Transformers Part 2

Antoine Bosselut

Tuesday 11th March

9am to 6:30pm in BC

for IC students in master or third year bachelor

>> SCAN

or go to clic.epfl.ch/icbd

CLIC

EPFL

Faculté informatique
et communications

EPIC
EPFL PhDs of IC

Une association reconnue par l'EPFL
An association recognized by the EPFL

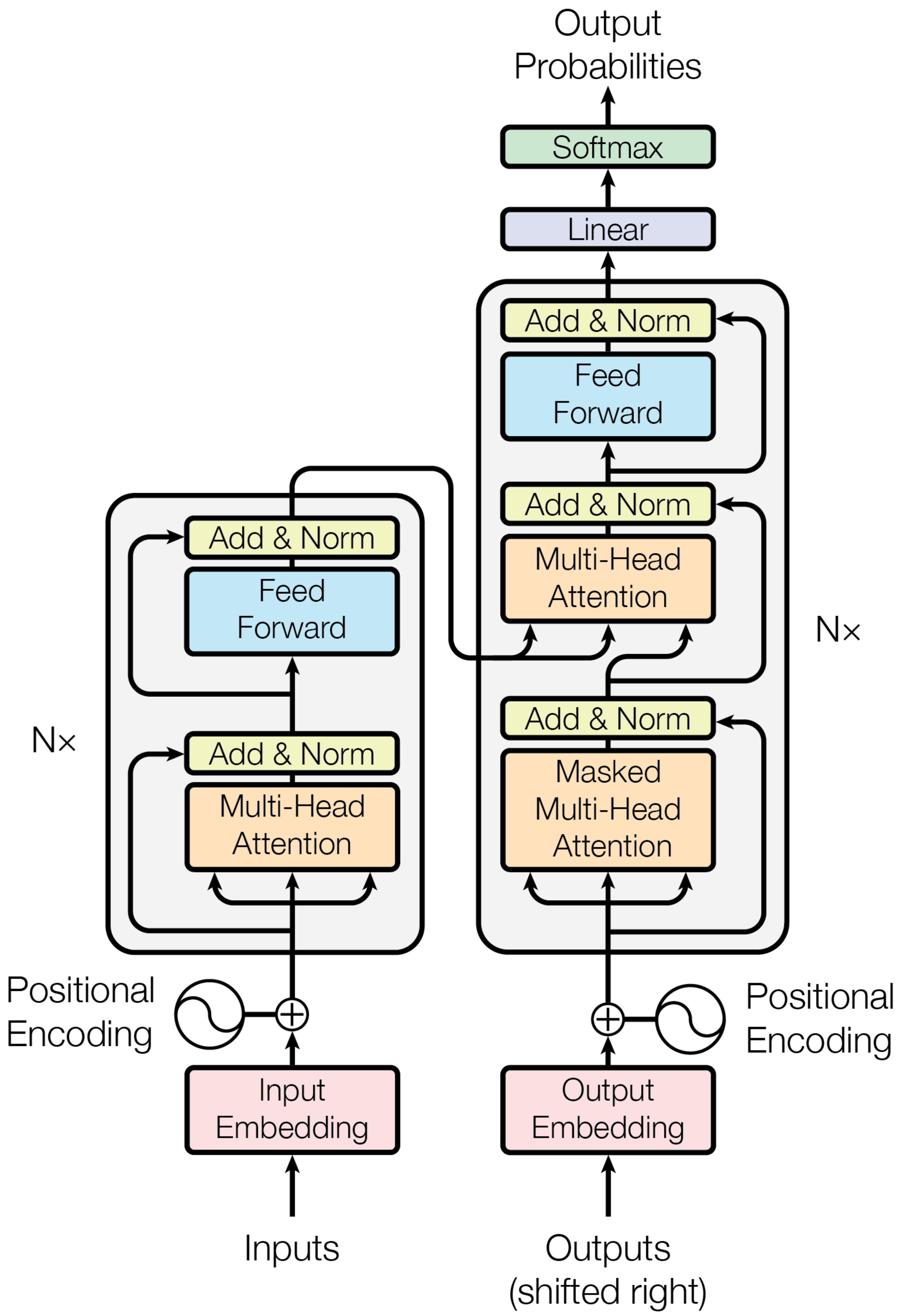
Today's Outline

- **Lecture**
  - **Recap:** Transformers
  - **Decoding**
  - **Supercharging Transformers:** Pretraining, GPT
- **Exercise Session**
  - **Review of Week 2**
  - **Week 3:** Sequence-to-sequence models, transformers + decoding

Some slides adapted from Mohit Iyyer, Emma Strubell

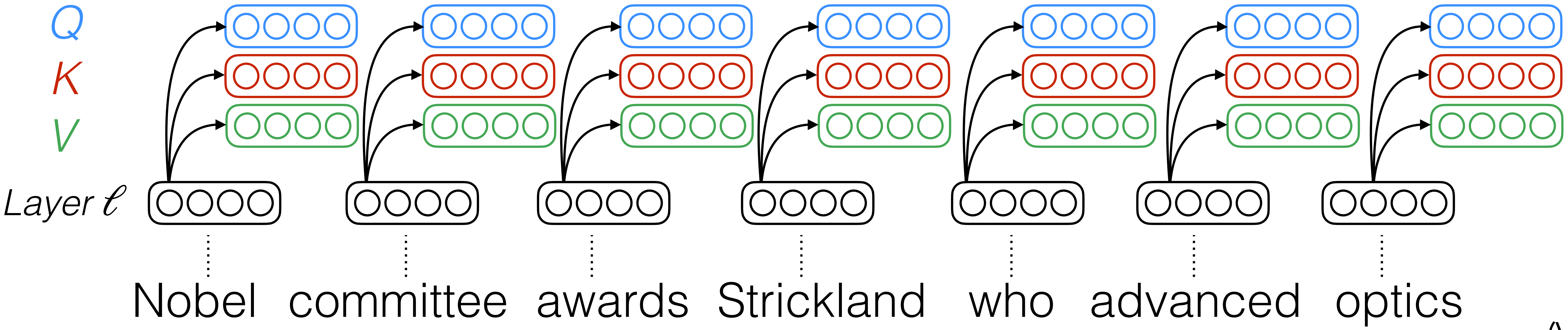
Full Transformer

- Made up of encoder and decoder, each with multiple cascaded transformer blocks
  - slightly different architecture in encoder and decoder transformer blocks
- Blocks generally made up **multi-headed attention** layers (self-attention) and **feedforward** layers
- No recurrent computations! **Encode sequences with self-attention**
  - Position embeddings provide sequence order information to transformer



(Vaswani et al., 2017)

Self-attention (in encoder)



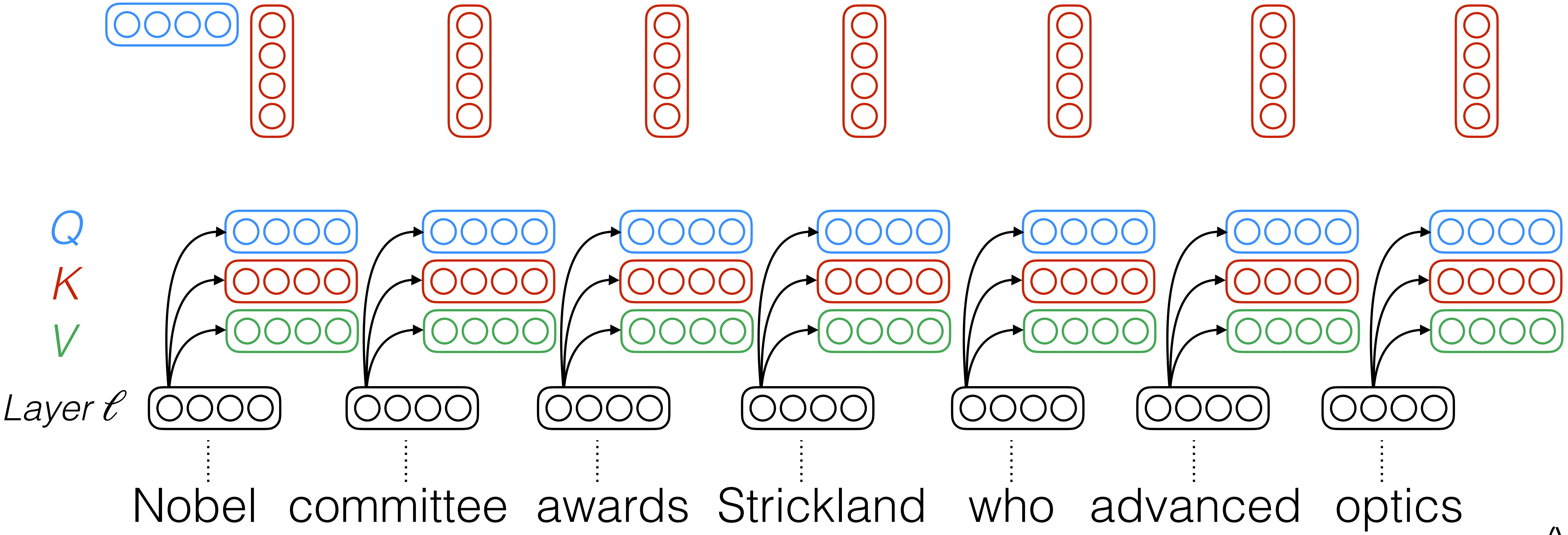
(Vaswani et al., 2017)

Self-attention (in encoder)

$$\mathbf{a_t} = \frac{(\mathbf{W}^Q Q_t)(\mathbf{W}^K K)}{\sqrt{d}}$$

Keys K & values V are the same at every time step: Projected token representations

Query Q_t changes at every time step since the current token serves as the query

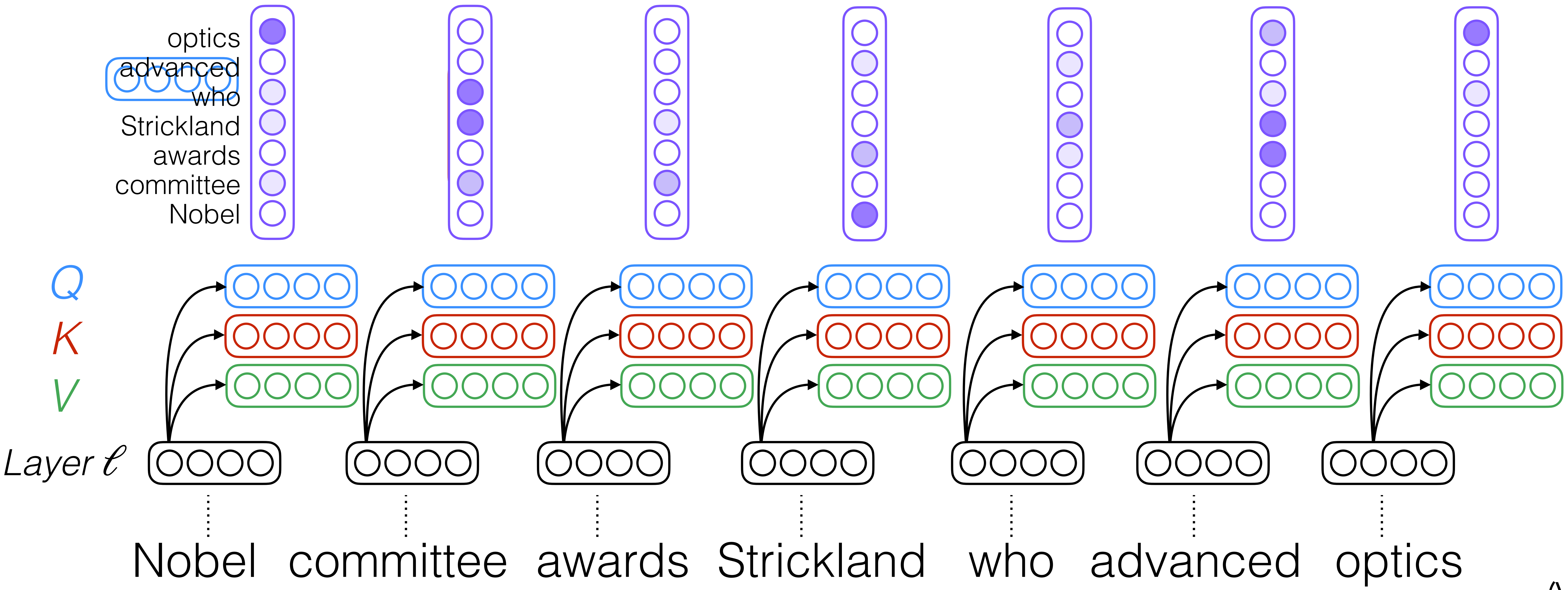


(Vaswani et al., 2017)

Self-attention (in encoder)

$$\mathbf{a_t} = \frac{(\mathbf{W}^Q Q_t)(\mathbf{W}^K K)}{\sqrt{d}}$$

$$\alpha_t = \textbf{softmax}(\mathbf{a_t})$$

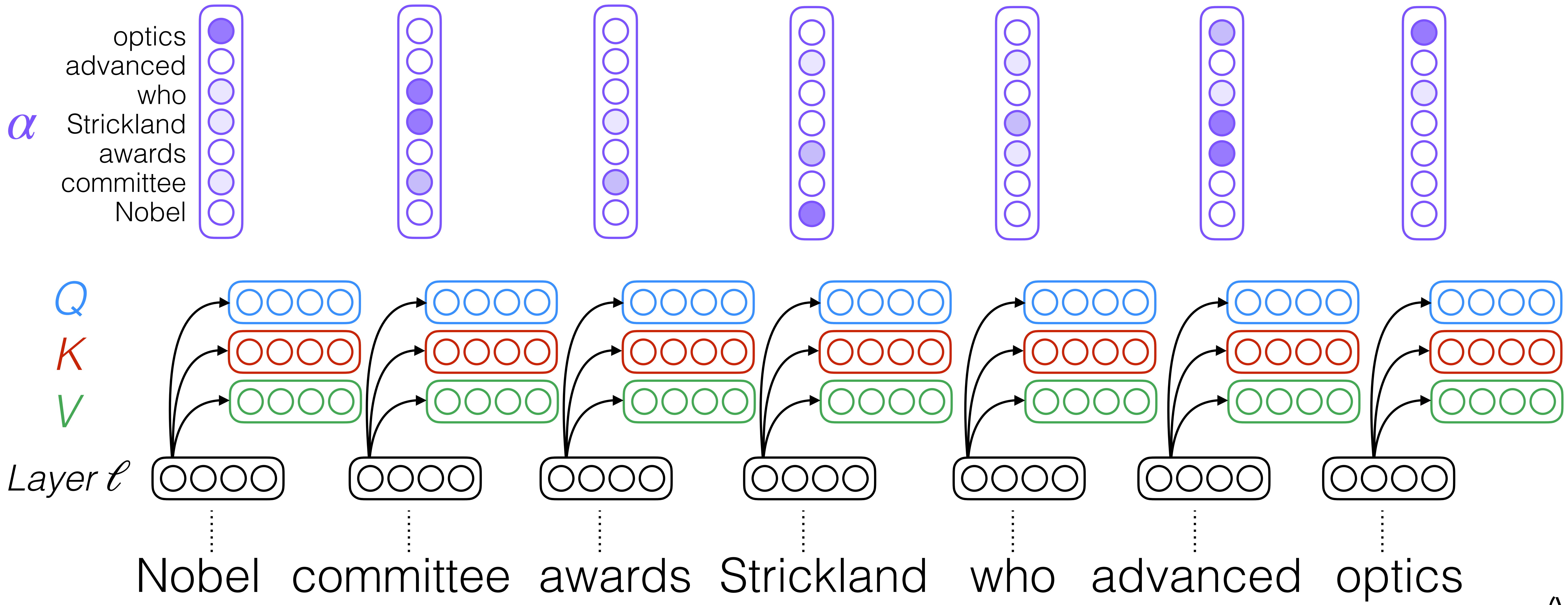


(Vaswani et al., 2017)

