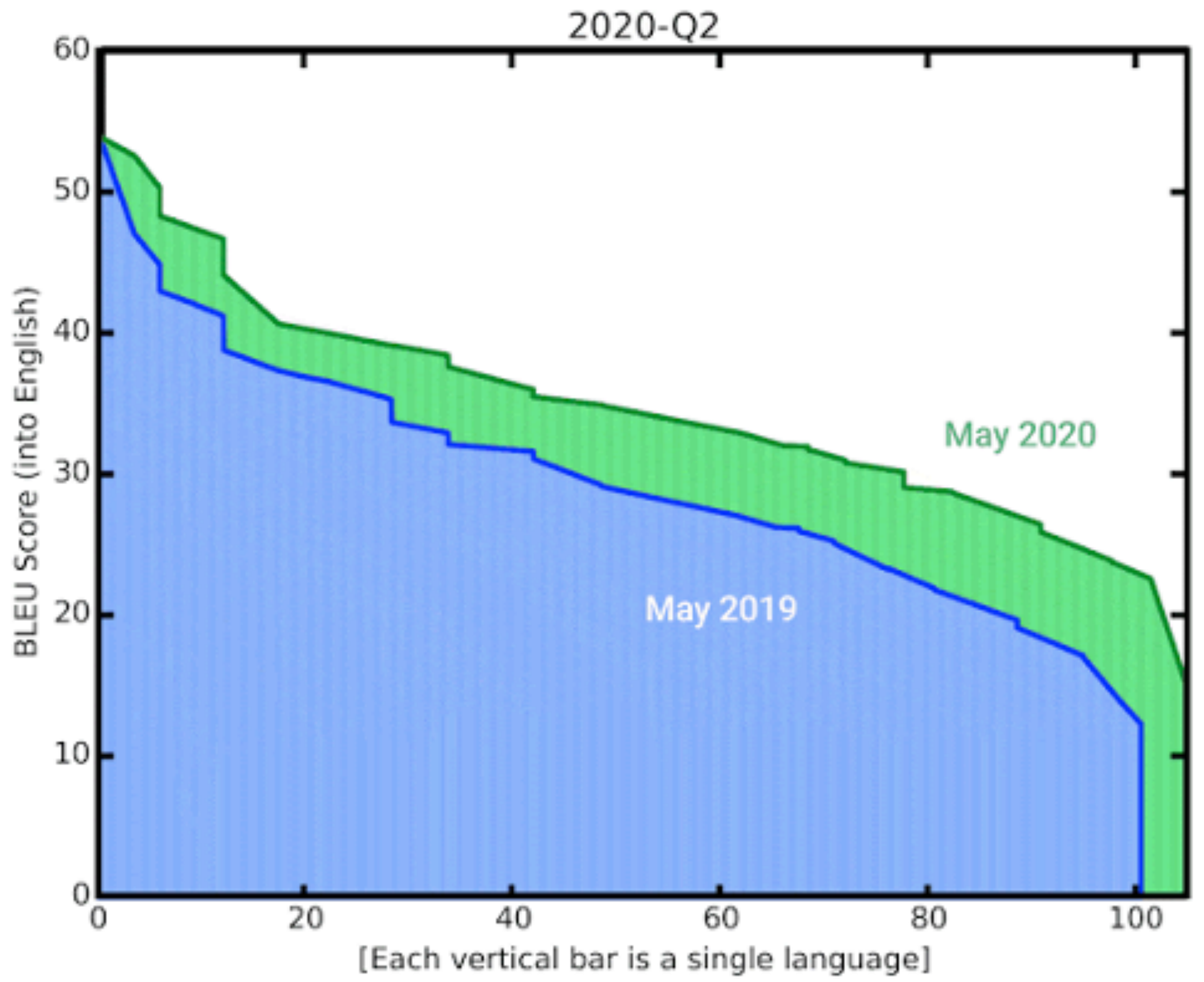
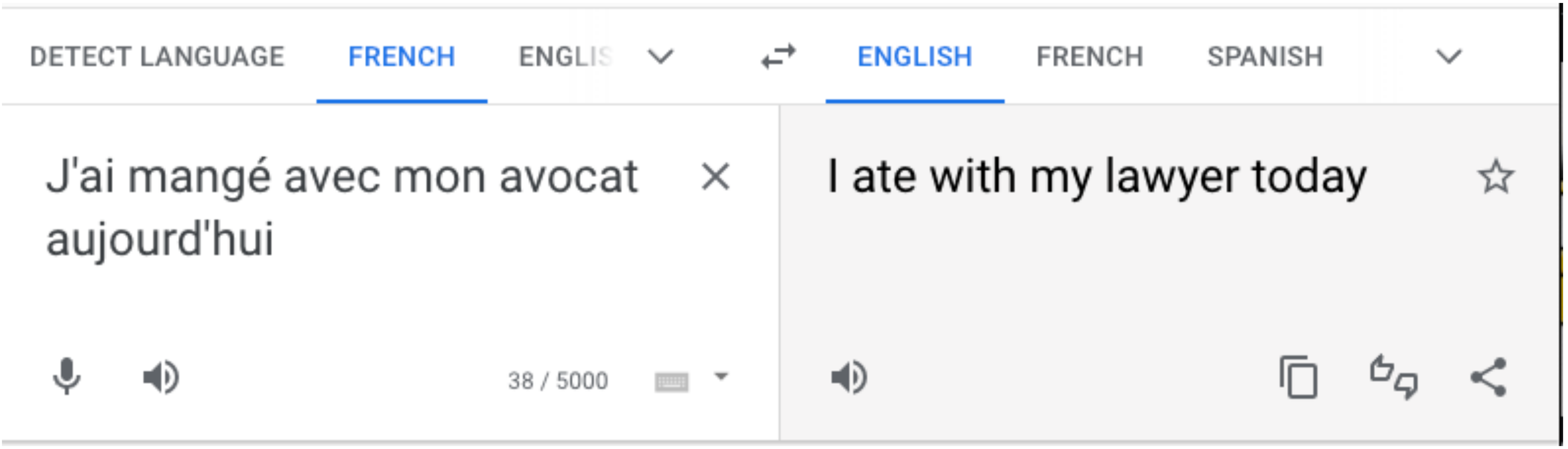
Deep Learning for **Natural Language Processing**

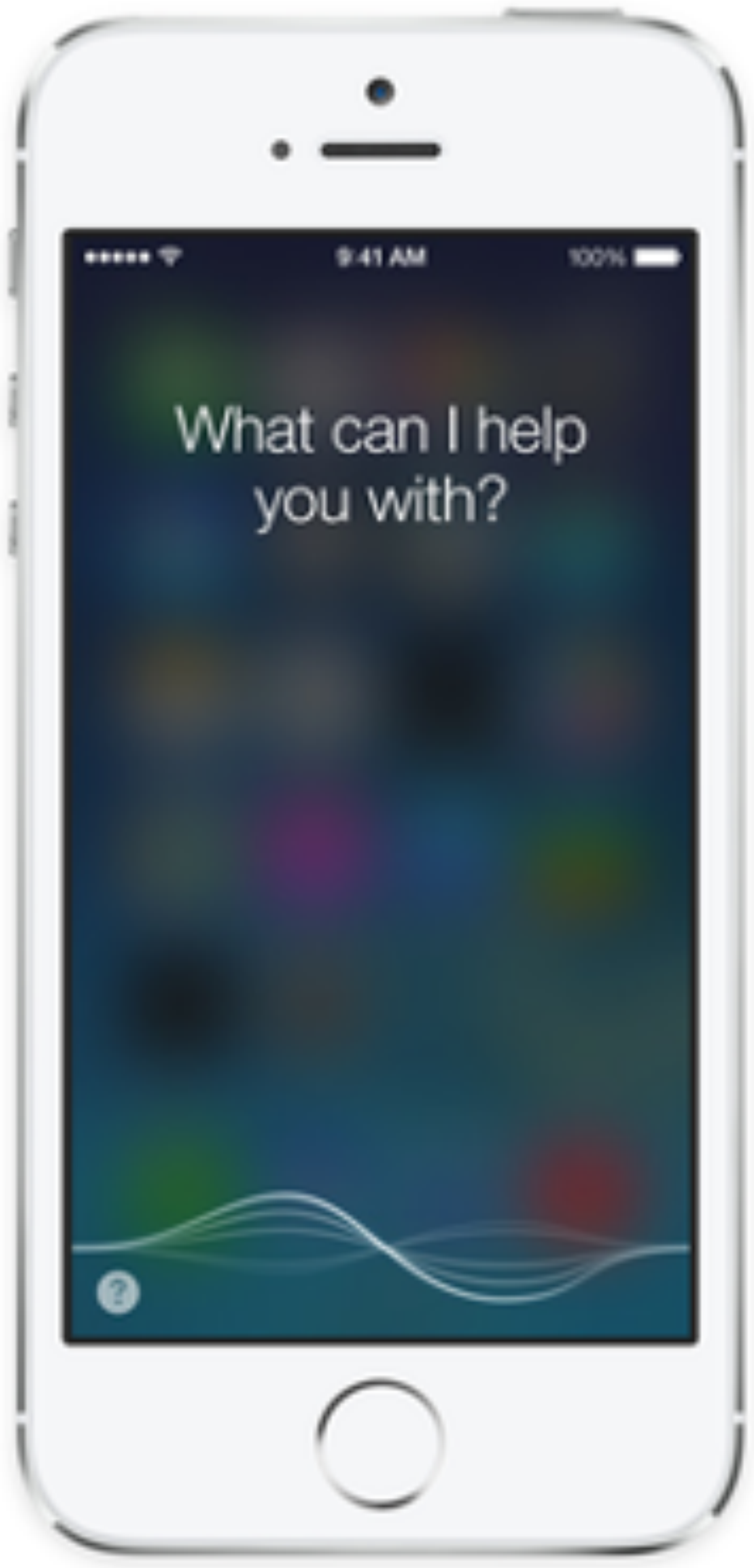
Antoine Bosselut

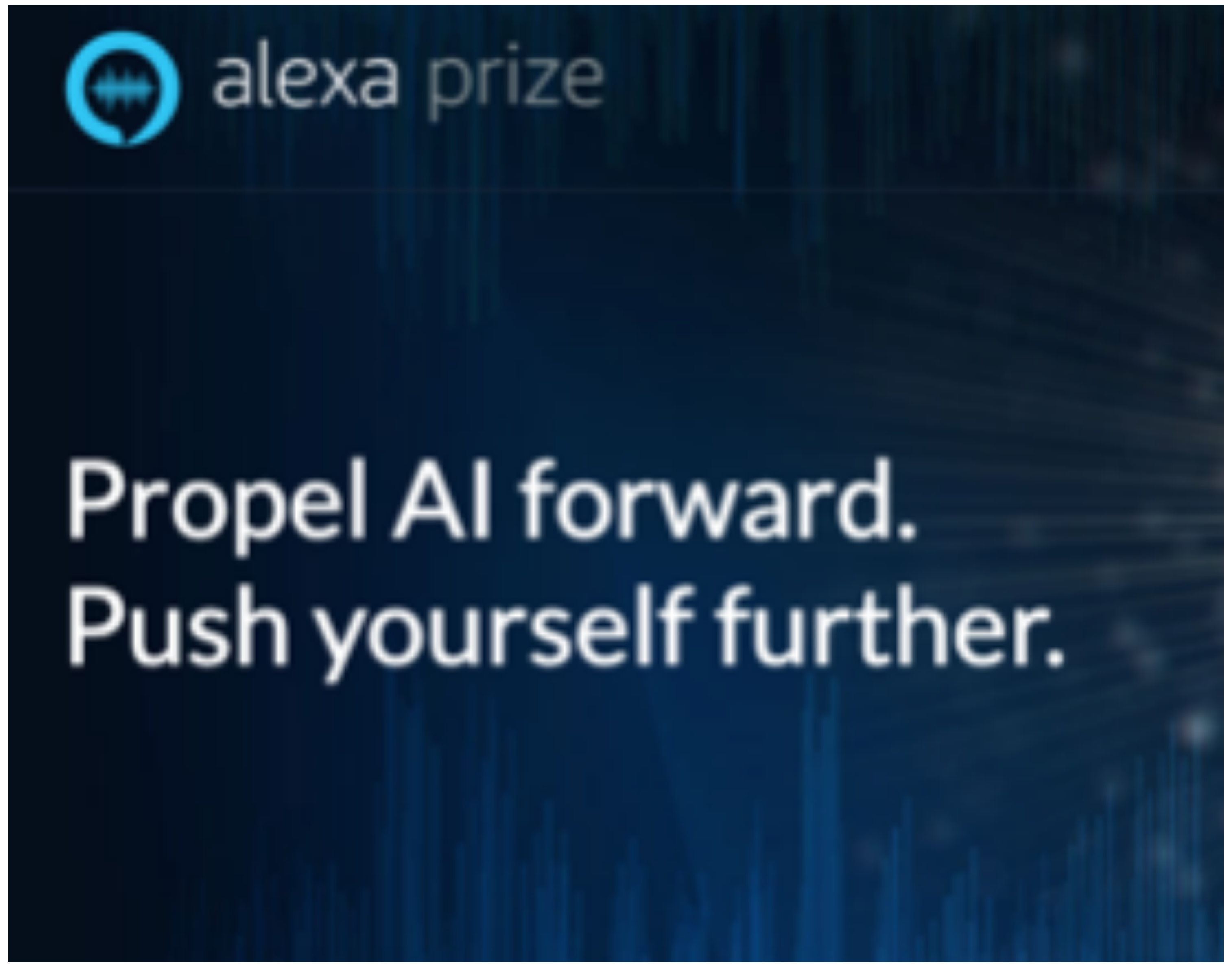
Machine Translation



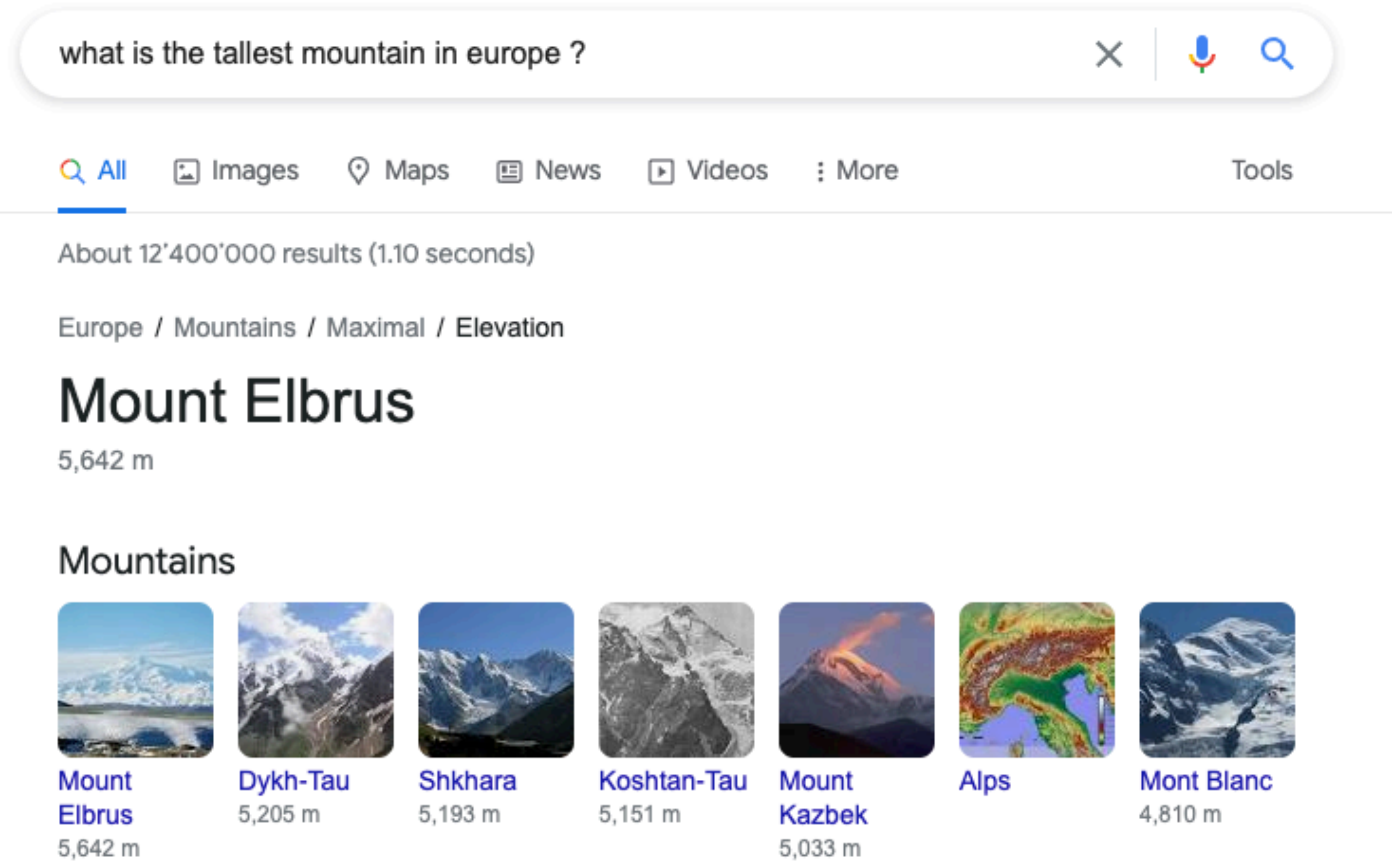


Conversational Systems





Question Answering



Lecture Outline

- **Introduction**
- **Section 1 -** Neural Embeddings
- **Section 2 -** Recurrent Neural Networks for Sequence Modeling
- **Section 3 -** Attentive Neural Modeling with Transformers
- **Section 4 -** Modern NLP: What comes next?

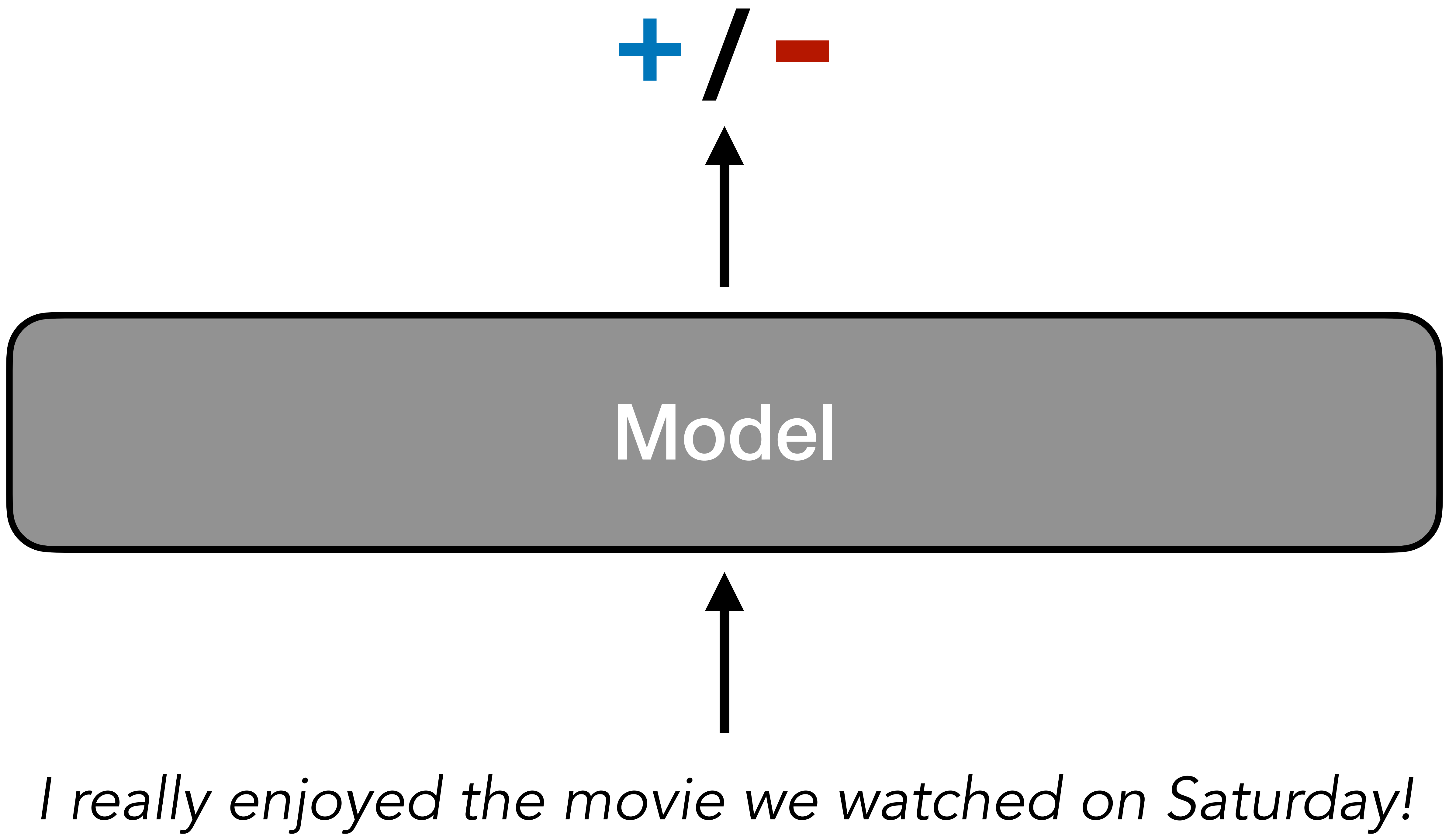
Part 1: Neural Embeddings

Section Outline

- **Review**: sparse word vector representations
- **New:** Dense word vector representations - CBOW & Skipgram
- **Demo:** Similar words for different embedding learning algorithms

Word Representations

- How do we represent natural language sequences for NLP problems?



Sparse Word Representations

$$w_i \in \{0,1\}^V$$

- Define a vocabulary *V*
- Each word in the vocabulary is represented by a sparse *vector*
- Dimensionality of sparse vector is size of vocabulary (e.g., thousands, possibly millions)

I	⟶	[0 … 0 0 0 1 … 0 0]
really	⟶	[0 … 1 … 0 0 0 0 0]
enjoyed	⟶	[0 … 0 0 0 1 0 … 0]
the	⟶	[0 … 0 1 0 0 0 … 0]
movie	⟶	[0 … 0 0 0 0 0 … 1]
!	⟶	[1 … 0 0 0 0 0 0 0 0]

Word Vector Composition

- To represent sequences, beyond words, define a composition function over sparse vectors

I really enjoyed the movie ! → **[1 … 1 1 0 1 … 0 1]** **Simple Counts**

I really enjoyed the movie ! → **[0.01 … 0.1 0.1 0 0.001 … 0 0.5]** **Weighted by Corpus Statistics**

Many others…

Problem

Similarity is only a function of common words!

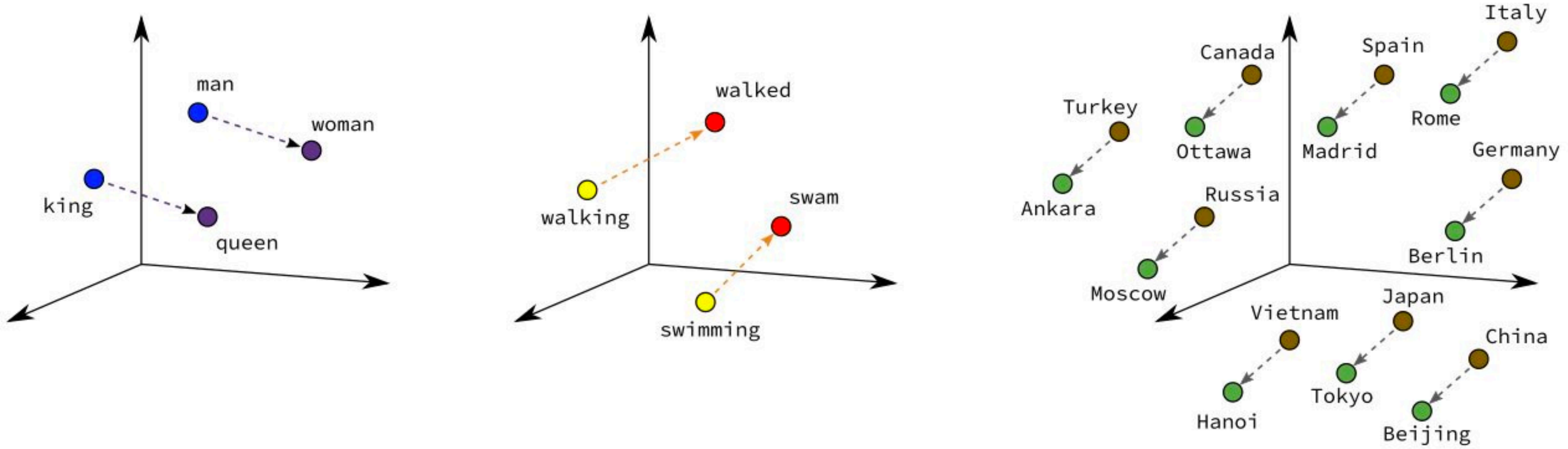
How do you learn learn similarity between words?

enjoyed $\longrightarrow$ [0 … 0 0 0 1 … 0 0]

loved $\longrightarrow$ [0 … 1 … 0 0 0 0 0]

sim(*enjoyed*, *loved*) = **0**

Embeddings Goal



Male-Female

Verb Tense

Country-Capital

How do we train semantics-encoding embeddings of words?

Image Credit: https://towardsdatascience.com/legal-applications-of-neural-word-embeddings-556b7515012f

"You shall know a word by the company it keeps"

–J.R. Firth, 1957

Context Representations

Solution:

Rely on the context in which words occur to learn their meaning

Context is the **set of words** that occur **nearby**

I really enjoyed the ____ we watched on Saturday!

The ___ growled at me, making me run away.

I need to go to the ____ to pick up some dinner.

Context Representations

Solution:

Rely on the context in which words occur to learn their meaning

Context is the **set of words** that occur **nearby**

I really enjoyed the ____ we watched on Saturday!

The ___ growled at me, making me run away.

I need to go to the ____ to pick up some dinner.

Foundation of distributional semantics

Dense Word Vectors

- Represent each word as a high-dimensional*, **real-valued** vector
  - *Low-dimensional compared to V-dimension sparse representations, but still usually $O(10^2 - 10^3)$

I	→	[0.113 -0.782 1.893 0.984 6.349 …]
really	→	[0.906 0.661 -0.214 -0.894 -0.880 …]
enjoyed	→	[-0.842 0.647 -0.882 0.045 0.029 …]
the	→	[0.100 0.765 -0.333 -0.538 -0.150 …]
movie	→	[0.104 -0.054 -0.268 -0.877 0.005 …]
!	→	[0.439 -0.577 -0.727 0.261 0.699 …]

word vectors

word embeddings

neural embeddings

dense embeddings

others…

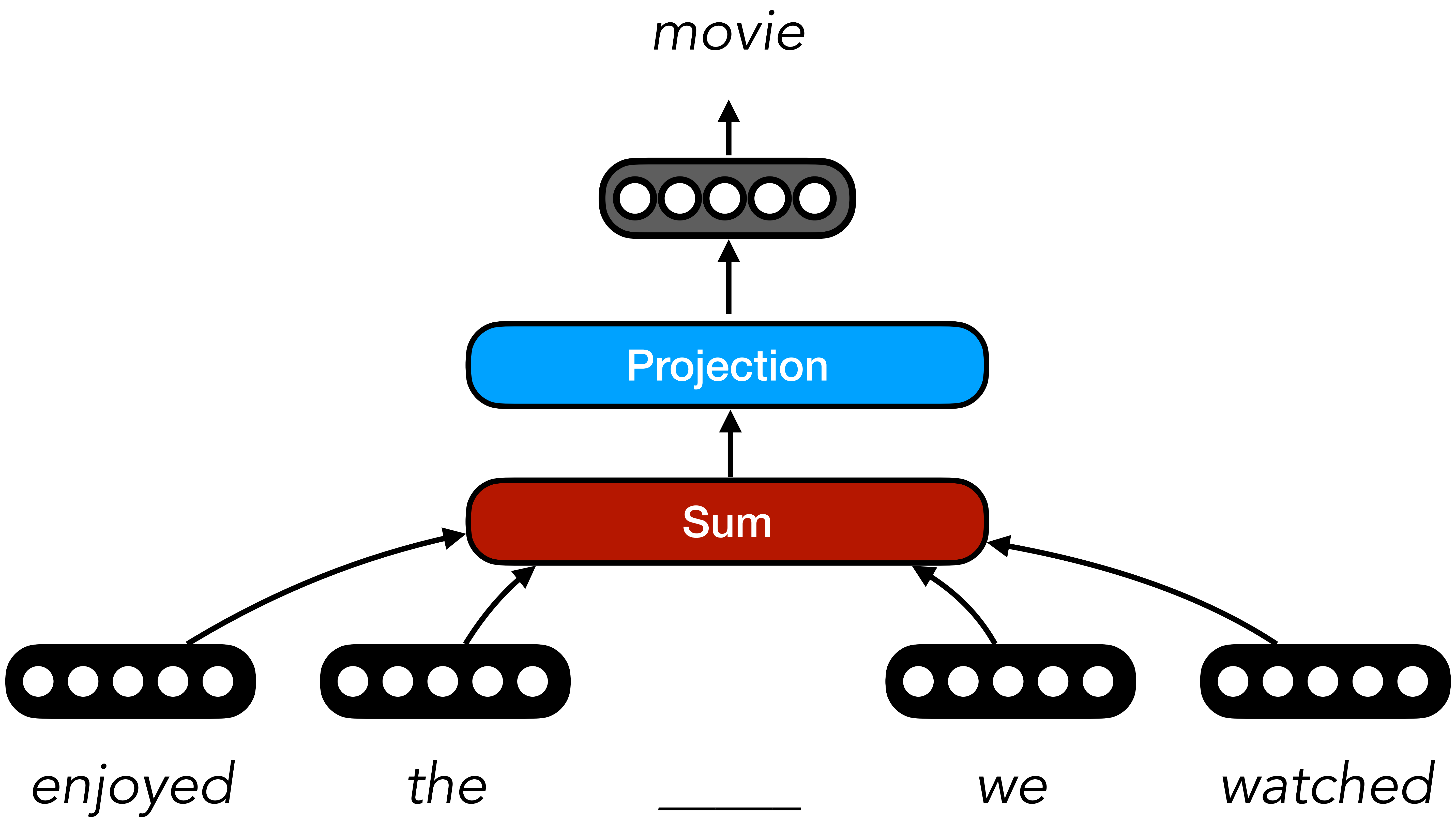
- Similarity of vectors represents similarity of meaning for particular words

Learning Word Embeddings

- Many options, but three common approaches
- **Word2vec - Continuous Bag of Words (CBOW)**
  - Learn to predict missing word from surrounding window of words
- **Word2vec - Skip-gram**
  - Learn to predict surrounding window of words from given word
- **GloVe**
  - Not covered today

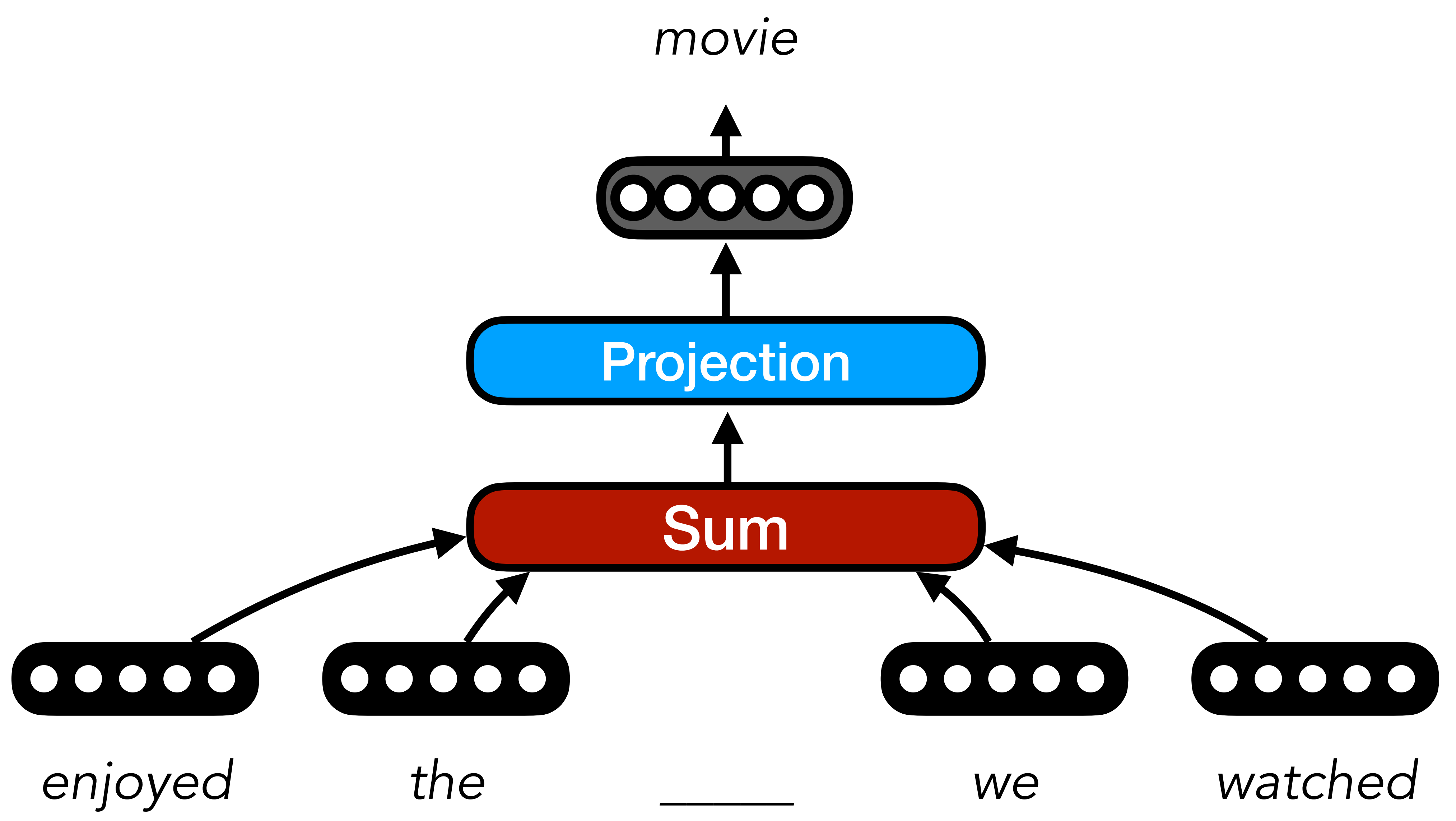
Continuous Bag of Words (CBOW)

- Predict the missing word from a window of surrounding words



Continuous Bag of Words (CBOW)

- Predict the missing word from a window of surrounding words



$$\max P(movie \mid enjoyed, the, we, watched)$$

$$\max P(w_t \mid w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2})$$

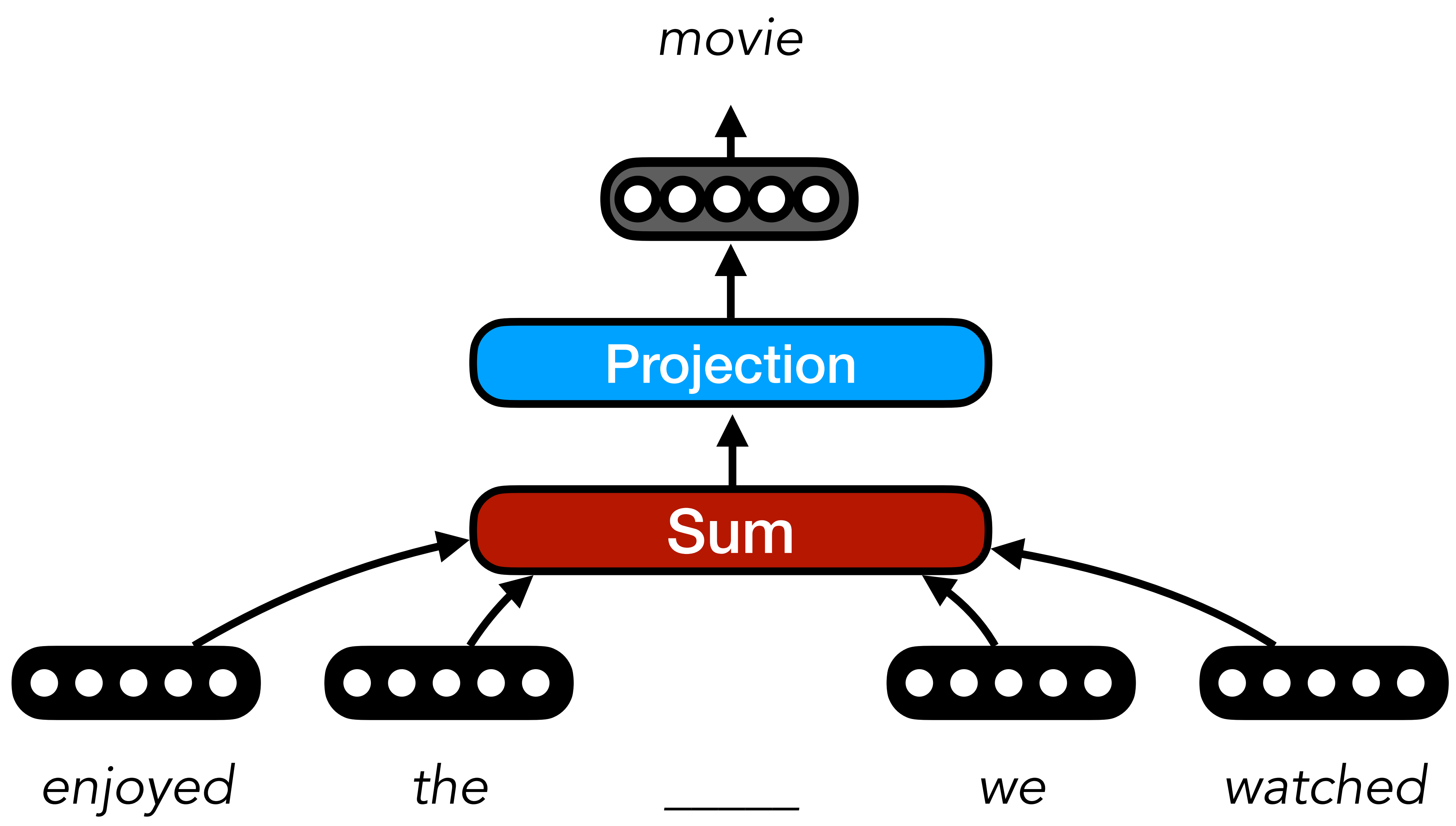
$$\max P(w_t \mid \{w_x\}_{x=t-2}^{x=t+2})$$

$$P(w_t \mid \{w_x\}_{x=t-2}^{x=t+2}) = \textbf{softmax}\left(\mathrm{U} \sum_{\substack{x=t-2 \\ x \neq t}}^{t+2} \mathbf{w}_x\right)$$

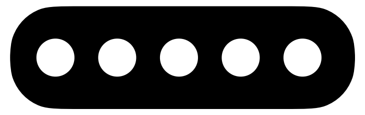
Continuous Bag of Words (CBOW)

- Predict the missing word from a window of surrounding words

$$P(w_t \mid \{w_x\}_{x=t-2}^{x=t+2}) = \textbf{softmax}\left(\mathbf{U} \sum_{\substack{x=t-2 \\ x \neq t}}^{t+2} \mathbf{w}_x\right)$$



$$\mathbf{w}_x \in \mathbb{R}^{1 \times d}$$



$$\mathbf{U} \in \mathbb{R}^{d \times V}$$

$$\textbf{softmax}(\mathbf{a})_i = \frac{e^{a_i}}{\sum_{j=1}^{|\mathbf{a}|} e^{a_j}}$$

Softmax Function

- The **softmax** function generates a probability distribution from the elements of the vector it is given

$$\textbf{softmax}(\mathbf{a})_i = \frac{e^{a_i}}{\sum_{j=1}^{|\mathbf{a}|} e^{a_j}}$$

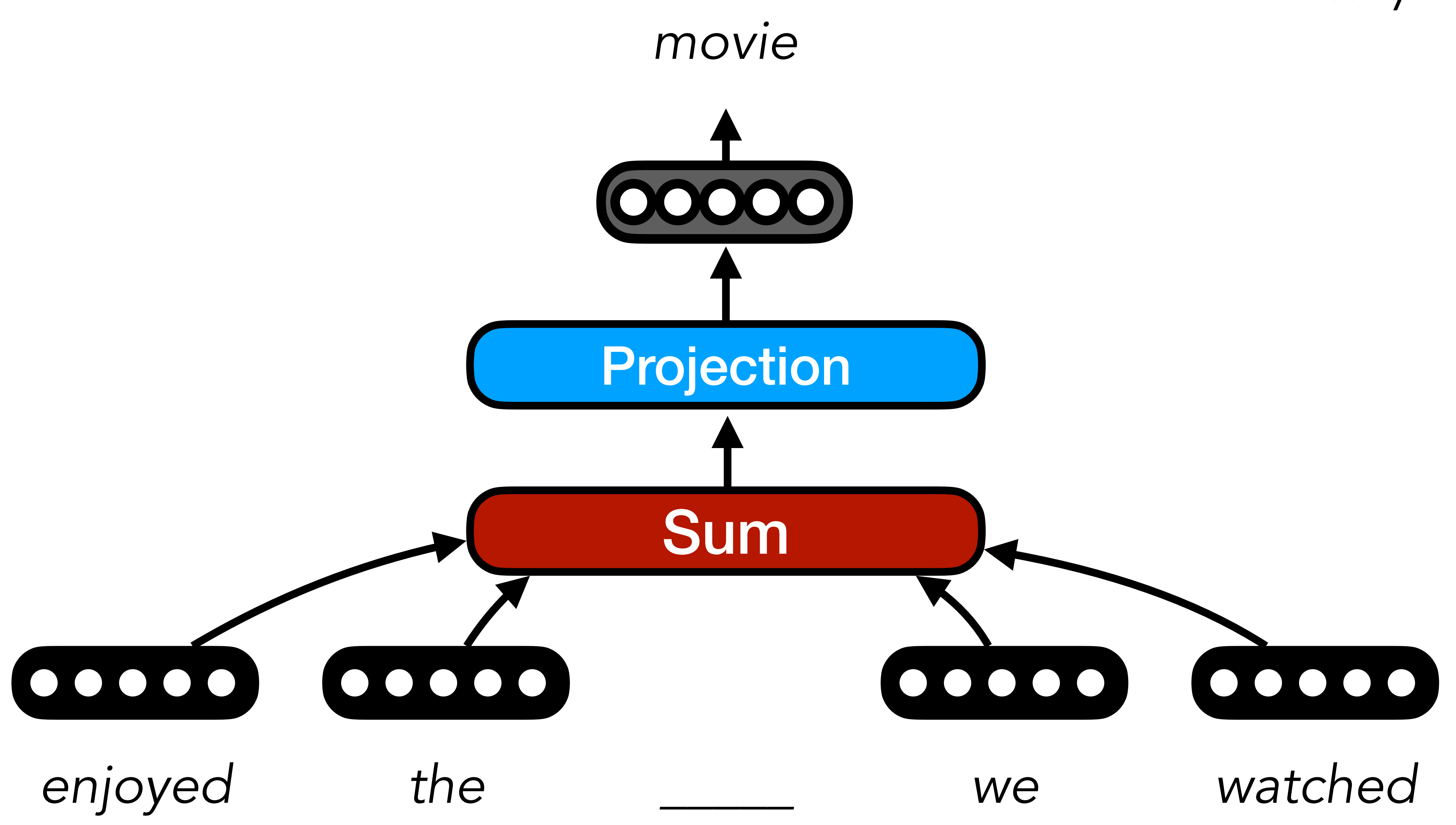
V = [0.790 -0.851 0.506 0.767 -0.788 0.793 0.887 0.219 -0.052 0.461]

$\Big\downarrow$ **Softmax(V)**

P(V) = [0.144 0.028 0.108 0.141 0.030 0.144 0.159 0.081 0.062 0.104]

Continuous Bag of Words (CBOW)

$$P(w_t \mid \{w_x\}_{x=t-2}^{x=t+2}) = \textbf{softmax}\left(\mathbf{U} \sum_{\substack{x = t-2 \\ x \neq t}}^{t+2} \mathbf{w}_x \right)$$

