Modeling Sequential Data with Recurrent Networks

Chris Dyer
DeepMind
Carnegie Mellon University

LxMLS 2016

July 28, 2016

Outline: Part I

- Neural networks as feature inducers
- Recurrent neural networks
  - Application: language models
- Learning challenges and solutions
  - Vanishing gradients
  - Long short-term memories
  - Gated recurrent units
- Break

Outline: Part II

- RNN performance tuning and implementation tricks
- Bidirectional RNNs
  - Application: better word representations
- Sequence to Sequence transduction with RNNs
  - Applications: machine translation & image caption generation
- Sequences as matrices and attention
  - Application: machine translation

Feature Induction

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathcal{F} = \frac{1}{M}\sum_{i=1}^{M} ||\hat{\mathbf{y}}_i - \mathbf{y}_i||_2^2$$

In linear regression, the goal is to learn **W** and **b** such that *F* is minimized for a dataset *D* consisting of *M* training instances. An engineer must select/design **x** *carefully*.

Feature Induction

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathcal{F} = \frac{1}{M}\sum_{i=1}^{M} ||\hat{\mathbf{y}}_i - \mathbf{y}_i||_2^2$$

In linear regression, the goal is to learn **W** and **b** such that *F* is minimized for a dataset *D* consisting of *M* training instances. An engineer must select/design **x** *carefully*.

$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

"nonlinear regression"

Use "naive features" **x** and *learn* their transformations (conjunctions, nonlinear transformation, etc.) into **h**.

Feature Induction

$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

- What functions can this parametric form compute?
    - If **h** is big enough (i.e., enough dimensions), it can represent any vector-valued function to any degree of precision
- This is a much more powerful regression model!

Feature Induction

$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

- What functions can this parametric form compute?
  - If **h** is big enough (i.e., enough dimensions), it can represent any vector-valued function to any degree of precision
- This is a much more powerful regression model!
- You can think of **h** as "induced features" in a linear classifier
  - The network did the job of a feature engineer

Recurrent Neural Networks

- Lots of interesting data is sequential in nature
  - Words in sentences
  - DNA
  - Stock market returns
  - …
- **How do we represent an arbitrarily long history?**

Recurrent Neural Networks

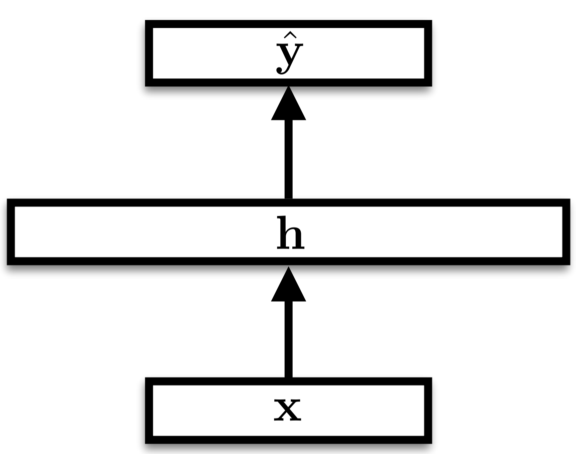
- Lots of interesting data is sequential in nature
  - Words in sentences
  - DNA
  - Stock market returns
  - …
- **How do we represent an arbitrarily long history?**
  - we will train neural networks to build a representation of these arbitrarily big sequences

Recurrent Neural Networks

Feed-forward NN

$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$



Recurrent Neural Networks

Feed-forward NN

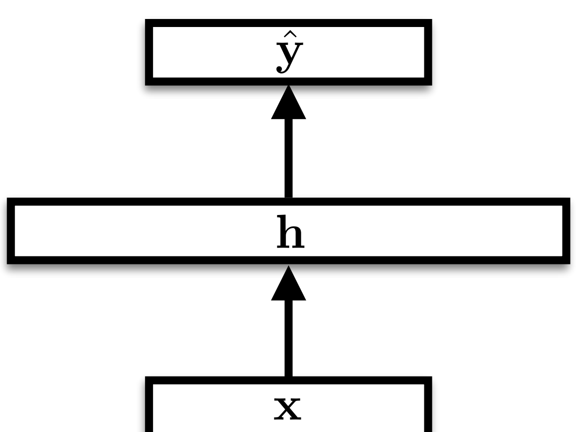
$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$

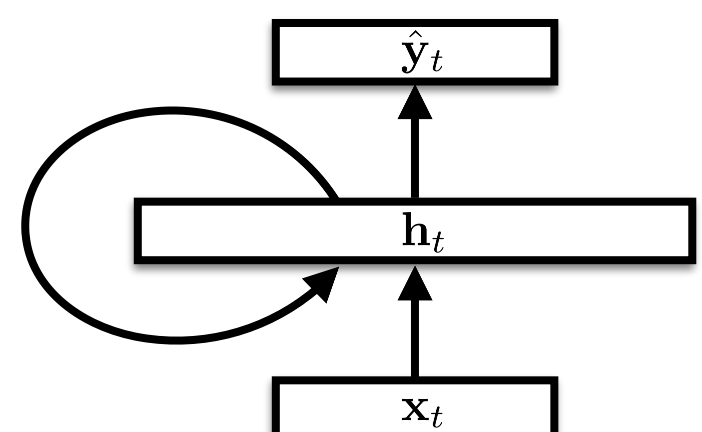
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

Recurrent NN

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$





Recurrent Neural Networks

Feed-forward NN

$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$

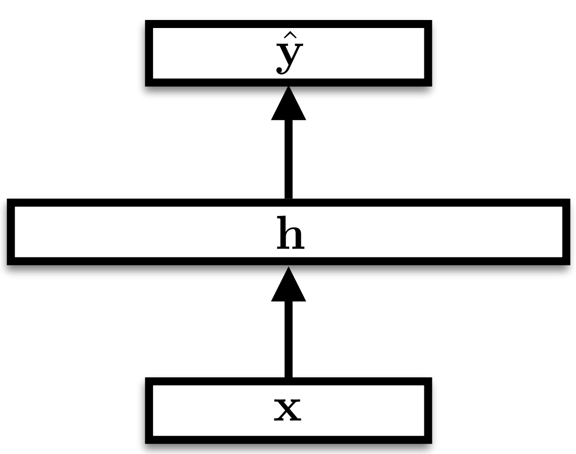
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

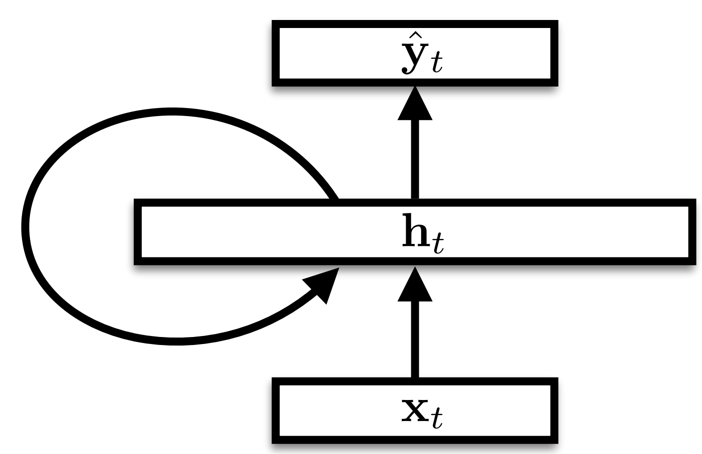
Recurrent NN

$$\cancel{\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})}$$

$$\mathbf{h}_t = g(\mathbf{V}[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{c})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$





Recurrent Neural Networks

Feed-forward NN

$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$

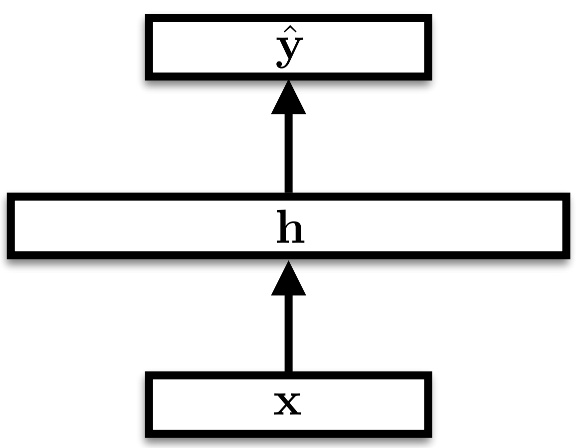
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

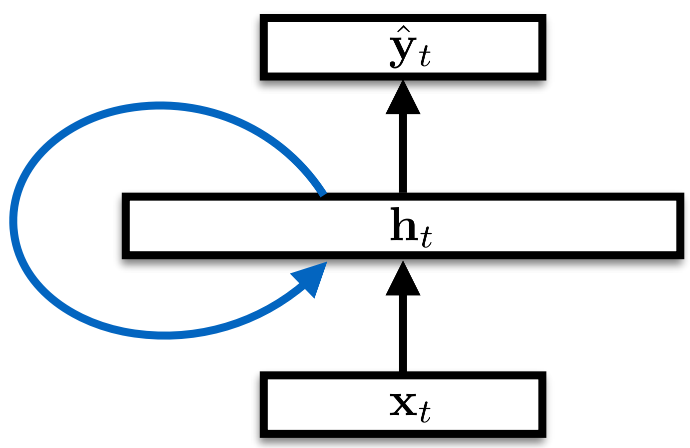
Recurrent NN

$$\cancel{\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})}$$

$$\mathbf{h}_t = g(\mathbf{V}[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{c})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$

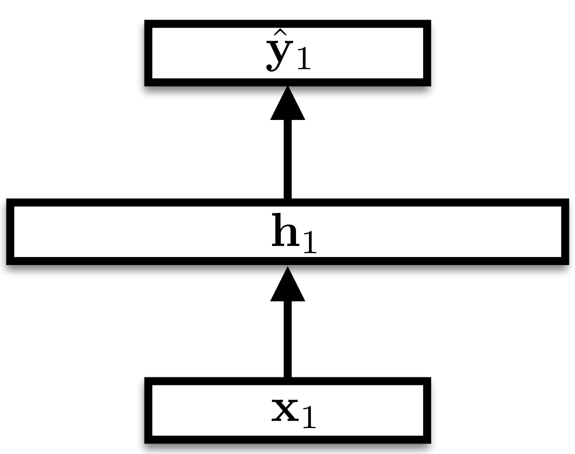




Recurrent Neural Networks

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

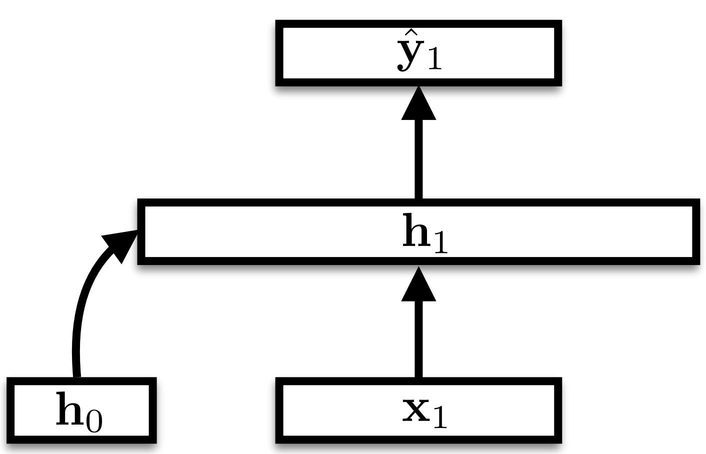
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

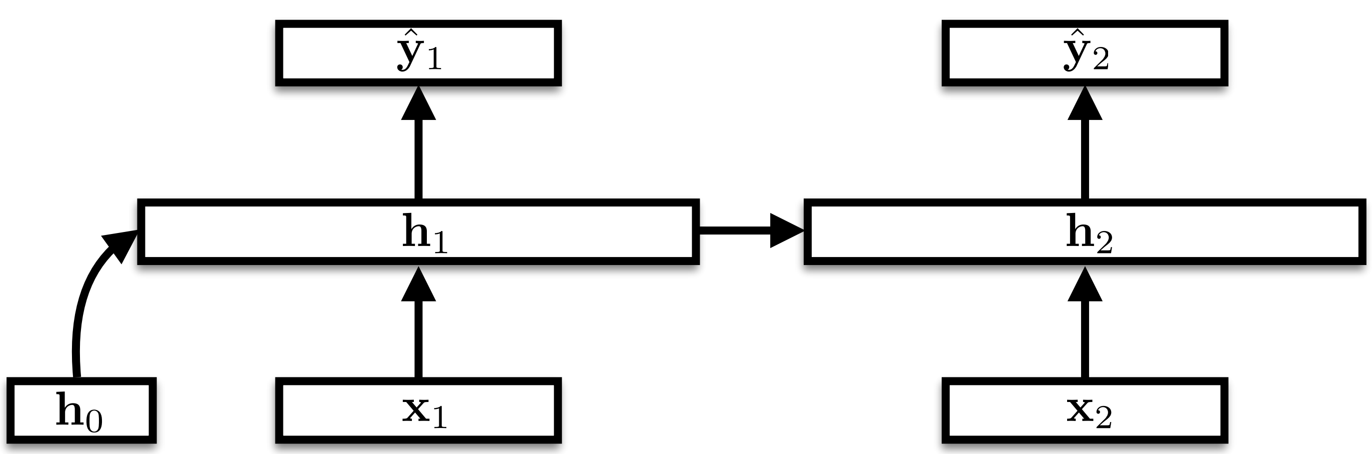
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



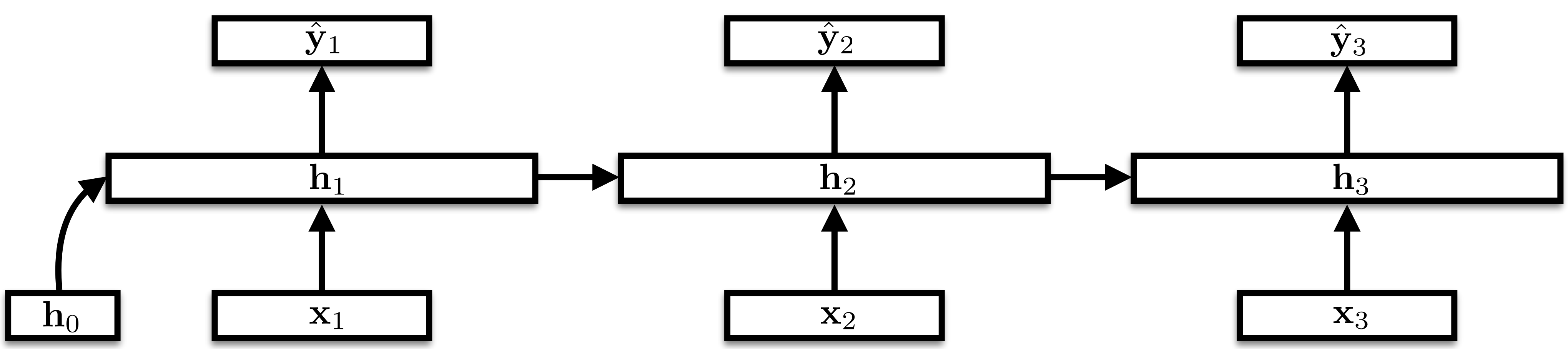
Recurrent Neural Networks

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



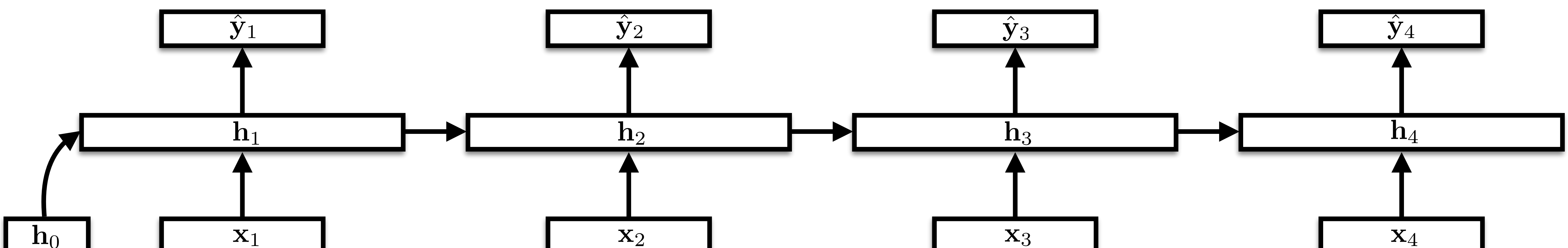
Recurrent Neural Networks

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

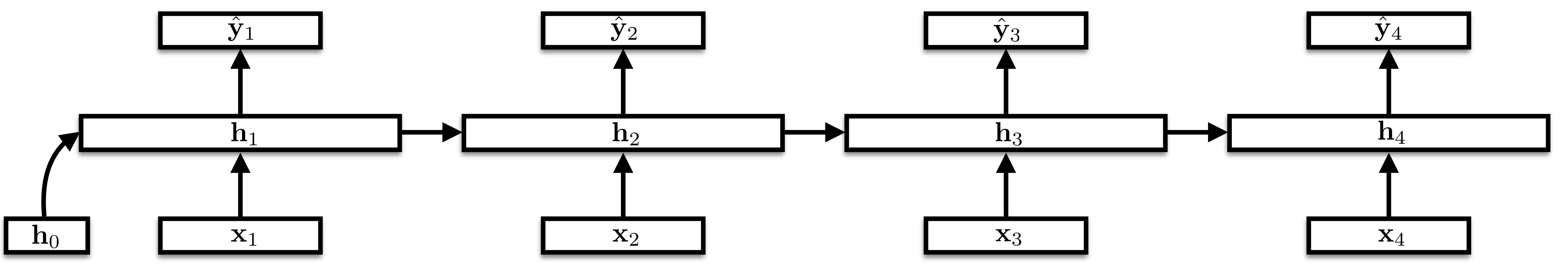
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

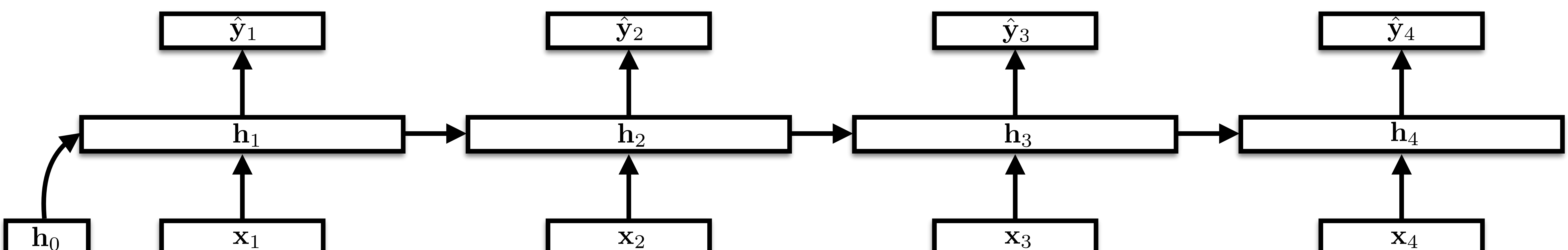
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$

How do we train the RNN's parameters?



Recurrent Neural Networks

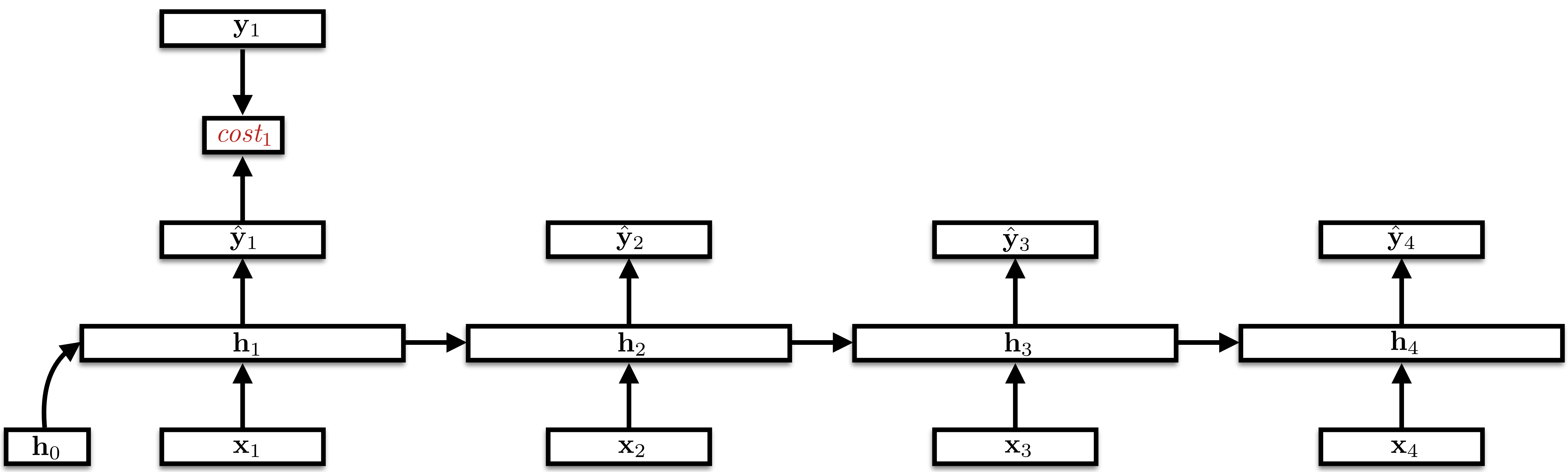
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

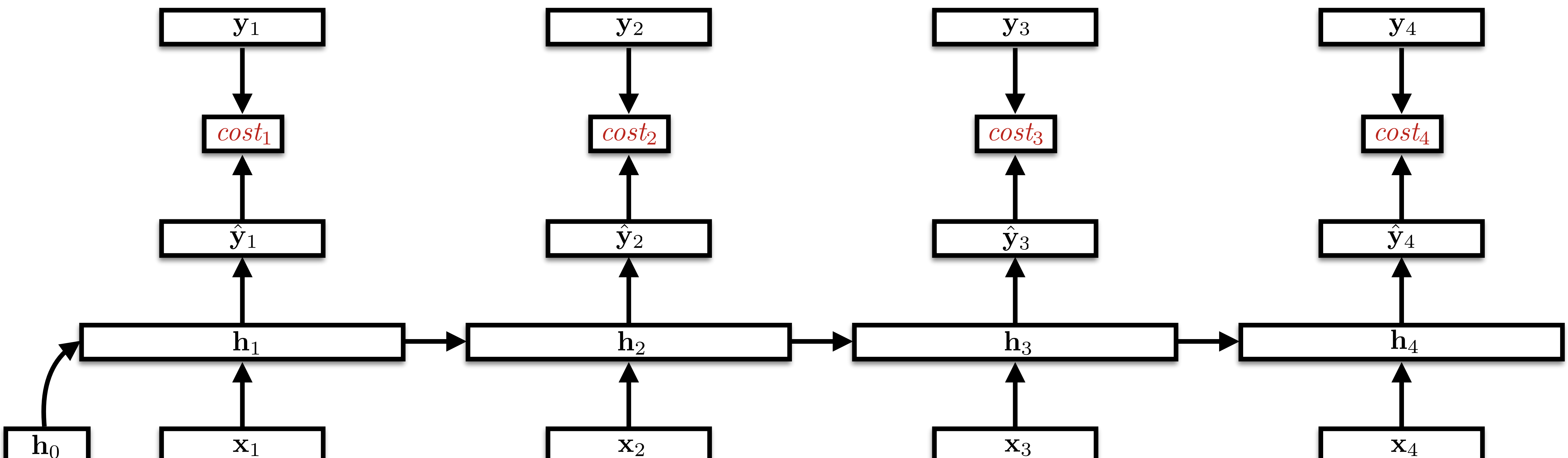
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

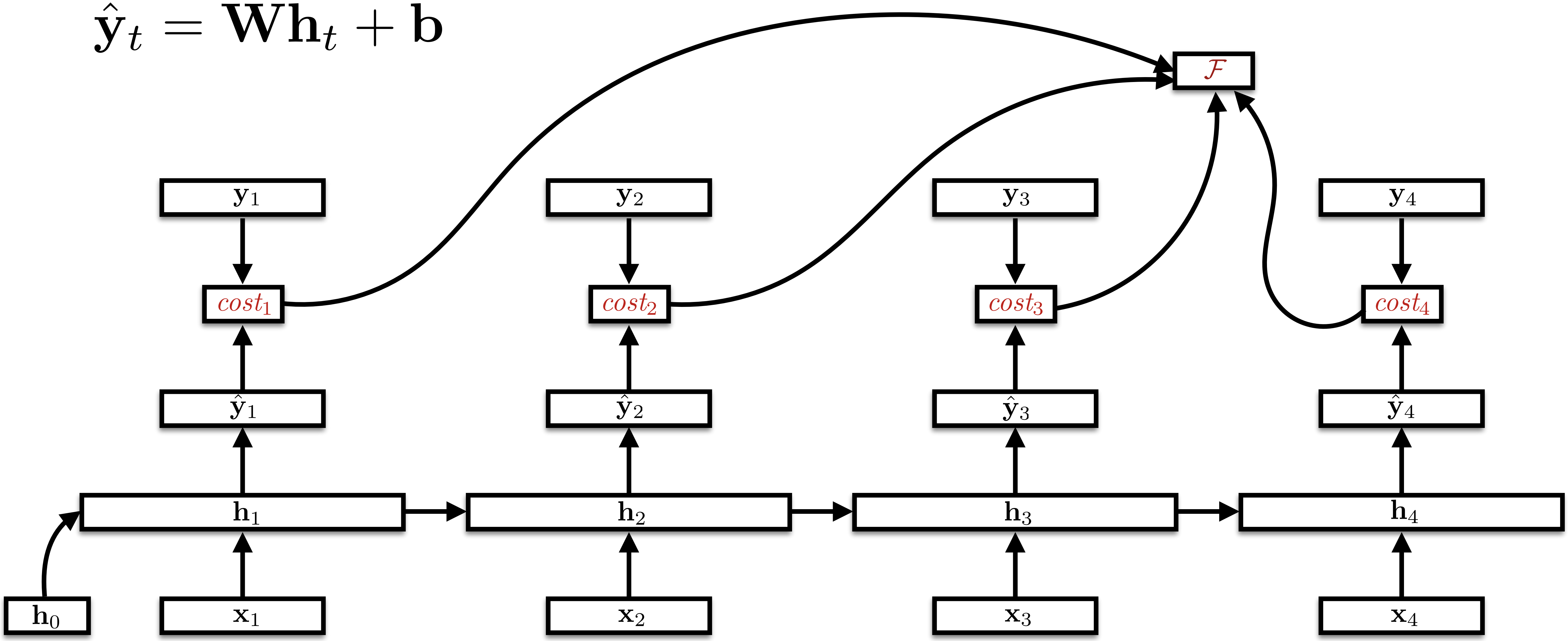
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



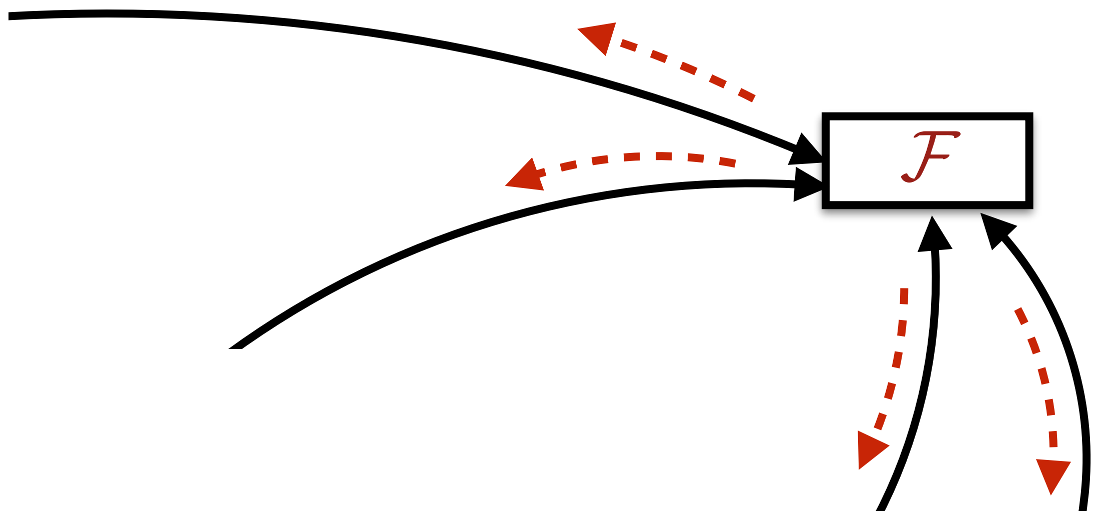
Recurrent Neural Networks

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

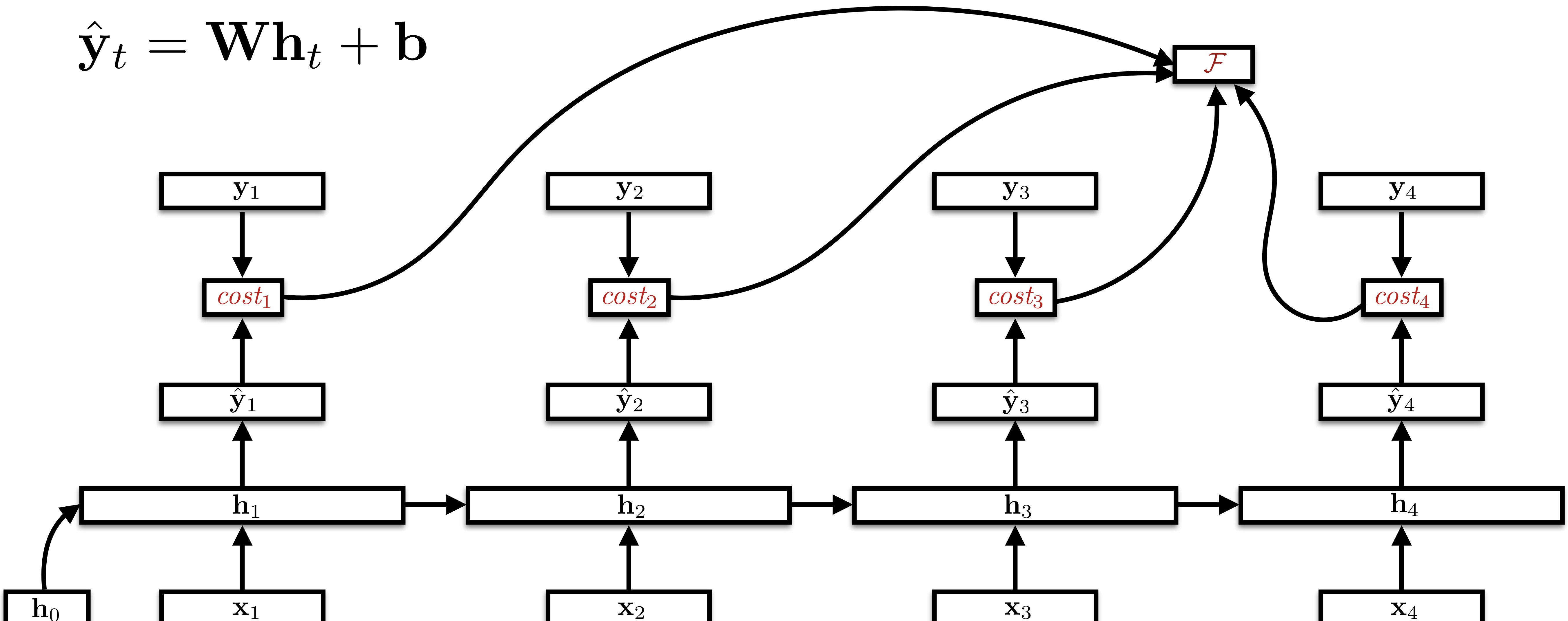


- The unrolled graph is a well-formed (DAG) computation graph—we can run backprop
  - Parameters are tied across time, derivatives are aggregated across all time steps
  - This is historically called "backpropagation through time" (BPTT)

