香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

COMP 4901B
Large Language Models

Recurrent Neural Networks, Transformers

Junxian He

Sep 12, 2025

Recap: Autoregressive Language Models

$$
\begin{aligned}
p(\text{the, mouse, ate, the, cheese}) = \; & p(\text{the}) \\
& p(\text{mouse} \mid \text{the}) \\
& p(\text{ate} \mid \text{the, mouse}) \\
& p(\text{the} \mid \text{the, mouse, ate}) \\
& p(\text{cheese} \mid \text{the, mouse, ate, the}).
\end{aligned}
$$

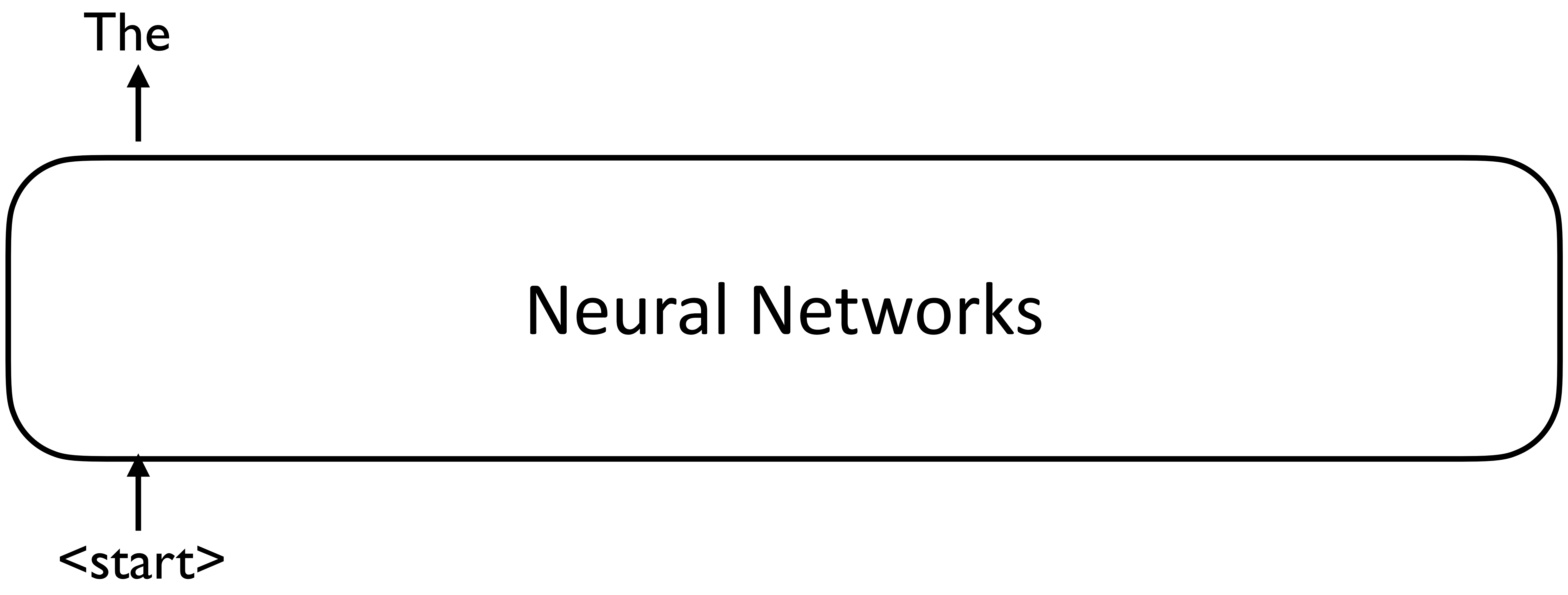
$$p(x_1, x_2, \ldots, x_I) = \prod_{i=1}^{I} p(x_i \mid x_{1:i-1})$$

Next Word Context

Recap: Neural Language Models

Neural language models are typically autoregressive

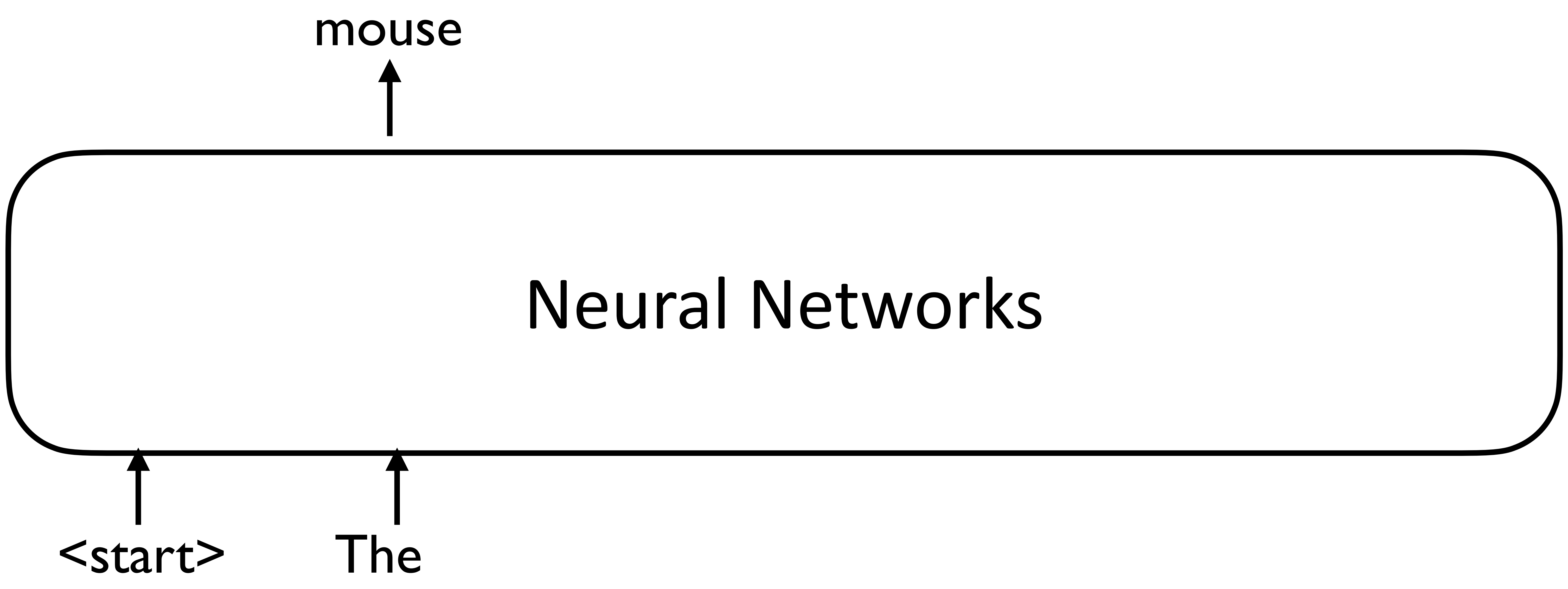
Data: "The mouse ate the cheese ."



Recap: Neural Language Models

Neural language models are typically autoregressive

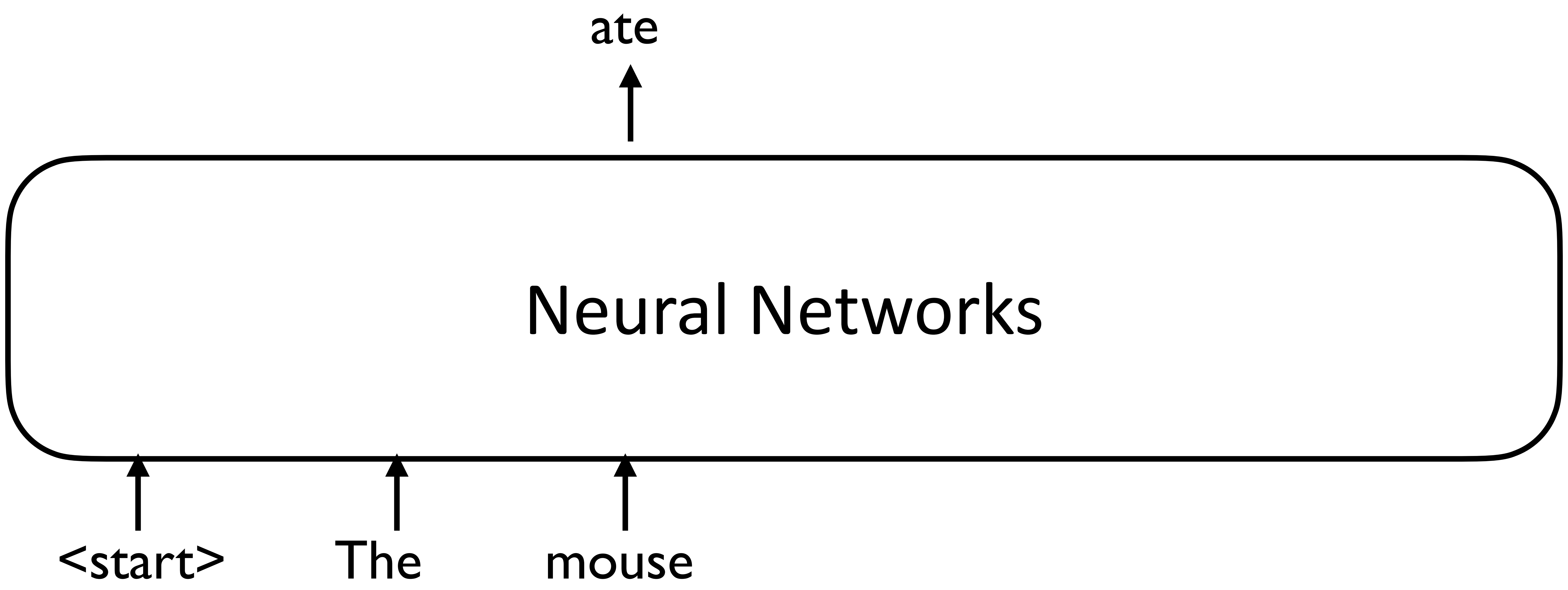
Data: "The mouse ate the cheese ."



Recap: Neural Language Models

Neural language models are typically autoregressive

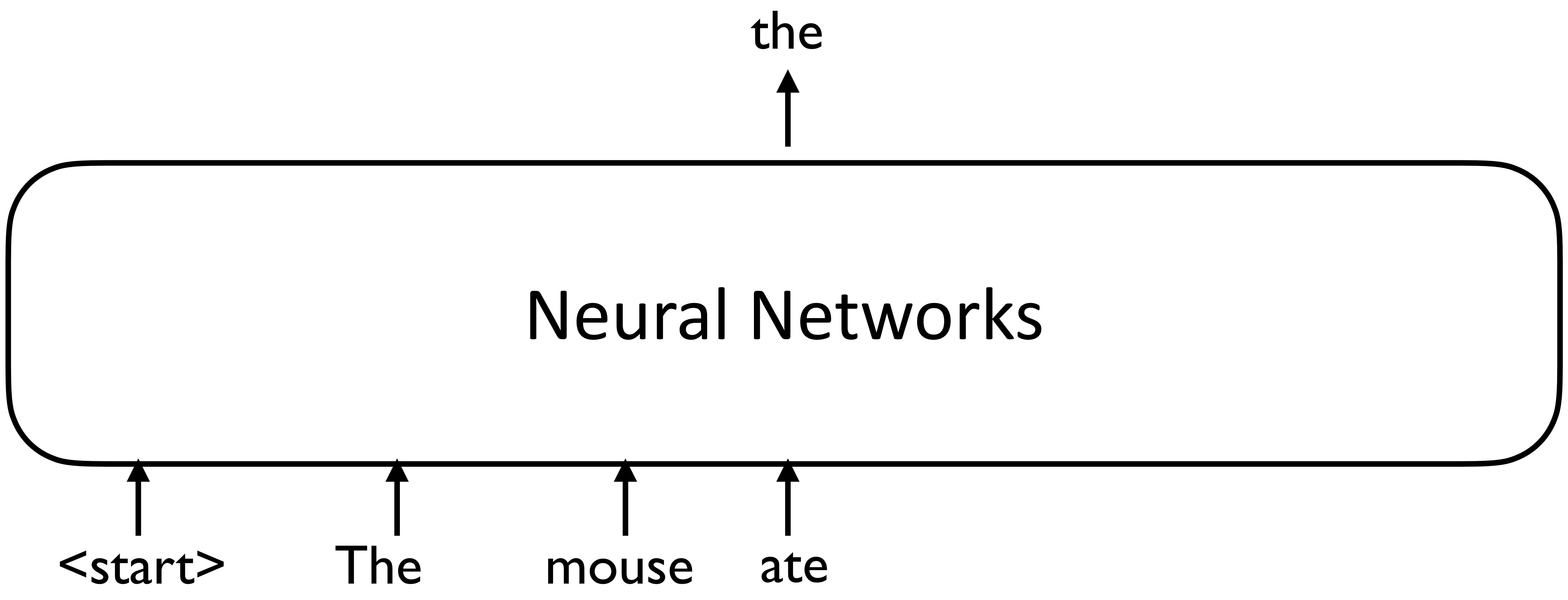
Data: "The mouse ate the cheese ."



Recap: Neural Language Models

Neural language models are typically autoregressive

Data: "The mouse ate the cheese ."

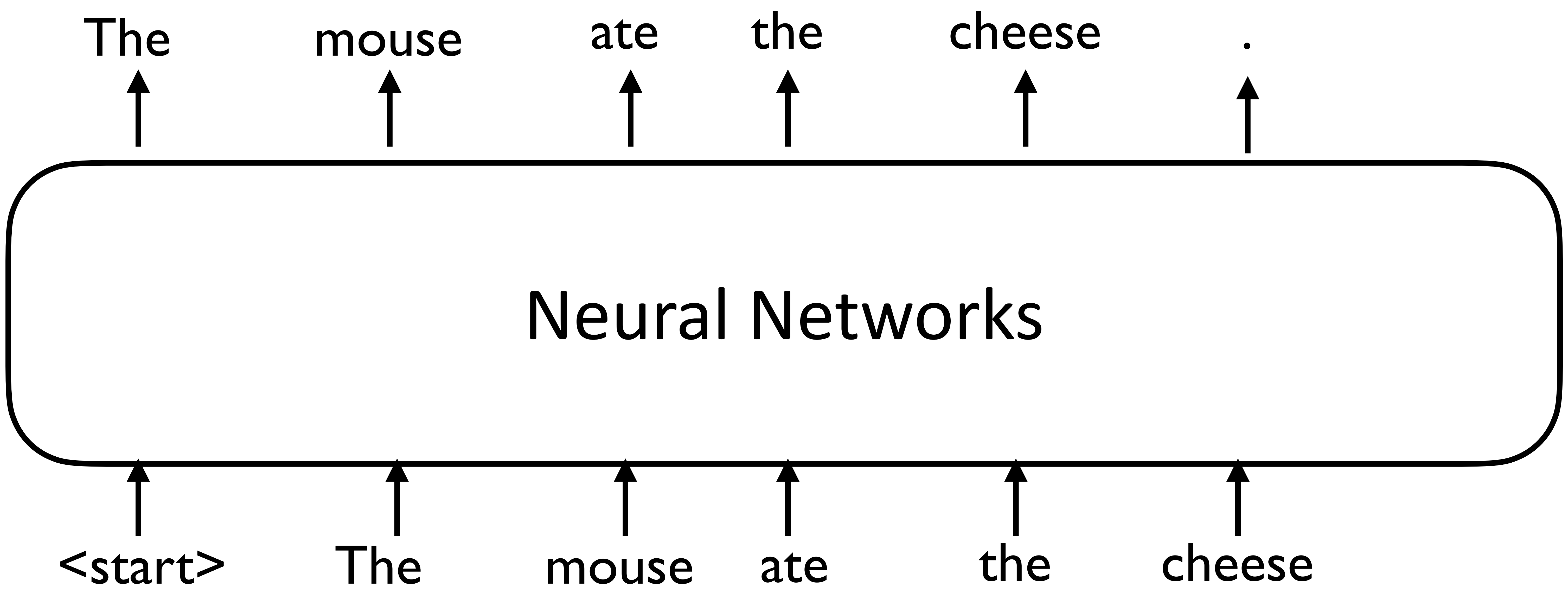


We can compute the loss on every token in parallel

Recap: Neural Language Models

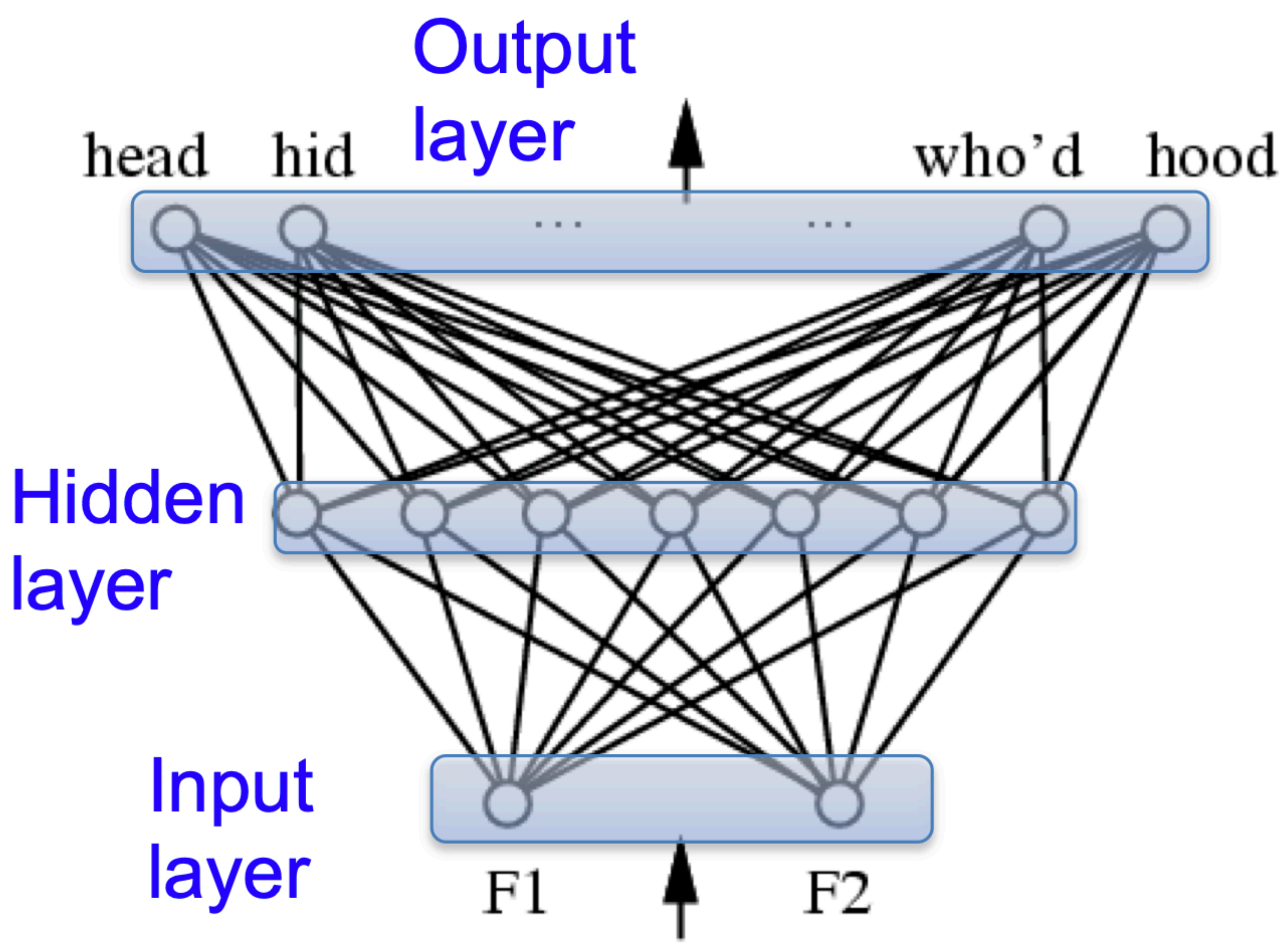
Neural language models are typically autoregressive

Data: "The mouse ate the cheese ."