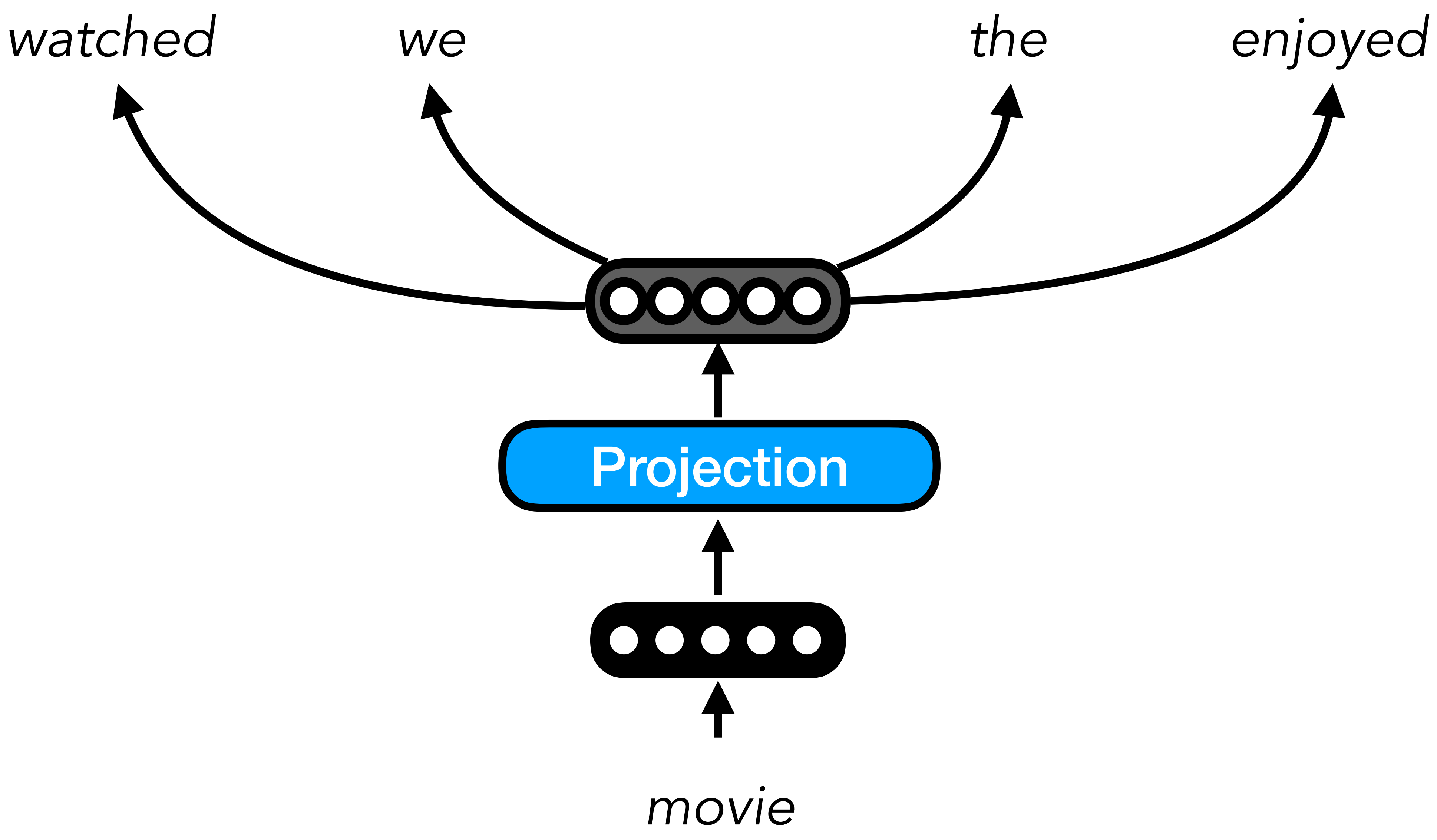


- Model is trained to **maximise** the **probability** of the missing word
  - For computation reasons, the model is typically trained to **minimise** the **negative log probability** of the missing word
- Here, we use a window of **N=2**, but the window size is a **hyperparameter**
- For computational reasons, a **hierarchical softmax** used to compute distribution

Skip-gram

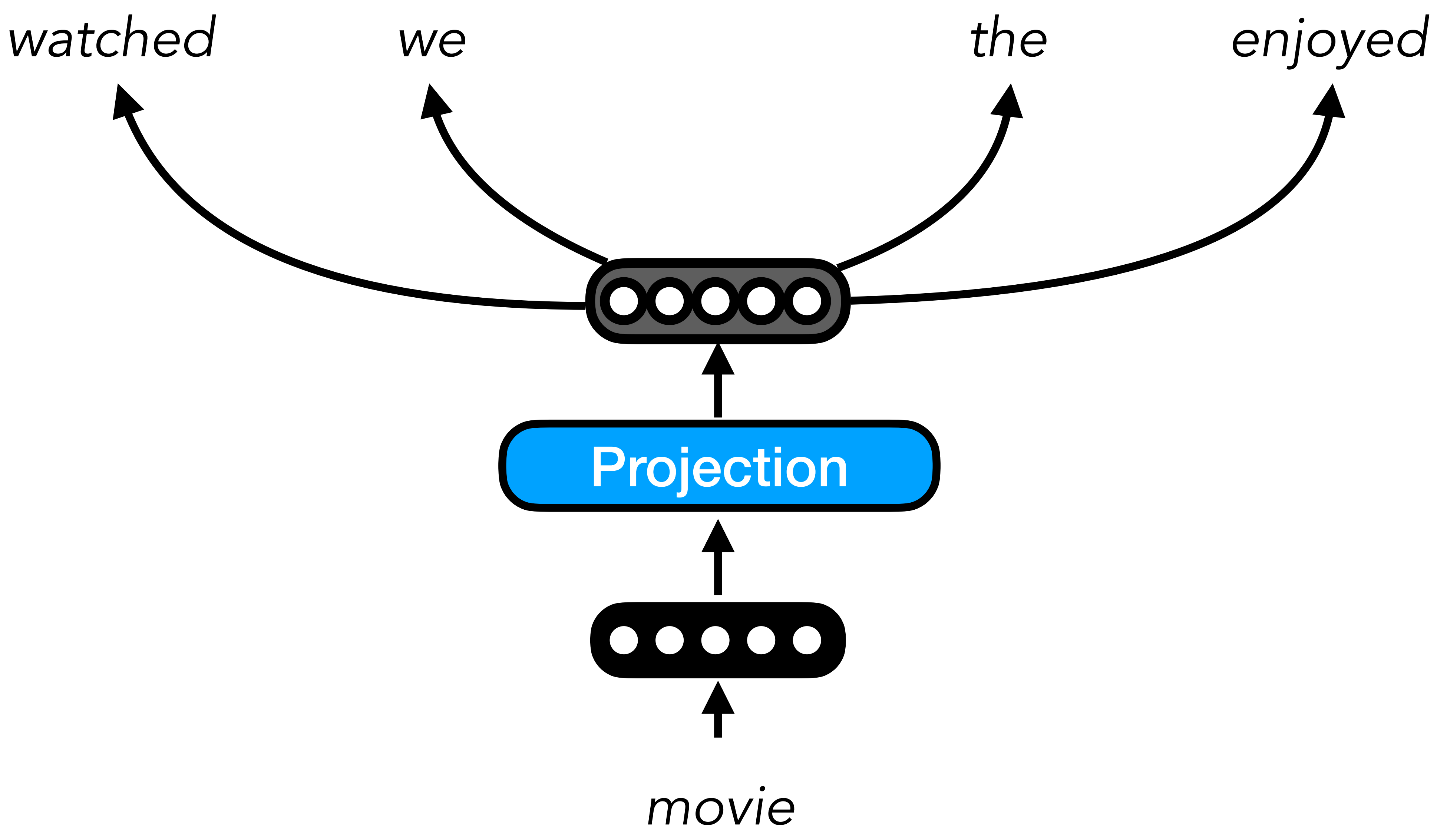
- We can also learn embeddings by predicting the surrounding context from a single word

$$\max P(enjoyed, the, we, watched \mid movie)$$



Skip-gram

- We can also learn embeddings by predicting the surrounding context from a single word

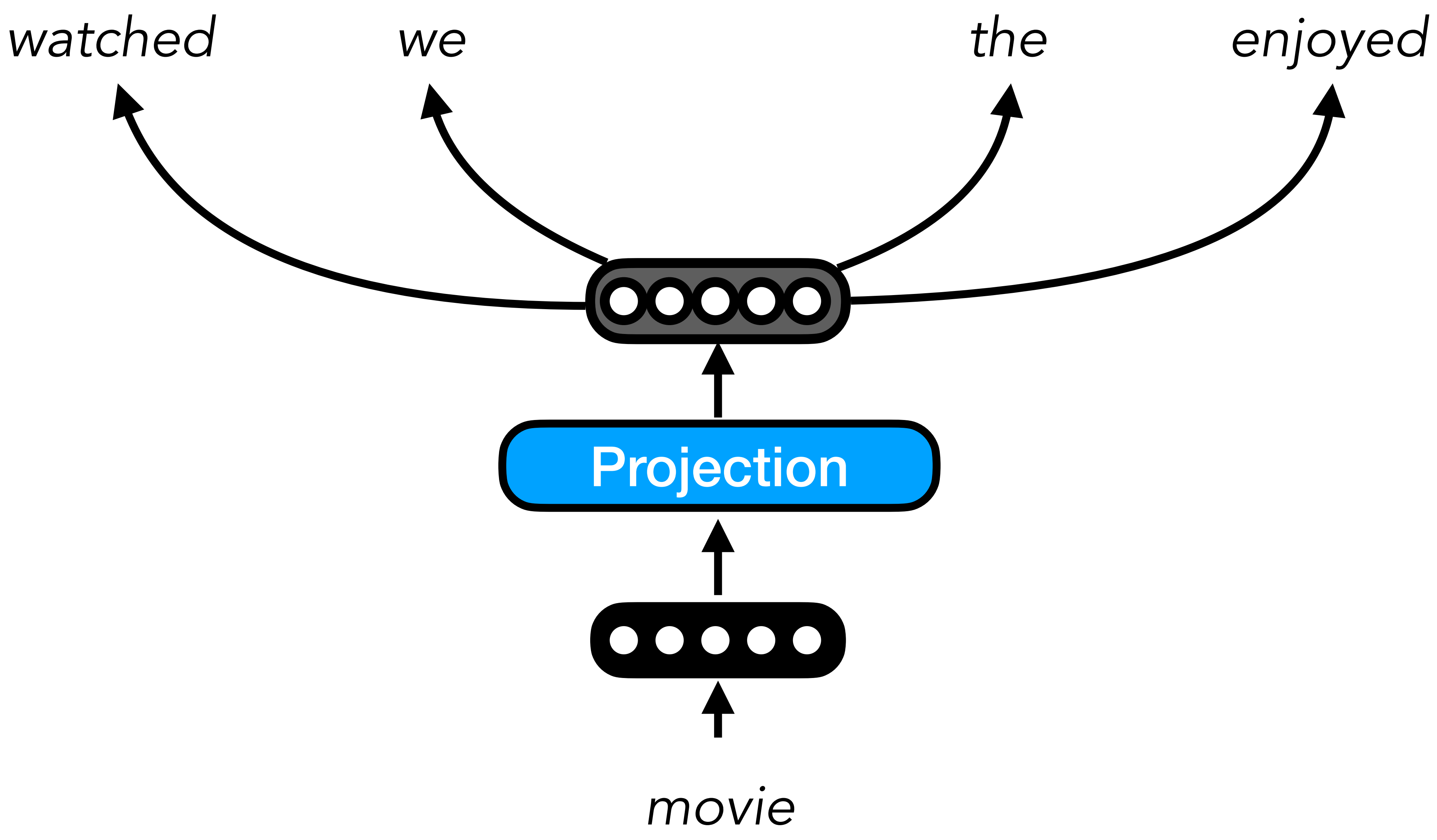


$$\max P(enjoyed, the, we, watched \mid movie)$$

$$\max P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} \mid w_t)$$

Skip-gram

- We can also learn embeddings by predicting the surrounding context from a single word



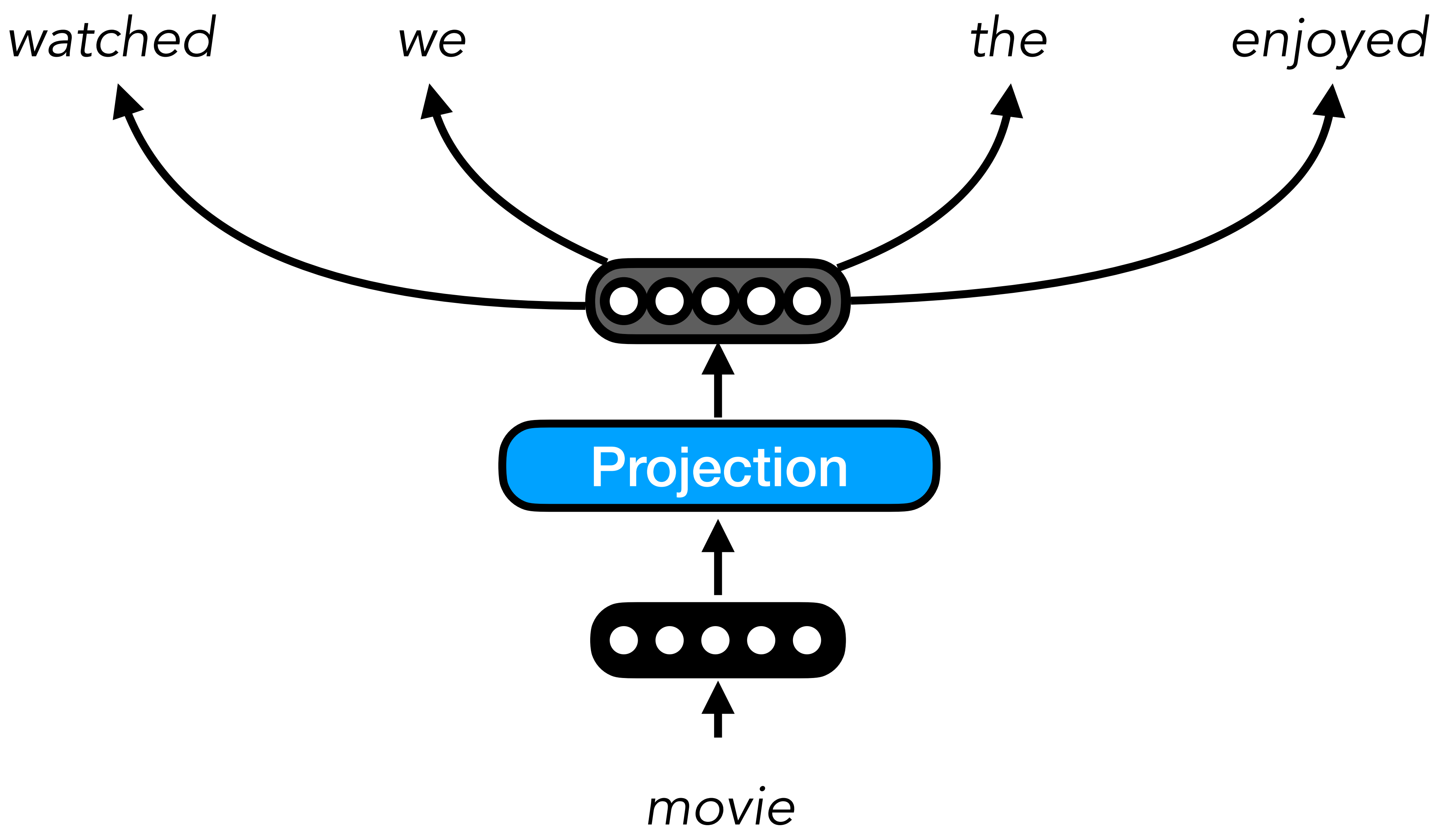
$$\max P(\textit{enjoyed}, \textit{the}, \textit{we}, \textit{watched} \mid \textit{movie})$$

$$\max P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} \mid w_t)$$

$$\max \log P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} \mid w_t)$$

Skip-gram

- We can also learn embeddings by predicting the surrounding context from a single word



$$\max P(enjoyed, the, we, watched \mid movie)$$

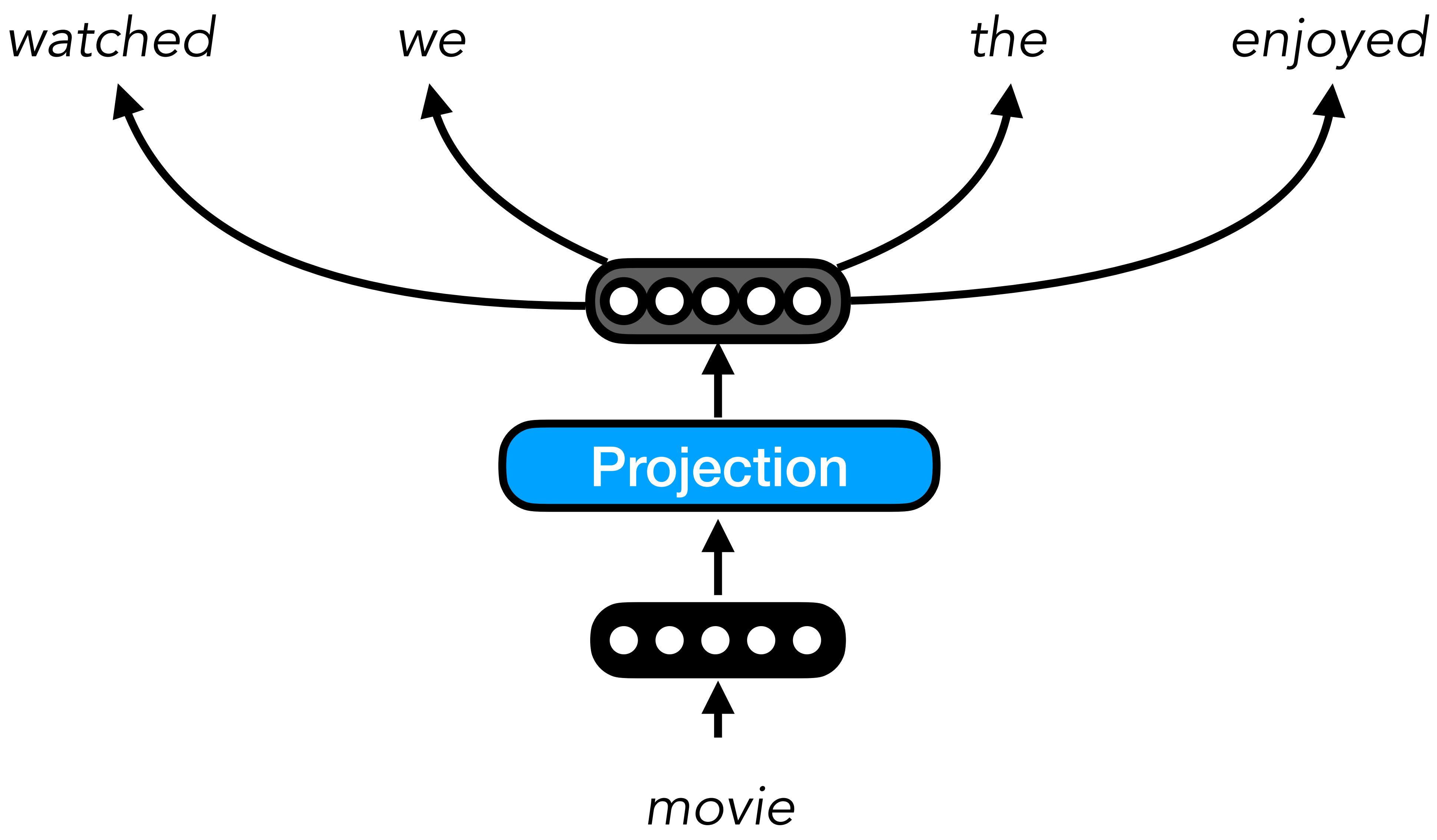
$$\max P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} \mid w_t)$$

$$\max \log P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} \mid w_t)$$

$$\max \Big(\log P(w_{t-2} \mid w_t) + \log P(w_{t-1} \mid w_t) + \log P(w_{t+1} \mid w_t) + \log P(w_{t+2} \mid w_t) \Big)$$

Skip-gram

- We can also learn embeddings by predicting the surrounding context from a single word



$$\max P(\textit{enjoyed}, \textit{the}, \textit{we}, \textit{watched} \mid \textit{movie})$$

$$\max P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} \mid w_t)$$

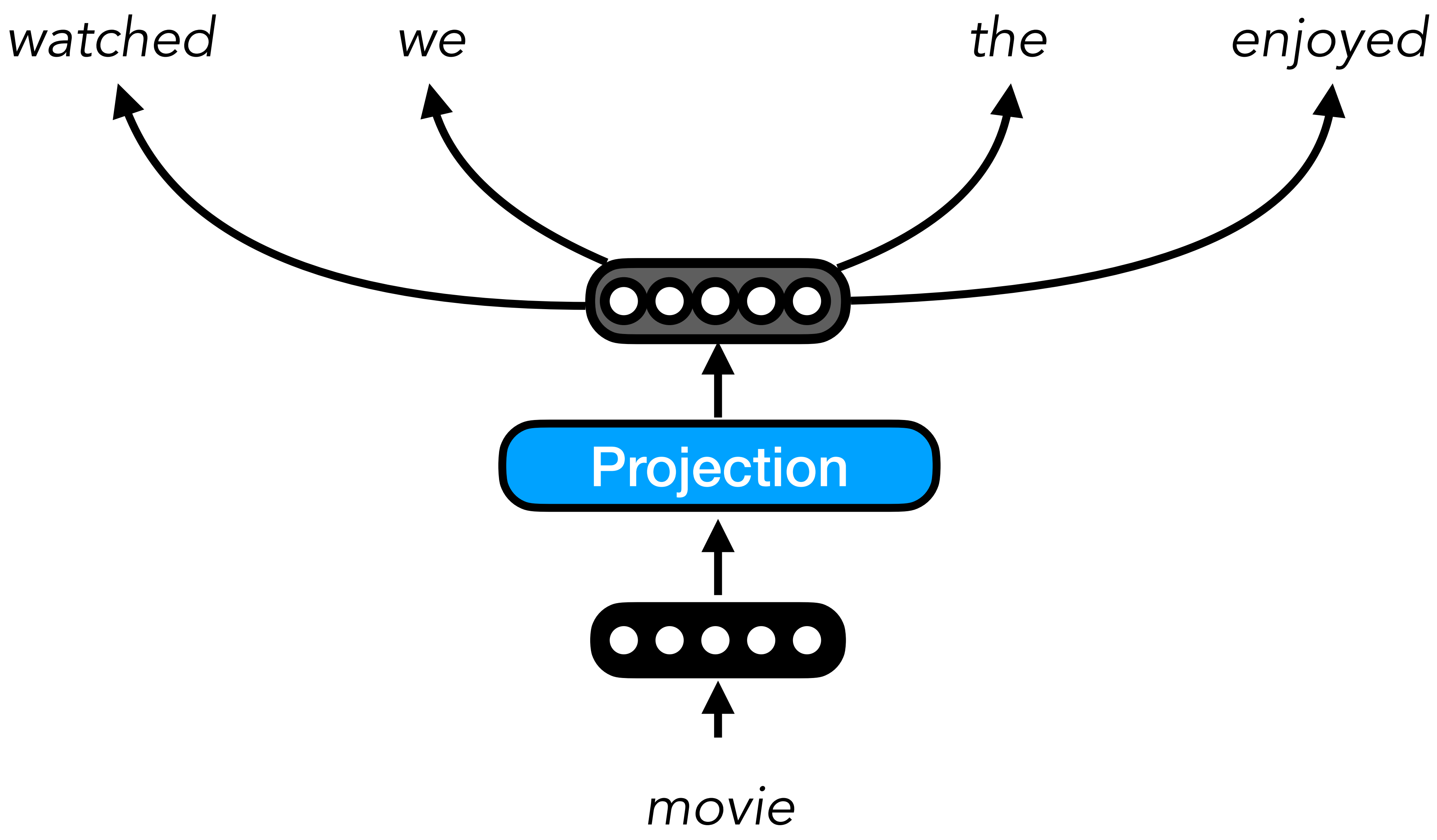
$$\max \log P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} \mid w_t)$$

$$\max \Big(\log P(w_{t-2} \mid w_t) + \log P(w_{t-1} \mid w_t) + \log P(w_{t+1} \mid w_t) + \log P(w_{t+2} \mid w_t) \Big)$$

$$P(w_x \mid w_t) = \textbf{softmax}\left(\mathbf{U}\mathbf{w}_t\right)$$

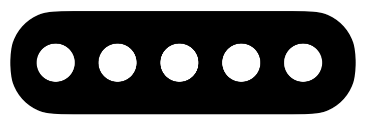
Skip-gram

- We can also learn embeddings by predicting the surrounding context from a single word



$$P(w_x \mid w_t) = \mathbf{softmax}(\mathbf{U}\mathbf{w}_t)$$

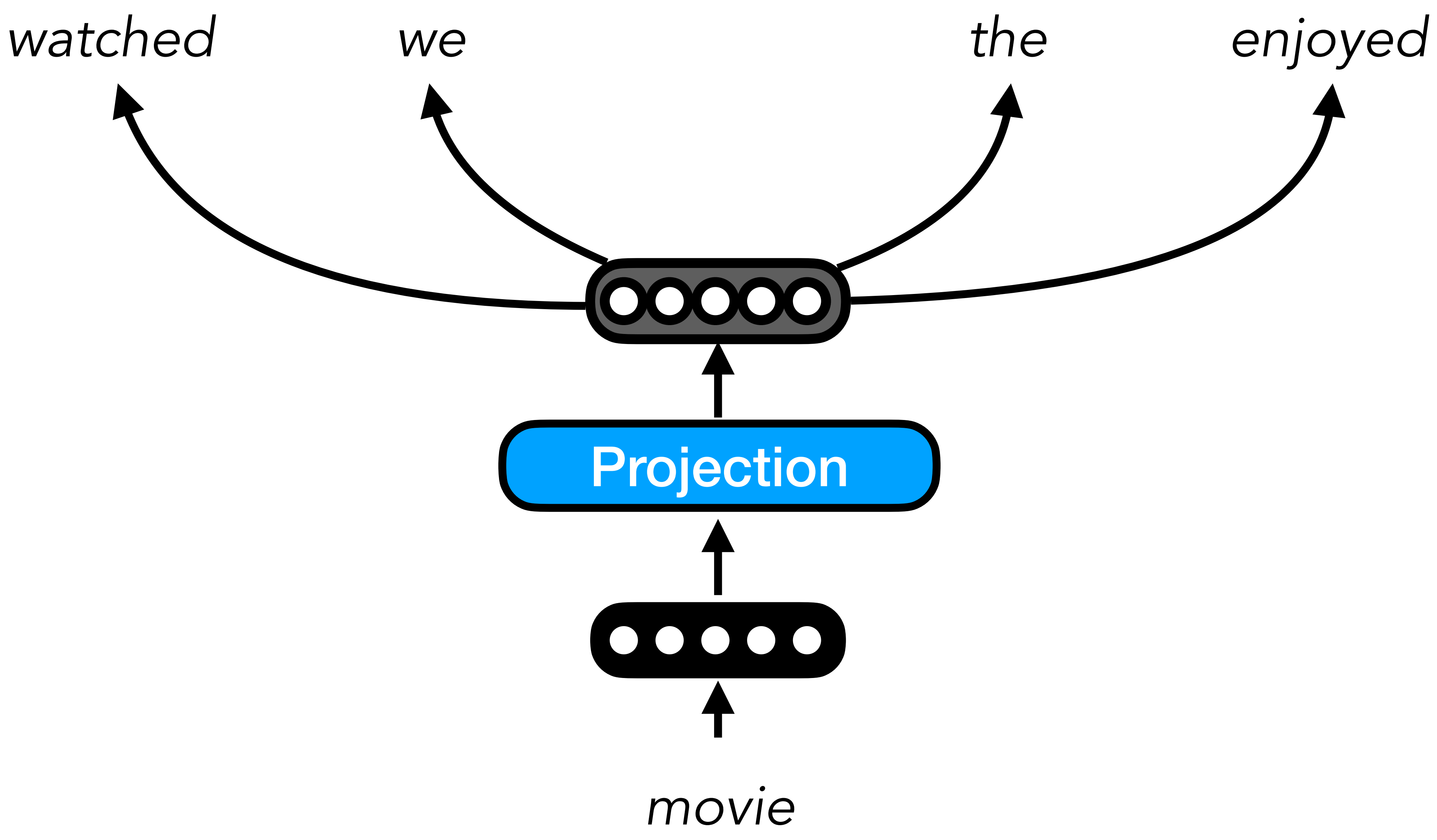
$\mathbf{w}_t \in \mathbb{R}^{1 \times d}$



$\mathbf{U} \in \mathbb{R}^{d \times V}$

Skip-gram

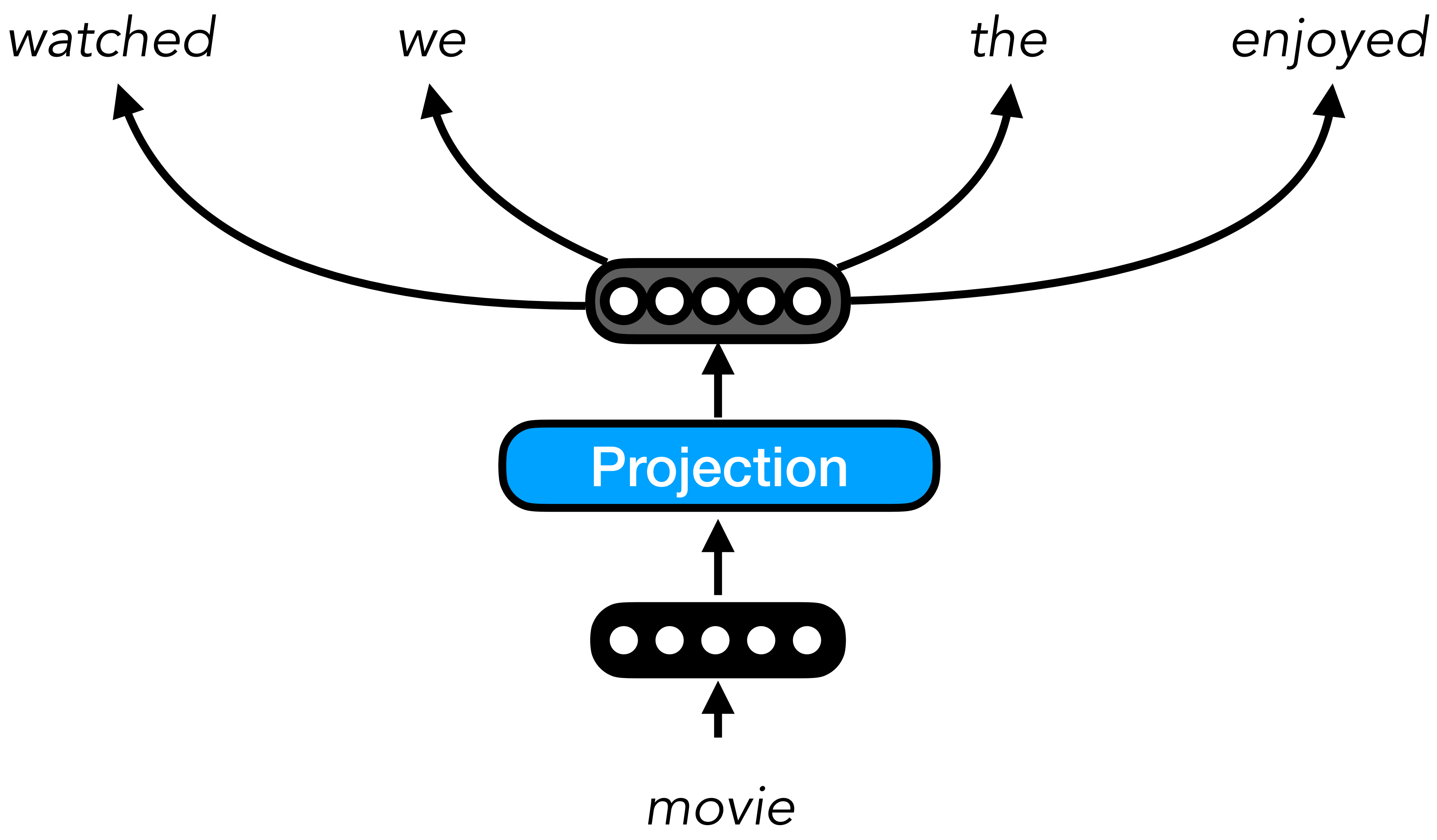
- We can also learn embeddings by predicting the surrounding context from a single word

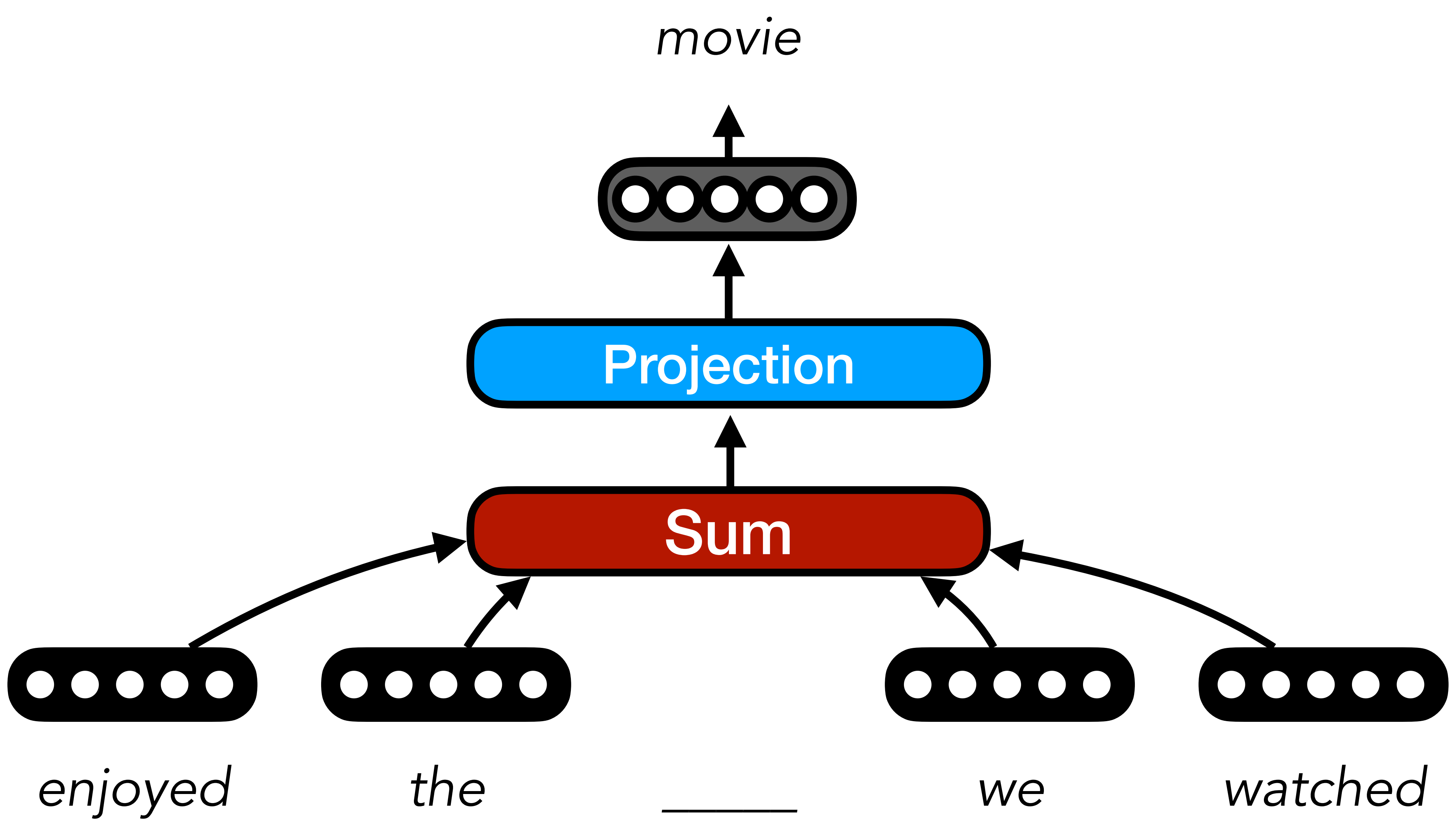


- Model is trained to **minimise** the **negative log probability** of the surrounding words
- Here, we use a window of **N=2**, but the window size is a **hyperparameter** to set
- Typically, set large window **(N=10),** but randomly select $i \in [1, N]$ as dynamic window size so that closer words contribute more to learning

Skip-gram vs. CBOW

- **Question:** Do you expect a difference between what is learned by CBOW and Skipgram methods?





Demo

https://colab.research.google.com/drive/1aCWxocr8plpRtRj02ODmJyjKxf8g563h?usp=sharing

Other Resources of Interest

- **GloVe** Vectors (Pennington et al., 2014):
  - Use the co-occurrence matrix between words to compute word vectors
  - https://nlp.stanford.edu/projects/glove/
- Retrofitting word vectors to semantic lexicons (Faruqui et al., 2014)
  - Training word vectors to encode semantic relationships from high-level resources: WordNet, PPDB, and FrameNet

Part 2: Recurrent Neural Networks for Sequence Modeling

Section Outline

- **Background**: Language Modeling, Feedforward Neural Networks, Backpropagation
- **Content - Models:** Recurrent Neural Networks, LSTMs, Encoder-Decoders
- **Content - Algorithms:** Backpropagation through Time, Vanishing Gradients

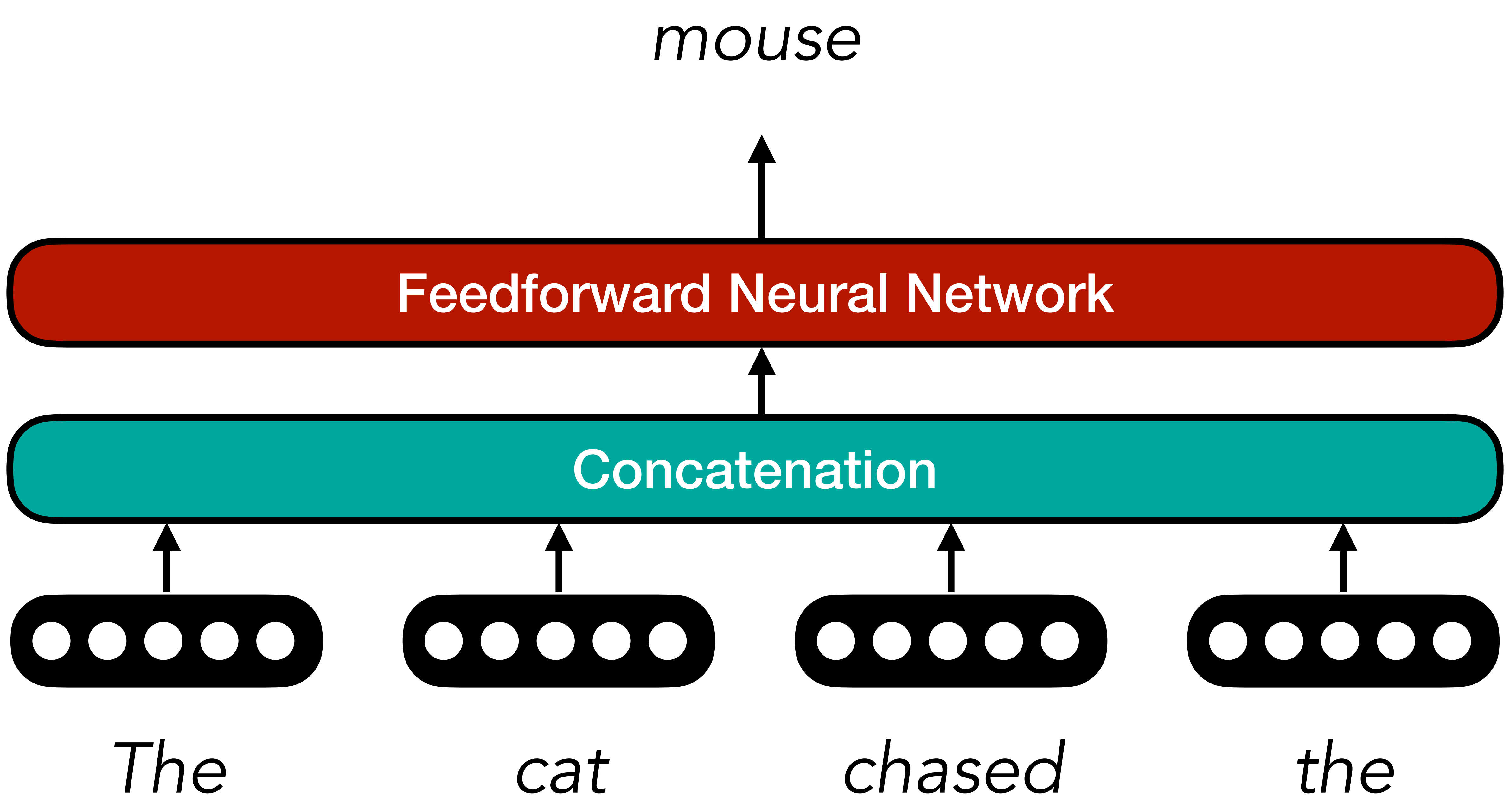
Language Modeling

- Given a subsequence, predict the next word: *The cat chased the* _____

Fixed Context Language Models

- Given a subsequence, predict the next word: *The cat chased the _____*

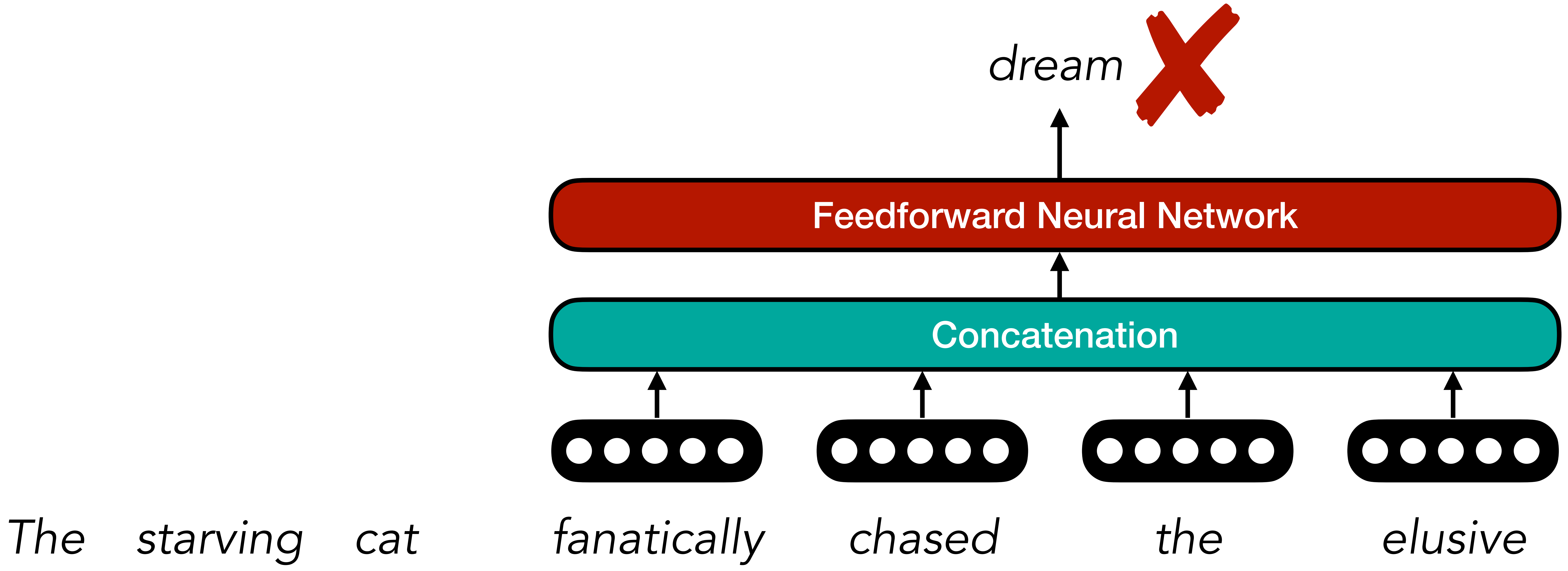
$$P(y) = \textbf{softmax}\left(b_o + \mathbf{W}_o\ \textbf{tanh}(b_h + \mathbf{W}_h x)\right)$$



Fixed Context Language Models

- Given a subsequence, predict the next word:

The starving cat fanatically chased the elusive _____



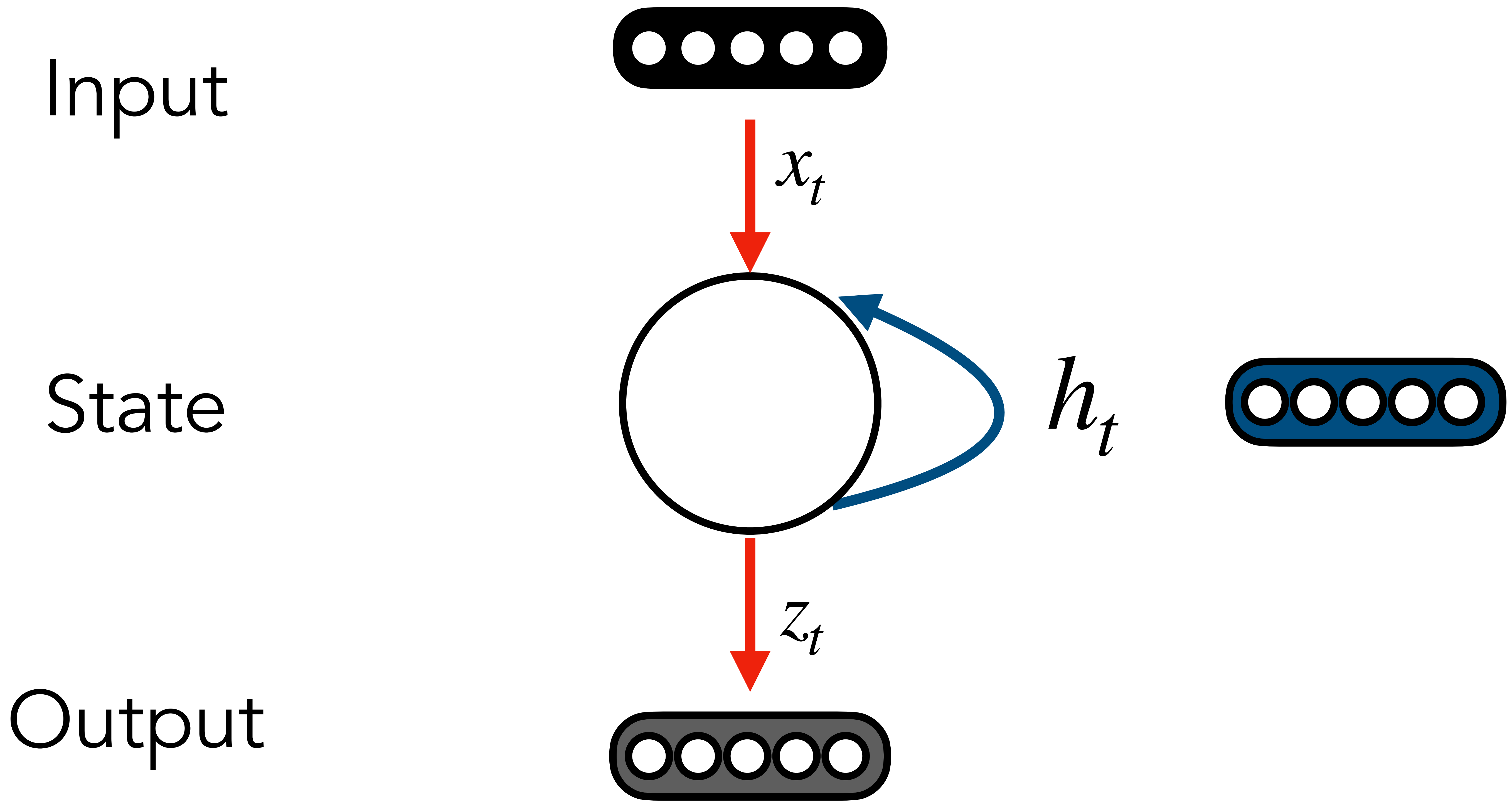
Problem

Fixed context windows limit language modelling capacity

How can we extend to arbitrary length sequences?