Parameter Tying

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

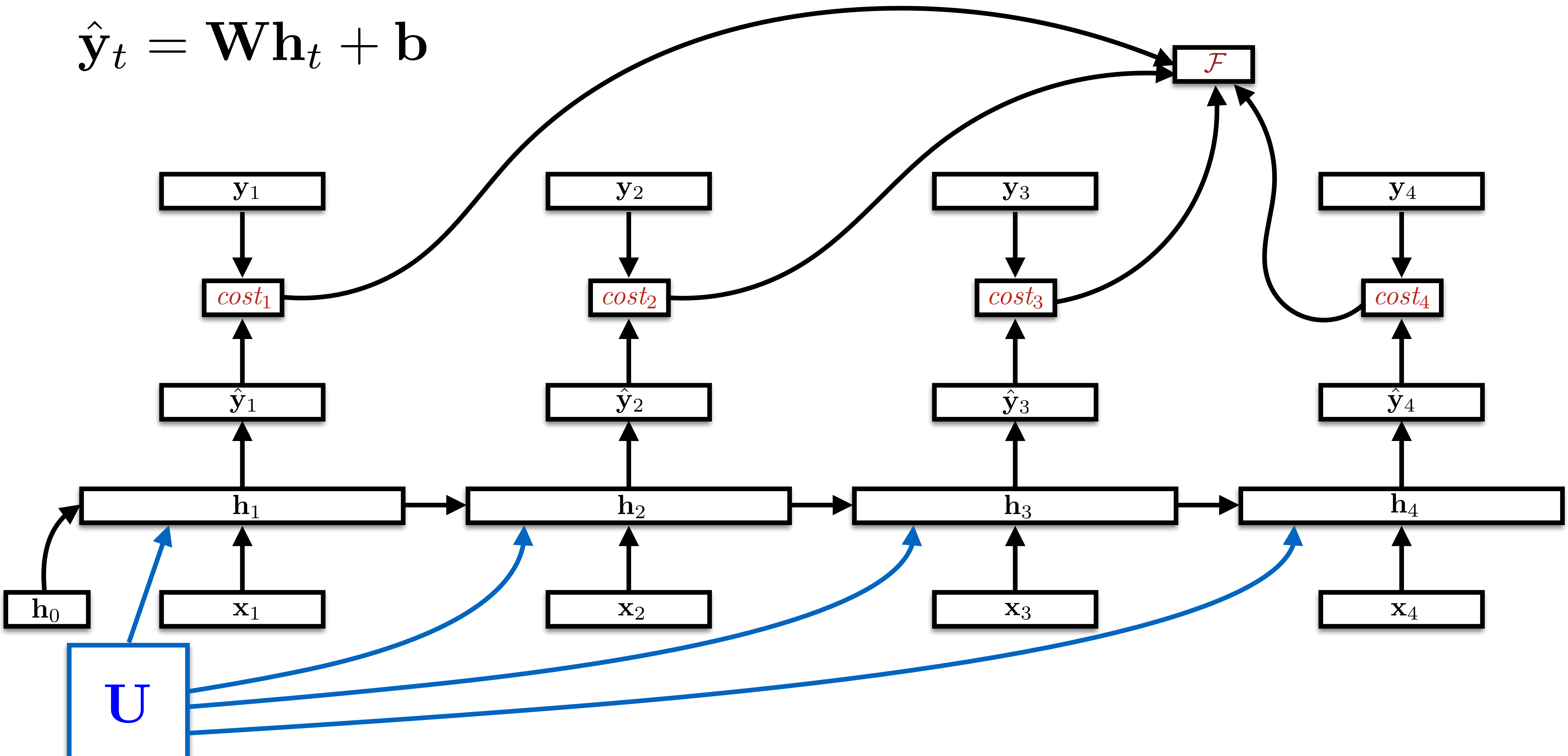
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



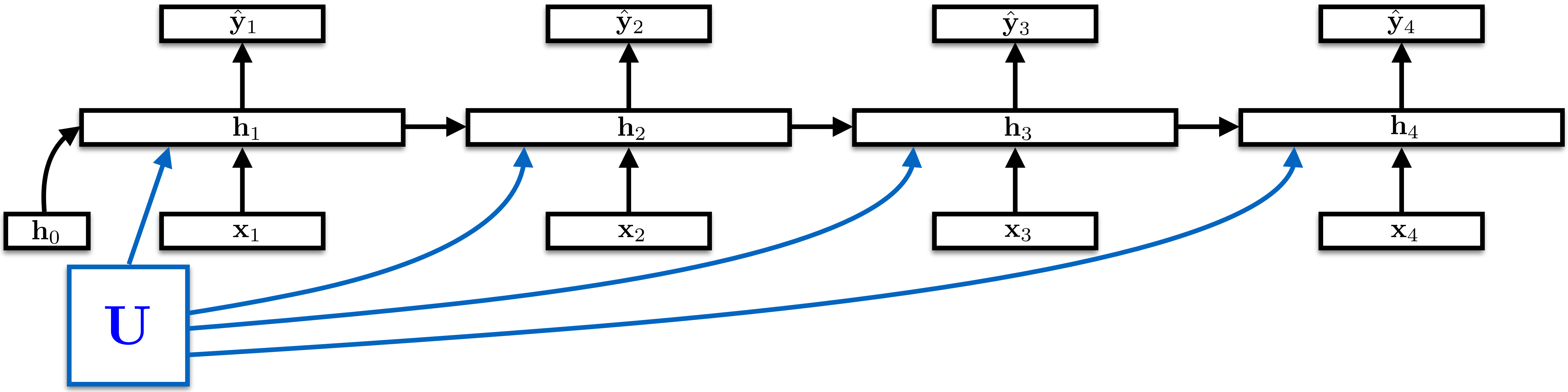
Parameter Tying

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

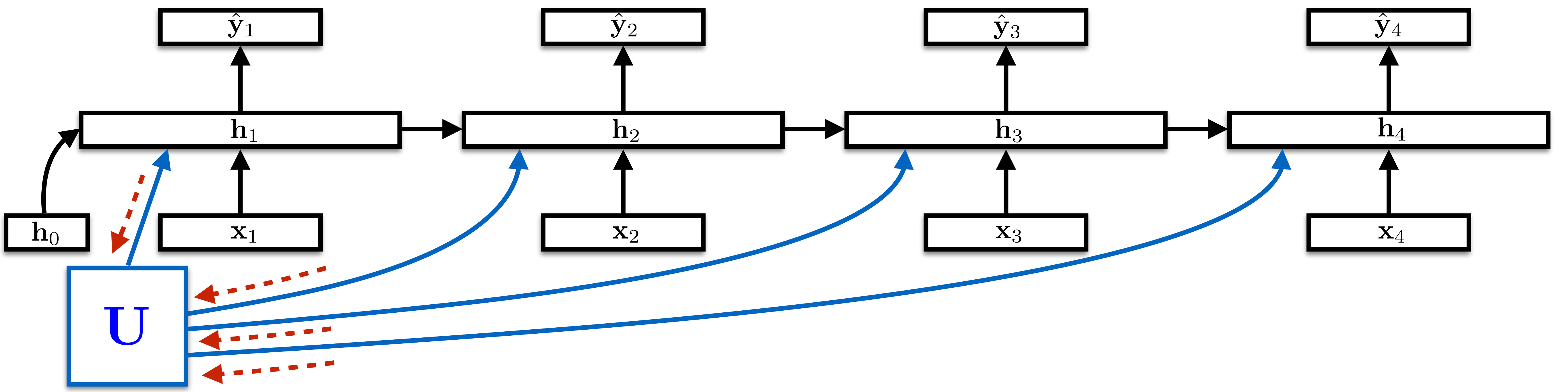
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Parameter Tying

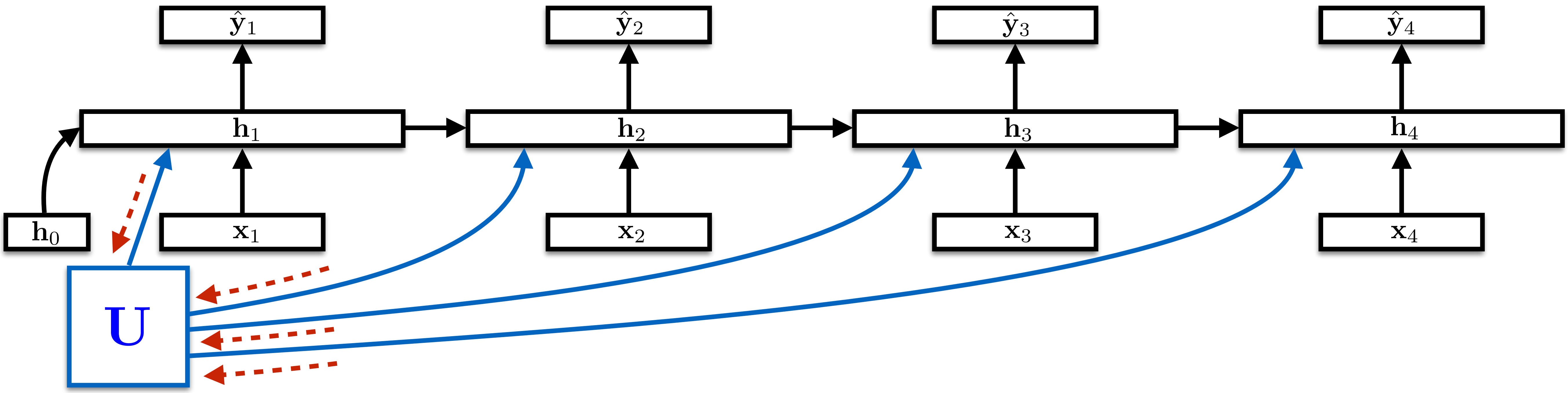


Parameter Tying



$$\frac{\partial \mathcal{F}}{\partial \mathbf{U}} = \sum_{t=1}^{4} \frac{\partial \mathbf{h}_t}{\partial \mathbf{U}} \frac{\partial \mathcal{F}}{\partial \mathbf{h}_t}$$

Parameter Tying



$$\frac{\partial \mathcal{F}}{\partial \mathbf{U}} = \sum_{t=1}^{4} \frac{\partial \mathbf{h}_t}{\partial \mathbf{U}} \frac{\partial \mathcal{F}}{\partial \mathbf{h}_t}$$

Parameter tying also came up when learning the filters in convolutional networks (and in the transition matrices for HMMs!).

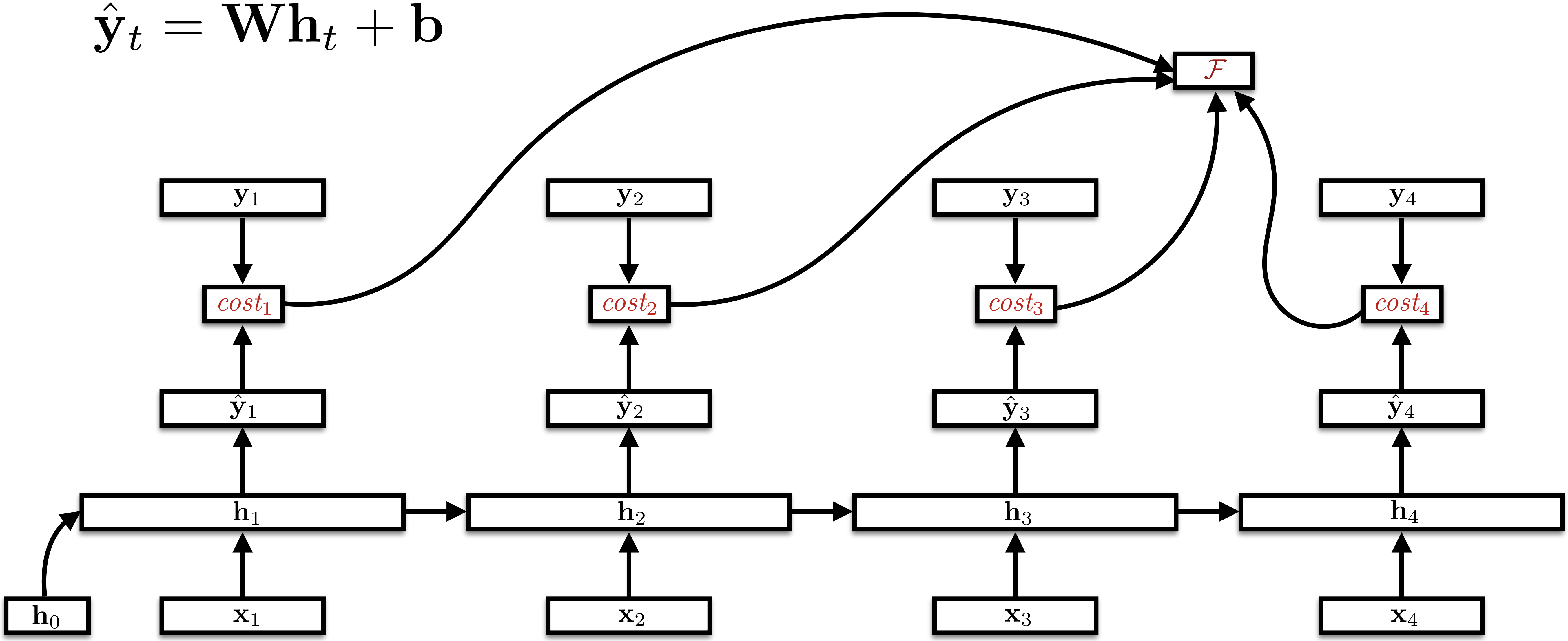
Parameter Tying

- Why do we want to tie parameters?
  - Reduce the number of parameters to be learned
  - Deal with arbitrarily long sequences
- What if we always have short sequences?
  - Maybe you might untie parameters, then. But you wouldn't have an RNN anymore!

What else can we do?

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

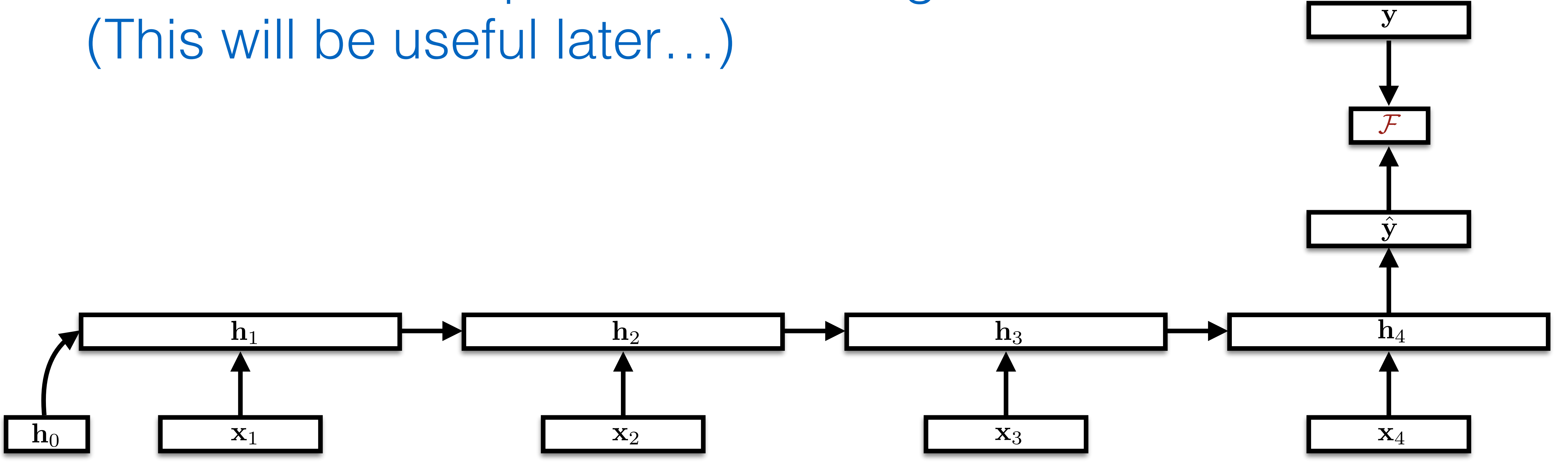
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



“Read and summarize”

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$

Summarize a sequence into a single vector.
(This will be useful later…)



View 2: Recursive Definition

- Recall how to construct a list recursively:
  base case
      [] is a list (the empty list)

View 2: Recursive Definition

- Recall how to construct a list recursively:
  base case

  [] is a list (the empty list)

  induction

  [**t** | `h`] where **t** is a list and `h` is an atom is a list

View 2: Recursive Definition

- Recall how to construct a list recursively:
  base case
      [] is a list (the empty list)

  induction
      [**t** | `h`] where **t** is a list and `h` is an atom is a list

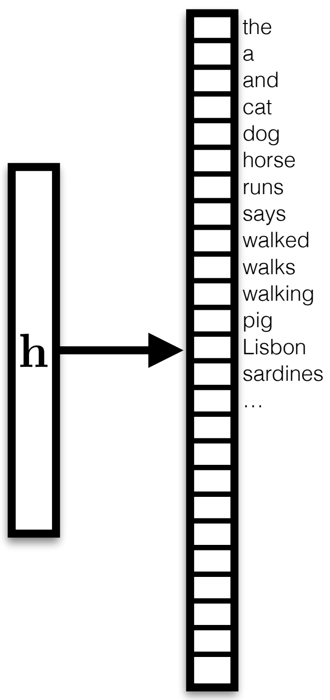
- RNNs define functions that compute representations recursively according to this definition of a list.

  - Define (learn) a representation of the base case

  - Learn a representation of the inductive step

- **Anything you can construct recursively, you can obtain an "embedding" of with neural networks using this general strategy**

Example: Language Model



$$\mathbf{u} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

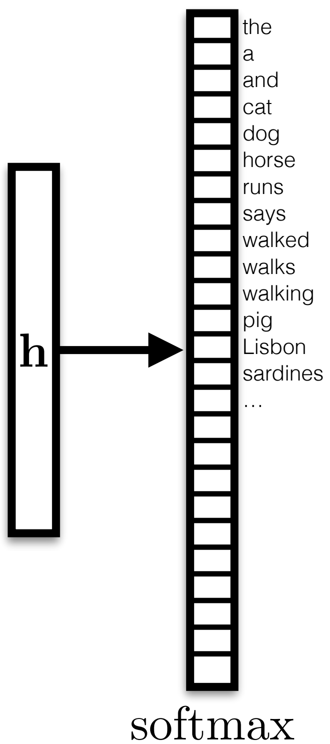
$$p_i = \frac{\exp u_i}{\sum_j \exp u_j}$$

$$\mathbf{h} \in \mathbb{R}^d$$

$$|V| = 100,000$$

What are the dimensions of $\mathbf{W}$?

Example: Language Model



$$\mathbf{u} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

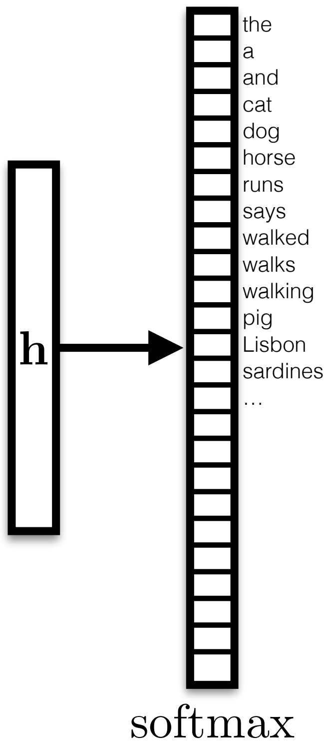
$$p_i = \frac{\exp u_i}{\sum_j \exp u_j}$$

$$\mathbf{h} \in \mathbb{R}^d$$

$$|V| = 100{,}000$$

What are the dimensions of $\mathbf{b}$?

Example: Language Model



$$\mathbf{u} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

$$p_i = \frac{\exp u_i}{\sum_j \exp u_j}$$

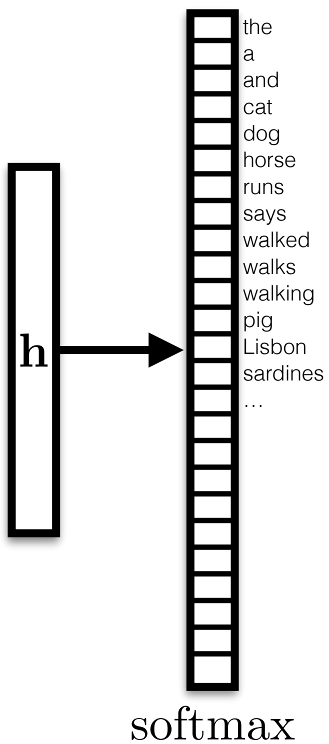
$$\mathbf{h} \in \mathbb{R}^d$$

$$|V| = 100,000$$

What are the dimensions of $\mathbf{b}$?

$$\begin{aligned} p(\boldsymbol{e}) = & p(e_1) \times \\ & p(e_2 \mid e_1) \times \\ & p(e_3 \mid e_1, e_2) \times \\ & p(e_4 \mid e_1, e_2, e_3) \times \\ & \cdots \end{aligned}$$

Example: Language Model



$$\mathbf{u} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

$$p_i = \frac{\exp u_i}{\sum_j \exp u_j}$$

$$\mathbf{h} \in \mathbb{R}^d$$

$$|V| = 100,000$$

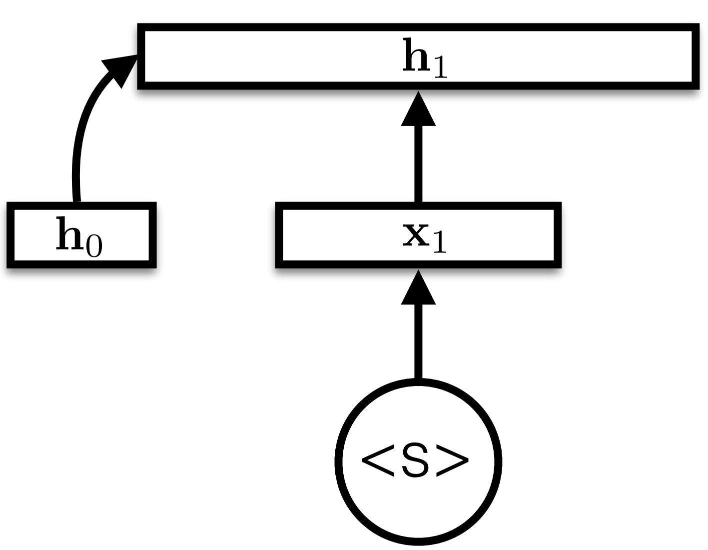
What are the dimensions of $\mathbf{b}$?

$$\begin{aligned} p(\boldsymbol{e}) = & p(e_1) \times \\ & p(e_2 \mid e_1) \times \\ & p(e_3 \mid e_1, e_2) \times \\ & p(e_4 \mid e_1, e_2, e_3) \times \\ & \cdots \end{aligned}$$