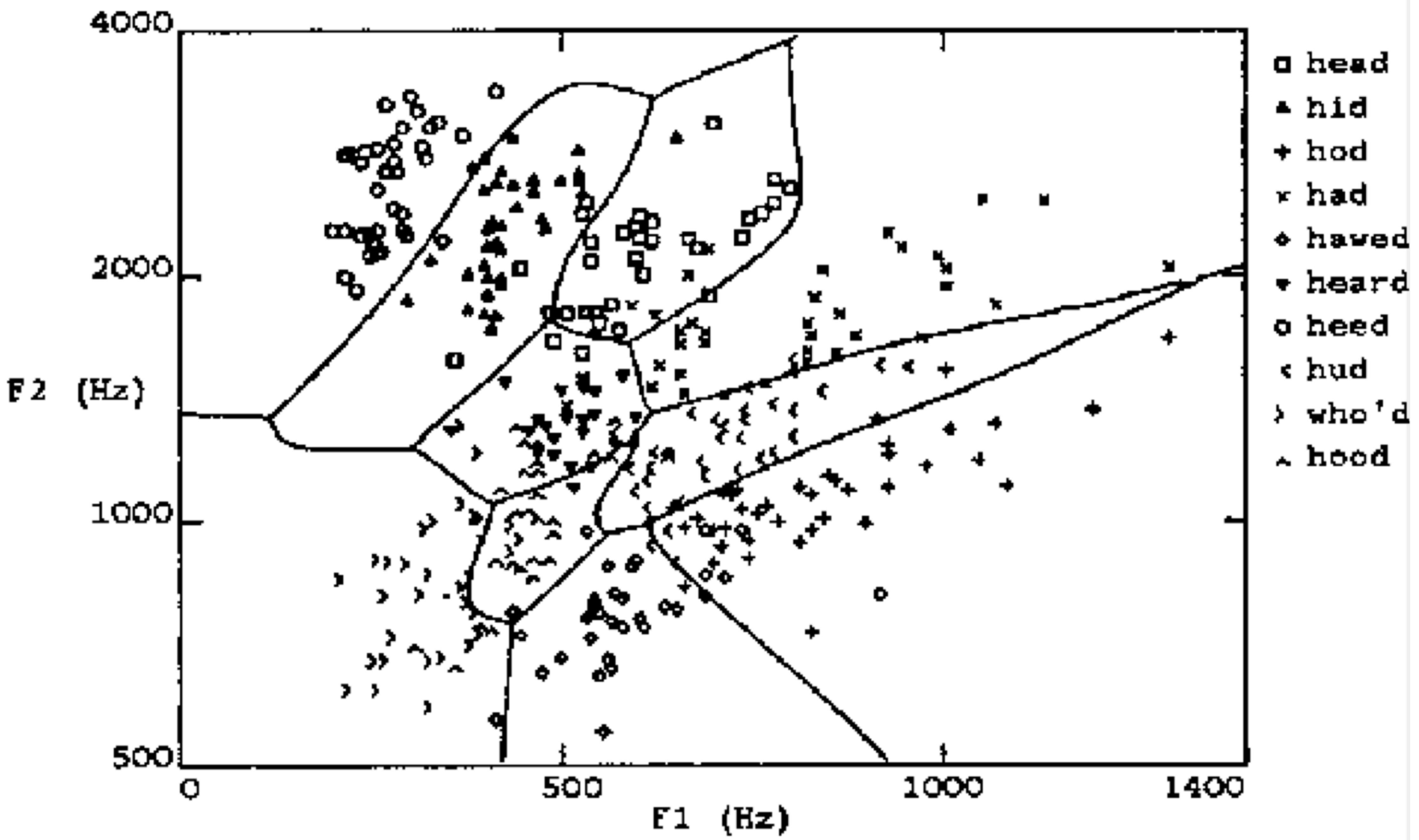


Each prediction only sees the inputs on its left

Recap: Multilayer Networks of Sigmoid Units



Two layers of logistic units

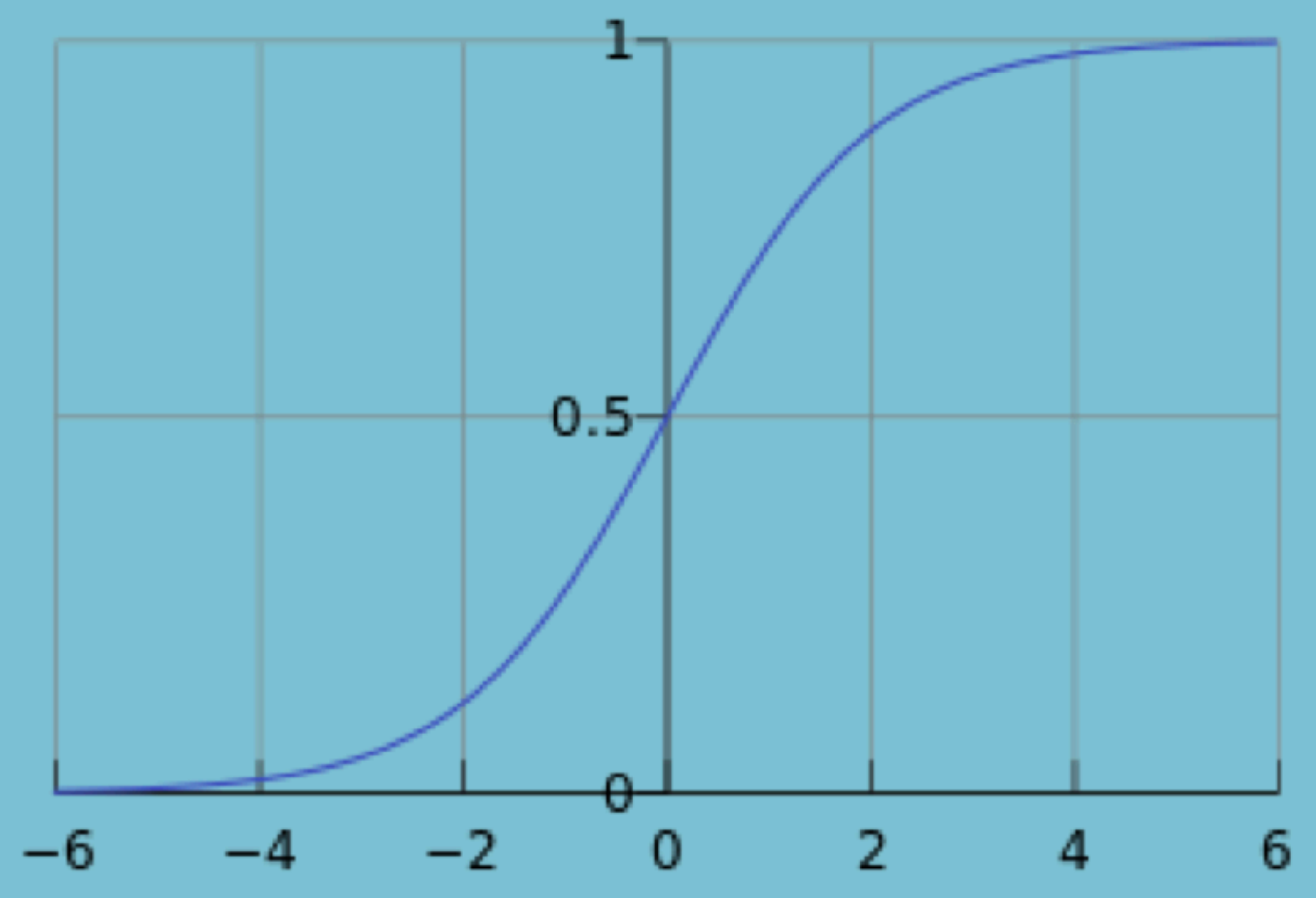


Highly non-linear decision surface

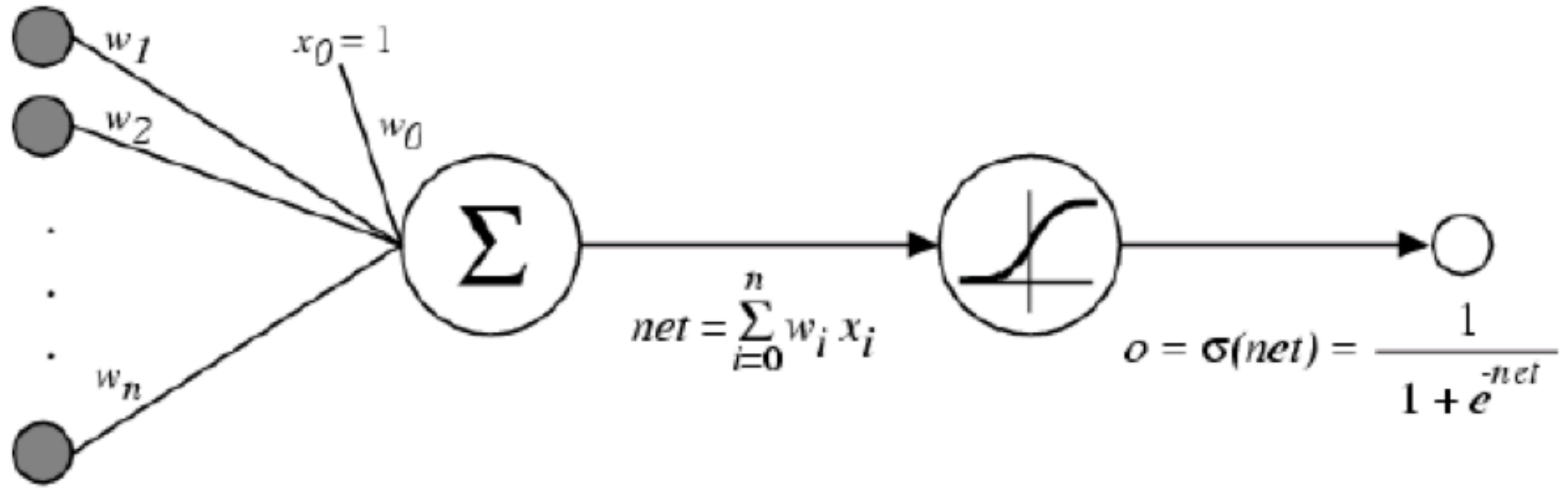
Activation Functions

Sigmoid / Logistic Function

$$\text{logistic}(u) \equiv \frac{1}{1+e^{-u}}$$

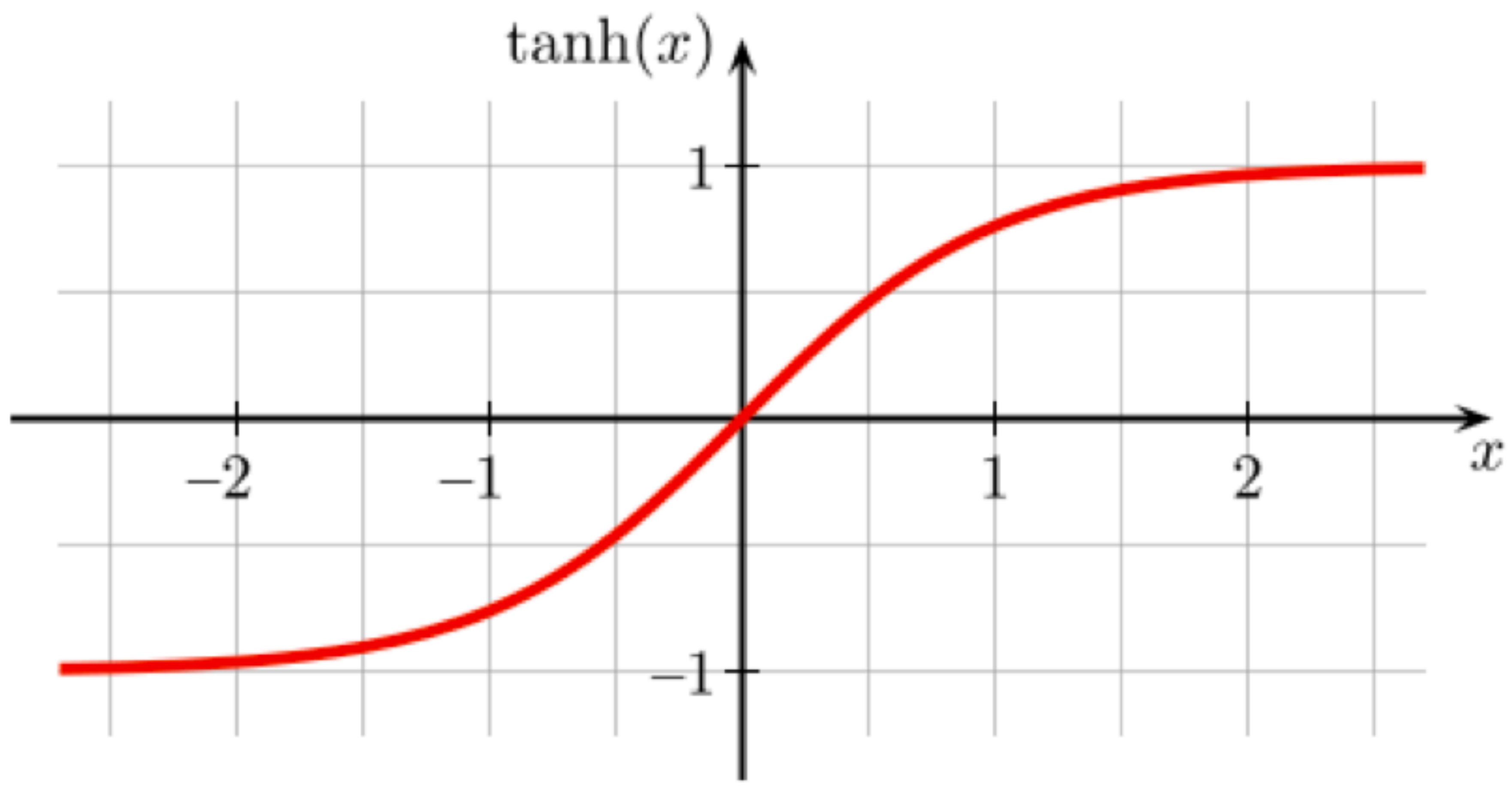


So far, we've assumed that the activation function (nonlinearity) is always the sigmoid function…



Tanh

- A new change: modifying the nonlinearity
  - The logistic is not widely used in modern ANNs



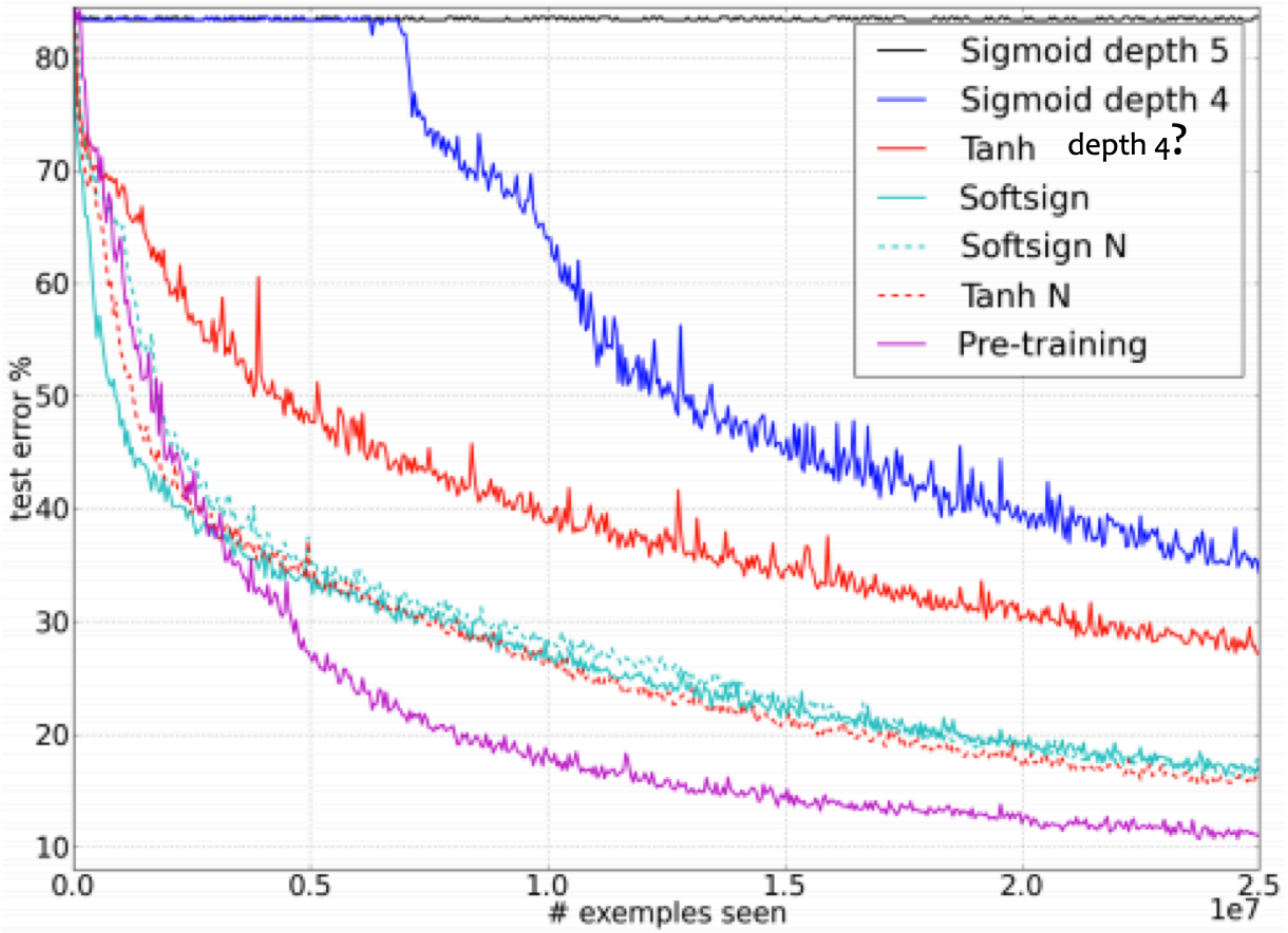
Alternate 1:
tanh

Like logistic function but shifted to range [-1, +1]

Activation Function

Understanding the difficulty of training deep feedforward neural networks

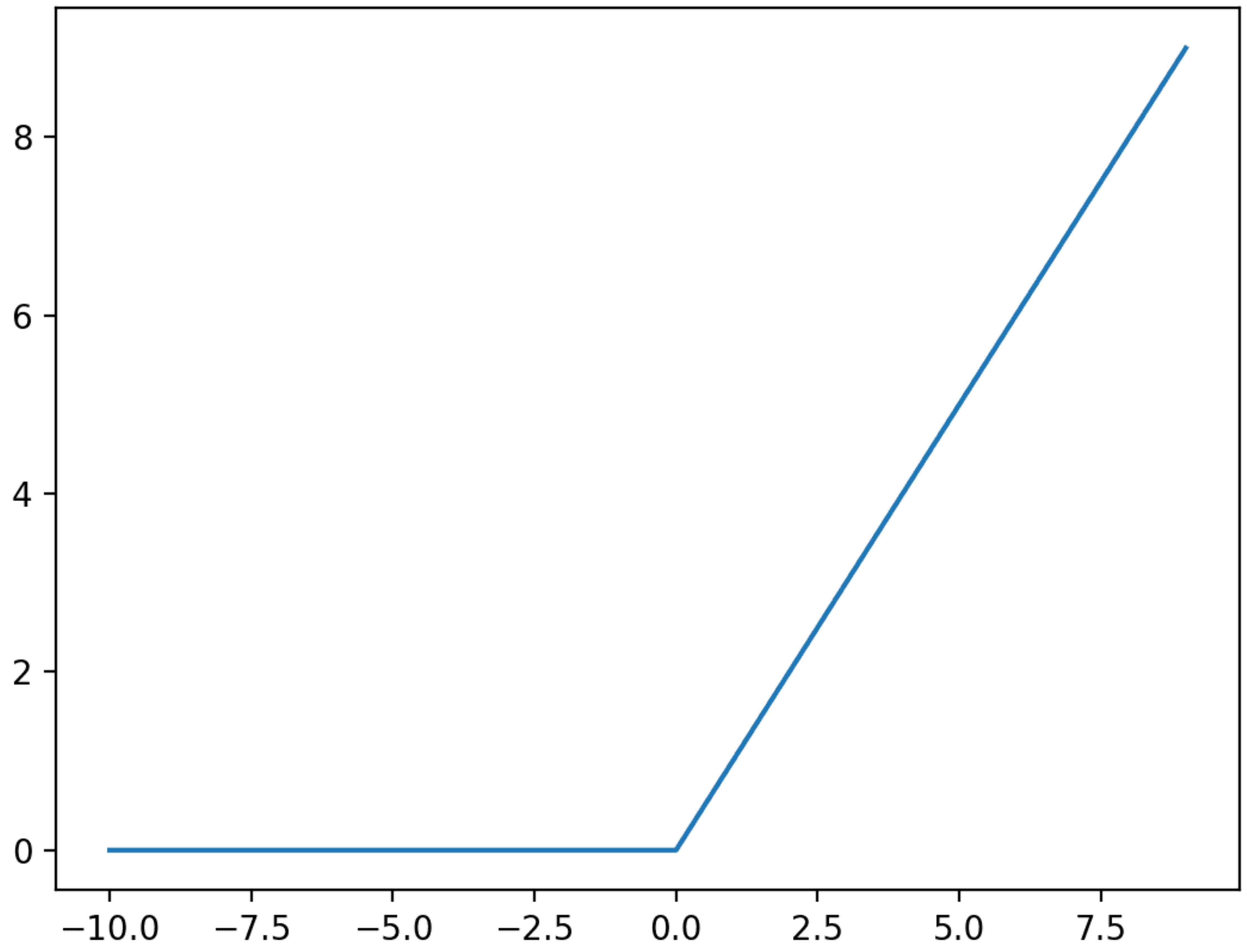
AI Stats 2010



sigmoid
vs.
tanh

Figure from Glorot & Bentio (2010)

ReLU



Other Activation Functions

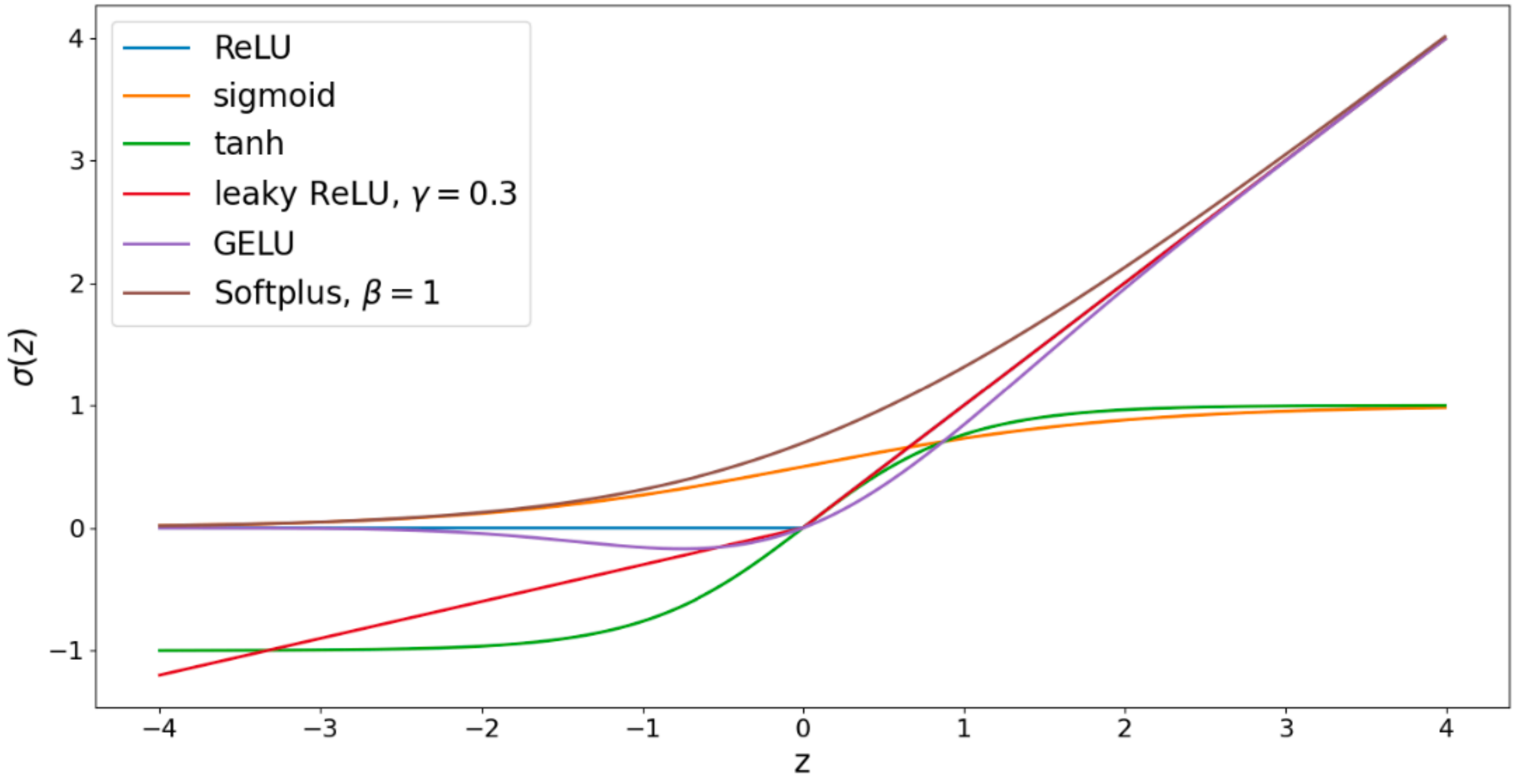
$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad \text{(sigmoid)}$$

$$\sigma(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \qquad \text{(tanh)}$$