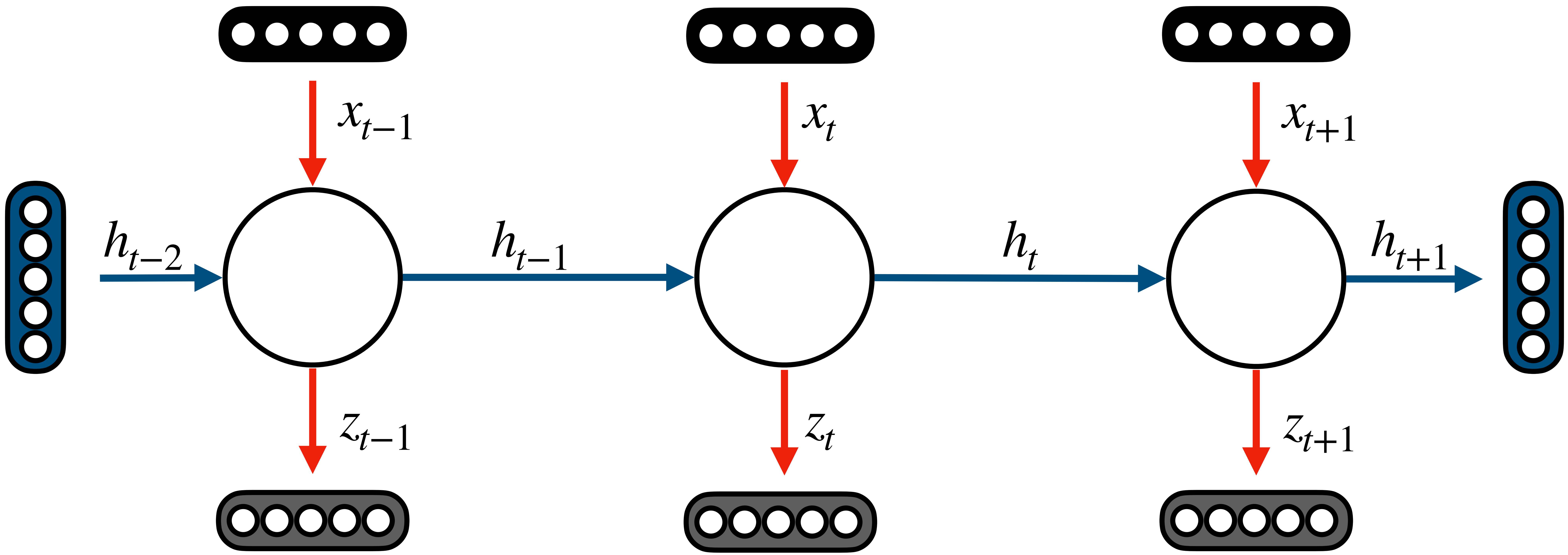
Recurrent Neural Networks

- **Solution:** Recurrent neural networks — NNs with feedback loops



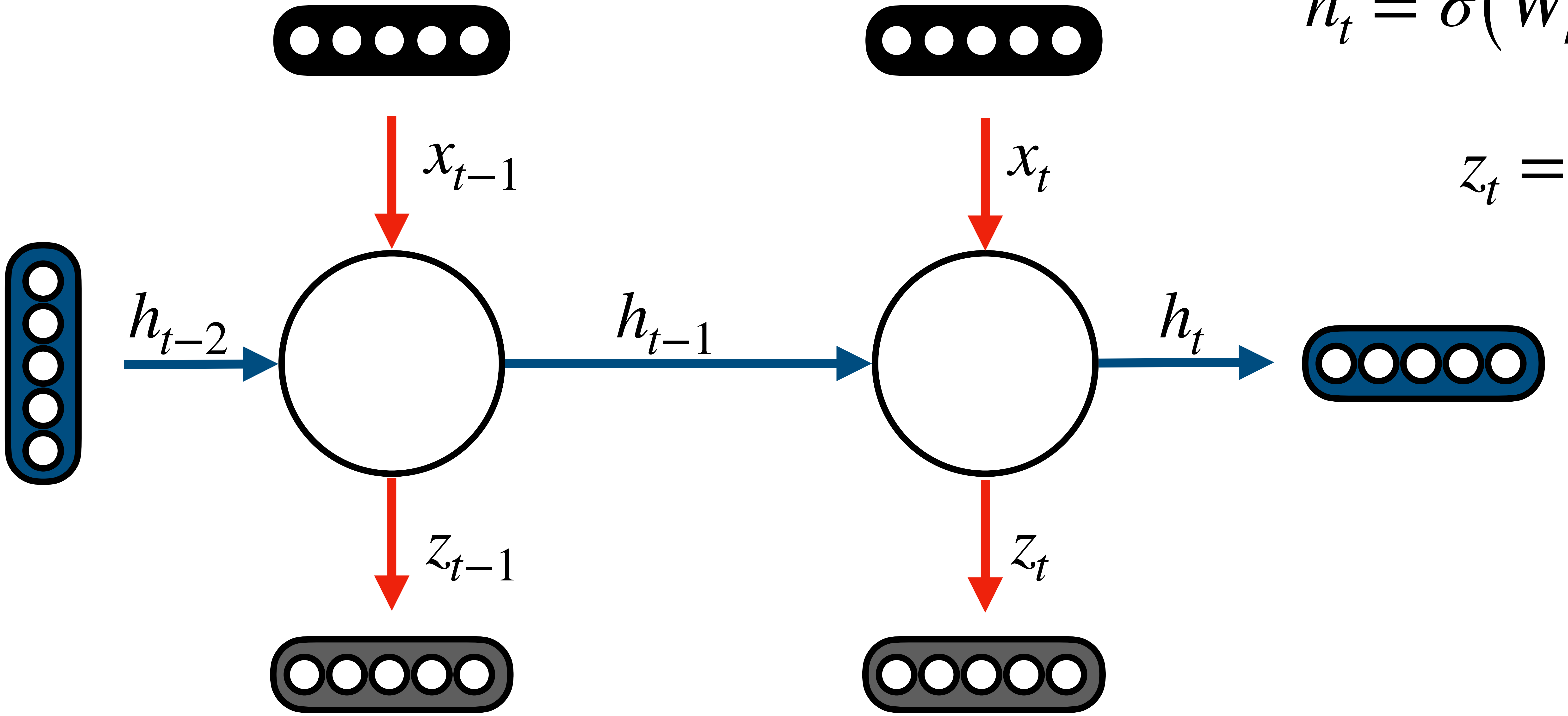
Unrolling the RNN

Unrolling the RNN across all time steps gives full computation graph



Allows for learning from entire sequence history, regardless of length

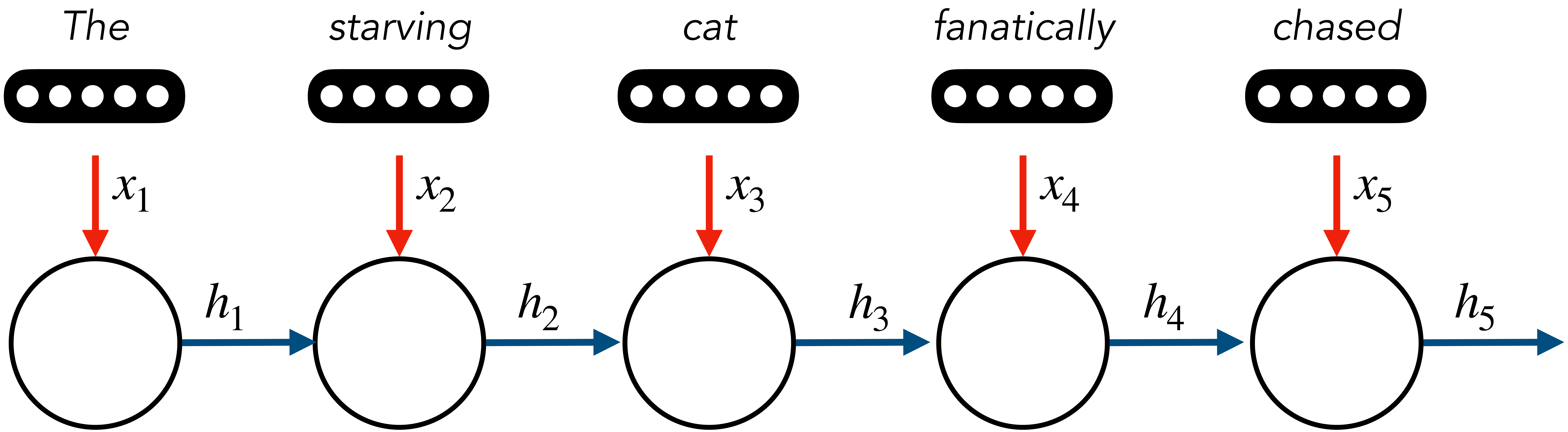
Classical RNN: Elman Network



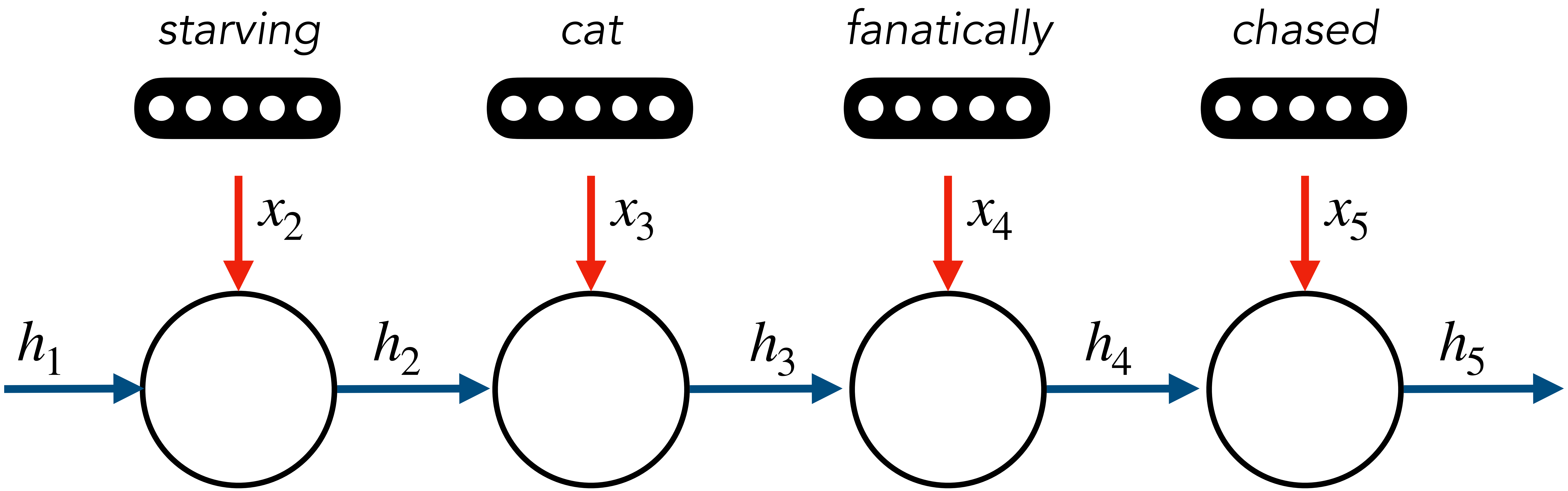
$$h_t = \sigma\left(W_{hx}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$z_t = \sigma\left(W_{zh}h_t + b_z\right)$$

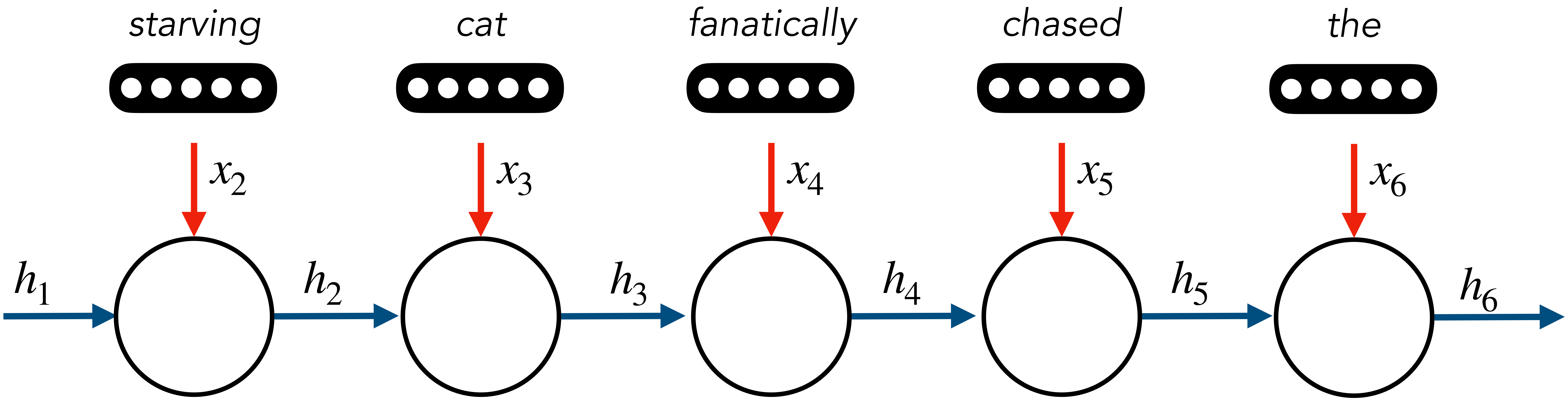
Classical RNN: Elman Network



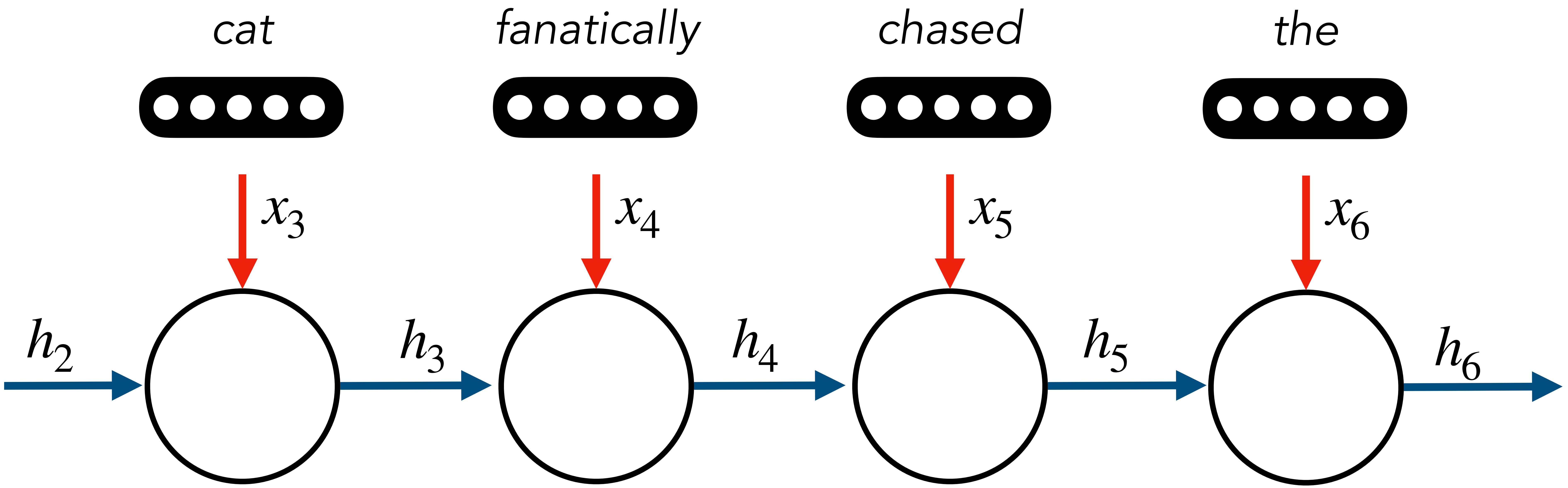
Classical RNN: Elman Network



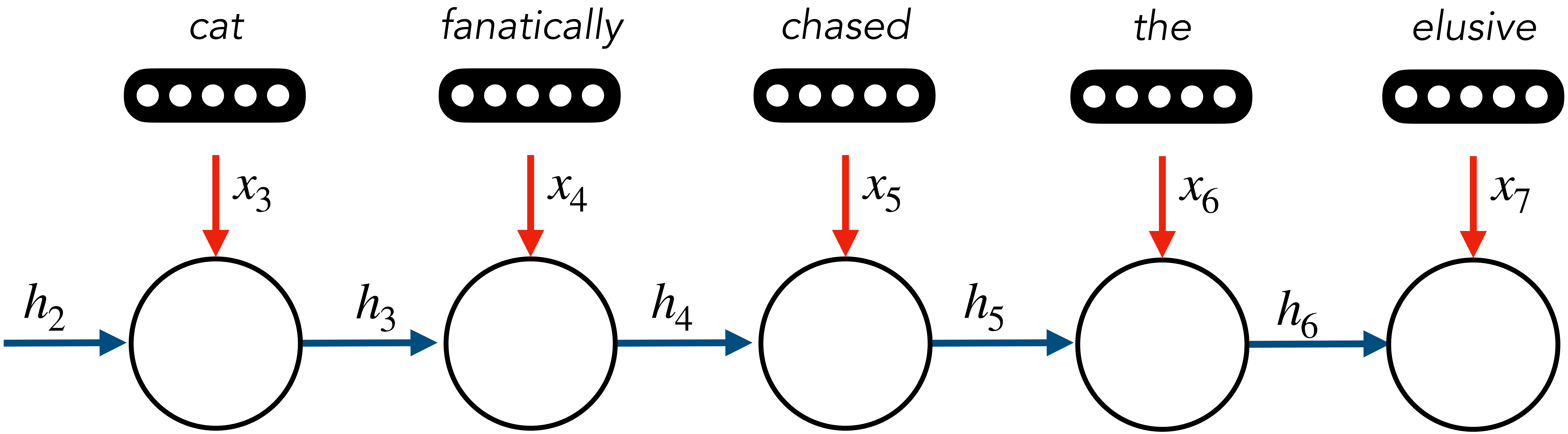
Classical RNN: Elman Network



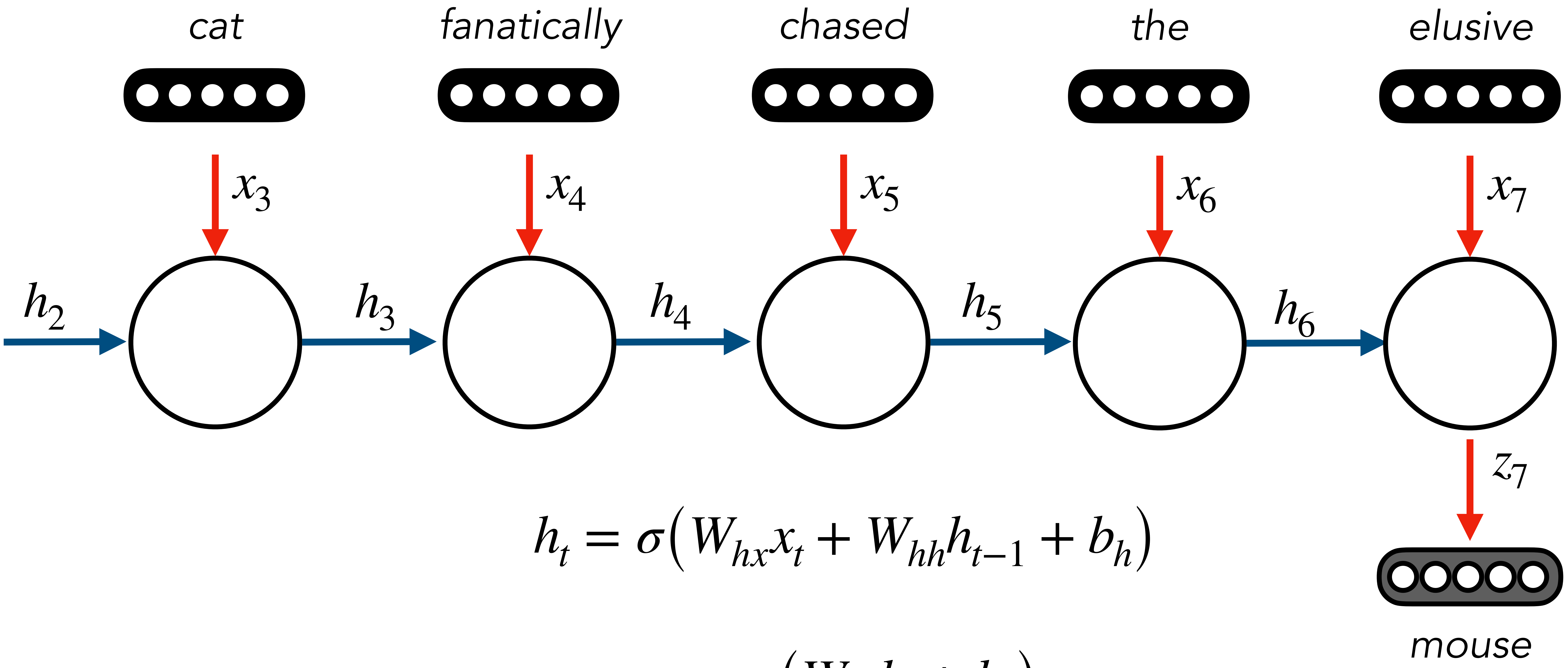
Classical RNN: Elman Network



Classical RNN: Elman Network



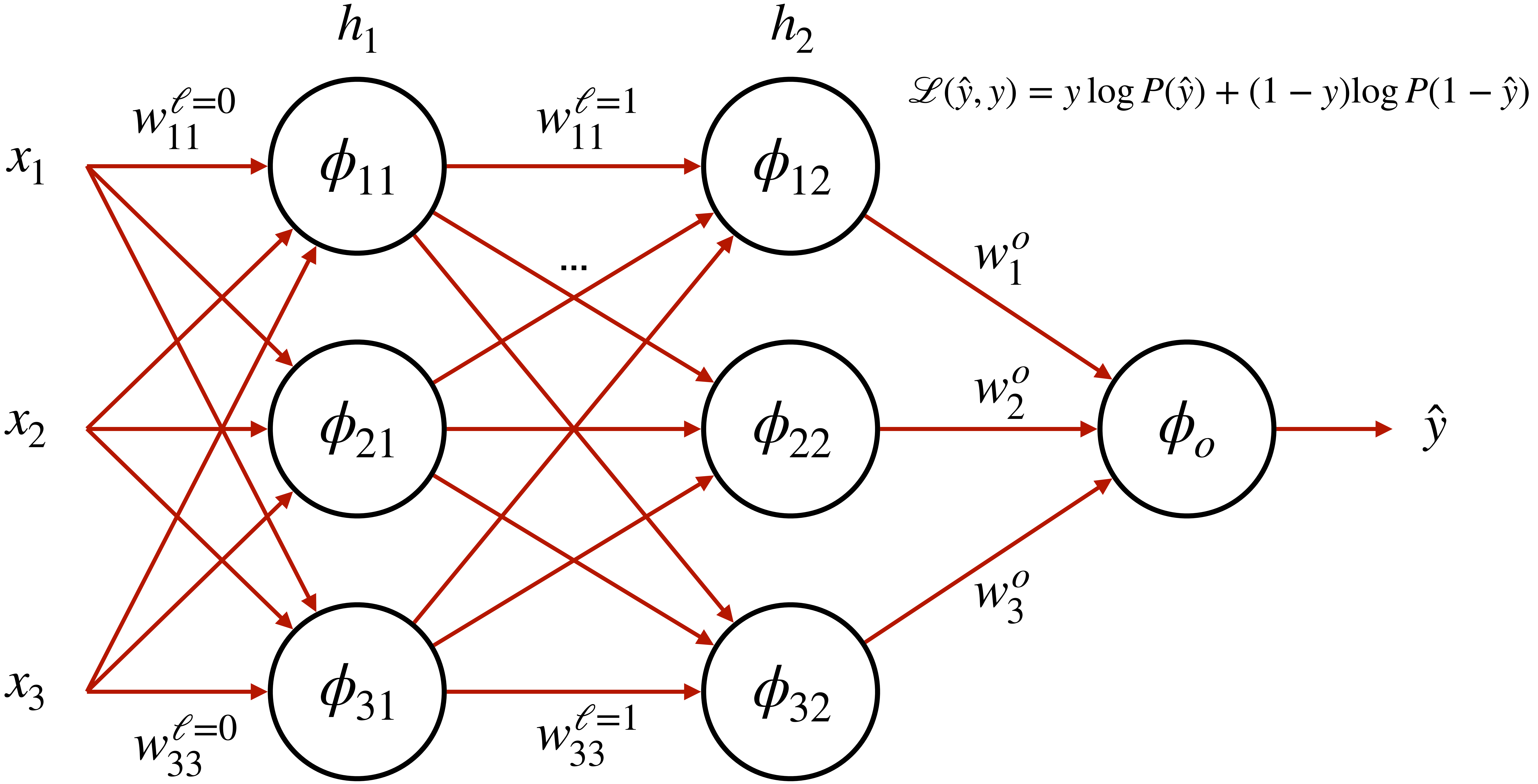
Classical RNN: Elman Network



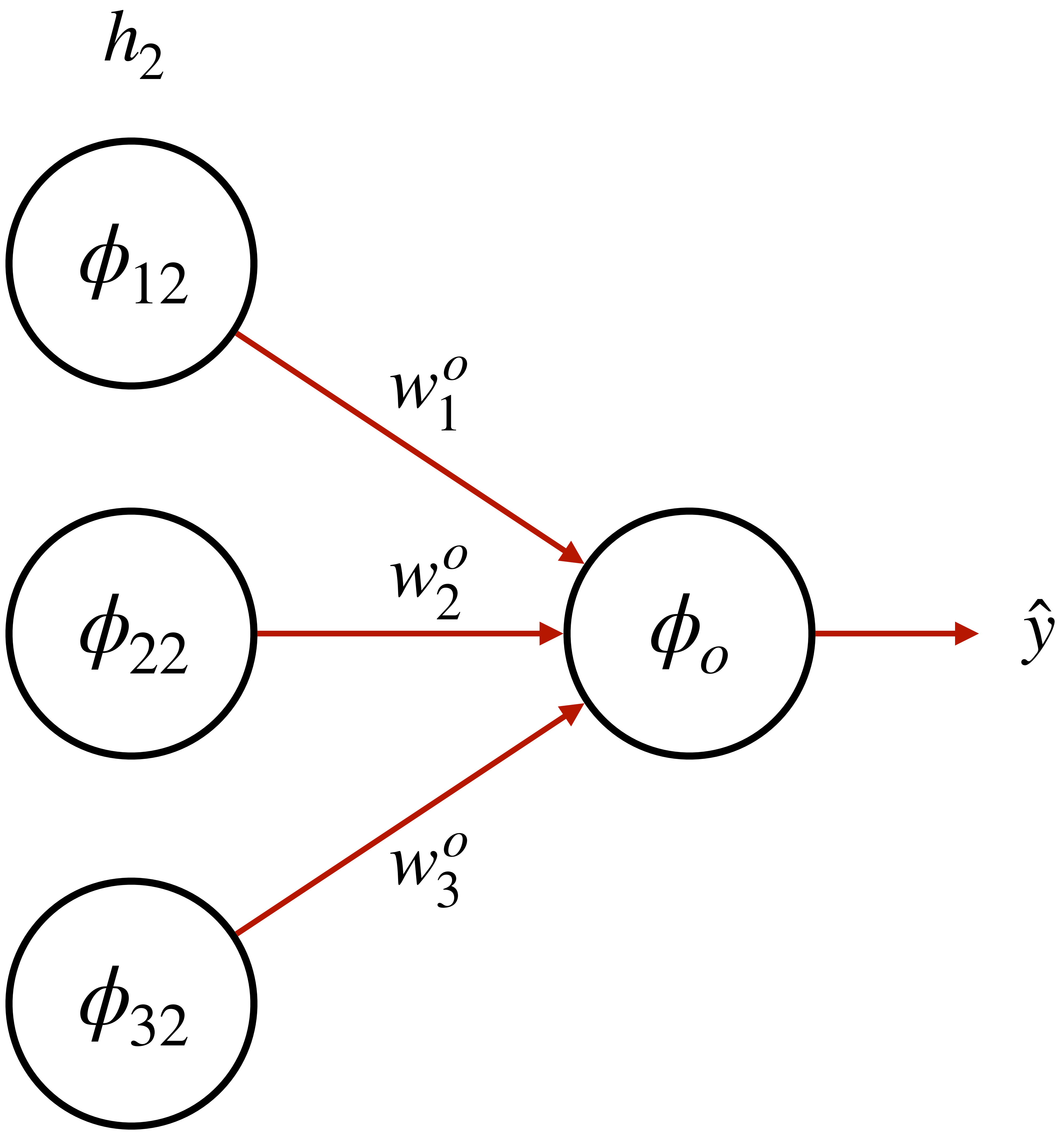
$$h_t = \sigma\left(W_{hx}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$z_t = \sigma\left(W_{zh}h_t + b_z\right)$$

Backpropagation Review: FFNs

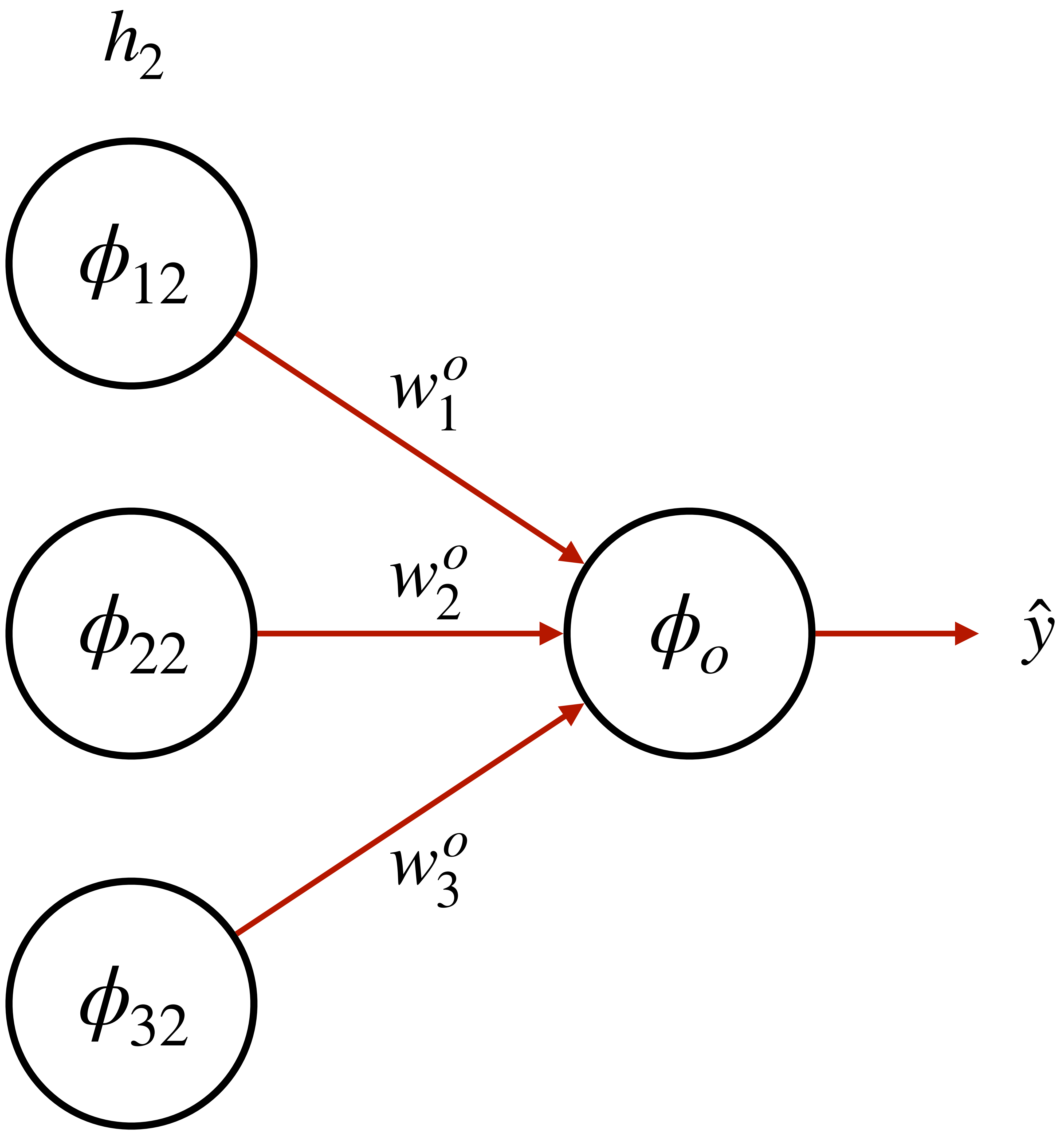


Backpropagation Review: FFNs



$$\mathscr{L}(\hat{y}, y) = y \log P(\hat{y}) + (1 - y)\log P(1 - \hat{y})$$

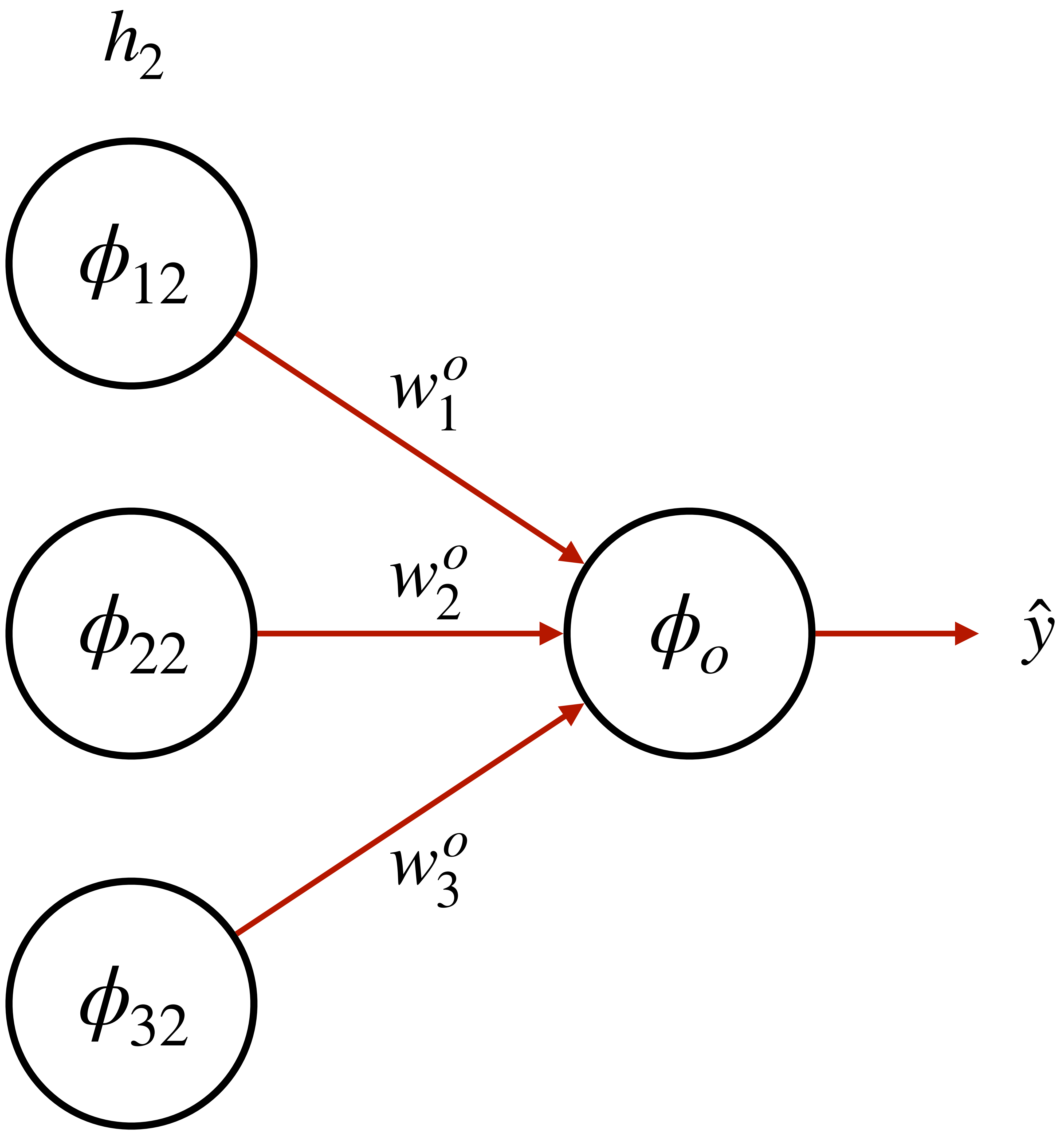
Backpropagation Review: FFNs



$$\mathscr{L}(\hat{y}, y) = y \log P(\hat{y}) + (1 - y) \log P(1 - \hat{y})$$

$$\hat{y} = \phi_o(u)$$

Backpropagation Review: FFNs

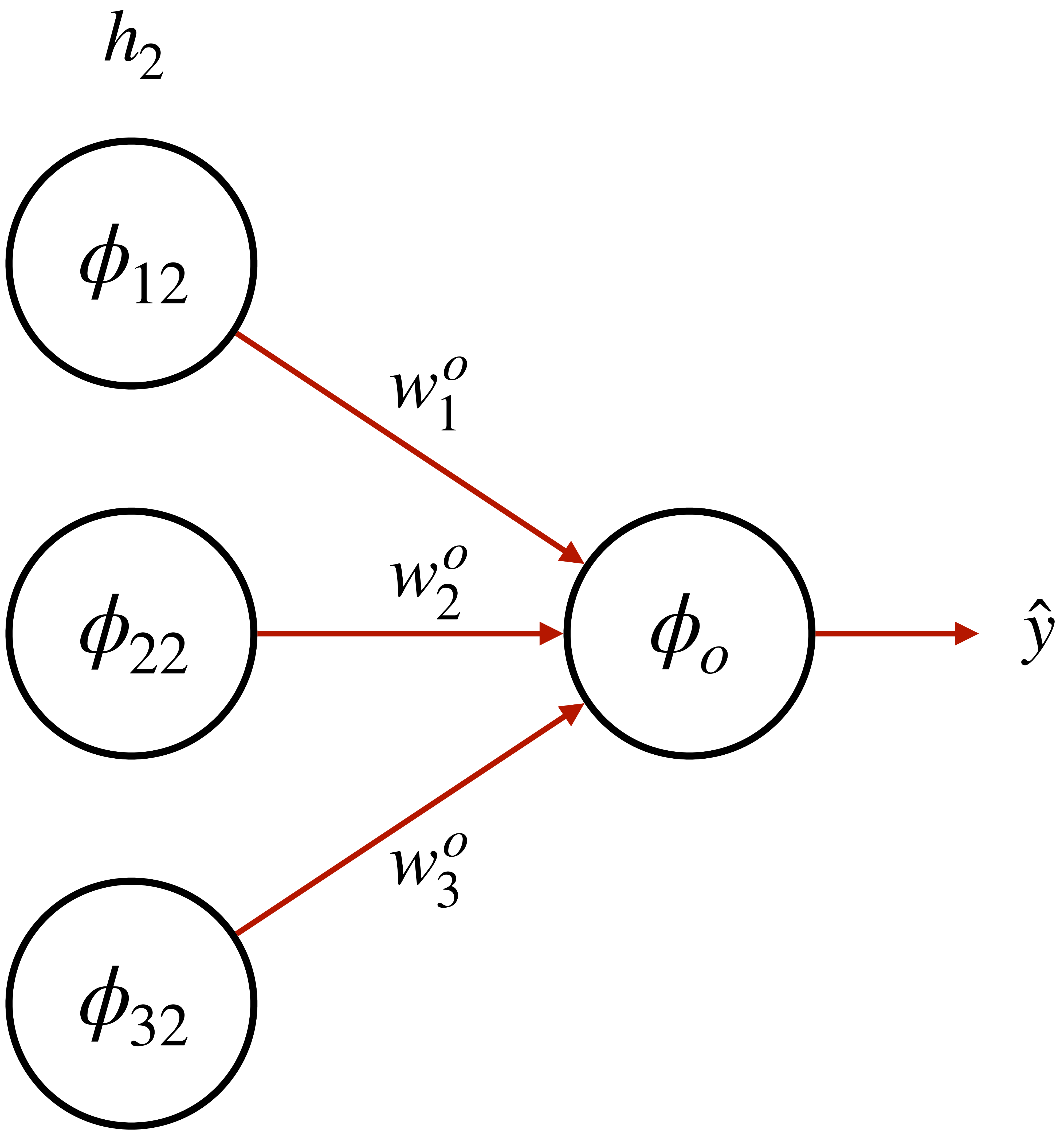


$$\mathscr{L}(\hat{y}, y) = y \log P(\hat{y}) + (1 - y) \log P(1 - \hat{y})$$

$$\hat{y} = \phi_o(u)$$

$$u = w_1^o \times \phi_{12}(\,.\,) + w_2^o \times \phi_{22}(\,.\,) + w_3^o \times \phi_{32}(\,.\,)$$

Backpropagation Review: FFNs



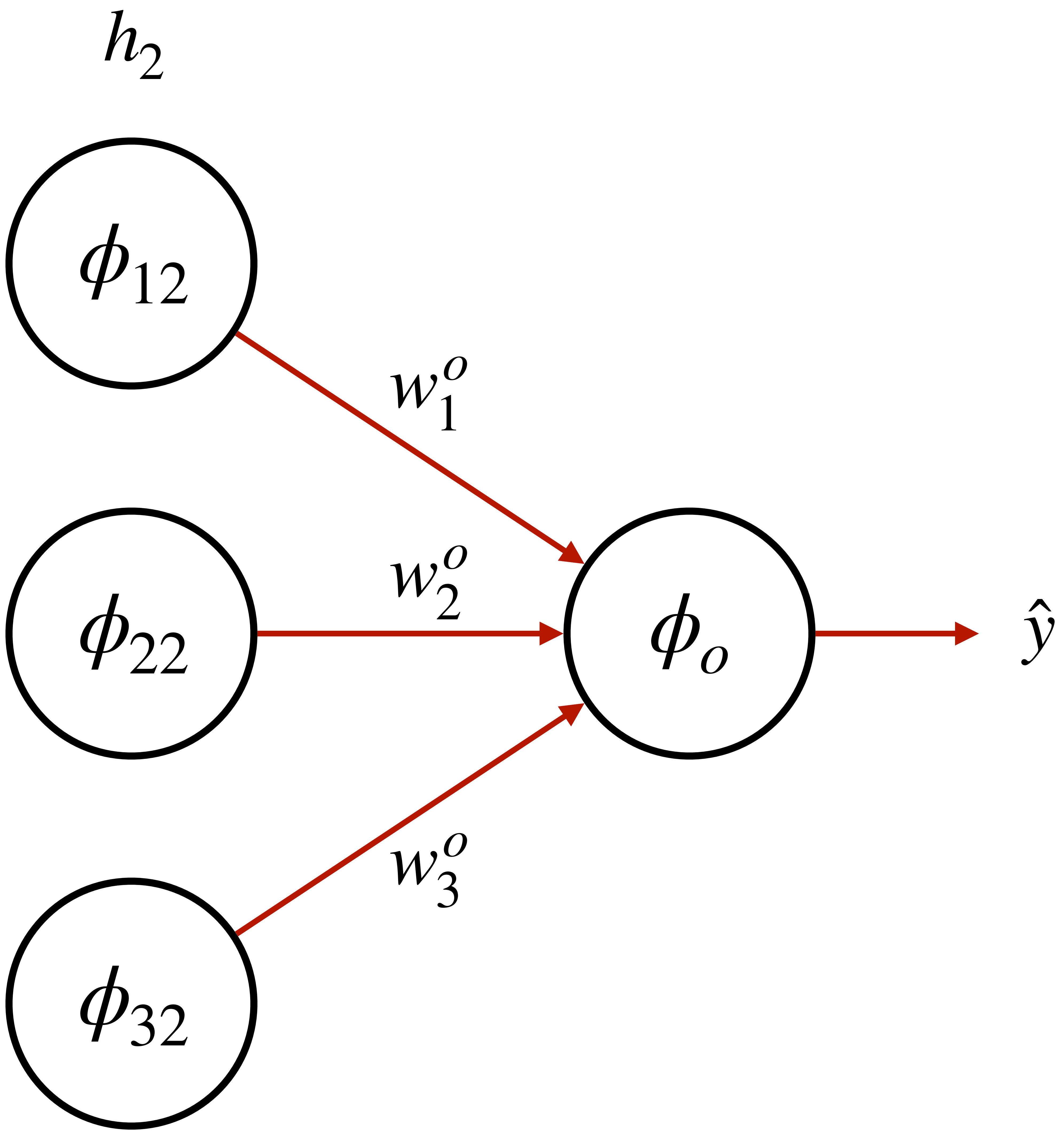
$$\mathscr{L}(\hat{y}, y) = y \log P(\hat{y}) + (1 - y) \log P(1 - \hat{y})$$

$$\hat{y} = \phi_o(u)$$

$$u = w_1^o \times \phi_{12}(\,.\,) + w_2^o \times \phi_{22}(\,.\,) + w_3^o \times \phi_{32}(\,.\,)$$

$$\frac{\partial \mathscr{L}(\hat{y}, y)}{\partial \phi_{12}(\,.\,)} = \frac{\partial \mathscr{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} \frac{\partial u}{\partial \phi_{12}(\,.\,)}$$