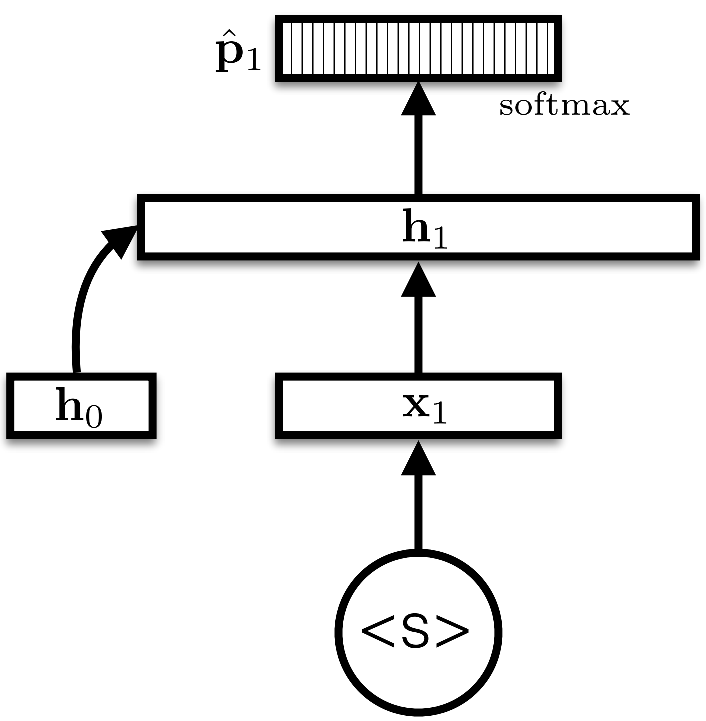
histories are sequences of words…

Example: Language Model

Example: Language Model

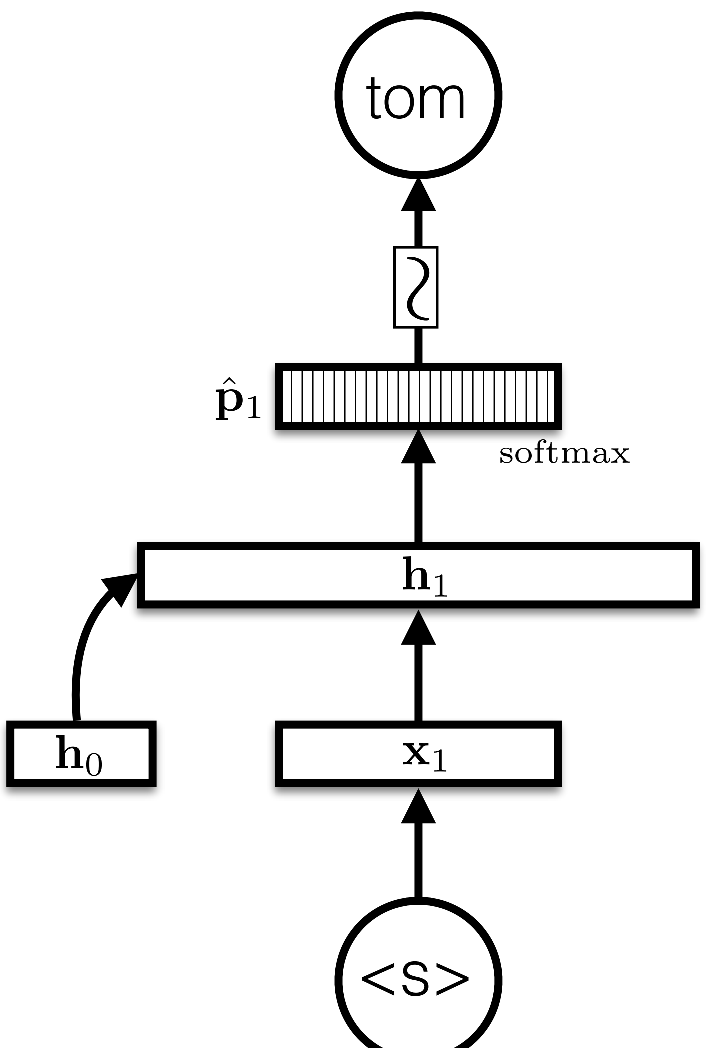


Example: Language Model



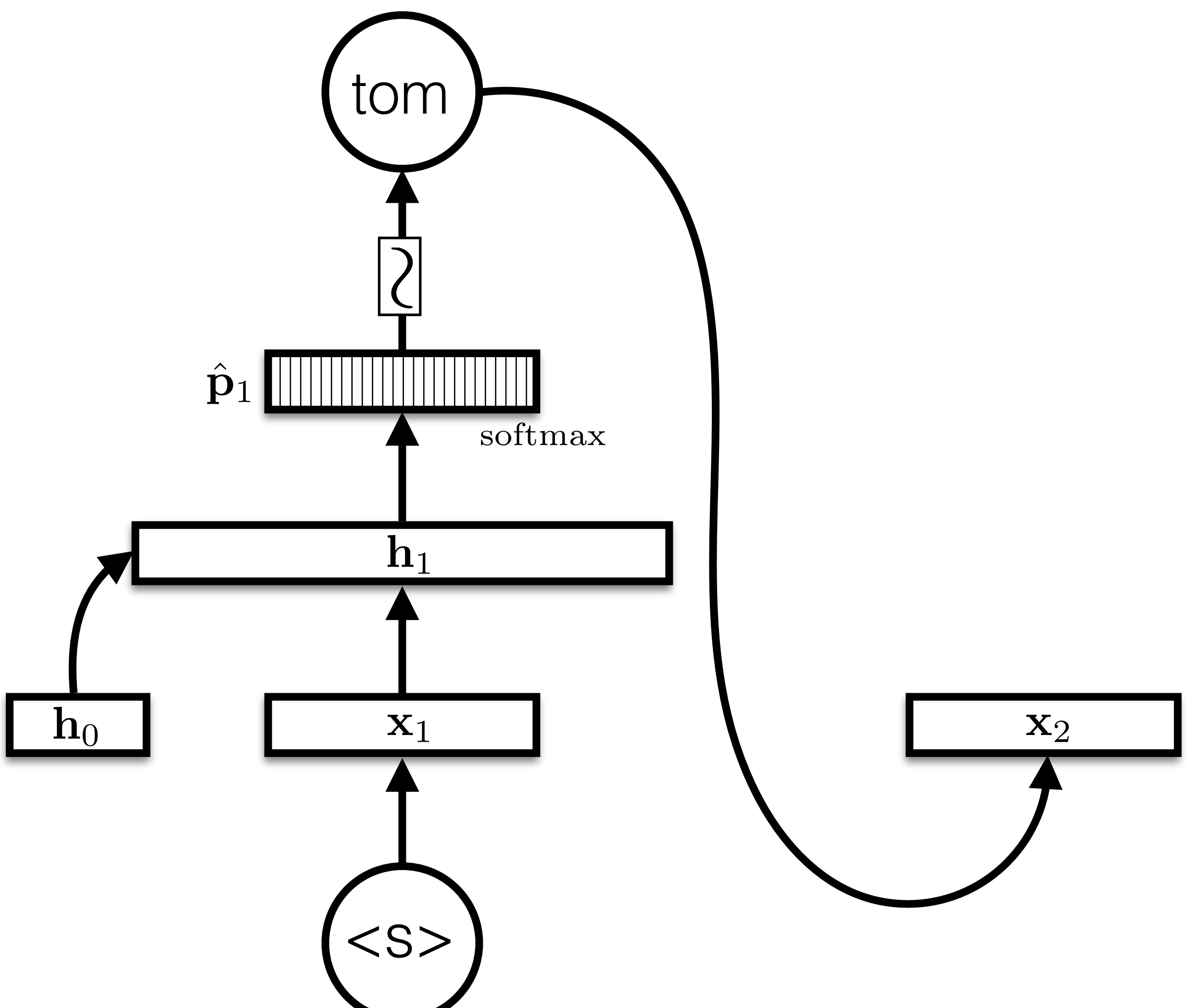
Example: Language Model

$$p(tom \mid \langle \mathbf{s} \rangle)$$



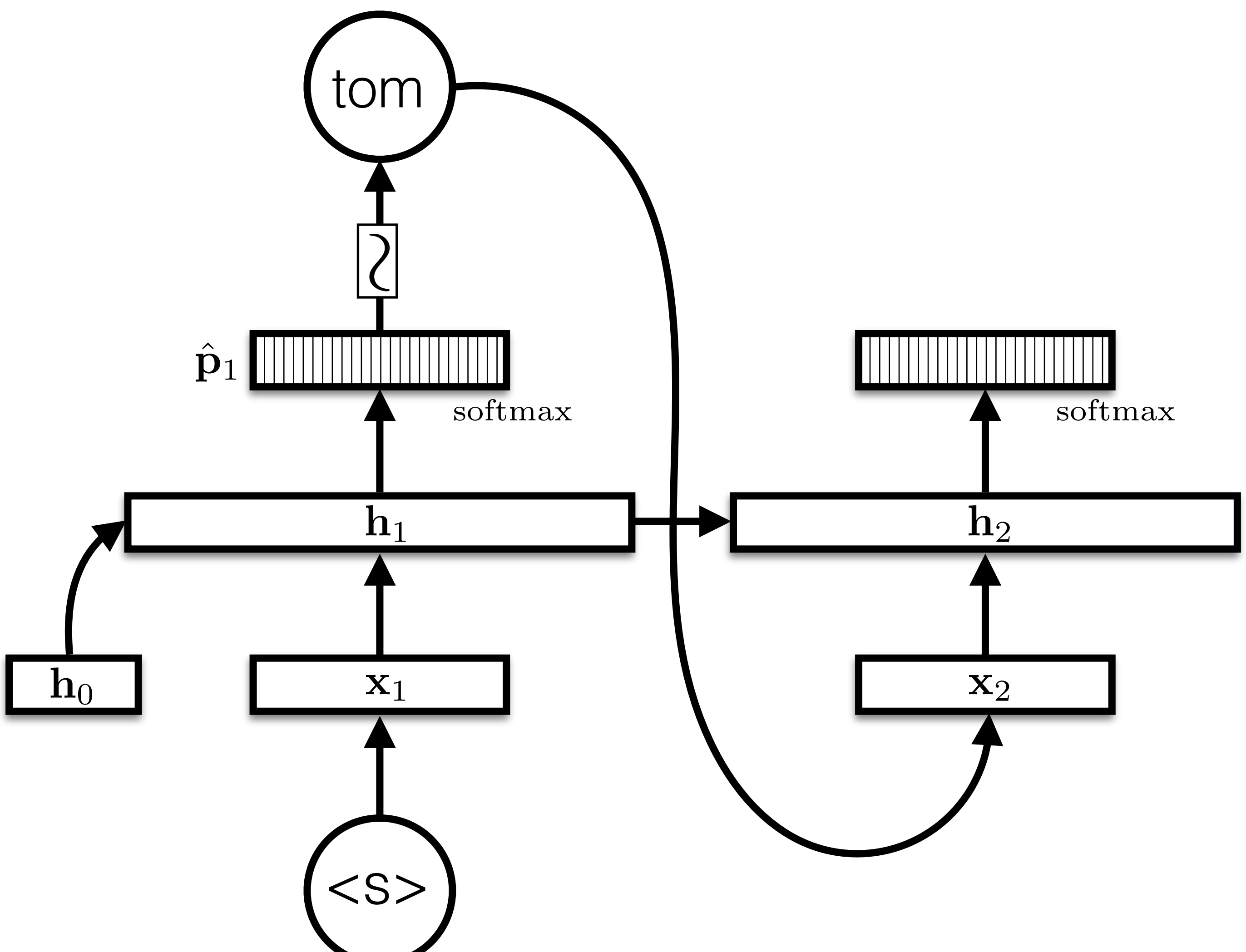
Example: Language Model

$p(tom \mid \langle \mathbf{s} \rangle)$



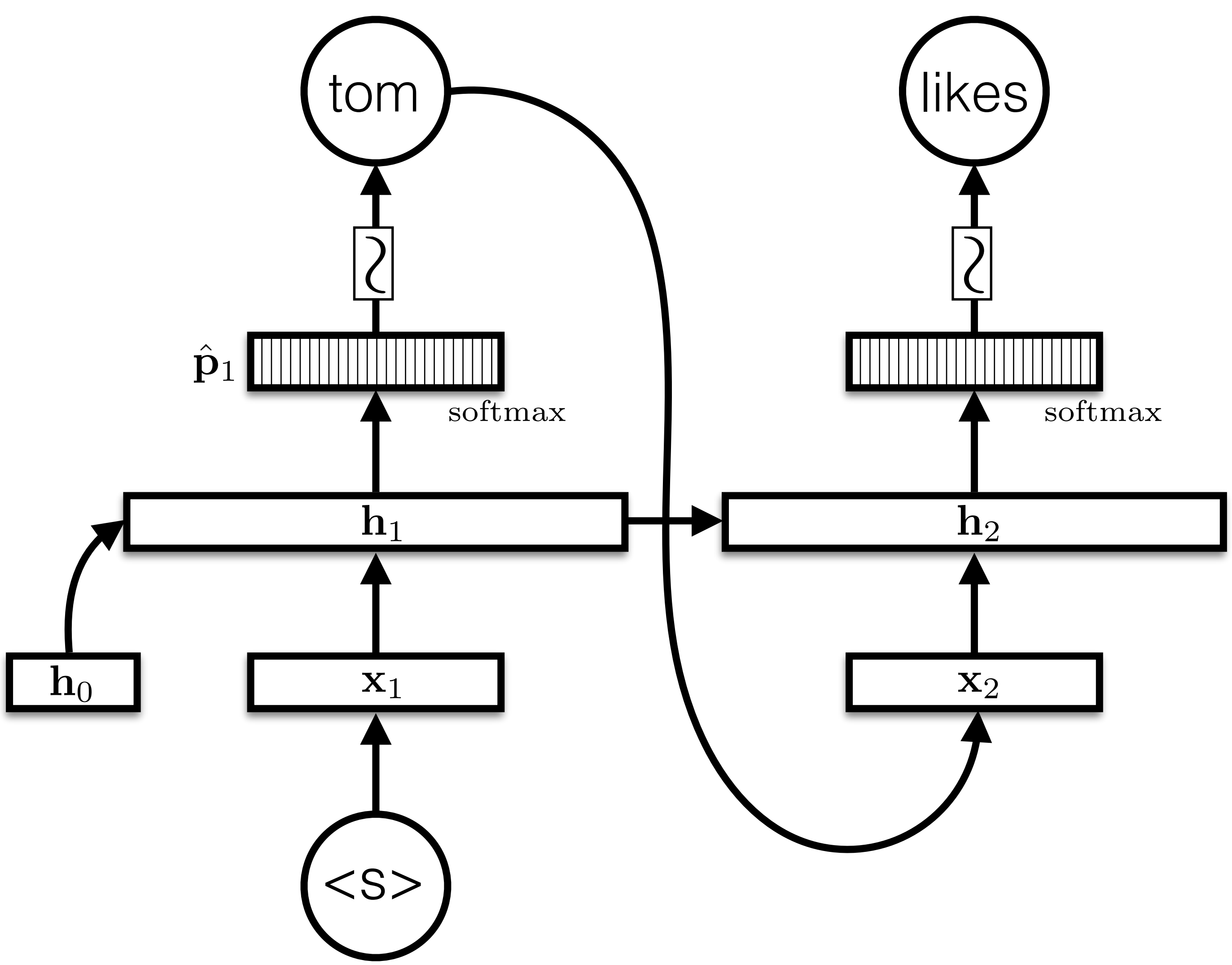
Example: Language Model

$p(tom \mid \langle \mathbf{s} \rangle)$



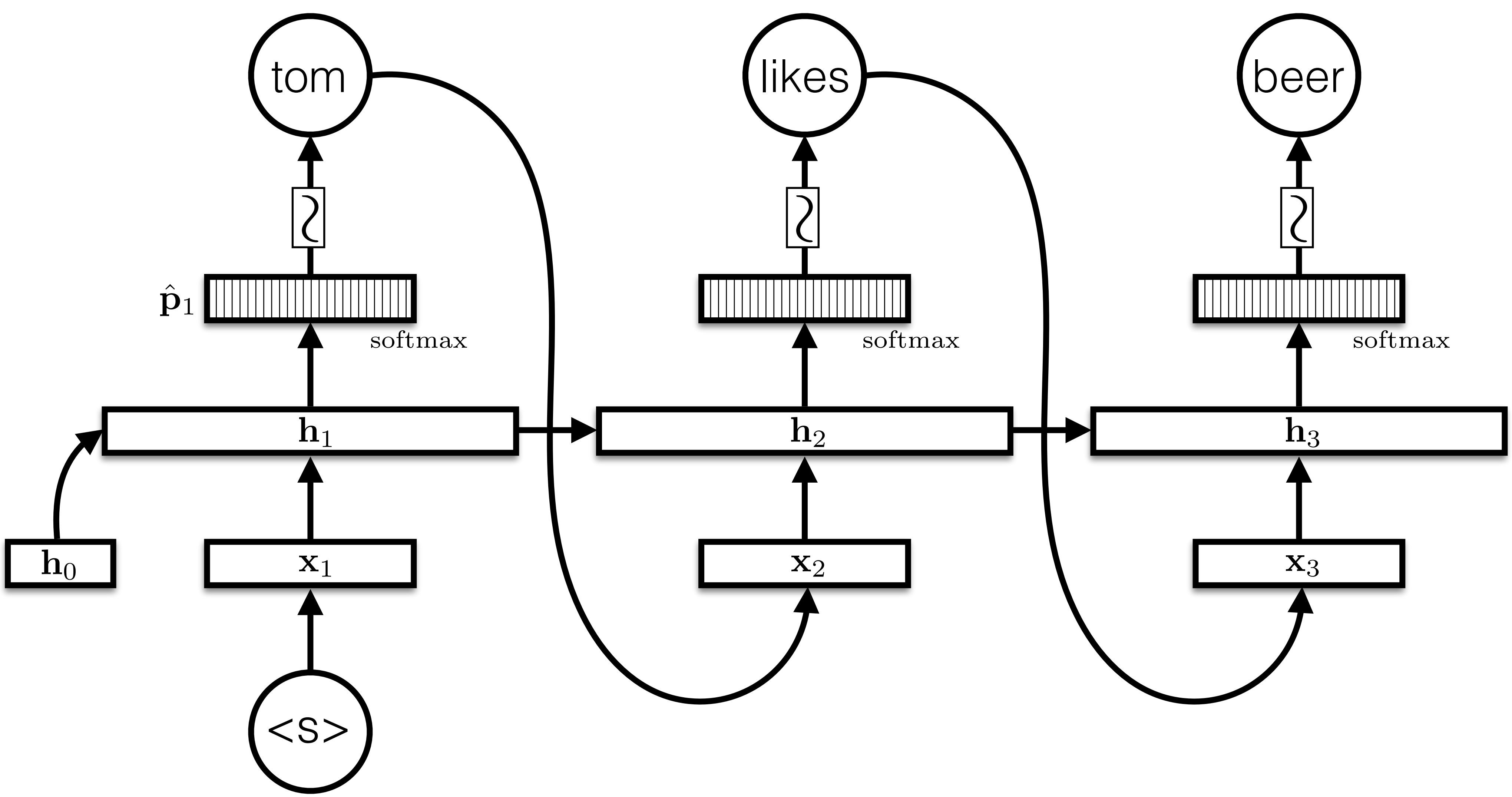
Example: Language Model

$$p(tom \mid \langle \mathbf{s} \rangle) \times p(likes \mid \langle \mathbf{s} \rangle, tom)$$



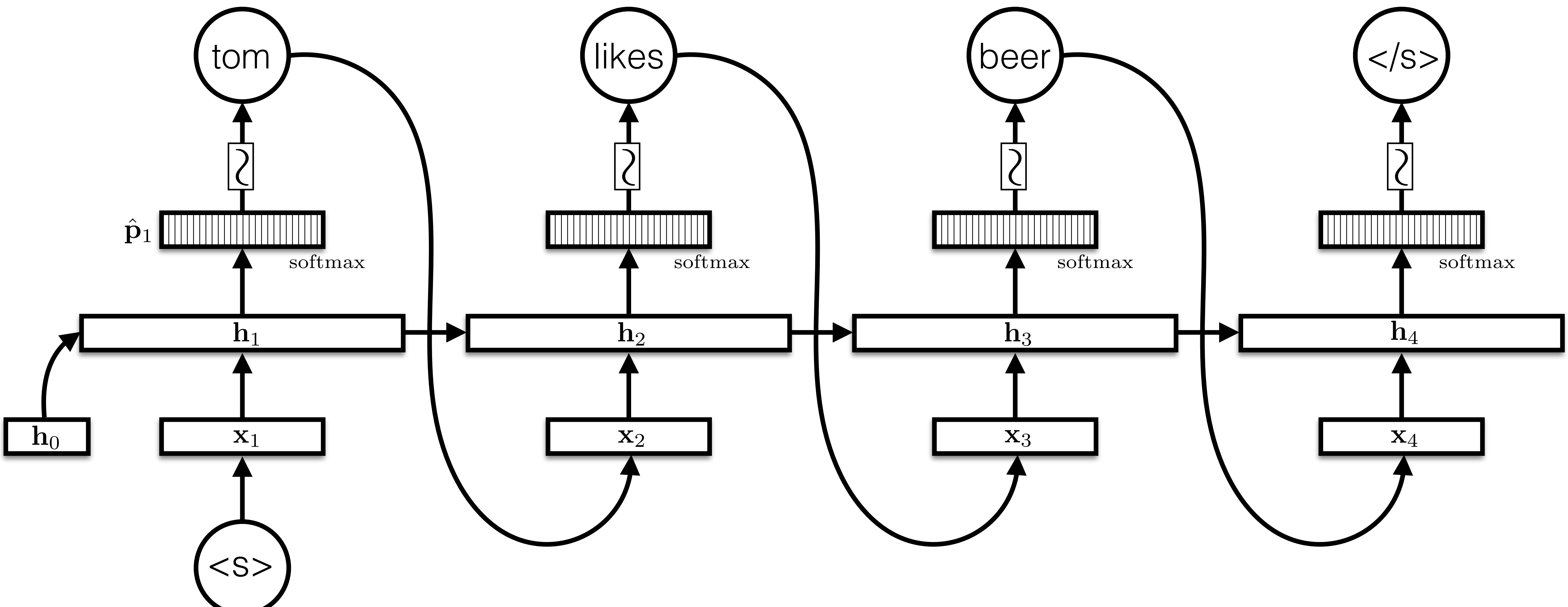
Example: Language Model

$$p(tom \mid \langle \mathbf{s} \rangle) \times p(likes \mid \langle \mathbf{s} \rangle, tom) \times p(beer \mid \langle \mathbf{s} \rangle, tom, likes)$$

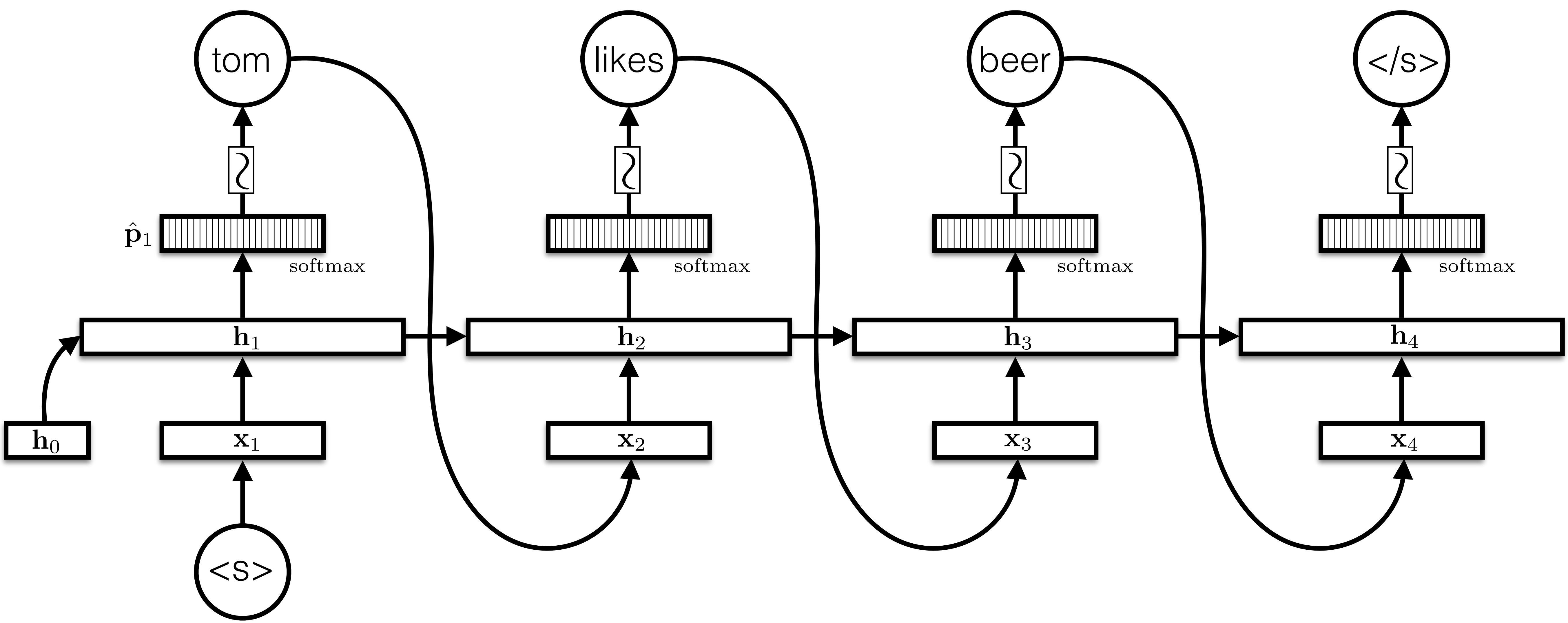


Example: Language Model

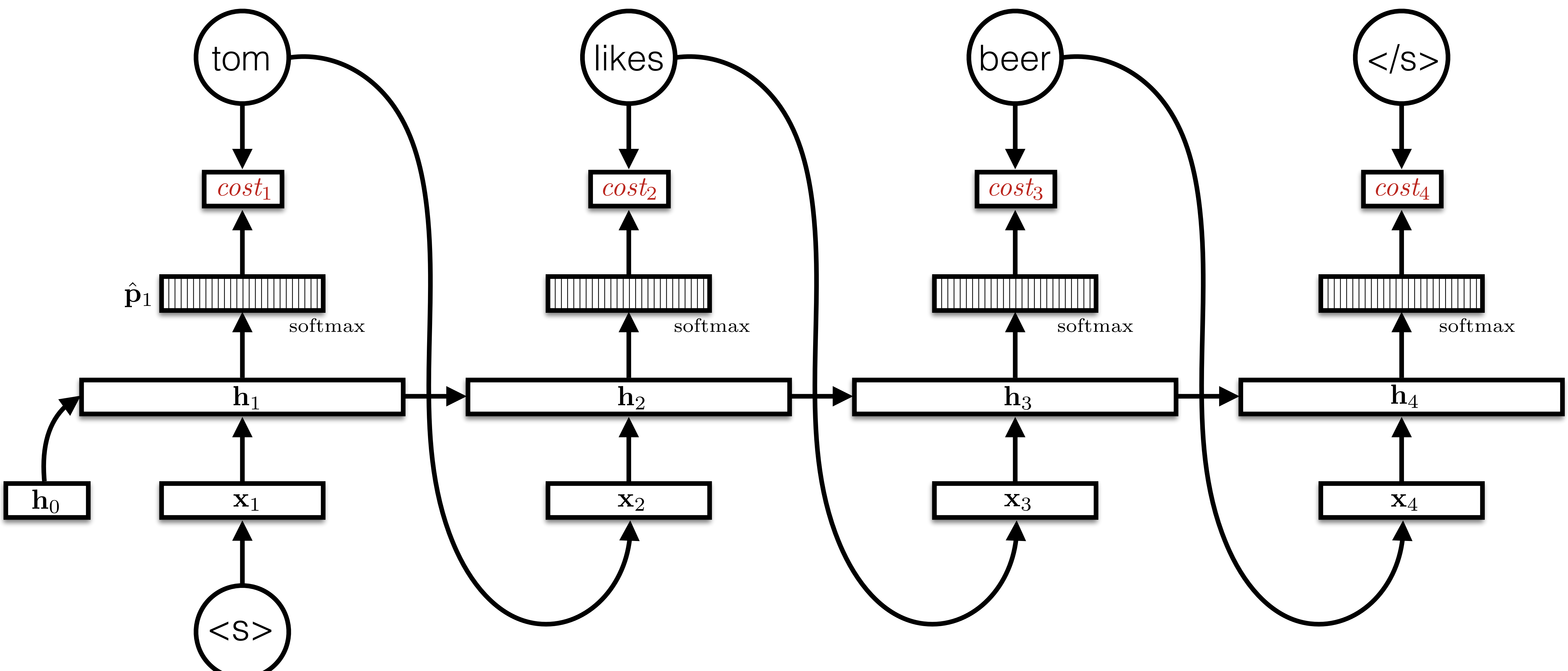
$$p(tom \mid \langle \mathbf{s} \rangle) \times p(likes \mid \langle \mathbf{s} \rangle, tom)$$
$$\times p(beer \mid \langle \mathbf{s} \rangle, tom, likes)$$
$$\times p(\langle / \mathbf{s} \rangle \mid \langle \mathbf{s} \rangle, tom, likes, beer)$$