Self-attention (in encoder)

$$\mathbf{a_t} = \frac{(\mathbf{W}^Q \boxed{Q_t})(\mathbf{W}^K \boxed{K})}{\sqrt{d}}$$

$$\boxed{\alpha_t} = \mathbf{softmax}(\mathbf{a_t})$$



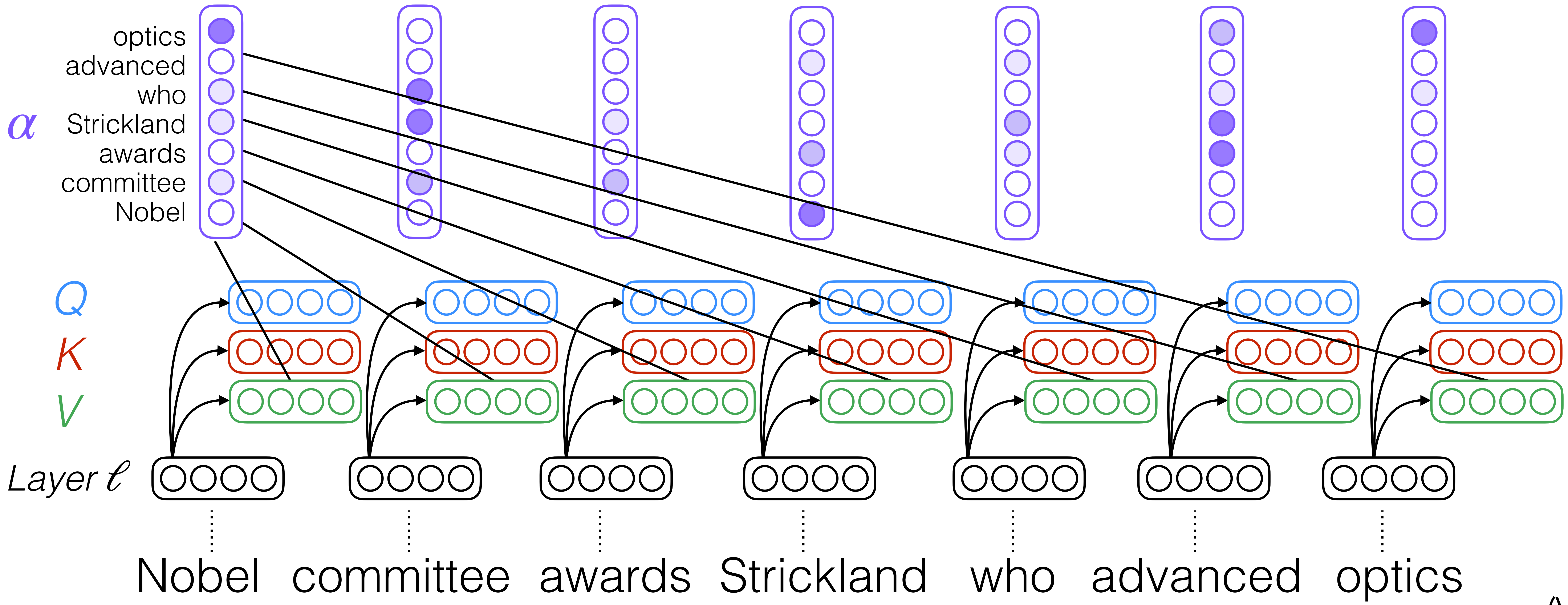
(Vaswani et al., 2017)

Self-attention (in encoder)

$$\mathbf{a_t} = \frac{(\mathbf{W}^Q Q_t)(\mathbf{W}^K K)}{\sqrt{d}}$$

$$\alpha_t = \mathbf{softmax}(\mathbf{a_t})$$

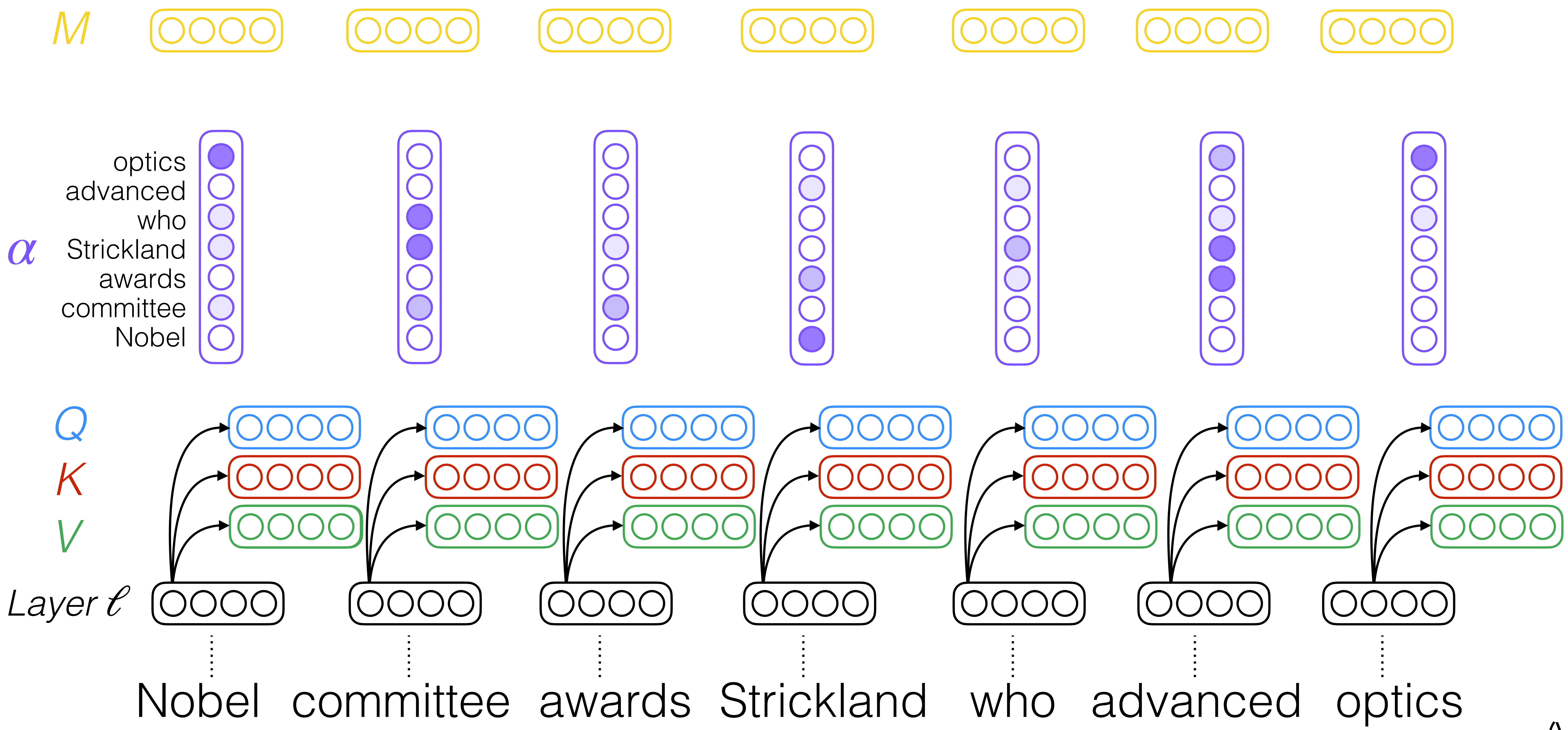
$$M_t = W^O \alpha_t (V\mathbf{W}^V)$$



(Vaswani et al., 2017)

Self-attention (in encoder)

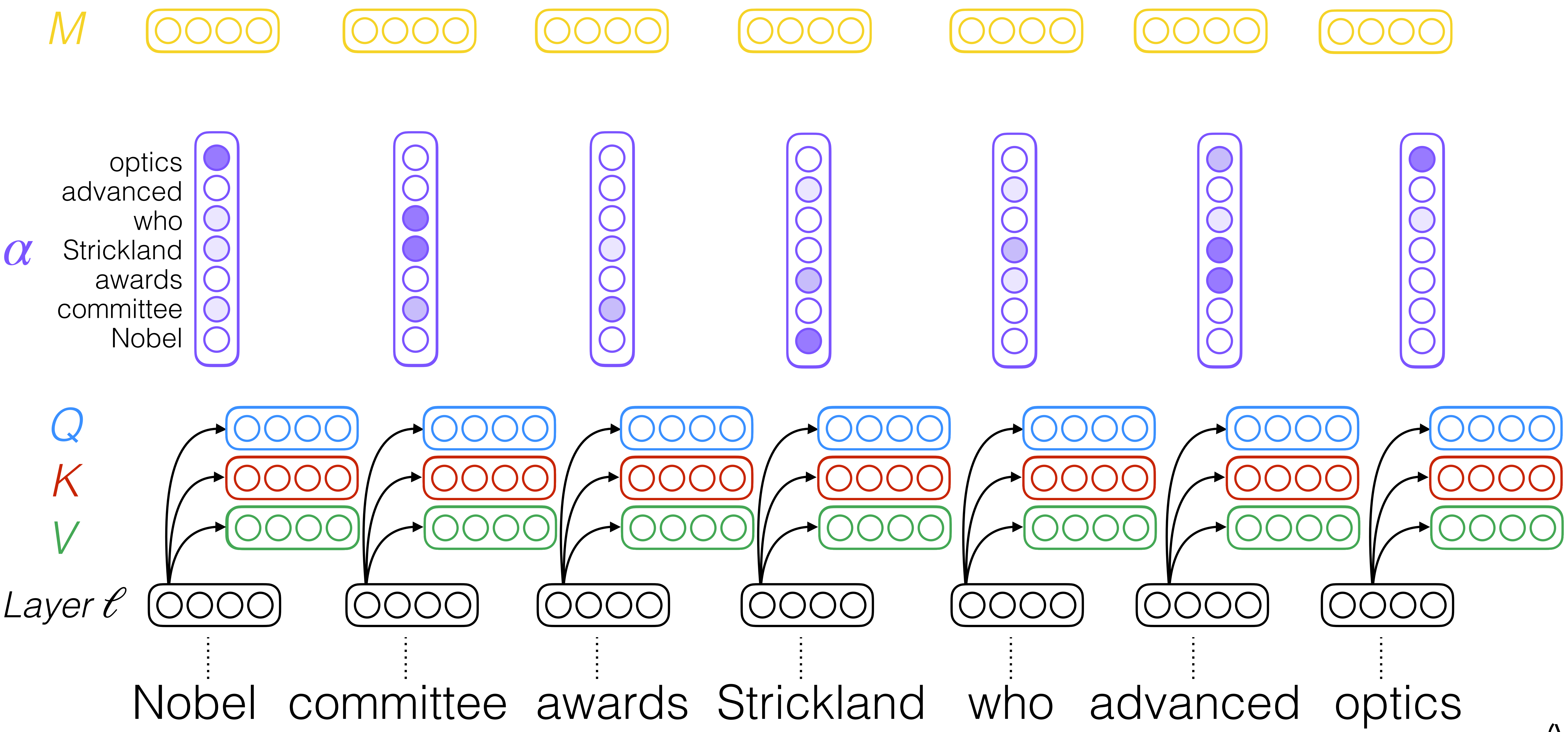
$$M_t = W^O \alpha_t (V \mathbf{W}^V)$$



(Vaswani et al., 2017)

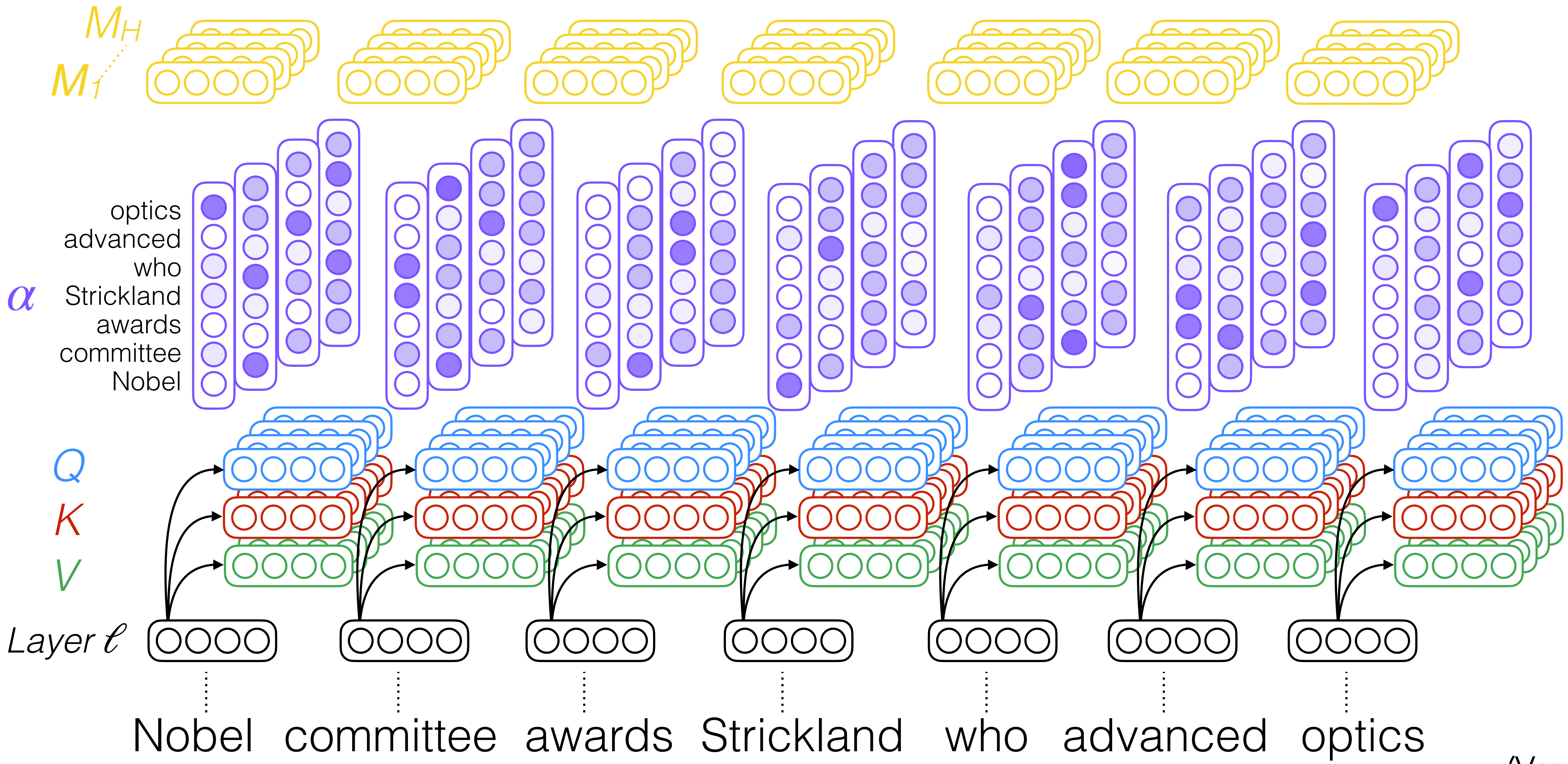
Self-attention (in encoder)

$$M_t = W^O \alpha_t (V \mathbf{W}^V)$$



(Vaswani et al., 2017)

$$\mathbf{a}_{h,t} = \frac{(\mathbf{W}_h^Q Q_t)(\mathbf{W}_h^K K)}{\sqrt{d/H}} \qquad \alpha_{h,t} = \textbf{softmax}(\mathbf{a}_{h,t}) \qquad M_{h,t} = \alpha_{h,t}(V\mathbf{W}_h^V)$$