Backpropagation Review: FFNs



$$\mathcal{L}(\hat{y}, y) = y \log P(\hat{y}) + (1 - y) \log P(1 - \hat{y})$$

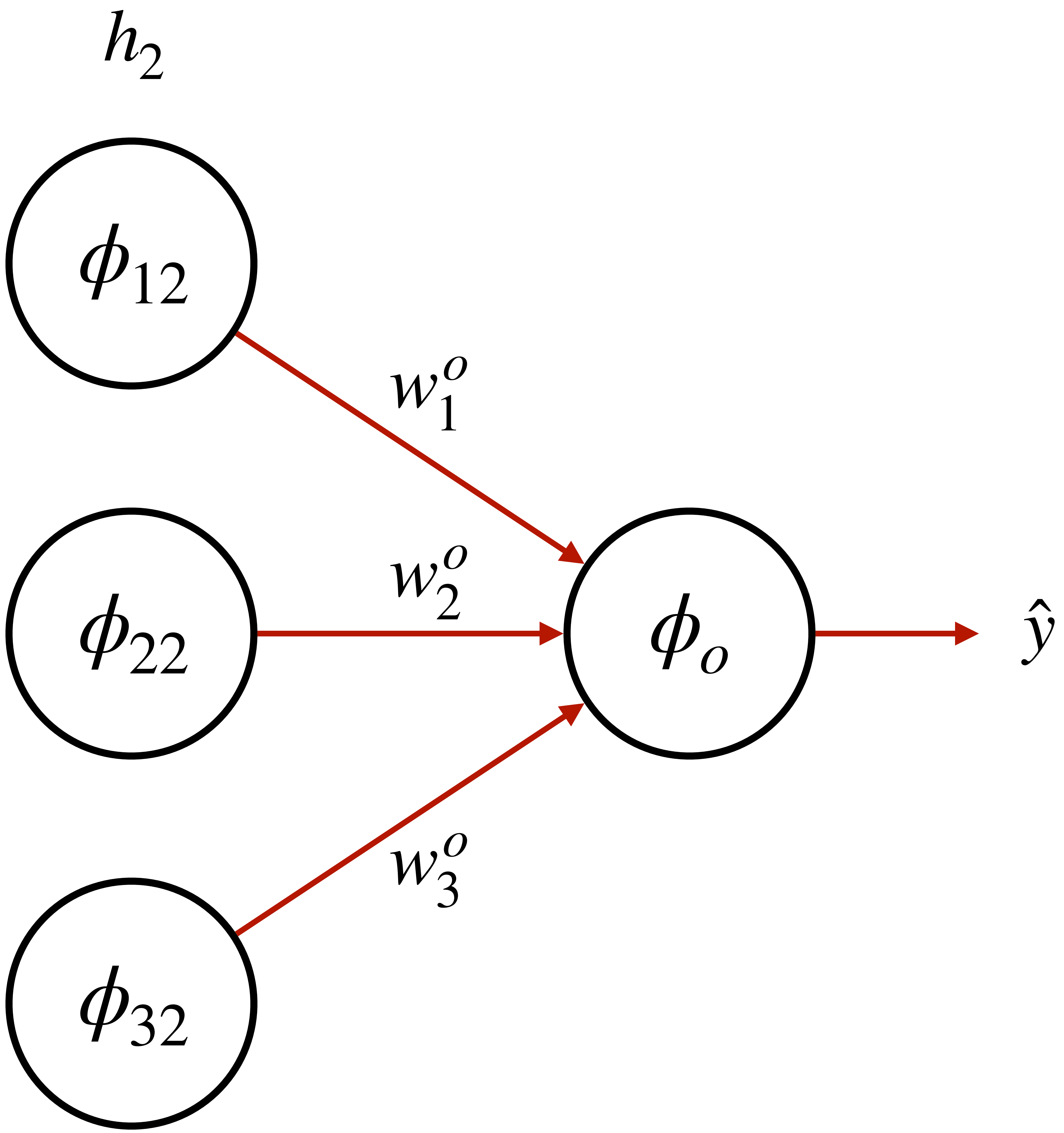
$$\hat{y} = \phi_o(u)$$

$$u = w_1^o \times \phi_{12}(\,.\,) + w_2^o \times \phi_{22}(\,.\,) + w_3^o \times \phi_{32}(\,.\,)$$

$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \phi_{12}(\,.\,)} = \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} \frac{\partial u}{\partial \phi_{12}(\,.\,)}$$

$$= \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \phi_o(u)}{\partial u} w_1^0$$

Backpropagation Review: FFNs



$$\mathscr{L}(\hat{y}, y) = y \log P(\hat{y}) + (1 - y) \log P(1 - \hat{y})$$

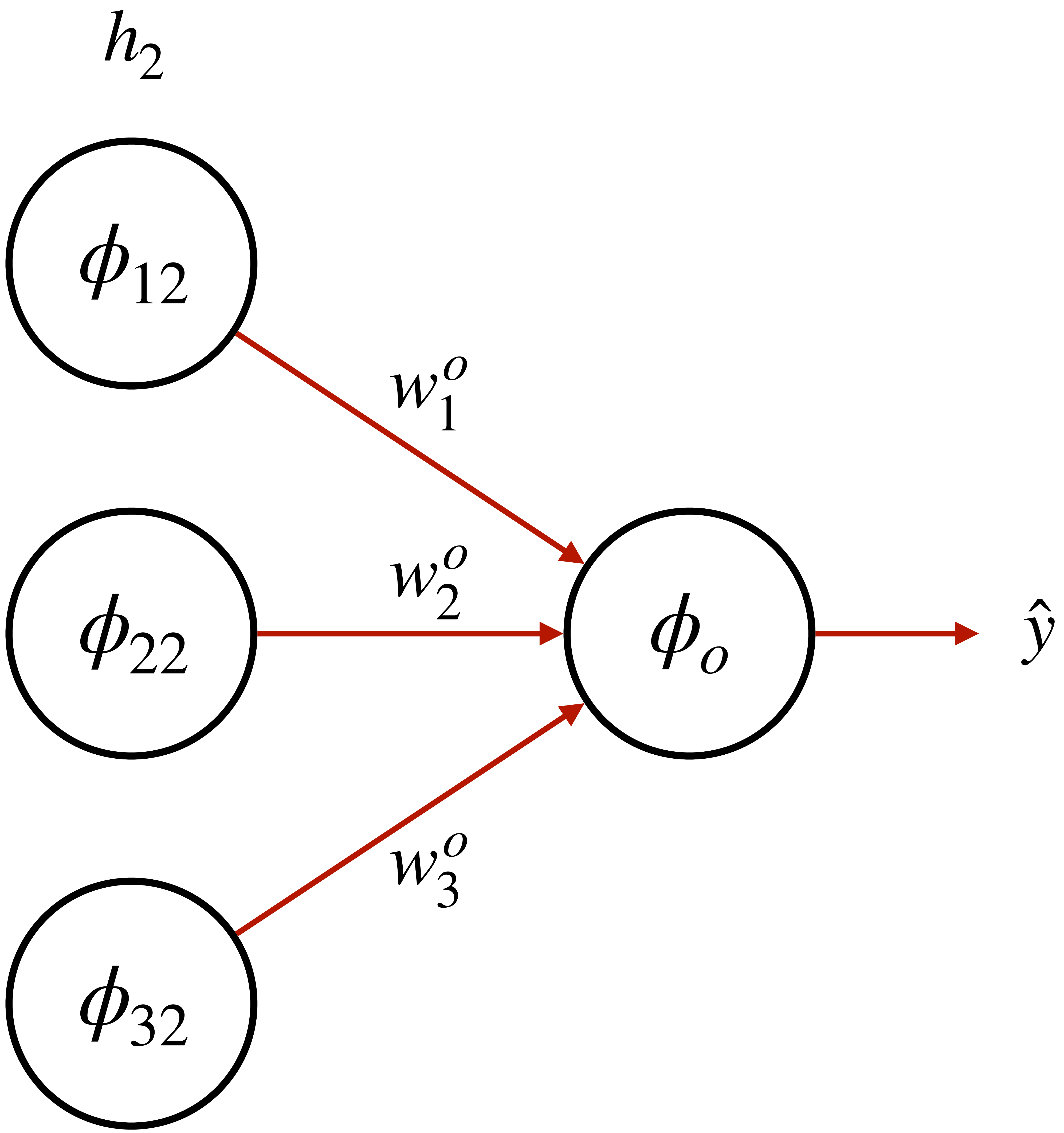
$$\hat{y} = \phi_o(u)$$

$$u = \boxed{w_1^o \times \phi_{12}(\,.\,)} + w_2^o \times \phi_{22}(\,.\,) + w_3^o \times \phi_{32}(\,.\,)$$

$$\frac{\partial \mathscr{L}(\hat{y}, y)}{\partial \phi_{12}(\,.\,)} = \frac{\partial \mathscr{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} \boxed{\frac{\partial u}{\partial \phi_{12}(\,.\,)}}$$

$$= \frac{\partial \mathscr{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \phi_o(u)}{\partial u} \boxed{w_1^0}$$

Backpropagation Review: FFNs



$$\mathscr{L}(\hat{y}, y) = y \log P(\hat{y}) + (1 - y) \log P(1 - \hat{y})$$

$$\hat{y} = \phi_o(u)$$

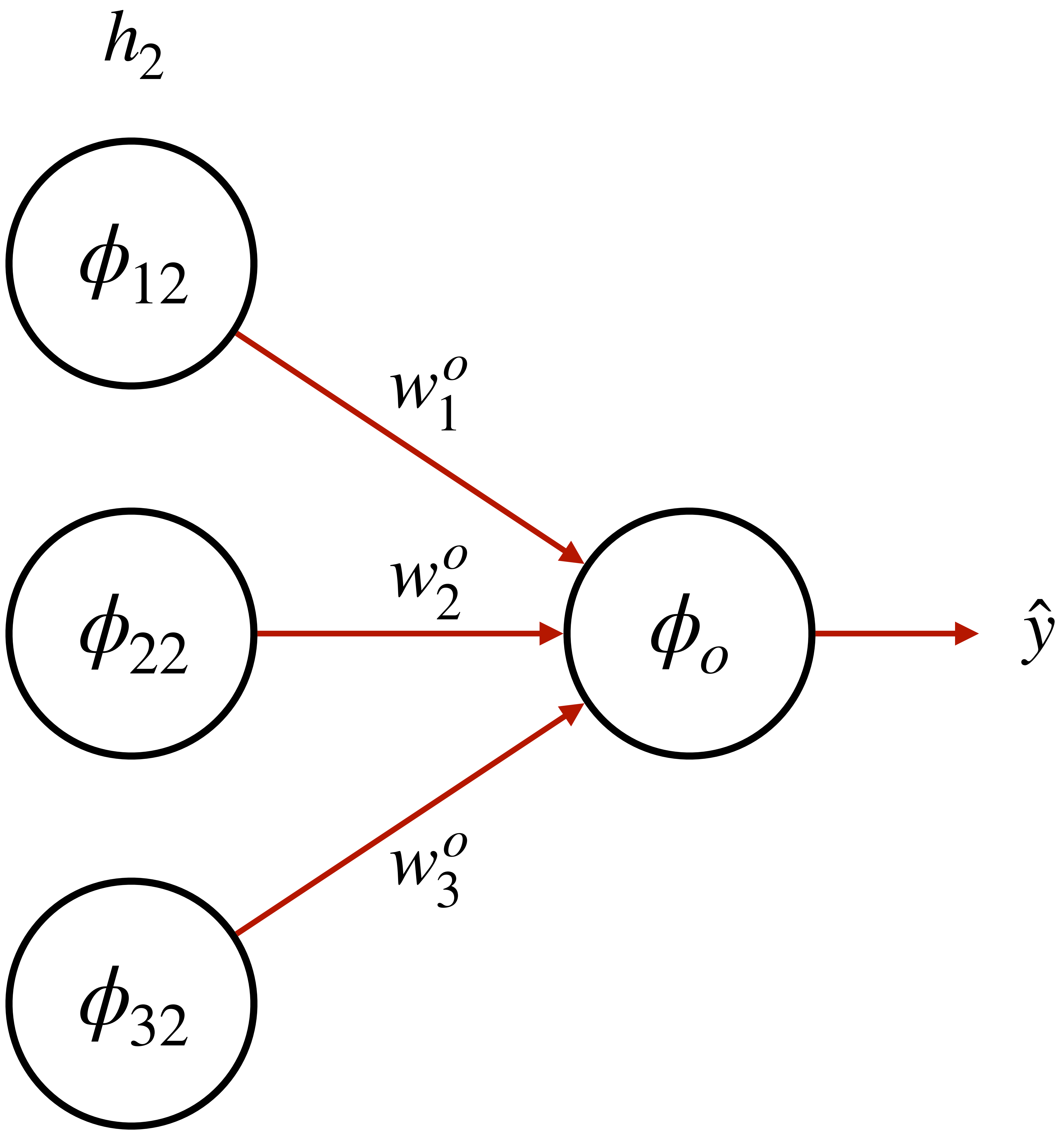
$$u = w_1^o \times \phi_{12}(\,.\,) + w_2^o \times \phi_{22}(\,.\,) + w_3^o \times \phi_{32}(\,.\,)$$

$$\frac{\partial \mathscr{L}(\hat{y}, y)}{\partial \phi_{12}(\,.\,)} = \frac{\partial \mathscr{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} \frac{\partial u}{\partial \phi_{12}(\,.\,)}$$

$$= \frac{\partial \mathscr{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \phi_o(u)}{\partial u} w_1^0$$

Depends on label y

Backpropagation Review: FFNs



$$\mathcal{L}(\hat{y}, y) = y \log P(\hat{y}) + (1 - y) \log P(1 - \hat{y})$$

$$\hat{y} = \phi_o(u)$$

$$u = w_1^o \times \phi_{12}(\,.\,) + w_2^o \times \phi_{22}(\,.\,) + w_3^o \times \phi_{32}(\,.\,)$$

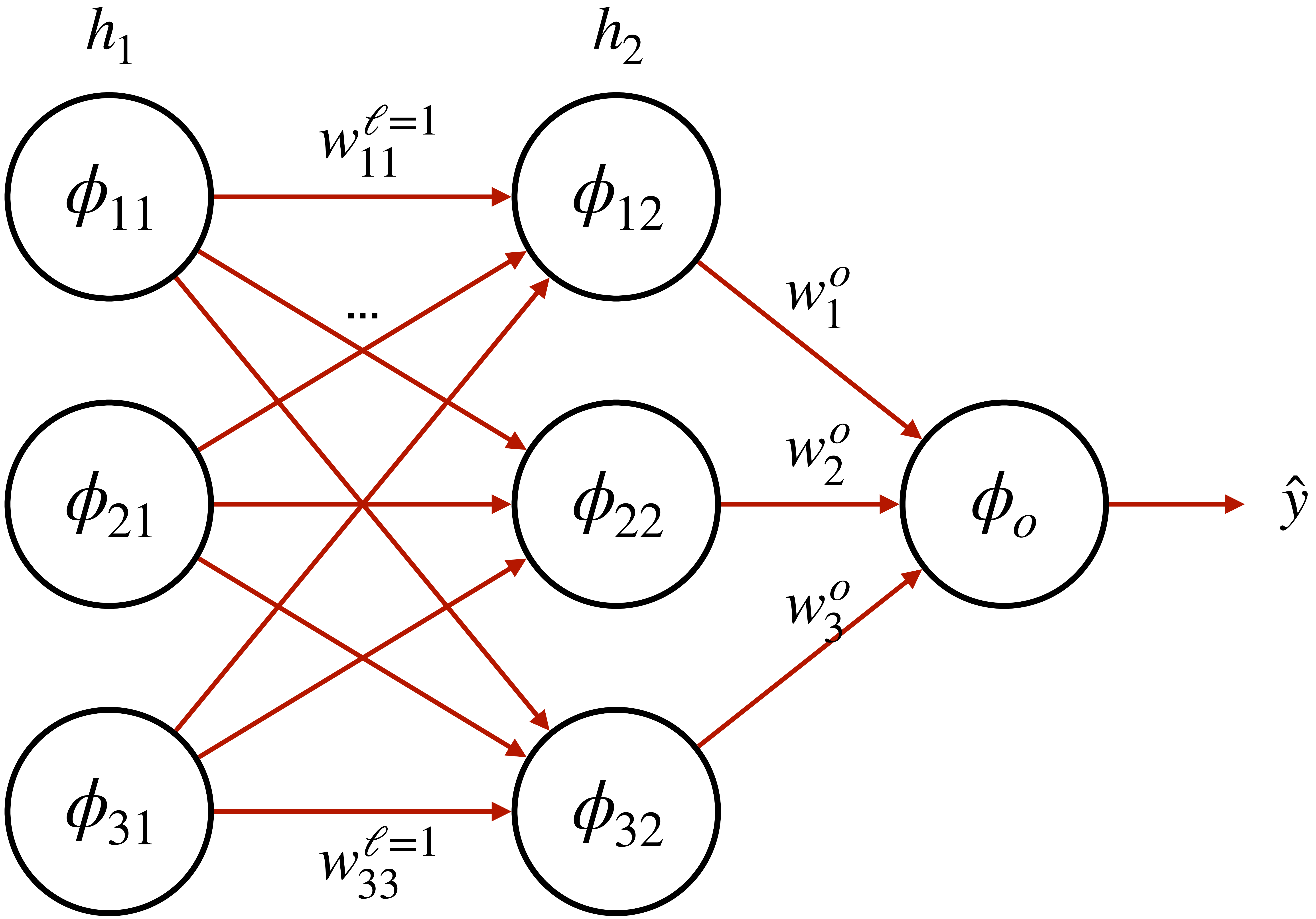
$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \phi_{12}(\,.\,)} = \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} \frac{\partial u}{\partial \phi_{12}(\,.\,)}$$

$$= \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \phi_o(u)}{\partial u} w_1^0$$

Depends on label y

Depends on ϕ_o

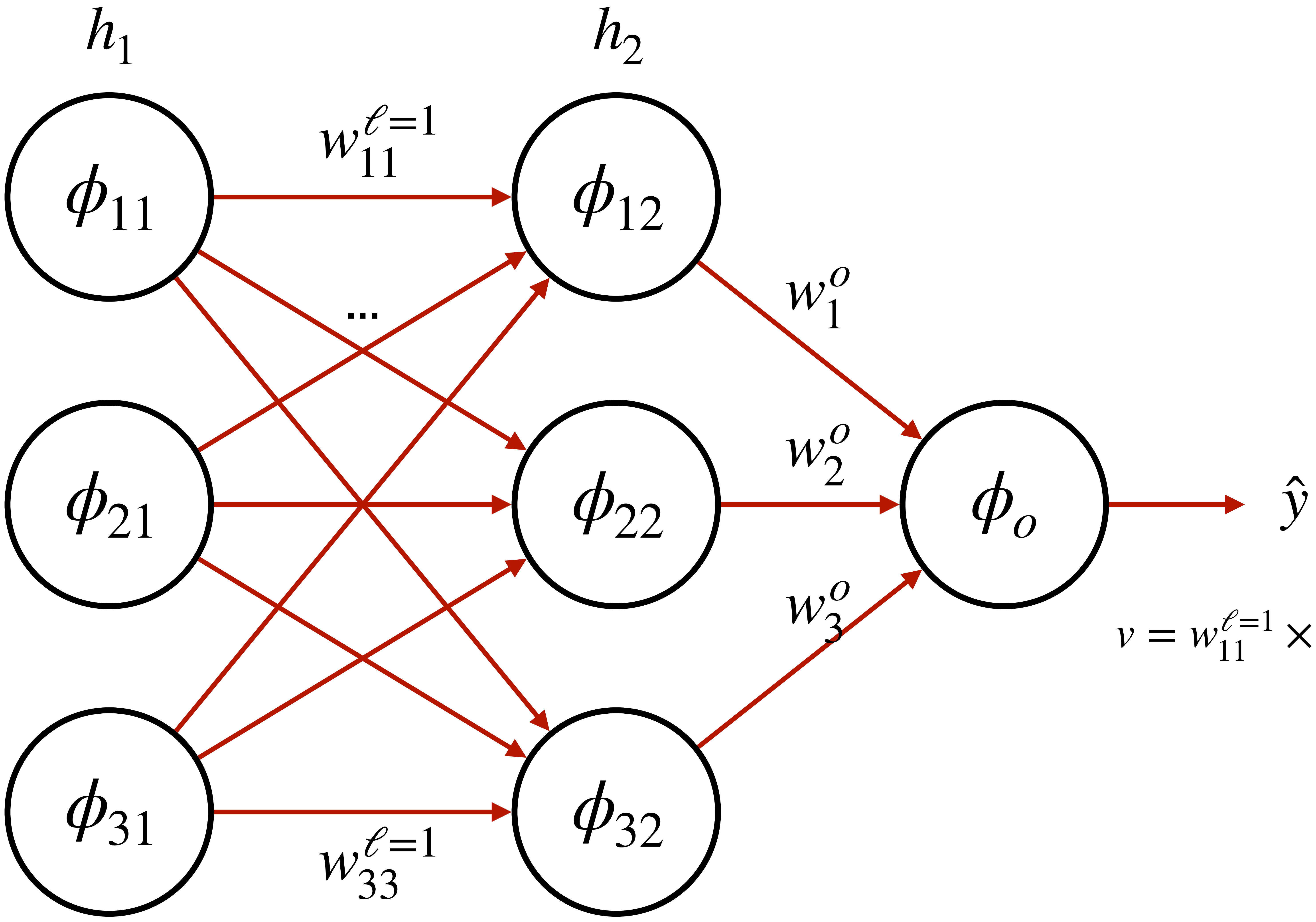
Backpropagation Review: FFNs



$$\frac{\partial \mathscr{L}(\hat{y}, y)}{\partial \phi_{12}(\,.\,)} = \frac{\partial \mathscr{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} \frac{\partial u}{\partial \phi_{12}(\,.\,)}$$

$$= \frac{\partial \mathscr{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \phi_o(u)}{\partial u} w_1^0$$

Backpropagation Review: FFNs

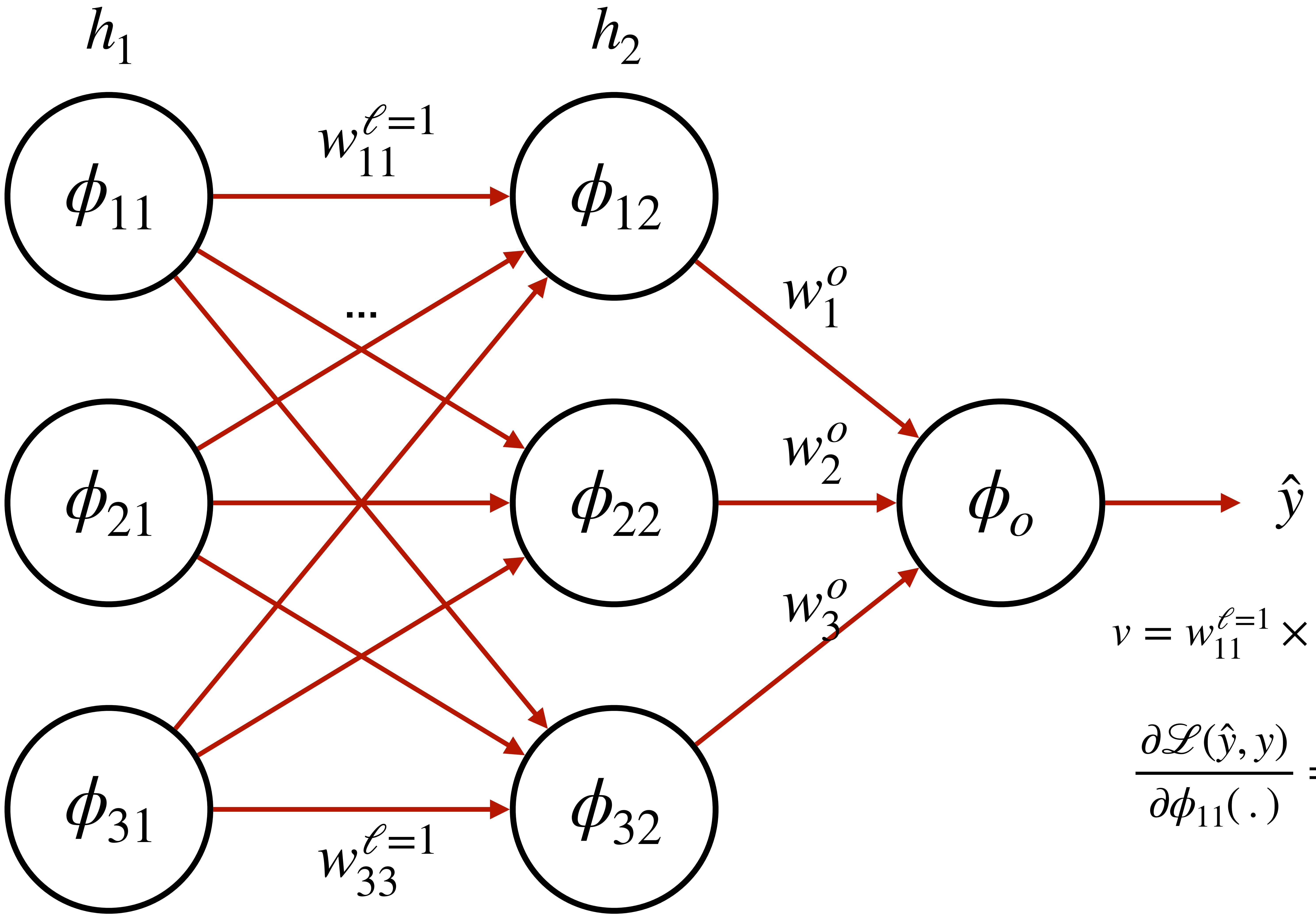


$$\frac{\partial \mathcal{L}(\hat{y},y)}{\partial \phi_{12}(\,.\,)} = \frac{\partial \mathcal{L}(\hat{y},y)}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial u}\frac{\partial u}{\partial \phi_{12}(\,.\,)}$$

$$= \frac{\partial \mathcal{L}(\hat{y},y)}{\partial \hat{y}}\frac{\partial \phi_o(u)}{\partial u}w_1^0$$

$$v = w_{11}^{\ell=1}\times \phi_{11}(\,.\,) + w_{21}^{\ell=1}\times \phi_{21}(\,.\,) + w_{31}^{\ell=1}\times \phi_{31}(\,.\,)$$

Backpropagation Review: FFNs



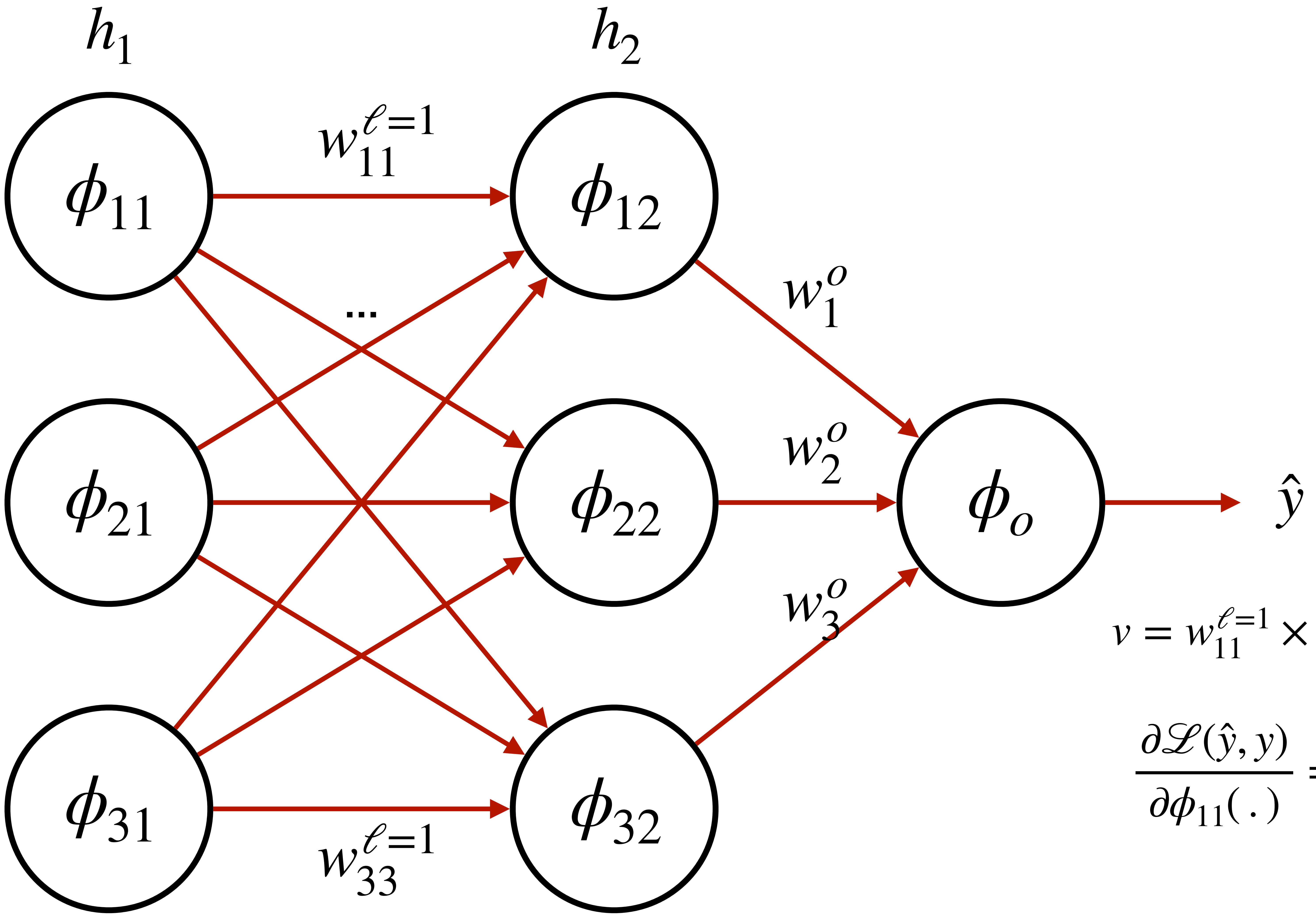
$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \phi_{12}(\,.\,)} = \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} \frac{\partial u}{\partial \phi_{12}(\,.\,)}$$

$$= \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \phi_o(u)}{\partial u} w_1^0$$

$$v = w_{11}^{\ell=1} \times \phi_{11}(\,.\,) + w_{21}^{\ell=1} \times \phi_{21}(\,.\,) + w_{31}^{\ell=1} \times \phi_{31}(\,.\,)$$

$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \phi_{11}(\,.\,)} = \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} \frac{\partial u}{\partial \phi_{12}(v)} \frac{\partial \phi_{12}(v)}{\partial v} \frac{\partial v}{\partial \phi_{11}(\,.\,)}$$

Backpropagation Review: FFNs



$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \phi_{12}(\,.\,)} = \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} \frac{\partial u}{\partial \phi_{12}(\,.\,)}$$

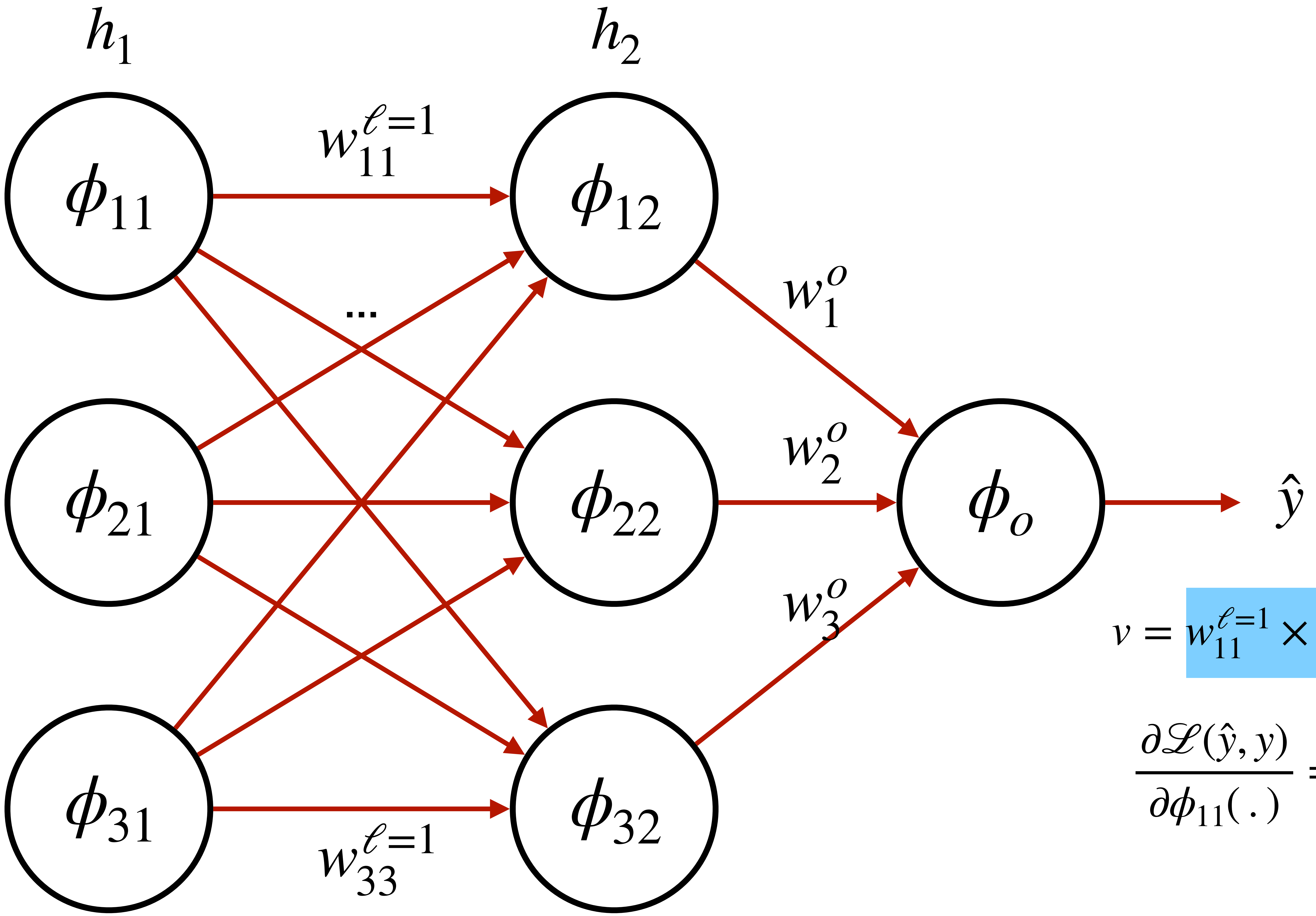
$$= \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \phi_o(u)}{\partial u} w_1^0$$

$$v = w_{11}^{\ell=1} \times \phi_{11}(\,.\,) + w_{21}^{\ell=1} \times \phi_{21}(\,.\,) + w_{31}^{\ell=1} \times \phi_{31}(\,.\,)$$

$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \phi_{11}(\,.\,)} = \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} \frac{\partial u}{\partial \phi_{12}(v)} \frac{\partial \phi_{12}(v)}{\partial v} \frac{\partial v}{\partial \phi_{11}(\,.\,)}$$

$$= \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \phi_o(u)}{\partial u} w_1^0 \frac{\partial \phi_{12}(v)}{\partial v} w_{11}^{\ell=1}$$

Backpropagation Review: FFNs



$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \phi_{12}(\,.\,)} = \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} \frac{\partial u}{\partial \phi_{12}(\,.\,)}$$

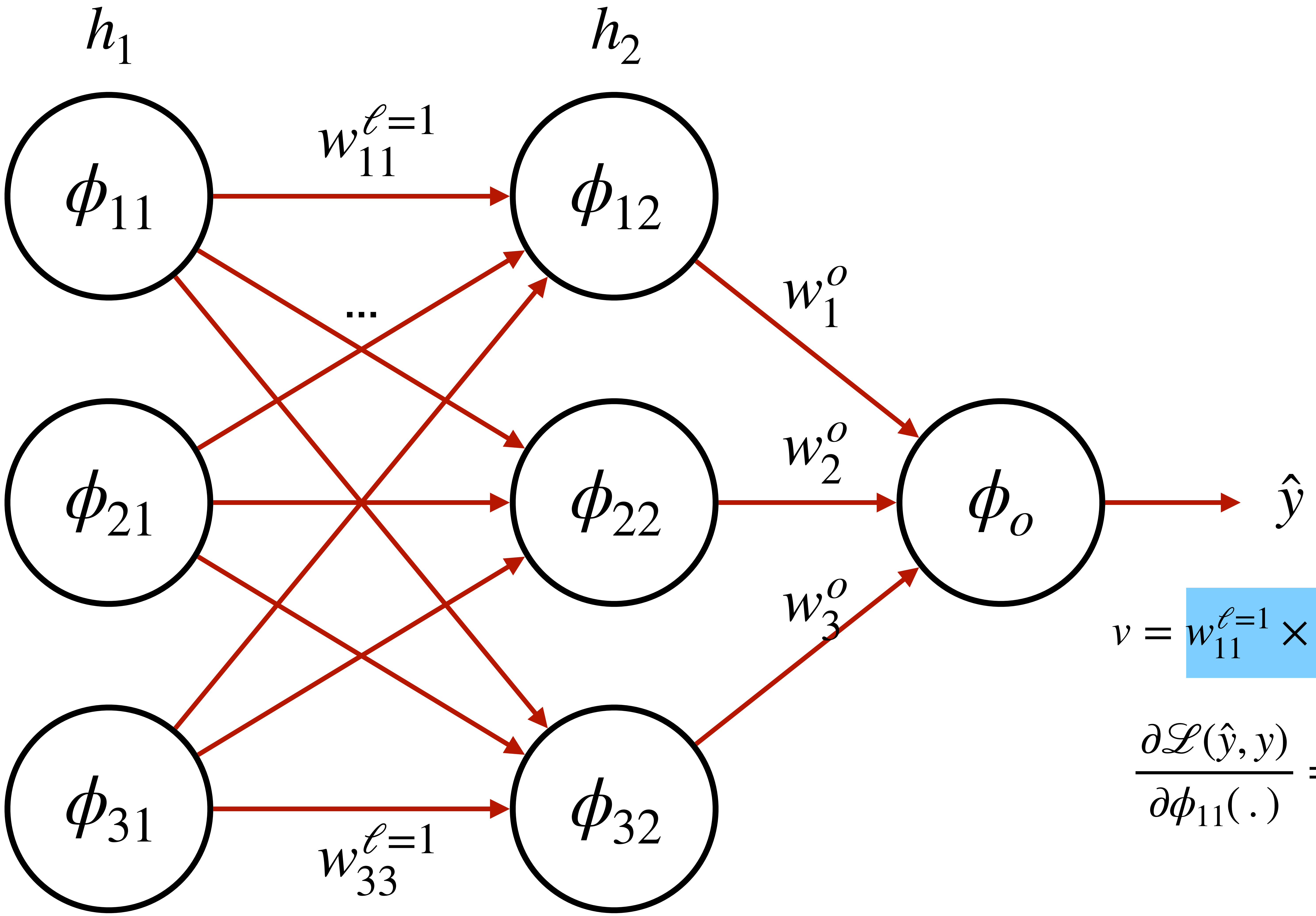
$$= \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \phi_o(u)}{\partial u} w_1^0$$

$$v = w_{11}^{\ell=1} \times \phi_{11}(\,.\,) + w_{21}^{\ell=1} \times \phi_{21}(\,.\,) + w_{31}^{\ell=1} \times \phi_{31}(\,.\,)$$

$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \phi_{11}(\,.\,)} = \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} \frac{\partial u}{\partial \phi_{12}(v)} \frac{\partial \phi_{12}(v)}{\partial v} \frac{\partial v}{\partial \phi_{11}(\,.\,)}$$

$$= \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \phi_o(u)}{\partial u} w_1^0 \frac{\partial \phi_{12}(v)}{\partial v} w_{11}^{\ell=1}$$

Backpropagation Review: FFNs



$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \phi_{12}(\,.\,)} = \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} \frac{\partial u}{\partial \phi_{12}(\,.\,)}$$

$$= \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \phi_o(u)}{\partial u} w_1^0$$

Depends on ϕ_{12}

$$v = w_{11}^{\ell=1} \times \phi_{11}(\,.\,) + w_{21}^{\ell=1} \times \phi_{21}(\,.\,) + w_{31}^{\ell=1} \times \phi_{31}(\,.\,)$$

$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \phi_{11}(\,.\,)} = \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} \frac{\partial u}{\partial \phi_{12}(v)} \frac{\partial \phi_{12}(v)}{\partial v} \frac{\partial v}{\partial \phi_{11}(\,.\,)}$$

$$= \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \phi_o(u)}{\partial u} w_1^0 \frac{\partial \phi_{12}(v)}{\partial v} w_{11}^{\ell=1}$$

Backpropagation through Time

$$z_t = \sigma\left(W_{zh}h_t + b_z\right)$$

$$h_t = \sigma\left(W_{hx}x_t + W_{hh}h_{t-1} + b_h\right)$$

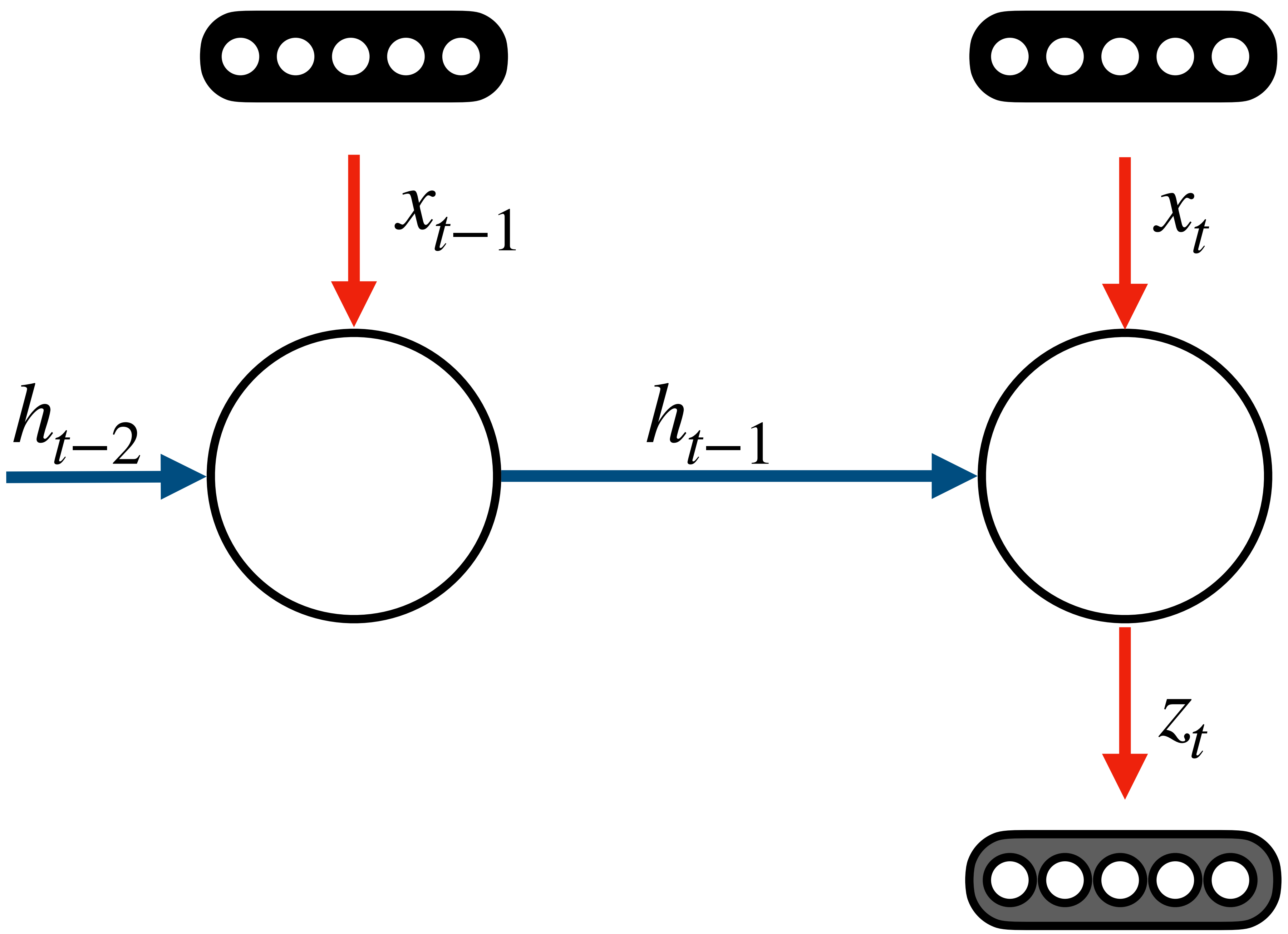
$$v = W_{zh}h_t + b_z \qquad z_t = \sigma(v)$$

$$u = W_{hx}x_t + W_{hh}h_{t-1} + b_h \qquad h_t = \sigma(u)$$

$$\frac{\partial z_t}{\partial h_t} = \frac{\partial \sigma(v)}{\partial v}\frac{\partial v}{\partial h_t} = \frac{\partial \sigma(v)}{\partial v}W_{zh}$$

$$\frac{\partial h_t}{\partial x_t} = \frac{\partial \sigma(u)}{\partial u}\frac{\partial u}{\partial x_t} = \frac{\partial \sigma(u)}{\partial u}W_{hx}$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial \sigma(u)}{\partial u}\frac{\partial u}{\partial h_{t-1}} = \frac{\partial \sigma(u)}{\partial u}W_{hh}$$