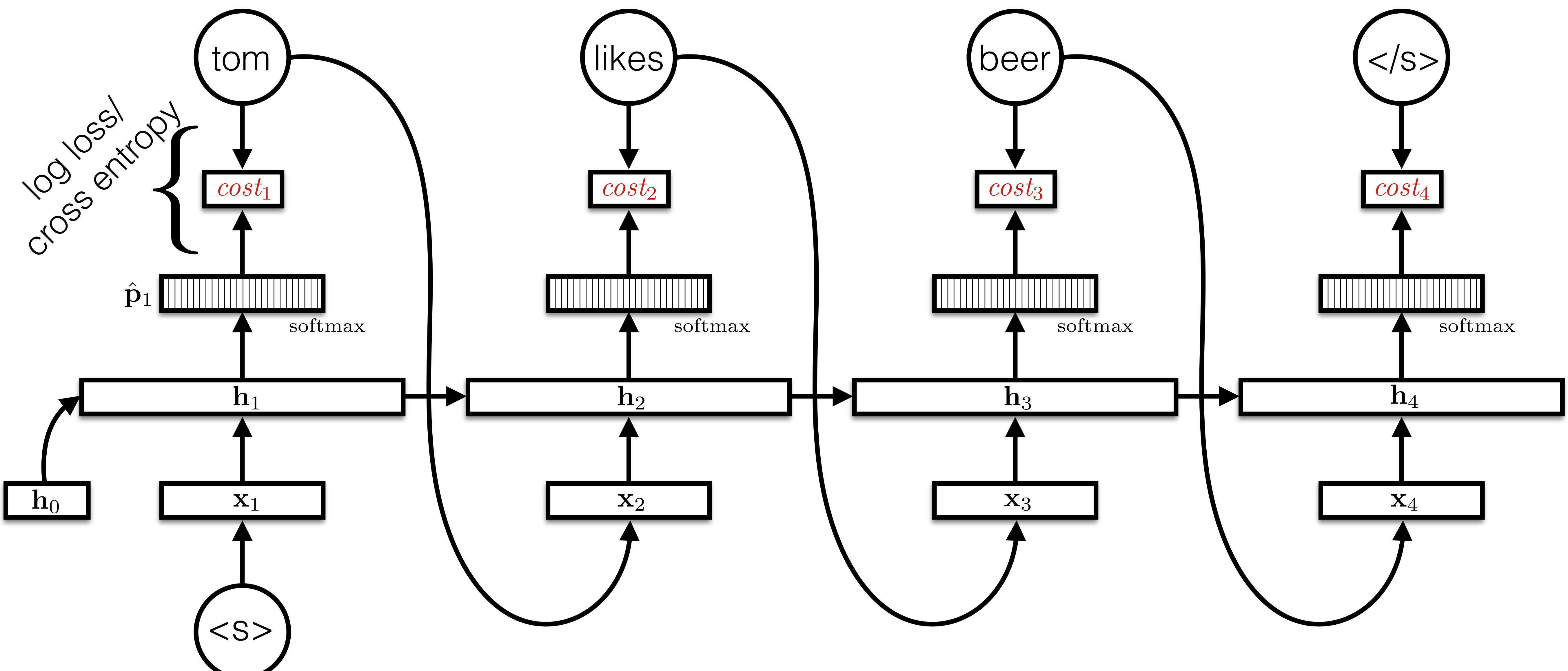
Language Model Training



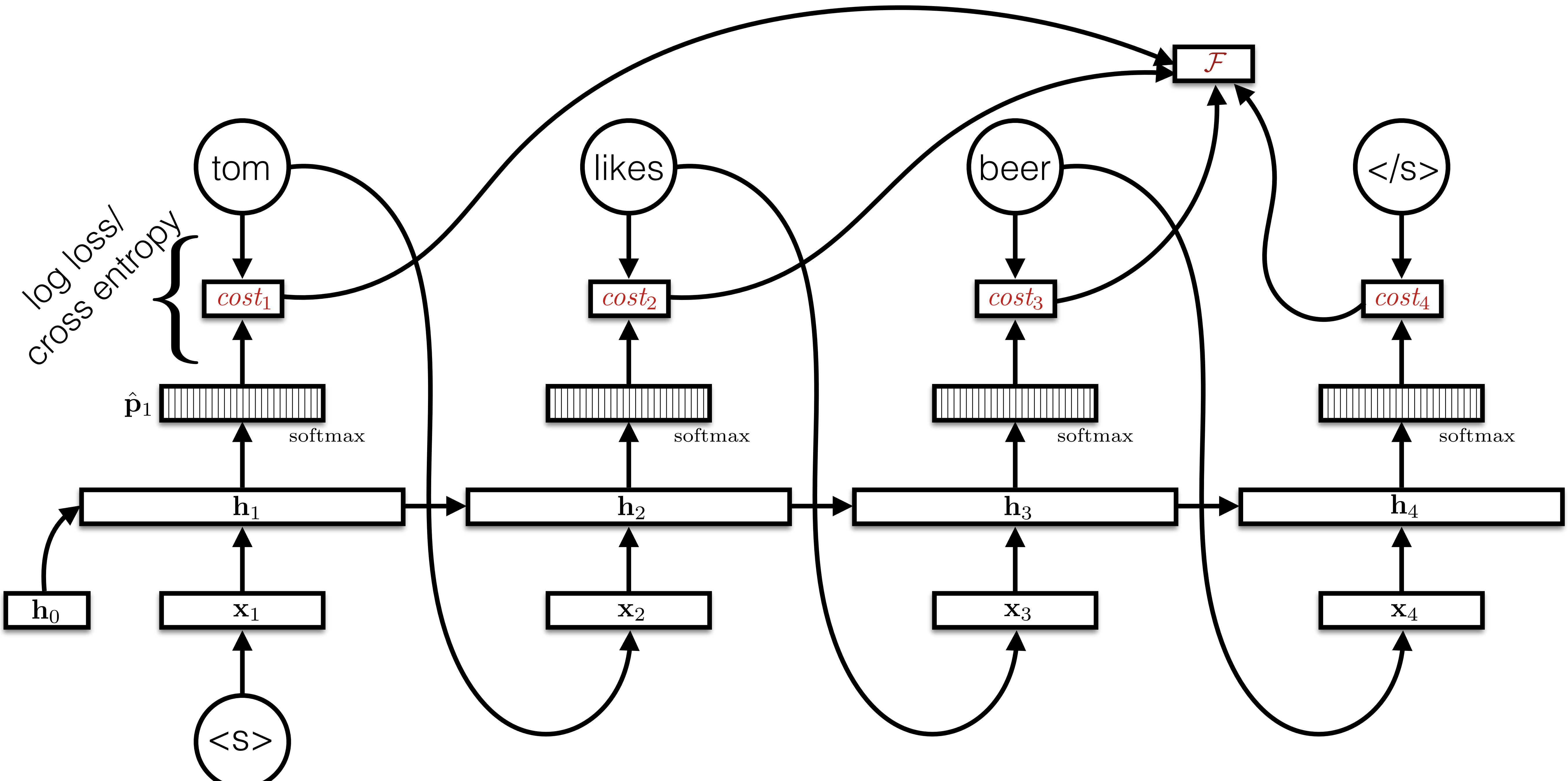
Language Model Training



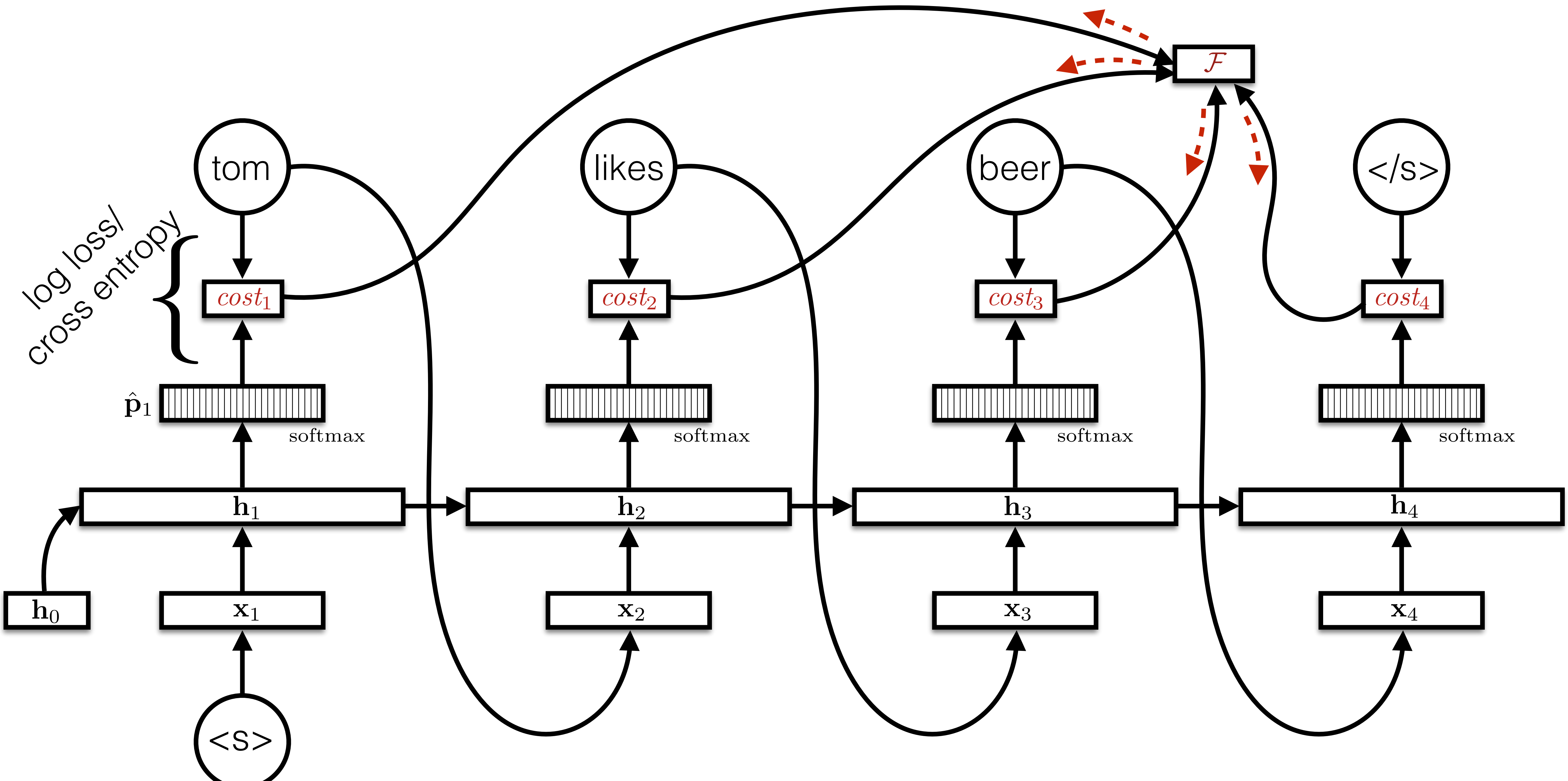
Language Model Training



Language Model Training



Language Model Training



RNN Language Models

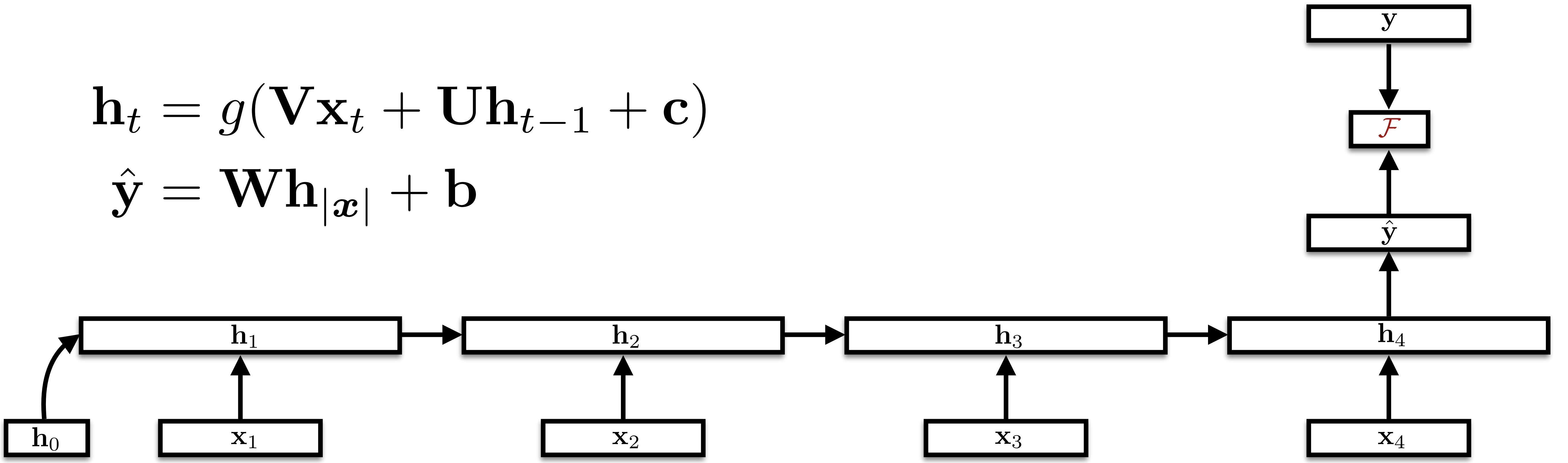
- Unlike Markov (*n*-gram) models, RNNs never forget
    - However we will see they might have trouble learning to use their memories (more soon…)
- Algorithms
    - Sample a sequence from the probability distribution defined by the RNN
    - Train the RNN to minimize cross entropy (aka MLE)
    - What about: what is the most probable sequence?

Questions?

Training Challenges

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$



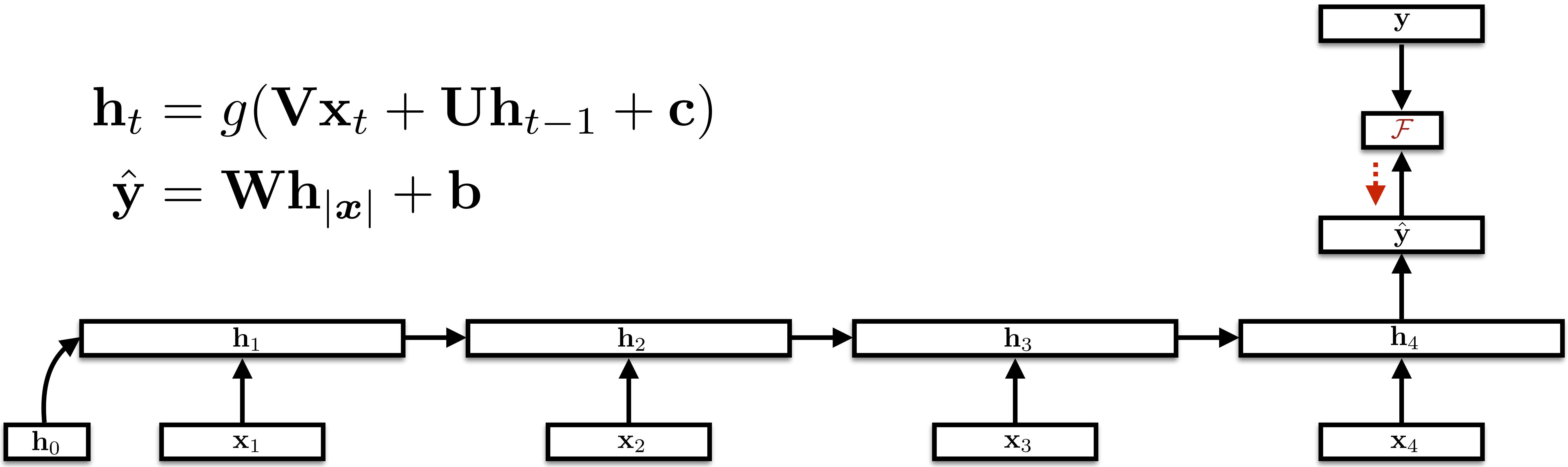
What happens to gradients as you go back in time?

$$\frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Training Challenges

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$



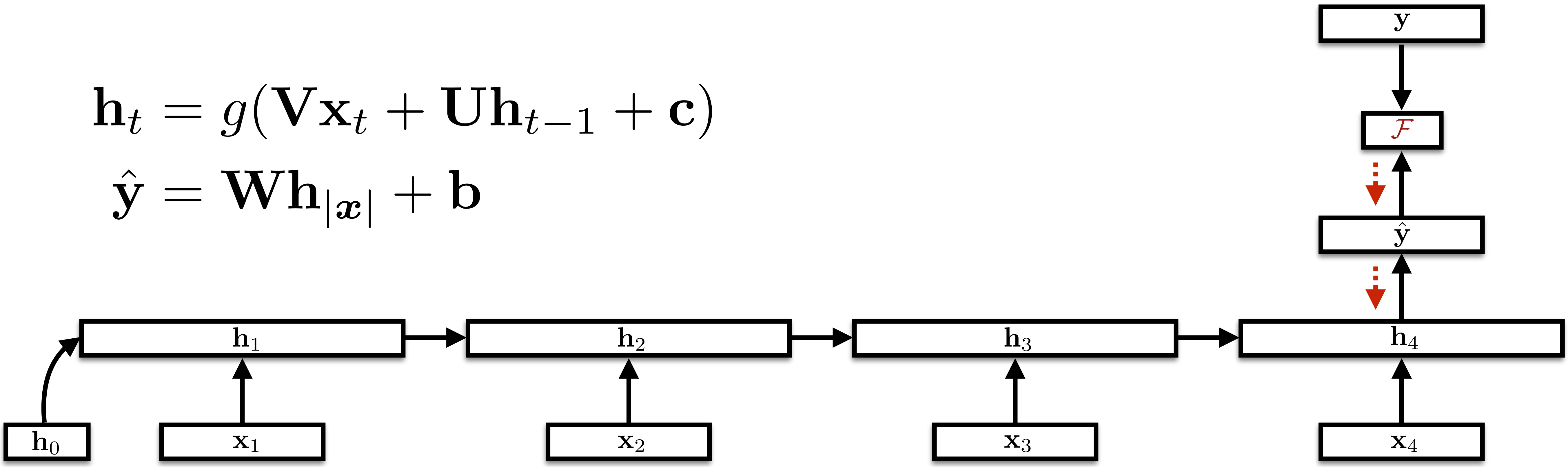
What happens to gradients as you go back in time?

$$\frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Training Challenges

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$



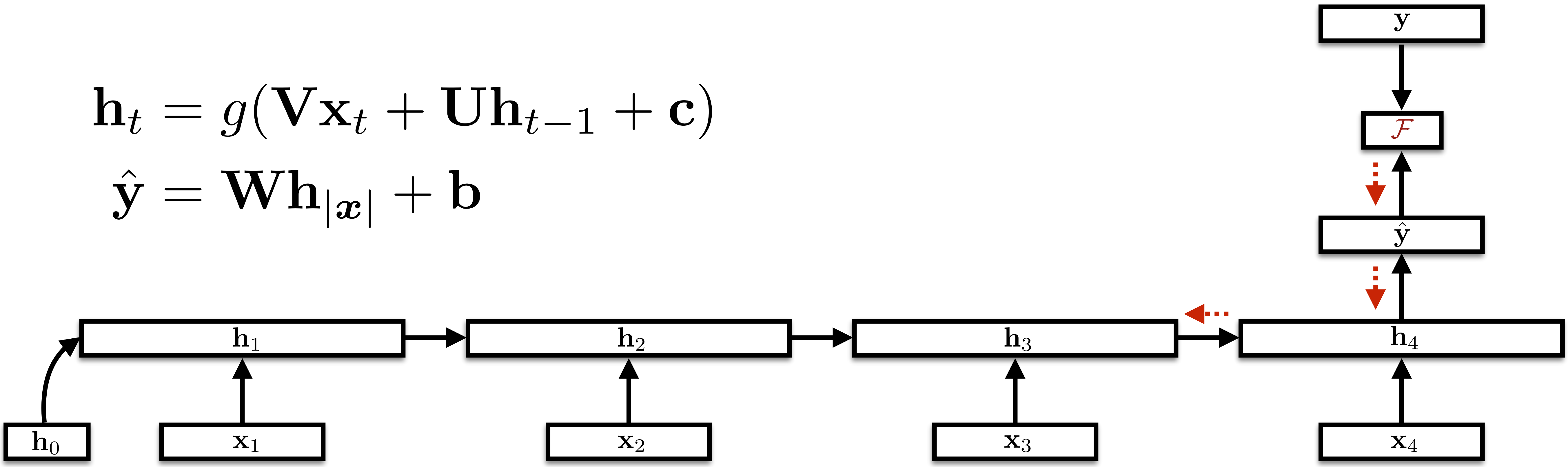
What happens to gradients as you go back in time?

$$\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_4} \frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Training Challenges

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$



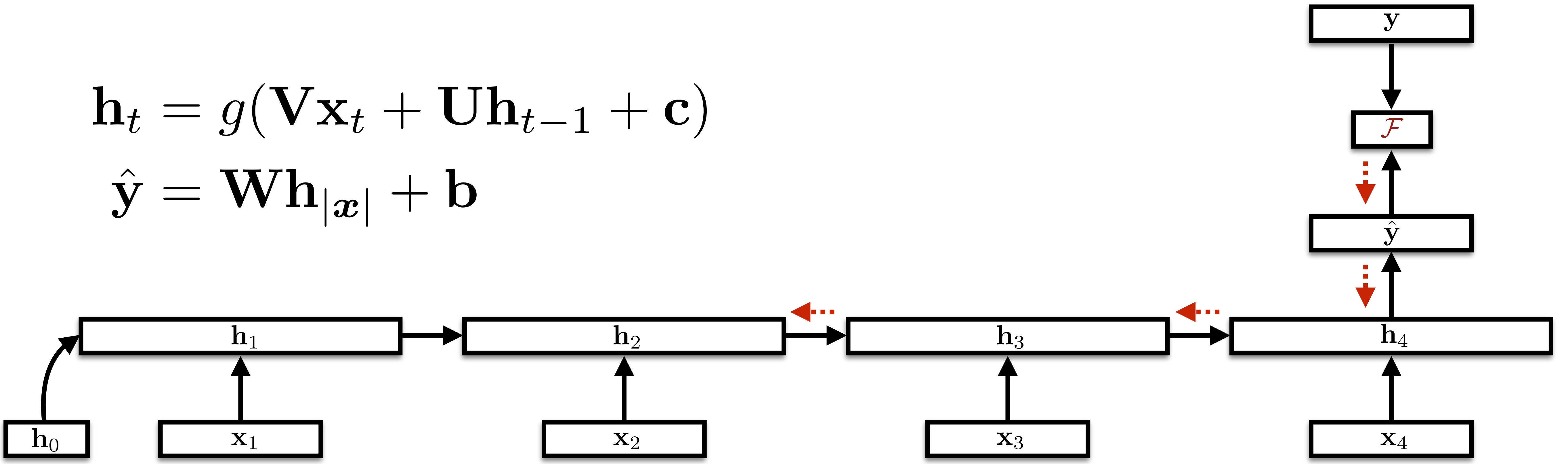
What happens to gradients as you go back in time?

$$\frac{\partial \mathbf{h}_4}{\partial \mathbf{h}_3} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_4} \frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Training Challenges

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$

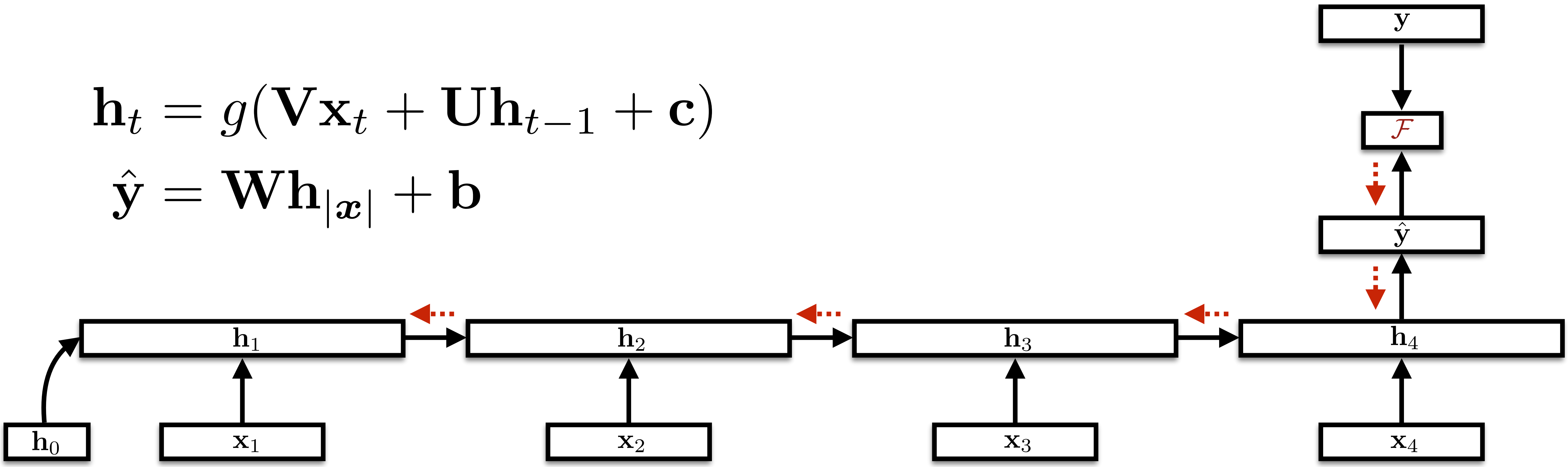


What happens to gradients as you go back in time?

$$\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_4}{\partial \mathbf{h}_3} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_4} \frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Training Challenges

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$



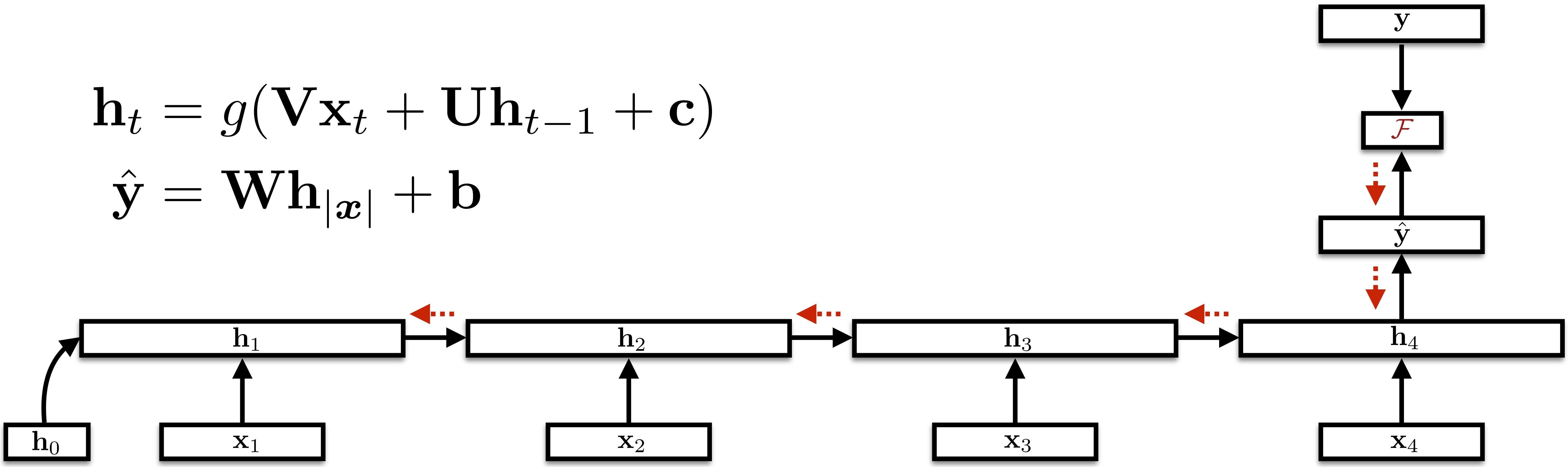
What happens to gradients as you go back in time?

$$\frac{\partial \mathcal{F}}{\partial \mathbf{h}_1} = \frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_4}{\partial \mathbf{h}_3}\;\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_4}\frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}}\frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Training Challenges

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$

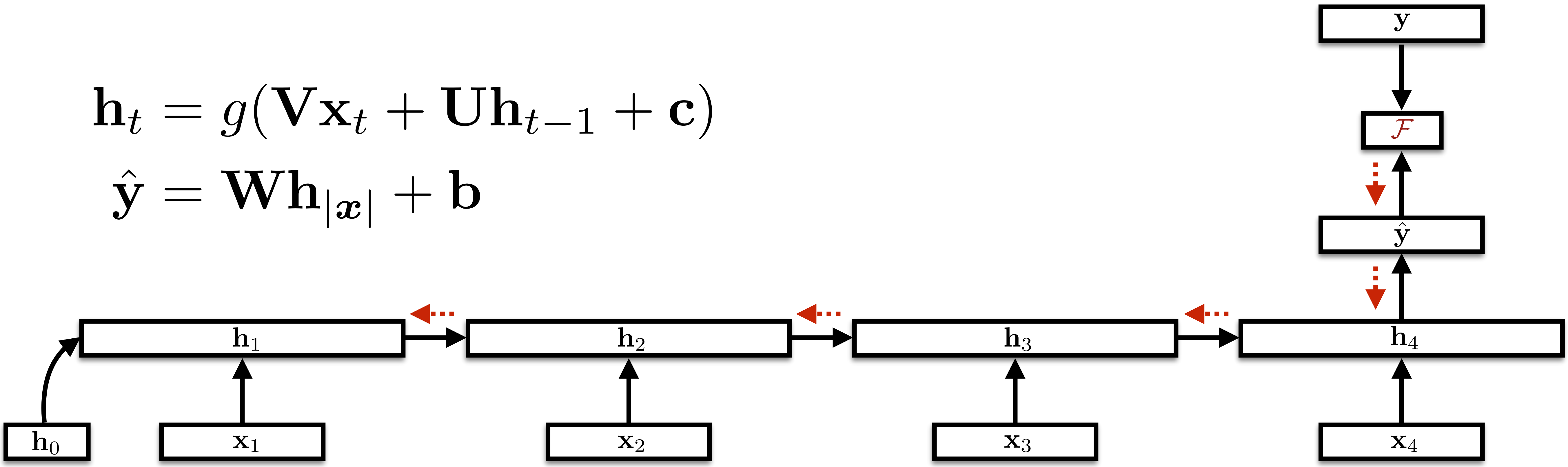


What happens to gradients as you go back in time?

$$\frac{\partial \mathcal{F}}{\partial \mathbf{h}_1} = \underbrace{\frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_4}{\partial \mathbf{h}_3}}_{\prod_{t=2}^{4} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_4}\frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}}\frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Training Challenges

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$



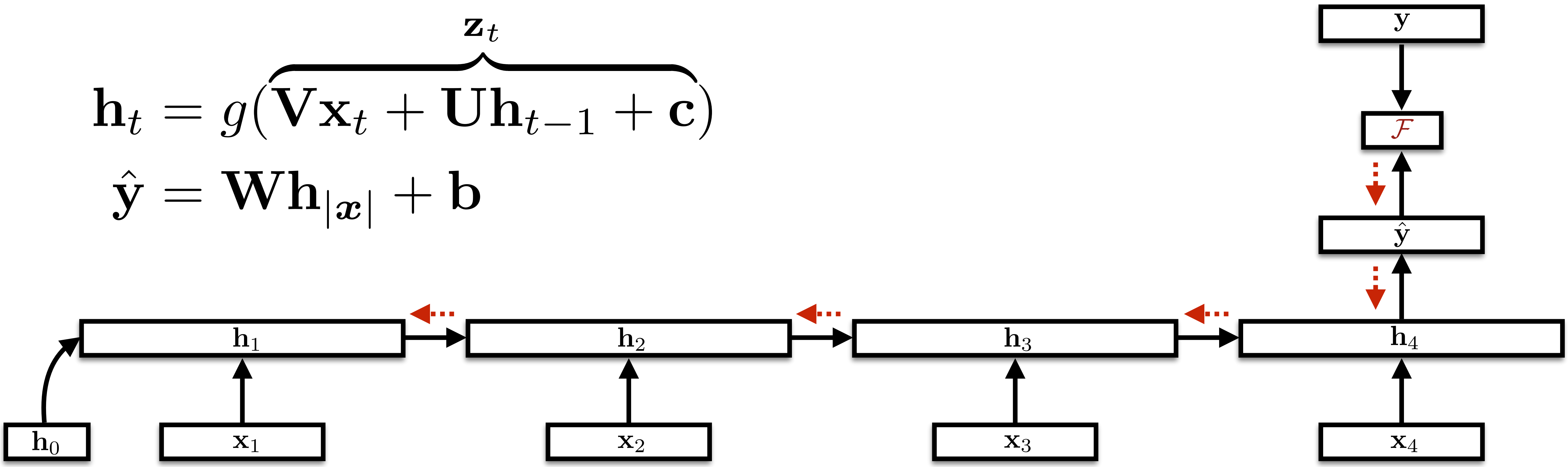
What happens to gradients as you go back in time?

$$\frac{\partial \mathcal{F}}{\partial \mathbf{h}_1} = \left(\prod_{t=2}^{|\boldsymbol{x}|} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \right) \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_{|\boldsymbol{x}|}} \frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Training Challenges

$$\mathbf{h}_t = g(\overbrace{\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c}}^{\mathbf{z}_t})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$