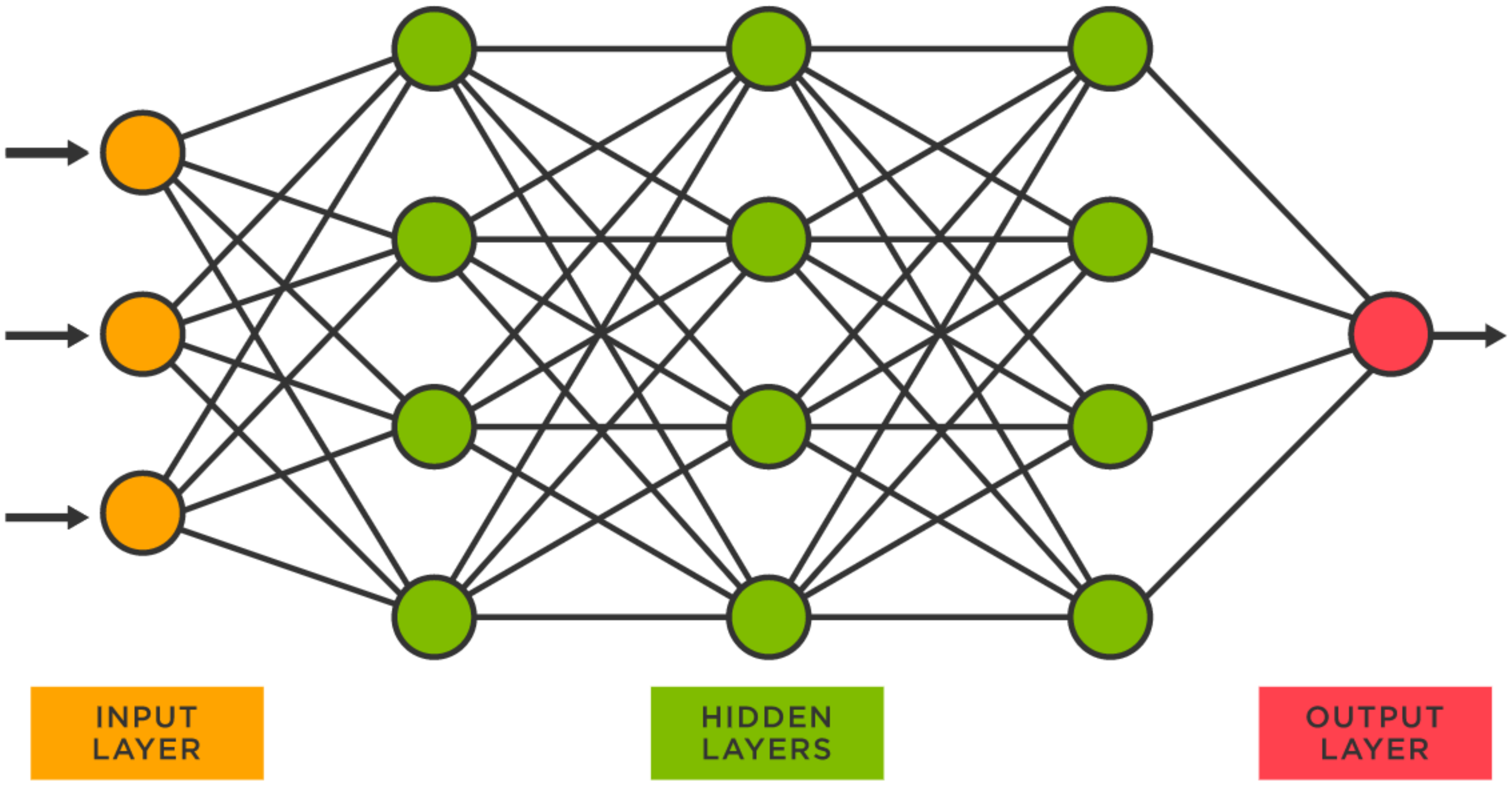
$$\sigma(z) = \max\{z, \gamma z\}, \gamma \in (0,1) \qquad \text{(leaky ReLU)}$$

$$\sigma(z) = \frac{z}{2}\left[1+\mathrm{erf}(\frac{z}{\sqrt{2}})\right] \qquad \text{(GELU)}$$

$$\sigma(z) = \frac{1}{\beta}\log(1+\exp(\beta z)), \beta > 0 \qquad \text{(Softplus)}$$



Multilayer Perceptron Neural Networks (MLP)



Residual Connection

We want deeper and deeper NNs, but going deep is difficult

- Troubles accumulate w/ more layers
- Signals get distorted when propagated
- in forward and backward passes

Commonly used techniques to train “Deep” NNs:

Weight initialization

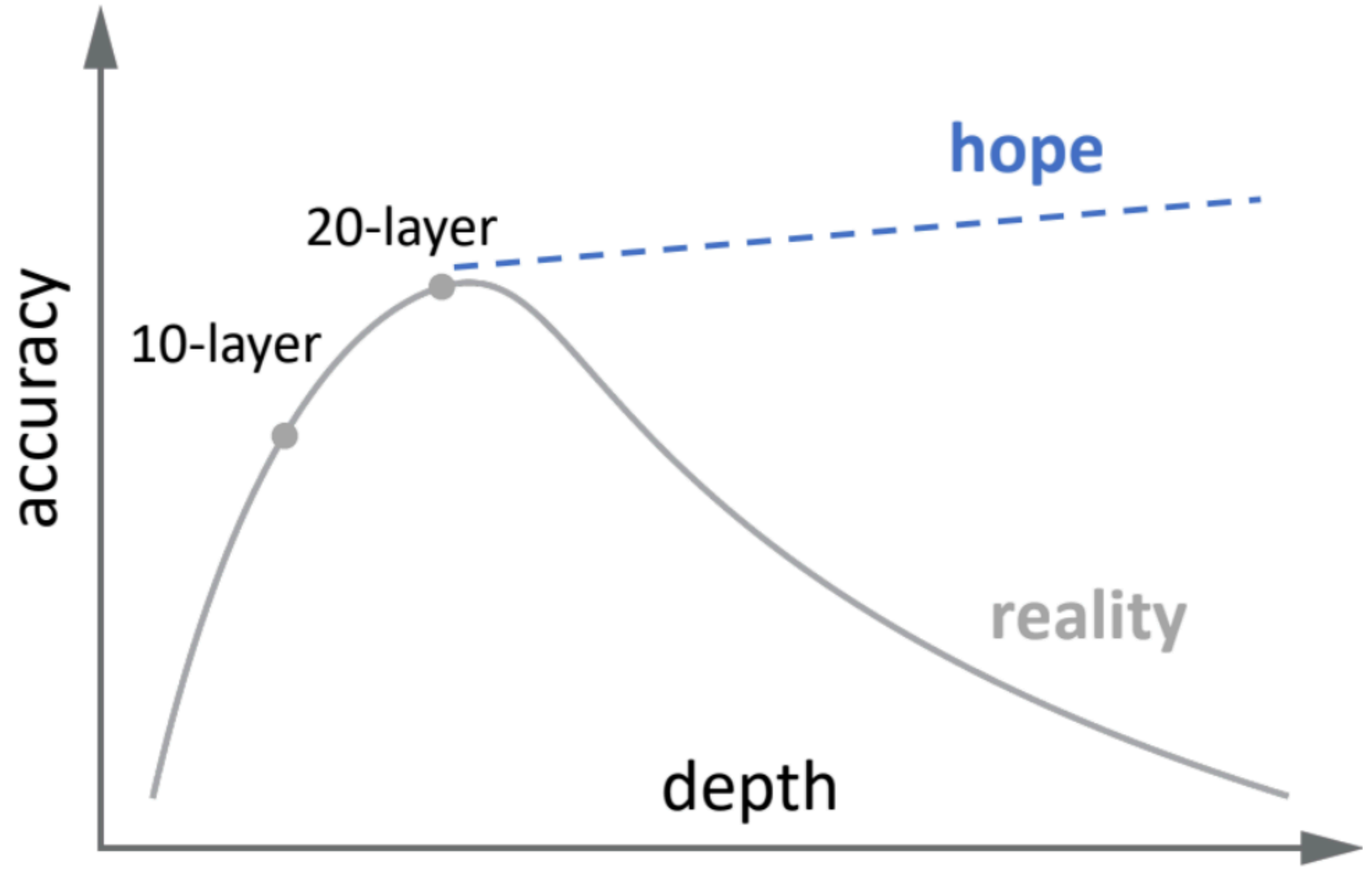
Normalization modules

Deep residual learning

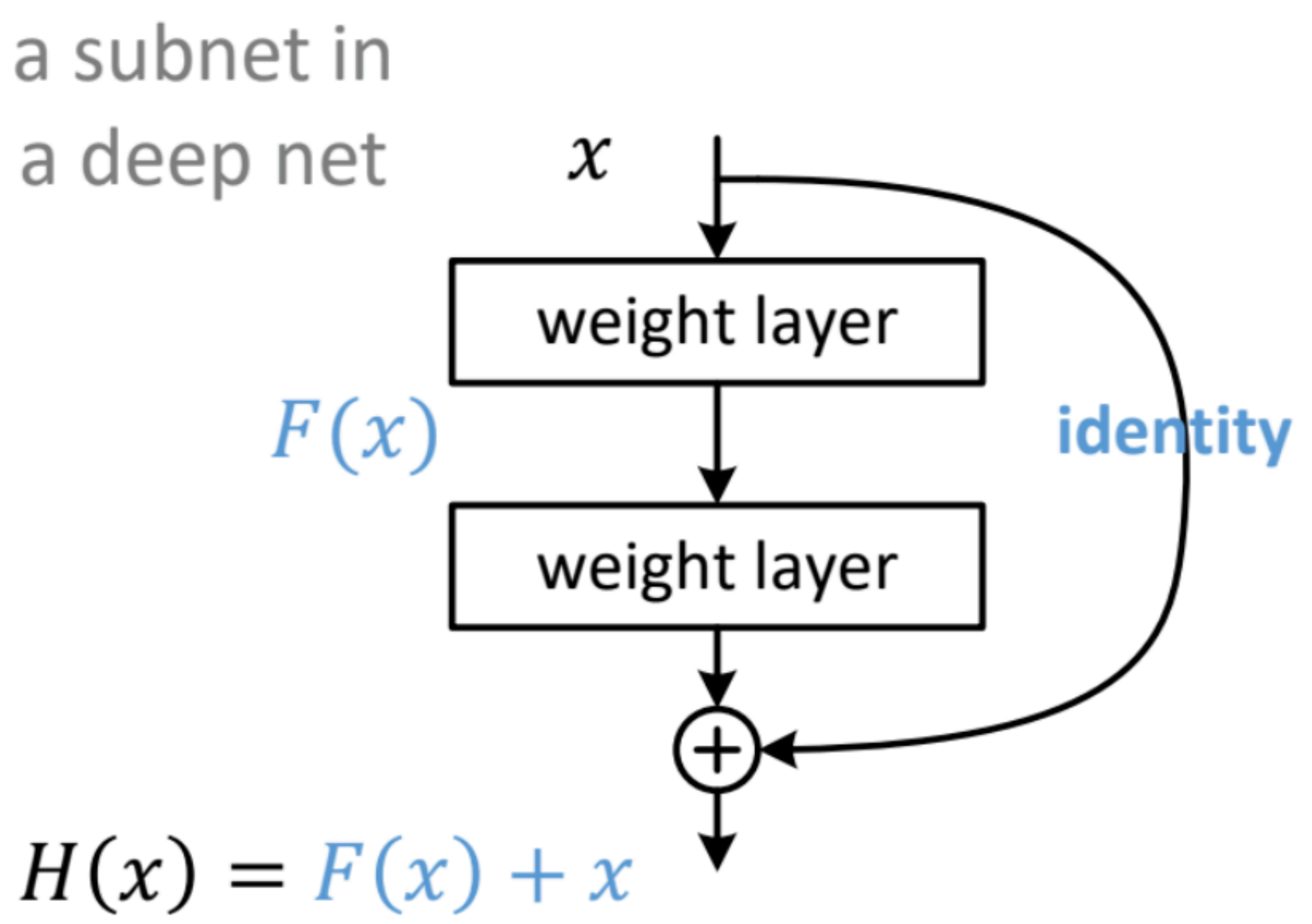
The Degradation Problem

- Good init + norm enable training deeper models
- Simply stacking more layers?

- Degrade after ~20 layers
- Not overfitting
- Difficult to train

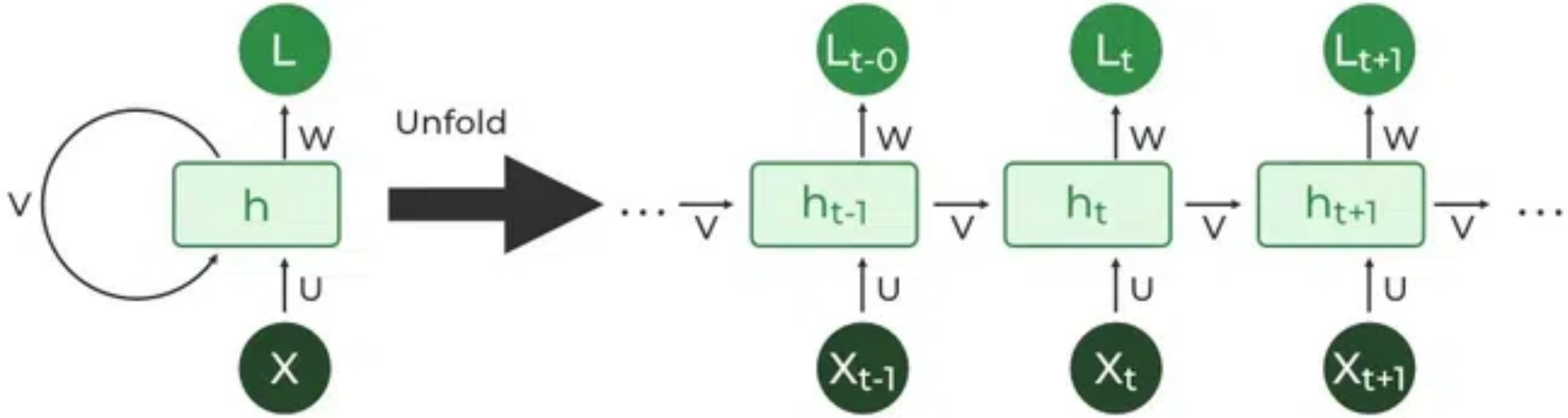


Deep Residual Learning



MLP network is hard to handle sequence data with varying length

Recurrent Neural Networks (RNNs)

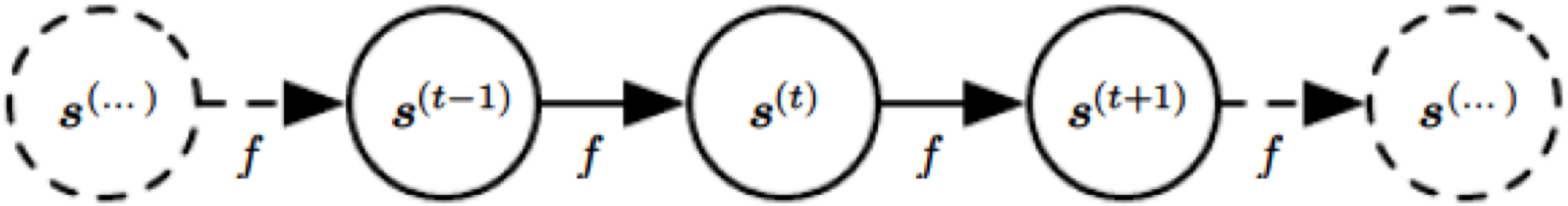


Recurrent Neural Networks

- Dates back to (Rumelhart *et al.*, 1986)
- A family of neural networks for handling sequential data, which involves variable length inputs or outputs

- Especially, for natural language processing (NLP)

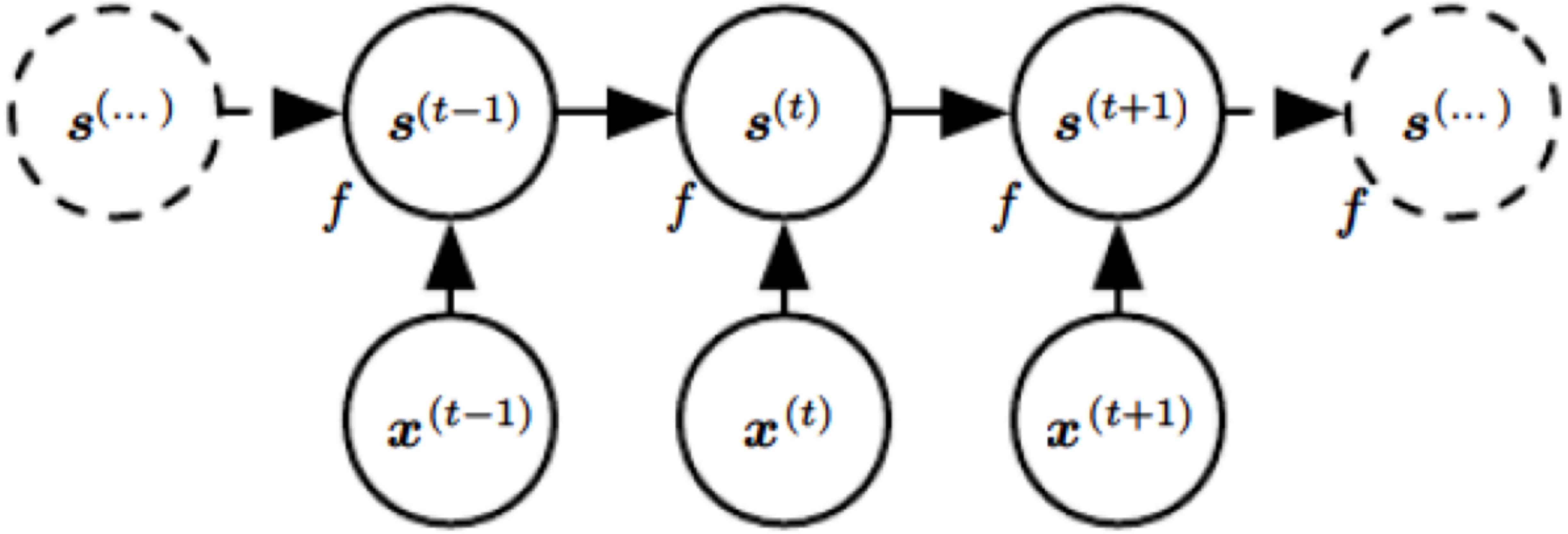
Computation Graph



$$s^{(t+1)} = f(s^{(t)}; \theta)$$

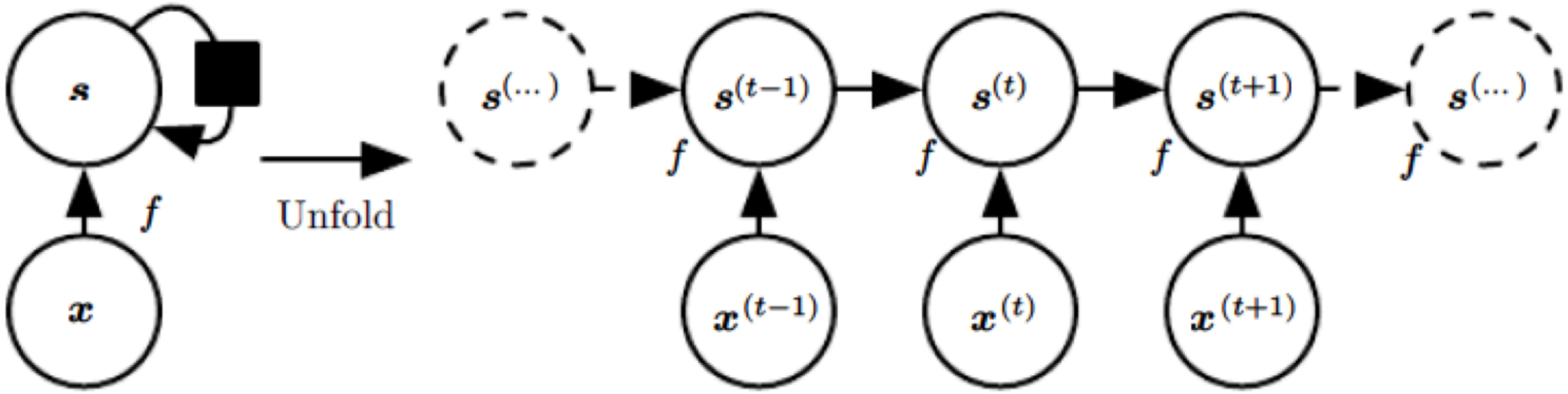
Figure from *Deep Learning*,
Goodfellow, Bengio and Courville

Computation Graph



$$s^{(t+1)} = f(s^{(t)}, x^{(t+1)}; \theta)$$

Compact view



$$s^{(t+1)} = f(s^{(t)}, x^{(t+1)}; \theta)$$

Key: the same f and θ for all time steps

Recurrent Neural Networks

- Use the same computational function and parameters across different time steps of the sequence
- Each time step: takes the input entry and the previous hidden state to compute the output entry
- Loss: typically computed every time step