Backpropagation through Time

$$z_t = \sigma\left(W_{zh}h_t + b_z\right)$$

$$h_t = \sigma\left(W_{hx}x_t + W_{hh}h_{t-1} + b_h\right)$$

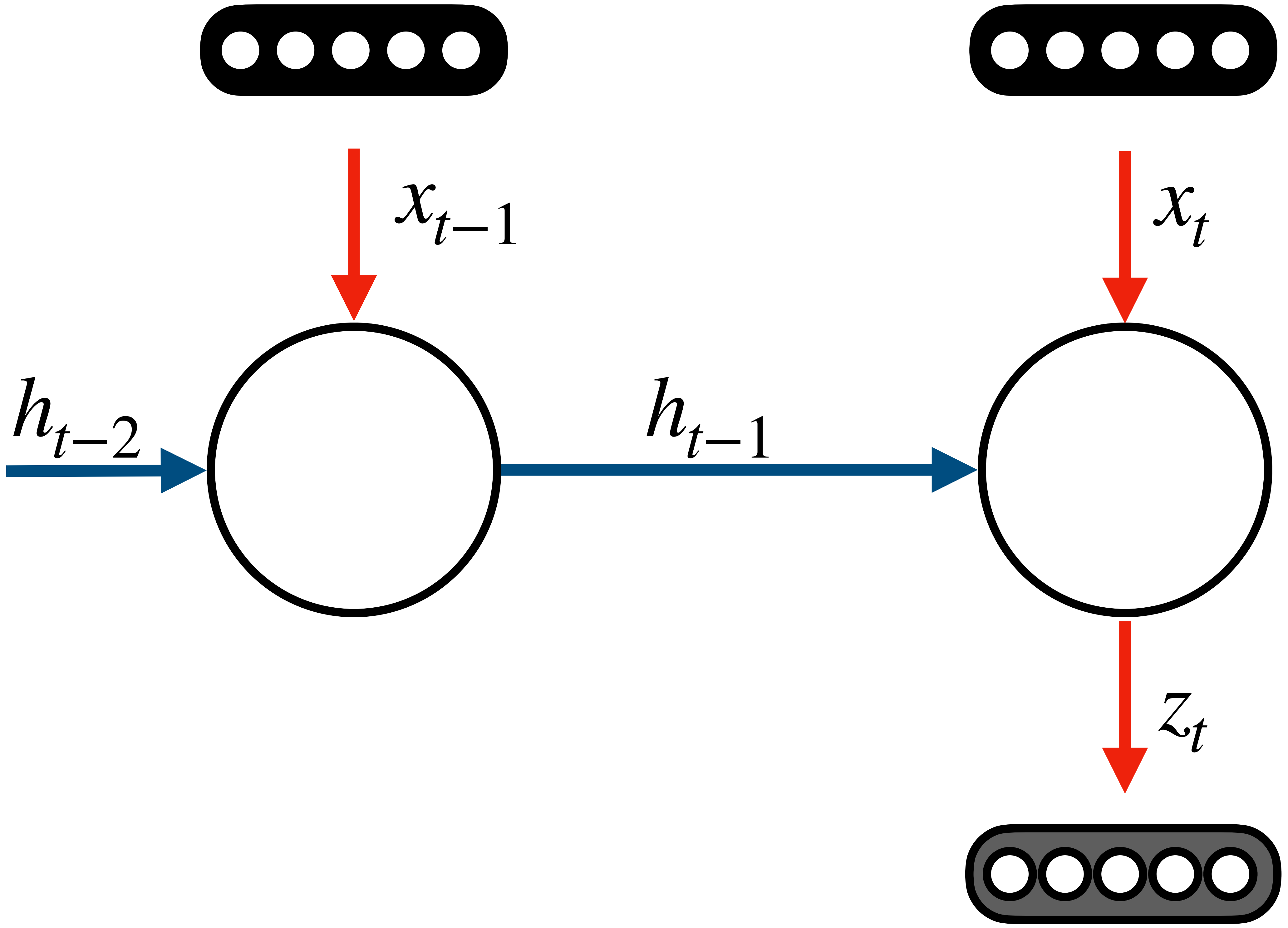
$$v = W_{zh}h_t + b_z \qquad z_t = \sigma(v)$$

$$u = W_{hx}x_t + W_{hh}h_{t-1} + b_h \qquad h_t = \sigma(u)$$

$$\frac{\partial z_t}{\partial h_t} = \frac{\partial \sigma(v)}{\partial v}\frac{\partial v}{\partial h_t} = \frac{\partial \sigma(v)}{\partial v}W_{zh}$$

$$\frac{\partial h_t}{\partial x_t} = \frac{\partial \sigma(u)}{\partial u}\frac{\partial u}{\partial x_t} = \frac{\partial \sigma(u)}{\partial u}W_{hx}$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial \sigma(u)}{\partial u}\frac{\partial u}{\partial h_{t-1}} = \frac{\partial \sigma(u)}{\partial u}W_{hh}$$



$$\frac{\partial z_t}{\partial h_{t-1}} = \frac{\partial \sigma(v)}{\partial v}\frac{\partial v}{\partial h_t}\frac{\partial \sigma(u)}{\partial u}\frac{\partial u}{\partial h_{t-1}} = \frac{\partial \sigma(v)}{\partial v}W_{zh}\frac{\partial \sigma(u)}{\partial u}W_{hh}$$

Vanishing Gradients

- **Learning Problem:** Long unrolled networks will crush gradients that backpropagate to earlier time steps

$$h_t = \boxed{\sigma}\left(W_{hx}x_t + W_{hh}\boxed{h_{t-1}} + b_h\right)$$

Vanishing Gradients

- **Learning Problem:** Long unrolled networks will crush gradients that backpropagate to earlier time steps

$$h_t = \boxed{\sigma}\left(W_{hx}x_t + W_{hh}\boxed{h_{t-1}} + b_h\right)$$

$$u = W_{hx}x_t + W_{hh}h_{t-1} + b_h$$

Vanishing Gradients

- **Learning Problem:** Long unrolled networks will crush gradients that backpropagate to earlier time steps

$$h_t = \boxed{\sigma}\left(W_{hx}x_t + W_{hh}\boxed{h_{t-1}} + b_h\right)$$

$$u = W_{hx}x_t + W_{hh}h_{t-1} + b_h$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial \sigma(u)}{\partial u}\frac{\partial u}{\partial h_{t-1}} = W_{hh}\boxed{\frac{\partial \sigma(u)}{\partial u}}$$

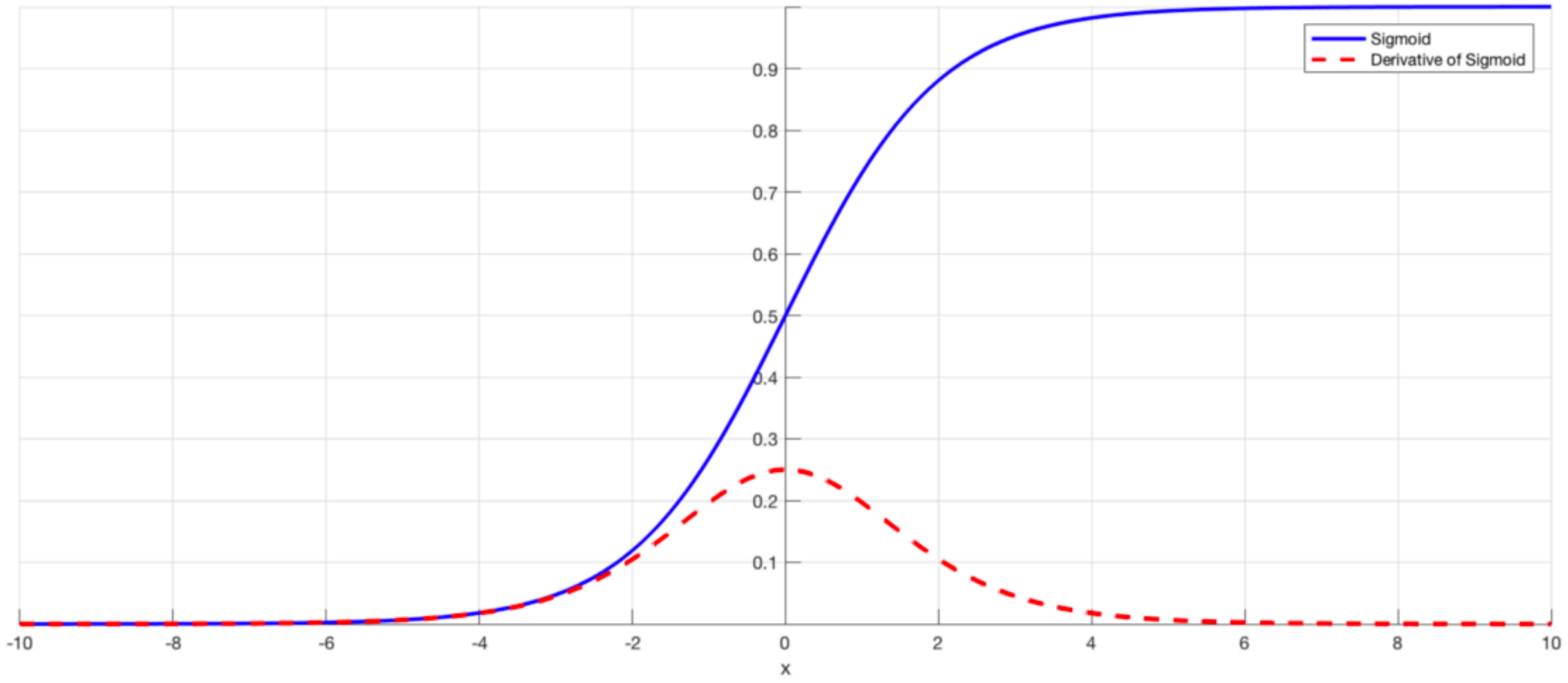
Vanishing Gradients

- **Learning Problem:** Long unrolled networks will crush gradients that backpropagate to earlier time steps

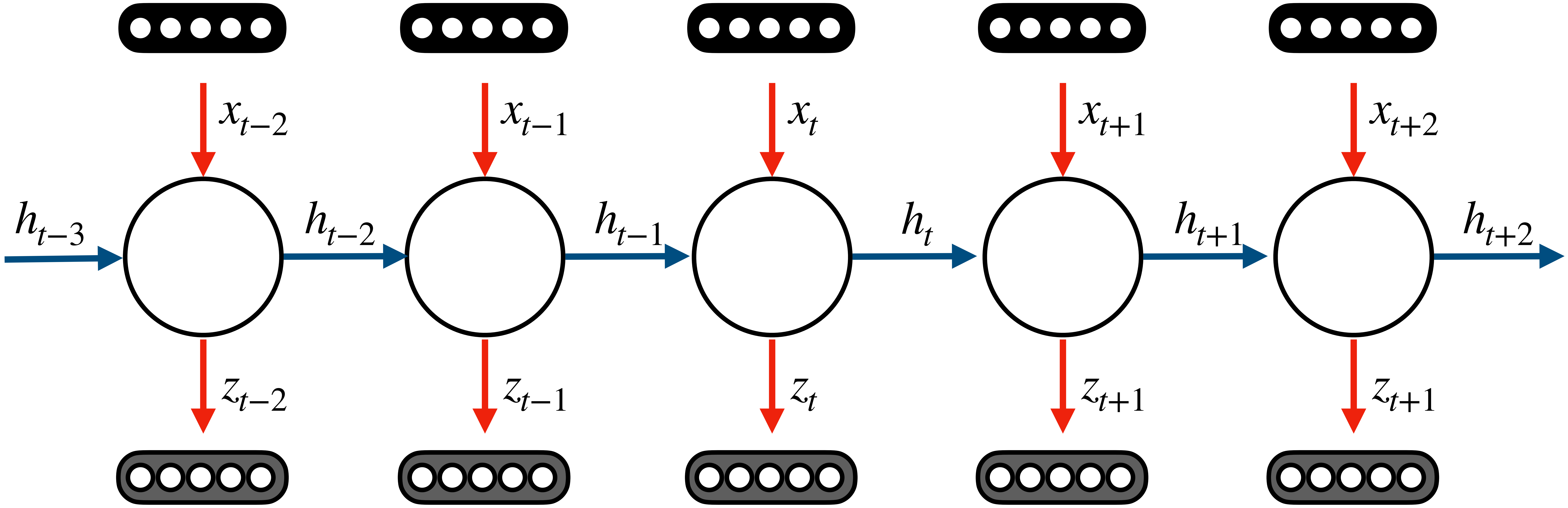
$$h_t = \boxed{\sigma}\left(W_{hx}x_t + W_{hh}\boxed{h_{t-1}} + b_h\right)$$

$$u = W_{hx}x_t + W_{hh}h_{t-1} + b_h$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial \sigma(u)}{\partial u}\frac{\partial u}{\partial h_{t-1}} = W_{hh}\boxed{\frac{\partial \sigma(u)}{\partial u}}$$

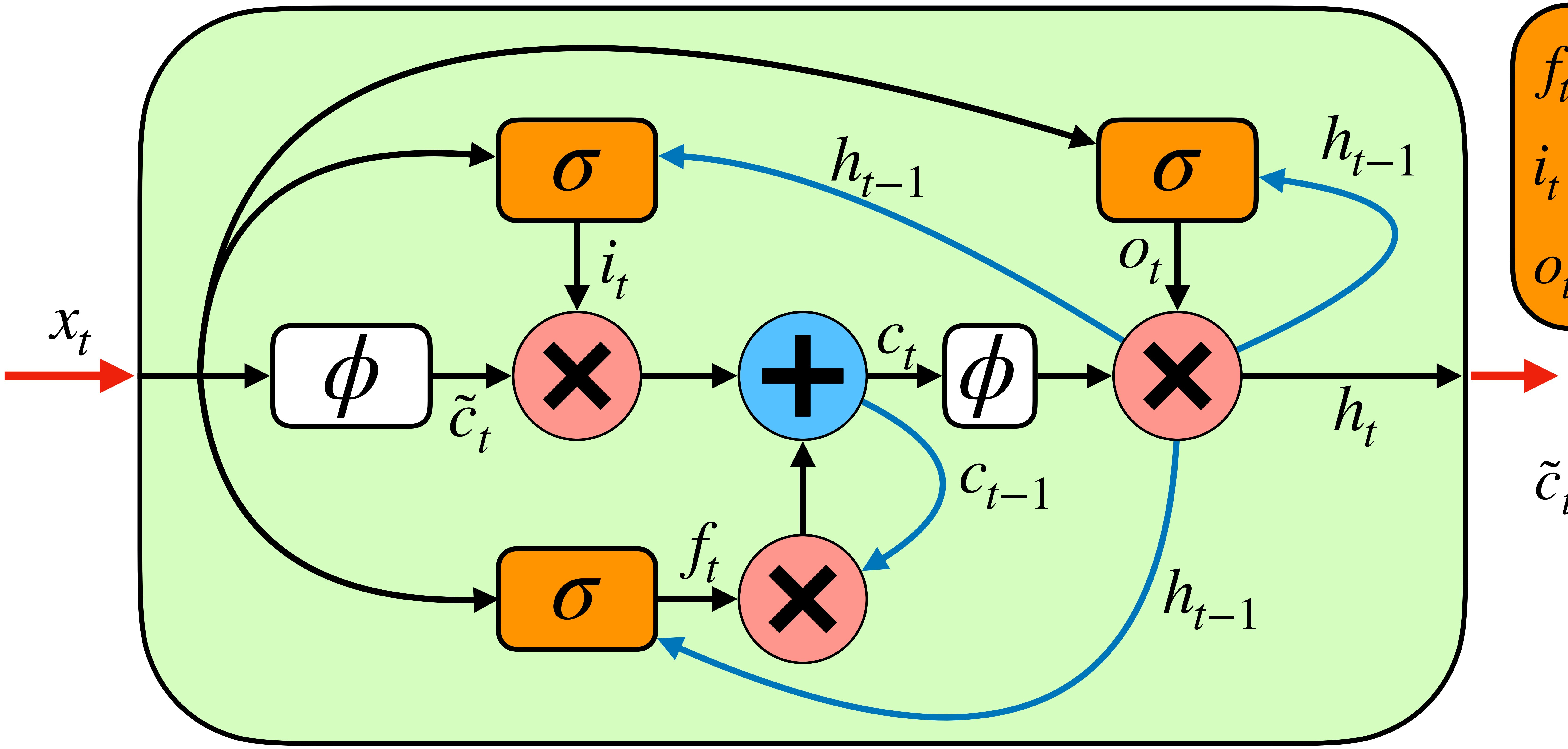


Vanishing Gradients



- While this is a problem in many neural networks, it is especially pronounced in Elman networks (RNNs) due to the sigmoid activation

Long Short Term Memory (LSTM)



Gates:

$$f_t = \sigma\left(W_{fx}x_t + W_{fh}h_{t-1} + b_f\right)$$

$$i_t = \sigma\left(W_{ix}x_t + W_{ih}h_{t-1} + b_i\right)$$

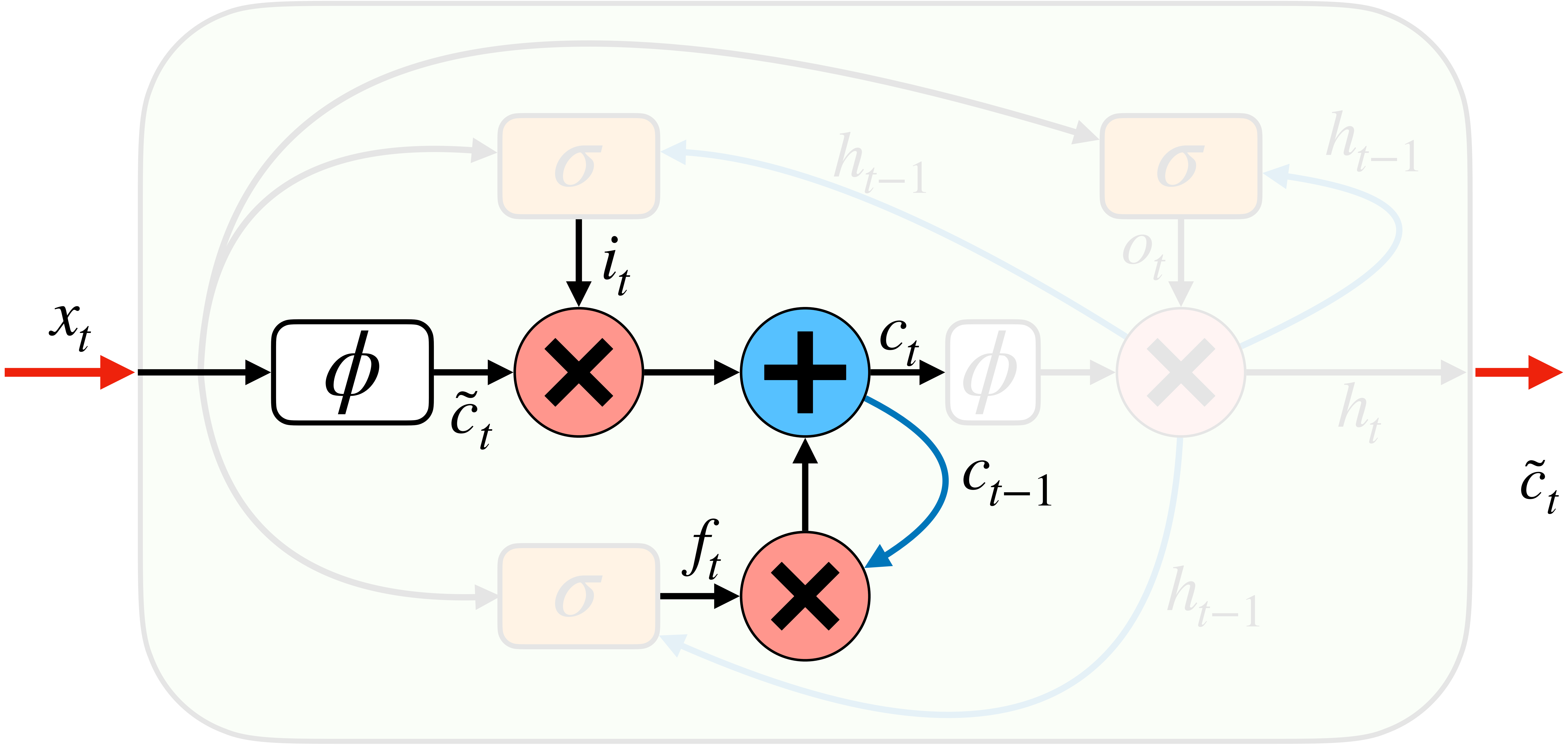
$$o_t = \sigma\left(W_{ox}x_t + W_{oh}h_{t-1} + b_o\right)$$

$$\tilde{c}_t = \phi\left(W_{cx}x_t + W_{ch}h_{t-1} + b_c\right)$$

$$c_t = i_t \times \tilde{c}_t + f_t \times c_{t-1}$$

$$h_t = o_t \times \phi(c_t)$$

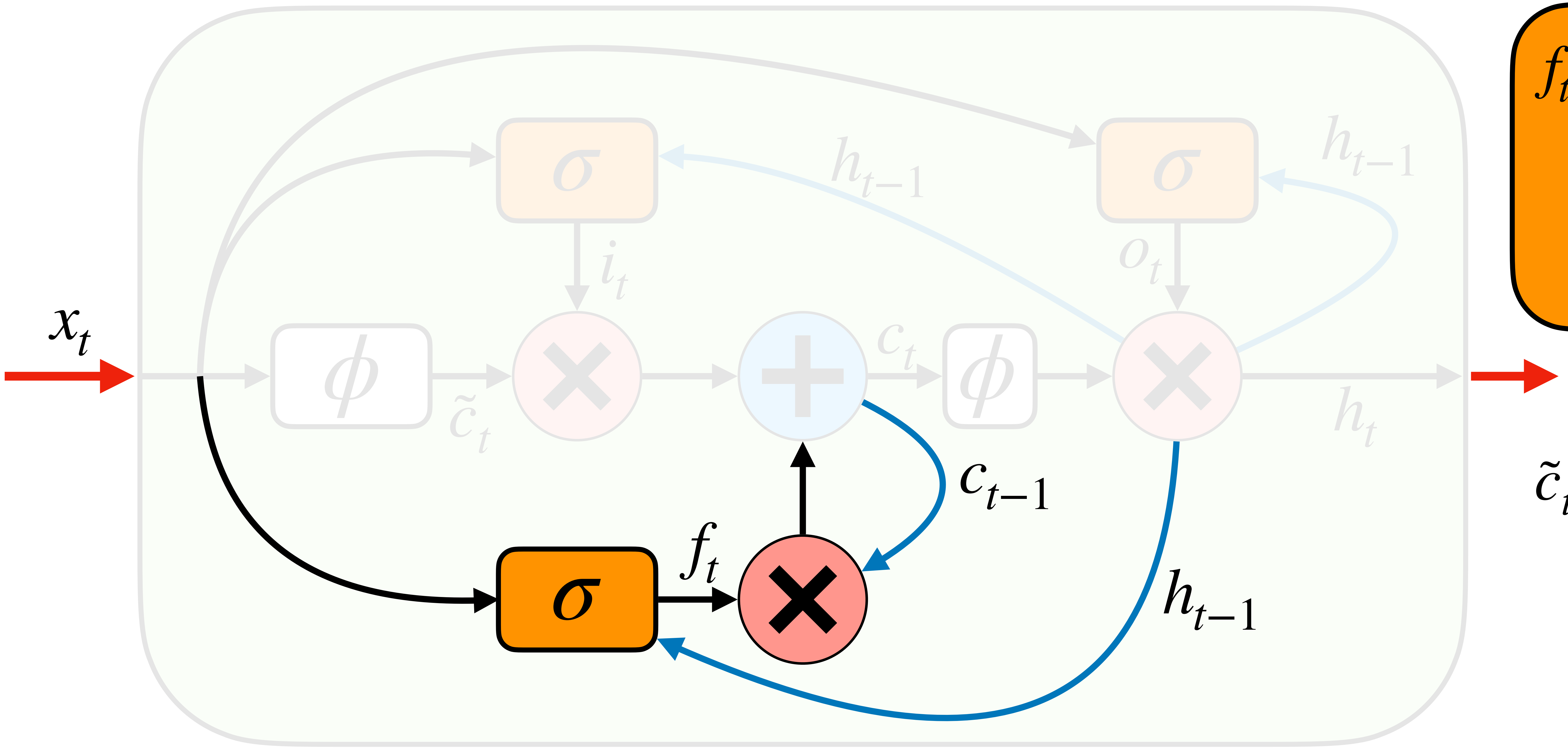
Cell State



$$\tilde{c}_t = \phi\left(W_{cx}x_t + W_{ch}h_{t-1} + b_c\right)$$

$$c_t = i_t \times \tilde{c}_t + f_t \times c_{t-1}$$

Forget Gate



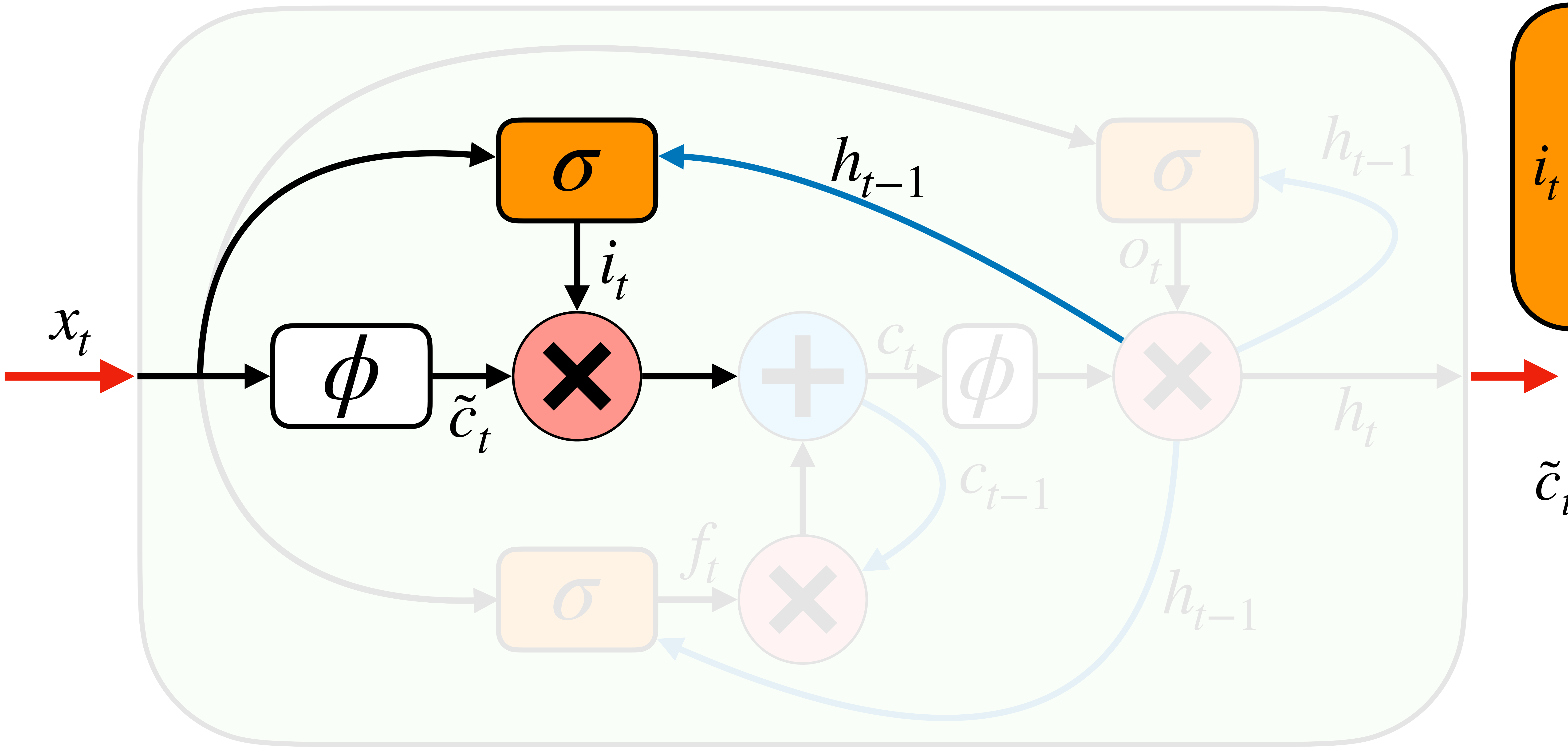
Gates:

$$f_t = \sigma\left(W_{fx}x_t + W_{fh}h_{t-1} + b_f\right)$$

$$\tilde{c}_t = \phi\left(W_{cx}x_t + W_{ch}h_{t-1} + b_c\right)$$

$$c_t = i_t \times \tilde{c}_t + f_t \times c_{t-1}$$

Input Gate



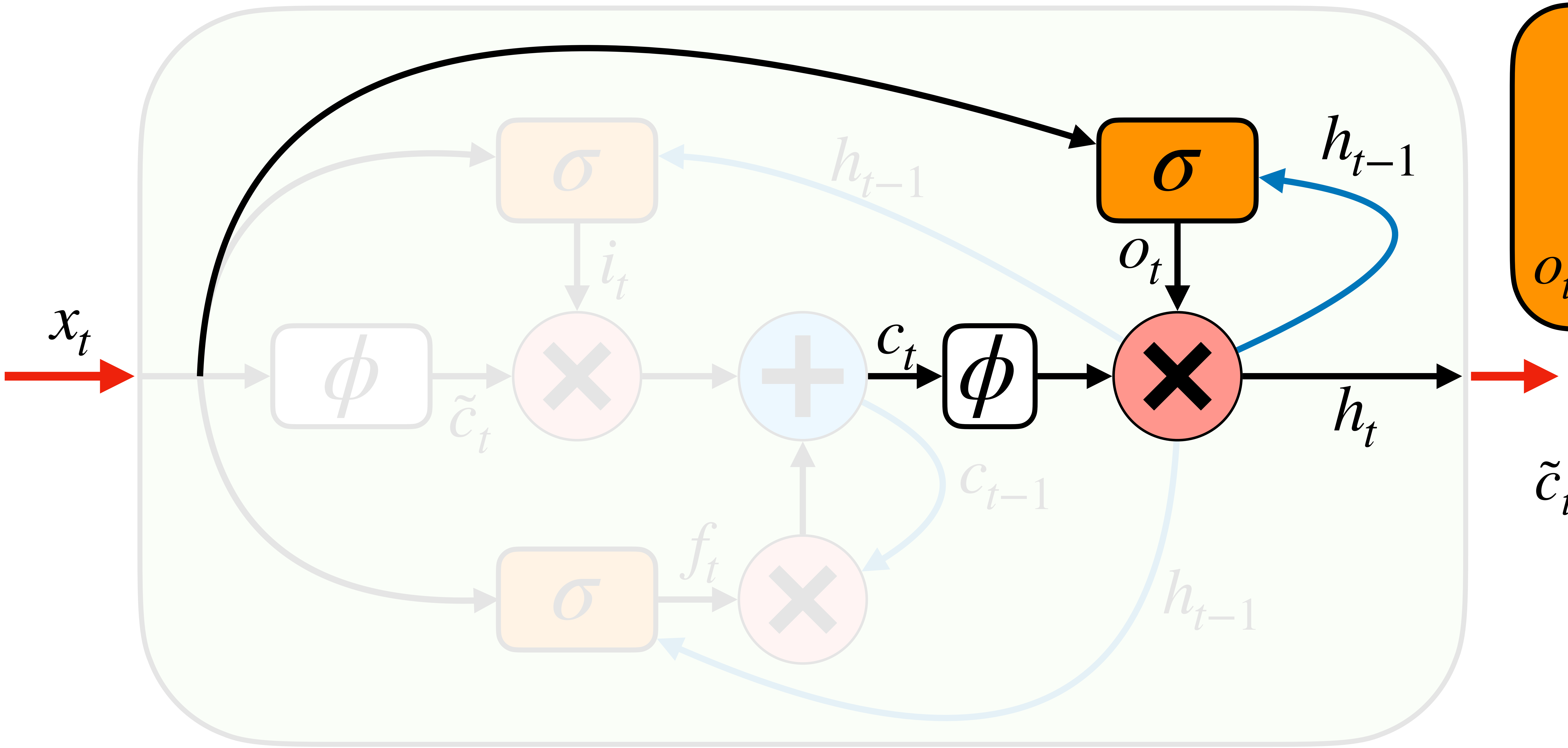
Gates:

$$i_t = \sigma\left(W_{ix}x_t + W_{ih}h_{t-1} + b_i\right)$$

$$\tilde{c}_t = \phi\left(W_{cx}x_t + W_{ch}h_{t-1} + b_c\right)$$

$$c_t = i_t \times \tilde{c}_t + f_t \times c_{t-1}$$

Output Gate



Gates:

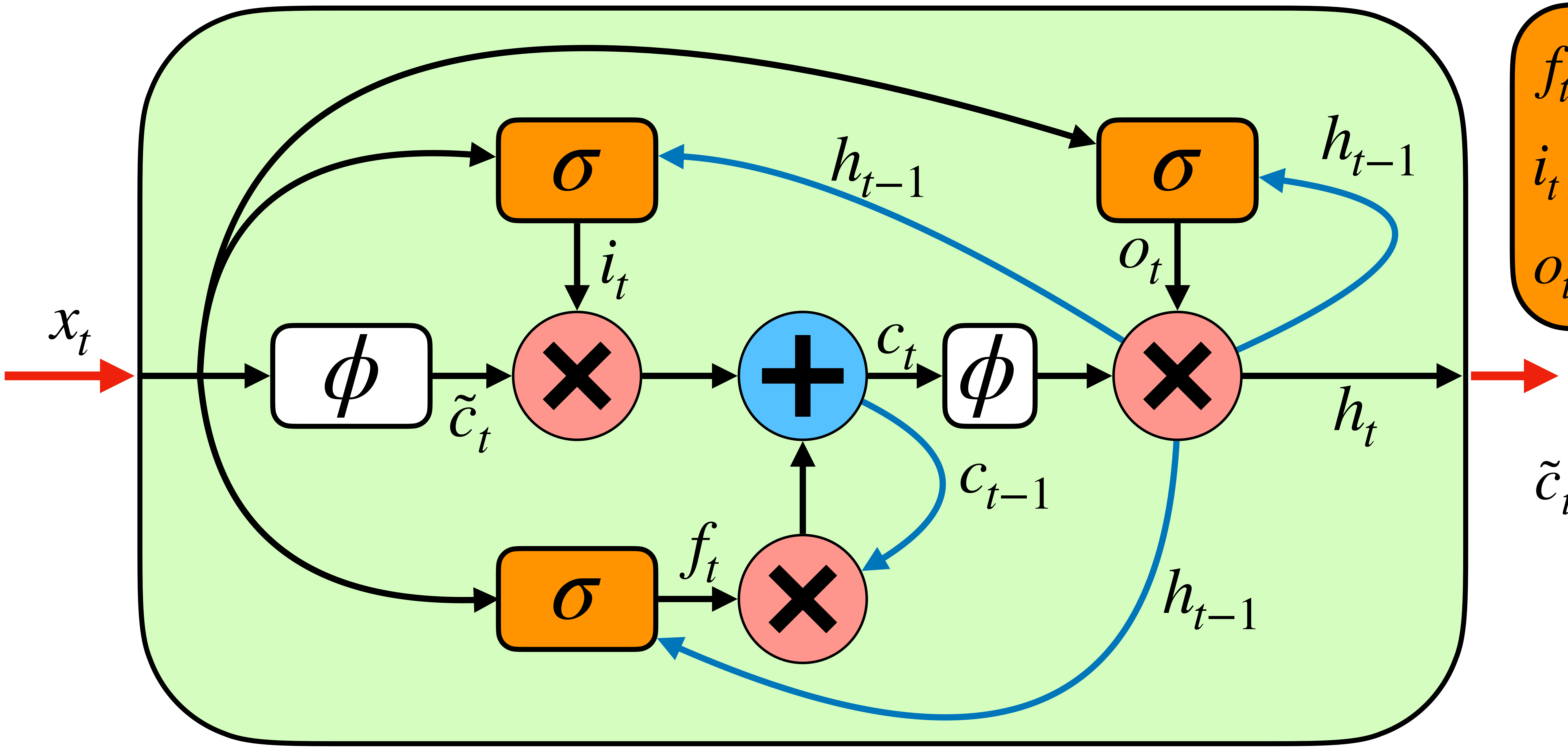
$$o_t = \sigma\left(W_{ox}x_t + W_{oh}h_{t-1} + b_o\right)$$

$$\tilde{c}_t = \phi\left(W_{cx}x_t + W_{ch}h_{t-1} + b_c\right)$$

$$c_t = i_t \times \tilde{c}_t + f_t \times c_{t-1}$$

$$h_t = o_t \times \phi(c_t)$$

Long Short Term Memory (LSTM)



Gates:

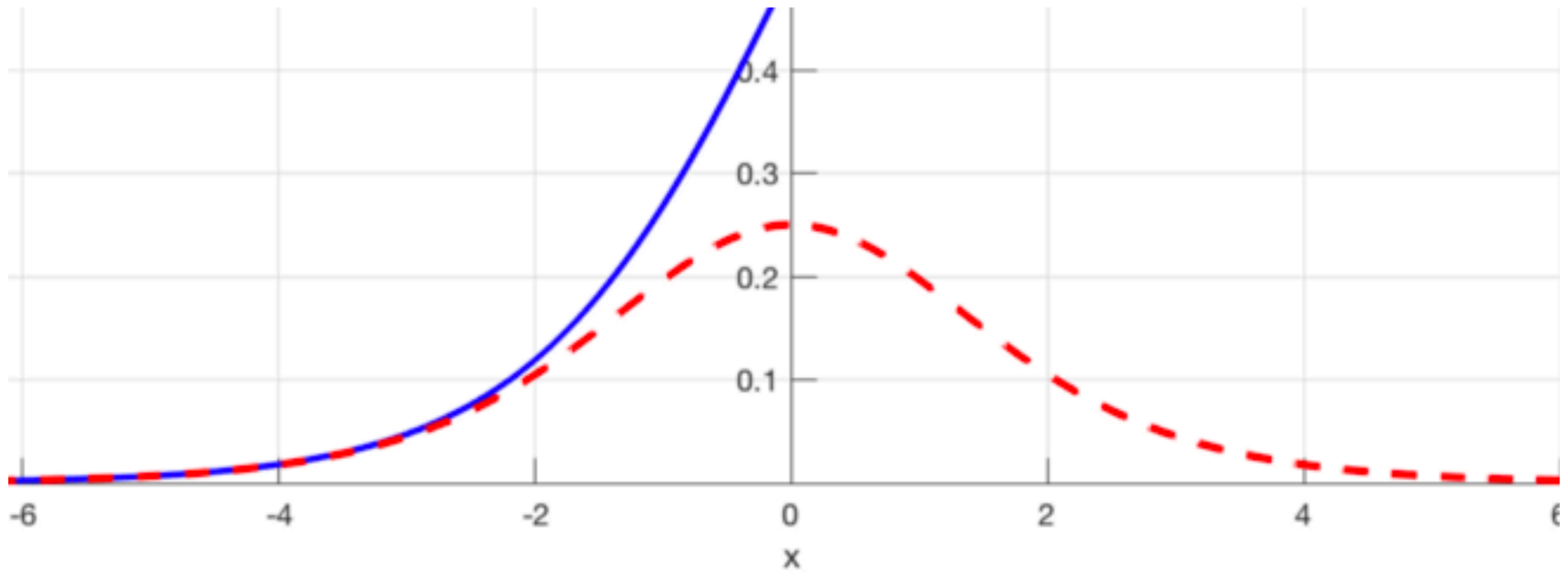
$$f_t = \sigma\left(W_{fx}x_t + W_{fh}h_{t-1} + b_f\right)$$
$$i_t = \sigma\left(W_{ix}x_t + W_{ih}h_{t-1} + b_i\right)$$
$$o_t = \sigma\left(W_{ox}x_t + W_{oh}h_{t-1} + b_o\right)$$

$$\tilde{c}_t = \phi\left(W_{cx}x_t + W_{ch}h_{t-1} + b_c\right)$$

$$c_t = i_t \times \tilde{c}_t + f_t \times c_{t-1}$$

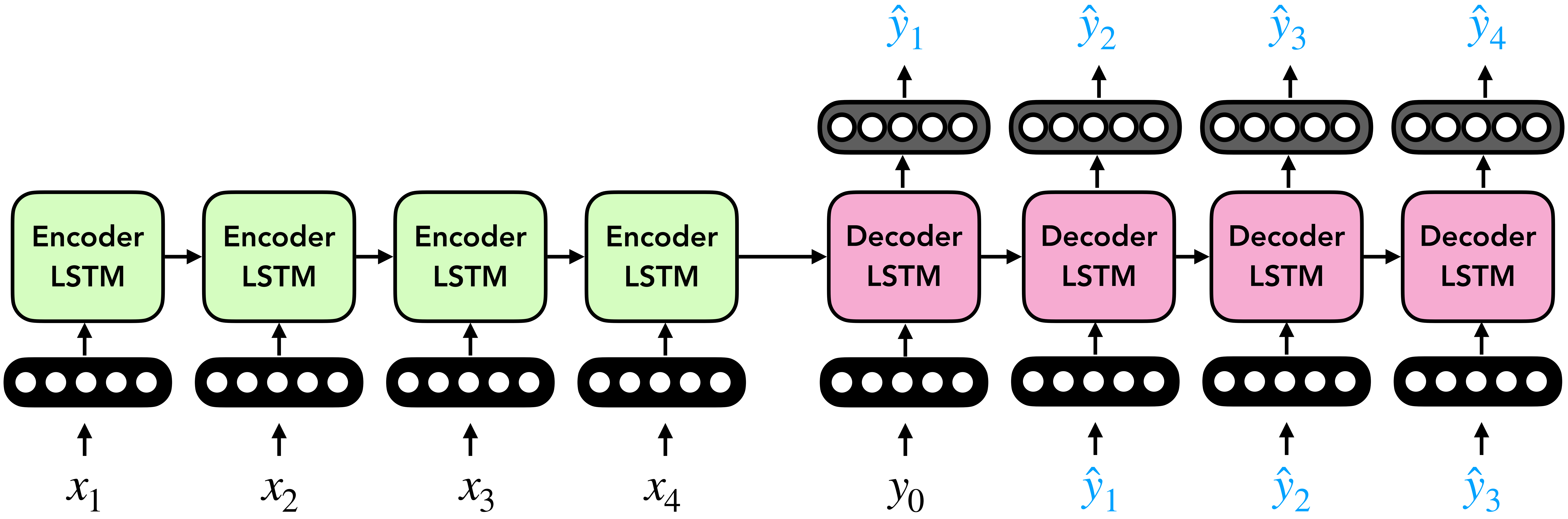
$$h_t = o_t \times \phi(c_t)$$

Vanishing Gradients?

Recurrent Neural Networks	Long Short Term Memory
State maintained by hidden state feedback	**State maintained by cell value**
$h_t = \sigma\left(W_{hx}x_t + W_{hh}h_{t-1} + b_h\right)$	$c_t = i_t \times \tilde{c}_t + f_t \times c_{t-1}$
Gradient systemically squashed by sigmoid	**Gradient set by value of forget gate**
	$\frac{\partial c_t}{\partial c_{t-1}} = f_t$
	Can still vanish, but only if forget gate closes!