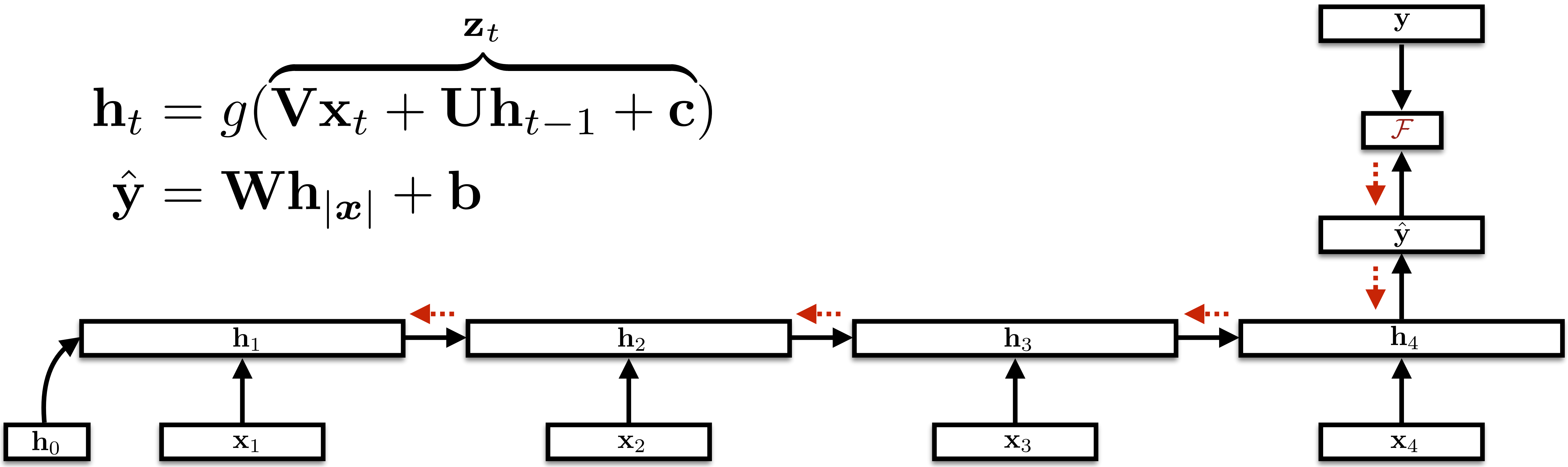
What happens to gradients as you go back in time?

$$\frac{\partial \mathcal{F}}{\partial \mathbf{h}_1} = \left(\prod_{t=2}^{|\boldsymbol{x}|} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \right) \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_{|\boldsymbol{x}|}} \frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Training Challenges

$$\mathbf{h}_t = g(\overbrace{\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c}}^{\mathbf{z}_t})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$



What happens to gradients as you go back in time?

$$\frac{\partial \mathcal{F}}{\partial \mathbf{h}_1} = \left(\prod_{t=2}^{|\boldsymbol{x}|} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} \right) \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_{|\boldsymbol{x}|}} \frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Training Challenges

$$\mathbf{h}_t = g(\overbrace{\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c}}^{\mathbf{z}_t})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$

$$\frac{\partial \mathcal{F}}{\partial \mathbf{h}_1} = \left(\prod_{t=2}^{|\boldsymbol{x}|} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} \right) \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_{|\boldsymbol{x}|}} \frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Training Challenges

$$\mathbf{h}_t = g(\overbrace{\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c}}^{\mathbf{z}_t})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$

$$\frac{\partial \mathcal{F}}{\partial \mathbf{h}_1} = \left(\prod_{t=2}^{|\boldsymbol{x}|} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} \right) \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_{|\boldsymbol{x}|}} \frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} = \mathrm{diag}(g'(\mathbf{z}_t))$$

Training Challenges

$$\mathbf{h}_t = g(\overbrace{\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c}}^{\mathbf{z}_t})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$

$$\frac{\partial \mathcal{F}}{\partial \mathbf{h}_1} = \left(\prod_{t=2}^{|\boldsymbol{x}|} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} \right) \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_{|\boldsymbol{x}|}} \frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} = \mathrm{diag}(g'(\mathbf{z}_t))$$

$$\frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} = \boxed{\textbf{?}}$$

Training Challenges

$$\mathbf{h}_t = g(\overbrace{\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c}}^{\mathbf{z}_t})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$

$$\frac{\partial \mathcal{F}}{\partial \mathbf{h}_1} = \left(\prod_{t=2}^{|\boldsymbol{x}|} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} \right) \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_{|\boldsymbol{x}|}} \frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} = \mathrm{diag}(g'(\mathbf{z}_t))$$

$$\frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} = \mathbf{U}$$

Training Challenges

$$\mathbf{h}_t = g(\overbrace{\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c}}^{\mathbf{z}_t})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$

$$\frac{\partial \mathcal{F}}{\partial \mathbf{h}_1} = \left(\prod_{t=2}^{|\boldsymbol{x}|} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} \right) \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_{|\boldsymbol{x}|}} \frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} = \mathrm{diag}(g'(\mathbf{z}_t))$$

$$\frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} = \mathbf{U}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} = \mathrm{diag}(g'(\mathbf{z}_t))\mathbf{U}$$

Training Challenges

$$\mathbf{h}_t = g(\overbrace{\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c}}^{\mathbf{z}_t})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$

$$\frac{\partial \mathcal{F}}{\partial \mathbf{h}_1} = \left(\prod_{t=2}^{|\boldsymbol{x}|} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} \right) \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_{|\boldsymbol{x}|}} \frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

$$\frac{\partial \mathcal{F}}{\partial \mathbf{h}_1} = \left(\prod_{t=2}^{|\boldsymbol{x}|} \mathrm{diag}(g'(\mathbf{z}_t))\mathbf{U} \right) \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_{|\boldsymbol{x}|}} \frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Training Challenges

$$\mathbf{h}_t = g(\overbrace{\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c}}^{\mathbf{z}_t})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$

$$\frac{\partial \mathcal{F}}{\partial \mathbf{h}_1} = \left(\prod_{t=2}^{|\boldsymbol{x}|} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} \right) \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_{|\boldsymbol{x}|}} \frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

$$\frac{\partial \mathcal{F}}{\partial \mathbf{h}_1} = \left(\prod_{t=2}^{|\boldsymbol{x}|} \mathrm{diag}(g'(\mathbf{z}_t))\mathbf{U} \right) \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_{|\boldsymbol{x}|}} \frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Three cases: largest eigenvalue is

exactly 1; gradient propagation is stable

<1; gradient vanishes (exponential decay)

>1; gradient explodes (exponential growth)

Vanishing Gradients

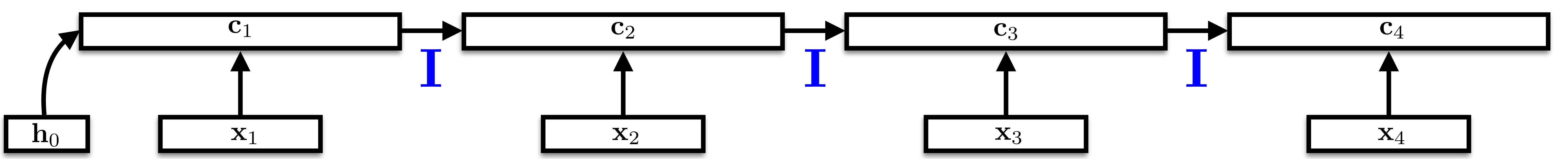
- In practice, the spectral radius of **U** is small, and gradients vanish
- In practice, this means that long-range dependencies are difficult to learn (although in theory they are learnable)
- Solutions
    - Better optimizers (second order methods, approximate second order methods)
    - Normalization to keep the gradient norms stable across time
    - Clever initialization so that you at least start with good spectra (e.g., start with random orthonormal matrices)
    - **Alternative parameterizations: LSTMs and GRUs**

Alternative RNNs

- Long short-term memories (LSTMs; Hochreiter and Schmidthuber, 1997)
- Gated recurrent units (GRUs; Cho et al., 2014)
- Intuition instead of **multiplying** across time (which leads to exponential growth), we want the error to be **constant**.
    - What is a function whose Jacobian has a spectral radius of exactly **I**: the identity function

Memory cells

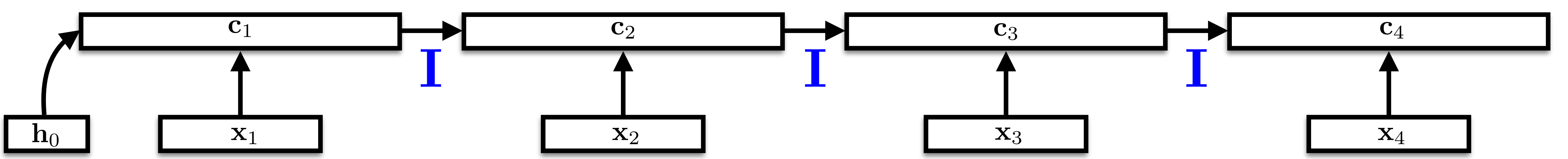
$$\mathbf{c}_t = \mathbf{c}_{t-1} + f(\mathbf{x}_t)$$



Memory cells

$$\mathbf{c}_t = \mathbf{c}_{t-1} + f(\mathbf{x}_t)$$

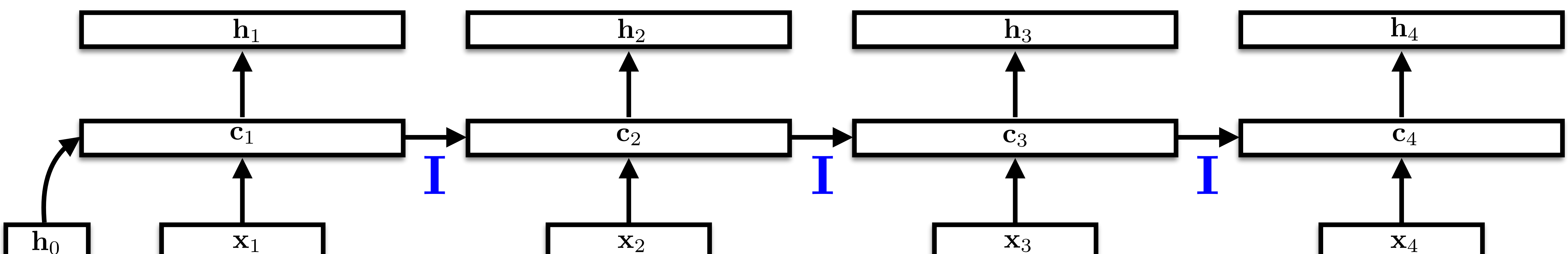
$$f(\mathbf{v}) = \tanh(\mathbf{W}\mathbf{v} + \mathbf{b})$$



Memory cells

$$\mathbf{c}_t = \mathbf{c}_{t-1} + f(\mathbf{x}_t) \qquad f(\mathbf{v}) = \tanh(\mathbf{W}\mathbf{v} + \mathbf{b})$$

$$\mathbf{h}_t = g(\mathbf{c}_t)$$

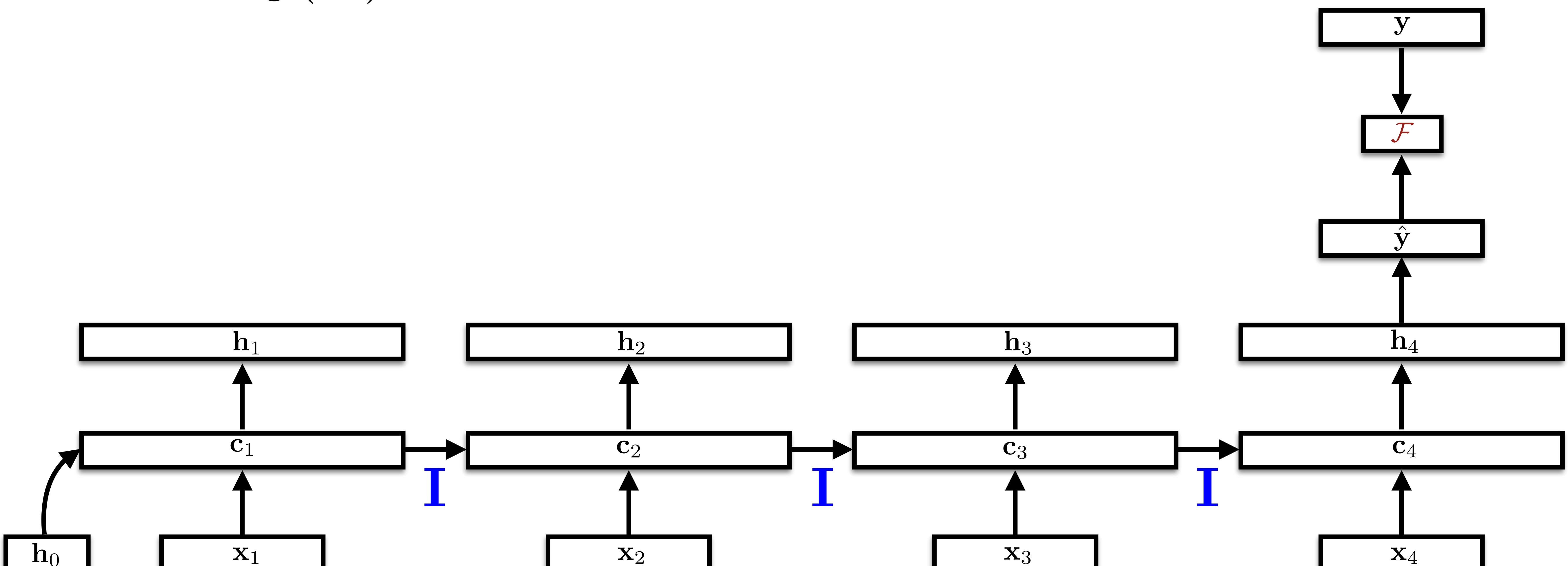


Memory cells

$$\mathbf{c}_t = \mathbf{c}_{t-1} + f(\mathbf{x}_t)$$

$$f(\mathbf{v}) = \tanh(\mathbf{W}\mathbf{v} + \mathbf{b})$$

$$\mathbf{h}_t = g(\mathbf{c}_t)$$



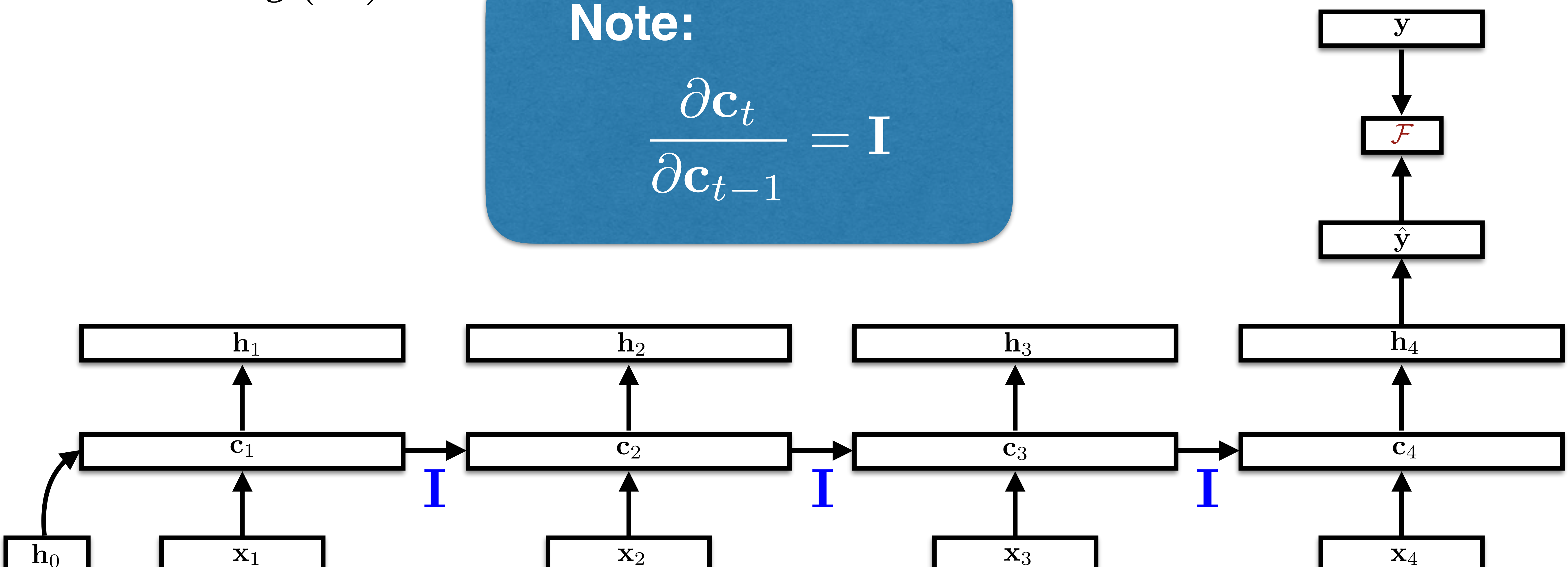
Memory cells

$$\mathbf{c}_t = \mathbf{c}_{t-1} + f(\mathbf{x}_t)$$
$$\mathbf{h}_t = g(\mathbf{c}_t)$$

$$f(\mathbf{v}) = \tanh(\mathbf{W}\mathbf{v} + \mathbf{b})$$

Note:

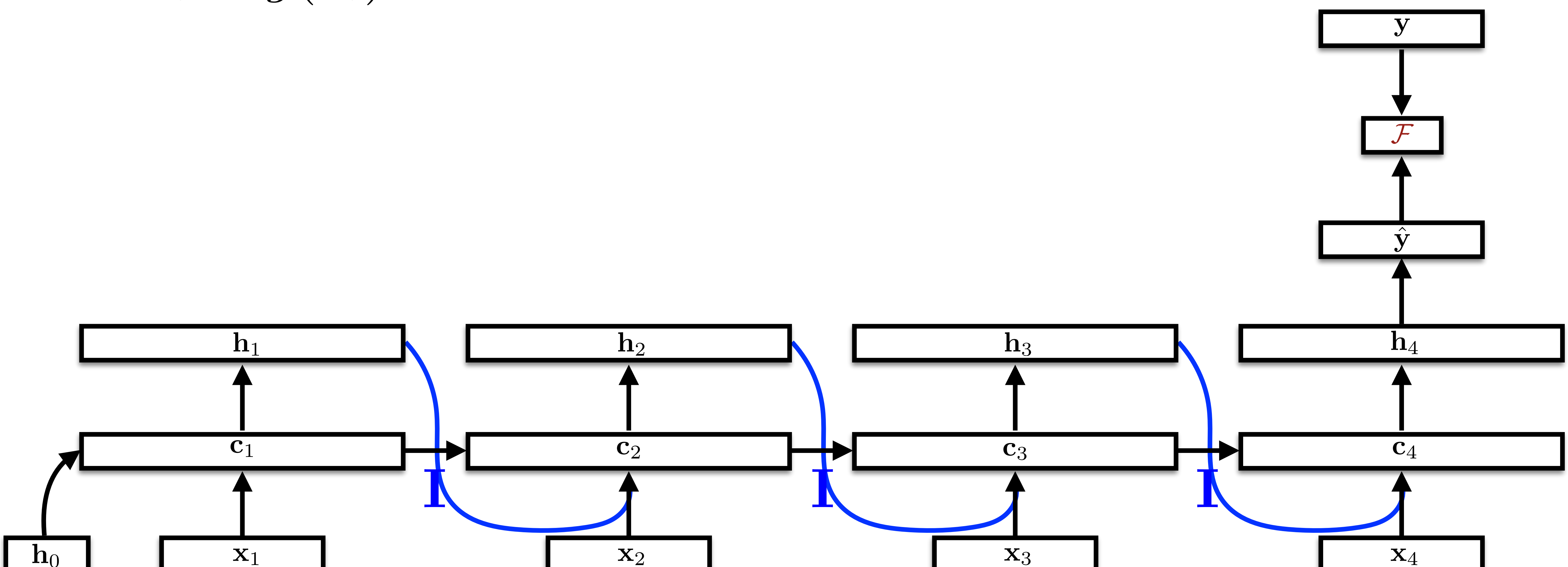
$$\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = \mathbf{I}$$



Memory cells

$$\mathbf{c}_t = \mathbf{c}_{t-1} + f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

$$\mathbf{h}_t = g(\mathbf{c}_t)$$



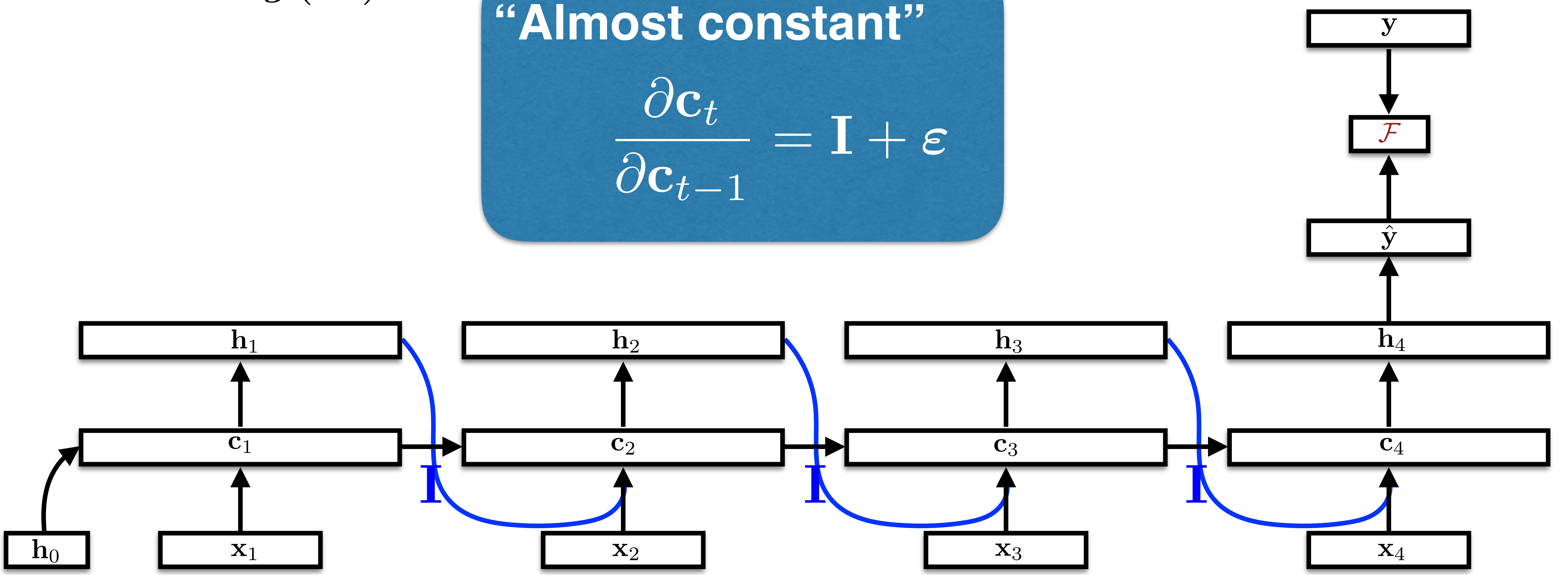
Memory cells

$$\mathbf{c}_t = \mathbf{c}_{t-1} + f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

$$\mathbf{h}_t = g(\mathbf{c}_t)$$

"Almost constant"

$$\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = \mathbf{I} + \varepsilon$$



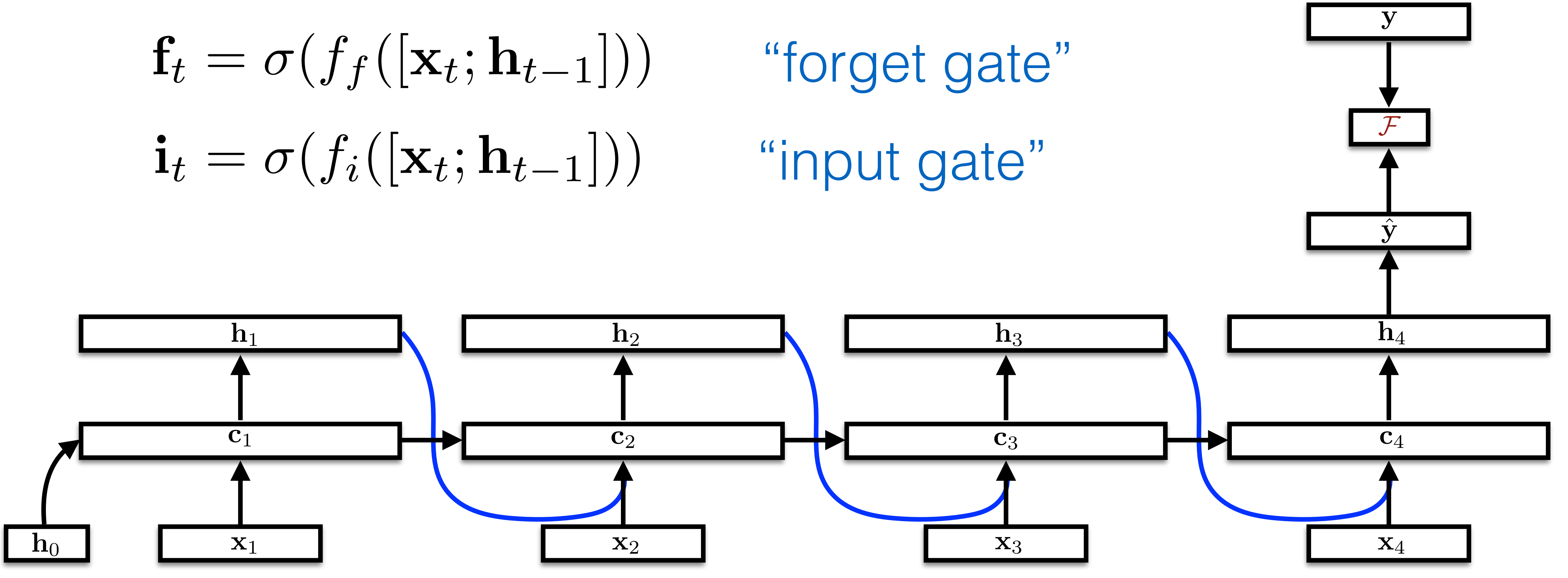
Memory cells

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

$$\mathbf{h}_t = g(\mathbf{c}_t)$$

$$\mathbf{f}_t = \sigma(f_f([\mathbf{x}_t; \mathbf{h}_{t-1}]))$$ "forget gate"

$$\mathbf{i}_t = \sigma(f_i([\mathbf{x}_t; \mathbf{h}_{t-1}]))$$ "input gate"



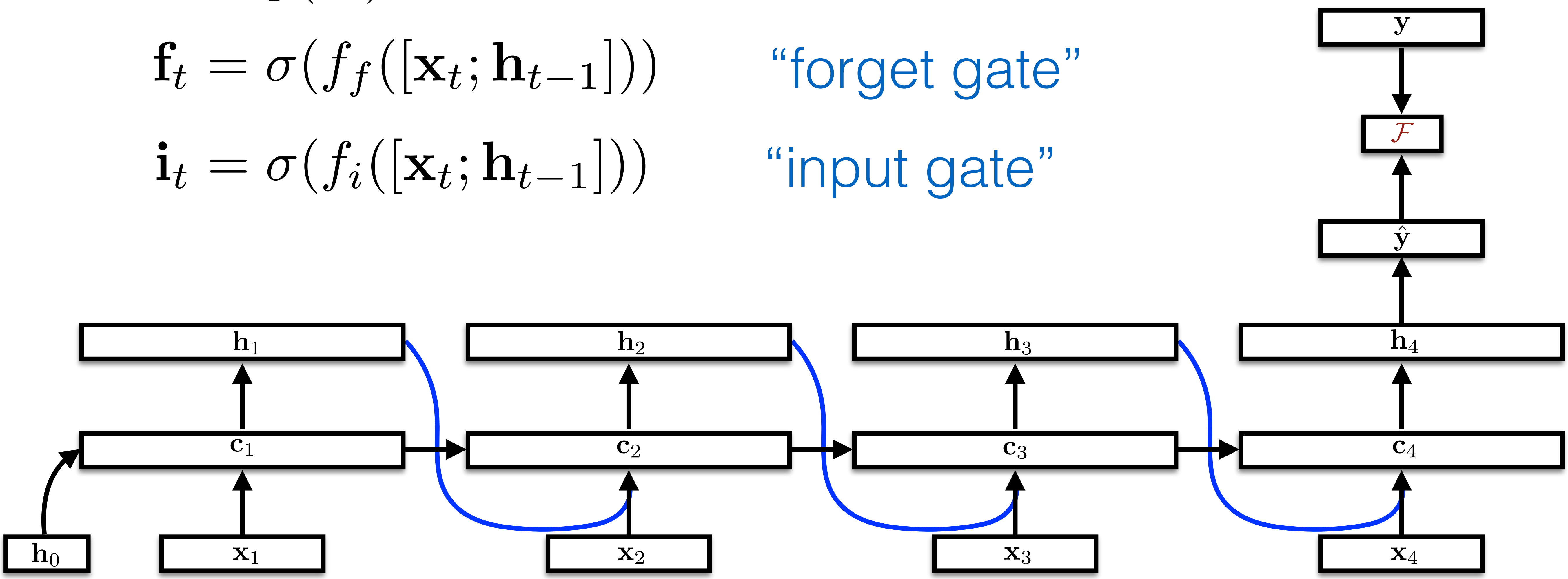
Memory cells

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

$$\mathbf{h}_t = g(\mathbf{c}_t)$$

$$\mathbf{f}_t = \sigma(f_f([\mathbf{x}_t; \mathbf{h}_{t-1}]))$$ "forget gate"

$$\mathbf{i}_t = \sigma(f_i([\mathbf{x}_t; \mathbf{h}_{t-1}]))$$ "input gate"



