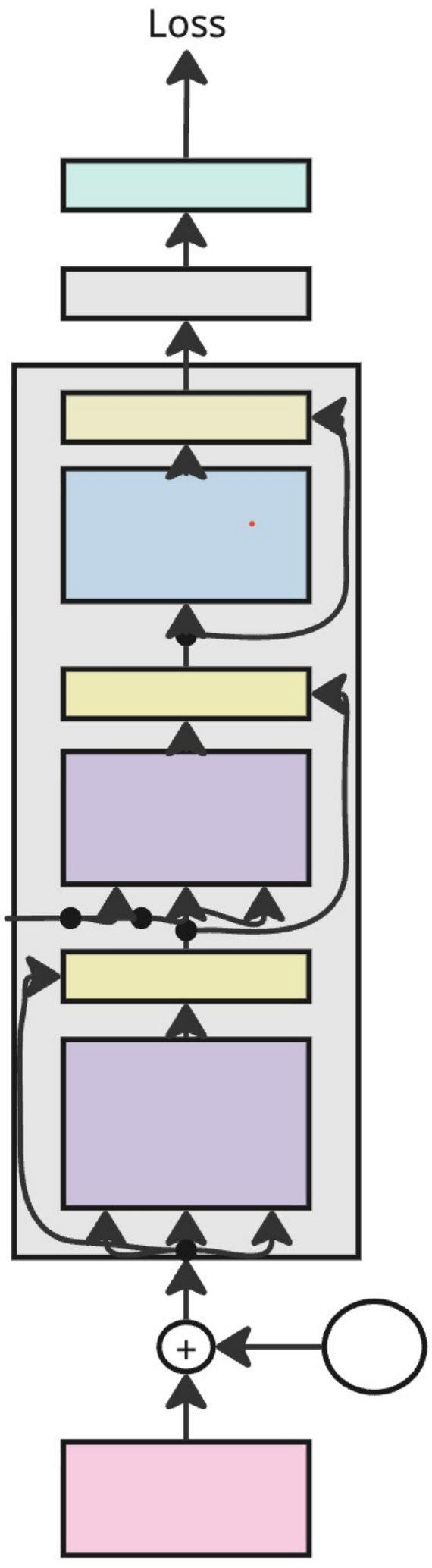
Residual connections (Add)

Before

$$x \rightarrow f(x)$$

After

$$x \rightarrow x + f(x)$$



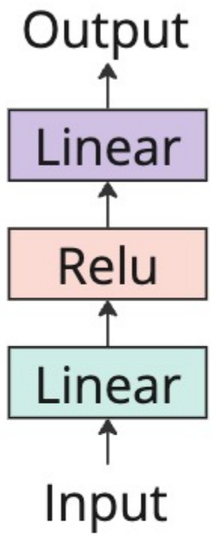
Allows direct gradient connection from loss to deep/early parts of network. Better training

Layer Norm

$$x \rightarrow \frac{x - E(x)}{\sqrt{\text{var}(x) + \epsilon}} * \gamma + \beta$$

- Applied across rows
- Makes all features mean 0, std 1
- Paper says after attention - today people do it before
- Helps smooth learning inside attention mechanism

Feed Forward



$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

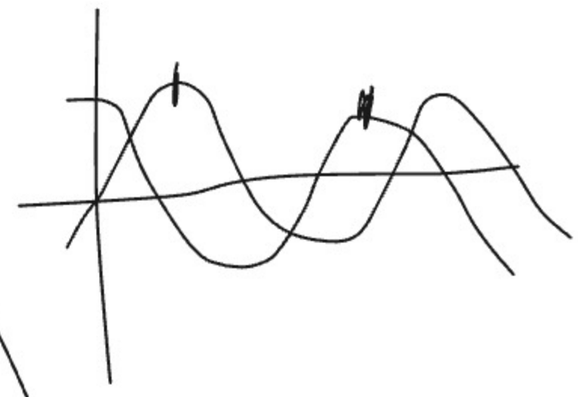
reference: https://arxiv.org/abs/1706.03762

```python
class FeedFoward(nn.Module):
    def __init__(self, n_embd):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(n_embd, 4 * n_embd),
            nn.ReLU(),
            nn.Linear(4 * n_embd, n_embd),
        )

    def forward(self, x):
        return self.net(x)
```

reference: https://github.com/karpathy/ng-video-lecture

Positional Encoding



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

- CNN or RNN have relative position built into structure
- Transformer does not -- must be added
- In paper, added via encoding functions above
    - Cyclical
    - Can extrapolate
    - Does not depend on data -- just index of data
- Allows model to utilize relative and absolute position info
    - "I'm a verb in position 7" (key) and "I need a noun in positions 2, 3, or 5" (query)
- In practice people often use "Embedding" layers

```python
def positional_encoding(bs, dmodel, n=10000):
    P = np.zeros((bs, dmodel))
    for b in range(bs):
        for i in range(dmodel//2):
            x = b / (n ** (2*i/dmodel))
            P[b, 2*i] = np.sin(x)
            P[b, 2*i+1] = np.cos(x)
    return P
```