Recurrent Neural Networks

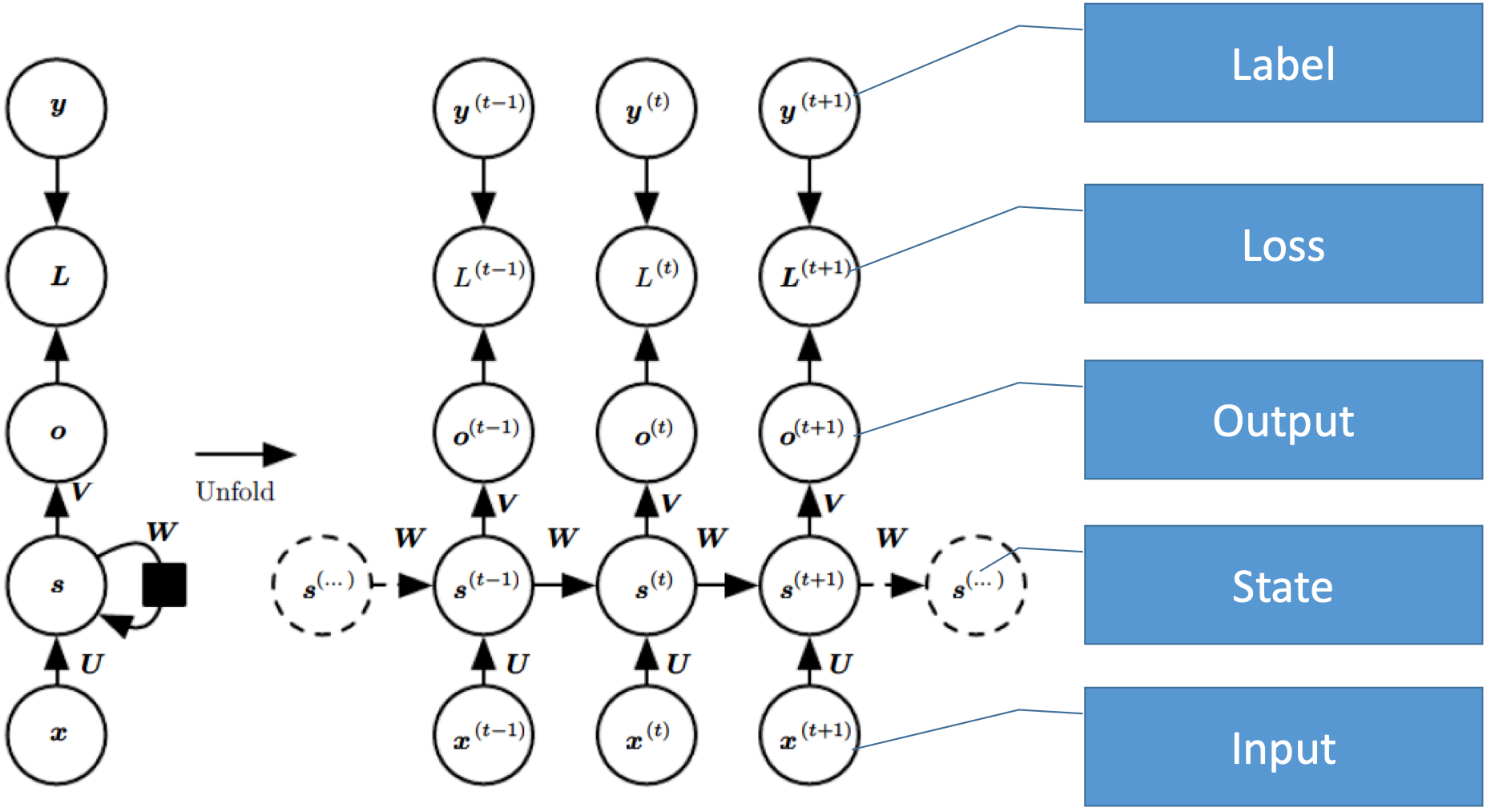
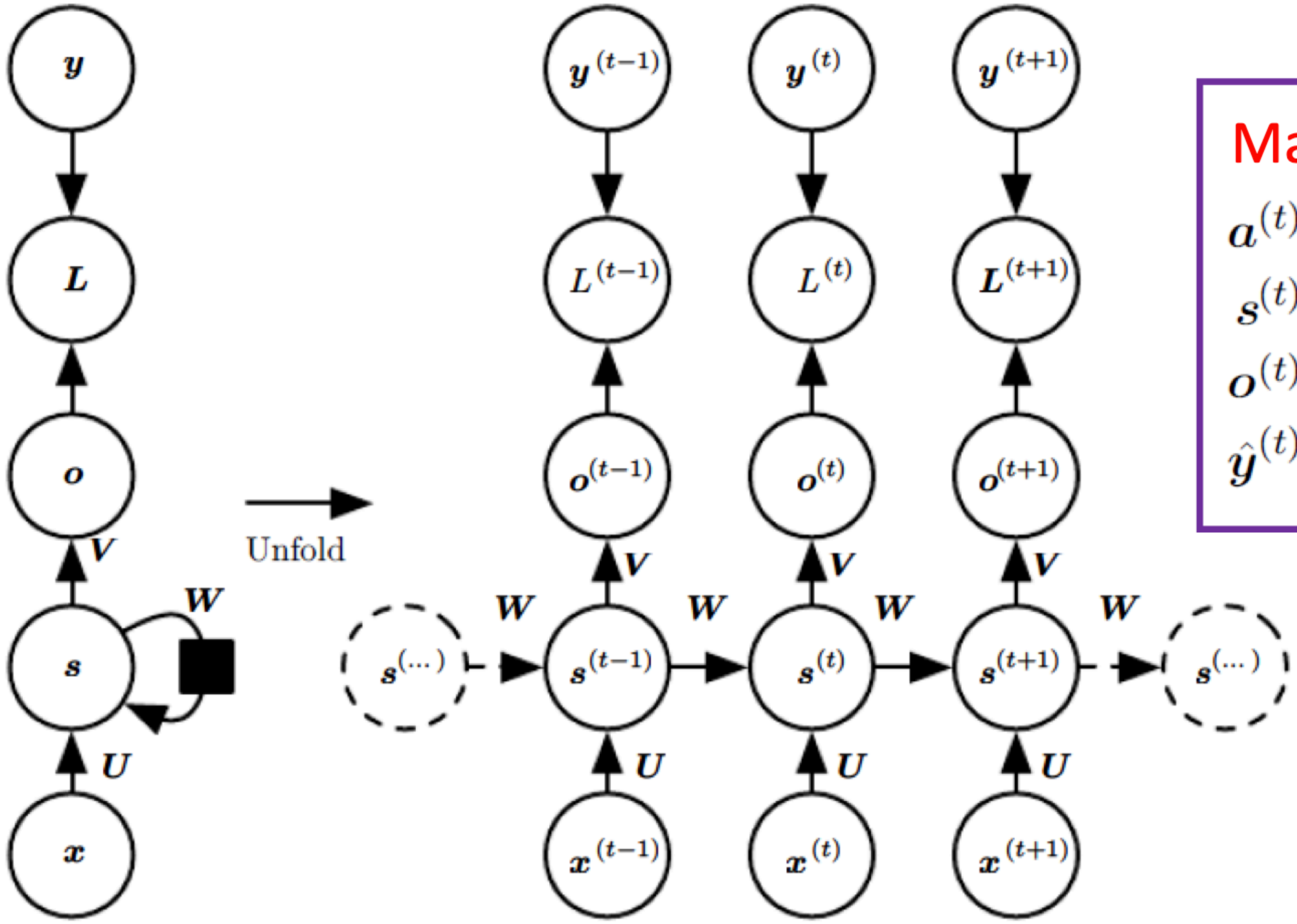


Figure from *Deep Learning*, by Goodfellow, Bengio and Courville

Recurrent Neural Networks



Math formula:

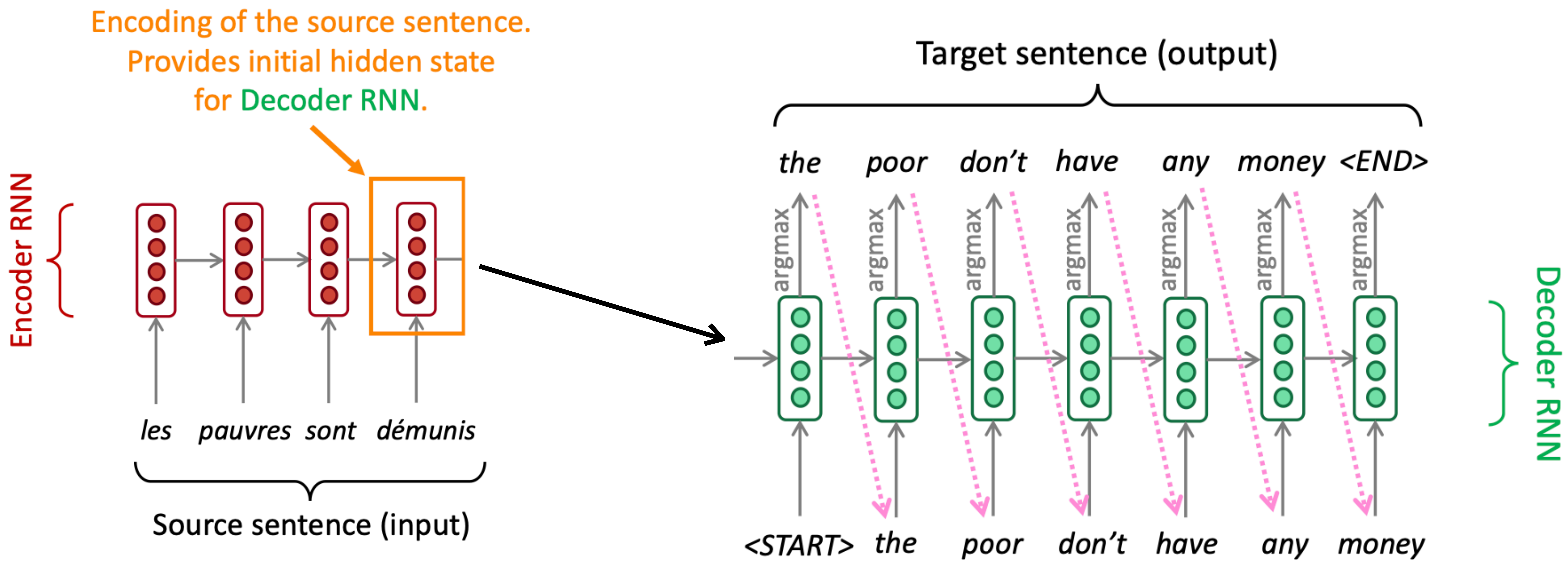
$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{s}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{s}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{s}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

Figure from *Deep Learning*, Goodfellow, Bengio and Courville

There are many variants of RNNs since the functional form to compute $s^{(t)}$ can vary, e.g., LSTM

Sequence-to-Sequence Learning

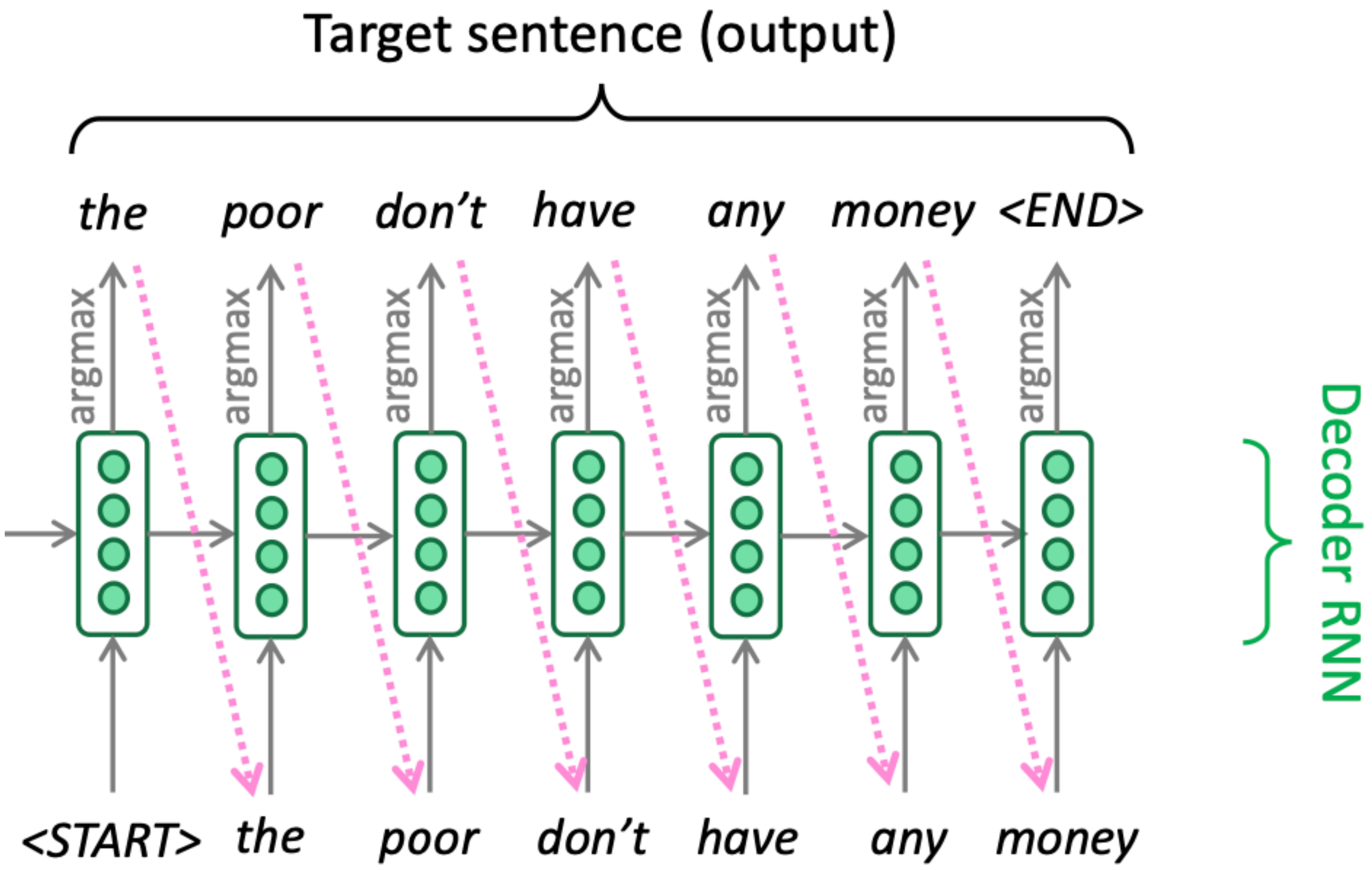
Example of Neural Machine Translation



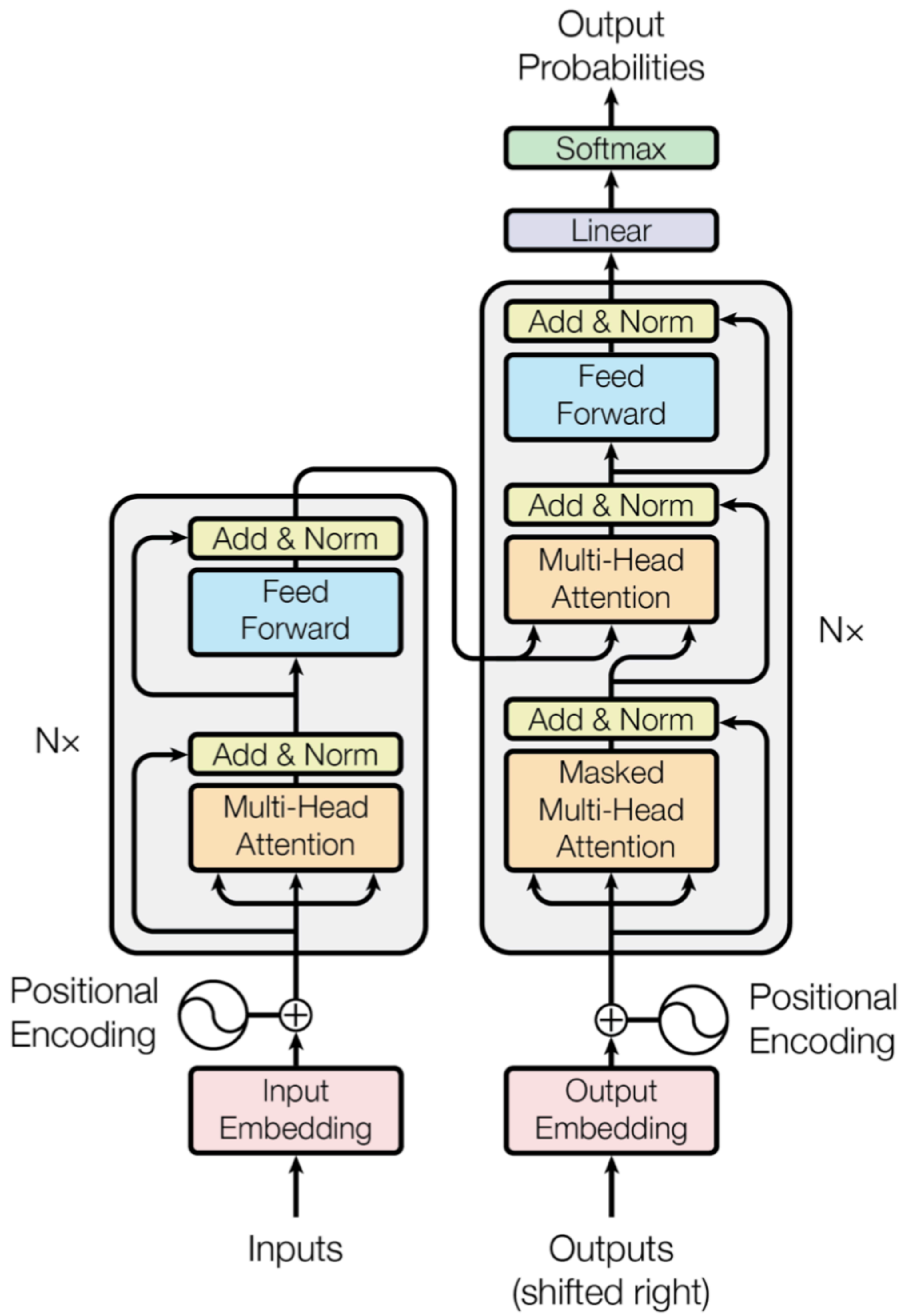
Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence conditioned on encoding.

RNN Language Model

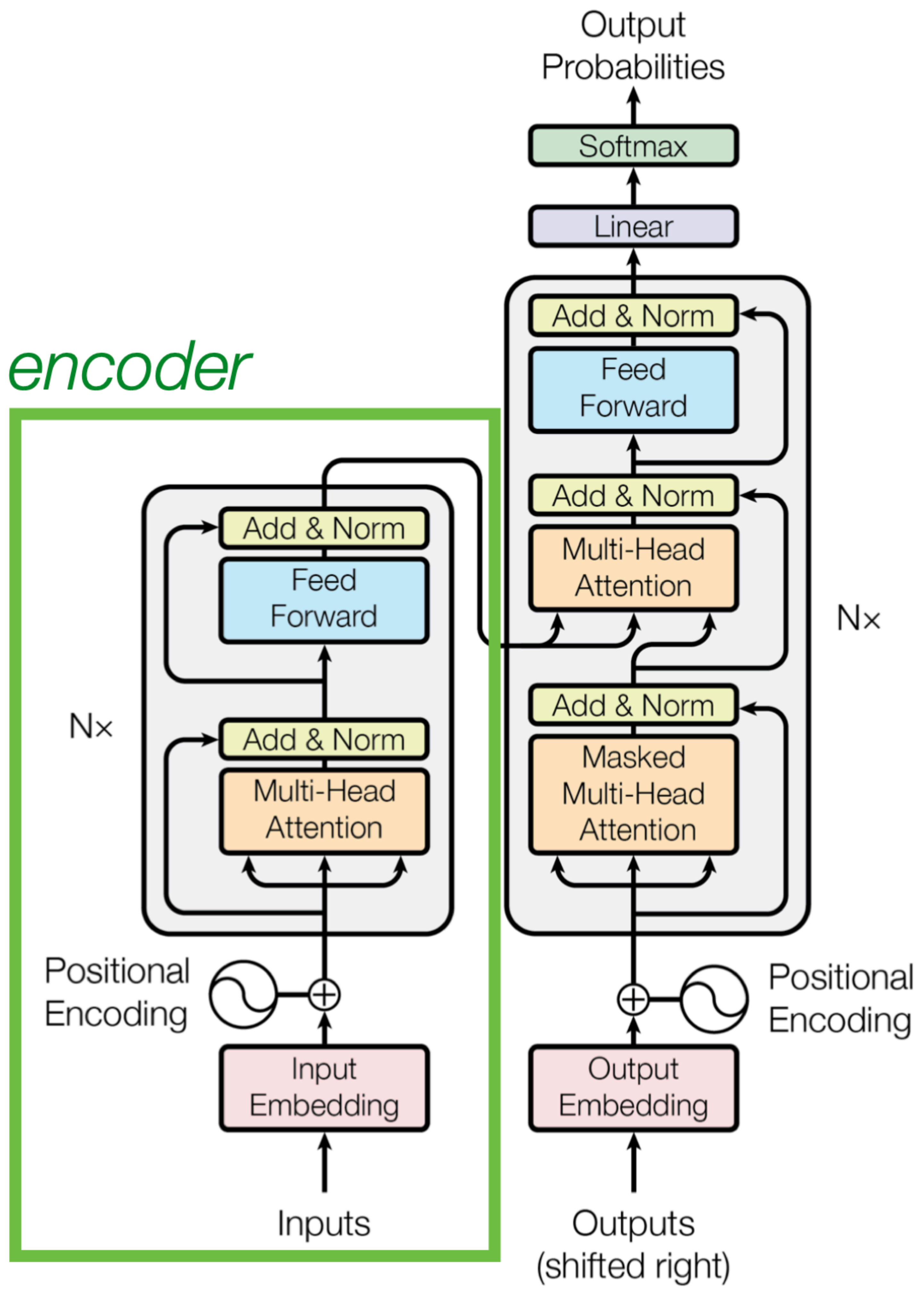


Transformer



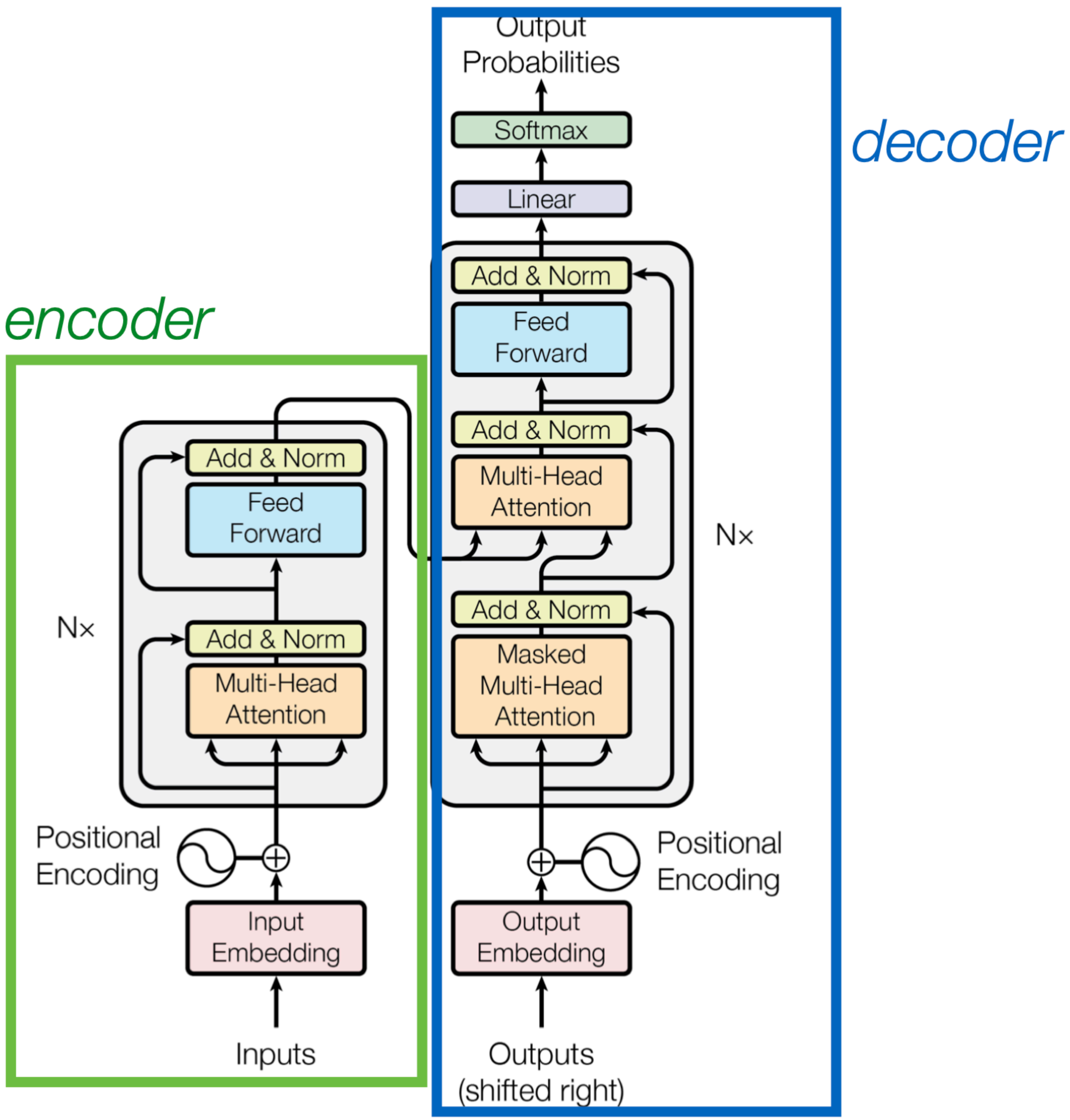
Vaswani et al. Attention is All You Need. NeurIPS 2017.