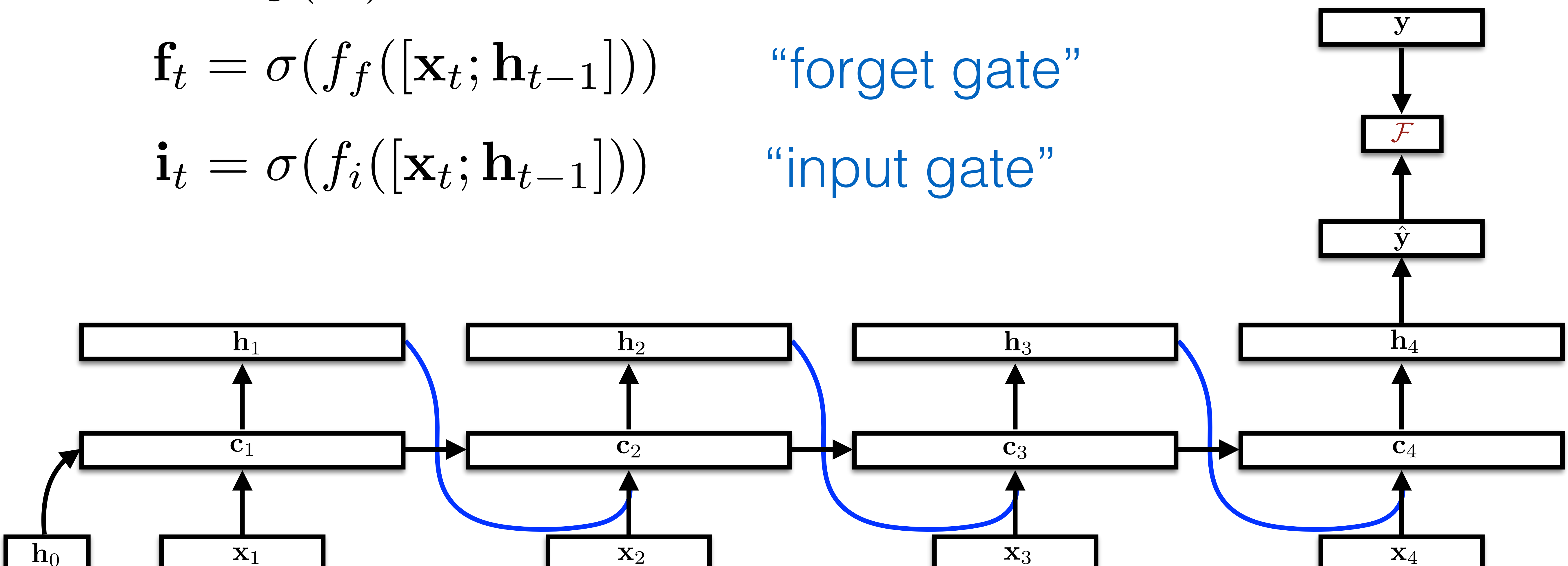
Memory cells

$$\mathbf{c}_t = \boxed{\mathbf{f}_t \odot} \mathbf{c}_{t-1} + \boxed{\mathbf{i}_t \odot} f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

$$\mathbf{h}_t = g(\mathbf{c}_t)$$

$$\mathbf{f}_t = \sigma(f_f([\mathbf{x}_t; \mathbf{h}_{t-1}]))$$ "forget gate"

$$\mathbf{i}_t = \sigma(f_i([\mathbf{x}_t; \mathbf{h}_{t-1}]))$$ "input gate"



LSTM

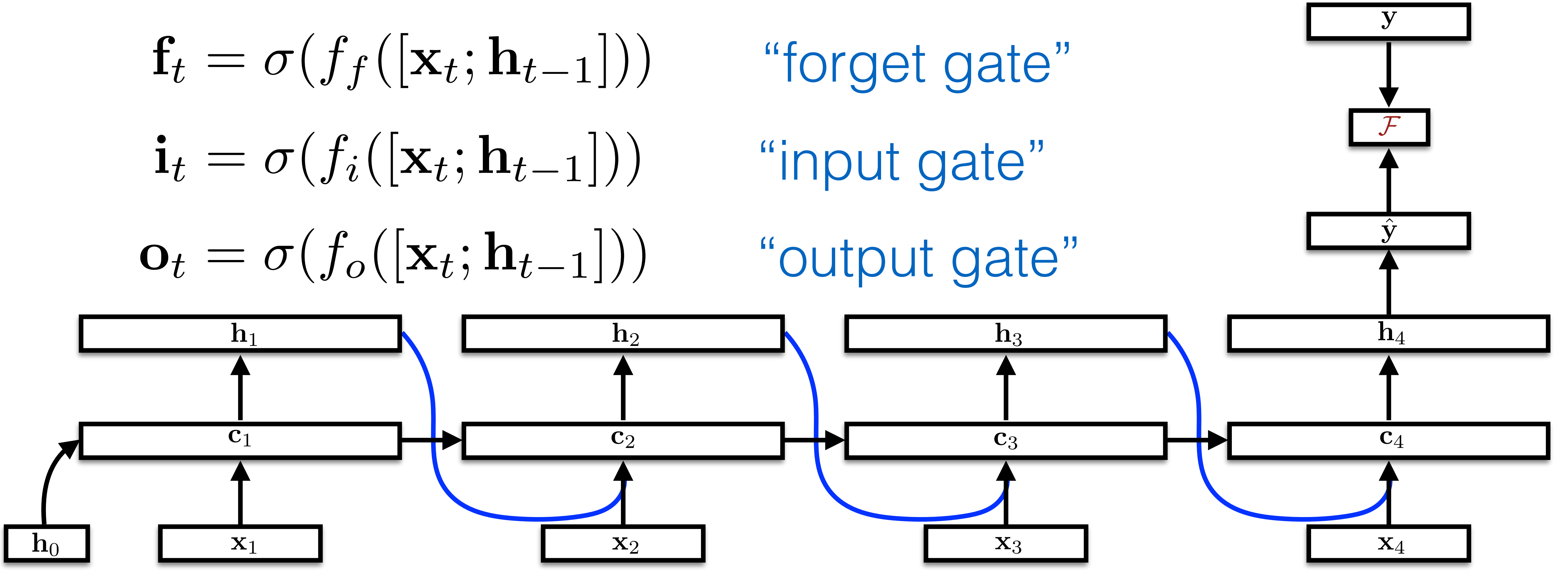
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

$$\mathbf{h}_t = \mathbf{o}_t \odot g(\mathbf{c}_t)$$

$$\mathbf{f}_t = \sigma(f_f([\mathbf{x}_t; \mathbf{h}_{t-1}]))$$ "forget gate"

$$\mathbf{i}_t = \sigma(f_i([\mathbf{x}_t; \mathbf{h}_{t-1}]))$$ "input gate"

$$\mathbf{o}_t = \sigma(f_o([\mathbf{x}_t; \mathbf{h}_{t-1}]))$$ "output gate"



LSTM

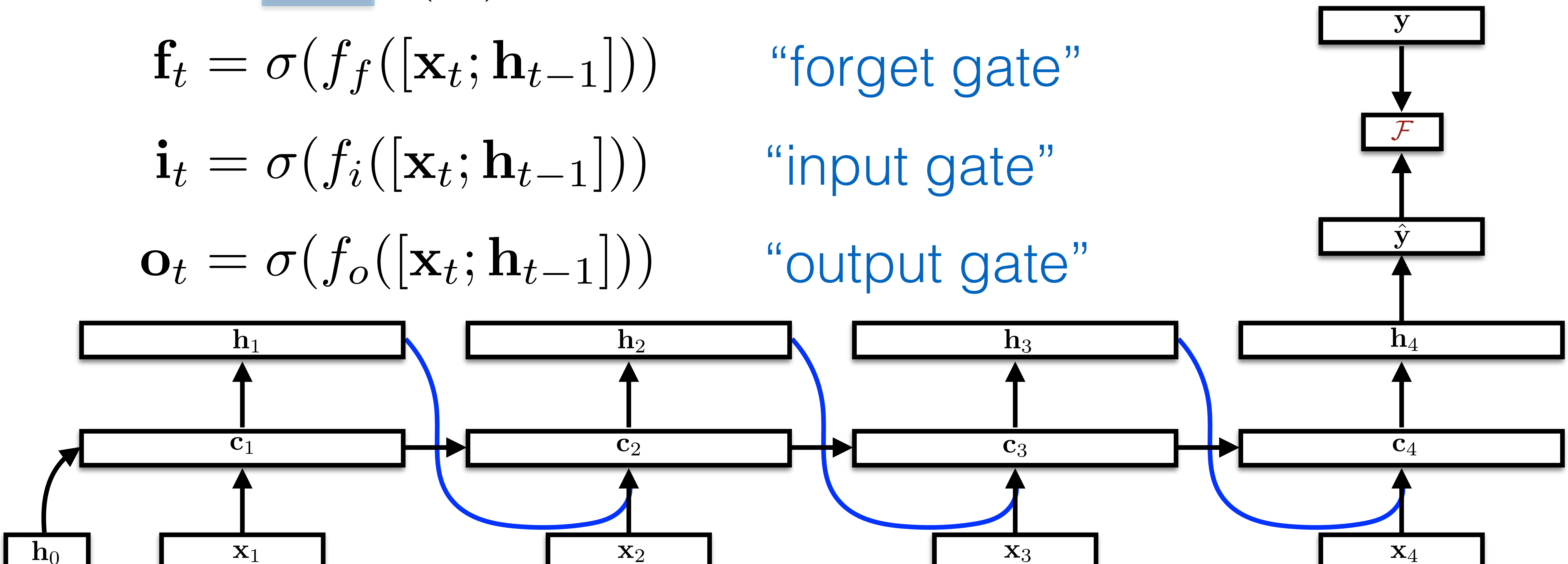
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

$$\mathbf{h}_t = \boxed{\mathbf{o}_t \odot} \, g(\mathbf{c}_t)$$

$$\mathbf{f}_t = \sigma(f_f([\mathbf{x}_t; \mathbf{h}_{t-1}]))$$ "forget gate"

$$\mathbf{i}_t = \sigma(f_i([\mathbf{x}_t; \mathbf{h}_{t-1}]))$$ "input gate"

$$\mathbf{o}_t = \sigma(f_o([\mathbf{x}_t; \mathbf{h}_{t-1}]))$$ "output gate"



LSTM Variant

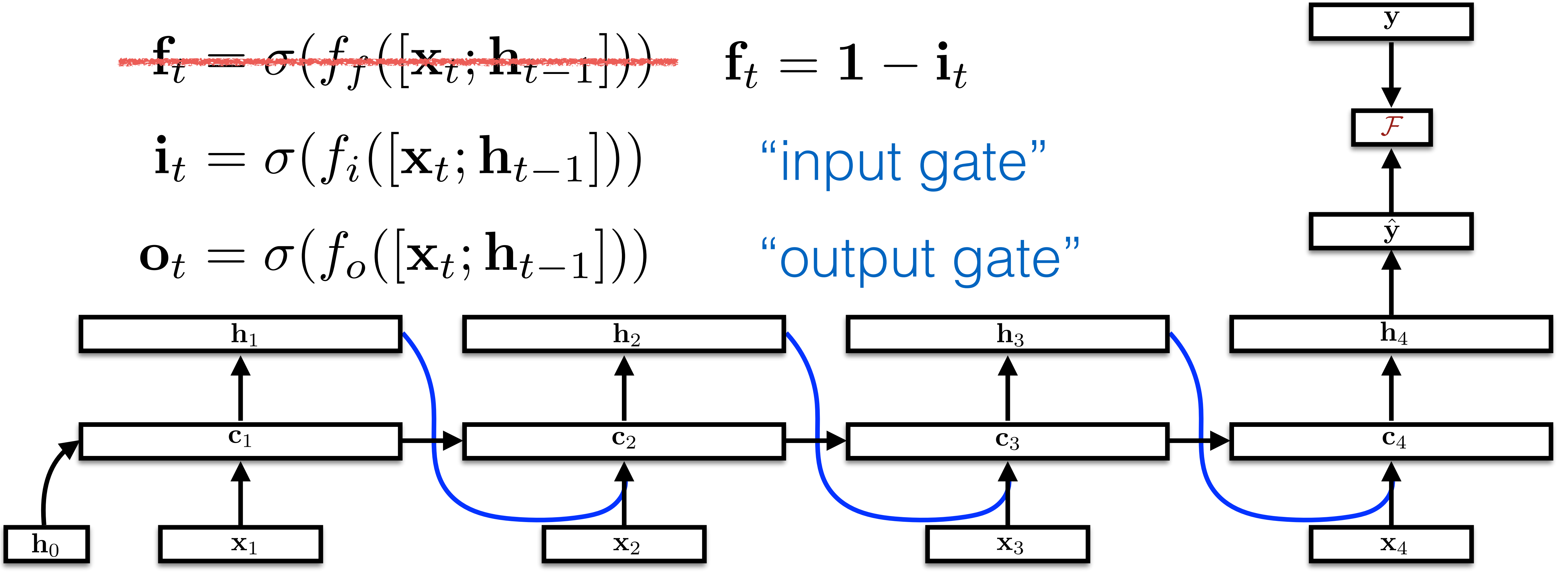
$$\mathbf{c}_t = (\mathbf{1} - \mathbf{i}_t) \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

$$\mathbf{h}_t = \mathbf{o}_t \odot g(\mathbf{c}_t)$$

$$\cancel{\mathbf{f}_t = \sigma(f_f([\mathbf{x}_t; \mathbf{h}_{t-1}]))} \qquad \mathbf{f}_t = \mathbf{1} - \mathbf{i}_t$$

$$\mathbf{i}_t = \sigma(f_i([\mathbf{x}_t; \mathbf{h}_{t-1}]))$$ "input gate"

$$\mathbf{o}_t = \sigma(f_o([\mathbf{x}_t; \mathbf{h}_{t-1}]))$$ "output gate"



LSTM Variant

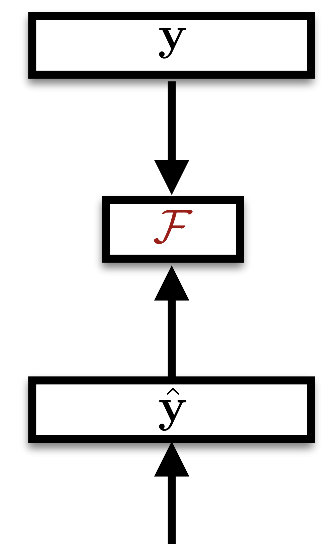
$$\mathbf{c}_t = \boxed{(\mathbf{1} - \mathbf{i}_t) \odot} \mathbf{c}_{t-1} + \boxed{\mathbf{i}_t \odot} f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

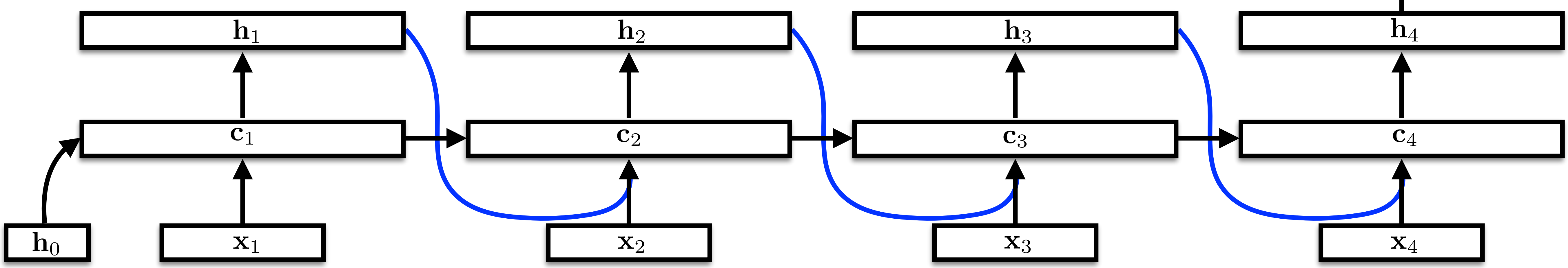
$$\mathbf{h}_t = \mathbf{o}_t \odot g(\mathbf{c}_t)$$

$$\cancel{\mathbf{f}_t = \sigma(f_f([\mathbf{x}_t; \mathbf{h}_{t-1}]))} \qquad \mathbf{f}_t = \mathbf{1} - \mathbf{i}_t$$

$$\mathbf{i}_t = \sigma(f_i([\mathbf{x}_t; \mathbf{h}_{t-1}]))$$ “input gate”

$$\mathbf{o}_t = \sigma(f_o([\mathbf{x}_t; \mathbf{h}_{t-1}]))$$ “output gate”





Another Visualization

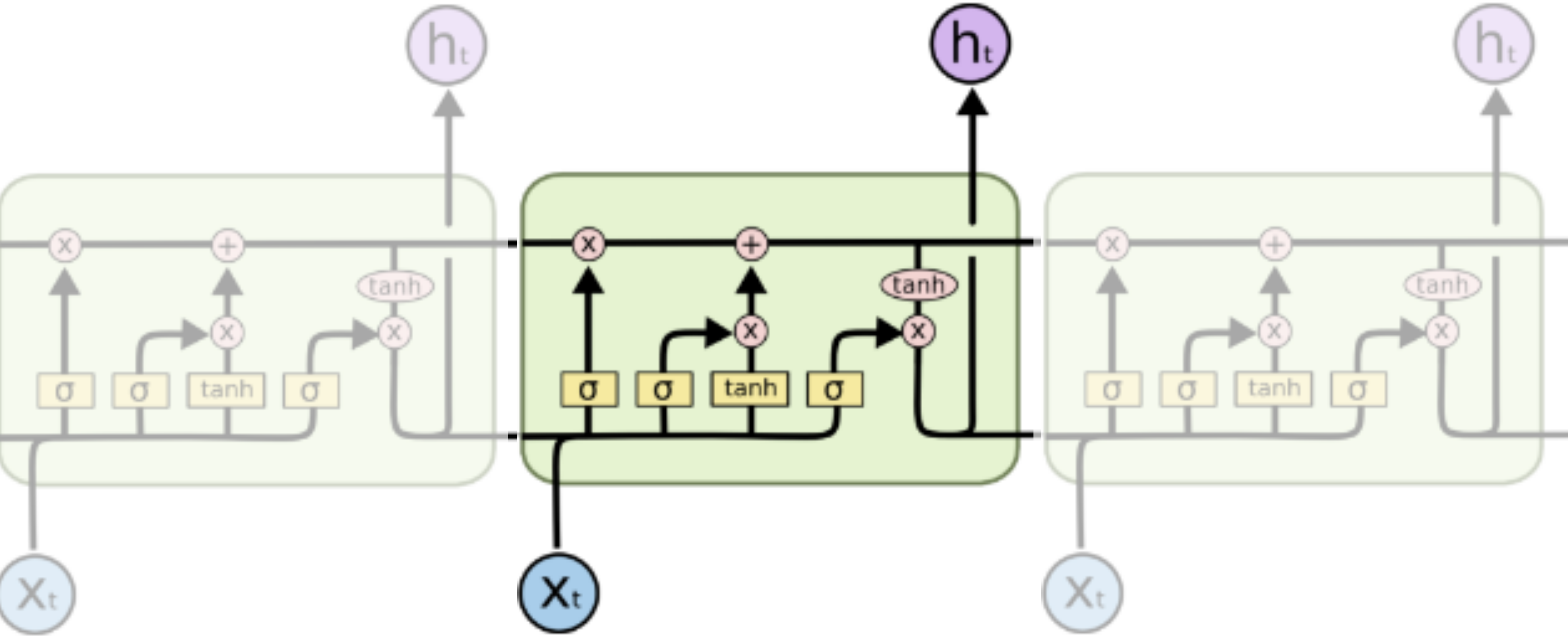


Figure credit: Christopher Olah

Another Visualization

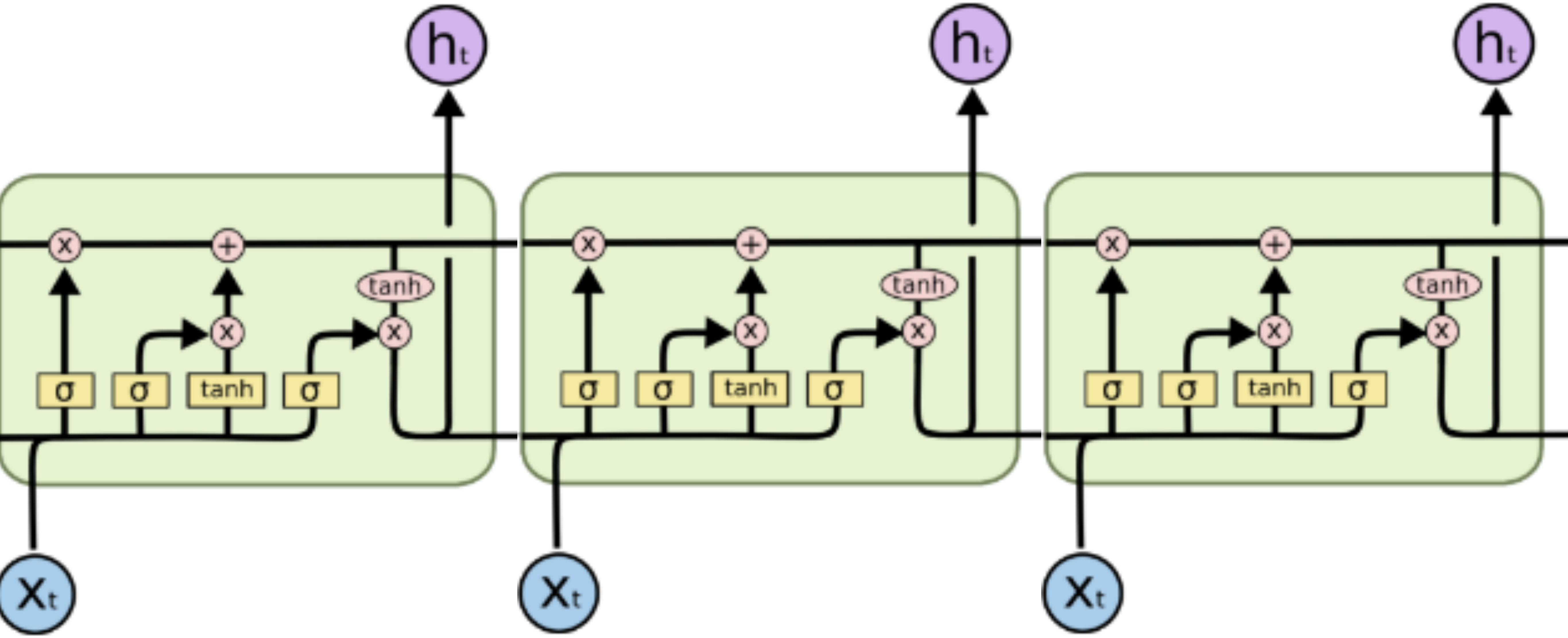


Figure credit: Christopher Olah

Another Visualization

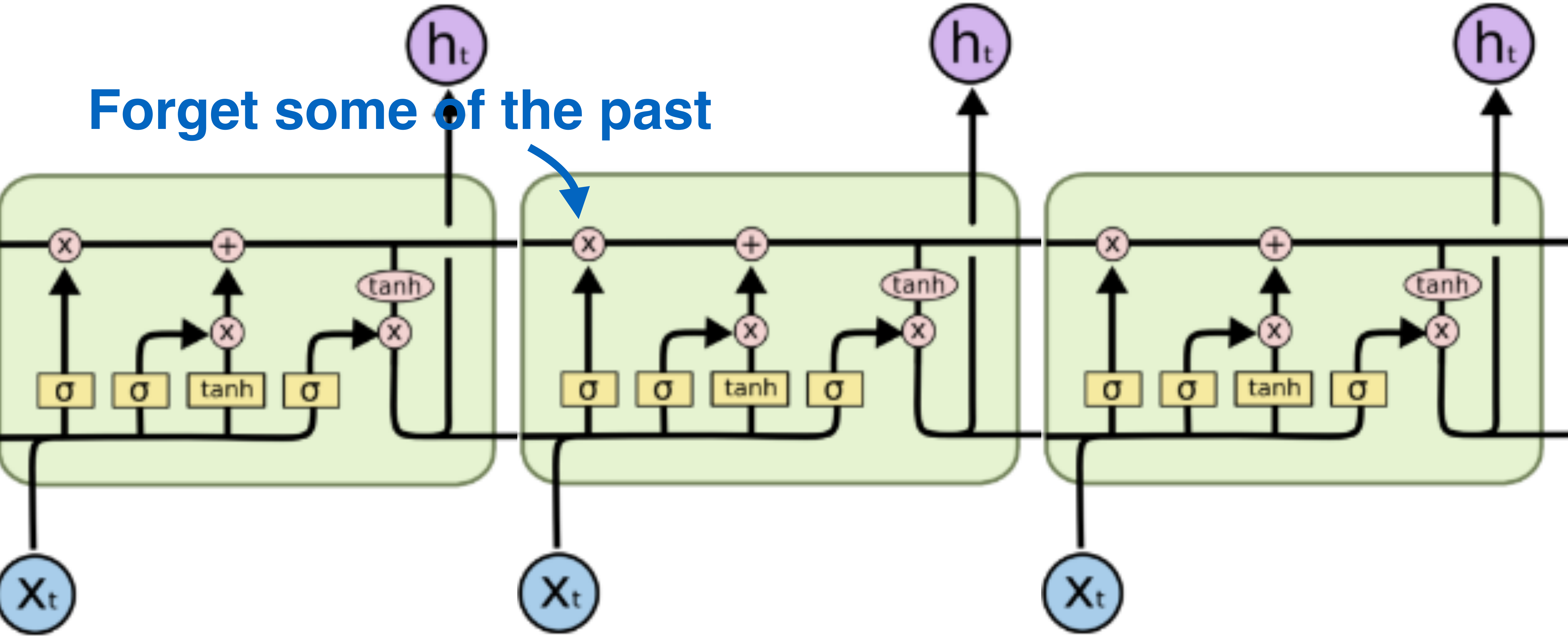


Figure credit: Christopher Olah

Another Visualization

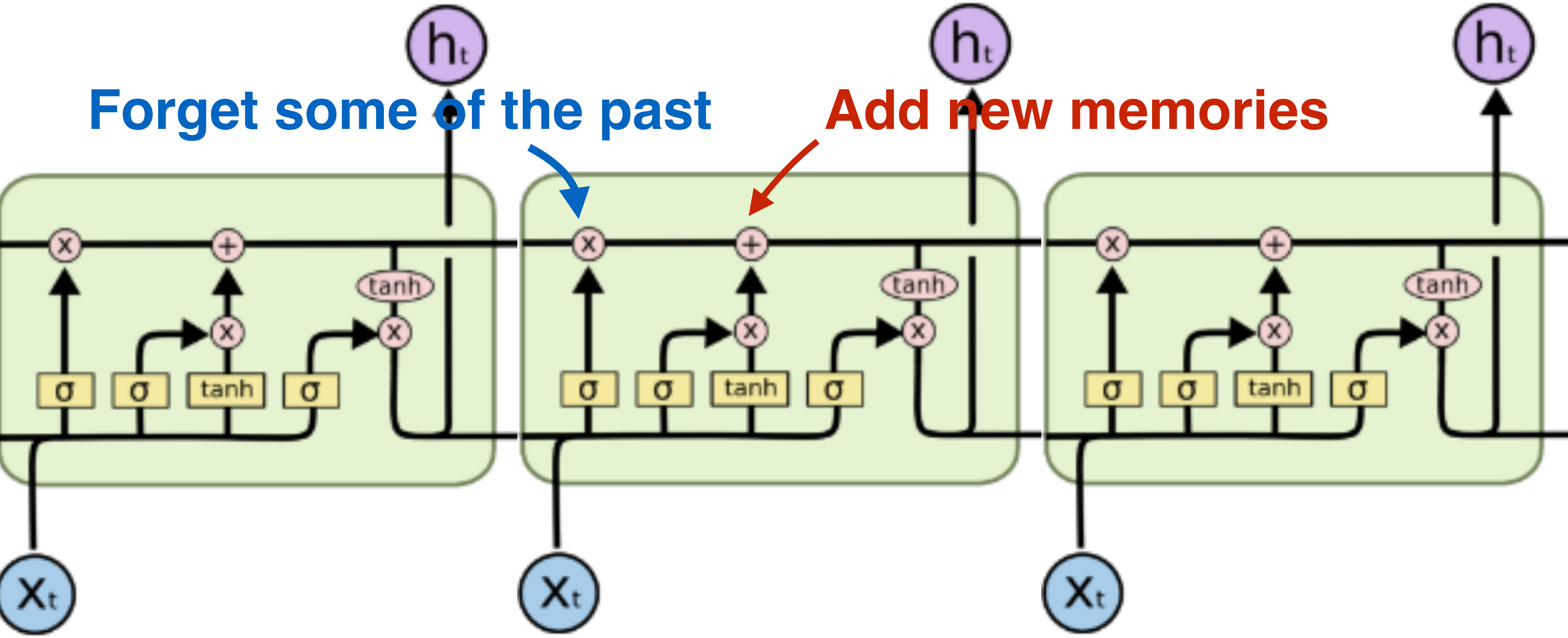


Figure credit: Christopher Olah

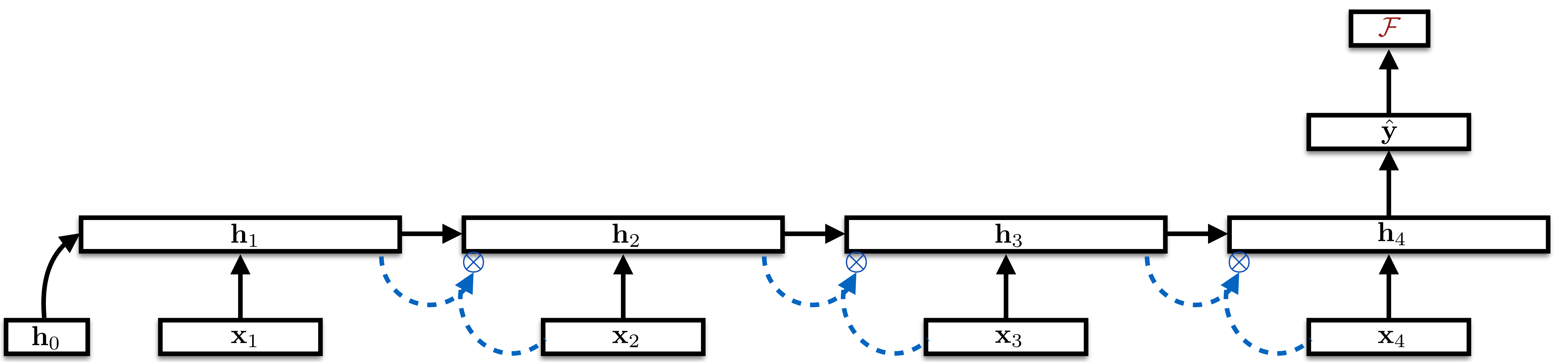
Gated Recurrent Units (GRUs)

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$$

$$\mathbf{z}_t = \sigma(f_z([\mathbf{h}_{t-1}; \mathbf{x}_t]))$$

$$\mathbf{r}_t = \sigma(f_r([\mathbf{h}_{t-1}; \mathbf{x}_t]))$$

$$\tilde{\mathbf{h}}_t = f([r_t \odot \mathbf{h}_{t-1}; \mathbf{x}_t]))$$



Summary

- Better gradient propagation is possible when you use additive rather than multiplicative/highly non-linear recurrent dynamics

RNN $$\mathbf{h}_t = f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

LSTM $$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

GRU $$\mathbf{h}_t = (\mathbf{1} - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot f([\mathbf{x}_t; \mathbf{r}_t \odot \mathbf{h}_{t-1}])$$

Summary

- Better gradient propagation is possible when you use additive rather than multiplicative/highly non-linear recurrent dynamics

RNN $\mathbf{h}_t = f([\mathbf{x}_t; \mathbf{h}_{t-1}])$

LSTM $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$

GRU $\mathbf{h}_t = (\mathbf{1} - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot f([\mathbf{x}_t; \mathbf{r}_t \odot \mathbf{h}_{t-1}])$

Summary

- Better gradient propagation is possible when you use additive rather than multiplicative/highly non-linear recurrent dynamics

RNN $\mathbf{h}_t = f([\mathbf{x}_t; \mathbf{h}_{t-1}])$

LSTM $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} \boxed{+} \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$

GRU $\mathbf{h}_t = (\mathbf{1} - \mathbf{z}_t) \odot \mathbf{h}_{t-1} \boxed{+} \mathbf{z}_t \odot f([\mathbf{x}_t; \mathbf{r}_t \odot \mathbf{h}_{t-1}])$

- Recurrent architectures are an active area of research, requires a mix of mathematical analysis, creativity, problem-specific knowledge
  - (LSTMs are hard to beat though!)