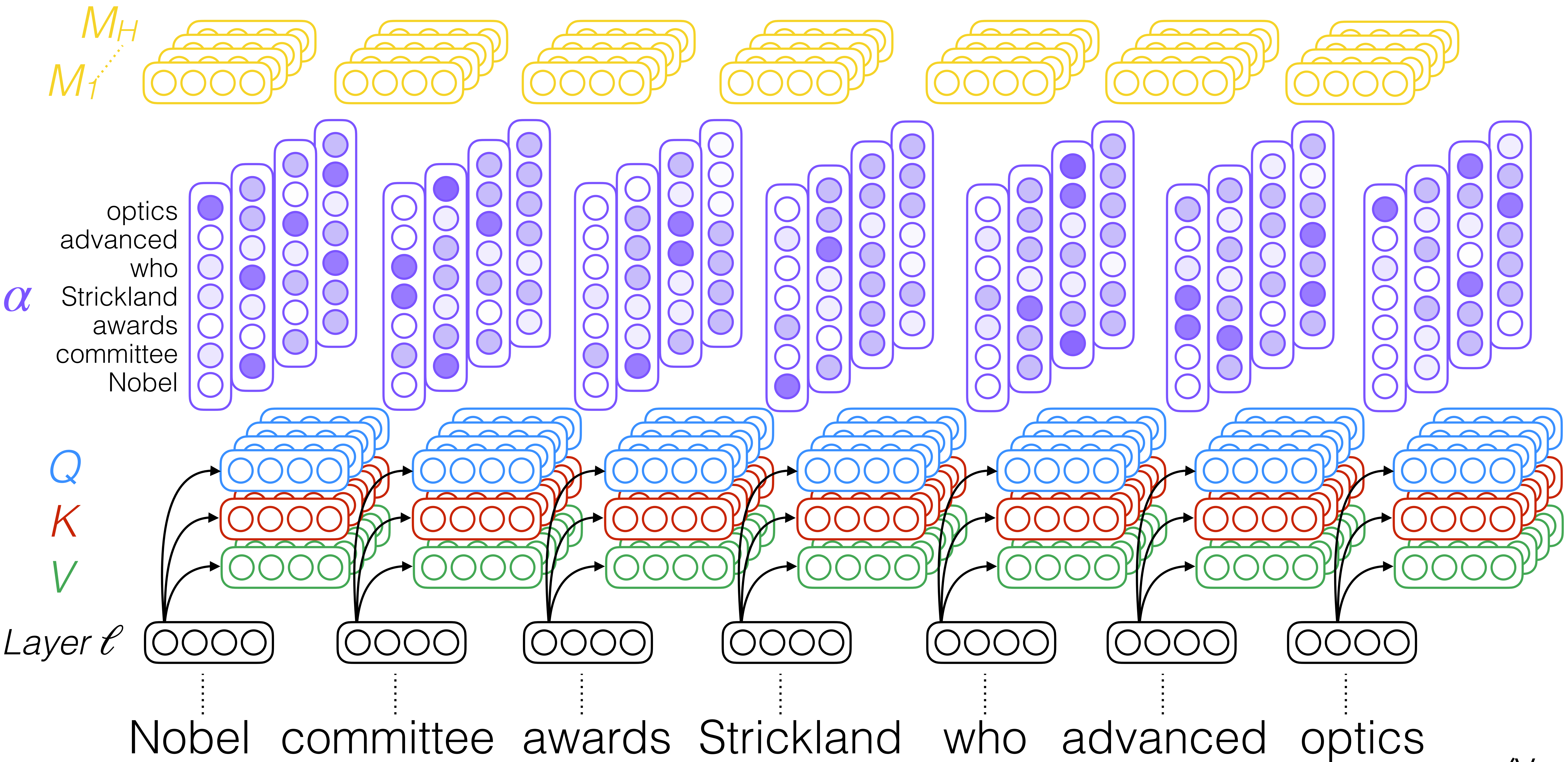
(Vaswani et al., 2017)

$$\mathbf{a}_{h,t} = \frac{(\mathbf{W}_h^Q Q_t)(\mathbf{W}_h^K K)}{\sqrt{d/H}}$$

$$\alpha_{h,t} = \mathbf{softmax}(\mathbf{a}_{h,t})$$

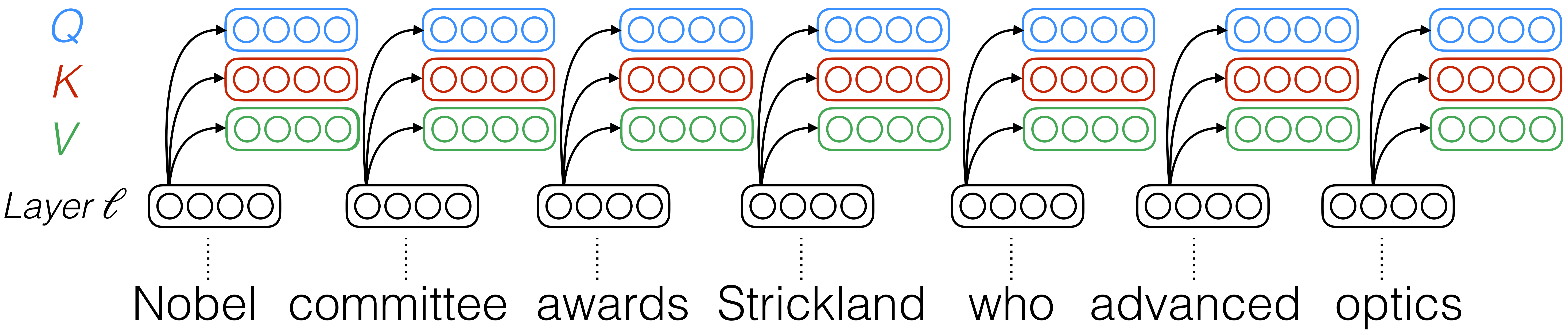
$$M_{h,t} = \alpha_{h,t}(V\mathbf{W}_h^V)$$

$$M_t = W^O[M_{1,t}; \ldots; M_{H,t}]$$



(Vaswani et al., 2017)

Single Headed Attention:

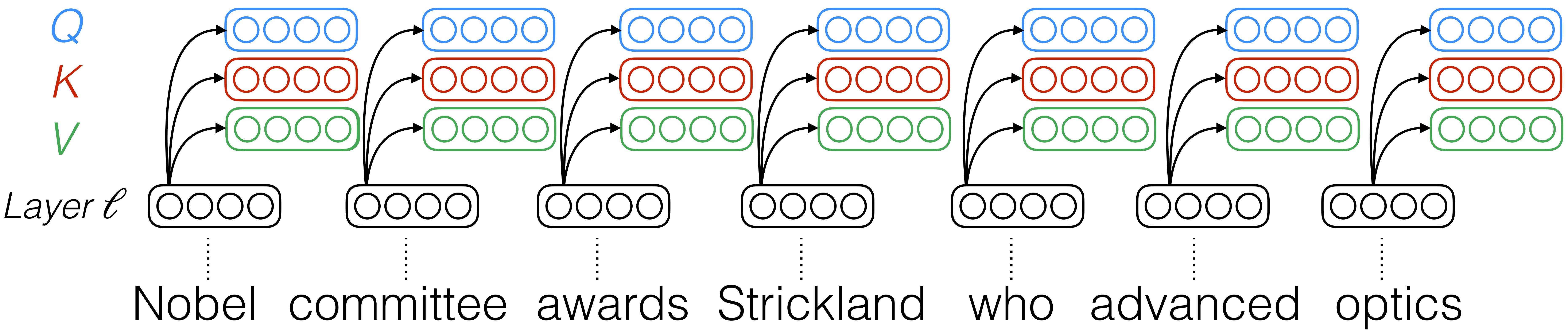


$$\mathbf{a_t} = \frac{(\mathbf{W}^Q Q_t)(\mathbf{W}^K K)}{\sqrt{d}}$$

$$\mathbf{W}^Q, \mathbf{W}^K \in \mathbb{R}^{d \times d}$$

(Vaswani et al., 2017)

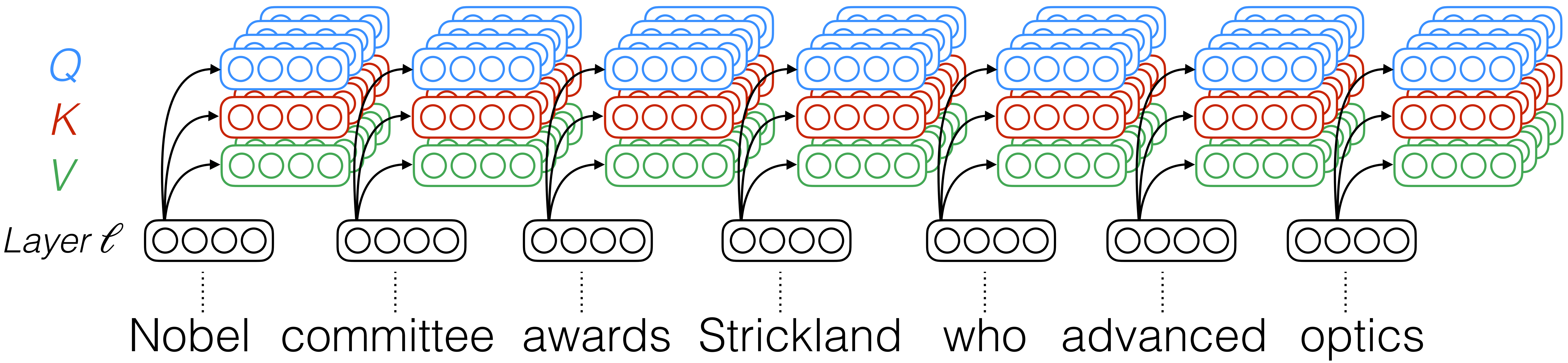
Single Headed Attention:



$$\mathbf{a}_\mathbf{t} = \frac{(\mathbf{W}^Q Q_t)(\mathbf{W}^K K)}{\sqrt{d}}$$

$$\mathbf{W}^Q, \mathbf{W}^K \in \mathbb{R}^{d \times d}$$

Multi-headed Headed Attention:

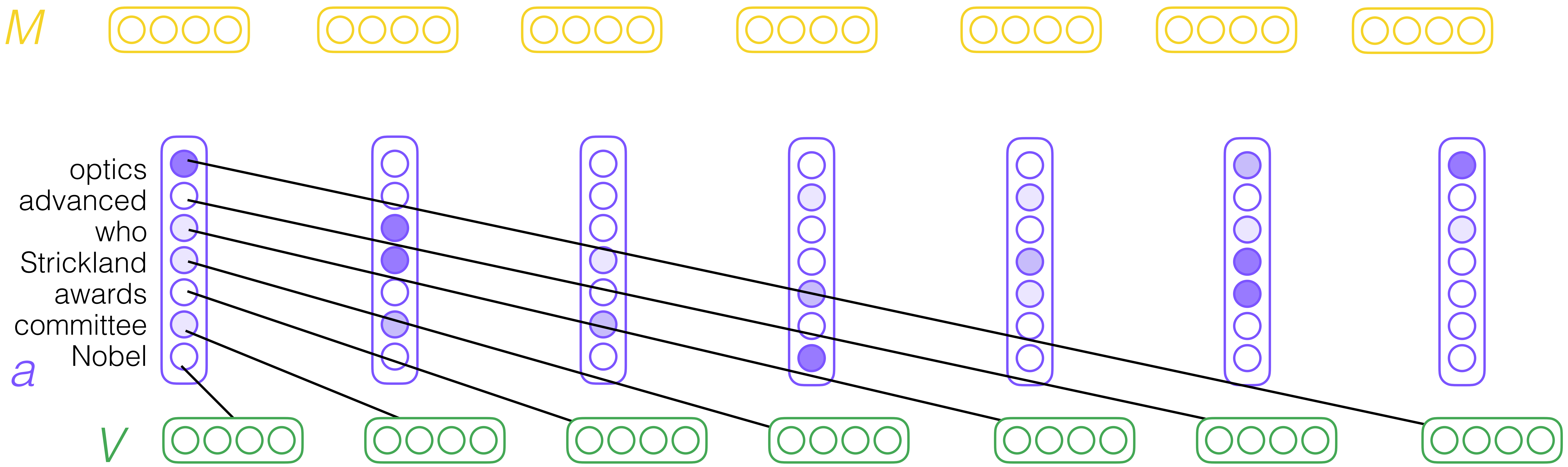


$$\mathbf{a}_{h,t} = \frac{(\mathbf{W}_h^Q Q_t)(\mathbf{W}_h^K K)}{\sqrt{d/H}}$$

$$\mathbf{W}_h^Q, \mathbf{W}_h^K \in \mathbb{R}^{d \times d/H}$$

(Vaswani et al., 2017)

Single Headed Attention:

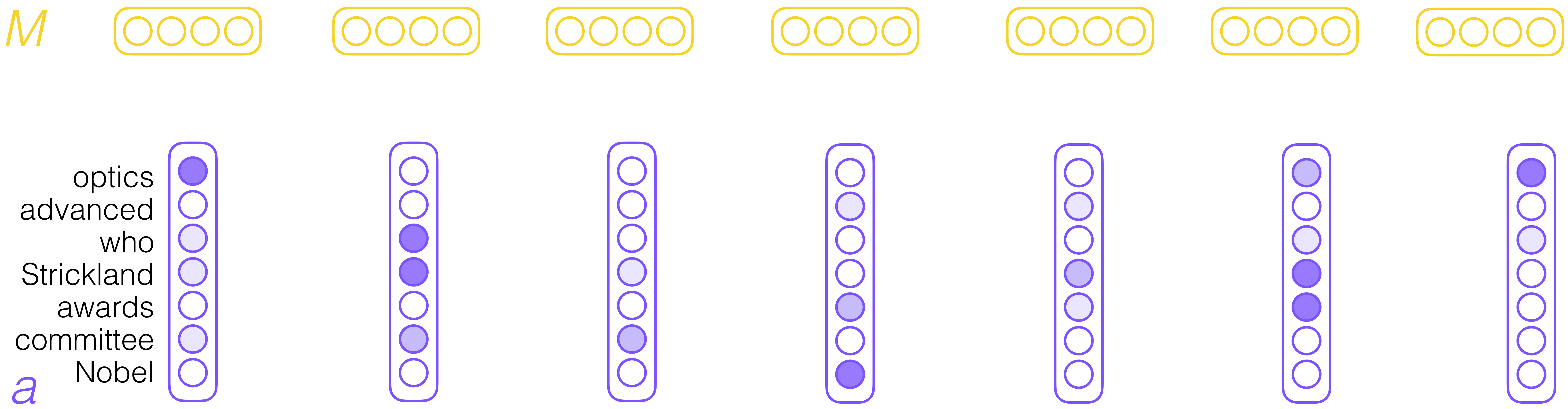


$$M_t = W^O \alpha_t (V\mathbf{W}^V)$$

$$\mathbf{W}^V, \mathbf{W}^O \in \mathbb{R}^{d \times d}$$

(Vaswani et al., 2017)