Encoder



encoder
Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Add & Norm
Masked
Multi-Head
Attention
Add & Norm
Feed
Forward
Nx
Add & Norm
Multi-Head
Attention
Positional
Encoding
Input
Embedding
Inputs
Positional
Encoding
Output
Embedding
Outputs
(shifted right)

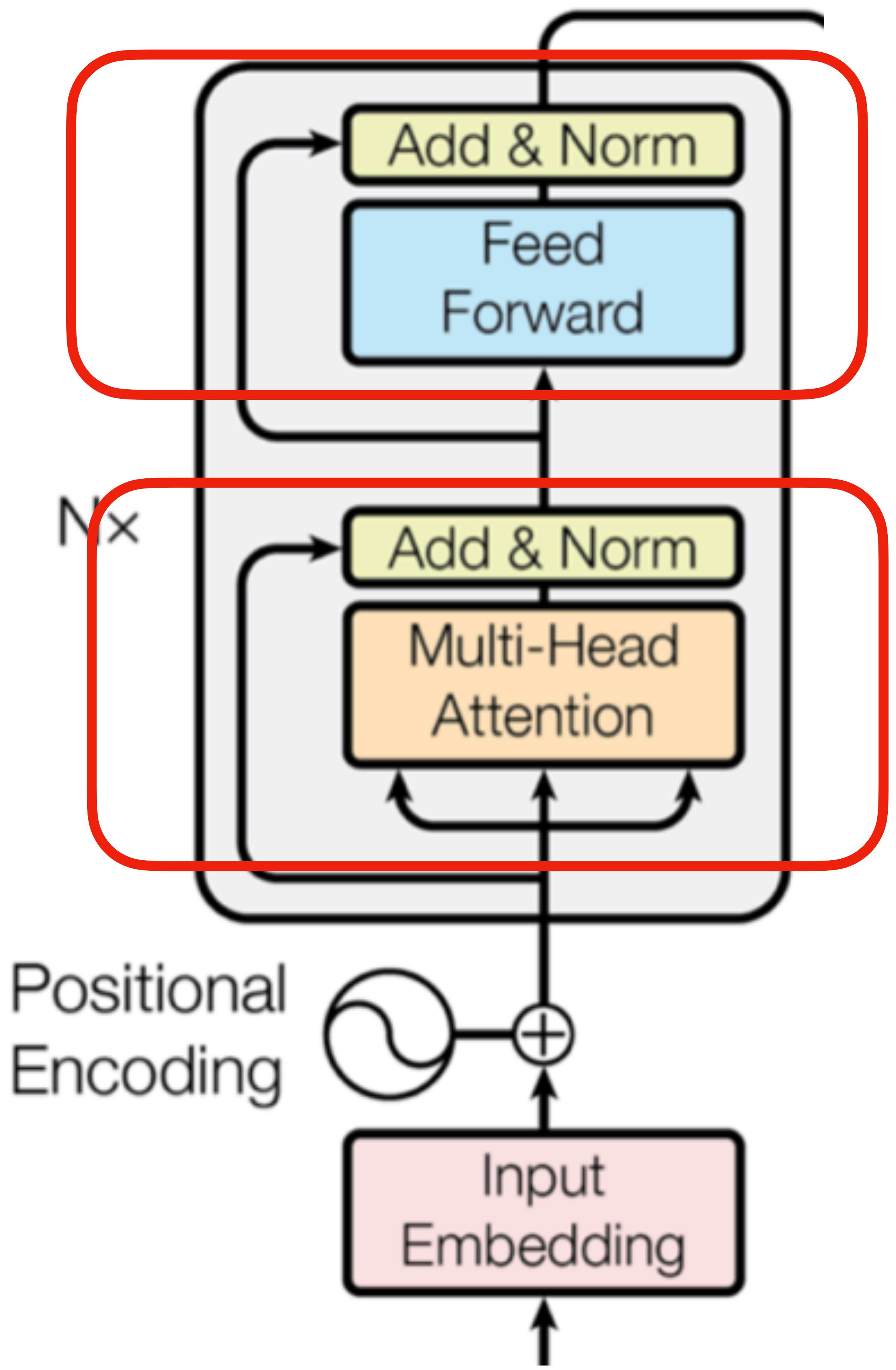
Decoder



encoder
decoder
Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Nx
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Positional
Encoding
Input
Embedding
Inputs
Positional
Encoding
Output
Embedding
Outputs
(shifted right)

Transformer Encoder

Residual connection

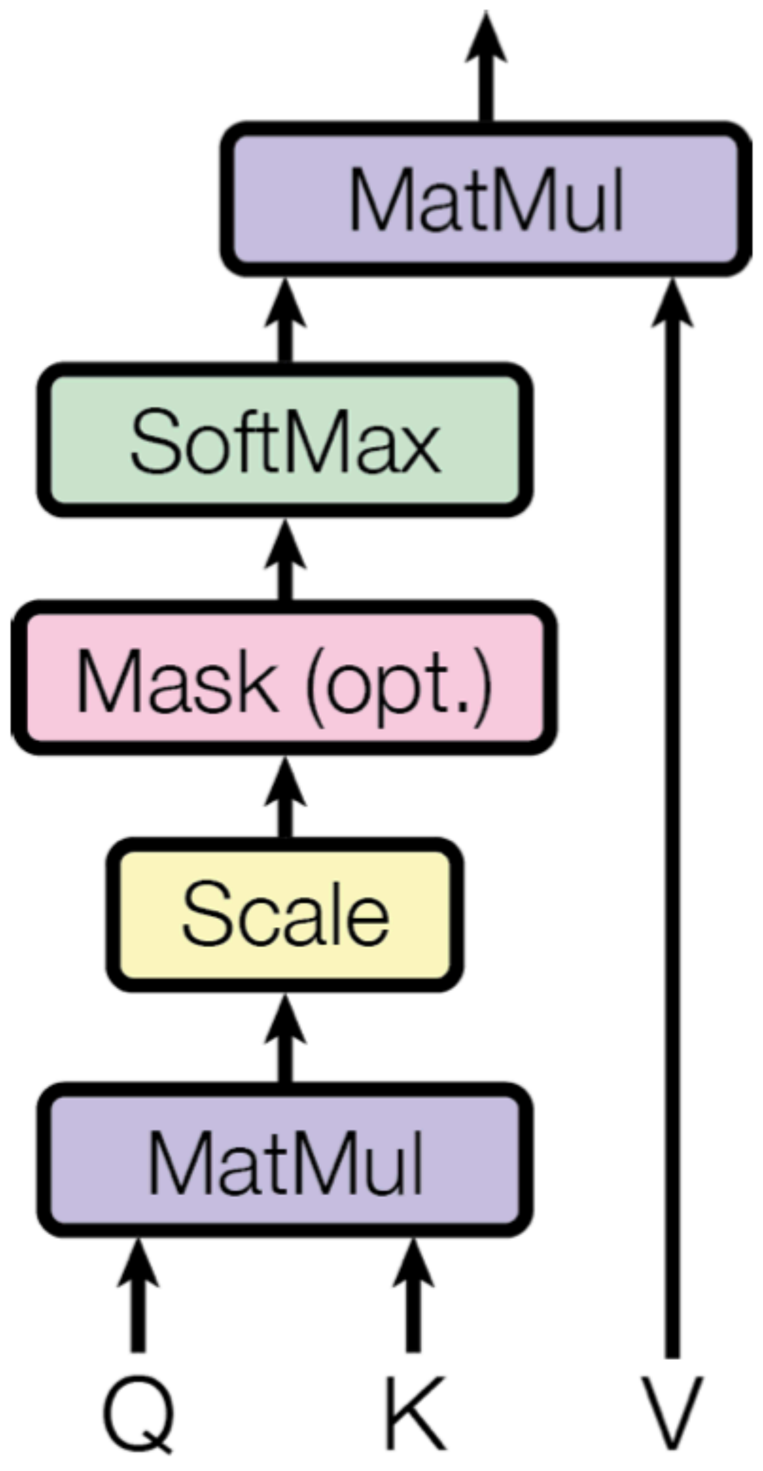


MLP

Self-attention

What is Attention

Scaled Dot-Product Attention



Q: Query
K: key
V: value

$Q \in R^{n \times d}$ $K \in R^{m \times d}$ $V \in R^{m \times d}$

We have n queries, m (key, value) pairs

Attention weight = softmax(QK^T)

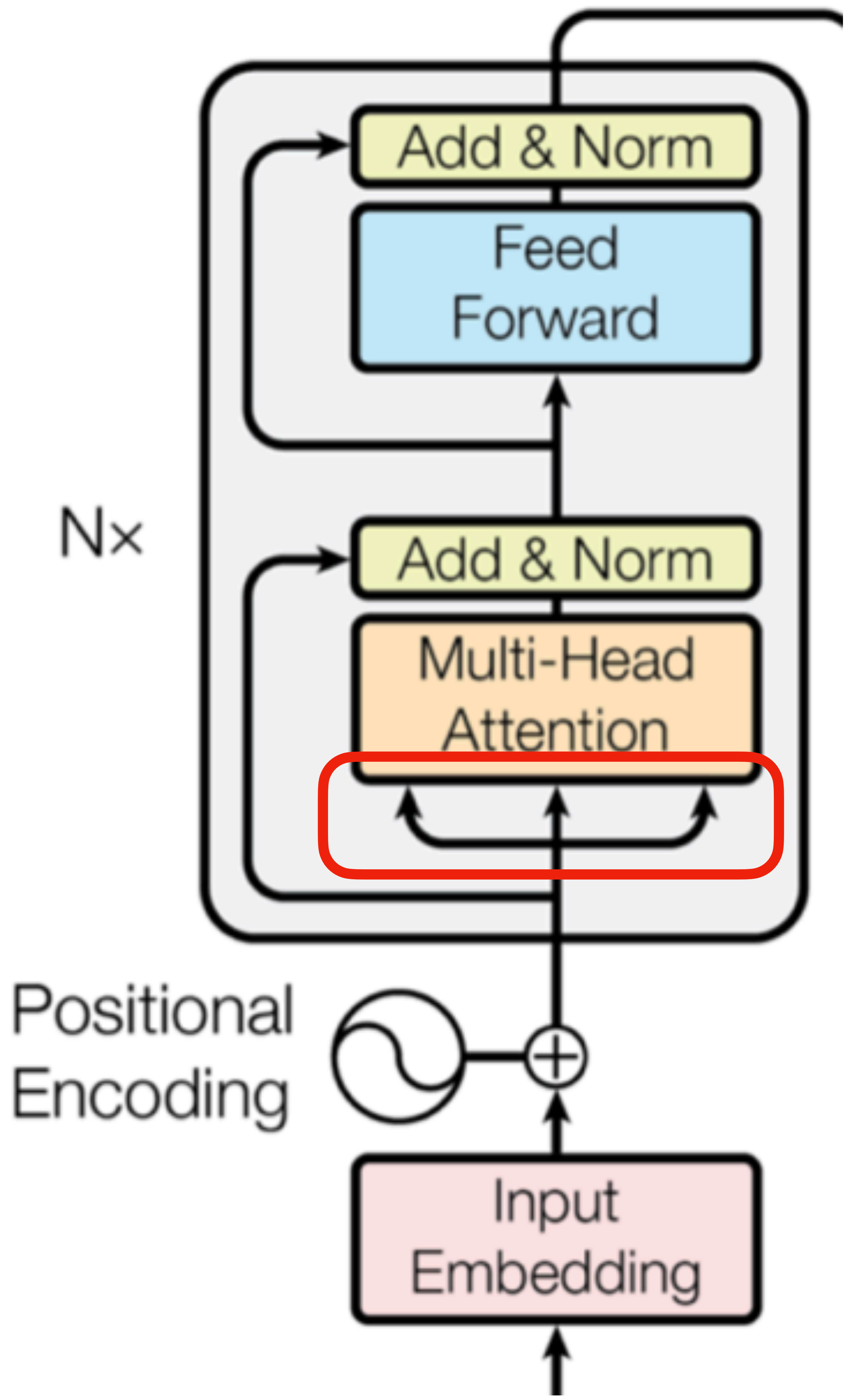
Dot-products grow large in magnitude

Scaled Attention weight = softmax($\frac{QK^T}{\sqrt{d_k}}$) Shape is mxn

Attention weight represents the strength to "attend" values V

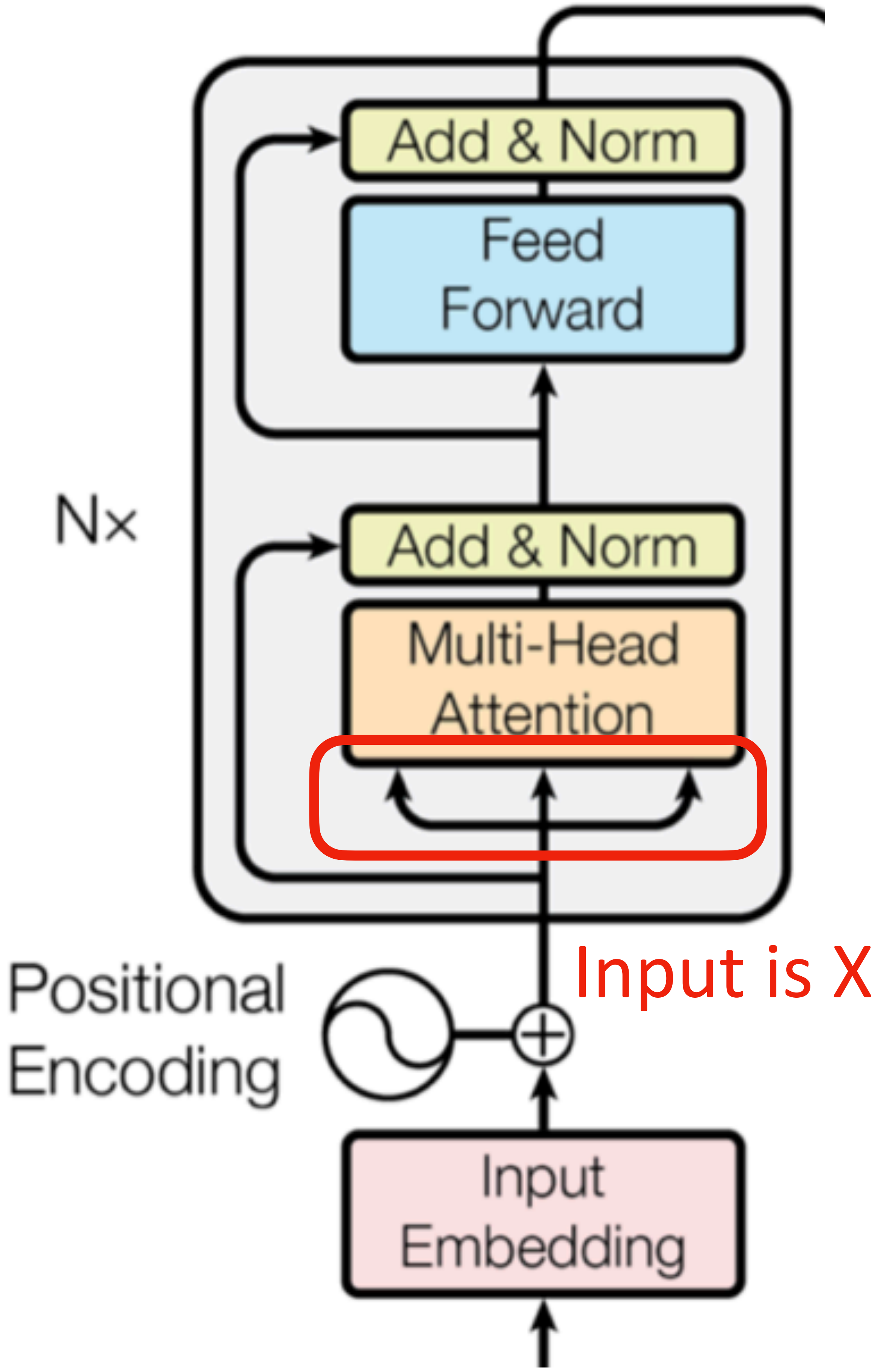
$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

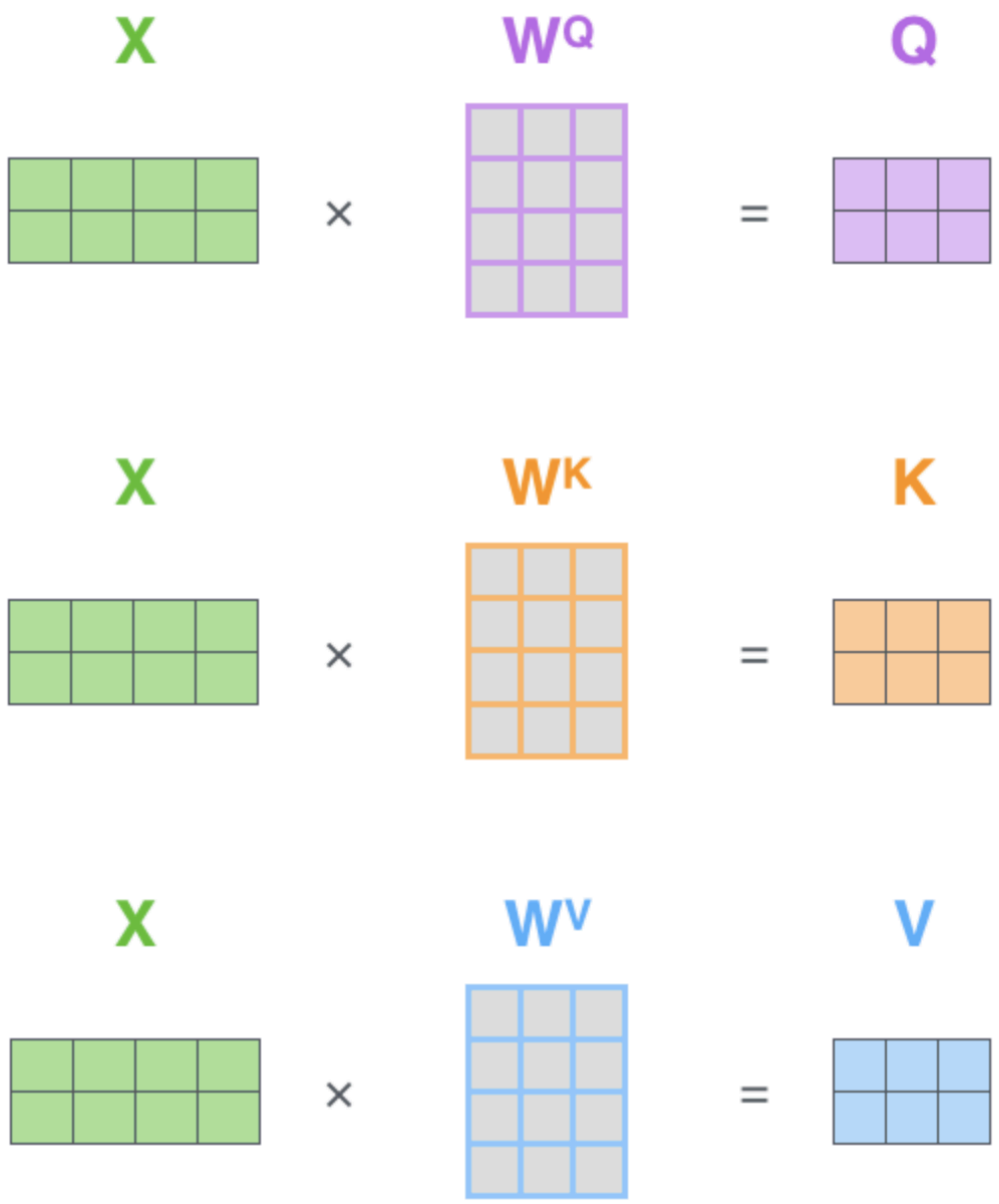
Q, K, V



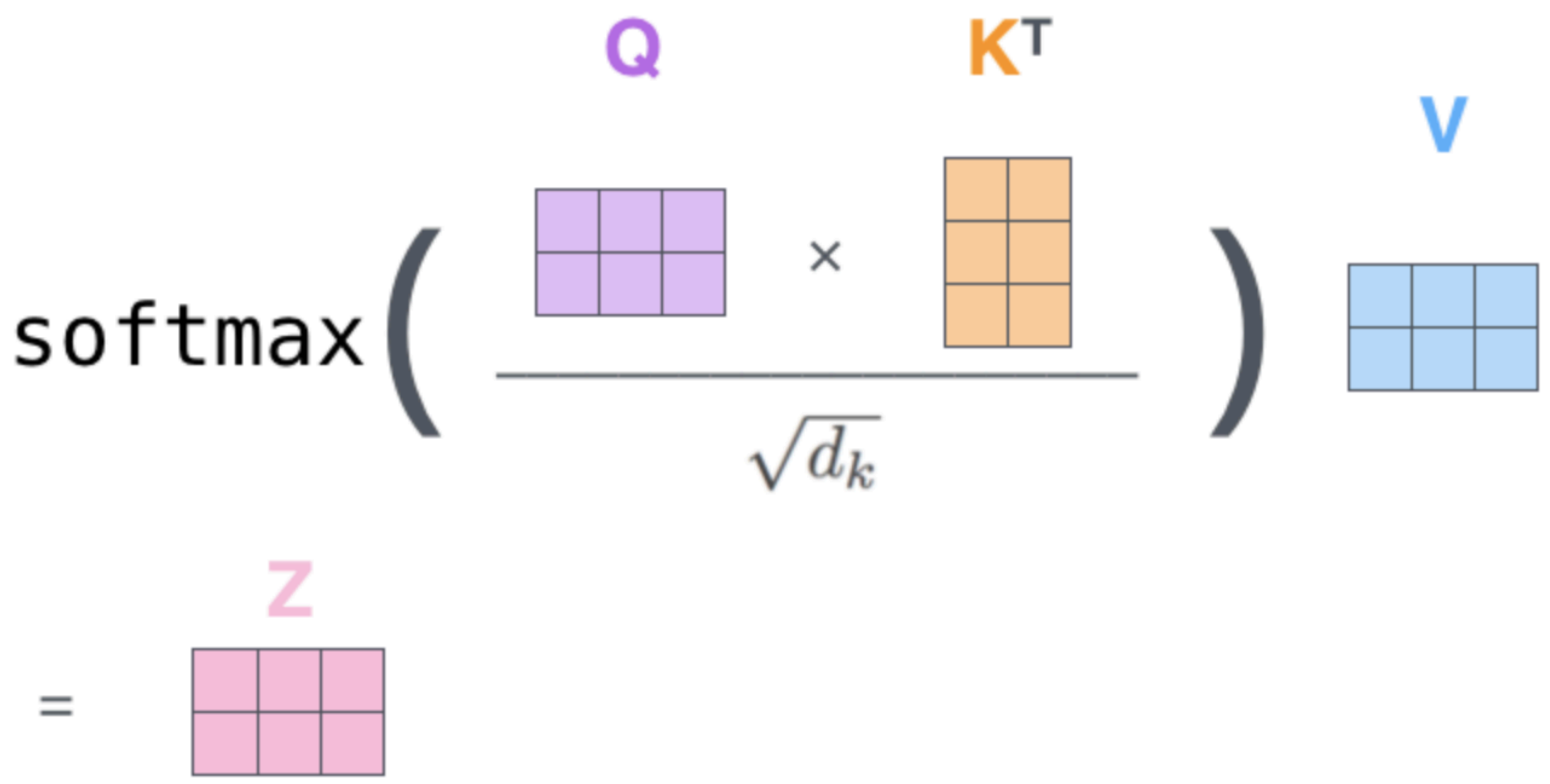
What are Q, K, V in the transformer

Self-Attention





Query, key, and value are from the same input, thus it is called “self”-attention