Questions?

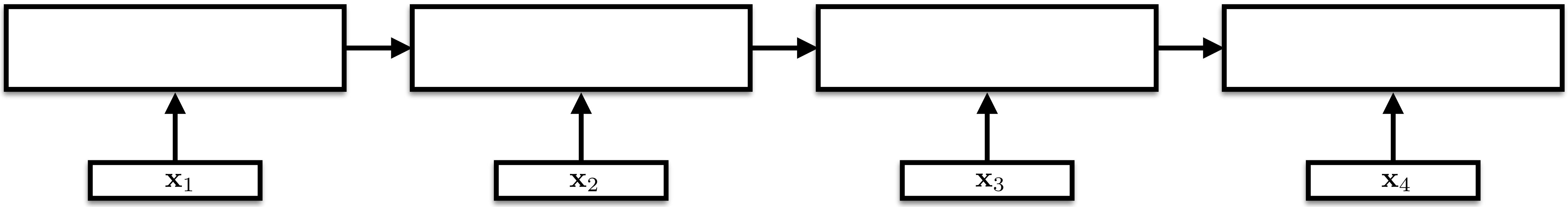
Break?

A Few Tricks of the Trade

- Depth
- Dropout
- Implementation tricks

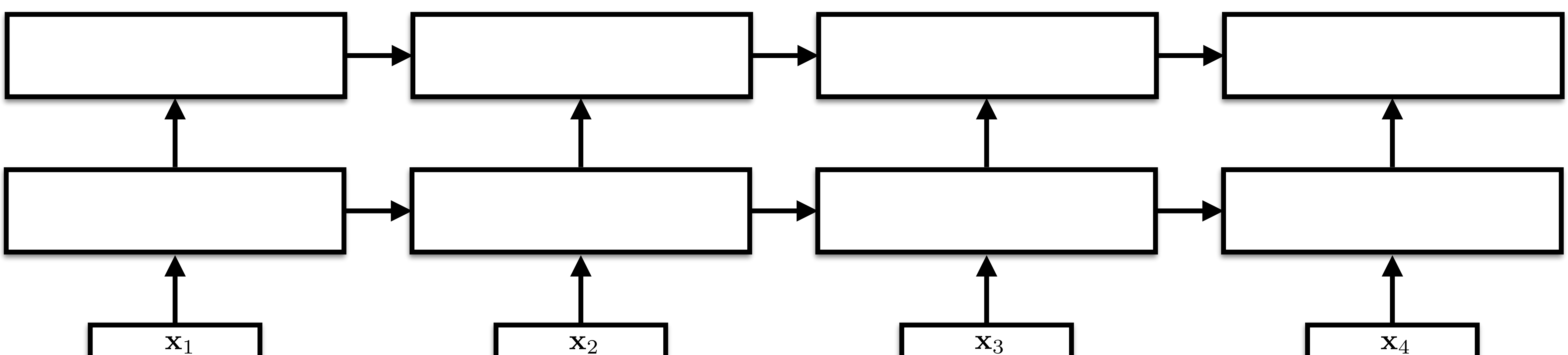
“Deep” LSTMs

- This term has been defined several times, but the following is the most standard convention



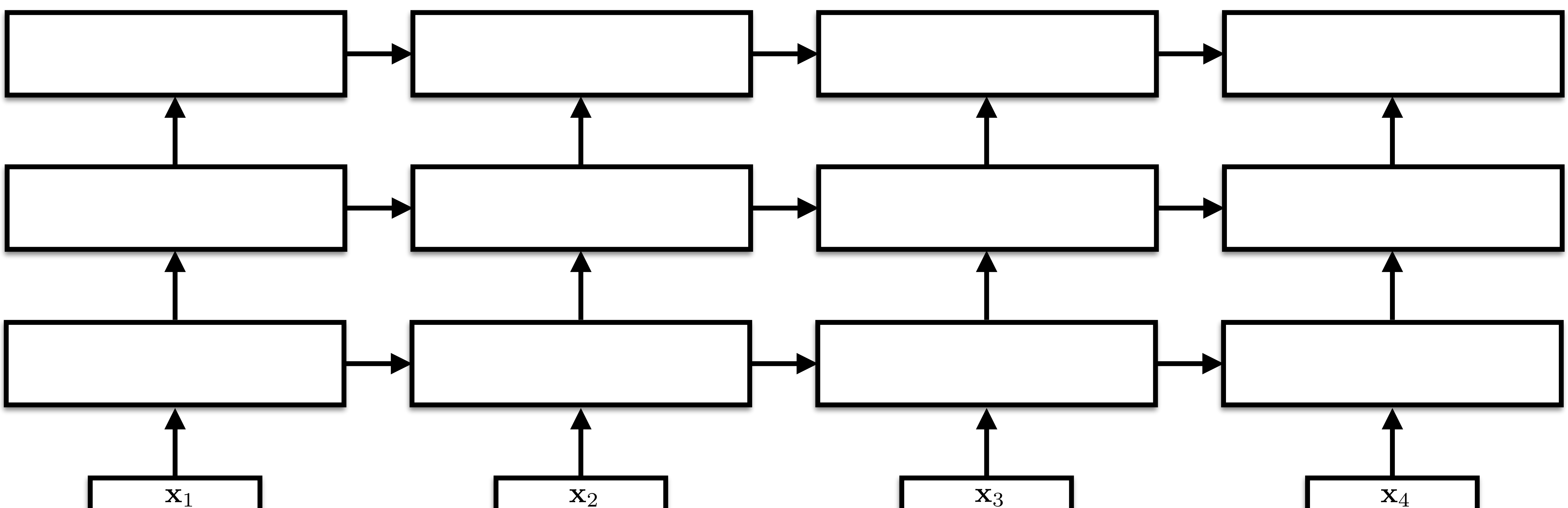
“Deep” LSTMs

- This term has been defined several times, but the following is the most standard convention



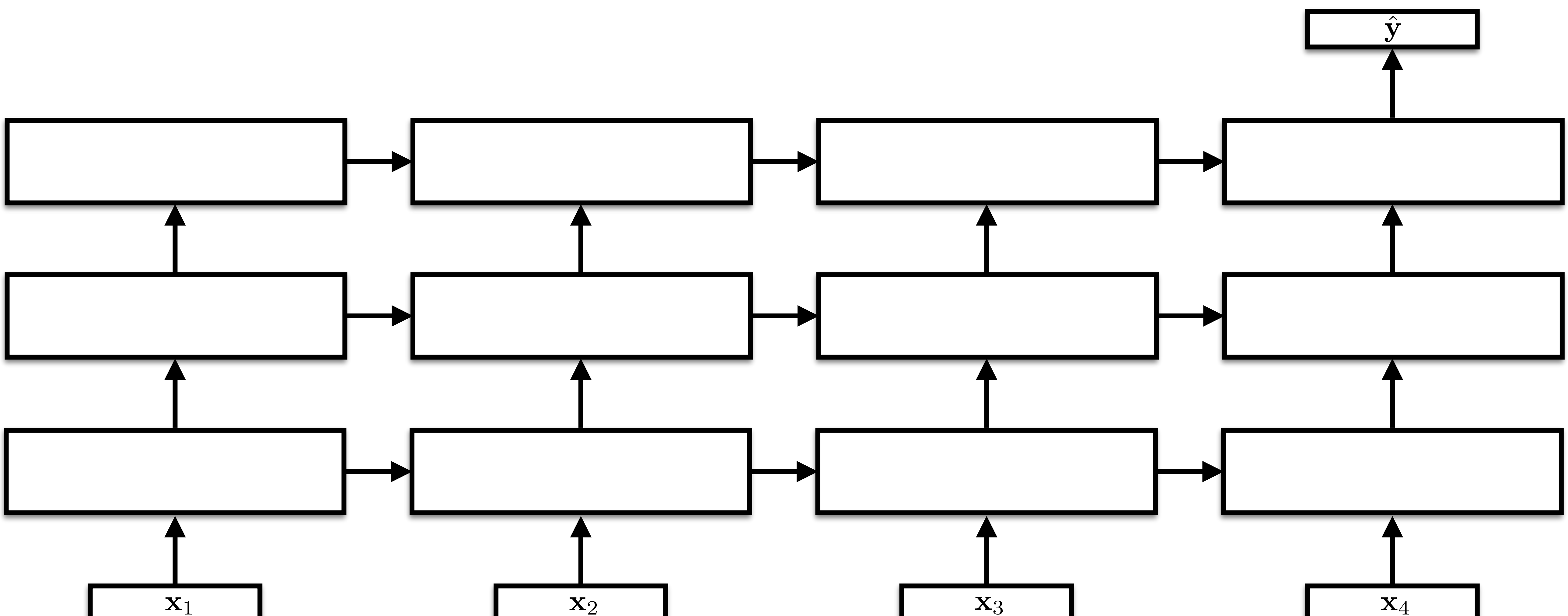
“Deep” LSTMs

- This term has been defined several times, but the following is the most standard convention



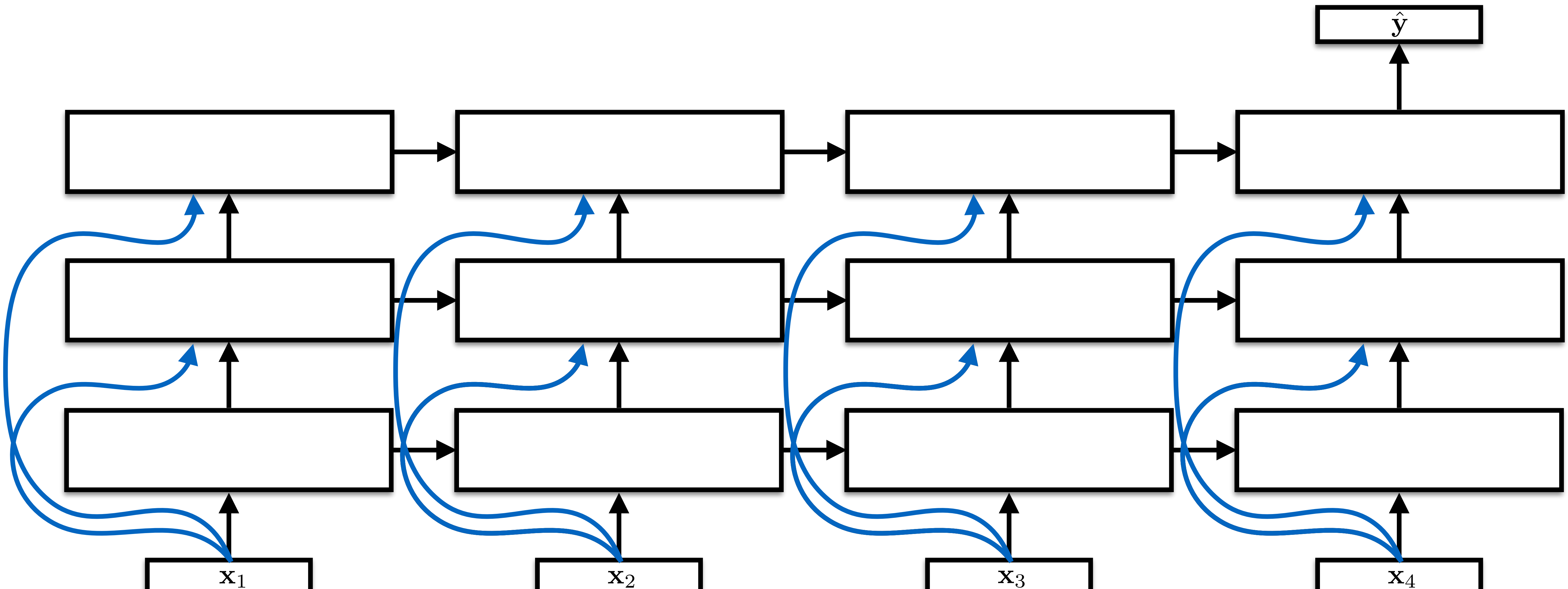
“Deep” LSTMs

- This term has been defined several times, but the following is the most standard convention



“Deep” LSTMs

- This term has been defined several times, but the following is the most standard convention

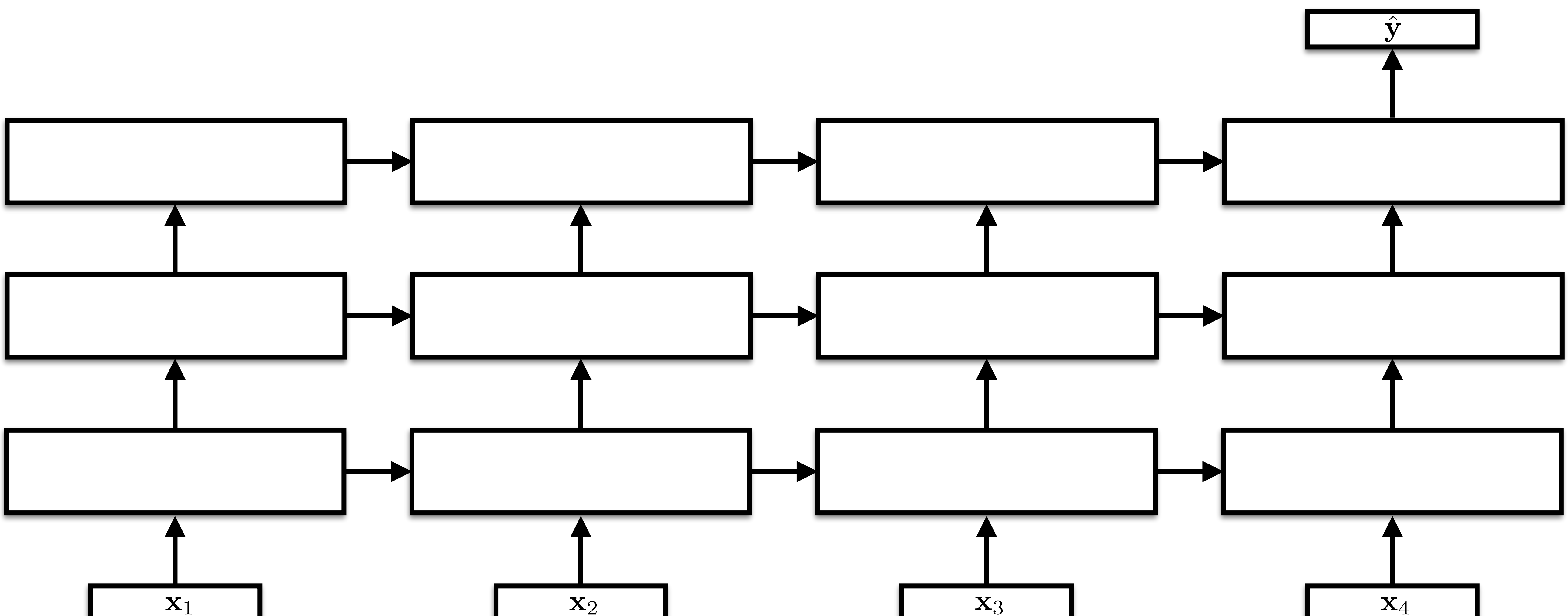


Does Depth Matter?

- Yes, it helps
- It seems to play a less significant role in text than in audio/visual processing
  - H1: More transformation of the input is required for ASR, image recognition, etc., than for common text applications (word vectors become customized to be “good inputs” to RNNs whereas you’re stuck with what nature gives you for speech/vision)
  - H2: less effort has been made to find good architectures (RNNs are expensive to train; have been widely used for less long)
  - H3: back prop through time + depth is hard and we need better optimizers
  - Many other possibilities…
- 2-8 layers seems to be standard
- Input “skip” connections are used often but by no means universally

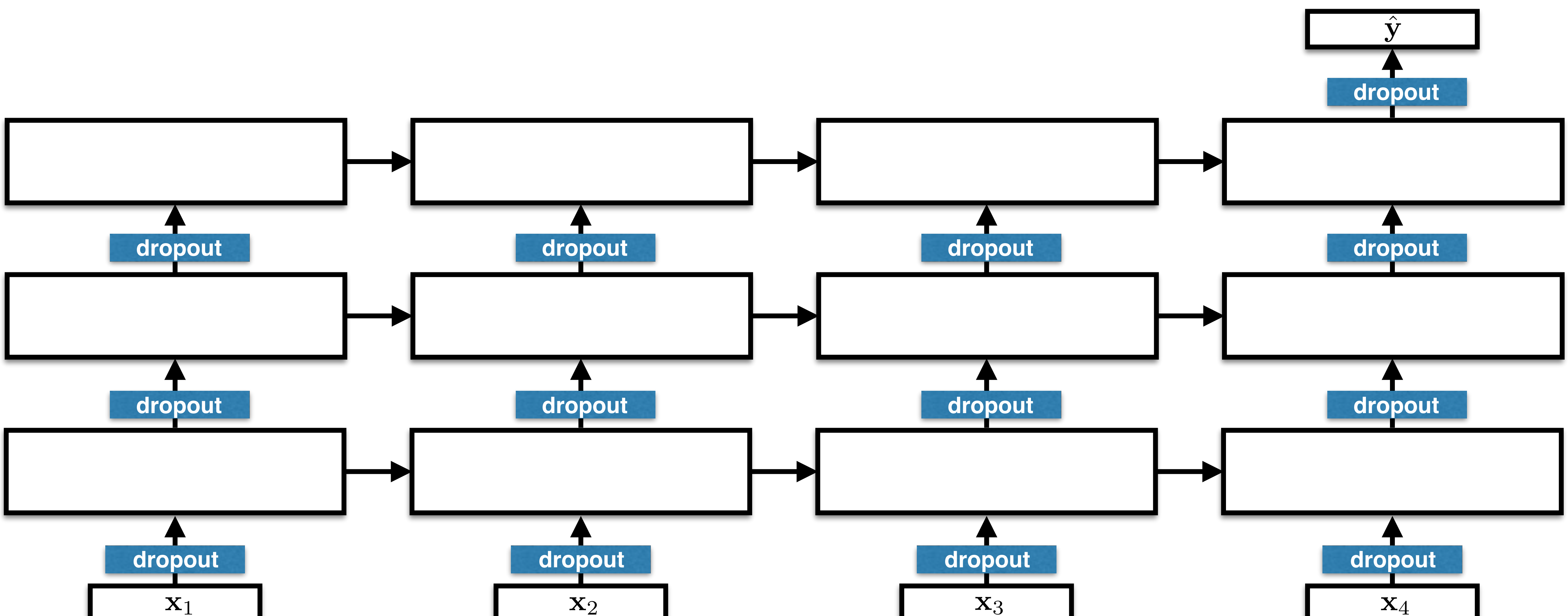
Dropout and Deep LSTMs

- Applying dropout layers requires some care



Dropout and Deep LSTMs

- Apply dropout between layers, but not on the recurrent connections



Implementation Details

- **For speed**
  - Use diagonal matrices instead of full matrices (esp. for gates)
  - Concatenate parameter matrices for all gates and do a single matrix-vector(/matrix) multiplication
  - Use optimized implementations (from NVIDIA)
  - Use GRUs or reduced-gate variant of LSTMs
- **For learning speed and performance**
  - Initialize so that the bias on the forget gate is large (intuitively: at the beginning of training, the signal from the past is unreliable)
  - Use random orthogonal matrices to initialize the square matrices

Implementation Details: Minibatching

- GPU hardware is
  - pretty fast for elementwise operations (IO bound- can't get enough data through the GPU)
  - very fast for matrix-matrix multiplication (usually compute bound - the GPU will work at 100% capacity, and GPU cores are **fast**)
- RNNs, LSTMs, GRUs all consist of
  - lots of elementwise operations (addition, multiplication, nonlinearities, …)
  - lots of matrix-vector products
- Minibatching: convert many matrix-vector products into a single matrix-matrix multiplication

Minibatching

Single-instance RNN

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$

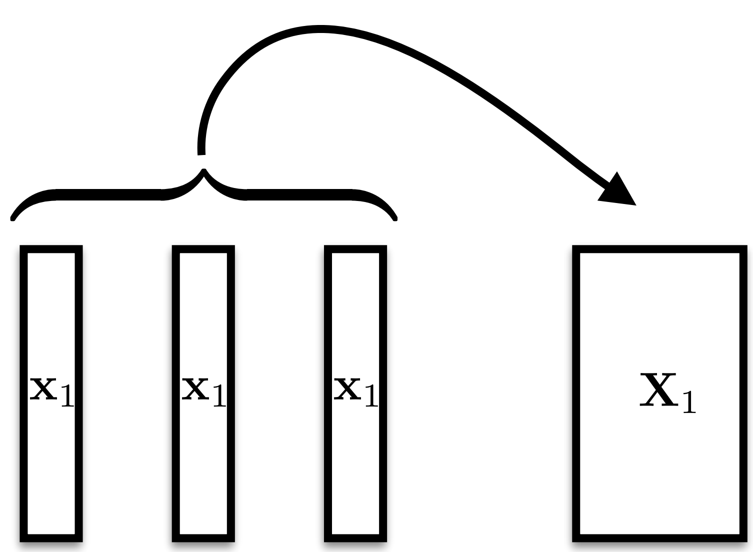
Minibatching

Single-instance RNN

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$

Minibatch RNN

$$\mathbf{H}_t = g(\mathbf{V}\mathbf{X}_t + \mathbf{U}\mathbf{H}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{Y}}_t = \mathbf{W}\mathbf{H}_t + \mathbf{b}$$



We batch across instances,
not across time.

Minibatching

Single-instance RNN

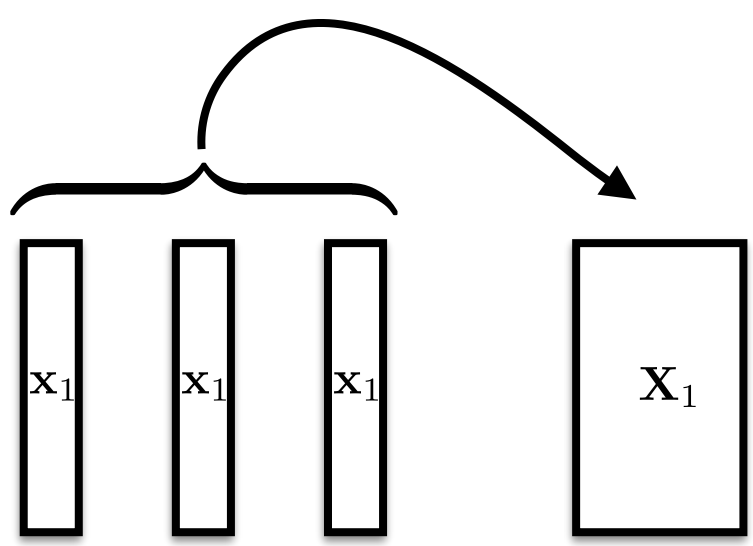
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$

Minibatch RNN

$$\mathbf{H}_t = g(\mathbf{V}\mathbf{X}_t + \mathbf{U}\mathbf{H}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{Y}}_t = \mathbf{W}\mathbf{H}_t + \mathbf{b}$$



We batch across instances,
not across time.

Minibatching

Single-instance RNN

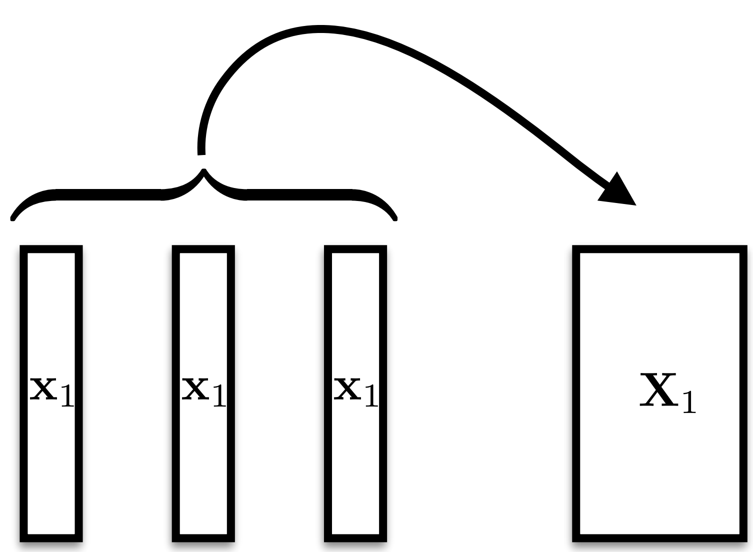
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$

Minibatch RNN

$$\mathbf{H}_t = g(\mathbf{V}\mathbf{X}_t + \mathbf{U}\mathbf{H}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{Y}}_t = \mathbf{W}\mathbf{H}_t + \mathbf{b}$$



We batch across instances,
not across time.

Minibatching

Single-instance RNN

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

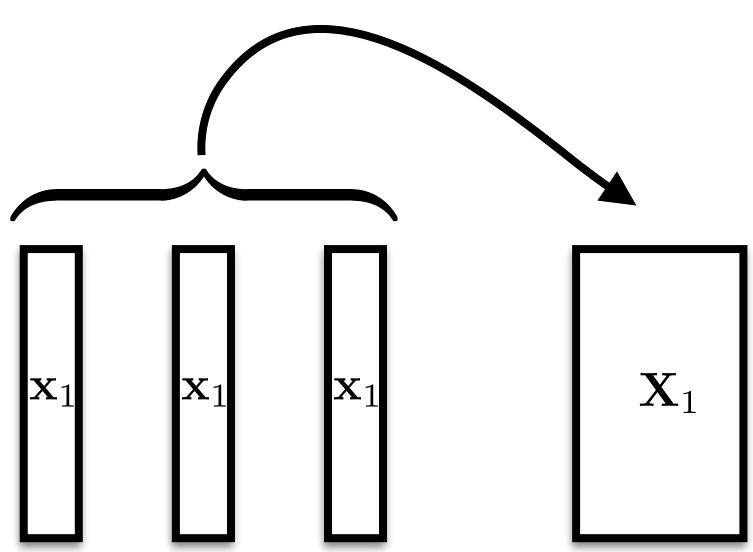
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$

Minibatch RNN

$$\mathbf{H}_t = g(\mathbf{V}\mathbf{X}_t + \mathbf{U}\mathbf{H}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{Y}}_t = \mathbf{W}\mathbf{H}_t + \mathbf{b}$$

anything wrong here?



We batch across instances, not across time.