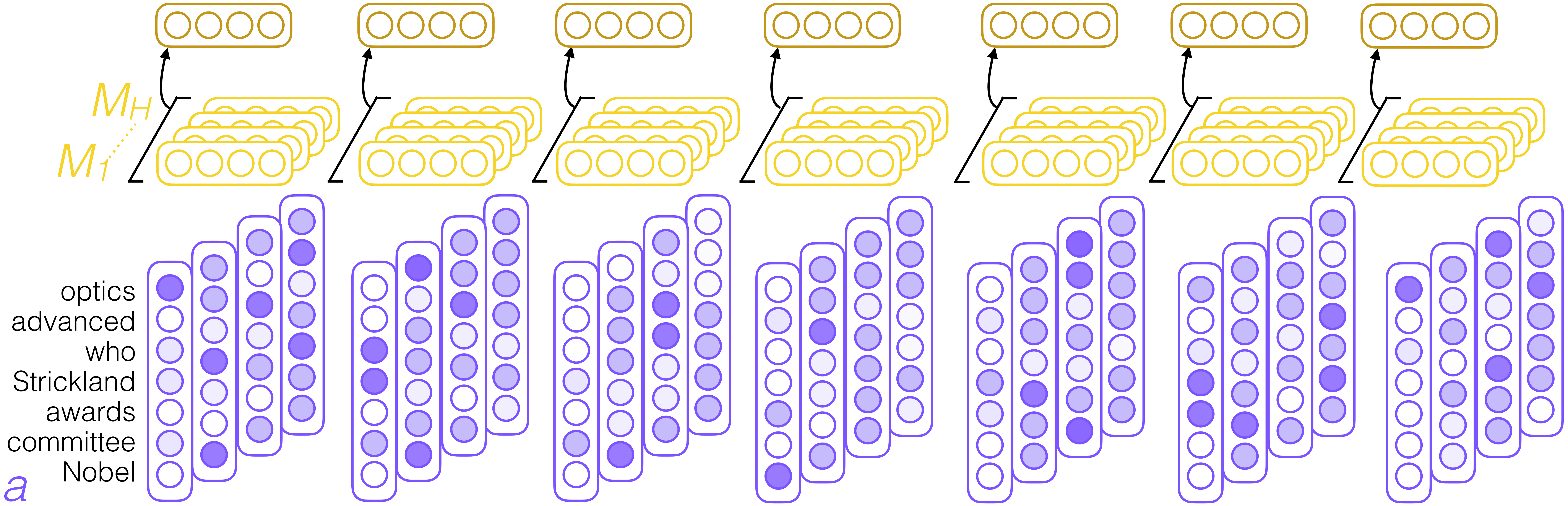
Single Headed Attention:



$$M_t = W^O \alpha_t (V \mathbf{W}^V)$$

$$\mathbf{W}^V, \mathbf{W}^O \in \mathbb{R}^{d\times d}$$

Multi-headed Headed Attention:



$$M_{h,t} = \alpha_{h,t}(V \mathbf{W}_h^V)$$

$$M_t = W^O [M_{1,t}; \ldots ; M_{H,t}]$$

$$\mathbf{W}_h^V \in \mathbb{R}^{d\times d/H}$$

$$\mathbf{W}^O \in \mathbb{R}^{d\times d}$$

(Vaswani et al., 2017)

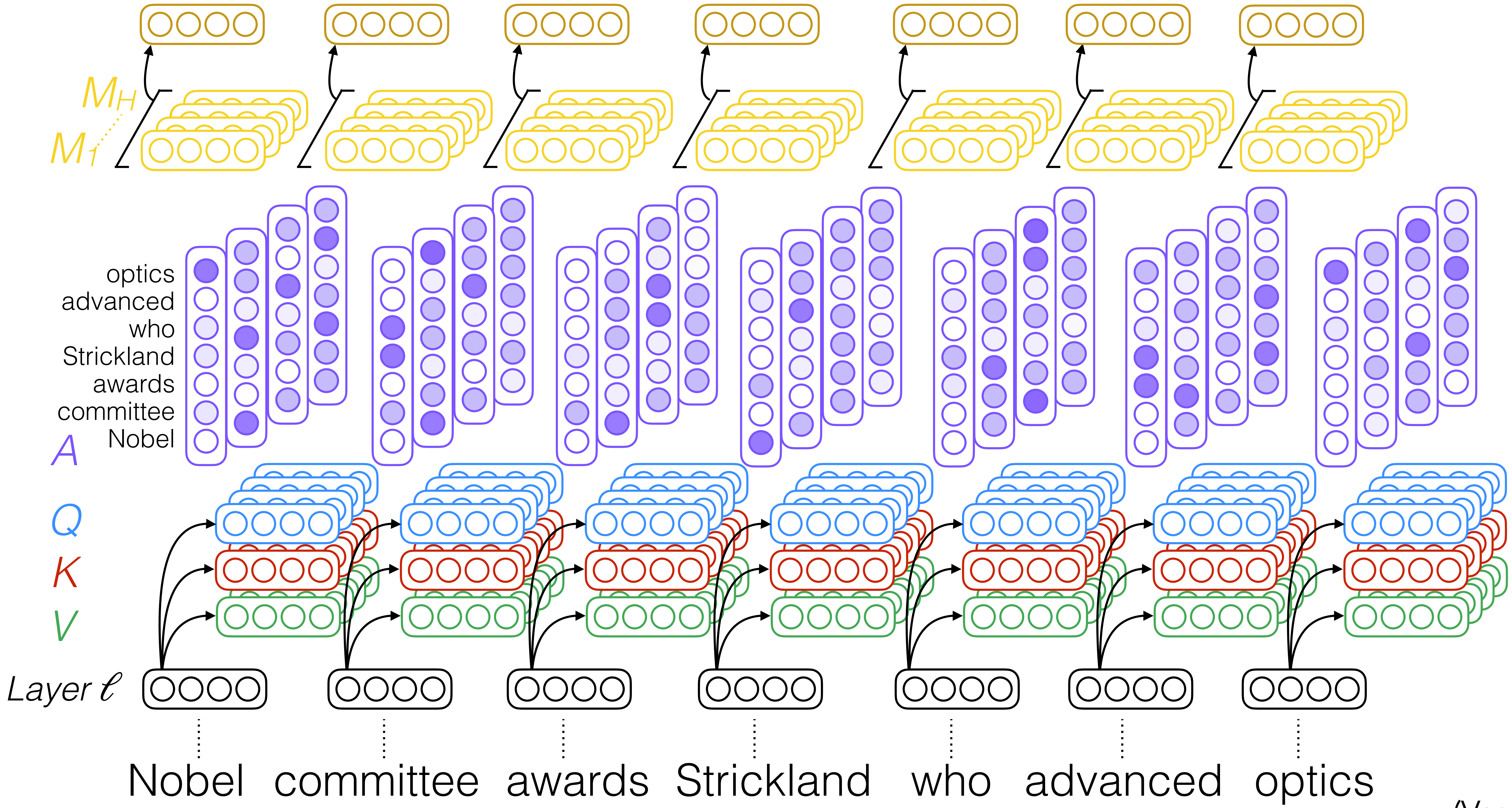
Question

What are the learnable parameters in these matrices?

$$\mathbf{a}_{h,t} = \frac{(\mathbf{W}_h^Q Q_t)(\mathbf{W}_h^K K)}{\sqrt{d/H}} \qquad \alpha_{h,t} = \mathbf{softmax}(\mathbf{a}_{h,t})$$

$$M_{h,t} = \alpha_{h,t}(V\mathbf{W}_h^V)$$

$$M_t = W^O[M_{1,t}; \ldots; M_{H,t}]$$

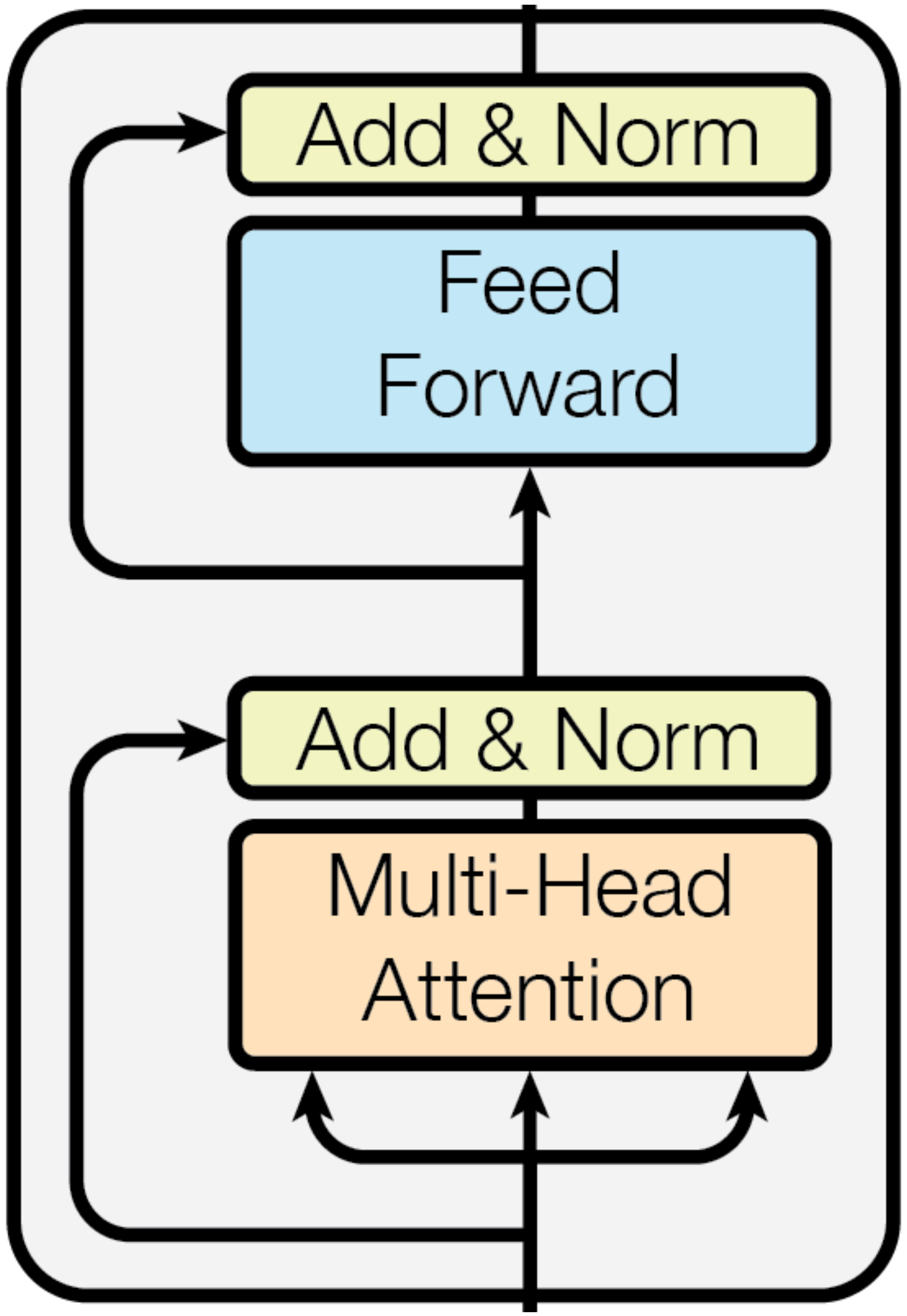


(Vaswani et al., 2017)

Transformer Block

- Multi-headed attention is the main innovation of the transformer model!
- Each block also composed of:
  - a layer normalisations
  - a feedforward network
  - residual connections
- Feedforward network also composed of trainable parameters

$$y = \text{gelu}(W_2\, \text{gelu}(W_1 x + b_1) + b_2)$$



Vaswani et al., 2017

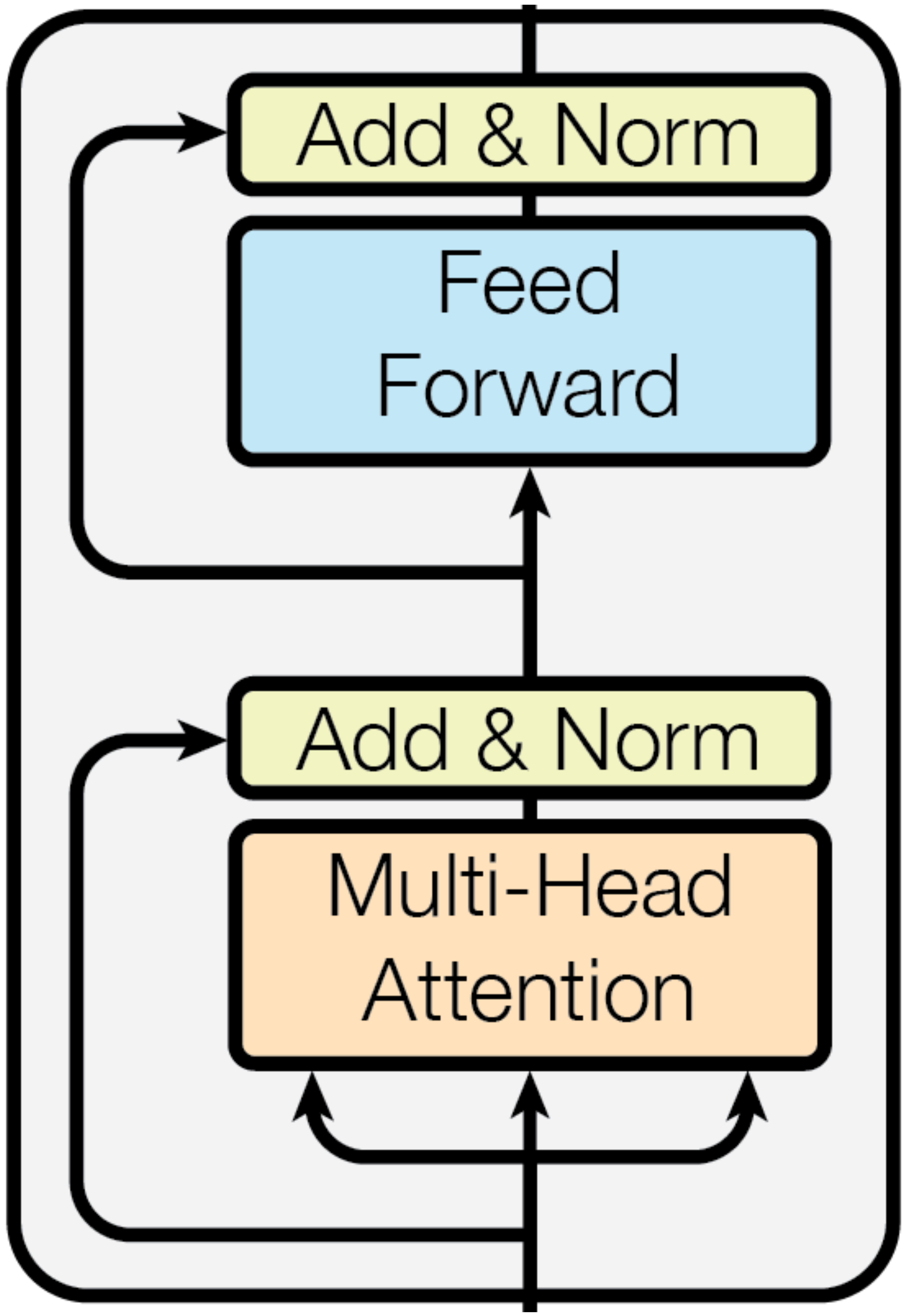
Question

What are the learnable parameters in these matrices?