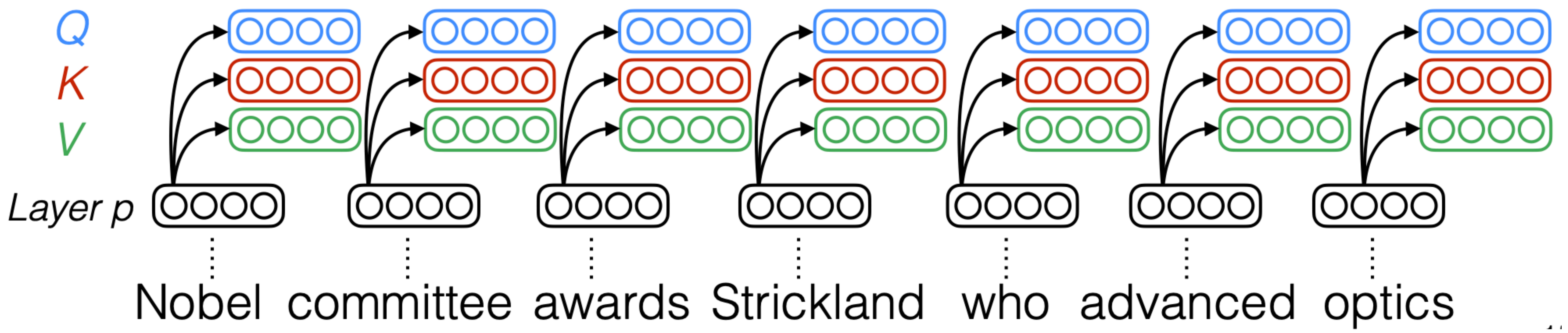


35

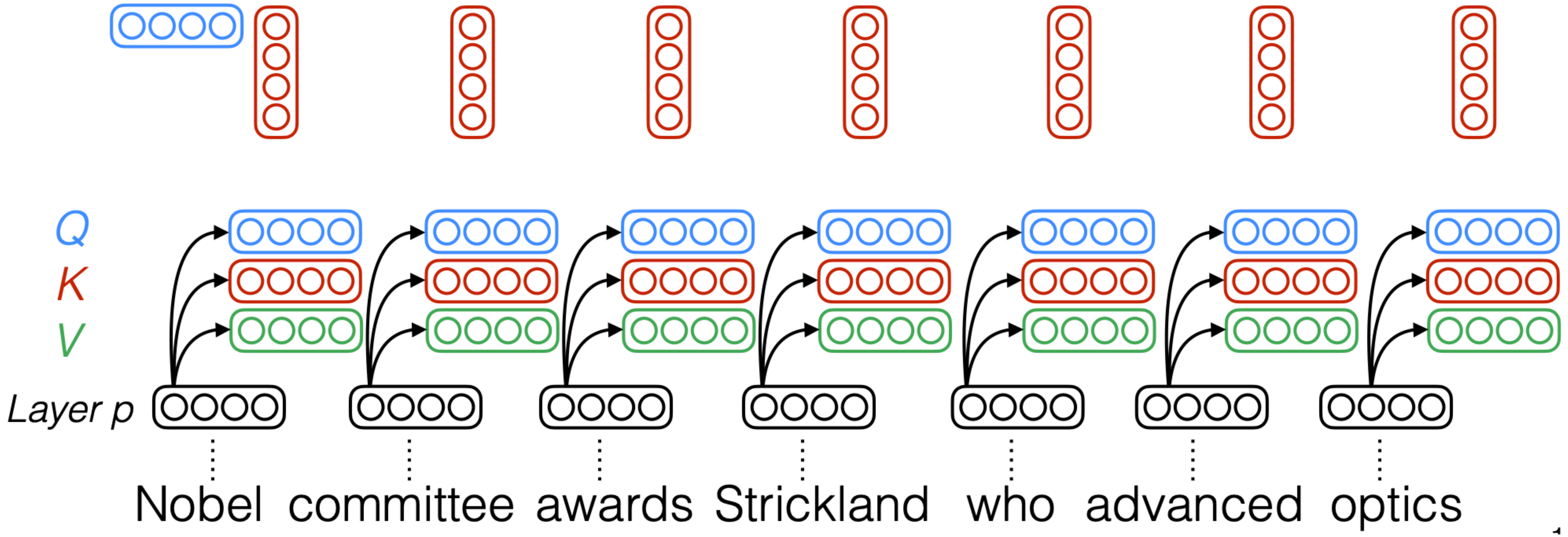
Jay Alammar. The Illustrated Transformer.

Self-Attention

At each step, the attention computation attends to all steps in the input example

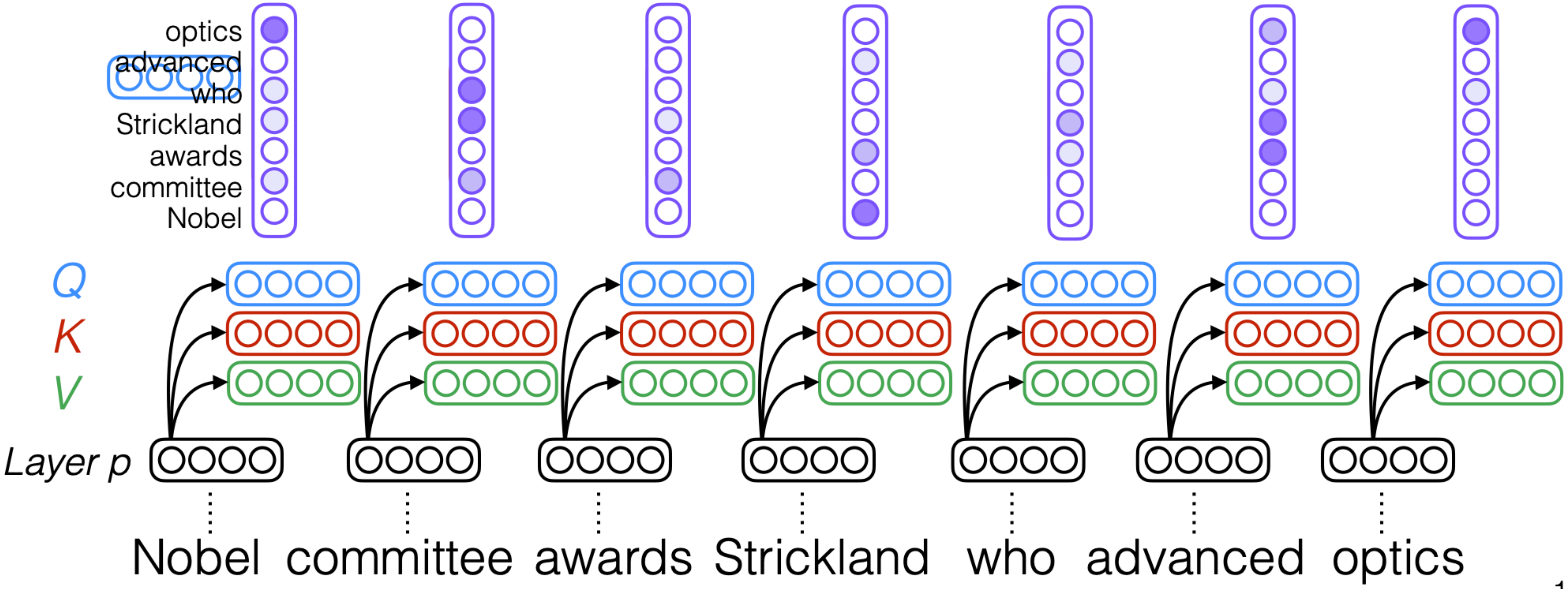


Self-Attention

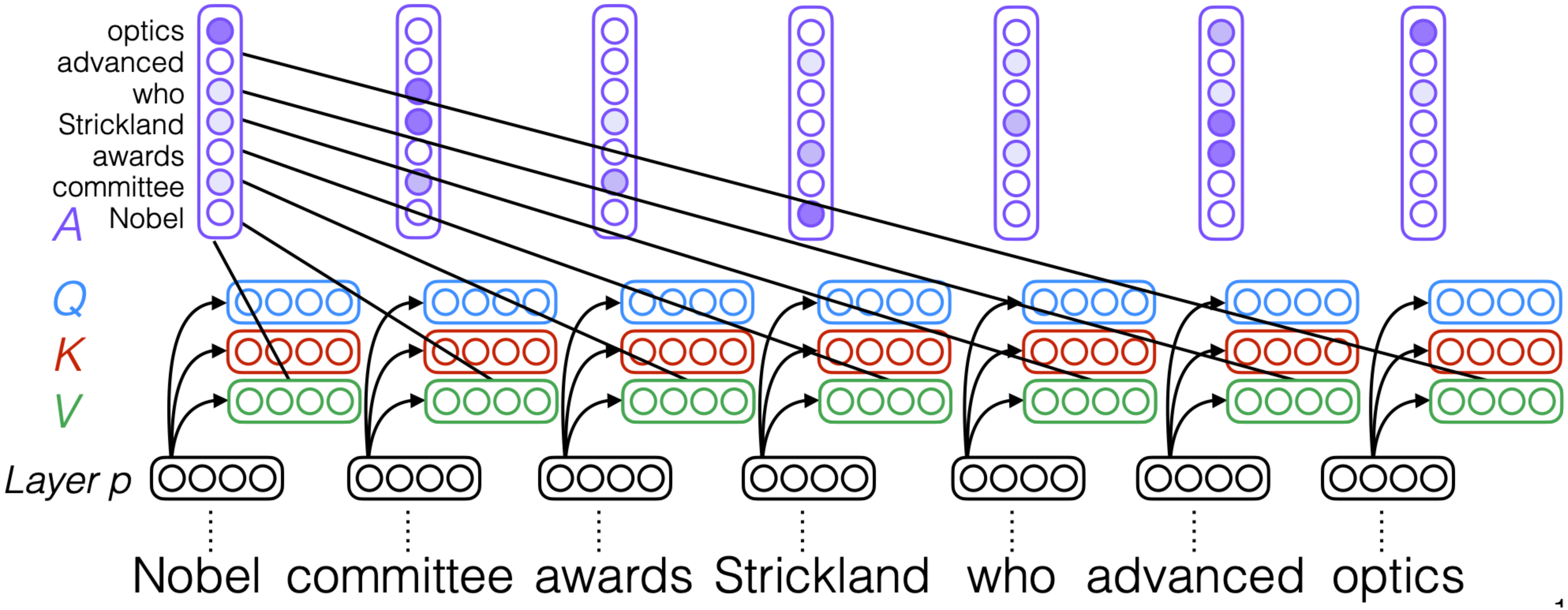


Self-Attention

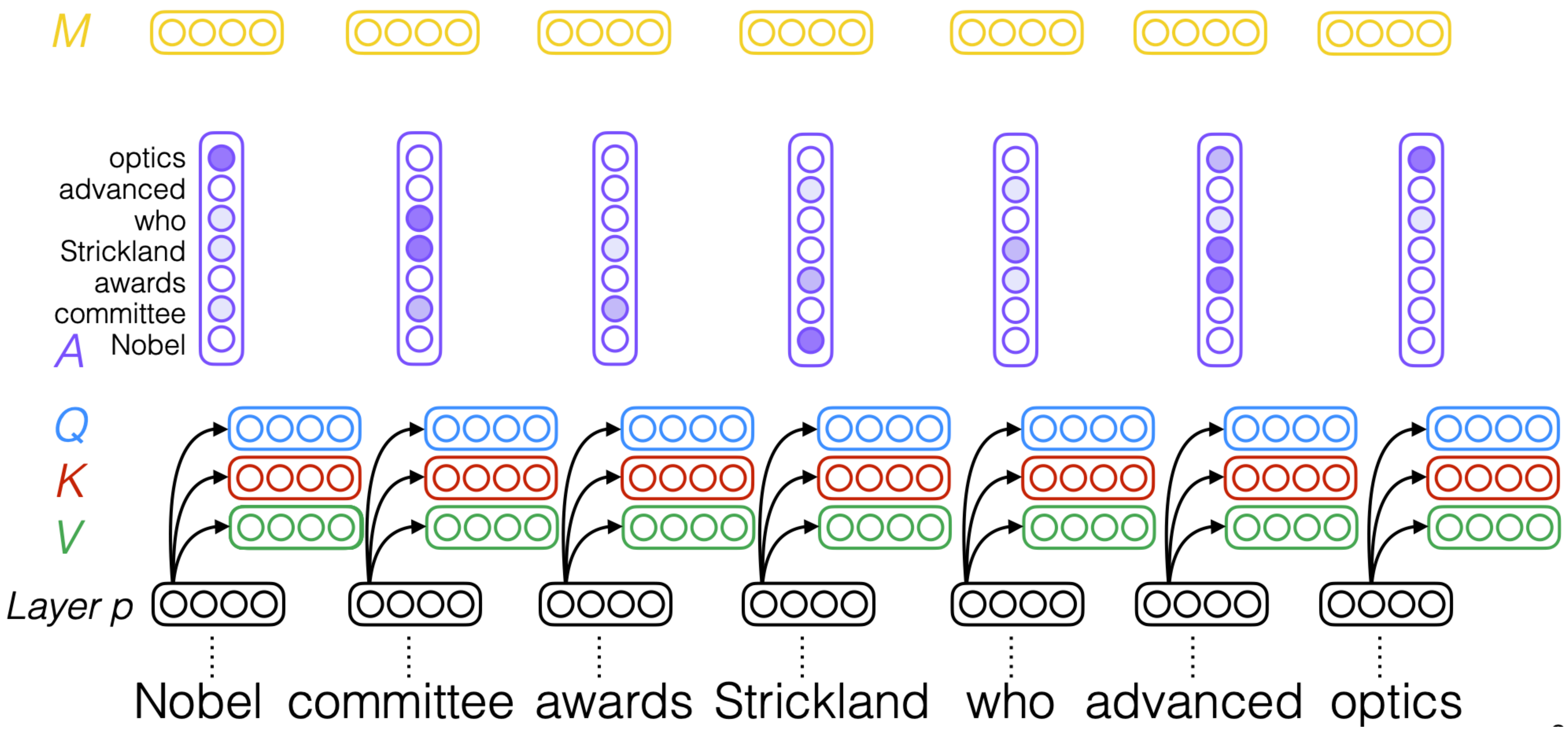
Attention weight on every word in the sequence