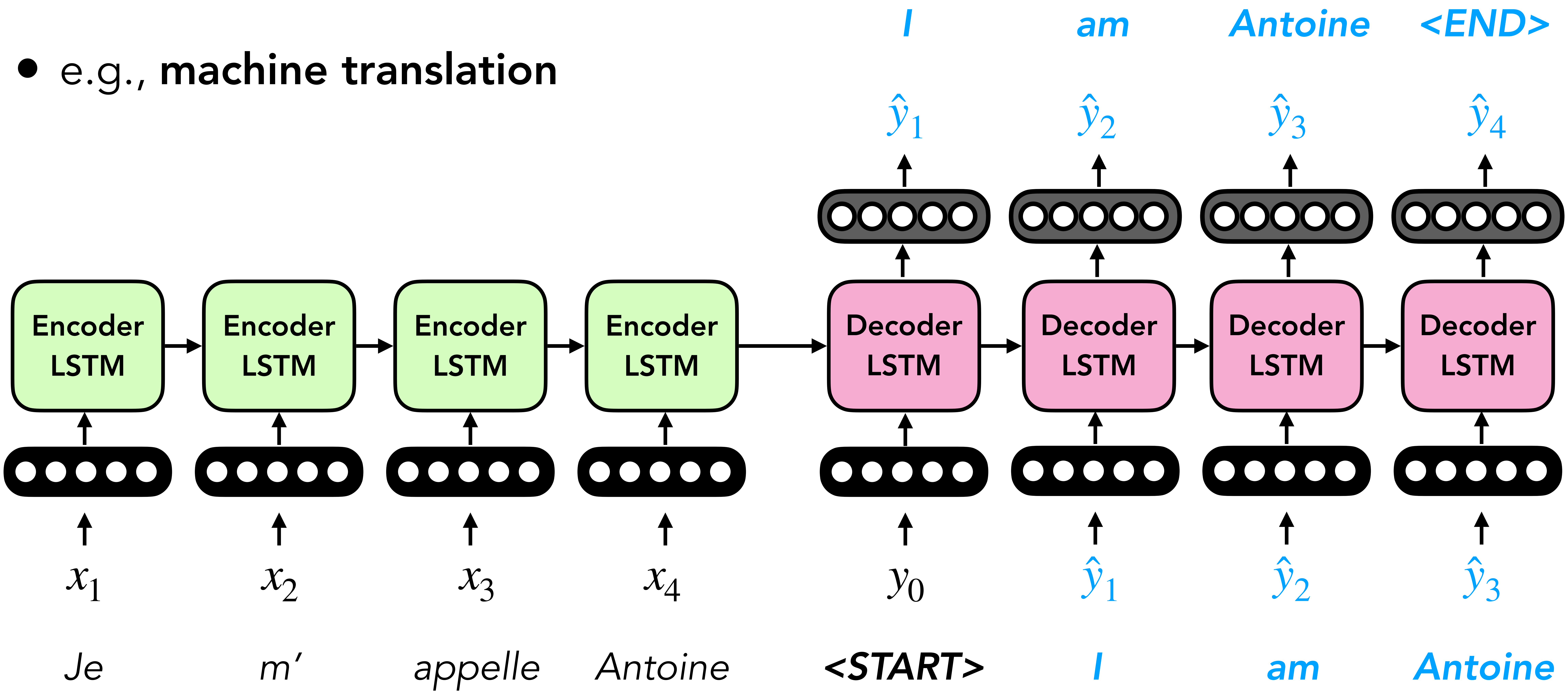
Encoder-Decoder Models

- Encode a sequence fully with one model and use its representation to seed a second model that decodes another sequence



Encoder-Decoder Models

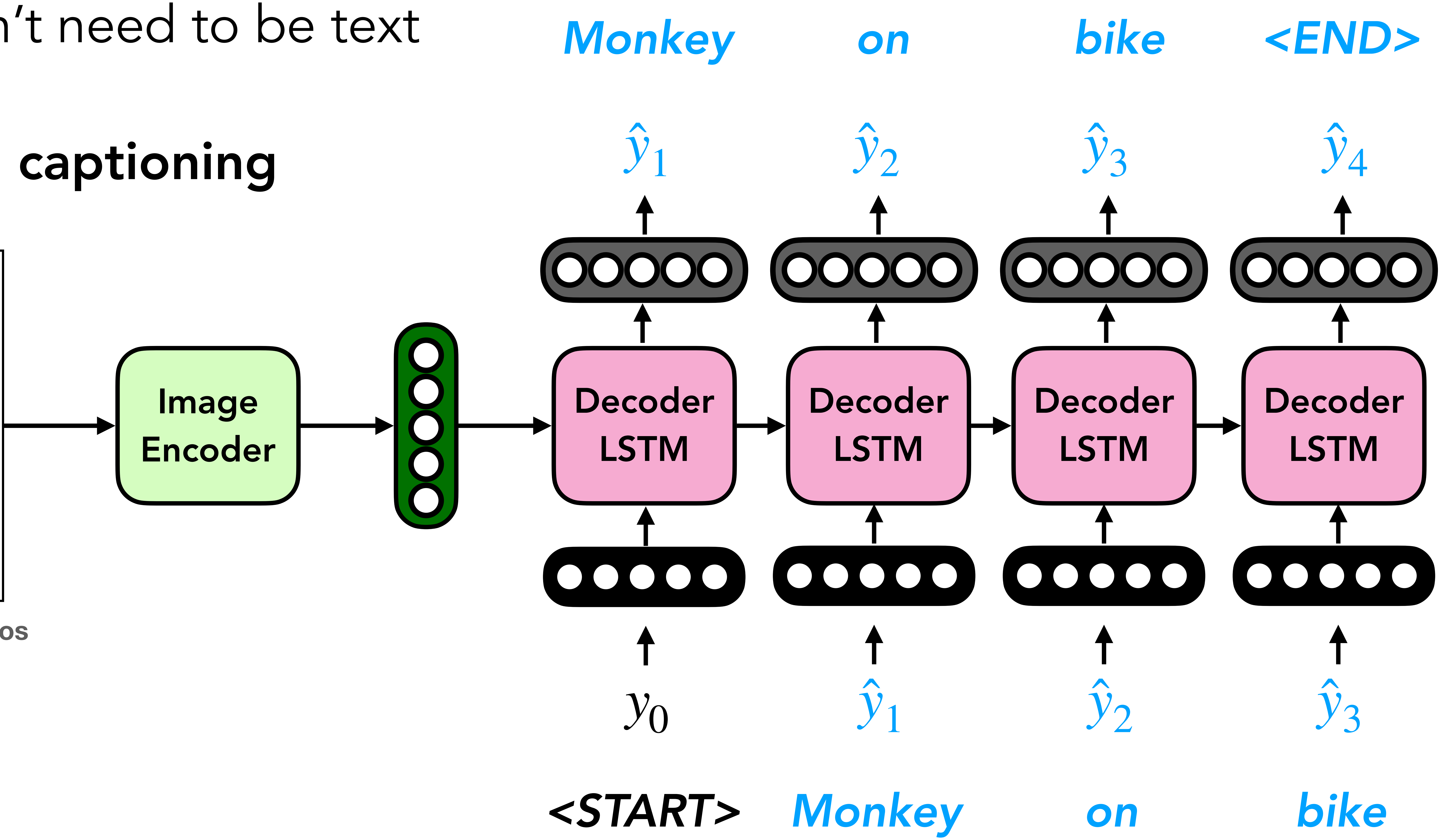
- e.g., **machine translation**



Encoder-Decoder Models

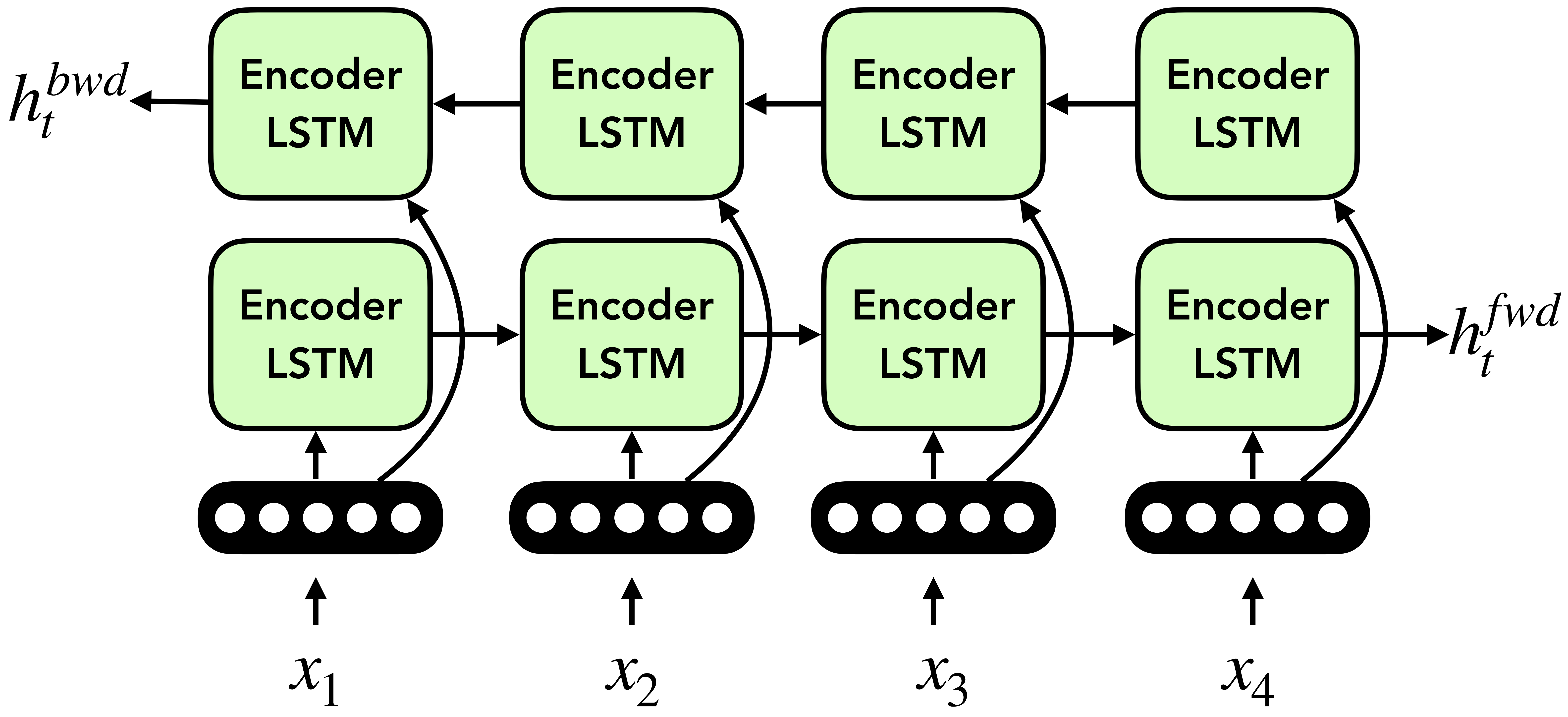
- Input doesn't need to be text
- e.g., **image captioning**

Photo credit: J Hovenstine Studios



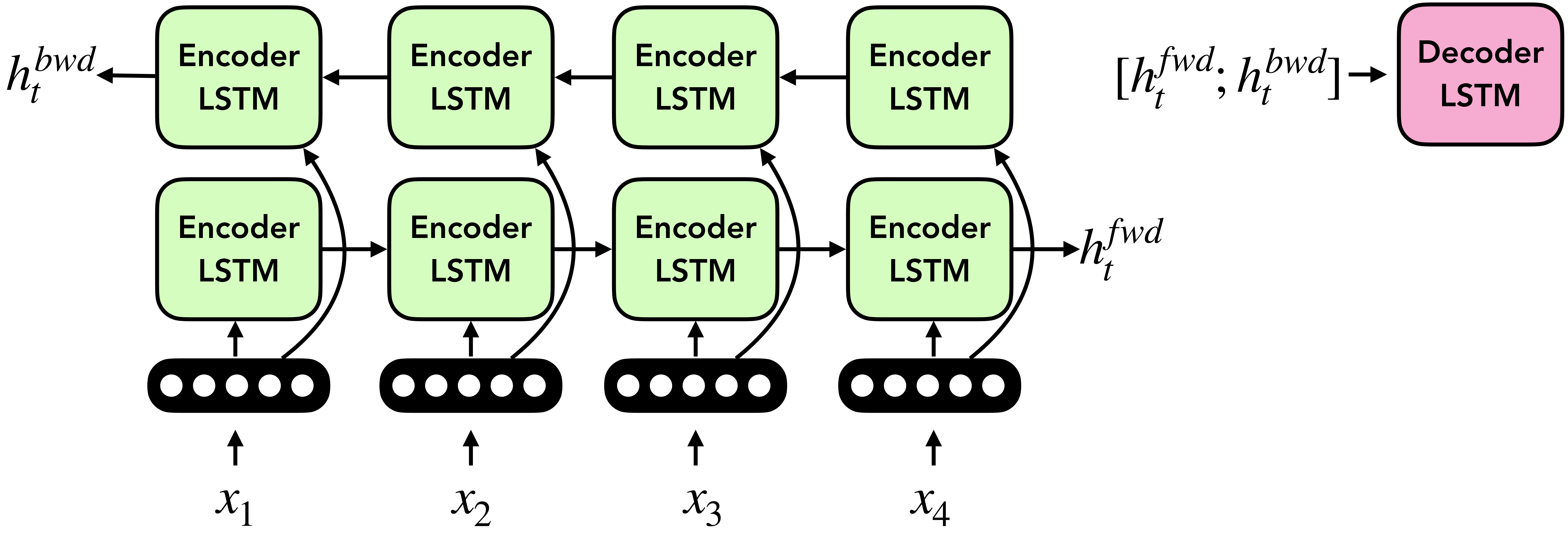
Bidirectionality

- Decoder needs to be unidirectional (can't know the future…)
- Encoder sequence representation augmented by encoding in both directions



Bidirectionality

- Decoder needs to be unidirectional (can't know the future…)
- Encoder sequence representation augmented by encoding in both directions



Other Resources of Interest

- Gated Recurrent Units (Cho et al., 2014):
  - Different approach for maintaining state and avoiding vanishing gradients
- LSTM: A Search Space Odyssey (Greff et al., 2015)
  - Examine 5000 different modifications to LSTMs — none significantly better than original architecture
- Only basics presented here today! Many offshoots of these techniques!

Part 3: Attentive Neural Modeling with Transformers

Section Outline

- **Background**: Long-term Dependency Modeling
- **Content:** Attention, Self-Attention, Multi-headed Attention, Transformer Blocks, Transformers
- **Demo:** Visualizing Transformer Attention

Issue with Recurrent Models

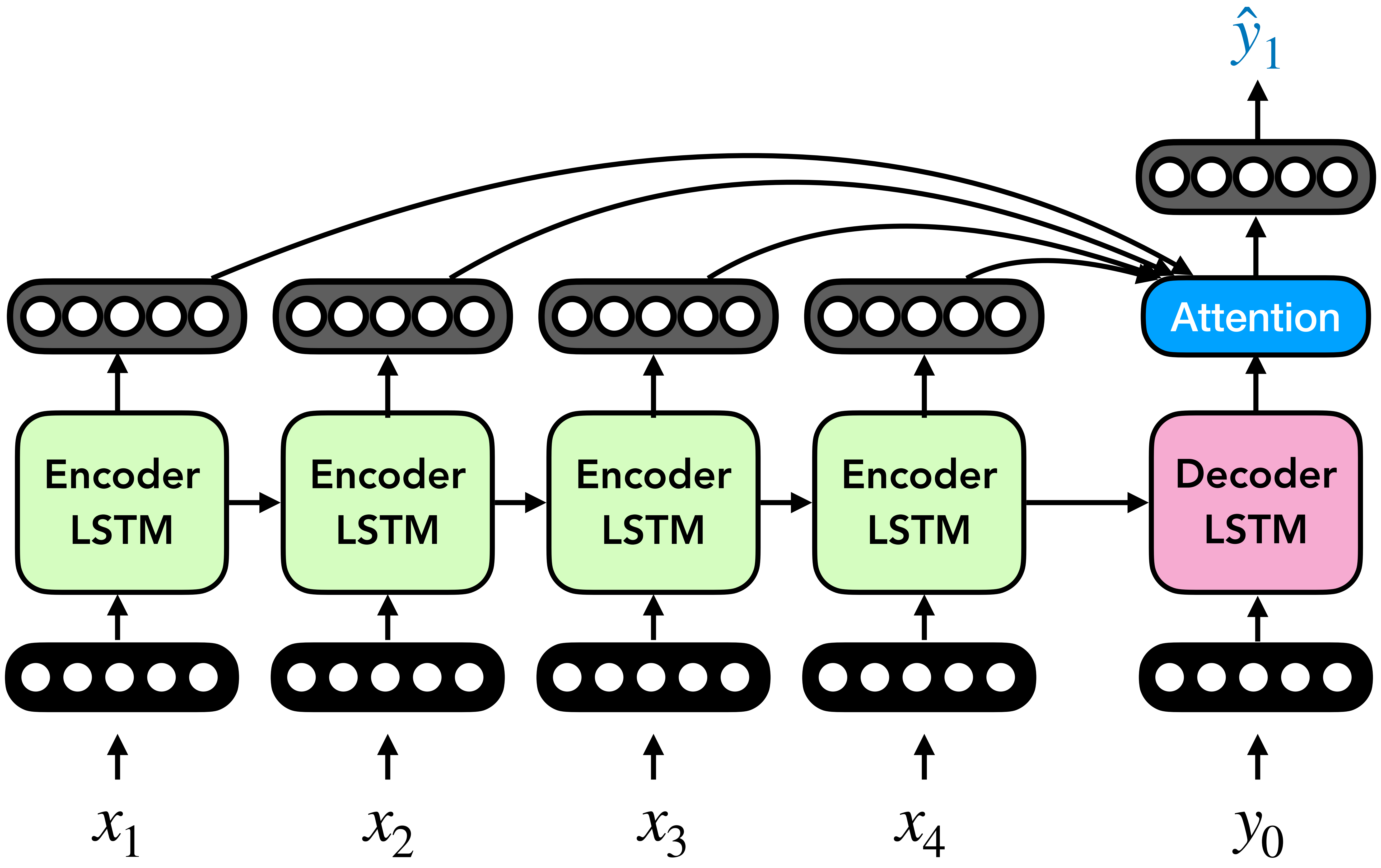
- Multiple steps of state overwriting makes it challenging to learn long-range dependencies.

They tuned, discussed for a moment, then struck up a lively jig. Everyone joined in, turning the courtyard into an even more chaotic scene, people now dancing in circles, swinging and spinning in circles, everyone making up their own dance steps. I felt my feet tapping, my body wanting to move. Aside from writing, I 've always loved **dancing** *.*

- Nearby words should affect each other more than farther ones, but RNNs make it challenging to learn **any** long-range interactions

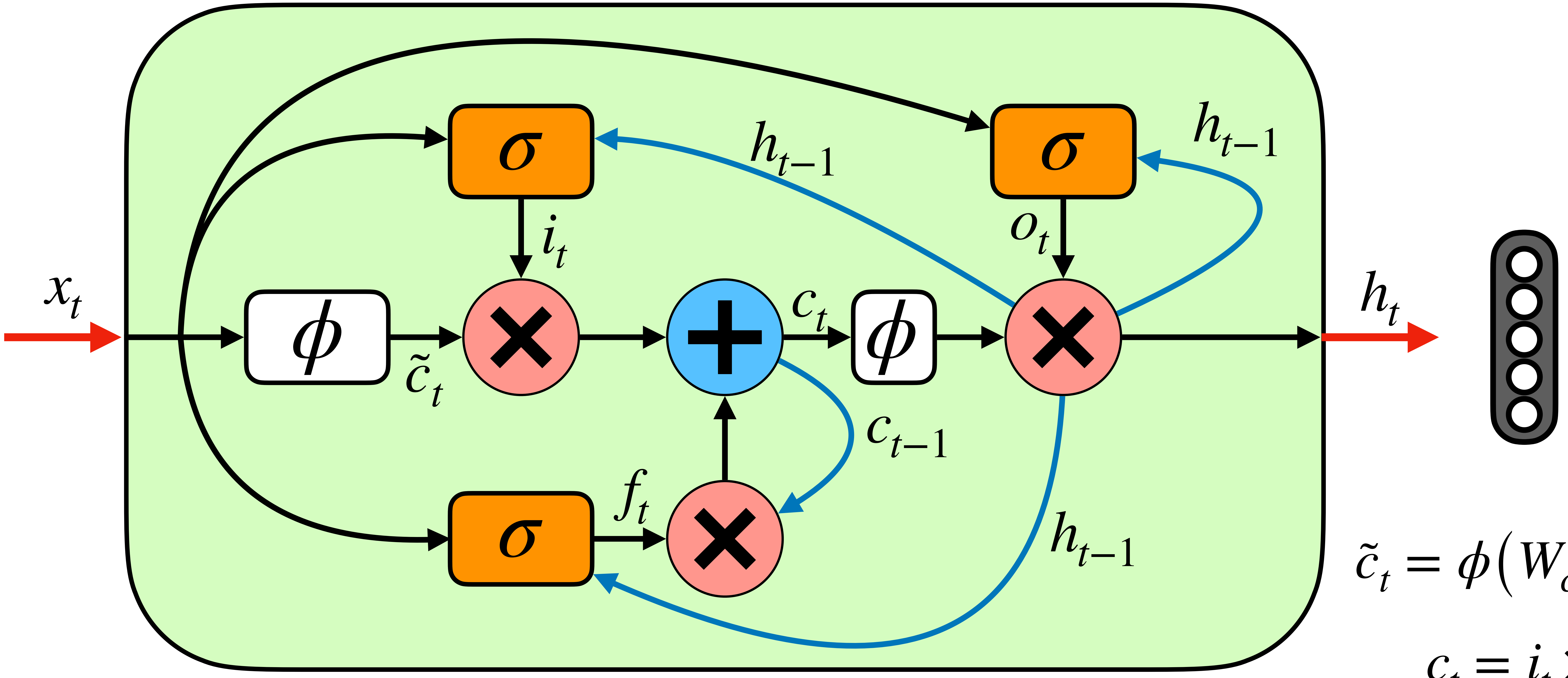
LAMBADA dataset, 2016

Attentive Encoder-Decoder Models



- **Idea:** Use the output of the Decoder LSTM to compute an **attention** over all the outputs of the encoder LSTM
- Attention is a weighted average over a set
- **Question:** what setting might this be useful in?

Review: LSTMs



$$\tilde{c}_t = \phi\left(W_{cx}x_t + W_{ch}h_{t-1} + b_c\right)$$

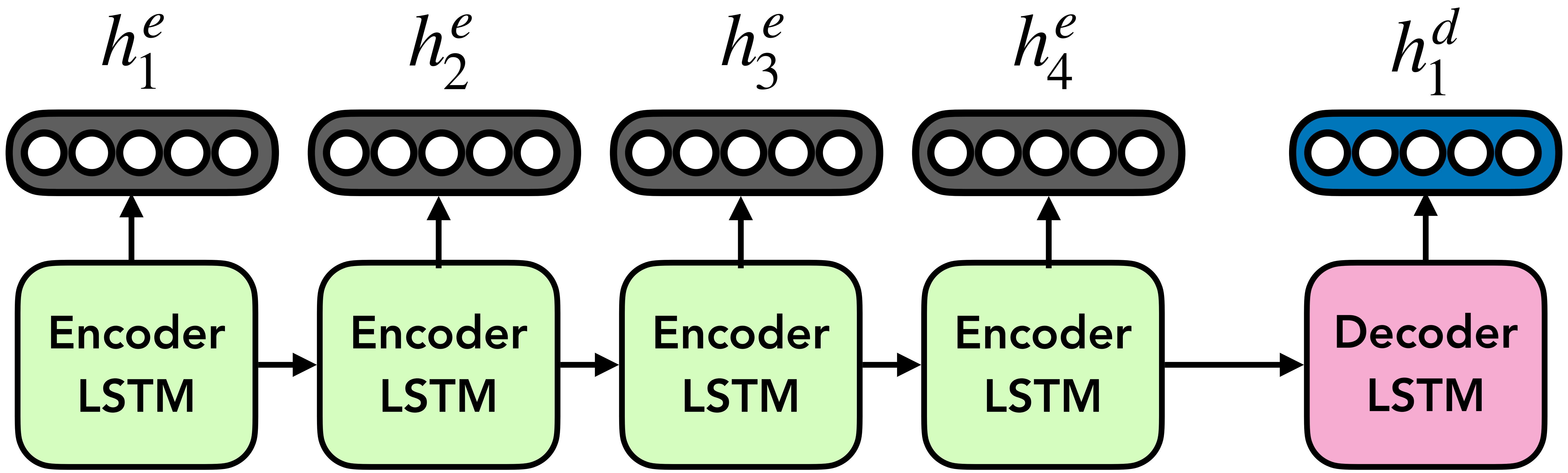
$$c_t = i_t \times \tilde{c}_t + f_t \times c_{t-1}$$

$$h_t = o_t \times \phi(c_t)$$

Attention Function

- Set output of decoder as weighted sum of encoder outputs
- Compute similarity between decoder hidden state and encoder output states

h_t^e = encoder output hidden states $\qquad h_t^d$ = decoder output hidden states

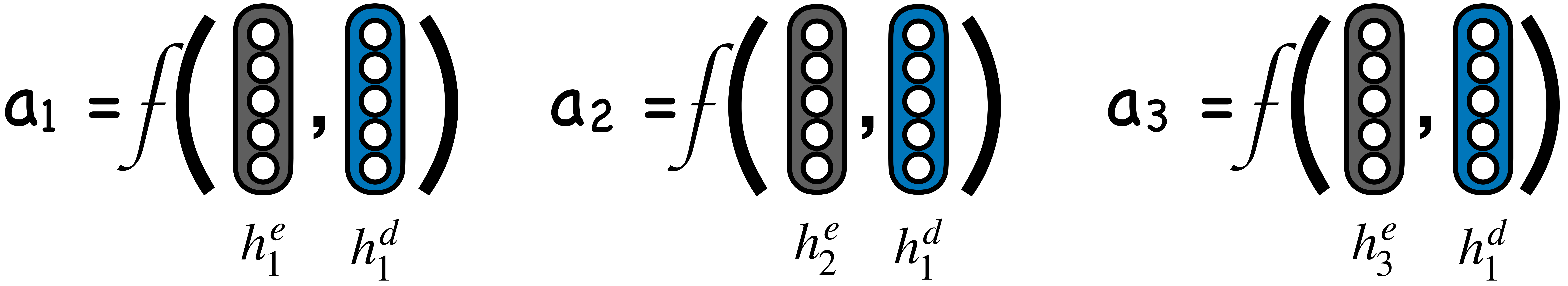


Attention Function

- Compute similarity between decoder hidden state and encoder output states

h_t^e = encoder output hidden states $\qquad$ h_t^d = decoder output hidden state

- Compute pairwise score between each encoder hidden state and decoder hidden state

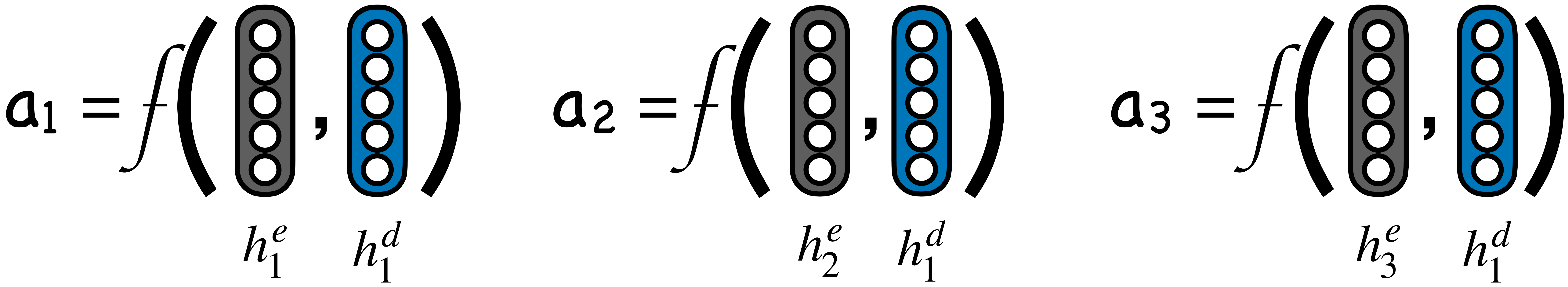


Attention Formulas

Attention Function	Formula
Bilinear	$a = h^e \mathbf{W} h^d$
Concatenation	$a = v^T \phi(\mathbf{W}[h^e; h^d])$
Dot Product	$a = h^e \cdot h^d$
Scaled Dot Product	$a = \dfrac{(\mathbf{W}h^e)^T(\mathbf{U}h^d)}{\sqrt{d}}$

Attention Function

- Compute pairwise score between each encoder hidden state and decoder hidden state

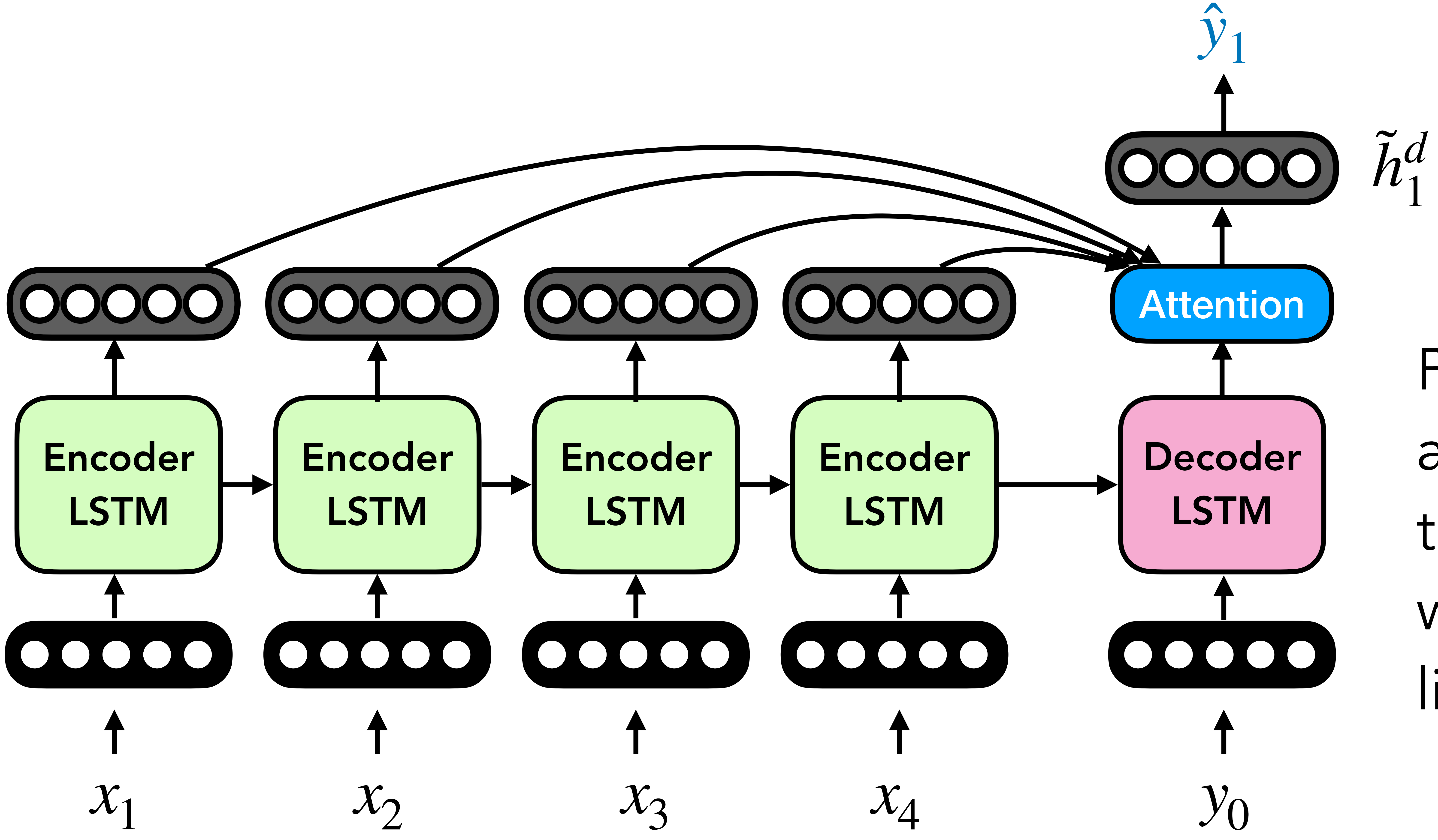


- Convert scores to distribution over encoder hidden states and computed weighted average:

Softmax! $\boxed{\alpha_t = \frac{e^{a_t}}{\sum_j e^{a_j}}}$

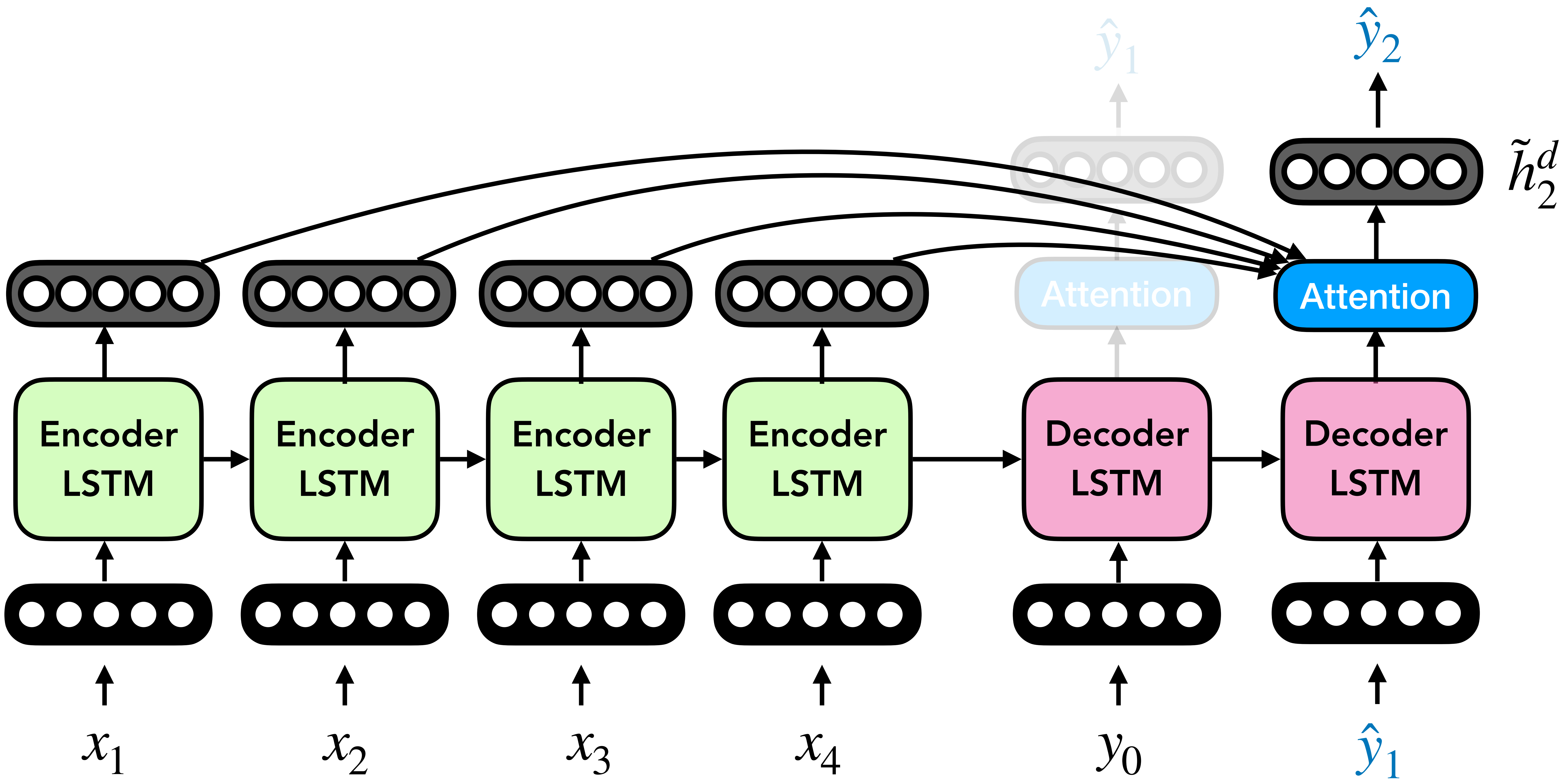
$$\tilde{h}_1^d = \sum_{t=1}^{T} \alpha_t h_t^e$$

Attentive Encoder-Decoder Models

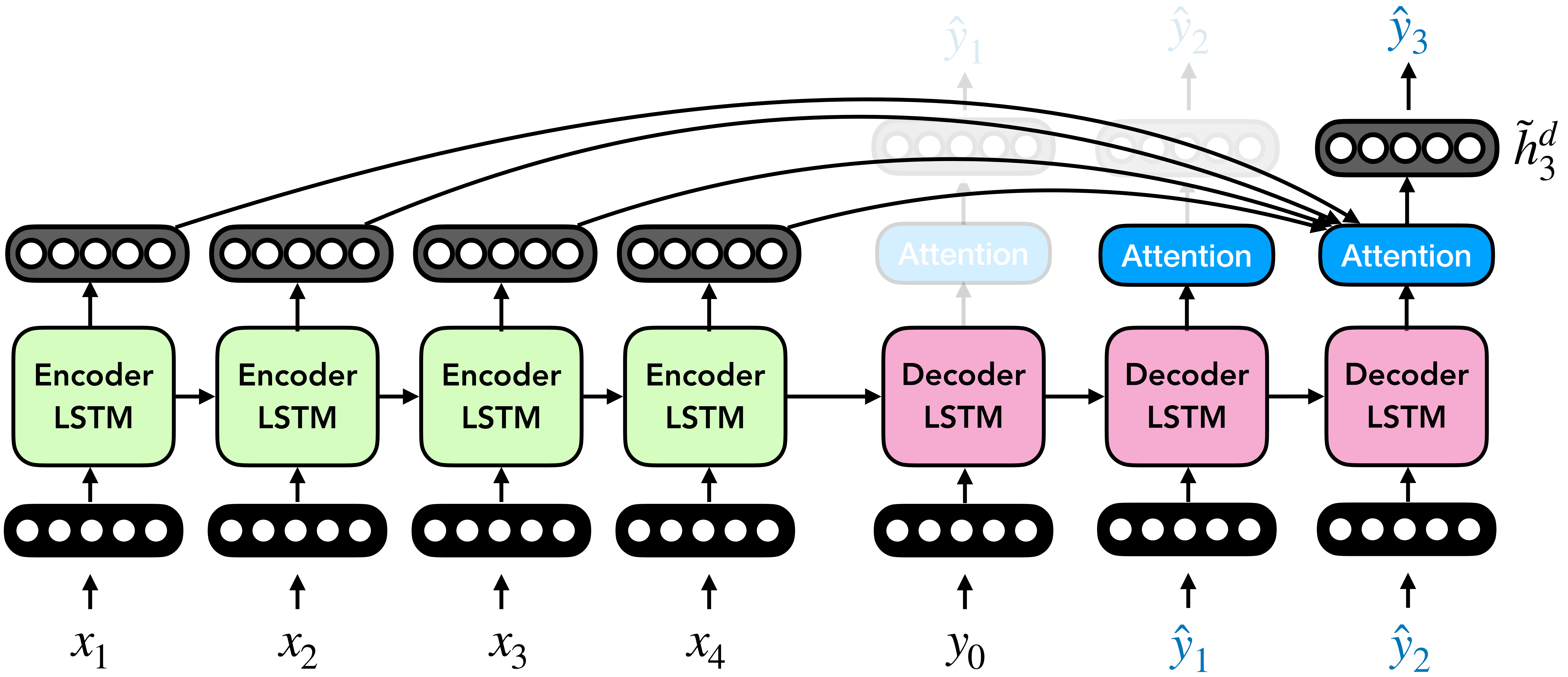


Pass the output of the attention layer $\tilde{h}_1^d$ to your output layer, which predicts the most likely output token $\hat{y}_1$

Attentive Encoder-Decoder Models



Attentive Encoder-Decoder Models



Attention Recap

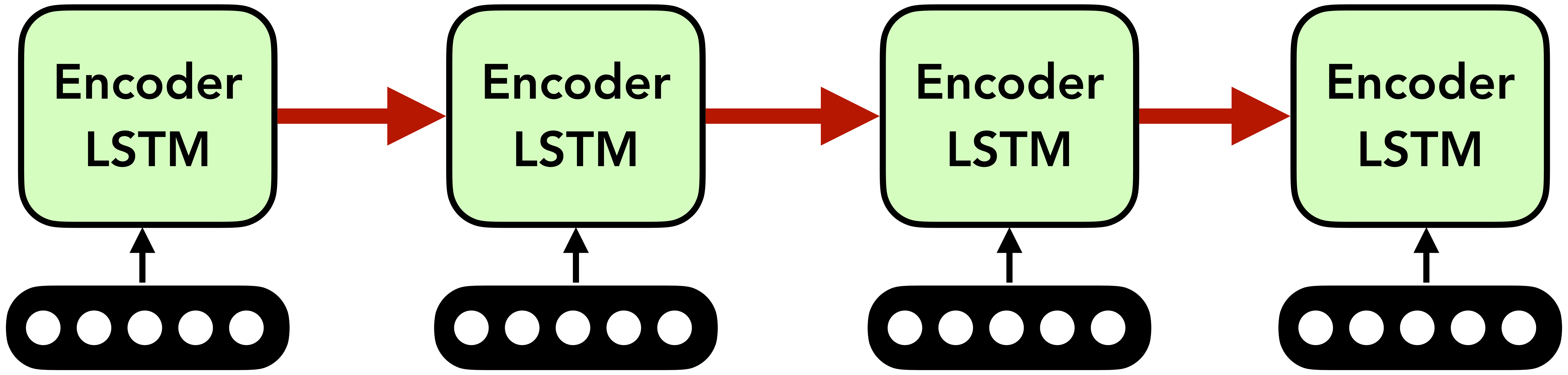
- Compute new output of decoder as weighted sum of encoder outputs
- Compute pairwise score between each encoder hidden state and decoder hidden state

h_t^e = encoder output hidden states $\qquad$ h_t^d = decoder output hidden state

- Many possible functions for computing scores (dot product, bilinear, etc.)
- Allows for direct connection between decoder and <u>**ALL**</u> encoder states

Issue with Recurrent Models

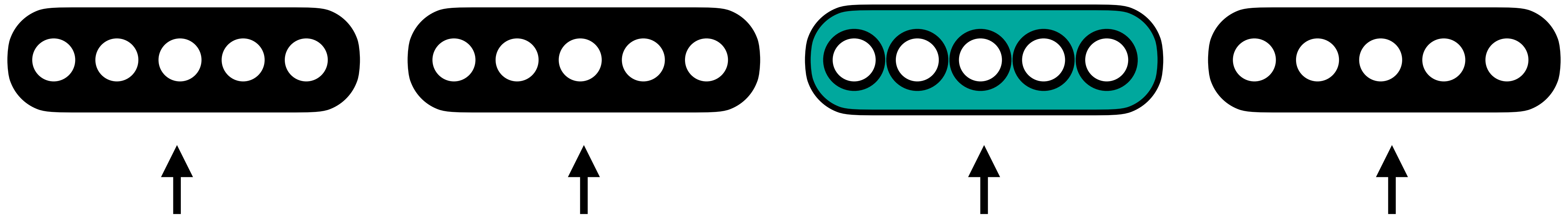
- Recurrent functions can't be parallelized because previous state needs to be computed to encode next one



Self-Attention

- Ditch recurrence and compute encoder state representations in parallel!
- Compute pairwise score between each encoder hidden state and the other encoder hidden states

h_t^ℓ = encoder hidden state at time step t at layer ℓ

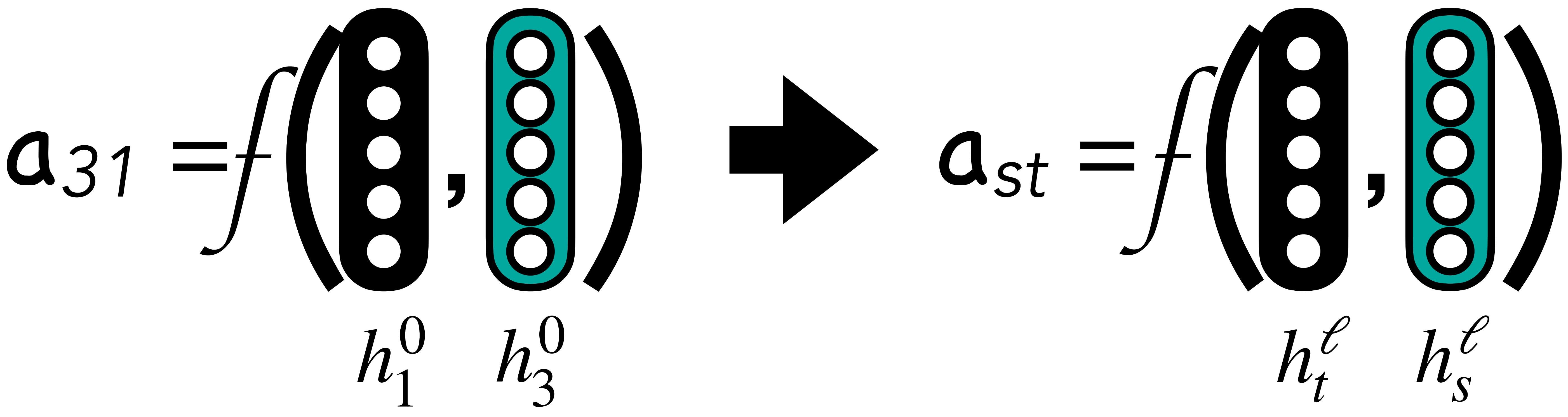


h_1^0 h_2^0 h_3^0 h_4^0

Self-Attention

- Compute pairwise score between each encoder hidden state and the other encoder hidden states

h_t^ℓ = encoder hidden state at time step t at layer ℓ



$$a_{st} = \frac{(\mathbf{W}h_s^\ell)^T(\mathbf{U}h_t^\ell)}{\sqrt{d}} \qquad \alpha_{st} = \frac{e^{a_{st}}}{\sum_j e^{a_{sj}}} \qquad \tilde{h}_s^\ell = \sum_{t=1}^{T} \alpha_{st}\mathbf{V}h_t^\ell$$

{1, …, t, …, T} includes s!