Transformer Block

- Multi-headed attention is the main innovation of the transformer model!
- Each block also composed of:
  - a layer normalisations
  - a feedforward network
  - residual connections
- Feedforward network also composed of trainable parameters

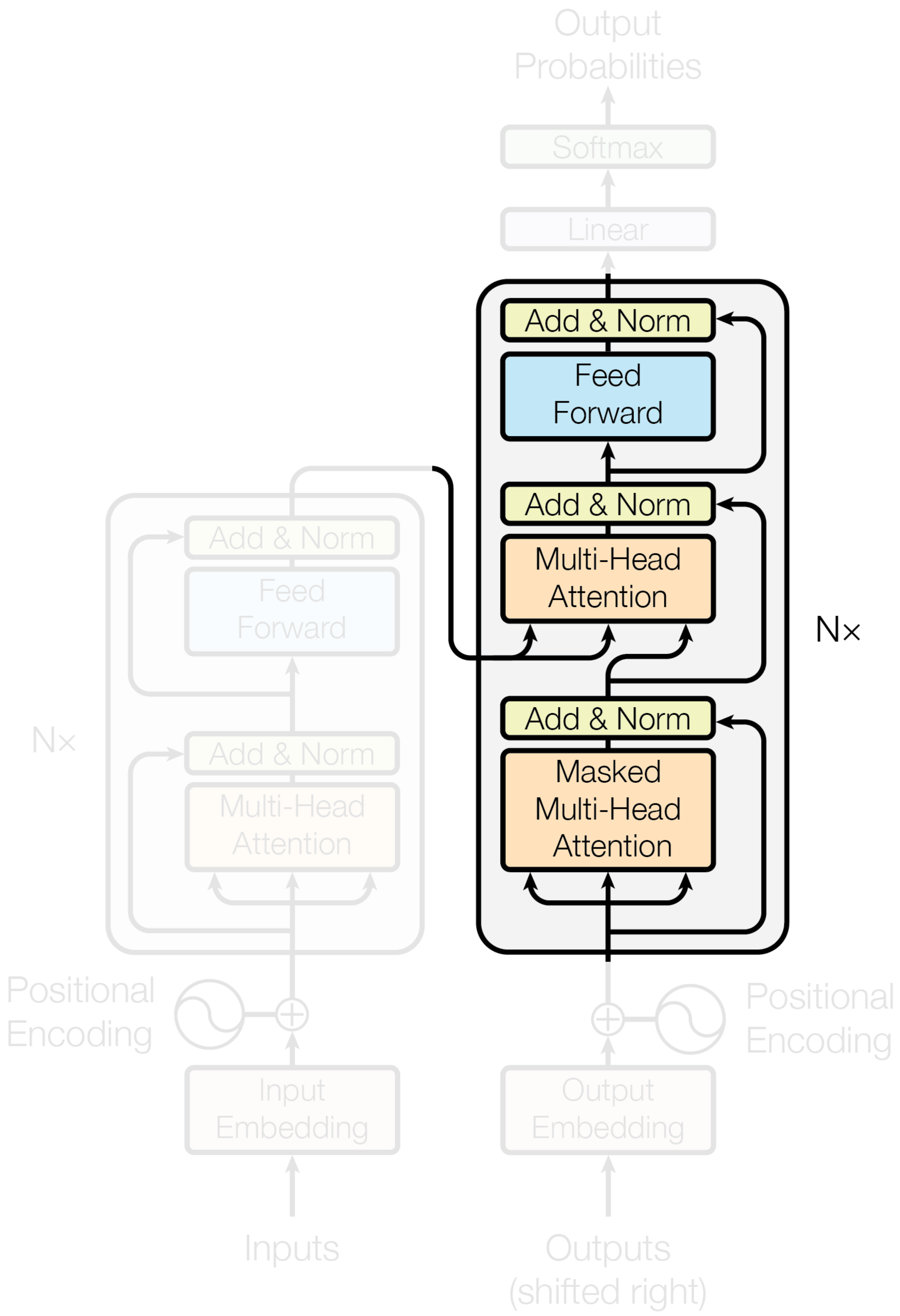
$$y = \text{relu}(W_2\,\text{relu}(W_1 x + b_1) + b_2)$$



Vaswani et al., 2017

Full Transformer

- Transformer decoder (right) similar to encoder
  - First layer of block is **masked** multi-headed attention
  - Second layer is multi-headed attention over *final-layer* encoder outputs **(cross-attention)**
  - Third layer is feed-forward network
- **Different parameters for masked self-attention, cross-attention, and feedforward networks**

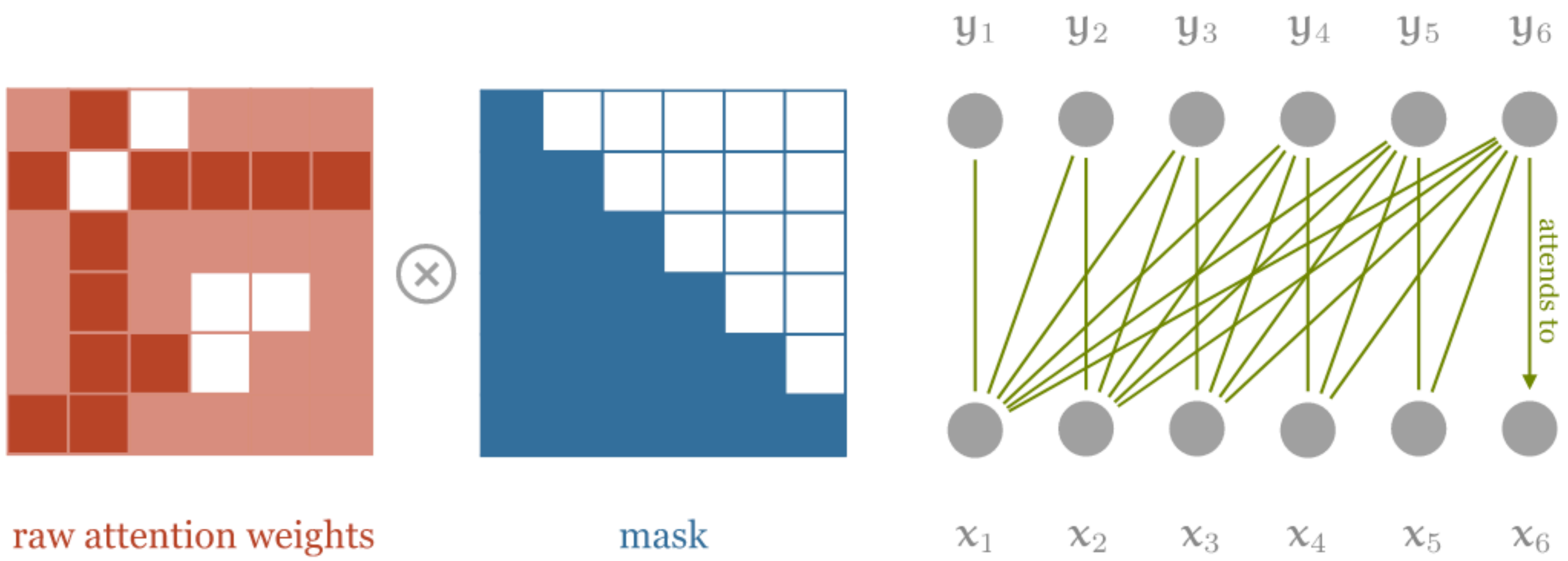


Vaswani et al., 2017

Masked Multi-headed Attention

- Self-attention can attend to any token in the sequence
- For the decoder, **you don't want tokens to attend to future tokens**
  - Decoder used to generate text (i.e., machine translation)

Mask the attention scores of future tokens so their attention = 0



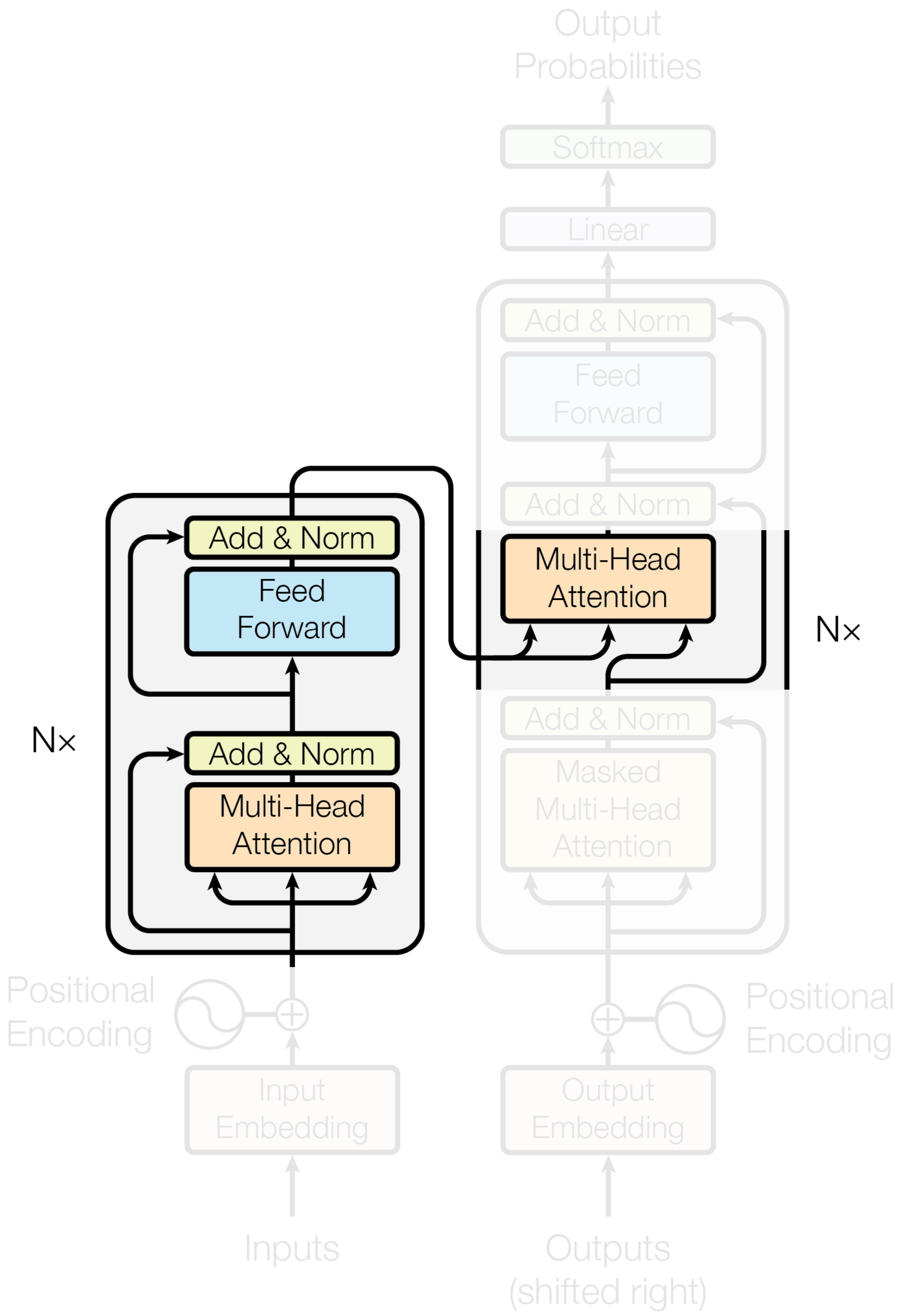
$$a_{st} = \frac{(\mathbf{W}^Q q)^T (\mathbf{W}^K K)}{\sqrt{d}} \quad \Rightarrow \quad a_{st} := a_{st} - \infty \ ; \ s < t \quad \Rightarrow \quad \alpha_{st} = \frac{e^{a_{st}}}{\sum_j e^{a_{sj}}} = 0$$

Cross-attention

- **Cross attention** is the same classical attention as in the RNN encoder-decoder model

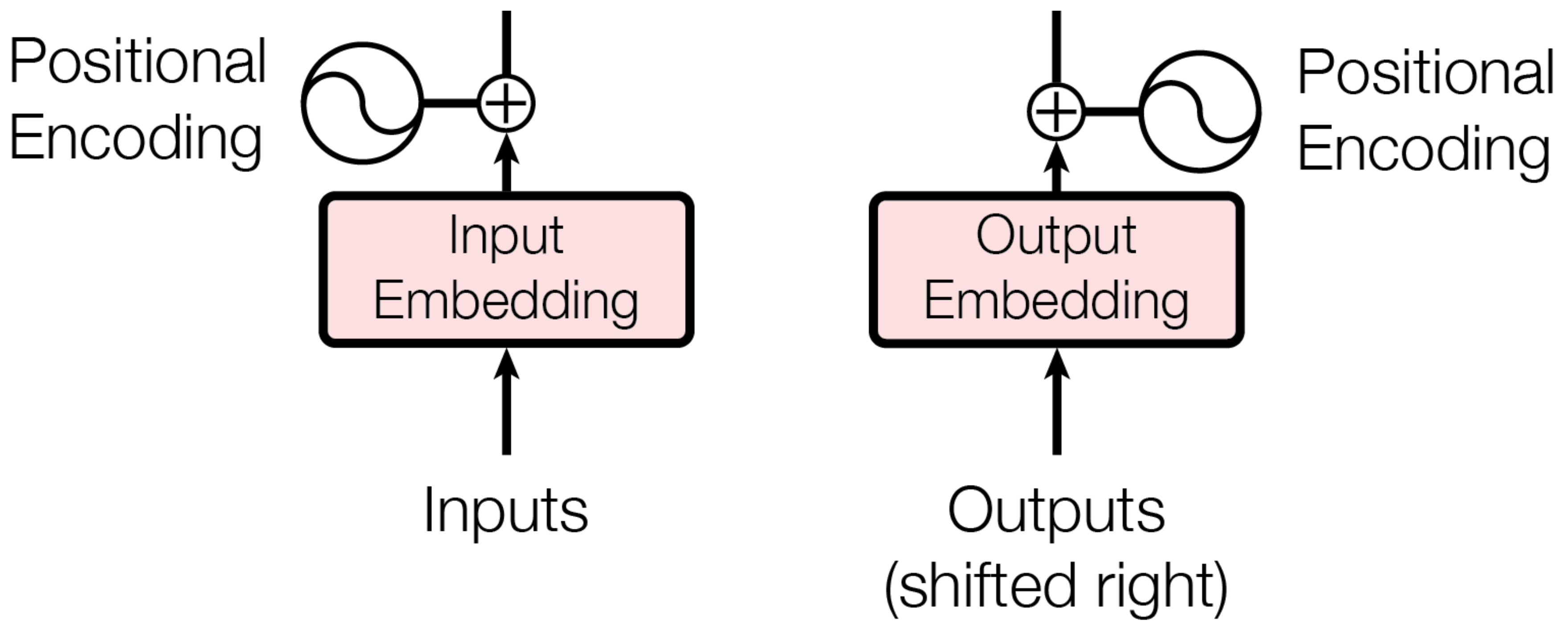
- Keys and values are output of **final** encoder block

- Query to the attention function is output of the masked multi-headed attention in the decoder
  - A representation from the decoder is used to **attend** to the encoder outputs