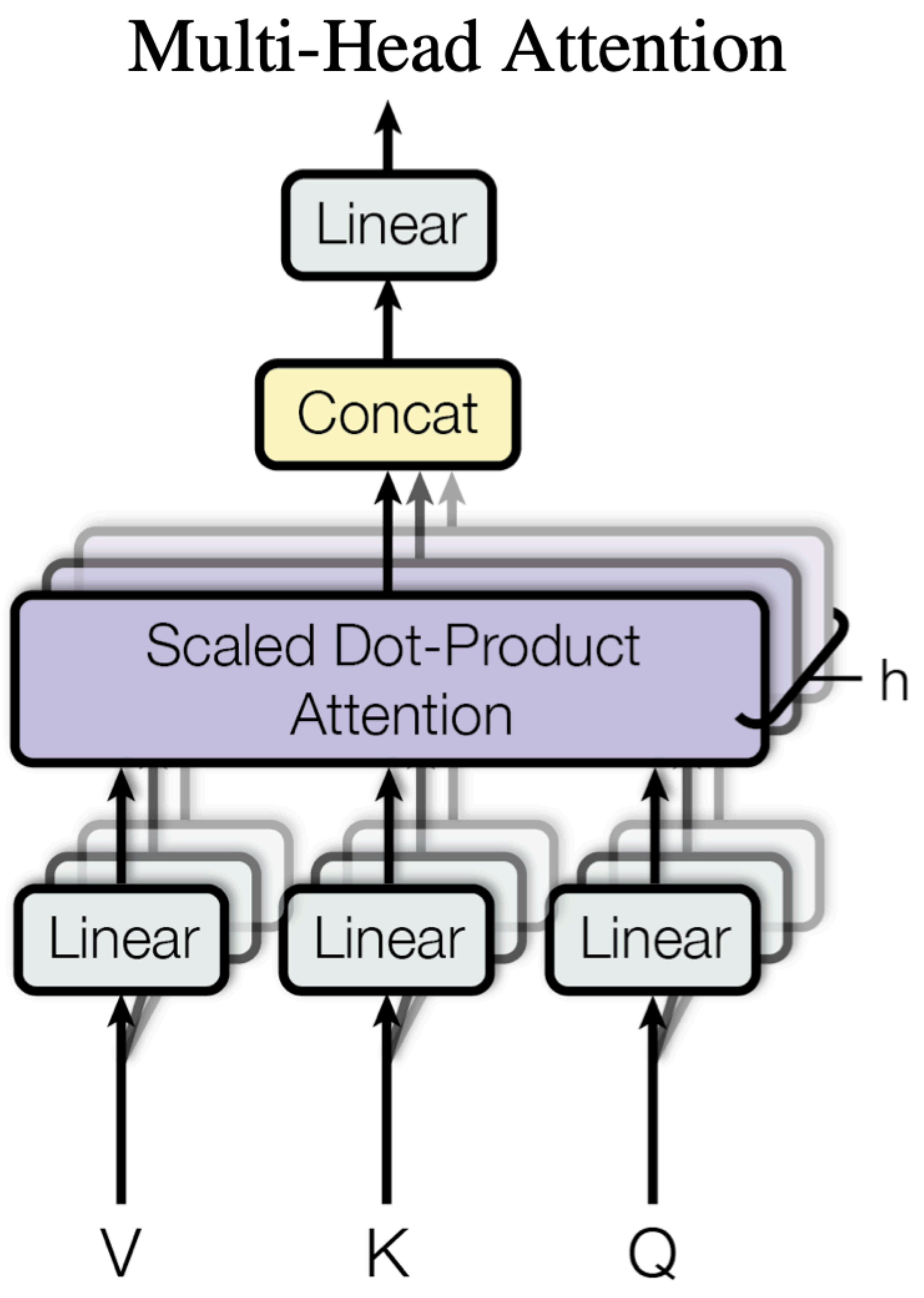
Self-Attention



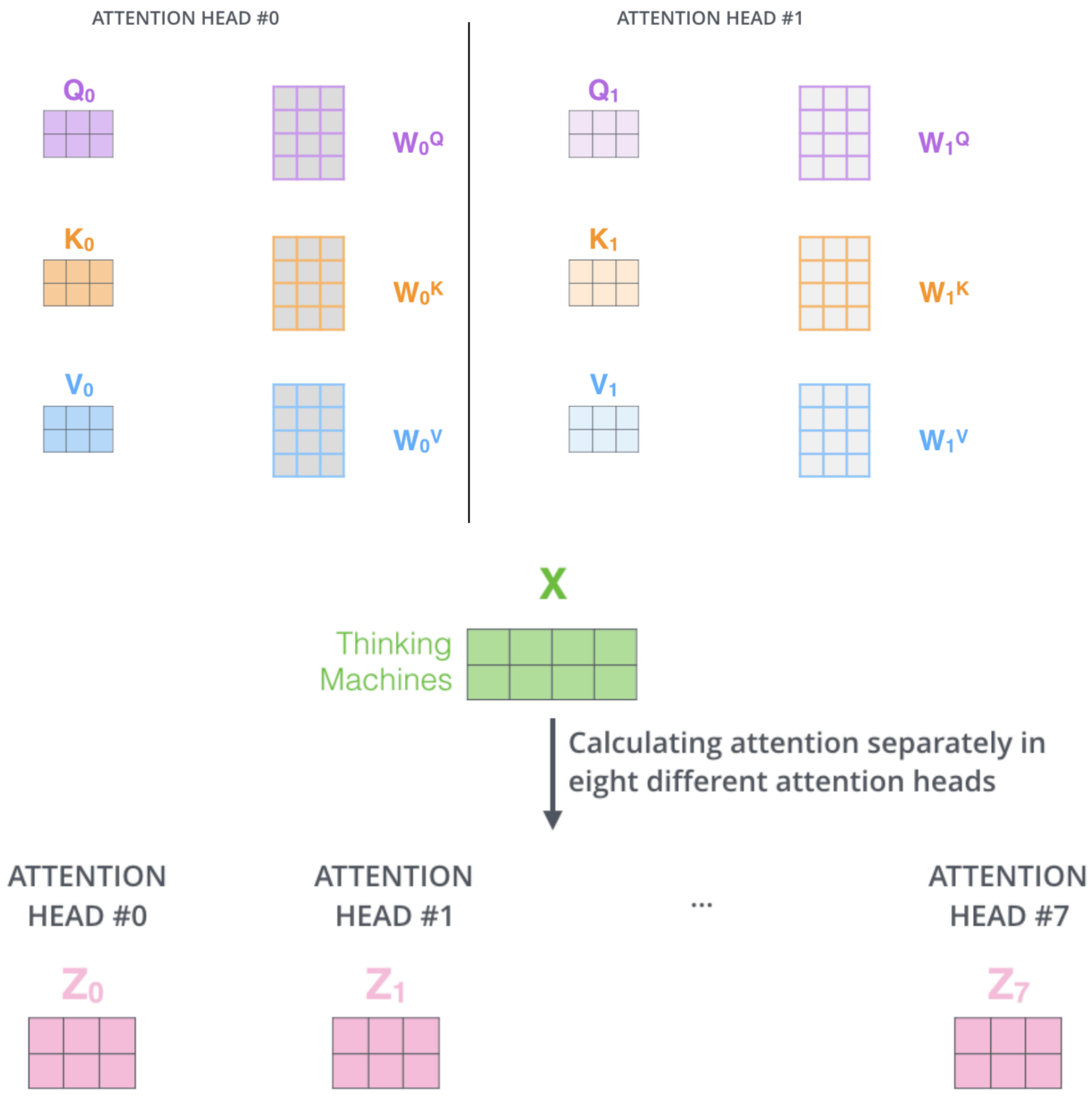
Self-Attention



Multi-Head Attention

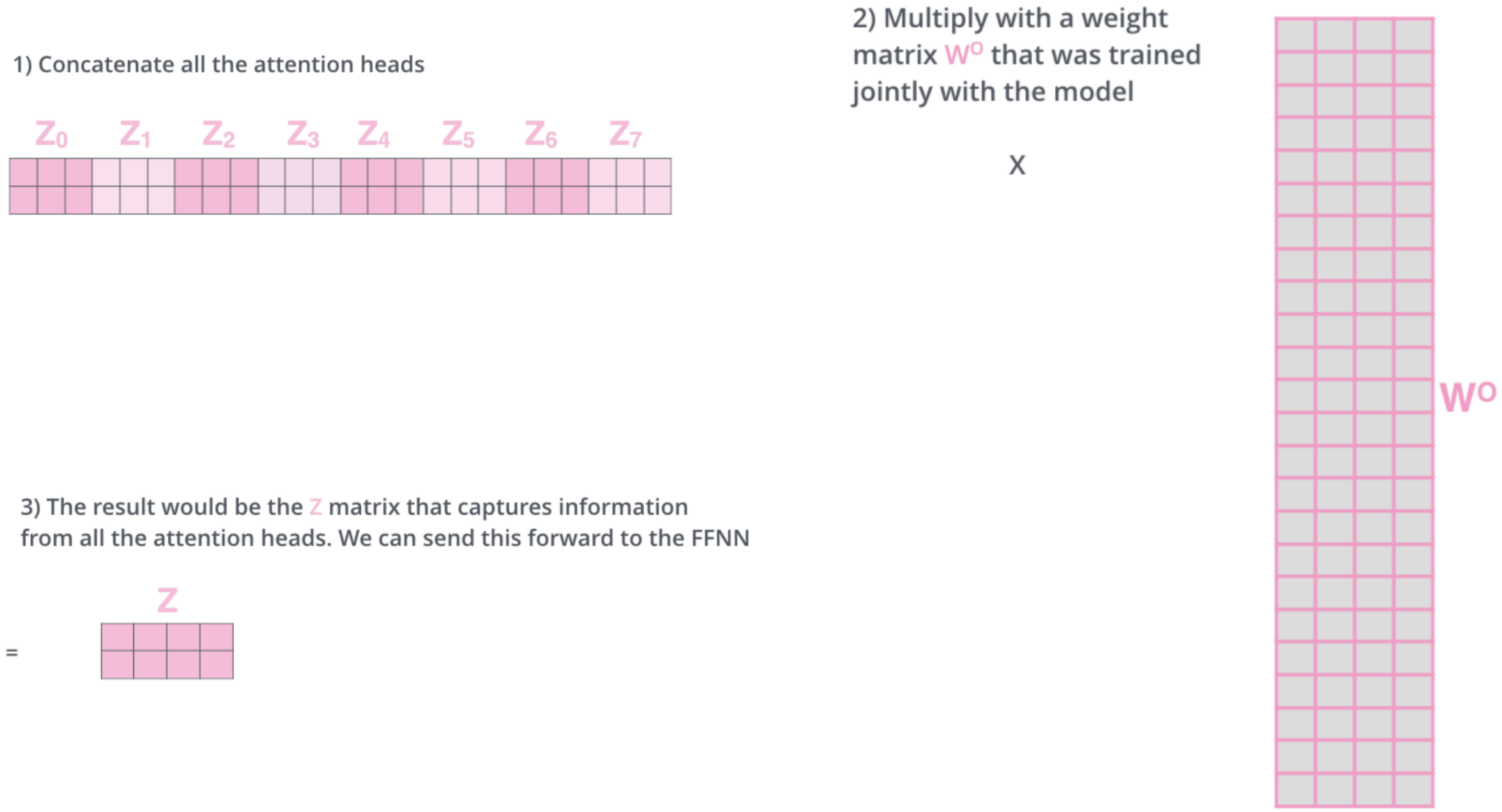


Multi-Head Self-Attention



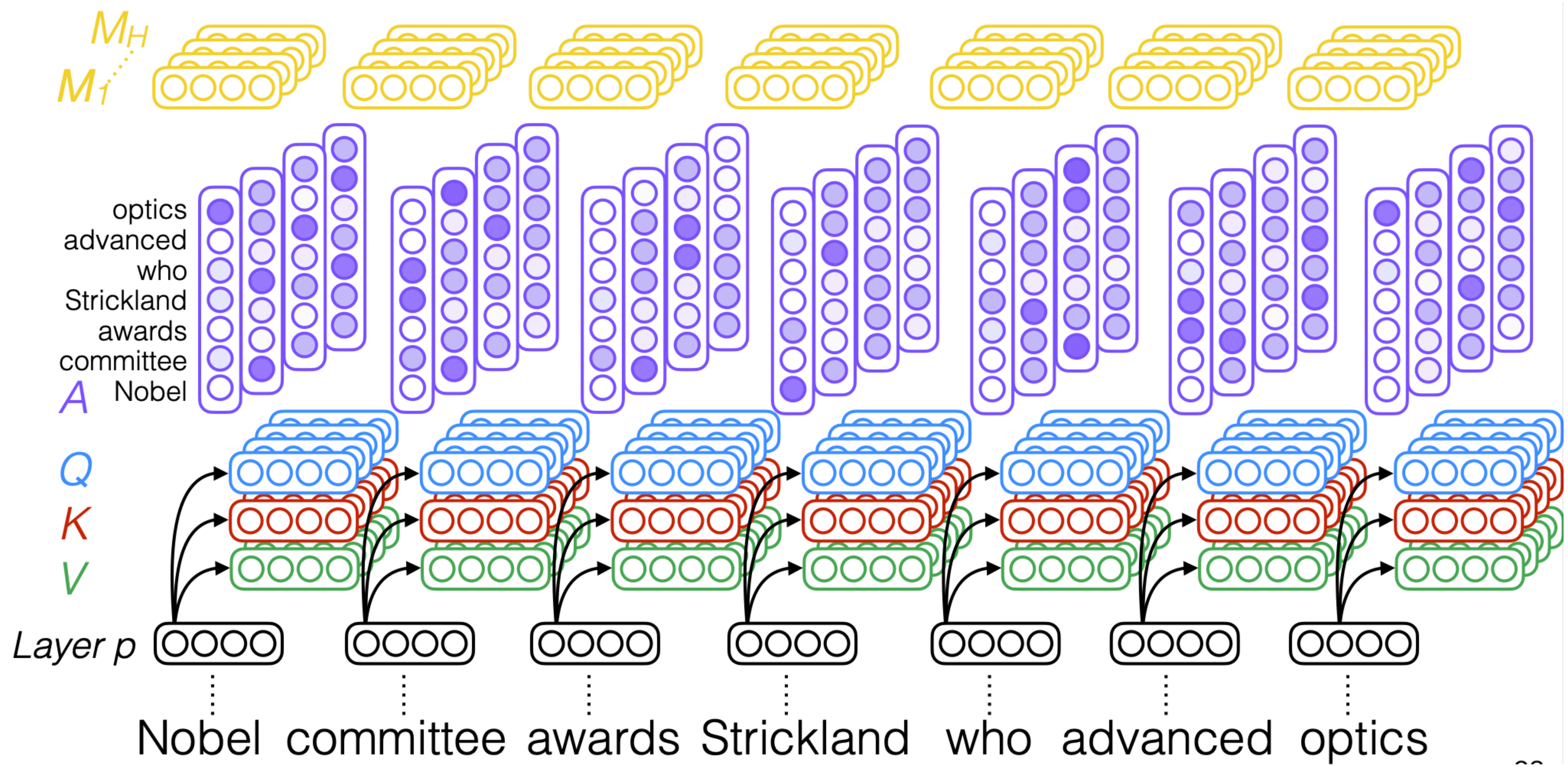
Jay Alammar. The Illustrated Transformer.

Multi-Head Self-Attention



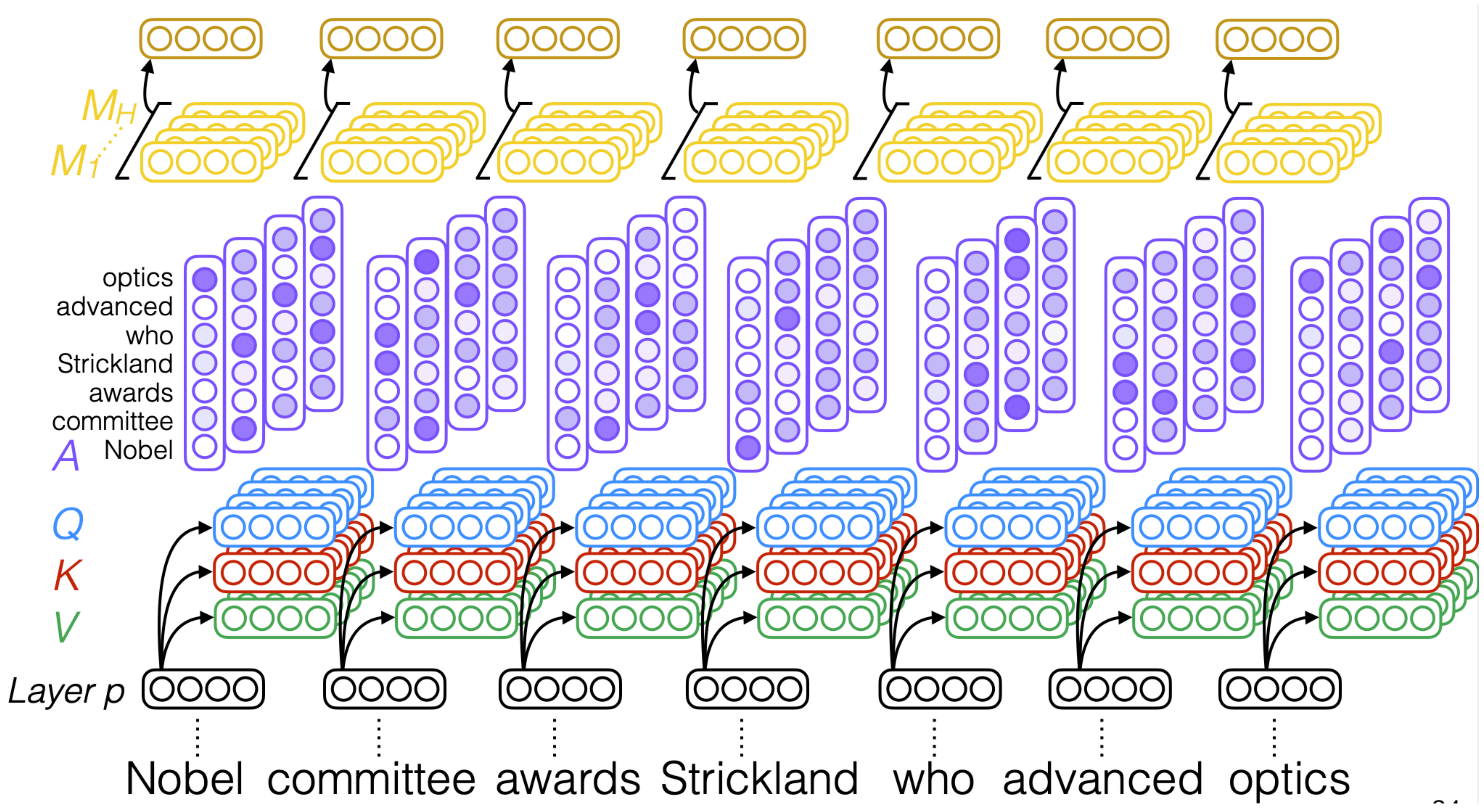
Jay Alammar. The Illustrated Transformer.