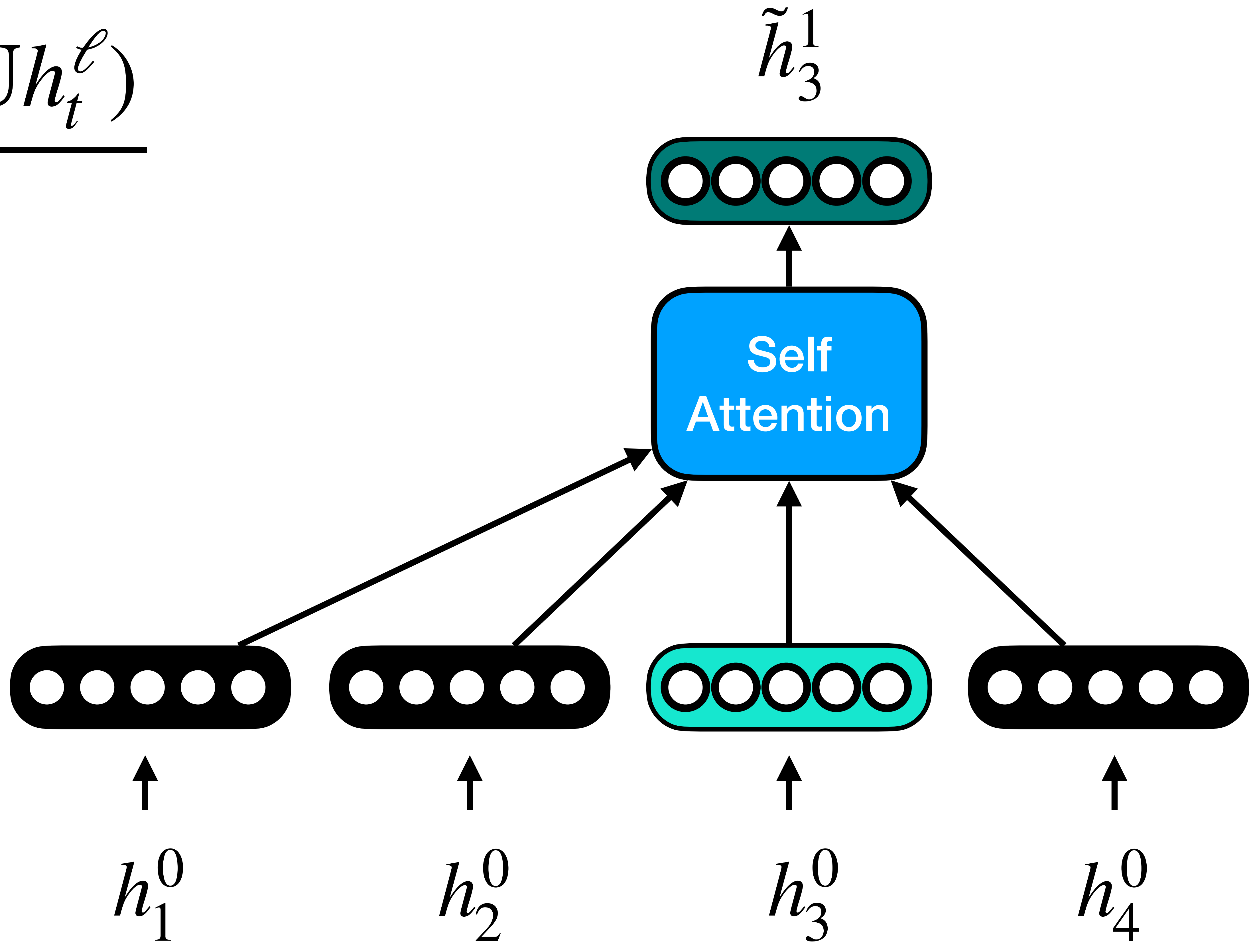
Self-attention!

Self-Attention

- Essentially, re-compute representation of state at every time step t using a weighted average of the representations of the other time steps

$$a_{st} = \frac{(\mathbf{W}h_s^\ell)^T(\mathbf{U}h_t^\ell)}{\sqrt{d}}$$

$$\alpha_{st} = \frac{e^{a_{st}}}{\sum_j e^{a_{sj}}}$$



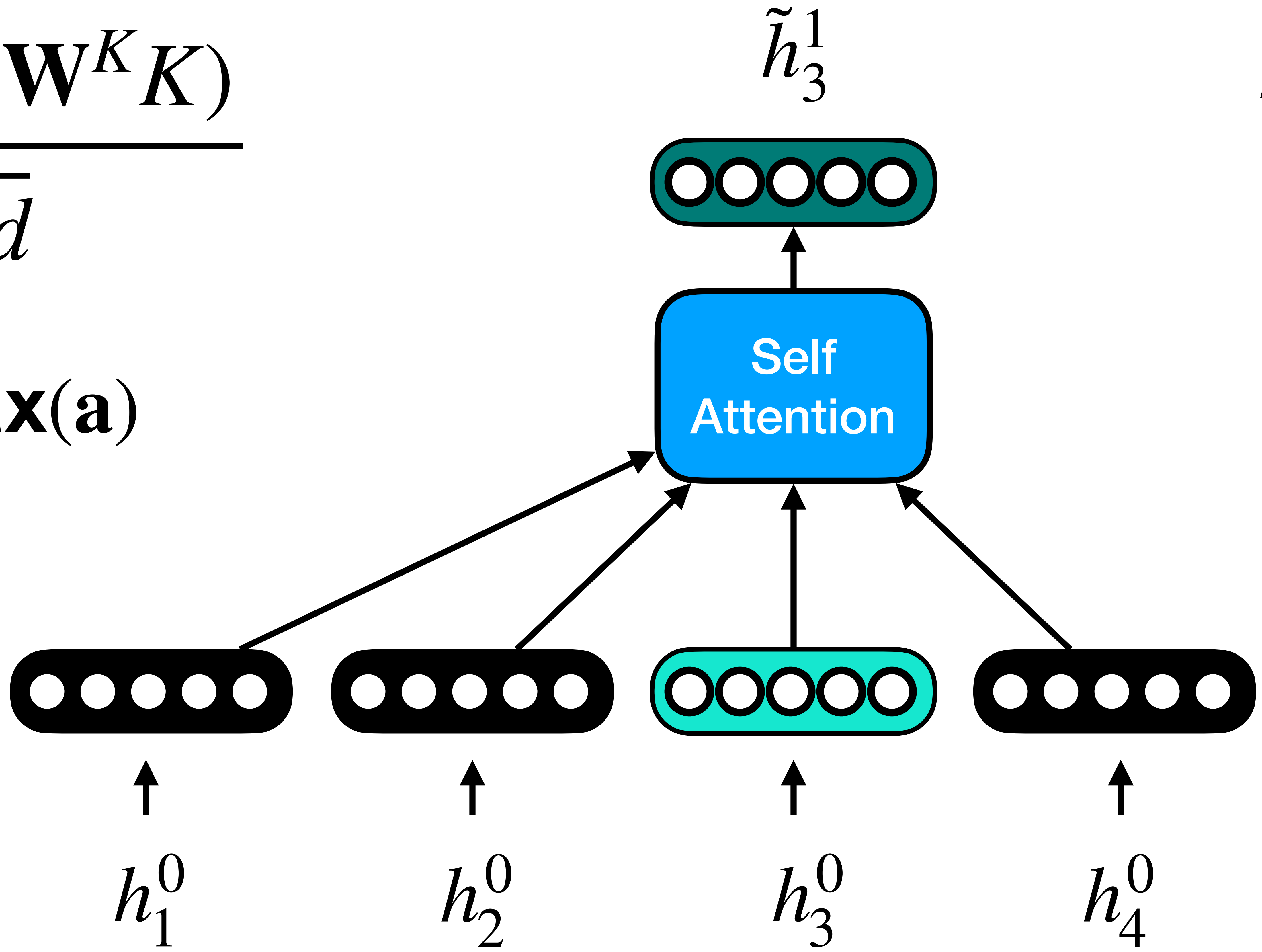
$$\tilde{h}_s^\ell = \sum_{t=1}^{T} \alpha_{st} \mathbf{V} h_t^\ell$$

Self-Attention

- Used same notation as before for consistency, but actual notation for self-attention in transformers use query (Q), keys (K), values (V):

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

$$\alpha = \textbf{softmax}(\mathbf{a})$$



$$\tilde{h}^\ell = W^O \alpha V \mathbf{W}^V$$

$$Q = h_s^\ell$$

$$K = V = \{h_t^\ell\}_{t=0}^T$$

Multi-Headed Self-Attention

- Project V, K, Q into H sub-vectors where H is the number of "heads"

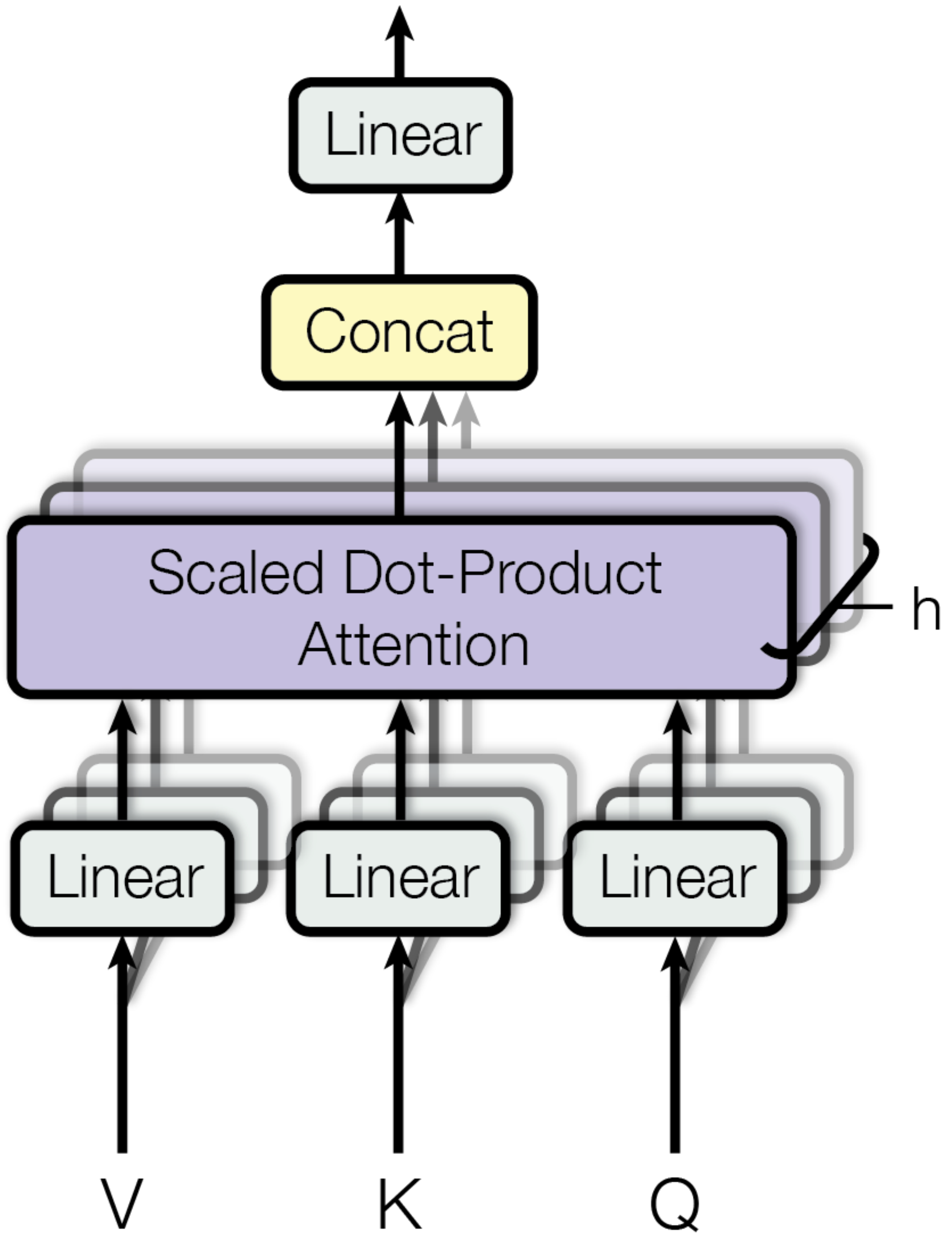
$$\mathbf{a}_i = \frac{(\mathbf{W}_i^Q Q)(\mathbf{W}_i^K K)}{\sqrt{d/H}}$$

- Compute attention weights separately for each sub-vector

$$\alpha_i = \textbf{softmax}(\mathbf{a}_i) \qquad \tilde{h}_i^\ell = \alpha V \mathbf{W}_i^V$$

- Concatenate sub-vectors for each head

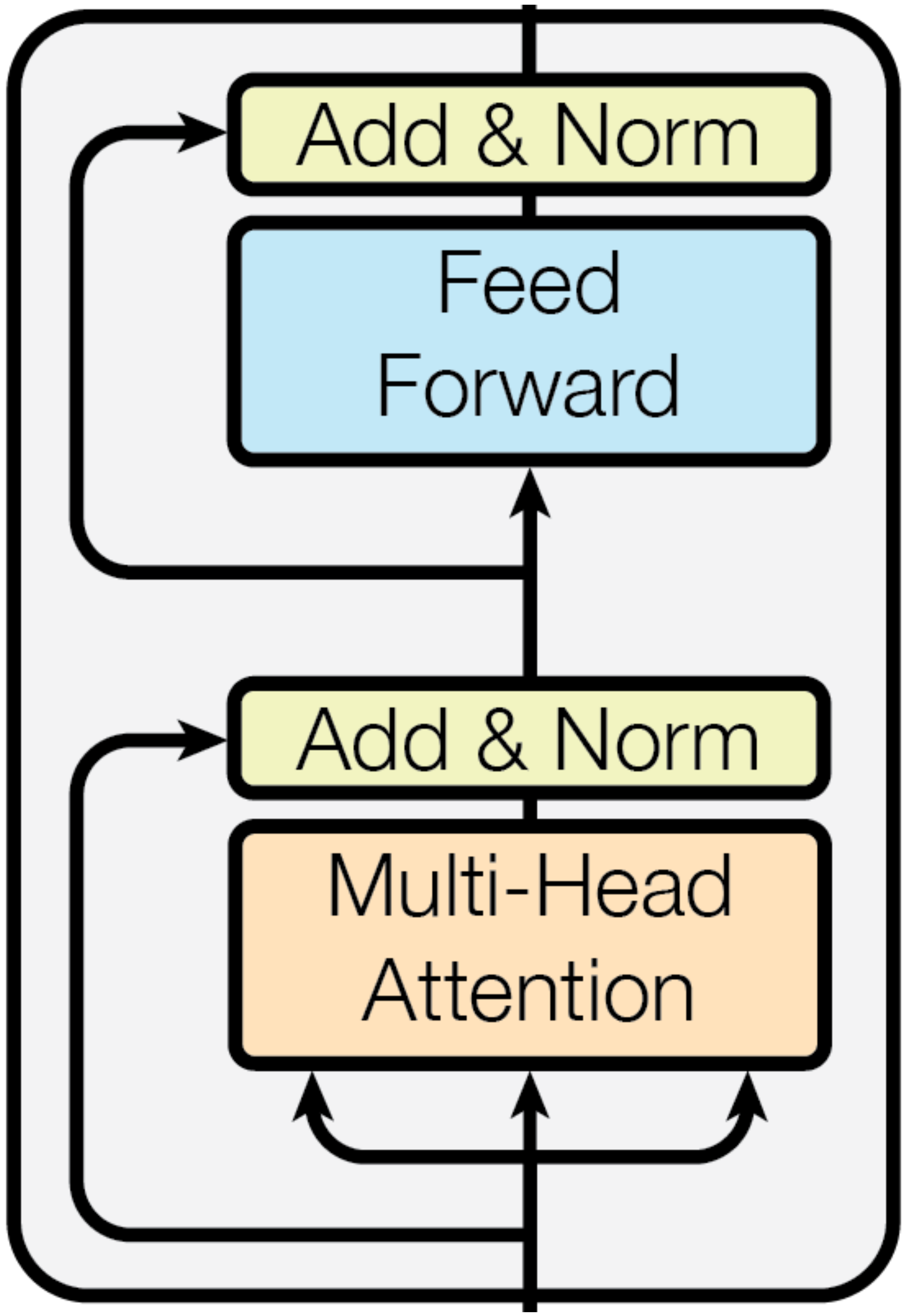
$$\tilde{h}^\ell = W^O[\ \tilde{h}_0^\ell;\ \dots;\tilde{h}_i^\ell;\ \dots;\tilde{h}_H^\ell\ ]$$



Vaswani et al., 2017

Transformer Block

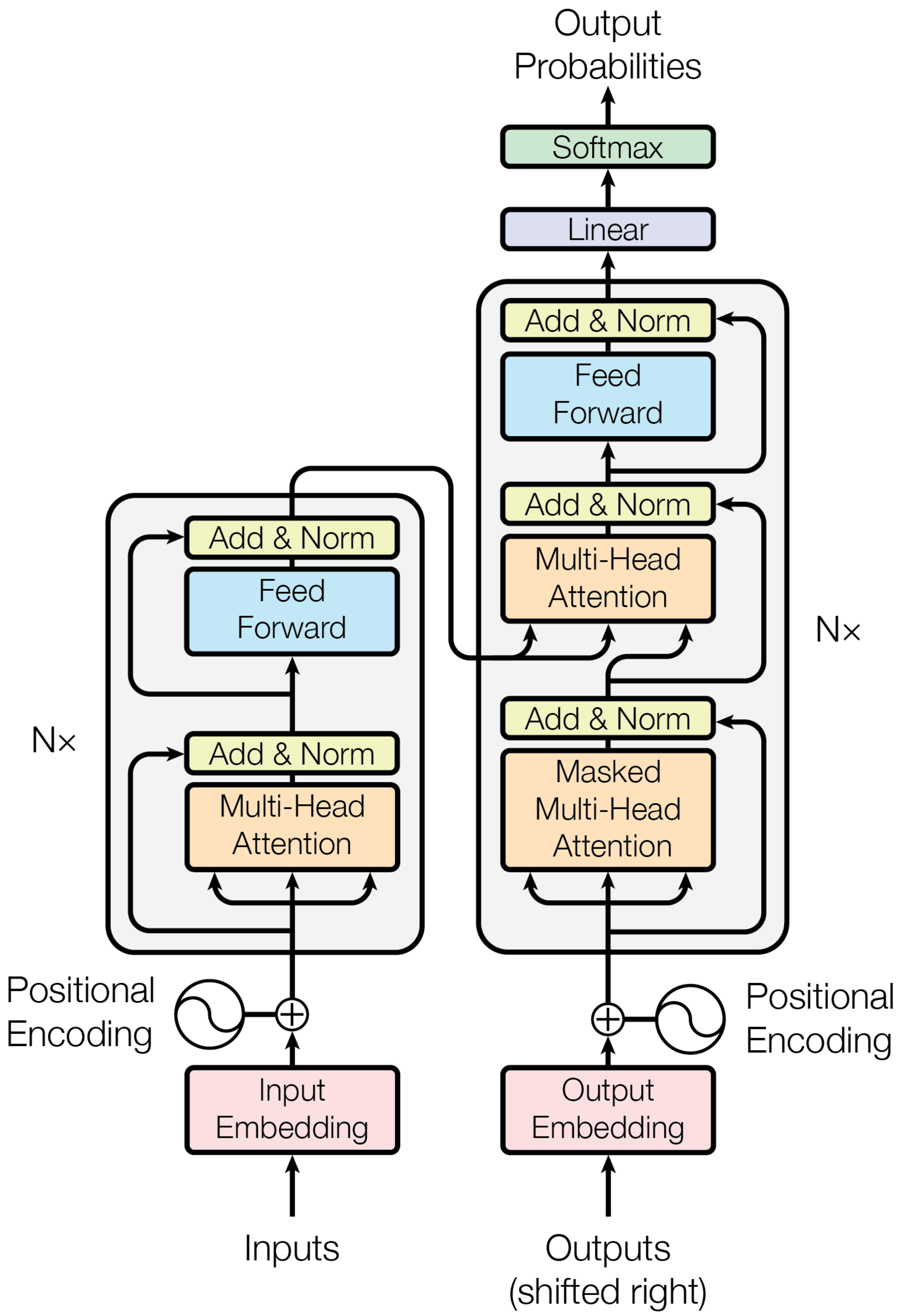
- Self-attention is the main innovation of the popular **transformer** model!
- Each transformer block receives as input the outputs of the previous layer at every time step
- Each block is composed of a multi-headed attention, a layer normalisation, a feedforward network, and another layer normalisation
- There are residual connections before every normalisation layer
- Layer normalisation + residual connections don't add capacity, but make training easier



Vaswani et al., 2017

Full Transformer

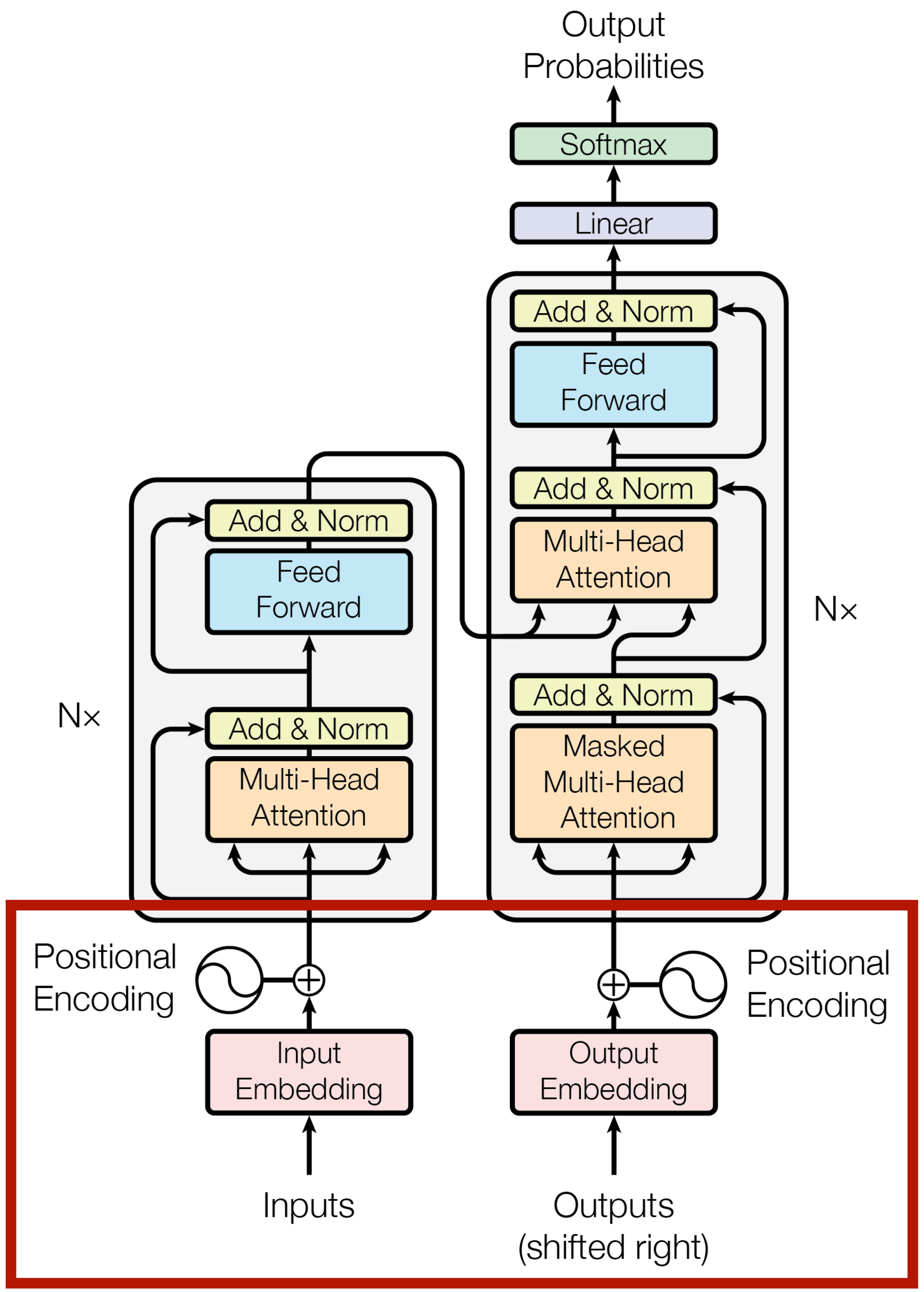
- Full transformer encoder is multiple cascaded transformer blocks
  - build up compositional representations of inputs
- No need to propagate state forward in time
  - states at each time step computed in parallel!
- Transformer decoder (right) similar to encoder
  - second attention layer to compute weighted average of encoder states before FFN



Vaswani et al., 2017

Full Transformer

- Full transformer encoder is multiple cascaded transformer blocks
  - build up compositional representations of inputs
- No need to propagate state forward in time
  - states at each time step computed in parallel!
- Transformer decoder (right) similar to encoder
  - second attention layer to compute weighted average of encoder states before FFN



Vaswani et al., 2017

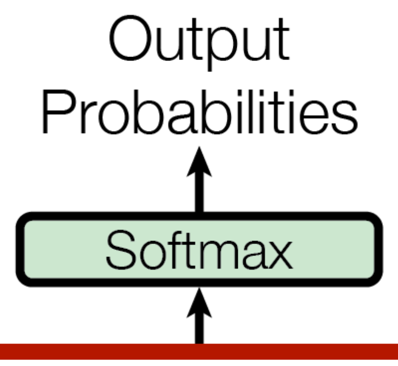
Full Transformer

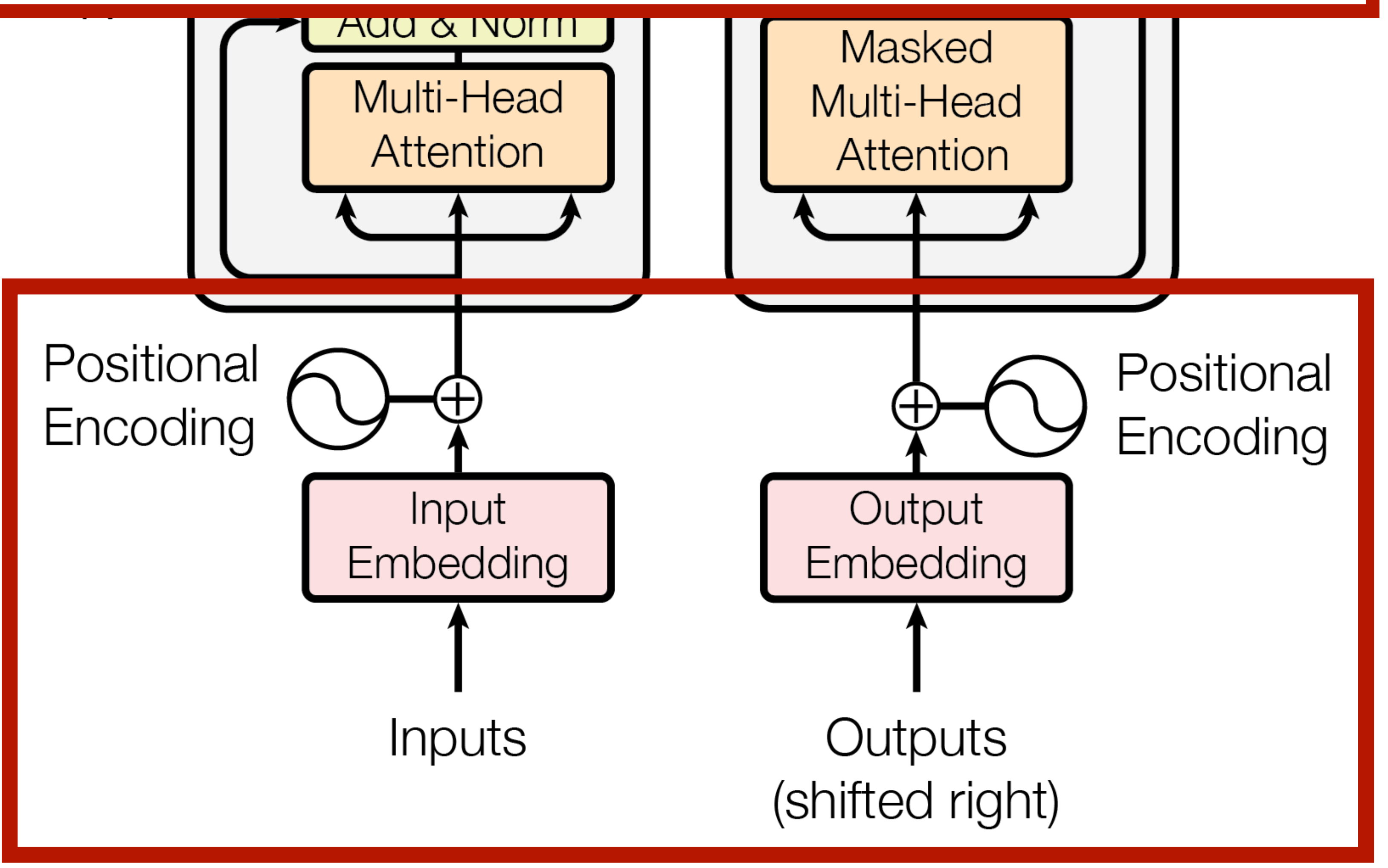
- Full transformer encoder is multiple cascaded
  tr
  -
- N
  - states at each time step computed in parallel

Recurrent models provided word order information

Does self-attention provide word order information?

- Transformer decoder (right) similar to encoder
  - second attention layer to compute weighted average of encoder states before FFN

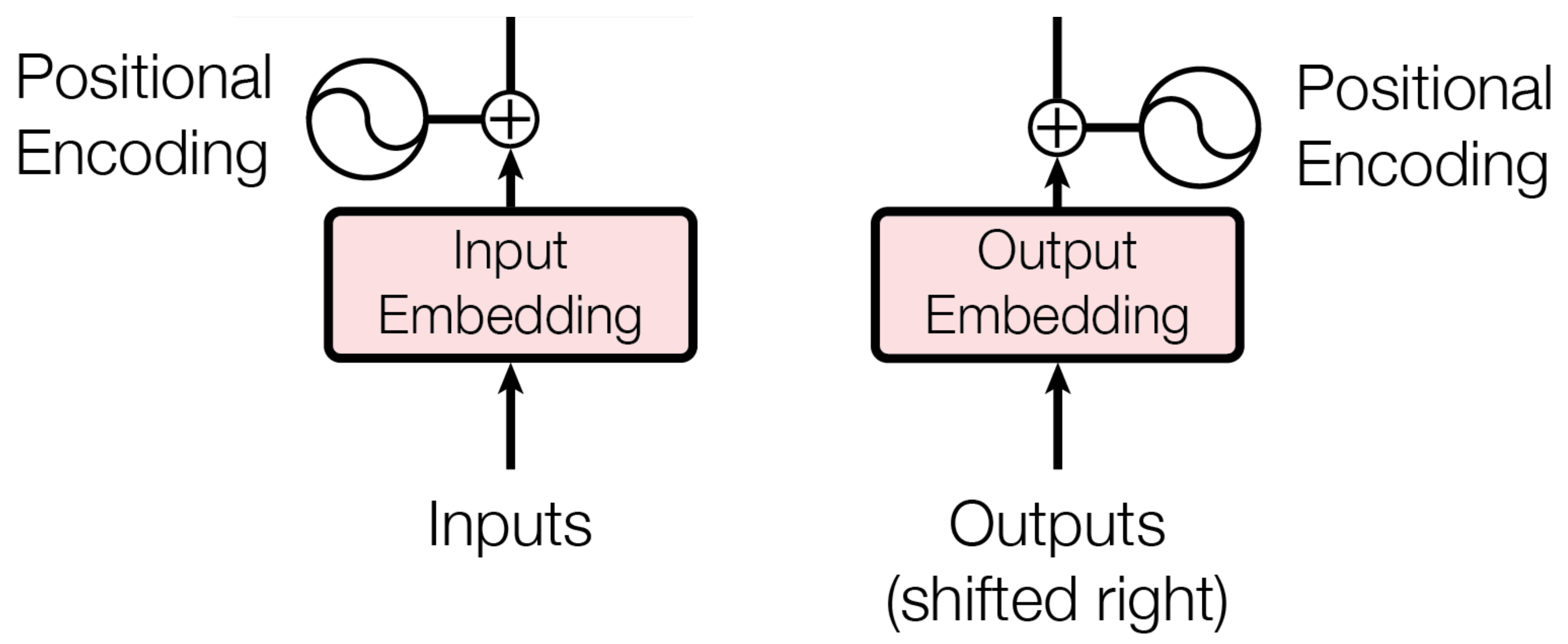




Vaswani et al., 2017

Position Embeddings

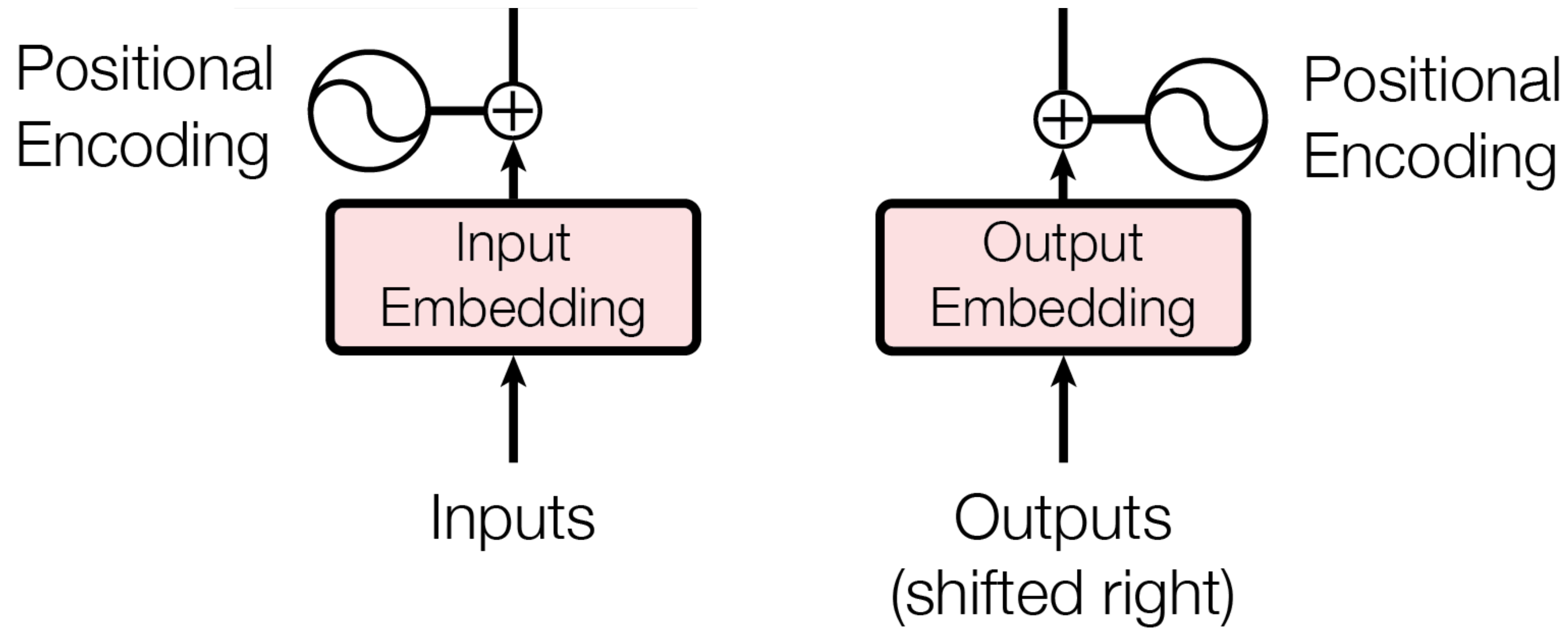
- Self-attention provides no word order information
    - Computes weighted average over set of vectors
- Word order is pretty crucial to understanding language
    - How do we fix this?
- Add an additional embedding to the input word that represents a position in the sequence



Vaswani et al., 2017

Position Embeddings

- Self-attention provides no word order information
  - Computes weighted average over set of vectors
- Word order is pretty crucial to understanding language
  - How do we fix this?
- Add an additional embedding to the input word that represents a position in the sequence



- Early position embeddings encoded a sinusoid function that was offset by a phase shift proportional to sequence position
- **In practice, everyone nowadays learns position embeddings from scratch**

Vaswani et al., 2017

Other Resources of Interest

- The Annotated Transformer
  - https://nlp.seas.harvard.edu/2018/04/03/attention.html
- The Illustrated Transformer
  - https://jalammar.github.io/illustrated-transformer/
- Only basics presented here today! Many modifications to initial transformers exist

Demo: Attention Visualization

https://colab.research.google.com/drive/1PEHWRHrvxQvYr9NFRC-E_fr3xDq1htCj

Part 4: Modern NLP
Where do we go from here?

Section Outline

- **Advances:** NLP Successes, Pretraining, Scale
- **New Problems:** Robustness, Multimodality, Knowledge, Prompting, Ethics
- **Demo:** Write with Transformers!

Deep Learning Successes in NLP

The New York Times

FEATURE

The Great A.I. Awakening

How Google used artificial in
Translate, one of its more po
machine learning is poised to

The New York Times

Finally, a Machine That Can Finish Your Sentence

se's thought is not an easy trick for A.I.
starting to crack the code of natural
language.

THE NEW YORKER

The Next Word

Where will pred

Text by Jo

Vox

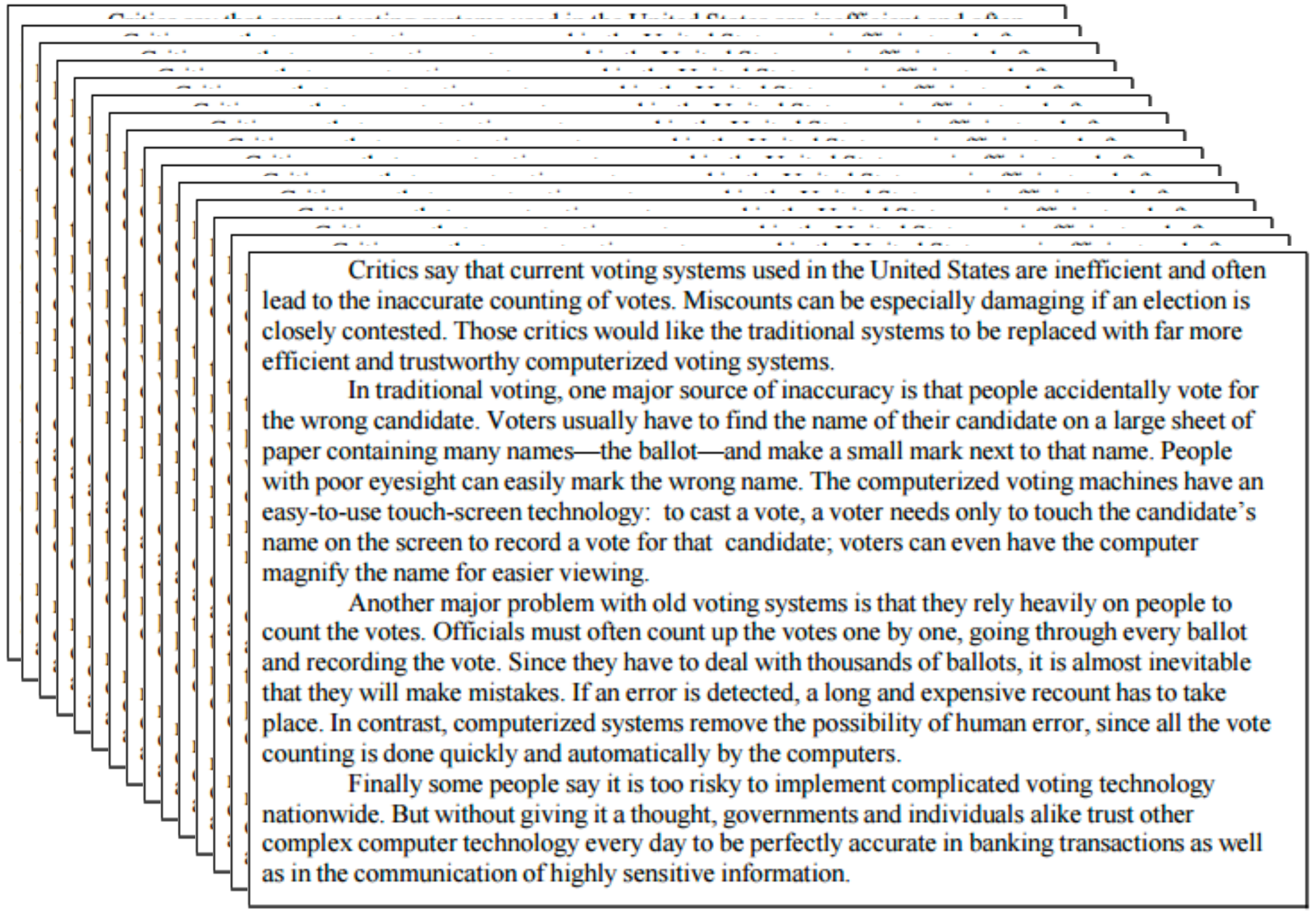
How I'm using AI to write my next novel

The New York Times

A Breakthrough for A.I. Technology: Passing an 8th-Grade Science Test

Pretraining

Massive Text Corpus


Critics say that current voting systems used in the United States are inefficient and often lead to the inaccurate counting of votes. Miscounts can be especially damaging if an election is closely contested. Those critics would like the traditional systems to be replaced with far more efficient and trustworthy computerized voting systems.

In traditional voting, one major source of inaccuracy is that people accidentally vote for the wrong candidate. Voters usually have to find the name of their candidate on a large sheet of paper containing many names—the ballot—and make a small mark next to that name. People with poor eyesight can easily mark the wrong name. The computerized voting machines have an easy-to-use touch-screen technology: to cast a vote, a voter needs only to touch the candidate's name on the screen to record a vote for that candidate; voters can even have the computer magnify the name for easier viewing.

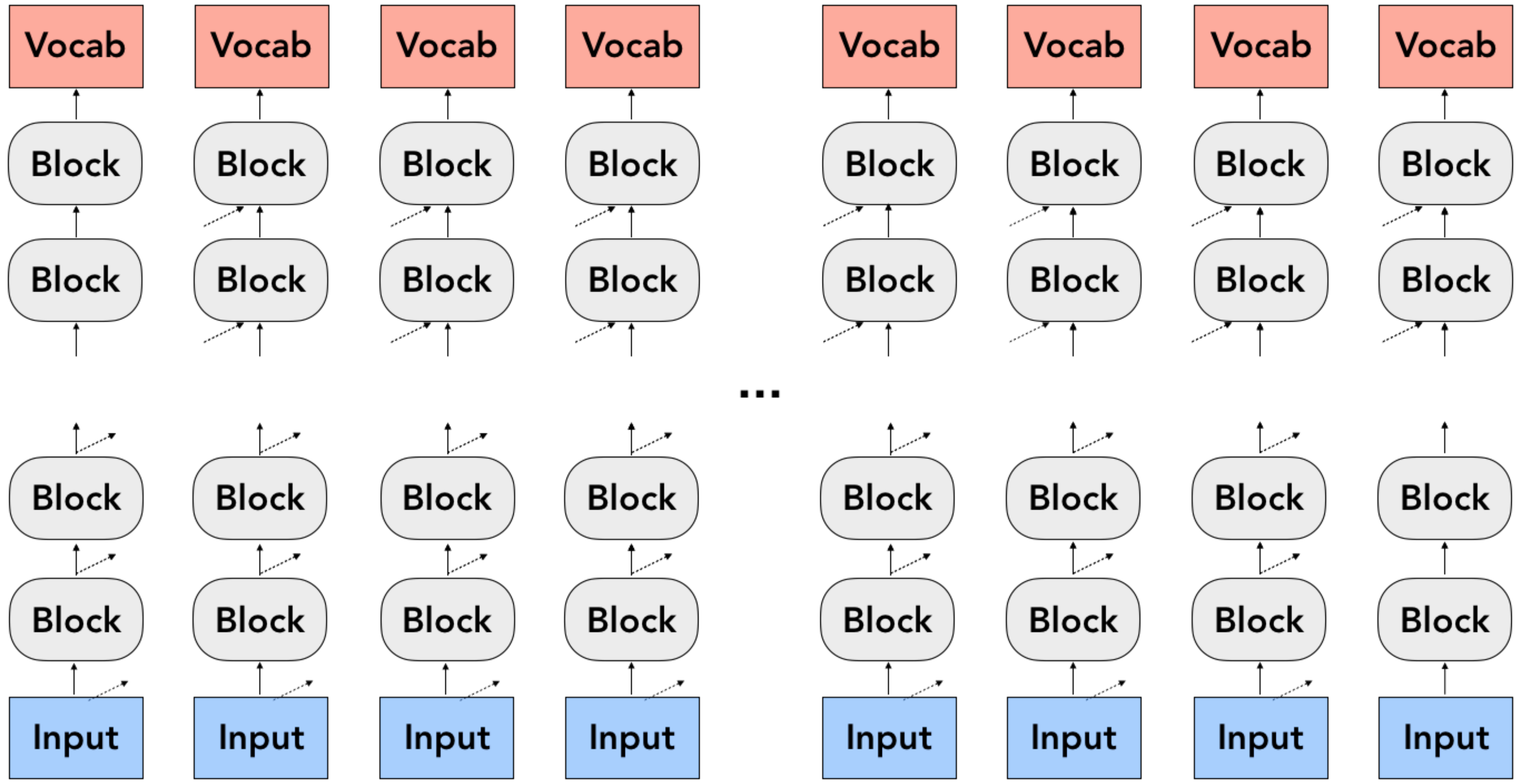
Another major problem with old voting systems is that they rely heavily on people to count the votes. Officials must often count up the votes one by one, going through every ballot and recording the vote. Since they have to deal with thousands of ballots, it is almost inevitable that they will make mistakes. If an error is detected, a long and expensive recount has to take place. In contrast, computerized systems remove the possibility of human error, since all the vote counting is done quickly and automatically by the computers.