Vaswani et al., 2017

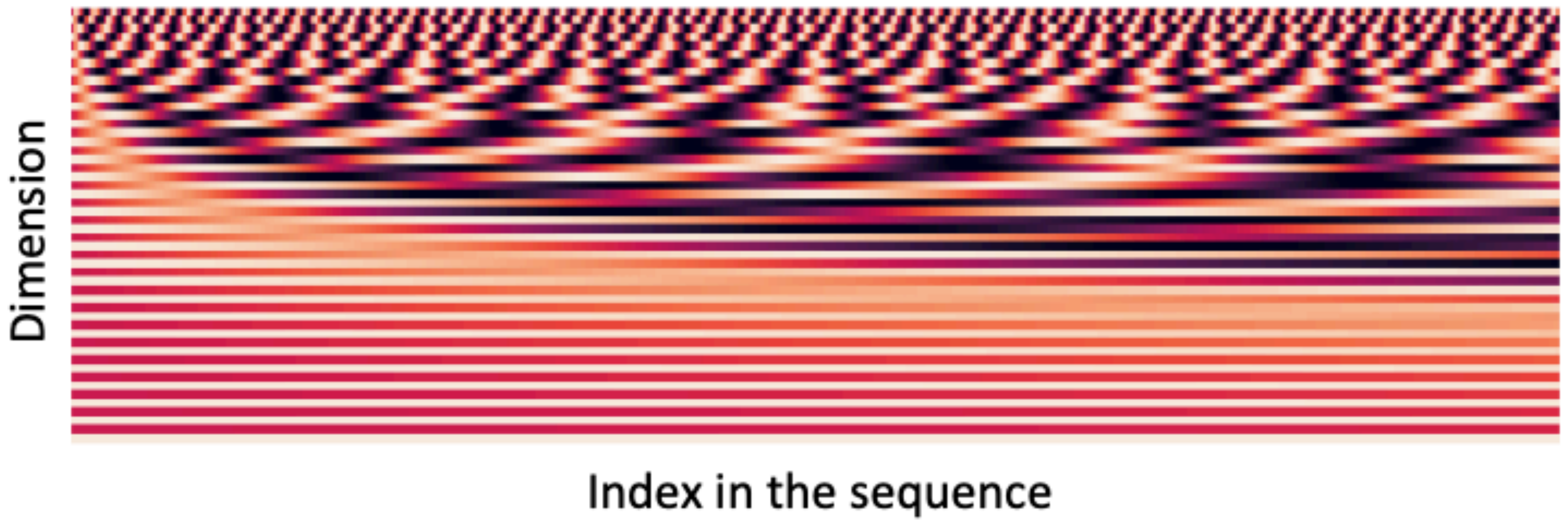
Position Embeddings



- Early position embeddings encoded a sinusoid function that was offset by a phase shift proportional to sequence position

- **In practice, easiest is to learn position embeddings from scratch**

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



Vaswani et al., 2017

Recap

- **Self-attention:** compute representations of tokens as mixtures of surrounding tokens
- Modern **Transformers** use self-attention as fundamental building block
- Decoder blocks mask attention on future tokens to not leak information
- Require position embeddings to capture sequence order

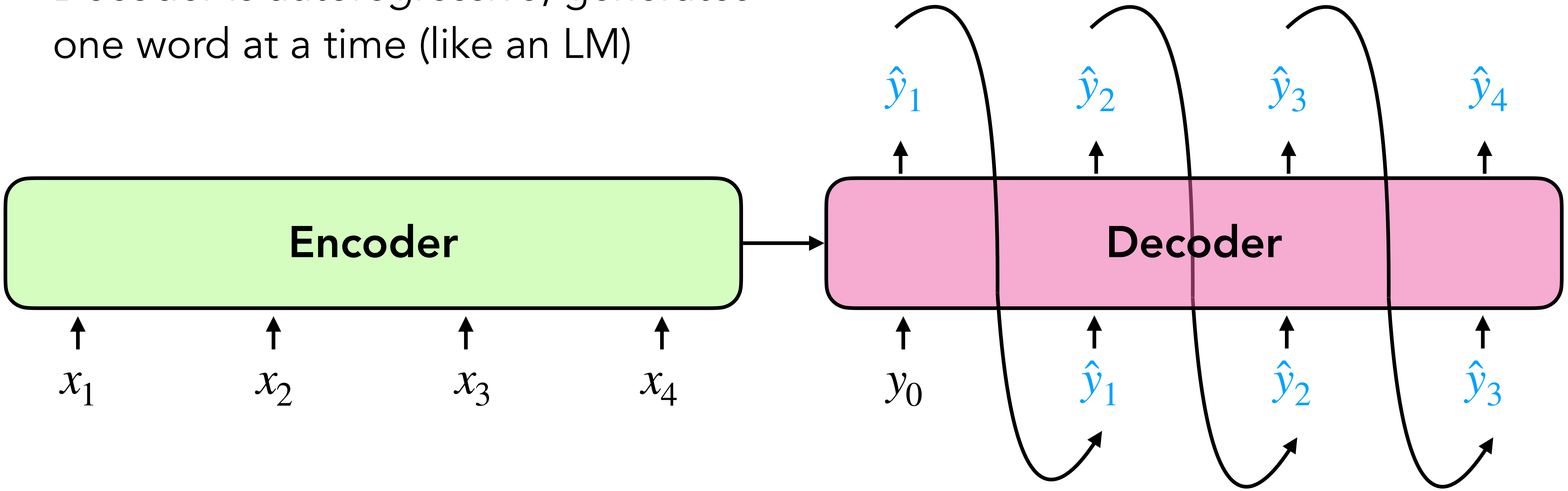
Decoding from Neural Models

Antoine Bosselut

Encoder-Decoder Models

- Encode a sequence fully with one model (**encoder**) and use its representation to seed a second model that decodes another sequence (**decoder**)

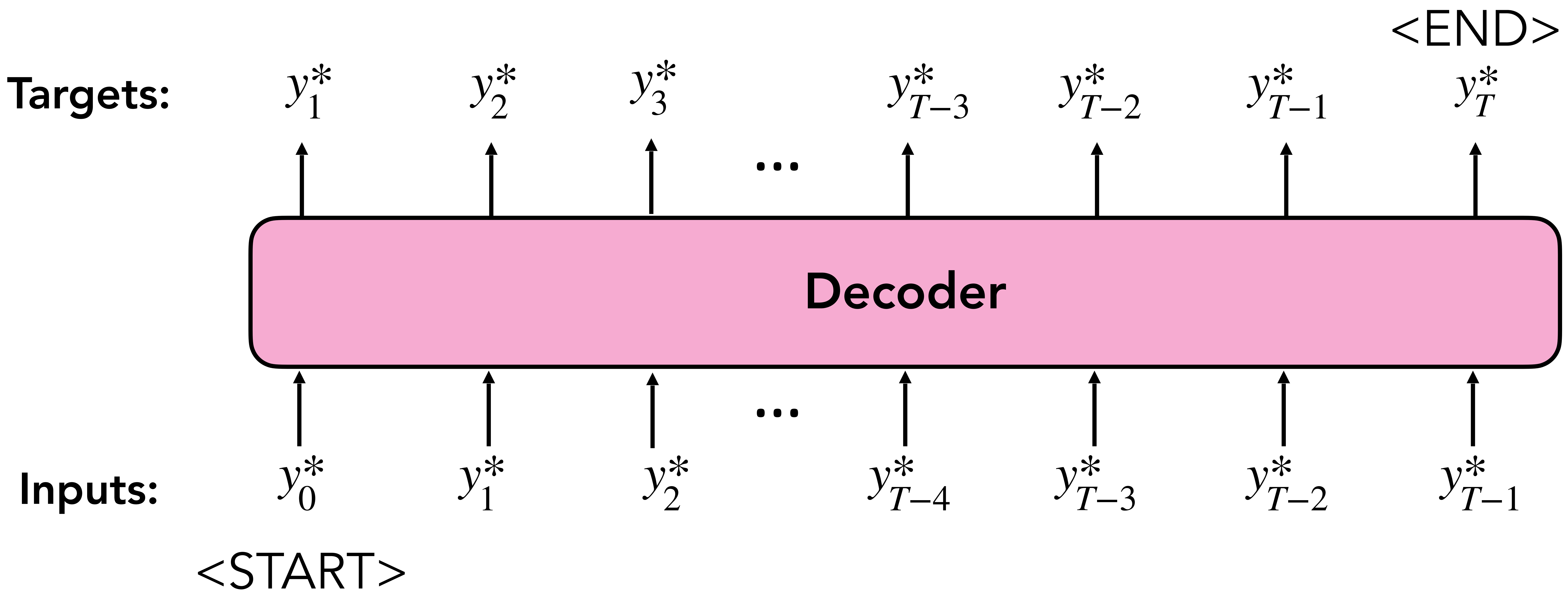
- Decoder is *autoregressive*, generates one word at a time (like an LM)



Training a Decoder

- Minimize the negative log probability of the gold* sequences in your dataset

$$\mathscr{L} = -\sum_{t=1}^{T} \log P\left(y_t^* \mid \{y_s^*\}_{s<t}\right)$$



Decoding: Main Idea

- At each time step *t*, our model computes a vector of scores for each token in our vocabulary, $S \in \mathbb{R}^V$:

$$S = f\left(\{y_{<t}\}\right)$$

$f(\,.\,)$ is your decoder

- Then, we compute a probability distribution P over these scores (with a softmax):

$$P\left(y_t = w \,\middle|\, \{y_{<t}\}\right) = \frac{\exp(S_w)}{\sum_{w' \in V} \exp(S_{w'})}$$

- Decoding algorithm defines a function to select a token from this distribution:

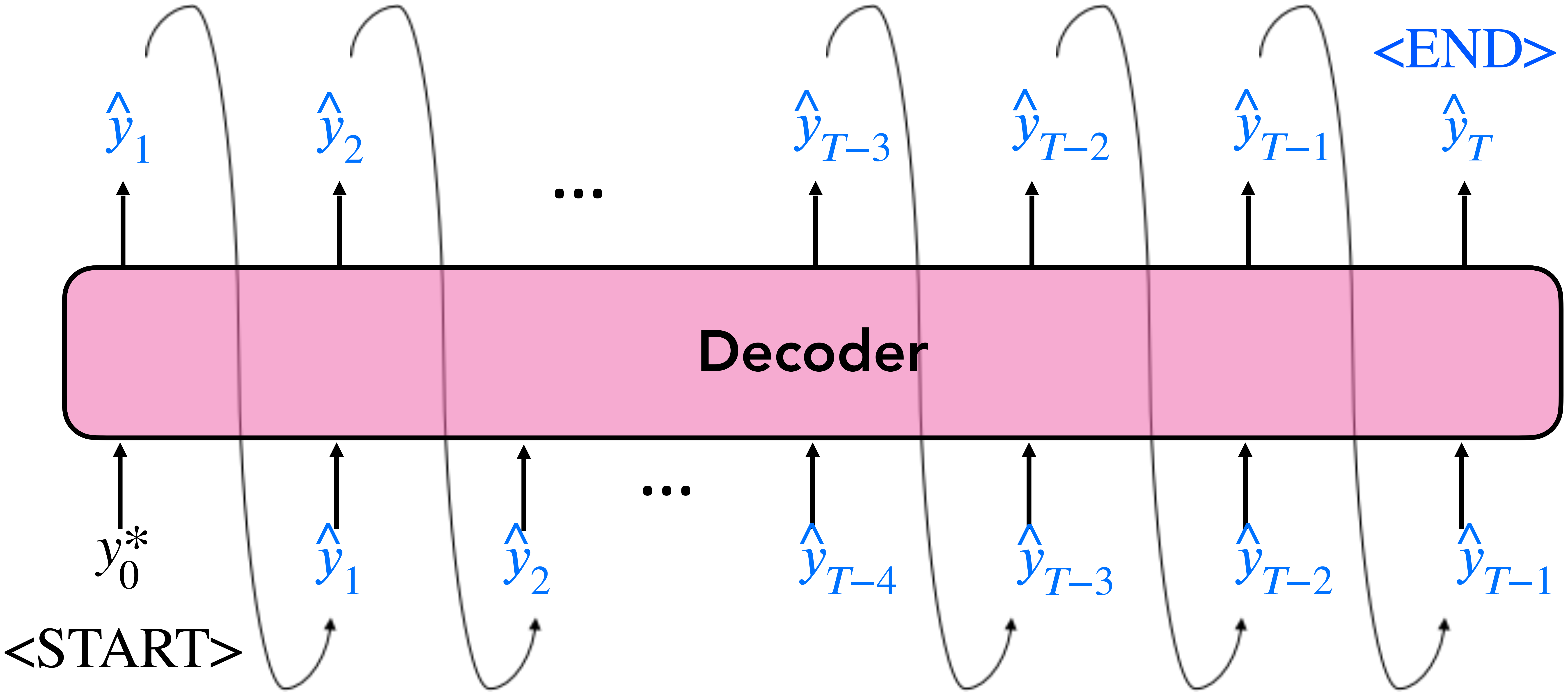
$$\hat{y}_t = g\left(P\left(y_t \,|\, \hat{y}_{<t}\right)\right)$$

$g(\,.\,)$ is your decoding algorithm

Decoding: Main Idea

- Decoding algorithm defines a function to select a token from this distribution

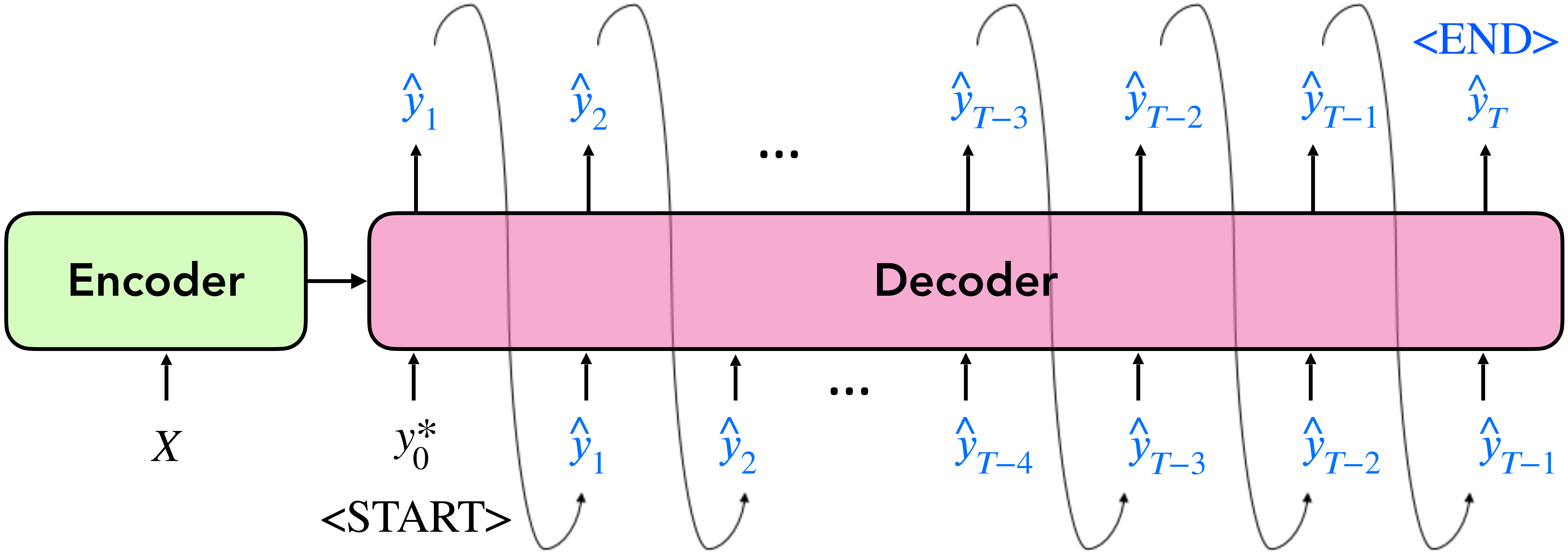
$$\hat{y}_t = g(P(y_t | \hat{y}_{<t}))$$