Multi-head Self-Attention

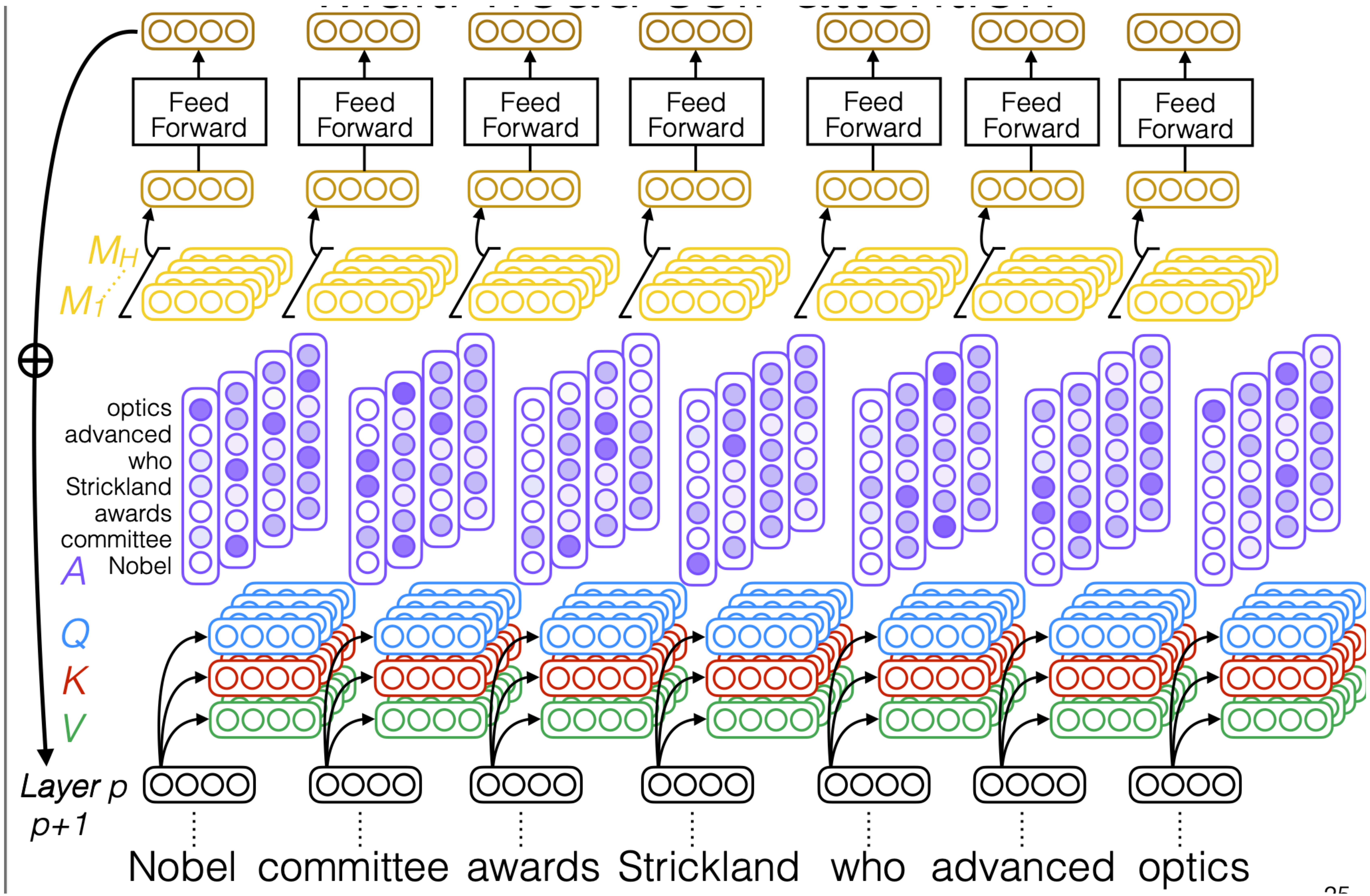


Multi-head Self-Attention

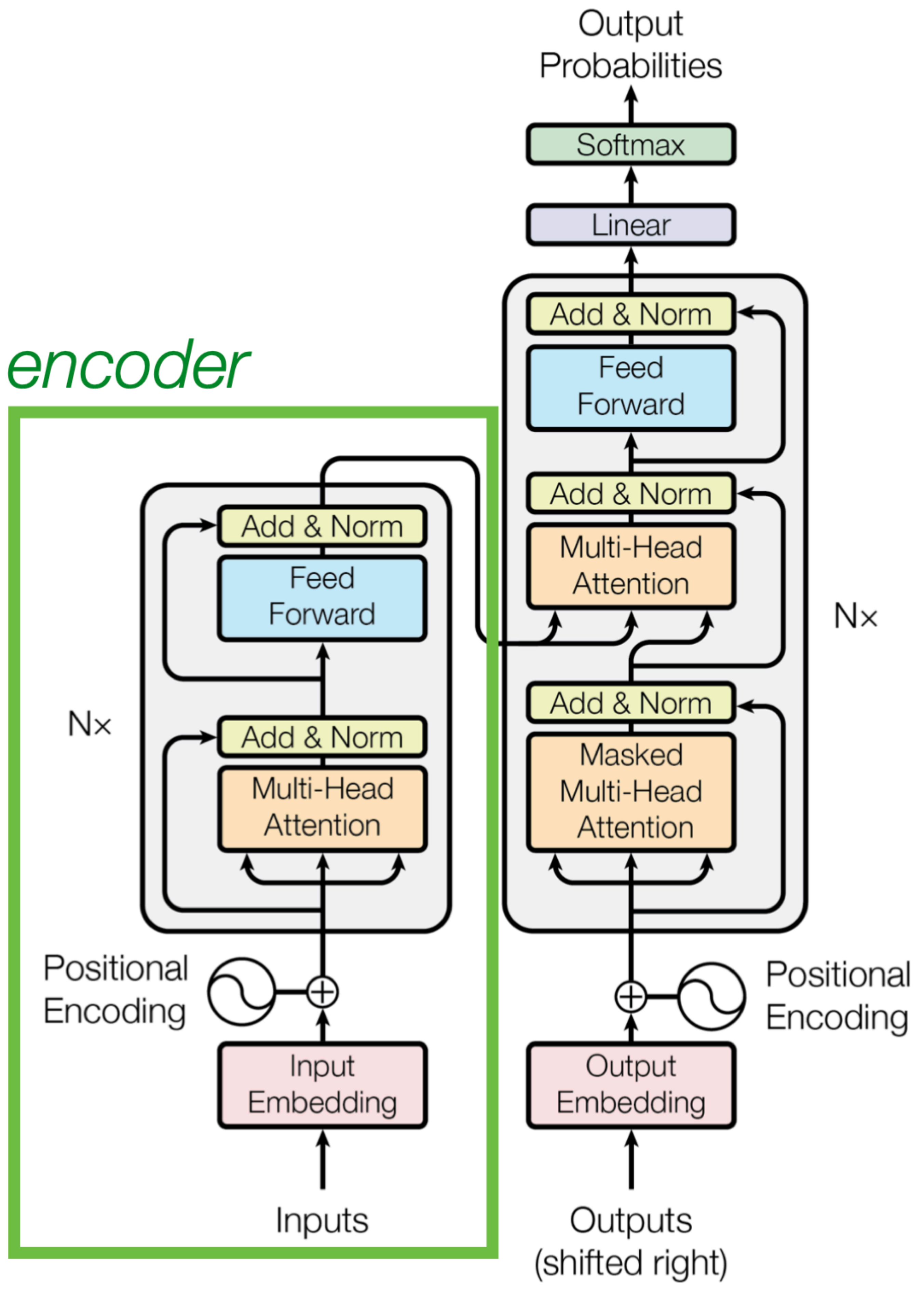
Concat and output projection



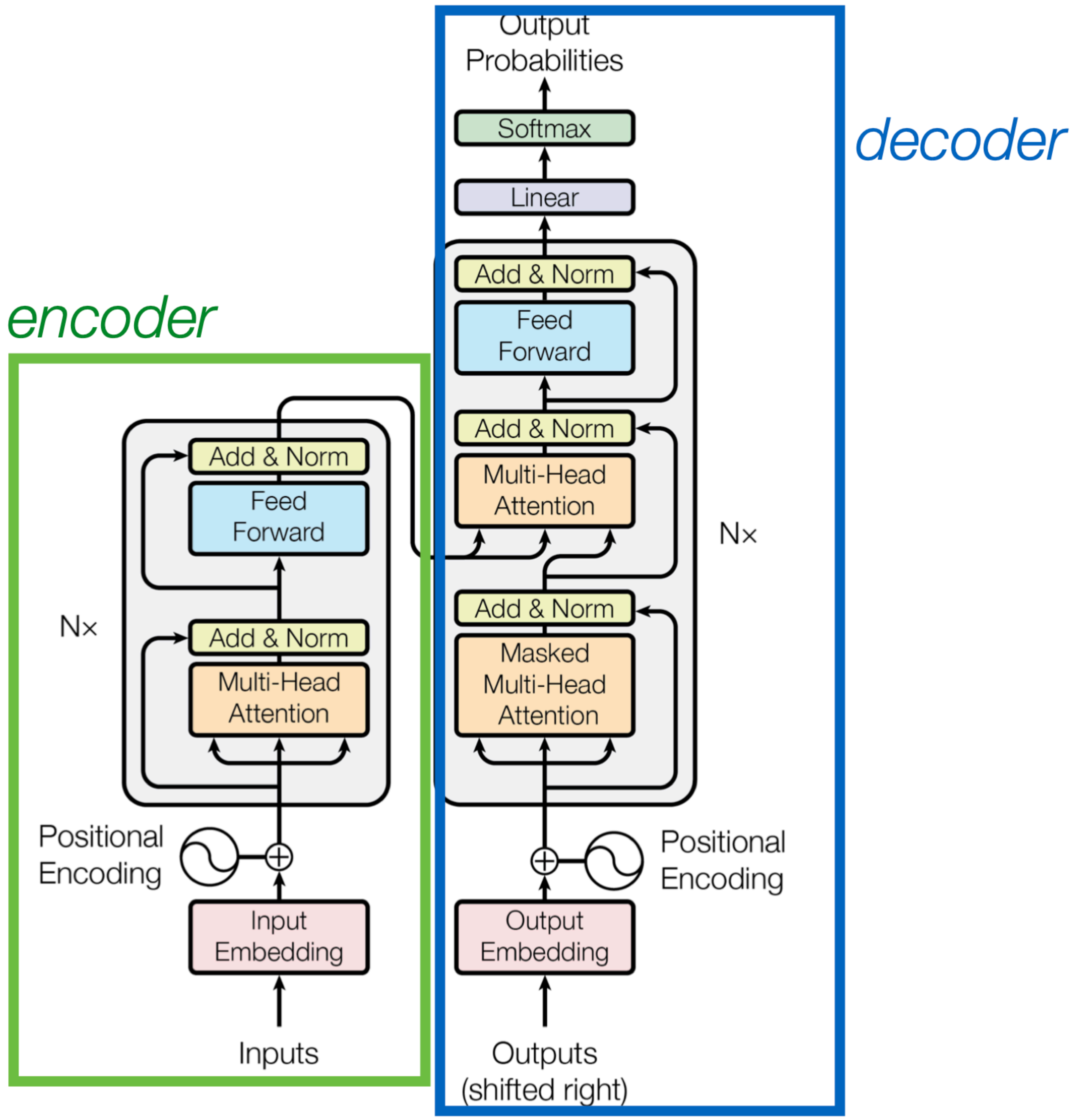
Multi-head Self-Attention + FFN



Transformer Encoder

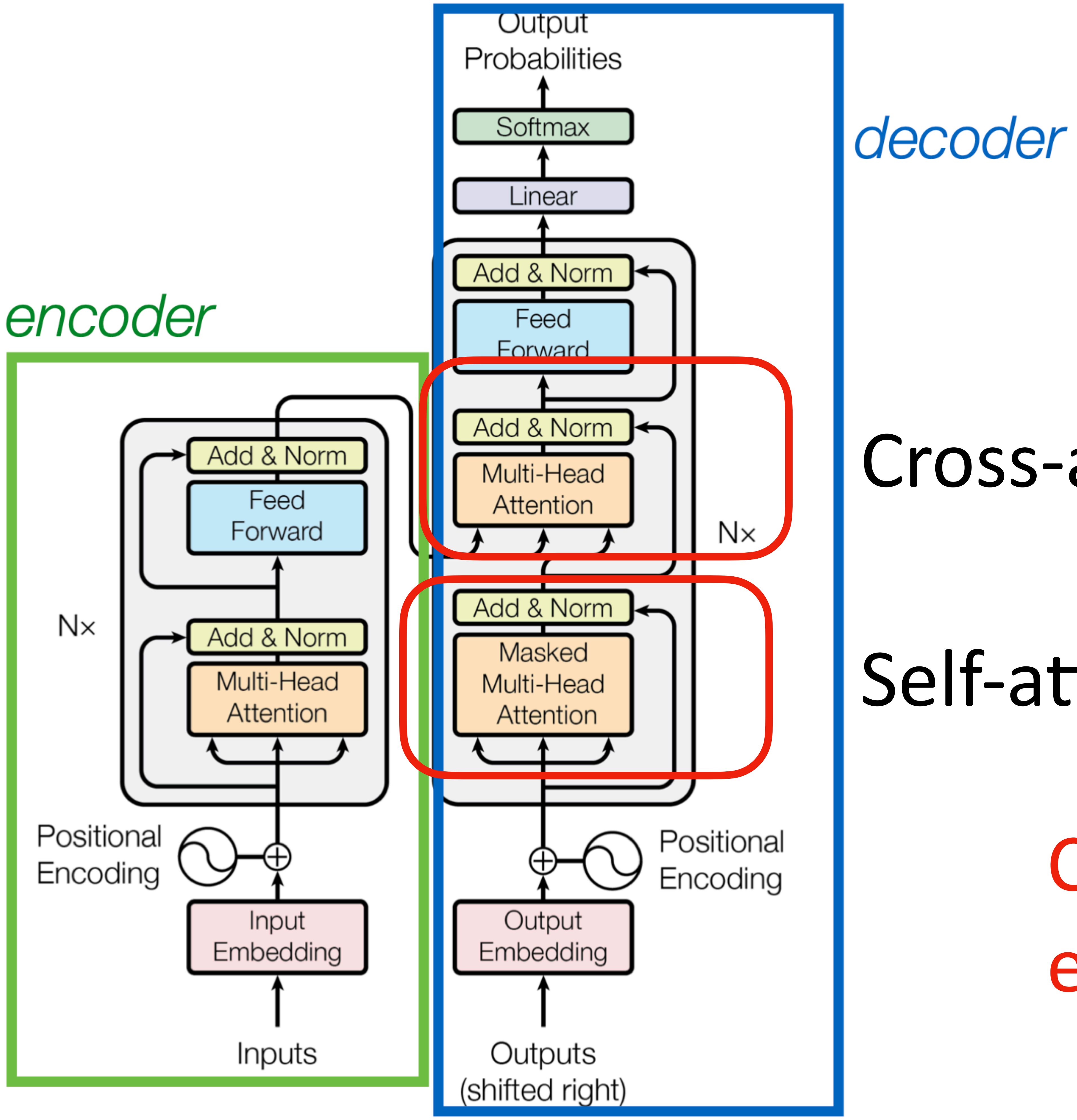


Currently we only cover the encoder side



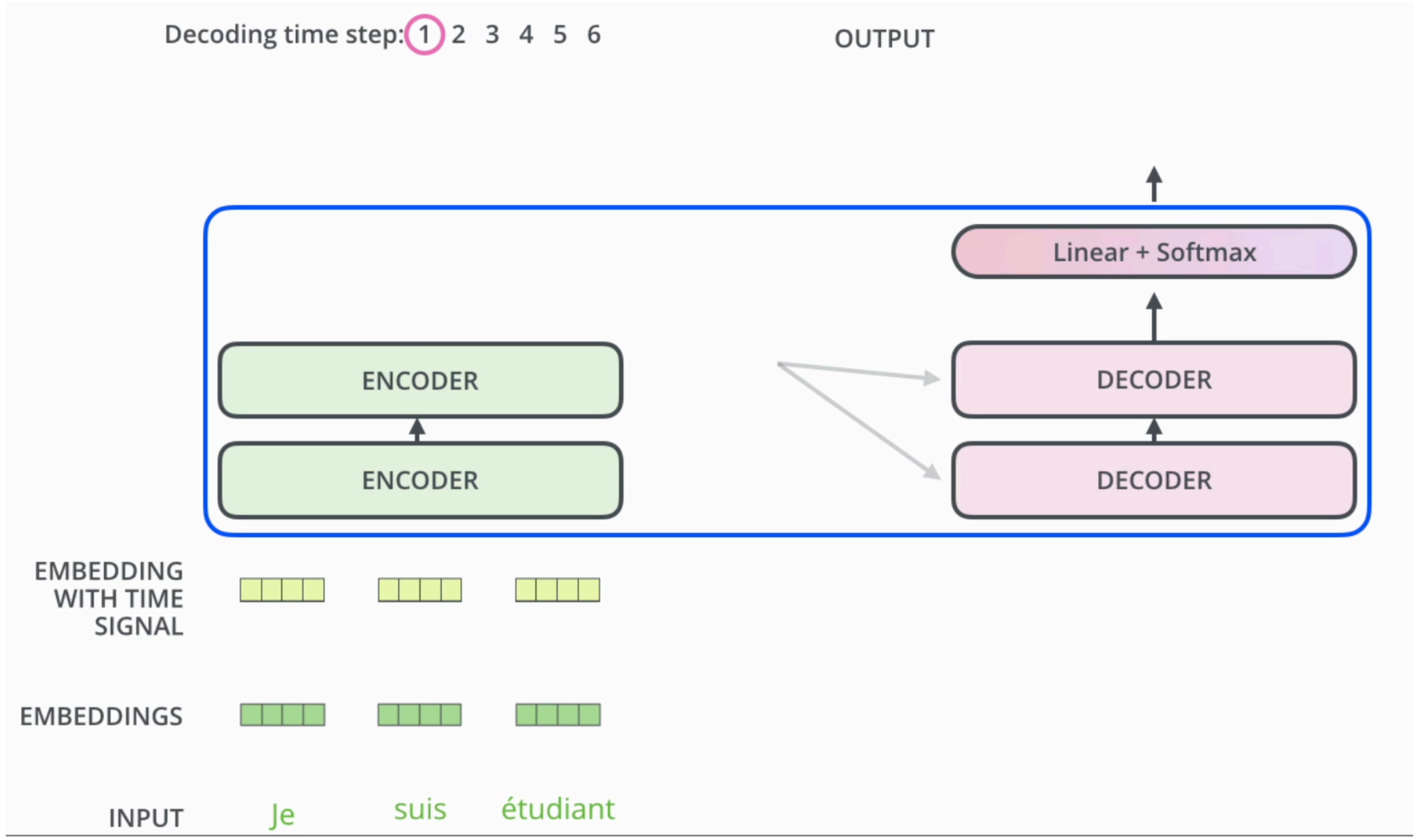
This encoder-decoder arch is originally proposed as a seq2seq arch, for classification tasks, often only encoder is used. And language models often only have a decoder

Transformer Decoder in Seq2Seq



Cross-attention

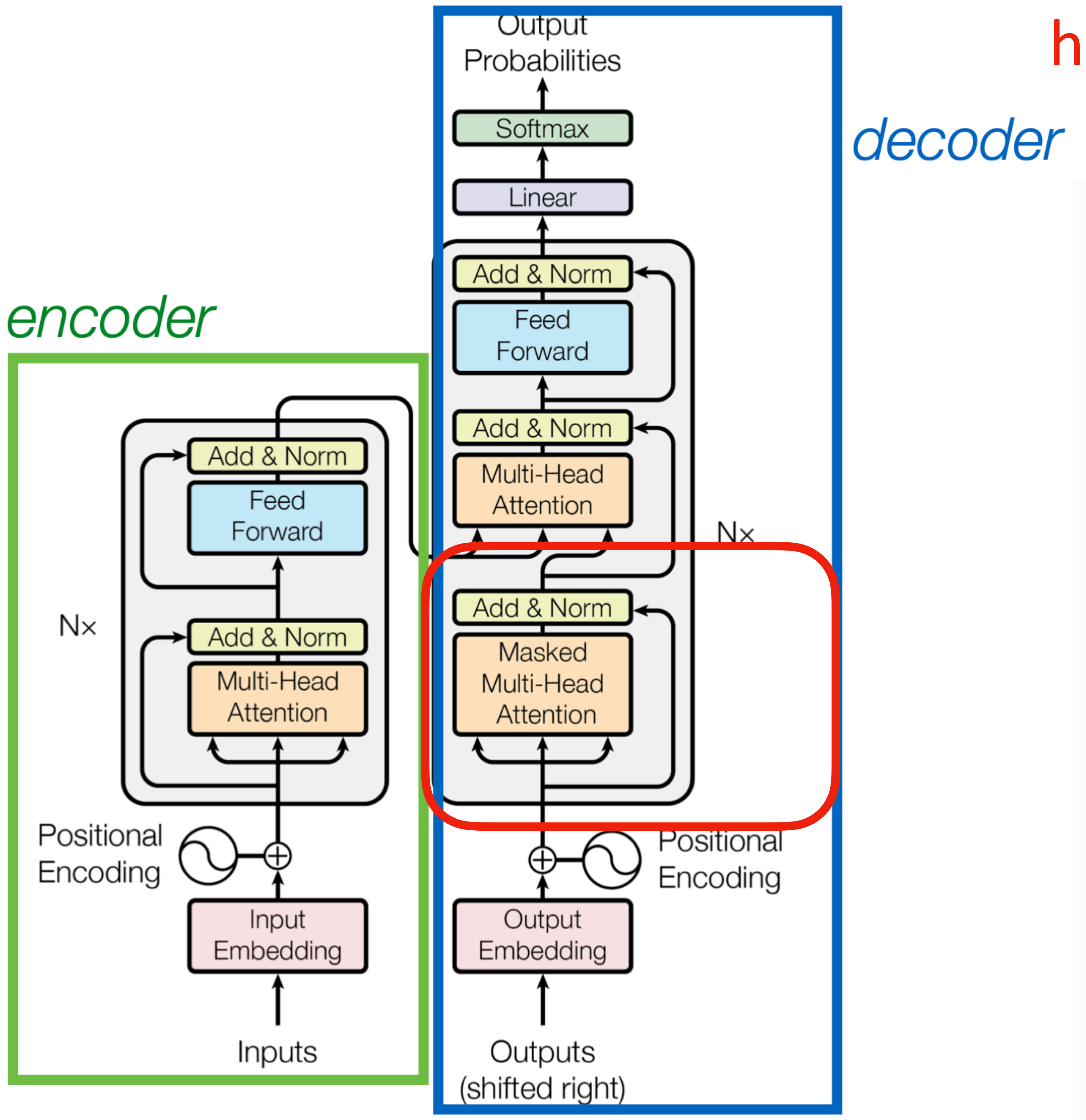
Self-attention

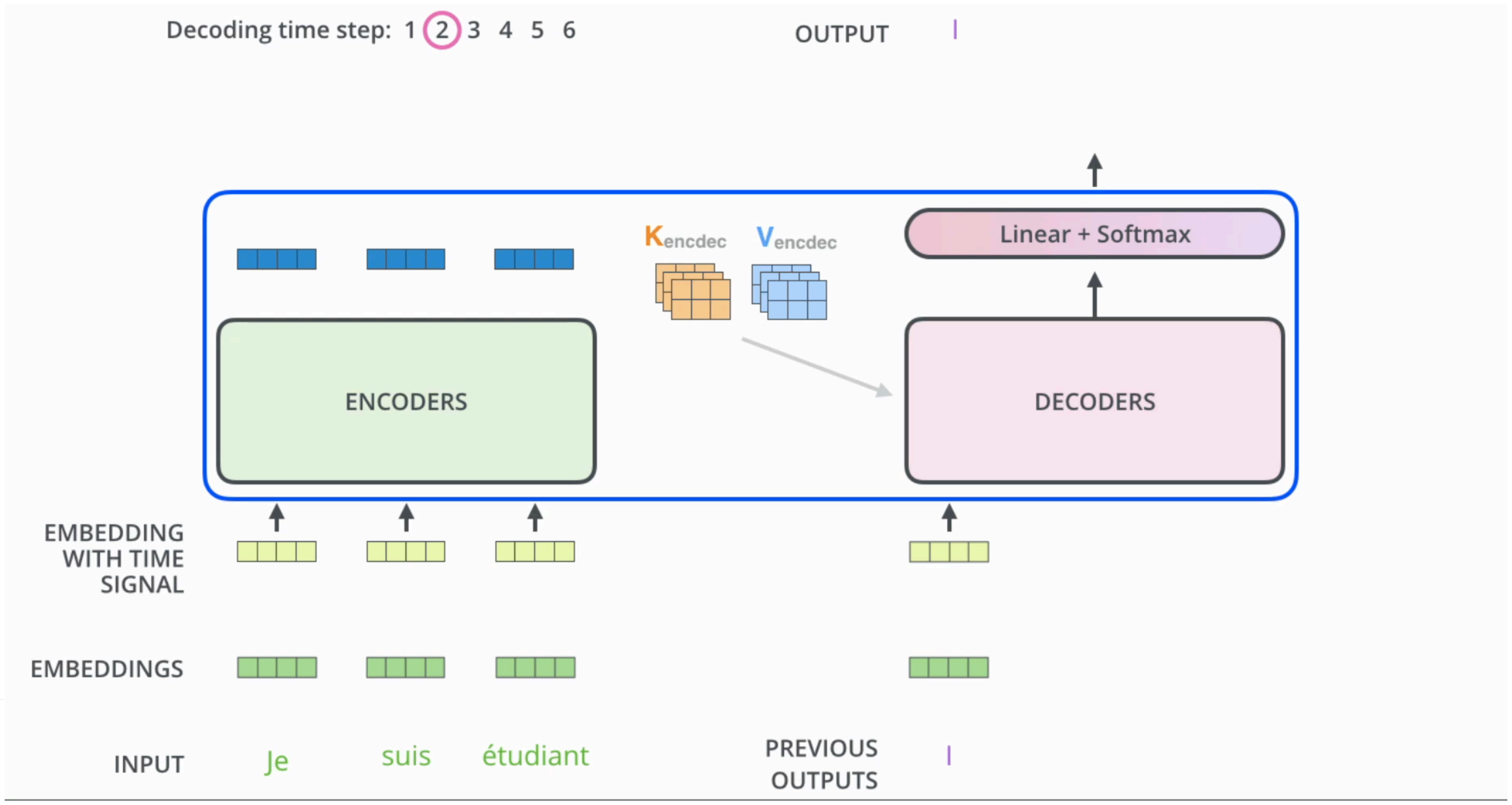


Cross-attention uses the output of encoder as input

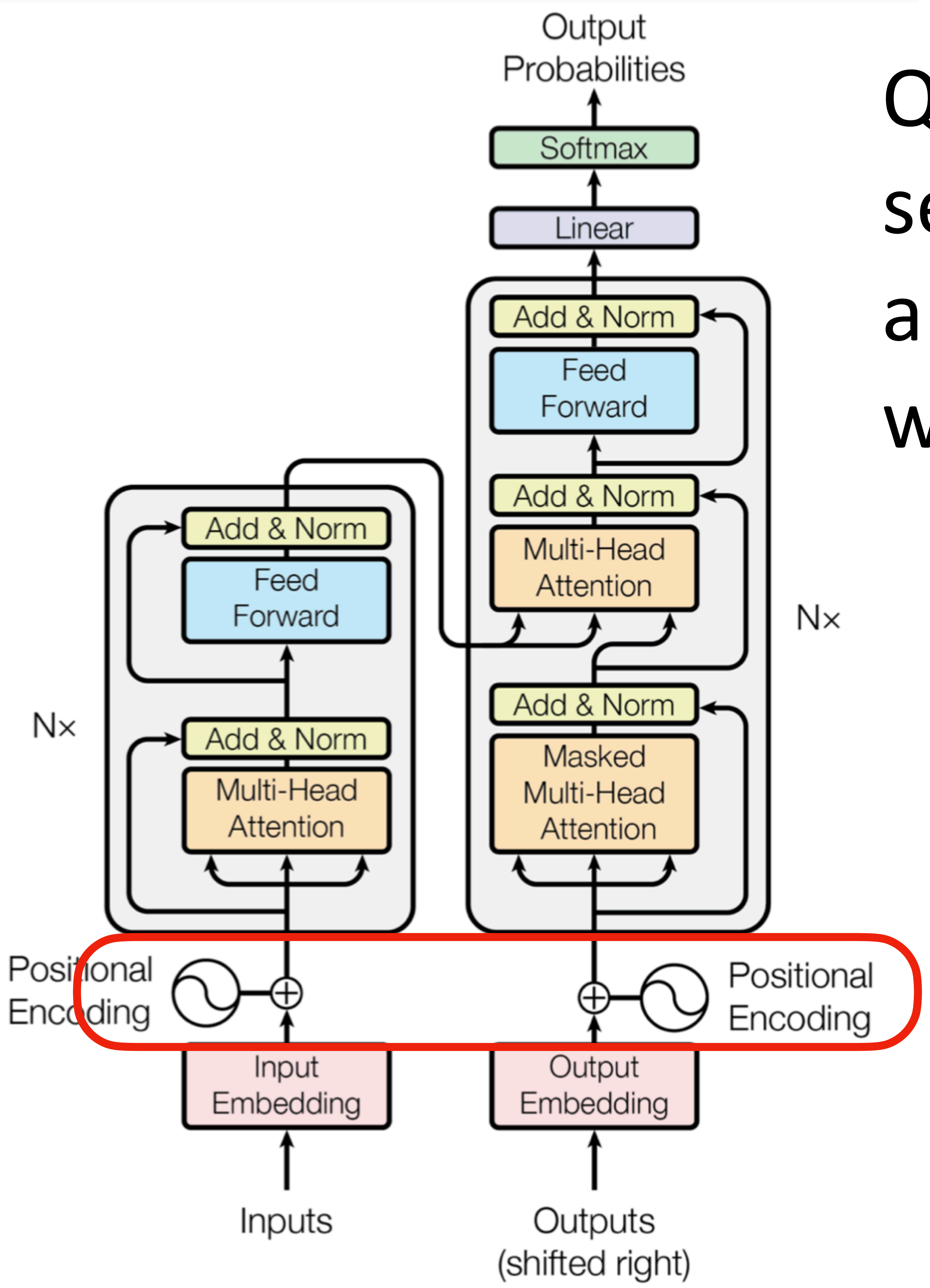
Masked Attention

Typical attention attends to the entire sequence, while masked attention only attends to the ones on the left because future words have not been generated





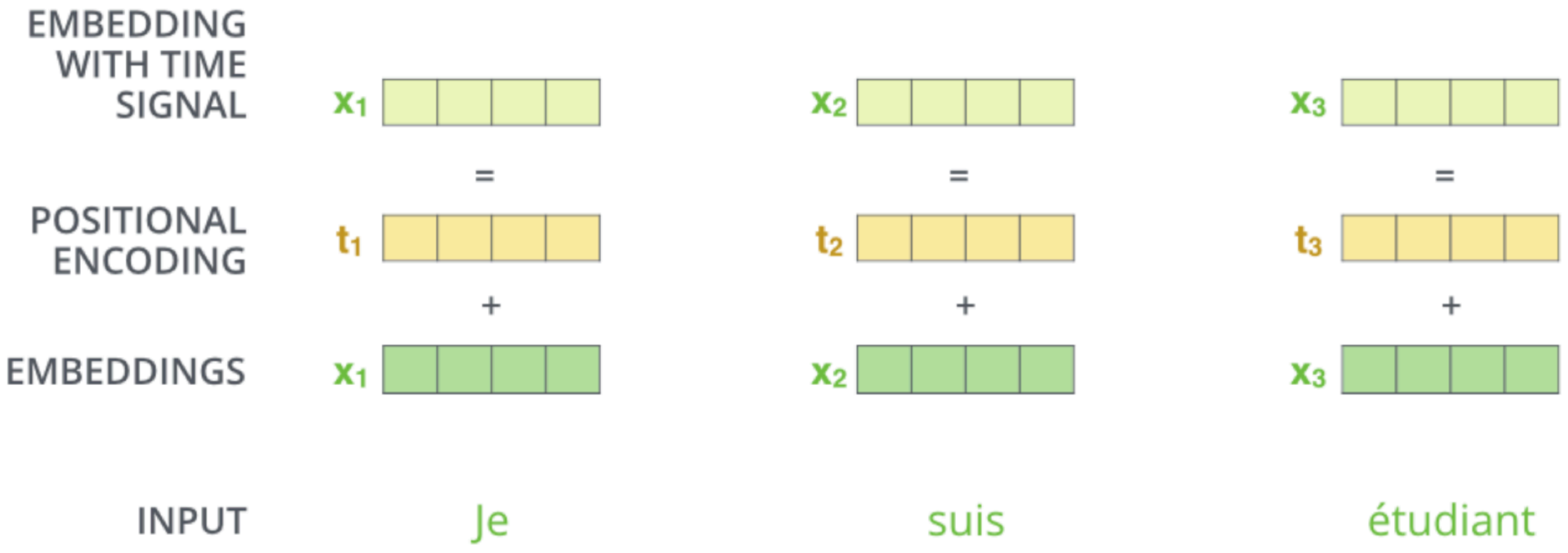
Position Embeddings



Question: If we shuffle the order of words in the sequence, will that change the attention output and feed forward output of the corresponding word?

Position embeddings are added to each word embedding, otherwise our model is unaware of the position of a word

Positional Encoding



Transformer Positional Encoding

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

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

Positional encoding is a 512d vector
i = a particular dimension of this vector
pos = dimension of the word
d_model = 512

Complexity

Layer Type	Complexity per Layer	Sequential Operations
Self-Attention	$O(n^2 \cdot d)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$

n is sequence length, d is embedding dimension.

Restricted self-attention means not attending all words in the sequence, but only a restricted field

Square complexity of sequence length is a major issue for transformers to deal with long sequence

Thank You!