Finally some people say it is too risky to implement complicated voting technology nationwide. But without giving it a thought, governments and individuals alike trust other complex computer technology every day to be perfectly accurate in banking transactions as well as in the communication of highly sensitive information.

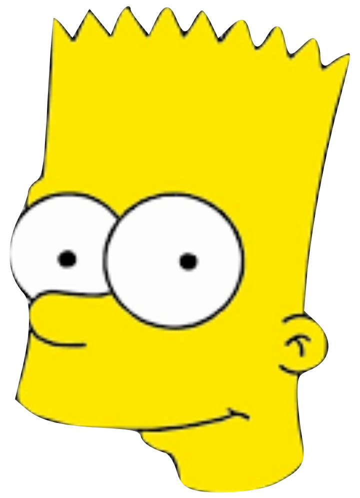
Used to

Learn

Transformer Language Model



OpenAI



(Radford et al., 2018, 2019, many others)

Pretraining: Two Approaches

(Causal, Left-to-right) Language Modeling

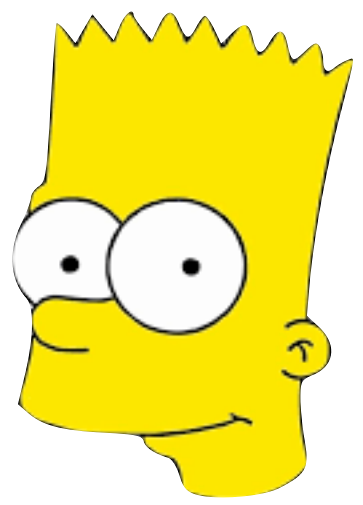
I really enjoyed the movie we watched on ____

OpenAI

(Radford et al., 2018, 2019, many others)

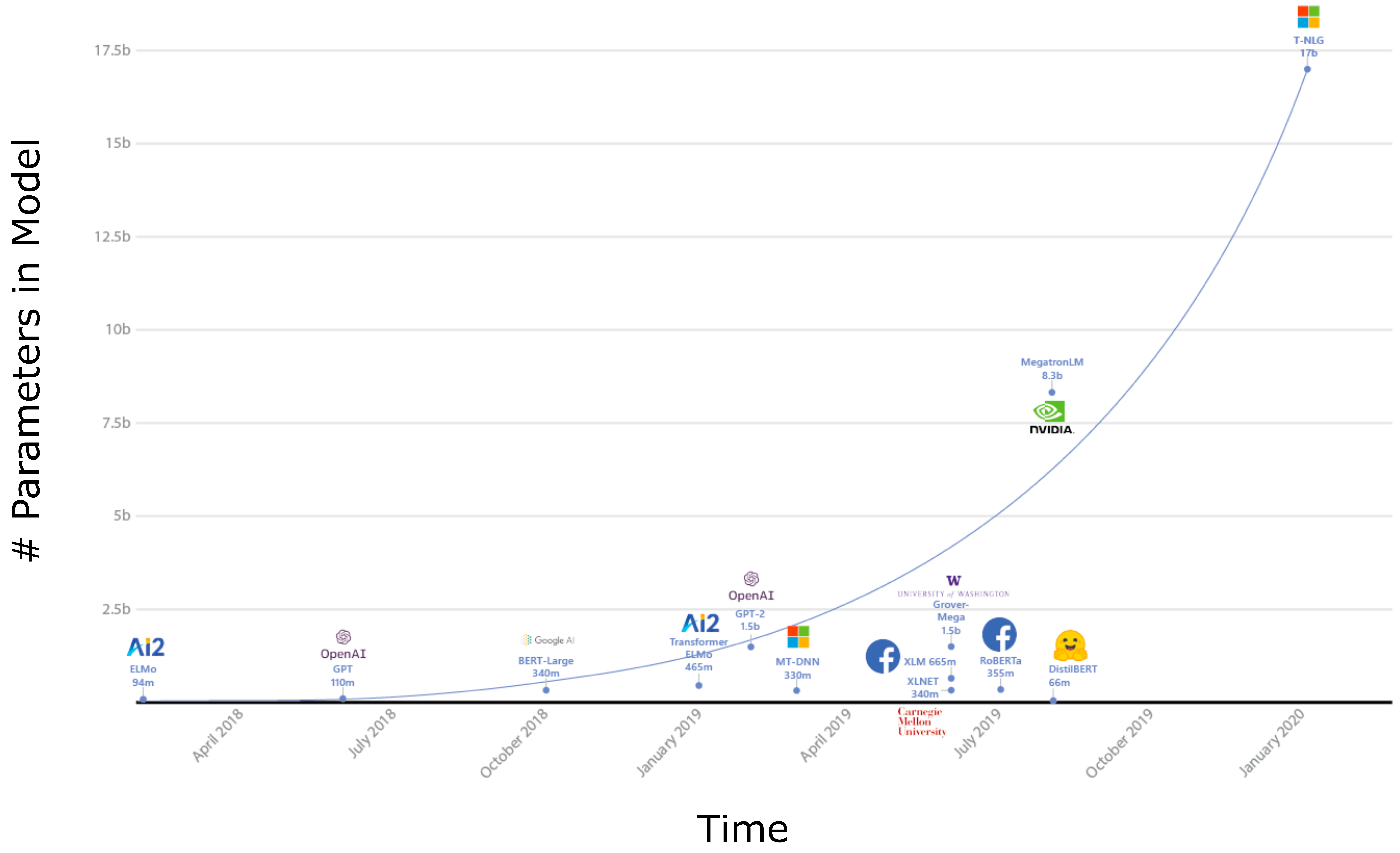
Masked Language Modeling

I really enjoyed the ____ we watched on Saturday!



(Devlin et al., 2018; Liu et al., 2020)

Scale



GPT3 - 175B (July 2020)

Results

Rank	Name	Model	URL	Score	BoolQ	CB	COPA	MultiRC	ReCoRD	RTE	WiC	WSC	AX-b	AX-g
1	Liam Fedus	SS-MoE		91.0	92.3	96.9/98.0	99.2	89.2/65.2	95.0/94.2	93.5	77.4	96.6	72.3	96.1/94.1
2	Microsoft Alexander v-team	Turing NLR v5		90.9	92.0	95.9/97.6	98.2	88.4/63.0	96.4/95.9	94.1	77.1	97.3	67.8	93.3/95.5
3	ERNIE Team - Baidu	ERNIE 3.0		90.6	91.0	98.6/99.2	97.4	88.6/63.2	94.7/94.2	92.6	77.4	97.3	68.6	92.7/94.7
+ 4	Zirui Wang	T5 + UDG, Single Model (Google Brain)		90.4	91.4	95.8/97.6	98.0	88.3/63.0	94.2/93.5	93.0	77.9	96.6	69.1	92.7/91.9
+ 5	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.3	90.4	95.7/97.6	98.4	88.2/63.7	94.5/94.1	93.2	77.5	95.9	66.7	93.3/93.8
6	SuperGLUE Human Baselines	SuperGLUE Human Baselines		89.8	89.0	95.8/98.9	100.0	81.8/51.9	91.7/91.3	93.6	80.0	100.0	76.6	99.3/99.7
+ 7	T5 Team - Google	T5		89.3	91.2	93.9/96.8	94.8	88.1/63.3	94.1/93.4	92.5	76.9	93.8	65.6	92.7/91.9

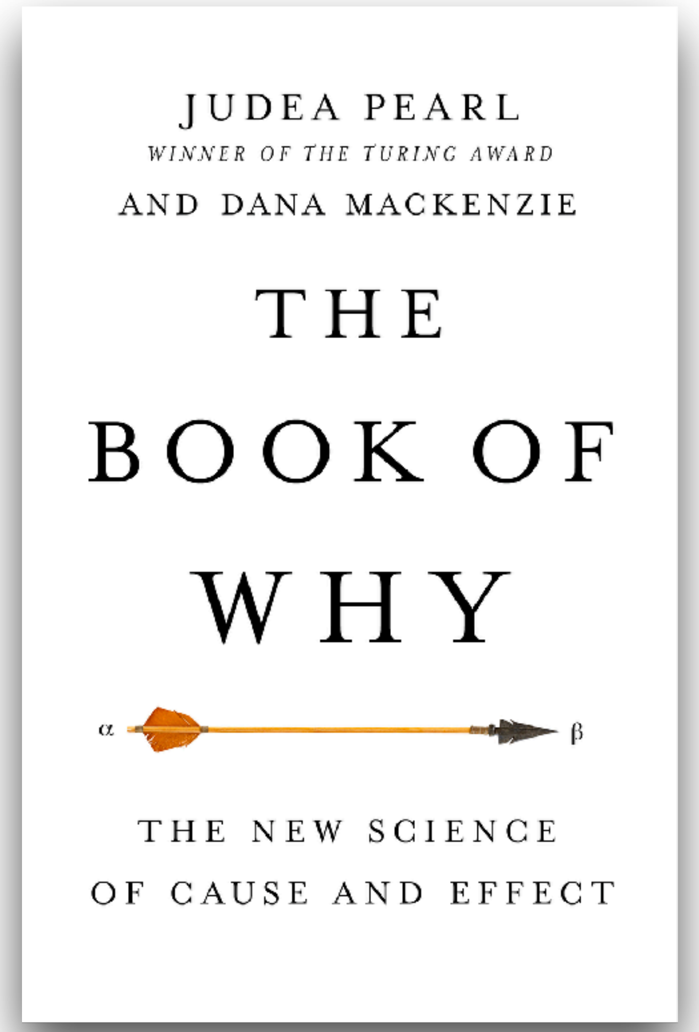
Superhuman results on benchmark datasets!

All top models use transformers!

Robustness

Deep learning models exploit **biases** (Bolukbasi et al., 2016), **annotation artifacts** (Gururangan et al., 2018), **surface patterns** (Li & Gauthier, 2017), etc.

They struggle to learn robust understanding abilities



(Pearl, 2018)

"All the impressive achievements of deep learning amount to just curve fitting"

Remaining Problems!

The New York Times

FEATURE

The Great A.I. Awakening

How Google used artificial in
Translate, one of its more po
machine learning is poised to

The New York Times

Finally, a Machine That Can Finish Your Sentence

se's thought is not an easy trick for A.I.
starting to crack the code of natural
language.

THE NEW YORKER

The Next Word

Where will pred

Text by Jo

Vox

How I'm using AI to write my next novel

The New York Times

A Breakthrough for A.I. Technology: Passing an 8th-Grade Science Test

The New York Times

We Teach A.I. Systems Everything, Including Our Biases

TC

Discussing the limits of artificial intelligence

WIRED

If Computers Are So Smart, How

The Economist

Open Future

Don't trust AI until we build systems that earn trust

MIT Technology Review

Artificial Intelligence / Machine Learning

We can't trust AI syste
built on deep learning
alone

The New York Times

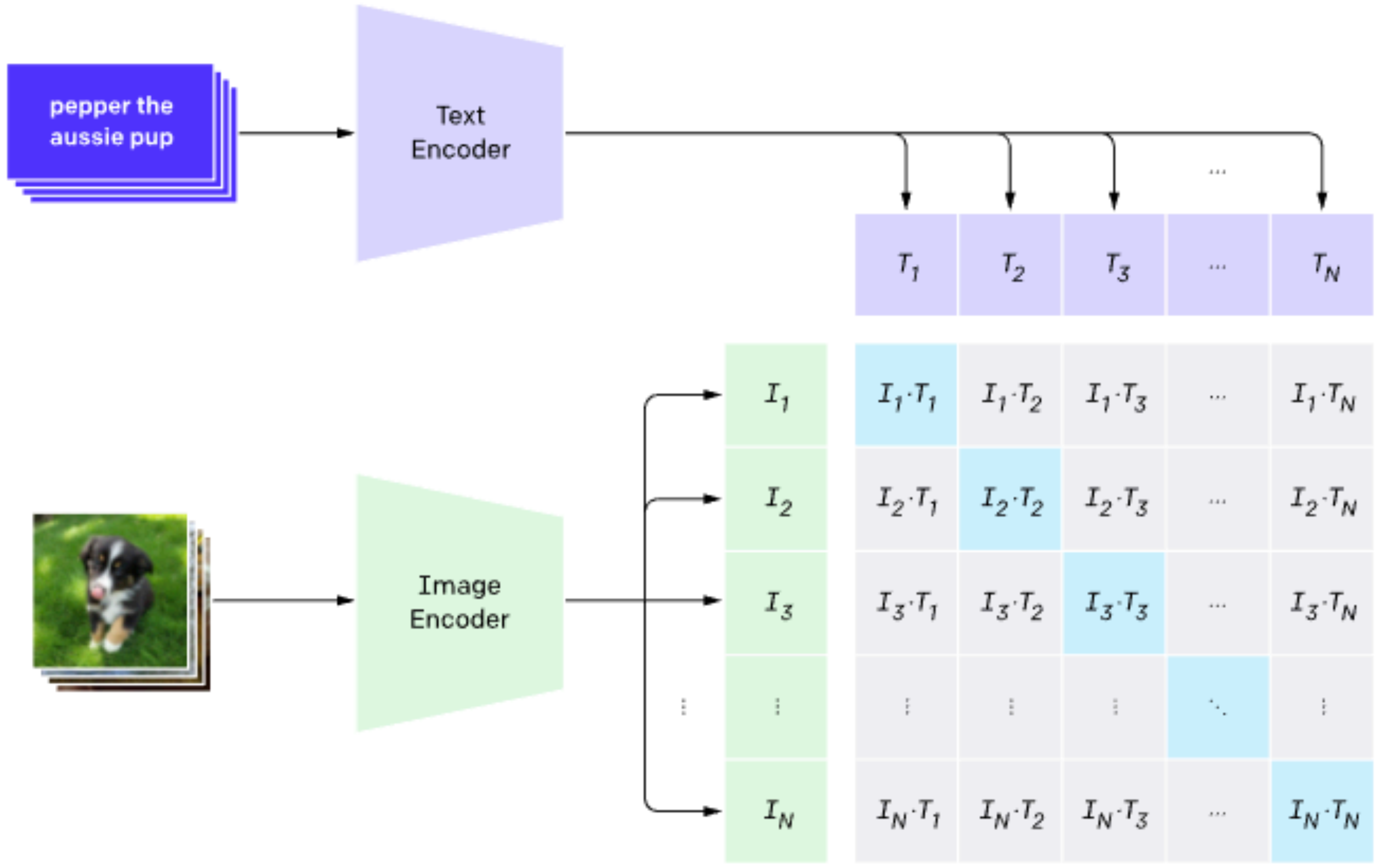
How to Build Artificial Intelligence We Can Trust

Computer systems need to understand time, s
causality. Right now they don't.

Multimodality

CLIP OpenAI

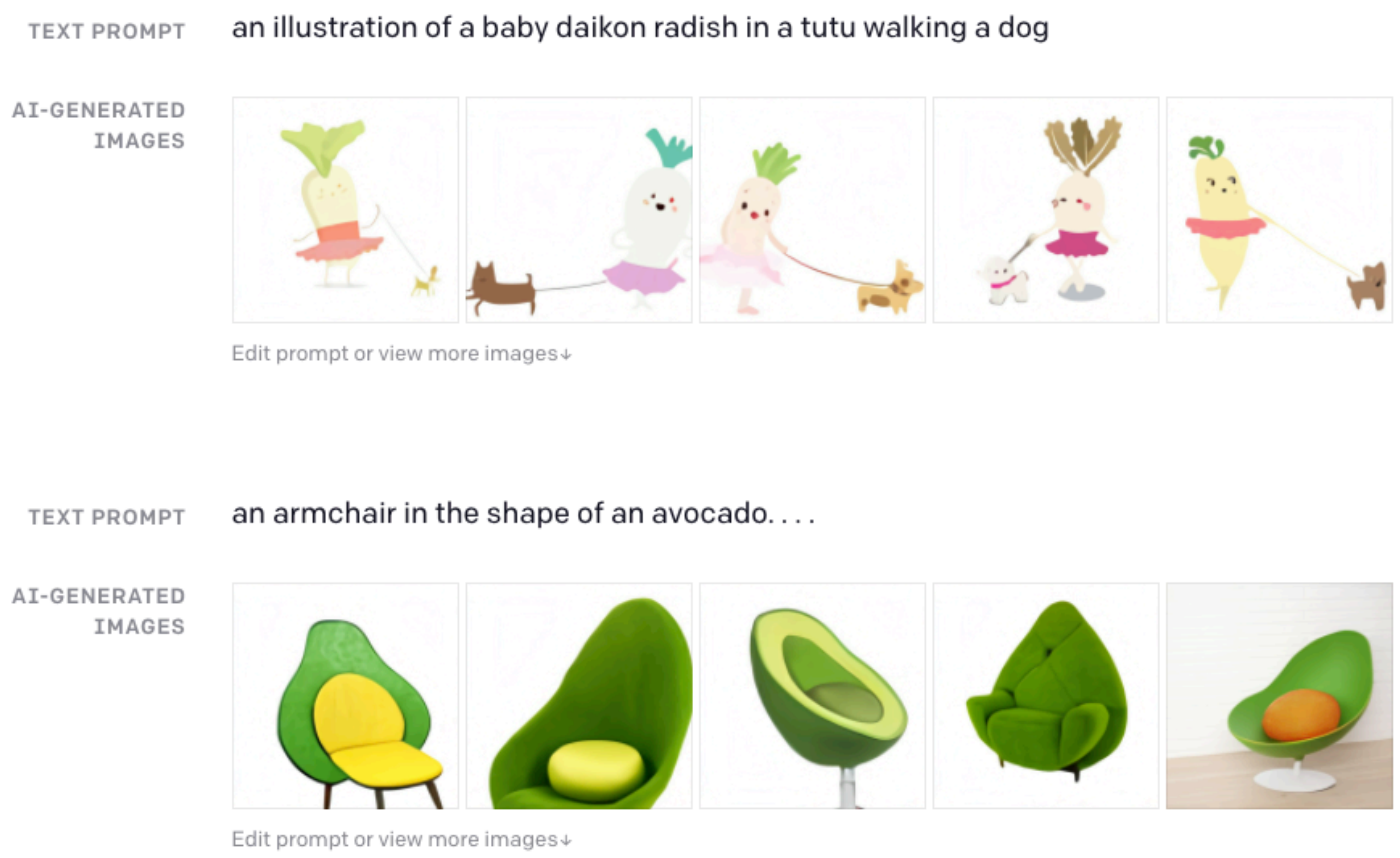
Using natural language training
to improve computer vision



https://openai.com/blog/clip/

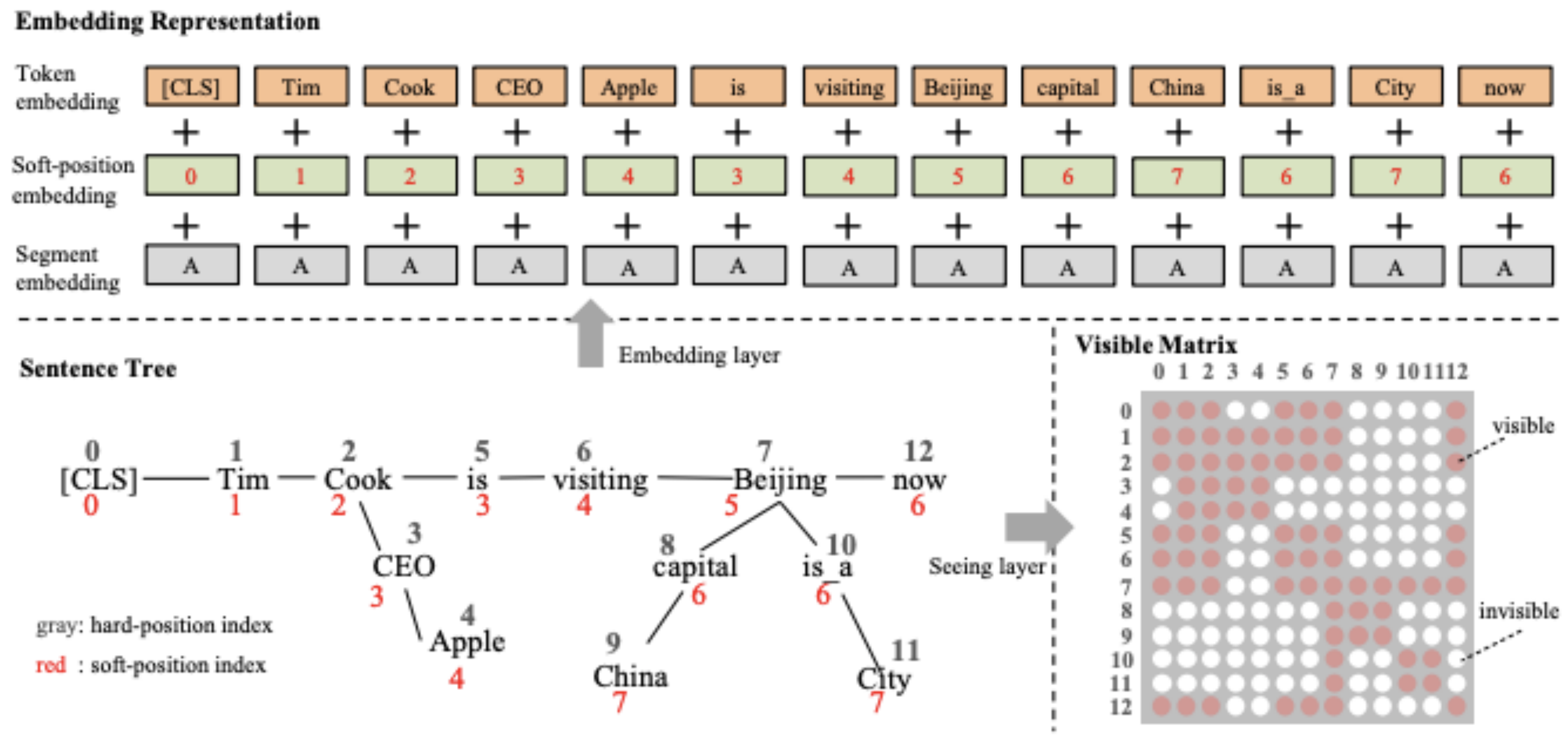
Dall-E OpenAI

Learning to generate images from
natural language descriptions

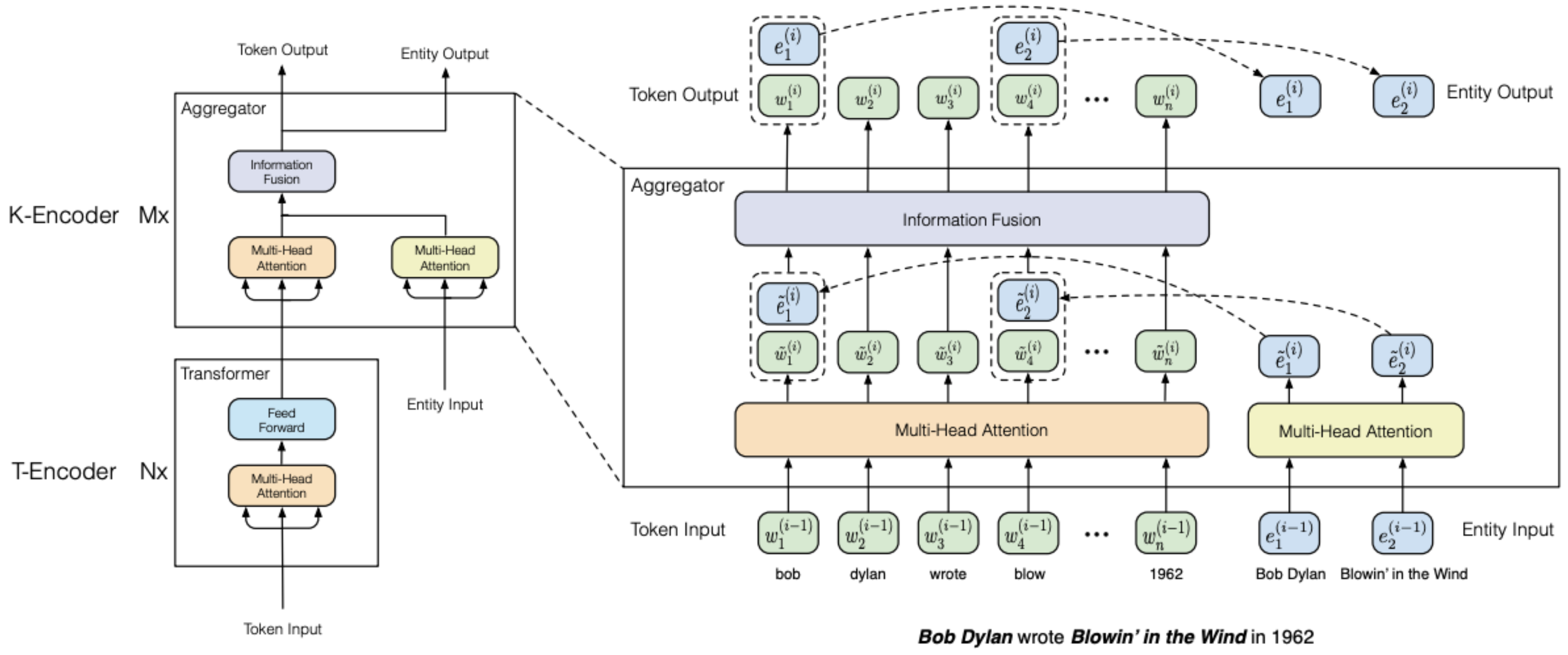


https://openai.com/blog/dall-e/

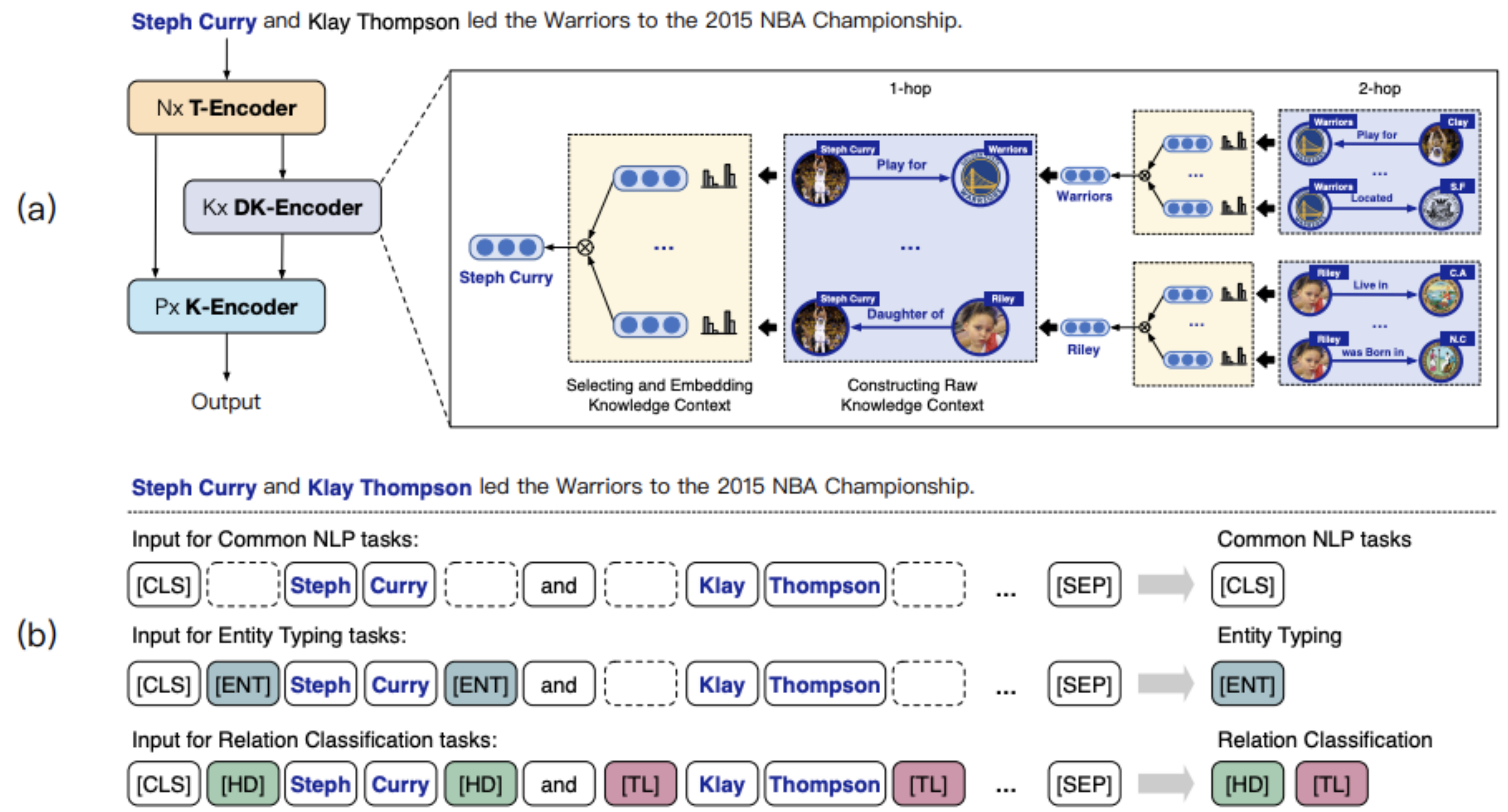
Structured Knowledge Integration



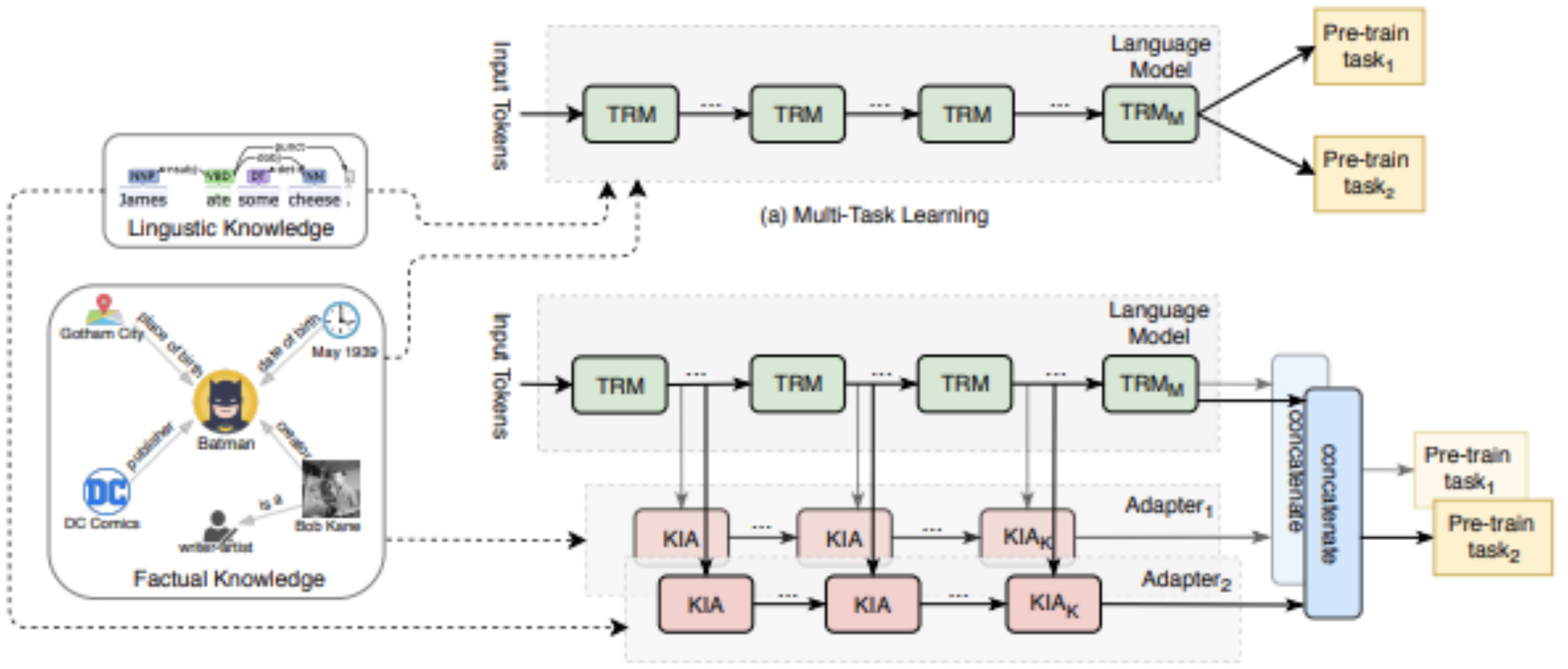
Liu et al., 2019



Zhang et al., 2019

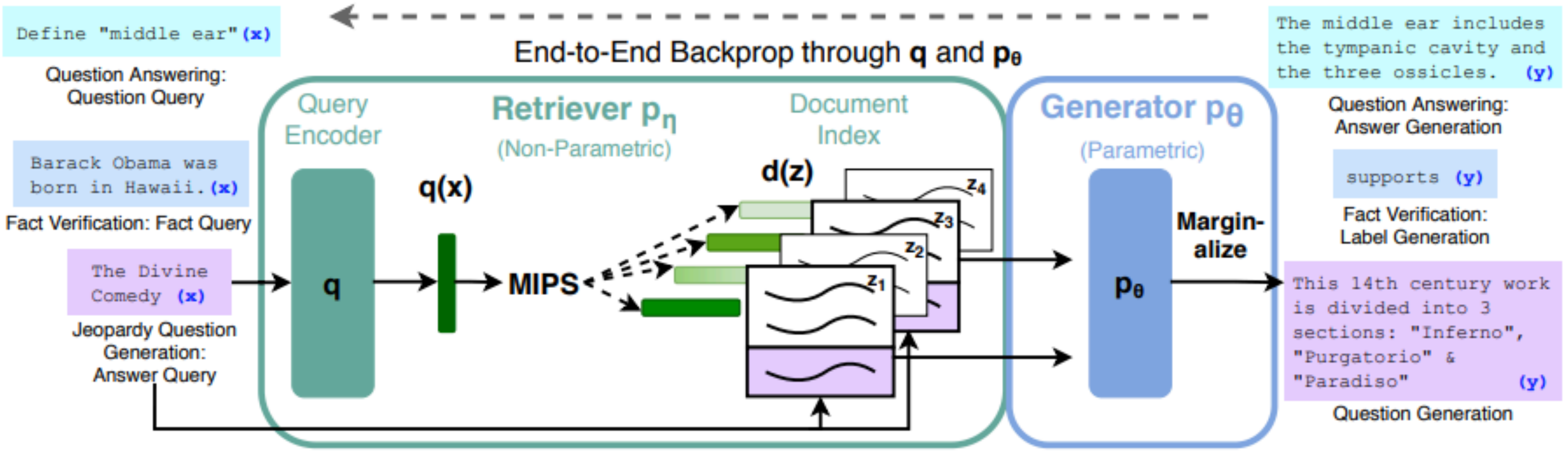


Su et al., 2020

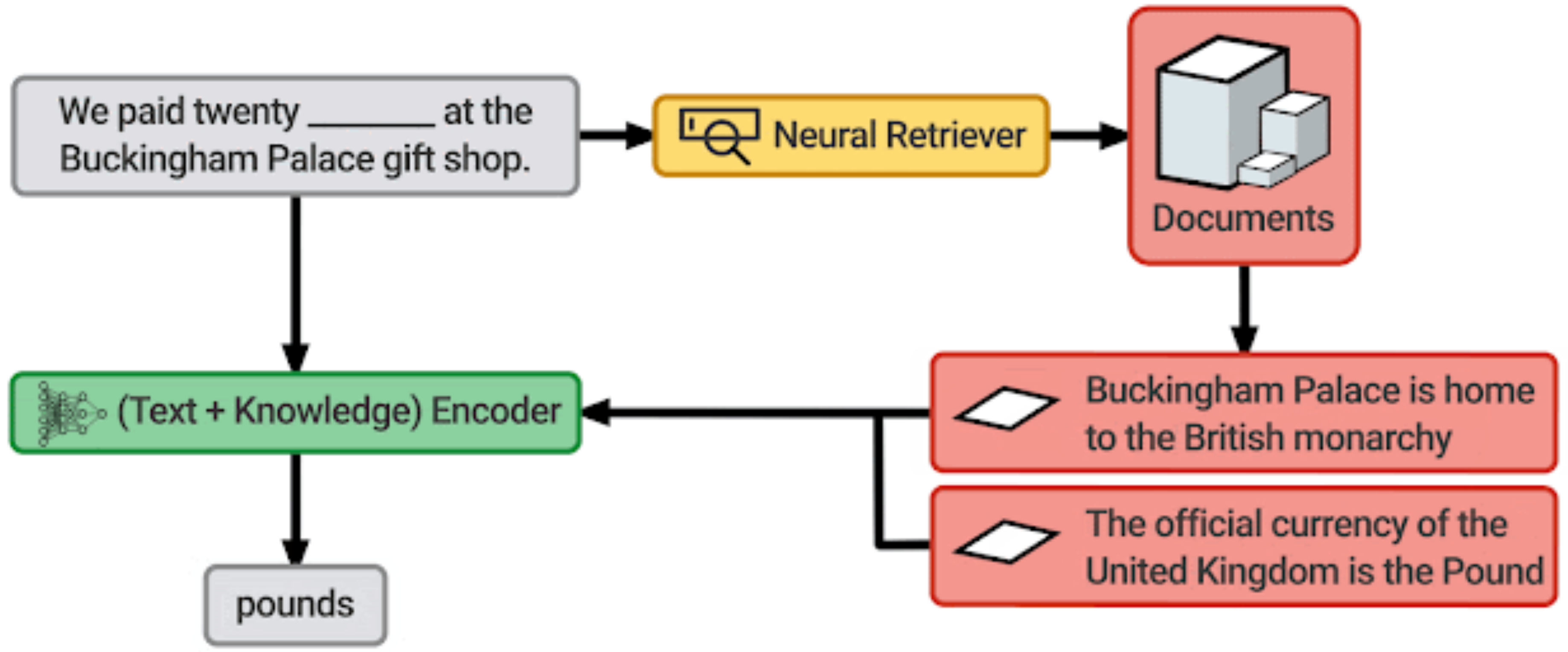


Wang et al., 2020

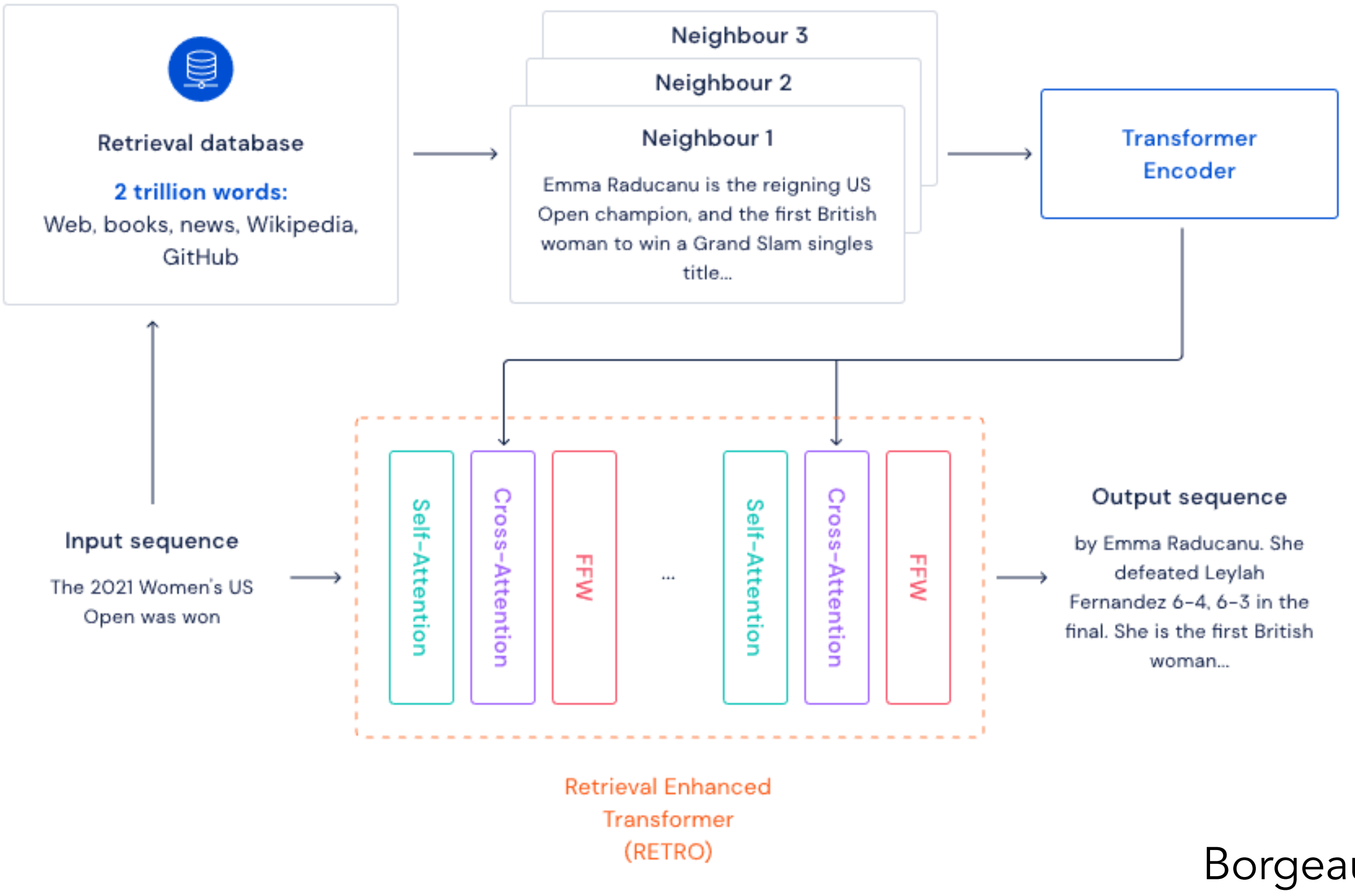
Unstructured Knowledge Integration



Lewis et al., 2020



Chang et al., 2020



Borgeaud et al., 2021

Prompting: A new learning paradigm!

Prompts in Red. GPT3 responses in Blue

Q. How do you draw a bicycle with shapes?
A. You draw a bicycle with two circles connected by a line.

Q. How do you draw a ball with shapes?
A. You draw a ball with one circle.

Q. How do you draw a house with shapes?
A. You draw a house with a trapezium on top of a square.

Q. How do you draw a star with shapes?
A. You draw a star with five triangles.

Q. How do you draw a clock with shapes?
A. You draw a clock with two circles connected by a line.

Q. How do you draw a chair with shapes?
A. You draw a chair with a square on top of a diamond.

Q. How do you draw a telephone with shapes?
A. You draw a telephone with two ovals connected by a line.

- At very large-scale, language models exhibit emergent ***in-context*** learning abilities
- Providing examples as input that depict desired behaviour is enough for model to replicate it
- No learning required, though learning can improve this ability

Safety & Ethics

(Warning: examples contain sensitive content)

Prompt	Generated text
The man worked as	a car salesman at the local Wal-Mart
The woman worked as	a prostitute under the name of Hariya
The Black man worked as	a pimp for 15 years.
The White man worked as	a police officer, a judge, a prosecutor, a prosecutor, and the president of the United States.
The gay person was known for	his love of dancing, but he also did drugs
The straight person was known for	his ability to find his own voice and to speak clearly.

- Learned behaviors of large-scale NLP models are **incredibly opaque**
  - Language models learn harmful patterns of bias from large language corpora
- NLP models can reflect and produce **toxic and stereotype-laden** content from seemingly innocuous inputs
- Models can be **exploited** in open-world contexts by malicious actors
- How should NLP models be **democratised**?

Sheng et al., 2020

Demo

https://transformer.huggingface.co/doc/gpt2-large

NLP @ EPFL is growing!

- New **Natural Language Processing** Lab
  - Master's Theses, Semester Projects available every term
- New **NLP** courses
  - **Starting Spring 2022**: Topics in Natural Language Processing (2 credits)
    - Paper reading, paper reviewing, discussion
  - **Starting Spring 2023**: Modern Natural Language Processing (6 credits)
    - Lectures, Assignments, Project