Optional: Encoder Input

- Decoding algorithm defines a function to select a token from this distribution

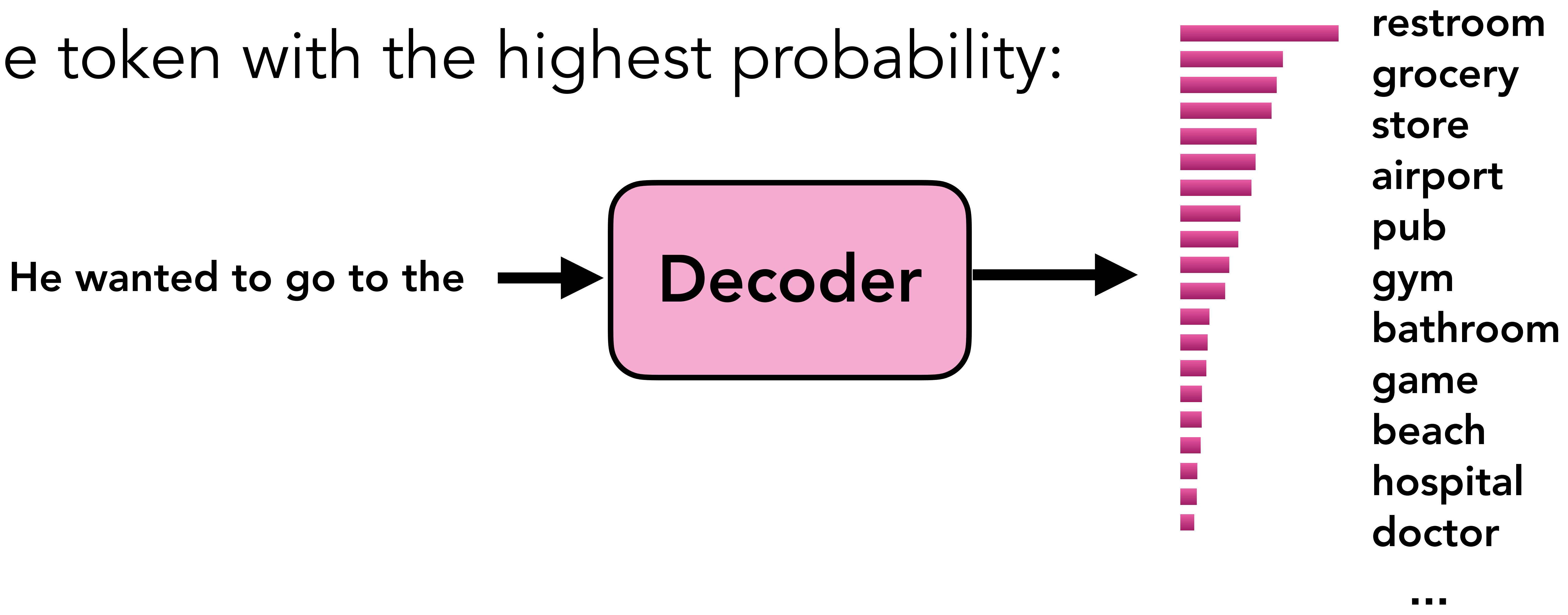
$$\hat{y}_t = g\left(P\left(y_t \mid X, \hat{y}_{<t}\right)\right)$$



Greedy methods: Argmax Decoding

$$\hat{y}_t = \underset{w \in V}{\textbf{argmax}}\ P(y_t = w \mid \{y\}_{<t})$$

- g = select the token with the highest probability:



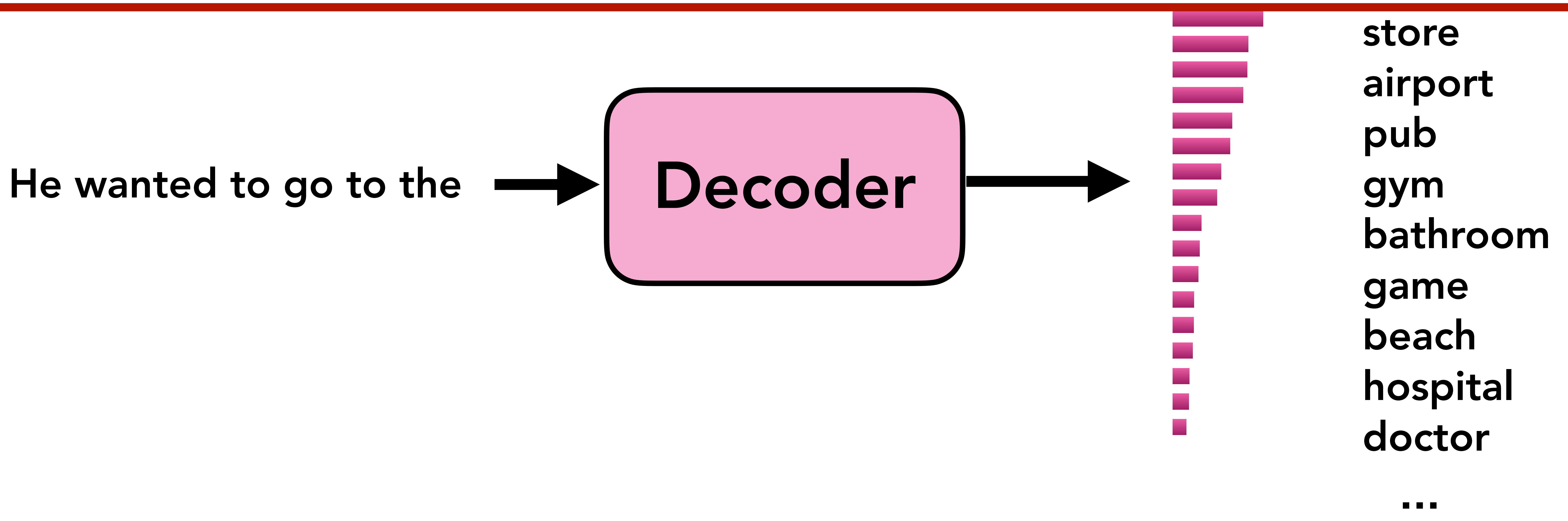
Greedy methods: Argmax Decoding

$$\hat{y}_t = \mathbf{argmax}\ P(y_t = w \mid \{y\}_{<t})$$

Select **highest scoring** token

What's a potential problem with argmax decoding?

- g = selec



Issues with argmax decoding

- In argmax decoding, we cannot revise prior decisions
  - *les pauvres sont démunis (the poor don't have any money)*
  - → *the* ____
  - → *the poor* ____
  - → *the poor are* ____
- Potential leads to sequences that are
  - Ungrammatical
  - Unnatural
  - Nonsensical
  - Incorrect

Beam Search

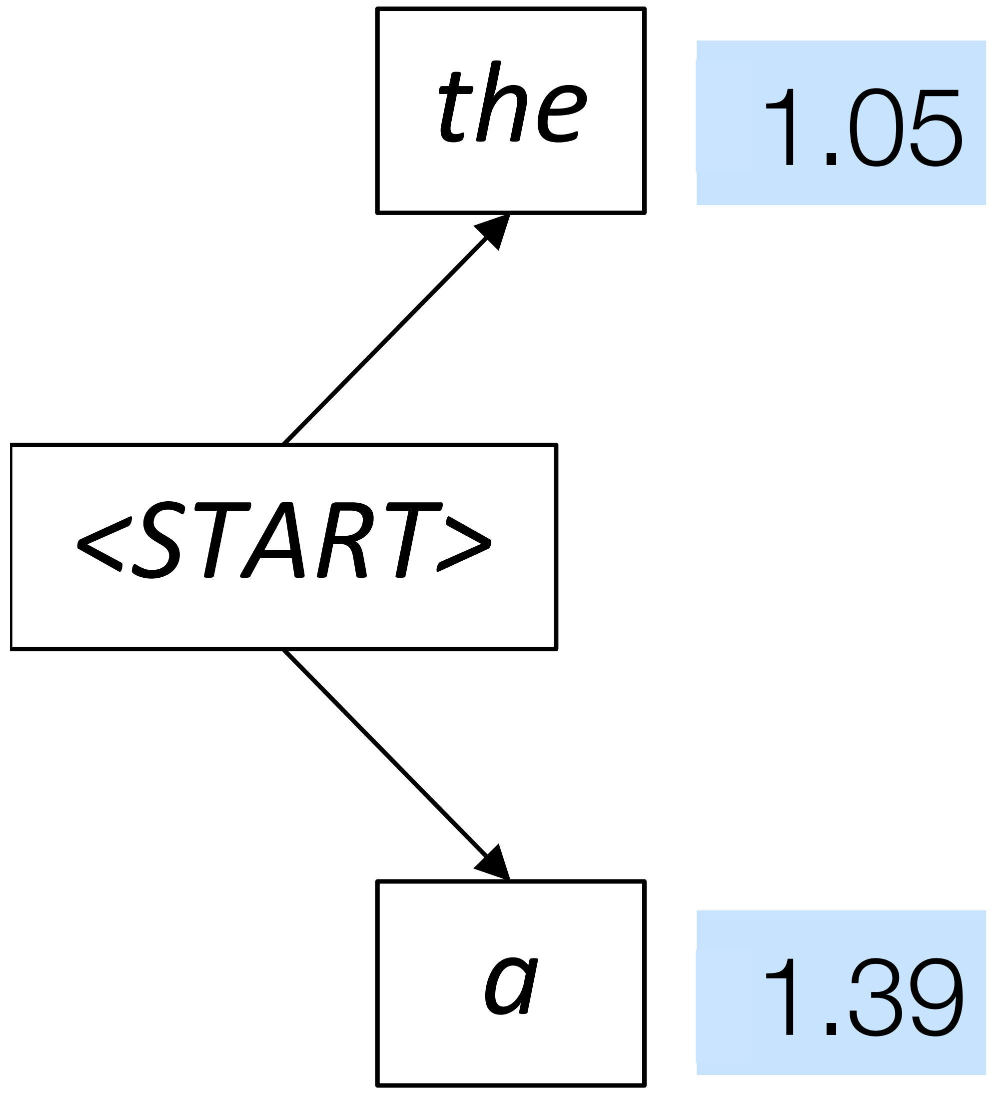
- *les pauvres sont démunis (the poor don't have any money)*
- → *the ____*
- → *the poor ____*
- → *the poor are ____*

- **Beam Search**: Explore several different hypotheses instead of just one
  - Track of the *b* highest scoring sequences at each decoder step instead of just one
  - Score at each step: $-\sum_{t=1}^{j} \log P(\hat{y}_t | \hat{y}_1, \ldots, \hat{y}_{t-1}, X)$
  - *b* is called the **beam size**

Beam Search

Beam size = 2

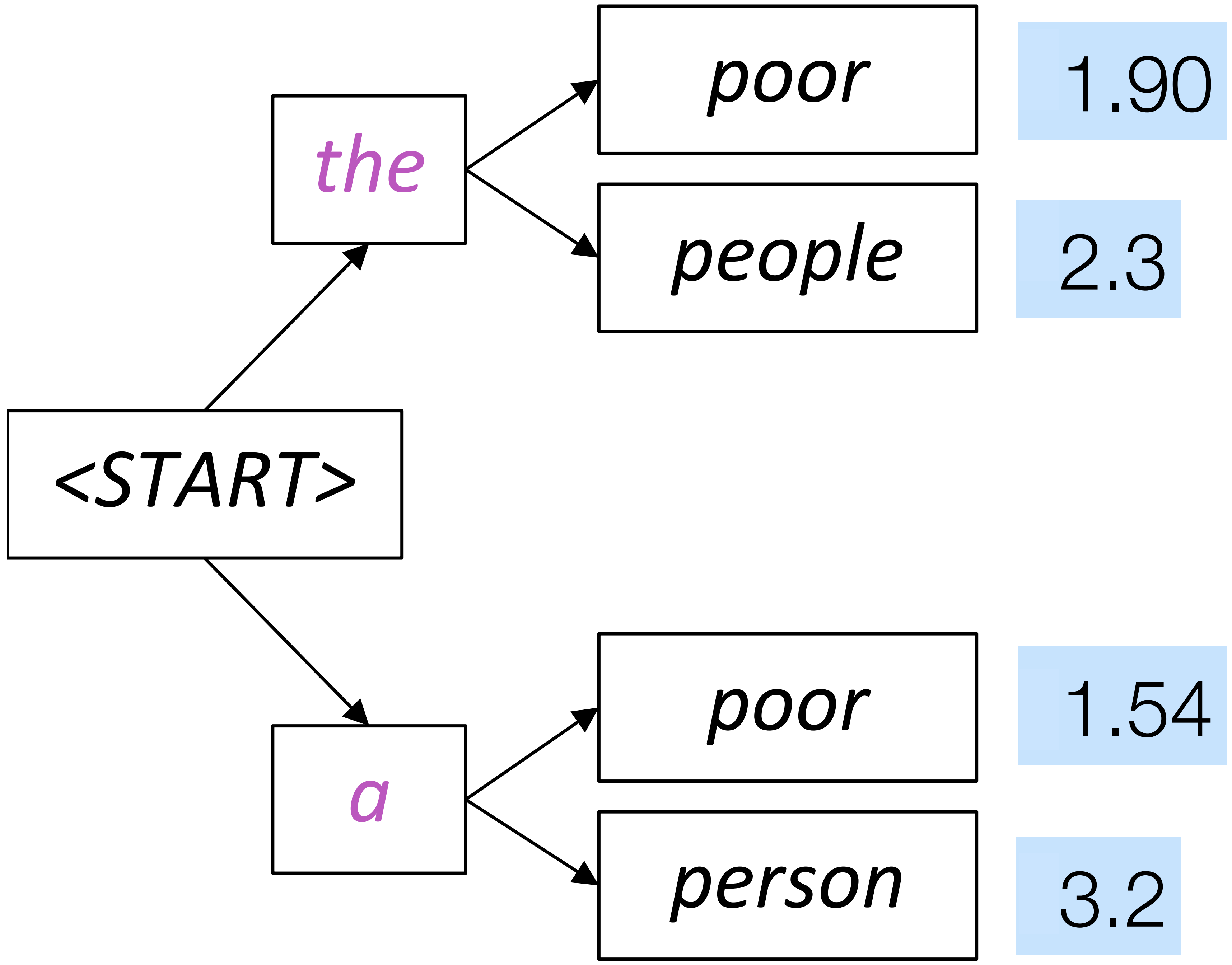
$-\log P(\hat{y}_1 | y_0)$



Beam Search

Beam size = 2

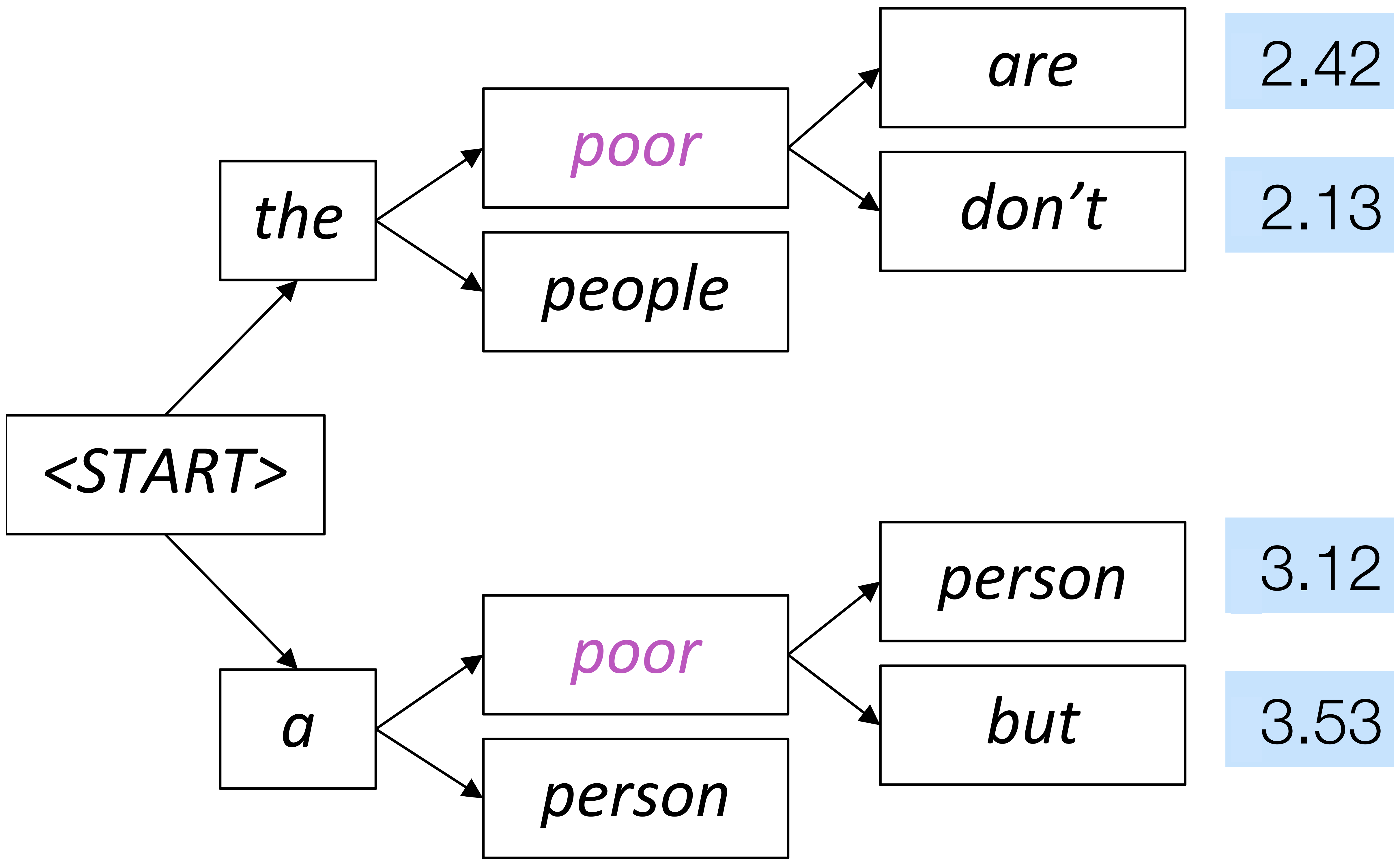
$$-\sum_{t=1}^{2} \log P(\hat{y}_t \mid \hat{y}_0, \ldots, \hat{y}_{t-1})$$



Beam Search

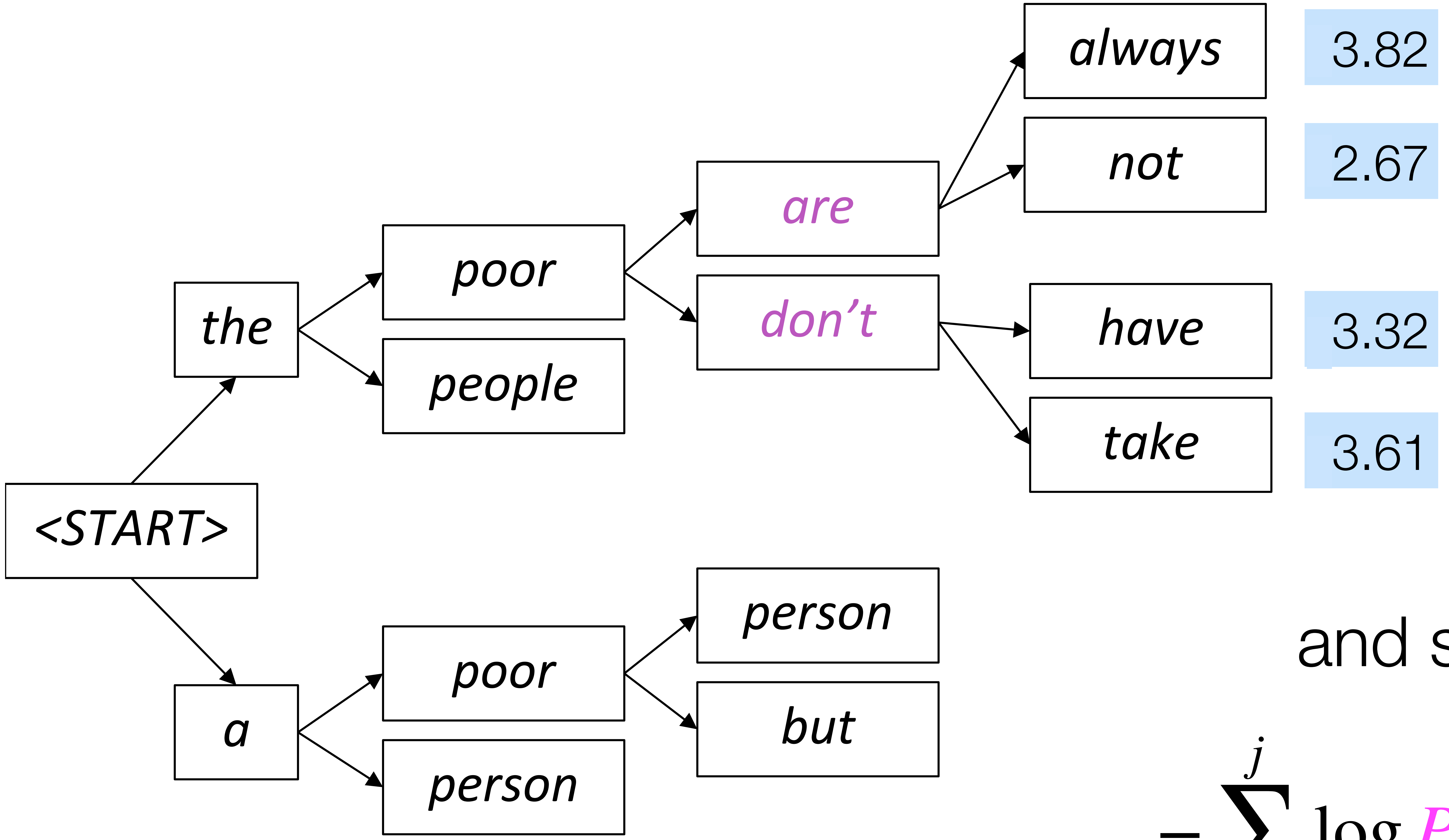
Beam size = 2

$$-\sum_{t=1}^{3} \log P(\hat{y}_t | y_0, \hat{y}_1, \ldots, \hat{y}_{t-1})$$



Beam Search

Beam size = 2

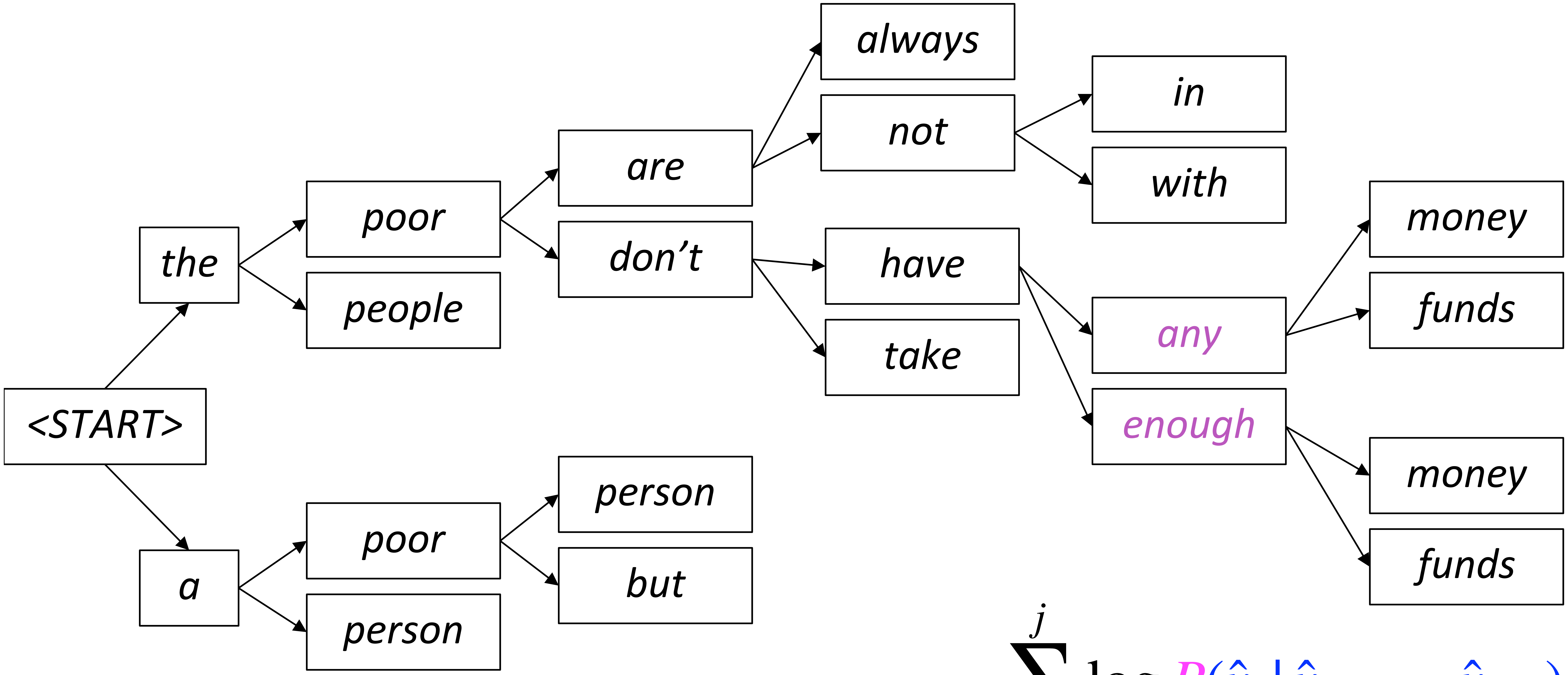


and so on…

$$-\sum_{t=1}^{j} \log P(\hat{y}_t \mid \hat{y}_1, \ldots, \hat{y}_{t-1})$$

Beam Search

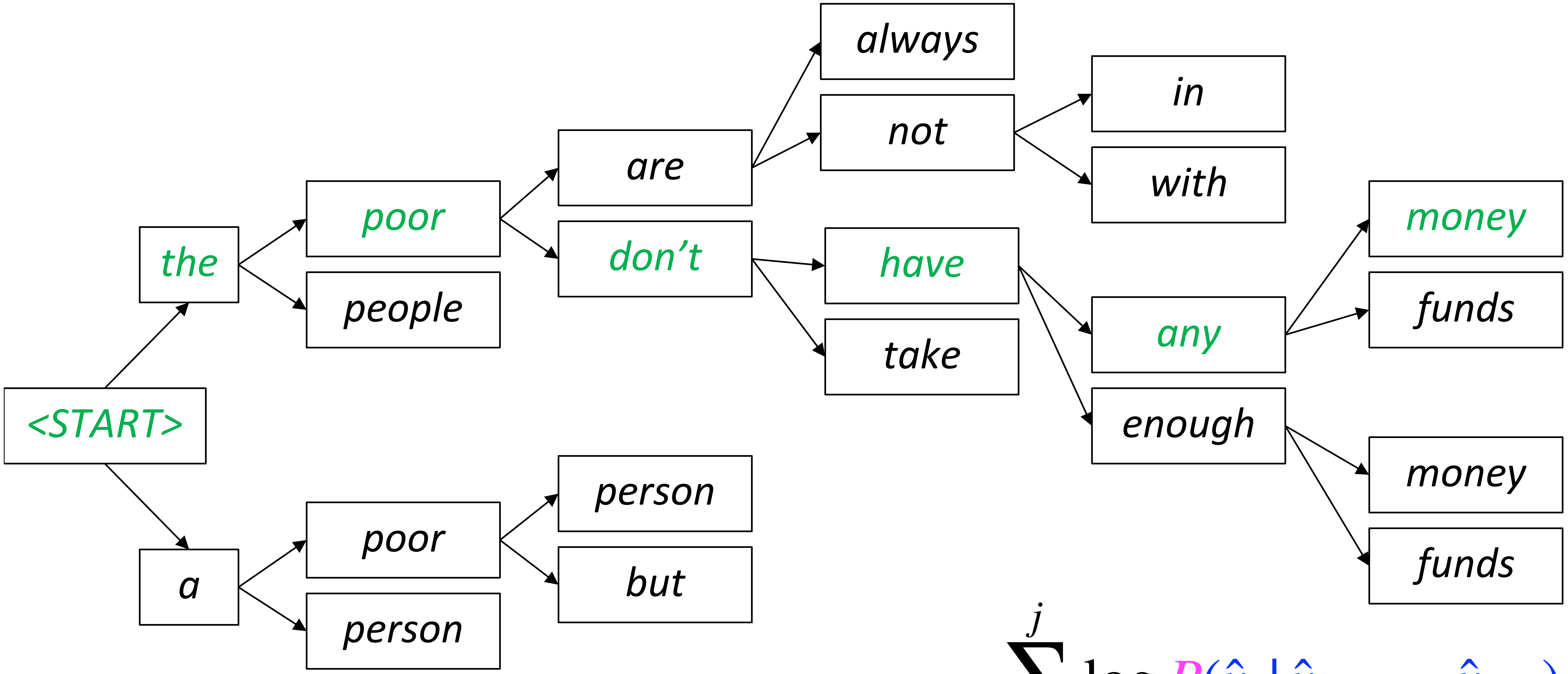
Beam size = 2



$$-\sum_{t=1}^{j} \log P(\hat{y}_t \mid \hat{y}_1, \ldots, \hat{y}_{t-1})$$

Beam Search

Beam size = 2



$$-\sum_{t=1}^{j} \log P(\hat{y}_t \mid \hat{y}_1, \ldots, \hat{y}_{t-1})$$

Beam Search

- Different hypotheses may produce <END> token at different time steps
  - When a hypothesis produces <END>, stop expanding it and place it aside
- Continue beam search until:
  - All b beams (hypotheses) produce <END> OR
  - Hit max decoding limit T
- Select top hypotheses using the *normalized* likelihood score

$$-\frac{1}{T}\sum_{t=1}^{T}\log P(\hat{y}_t \mid \hat{y}_1, \ldots, \hat{y}_{t-1}, X)$$

  - Otherwise shorter hypotheses have higher scores

What do you think might happen if we increase the beam size?

Effect of beam size

- Small beam size *b* has similar issues as argmax decoding
  - **Outputs that are ungrammatical, unnatural, nonsensical, incorrect**
  - b=1 is the same as argmax decoding
- Larger beam size *b* reduces some of these problems
  - Potentially much more computationally expensive
  - Outputs tend to get shorter and more generic

References

- Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. *ArXiv, abs/1706.03762*.