Minibatching

- The challenge with working with mini batches of sequences is … sequences are of different lengths
- This usually means you bucket training instances based on similar lengths, and pad with 0's
  - Be careful when padding not to back propagate a non-zero value!
- Manual minibatching convinces me that this is the era of assembly language programming for neural networks. Make the future an easier place to program!

Questions?

Bidirectional RNNs

- We can read a sequence from left to right to obtain a representation
- Or we can read it from right to left
- Or we can read it from both and combine the representations

Word Embedding Models

car

Memorize

Generalize

Word Embedding Models

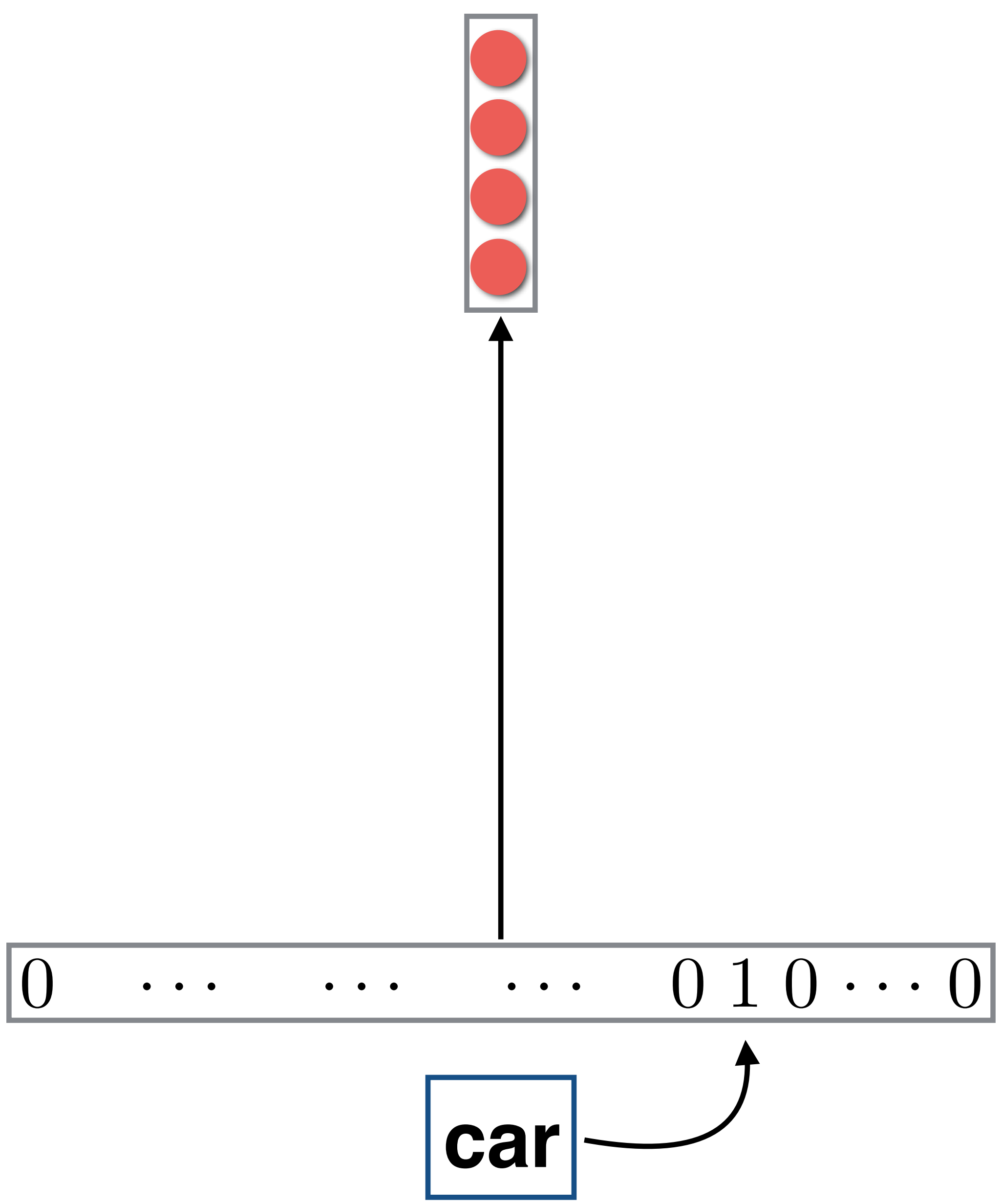
$$\boxed{0 \quad \cdots \quad \cdots \quad \cdots \quad 0\ 1\ 0 \cdots 0}$$

car

Memorize

Generalize

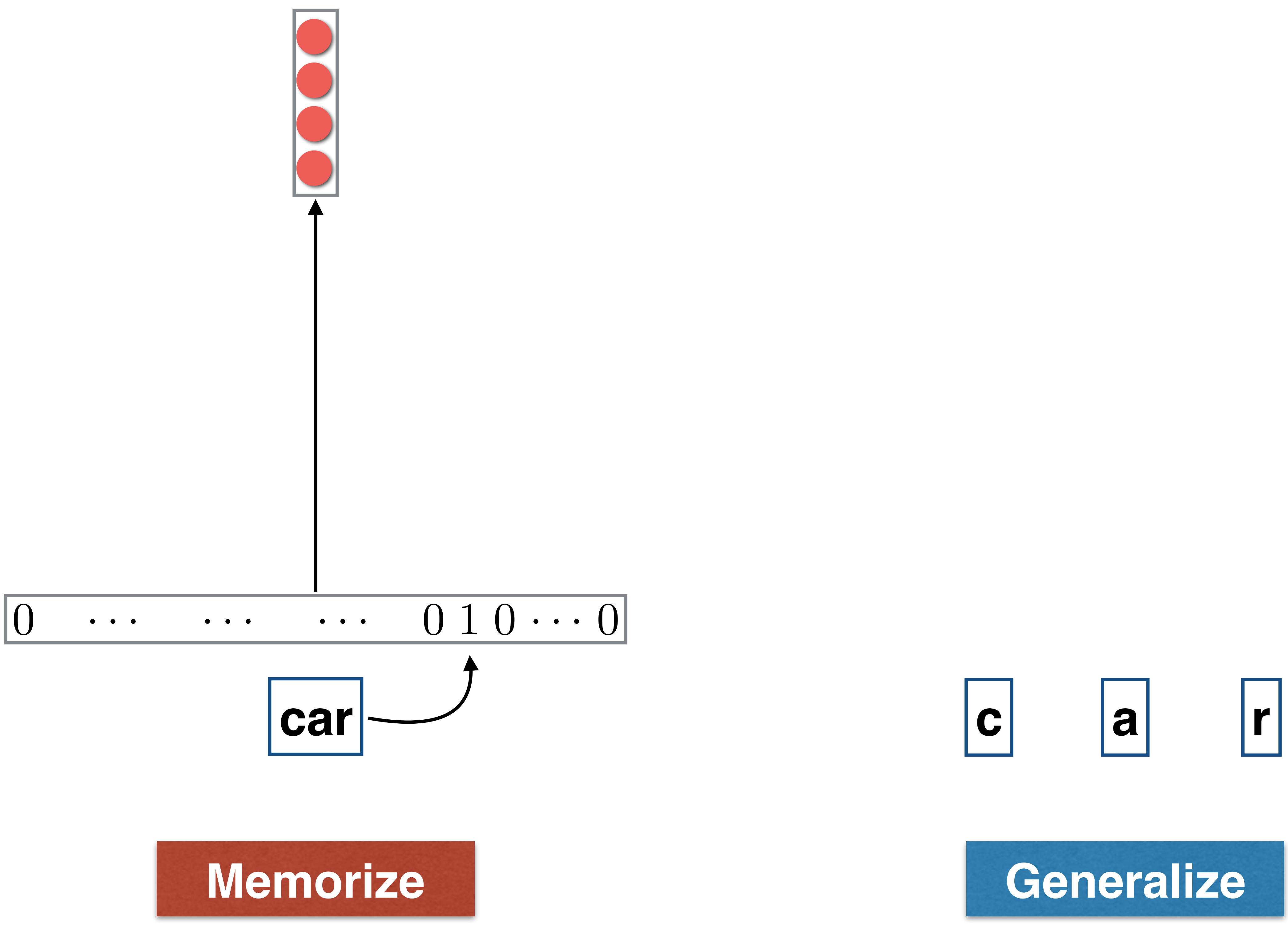
Word Embedding Models



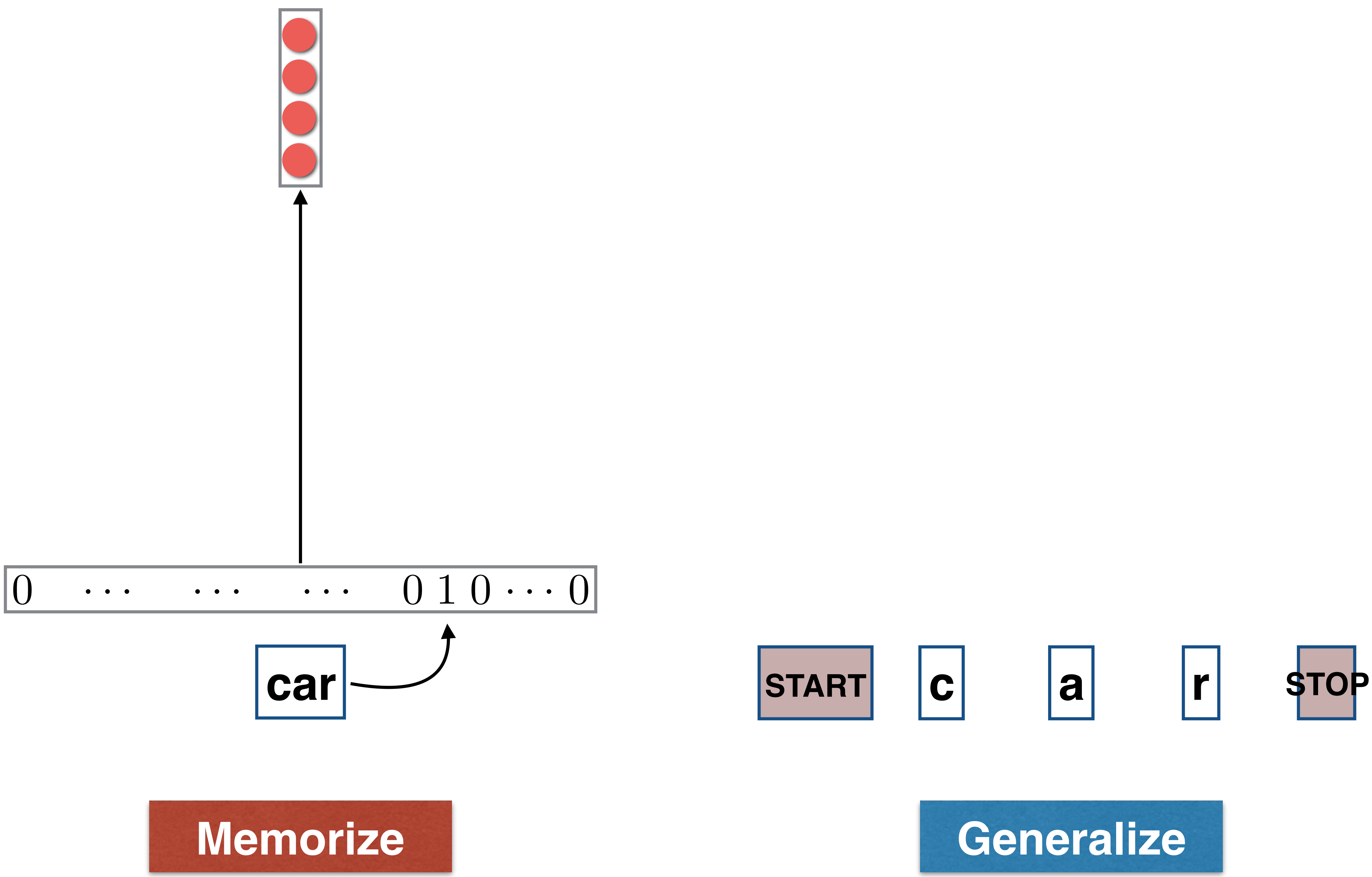
Memorize

Generalize

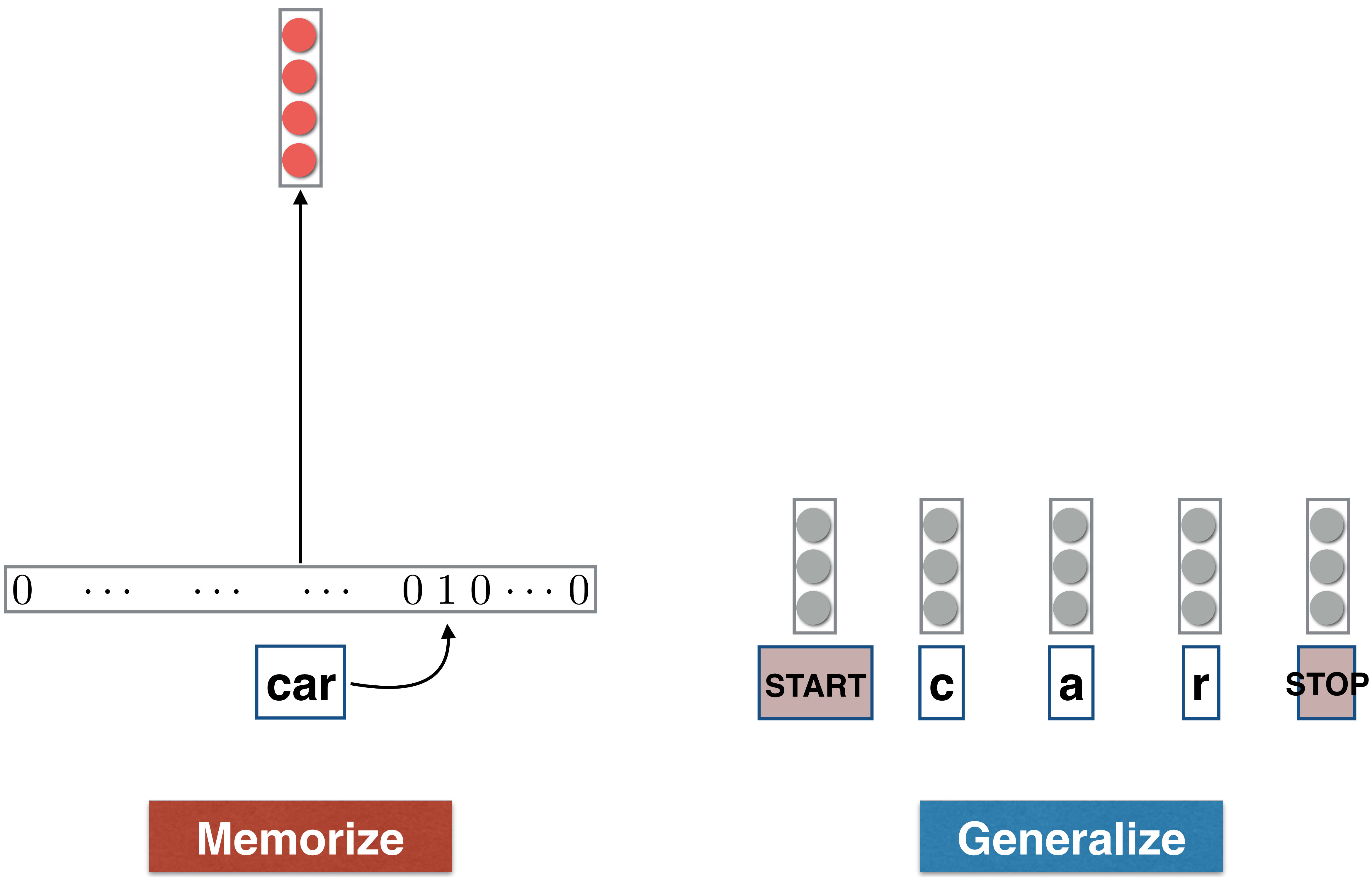
Word Embedding Models



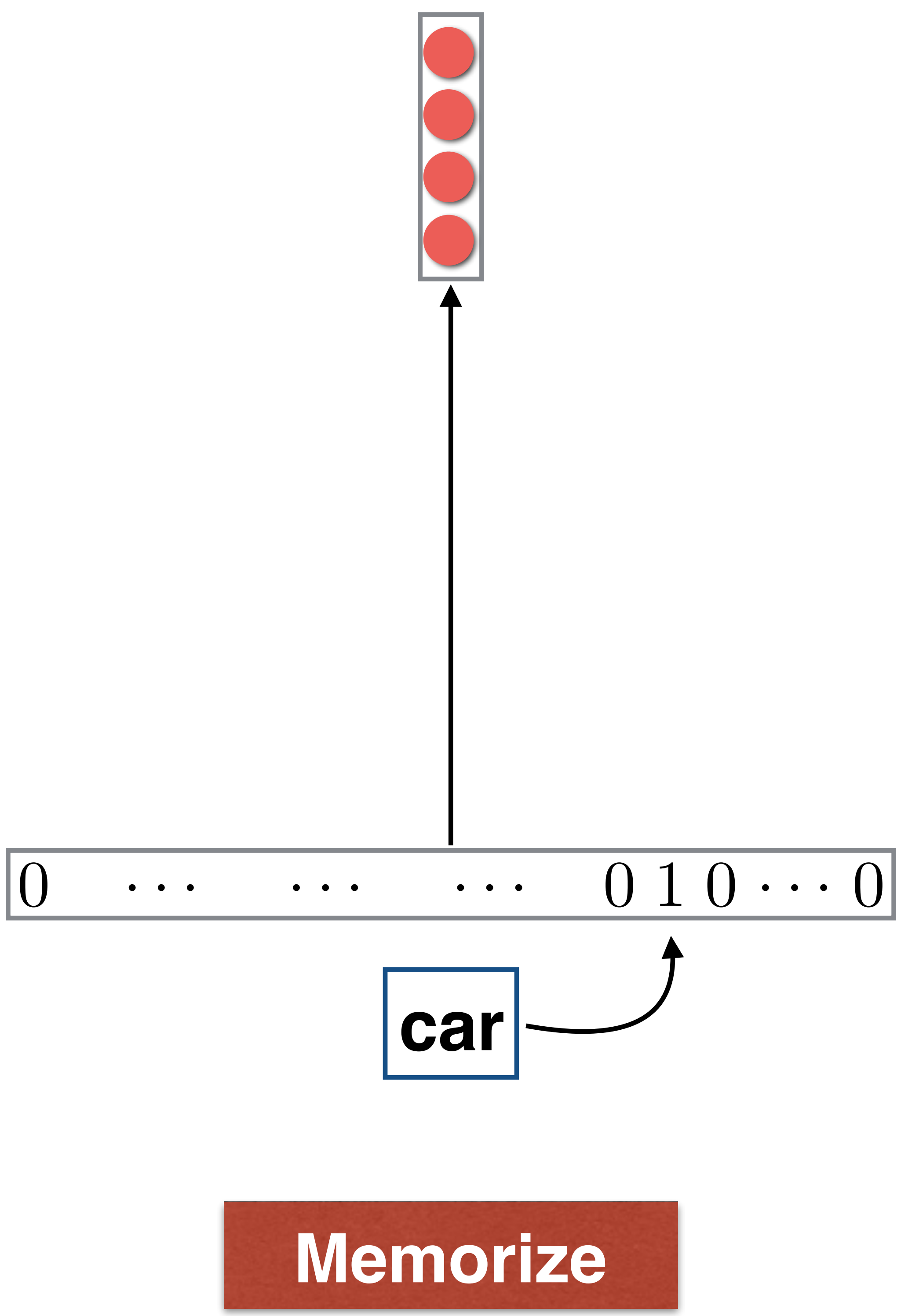
Word Embedding Models

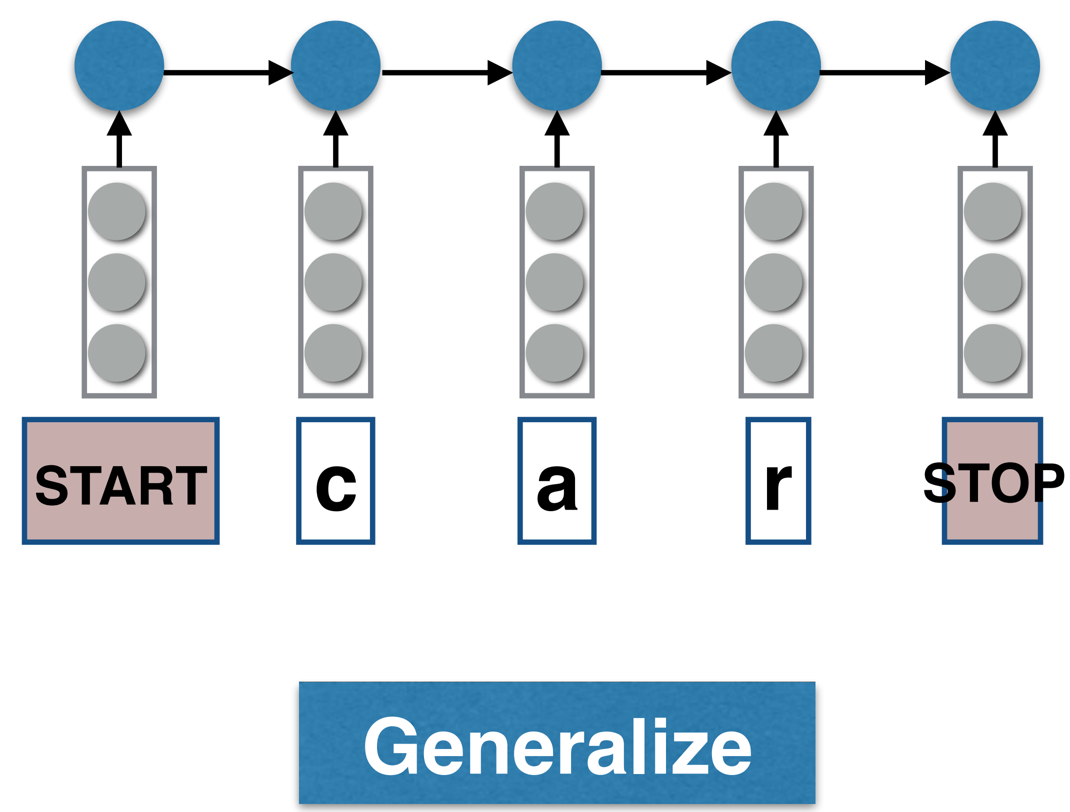


Word Embedding Models

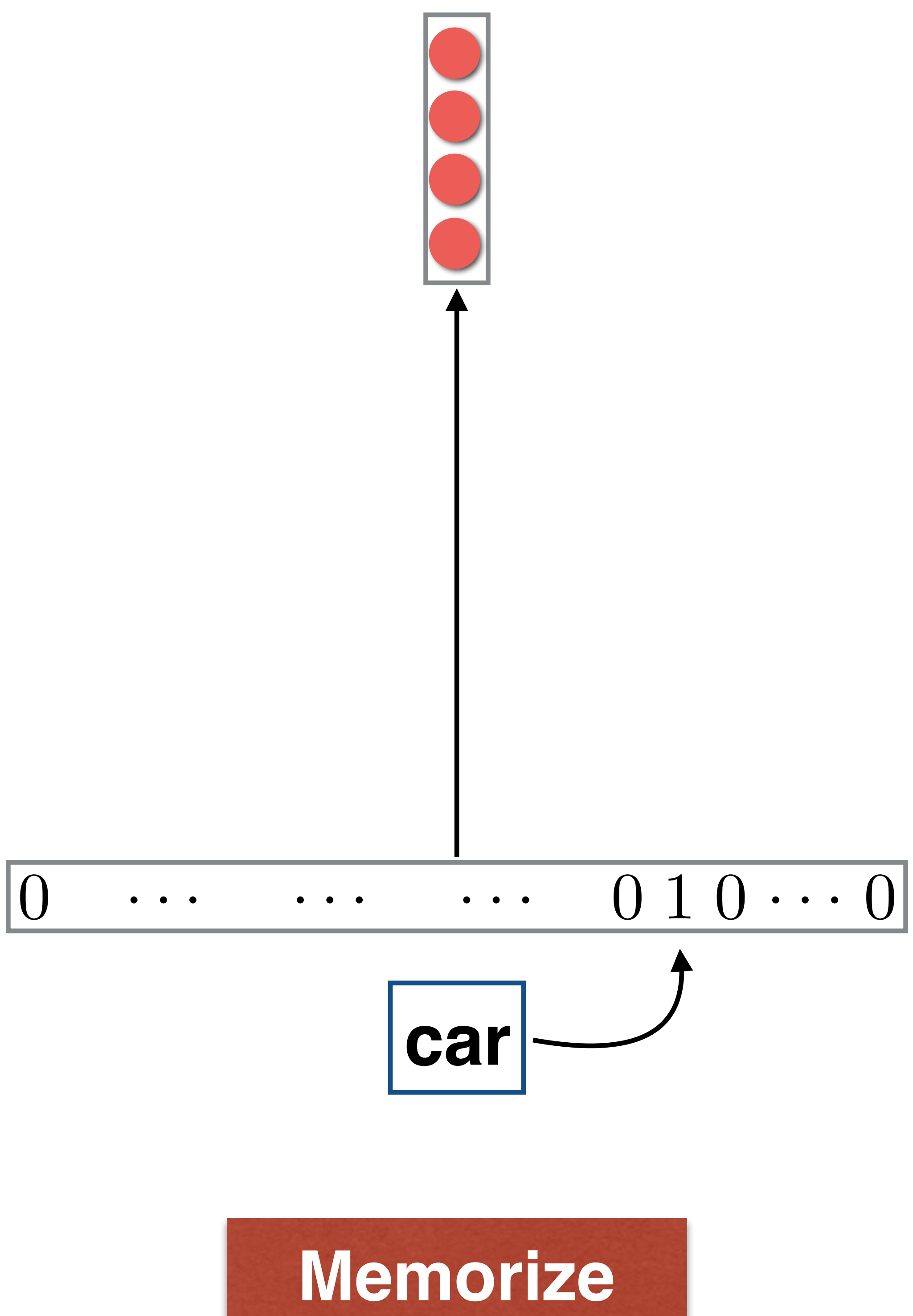


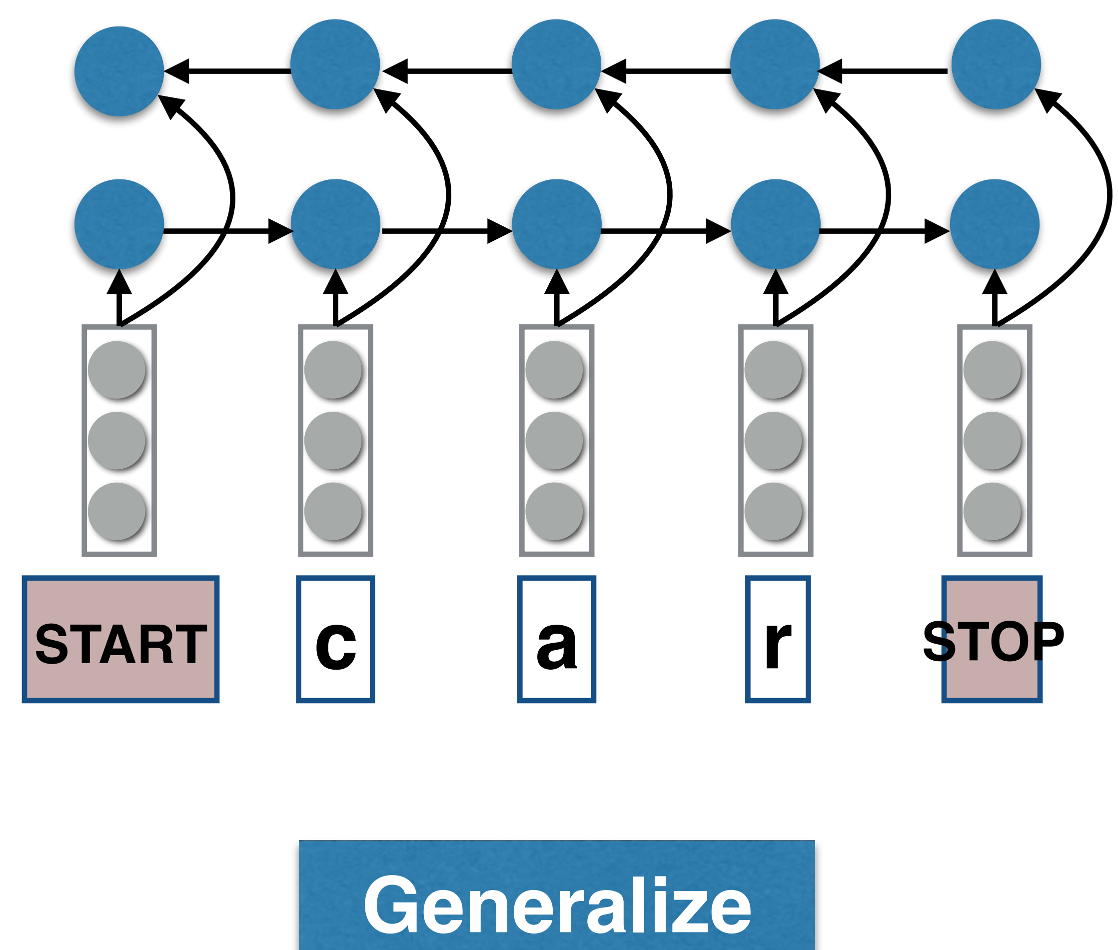
Word Embedding Models



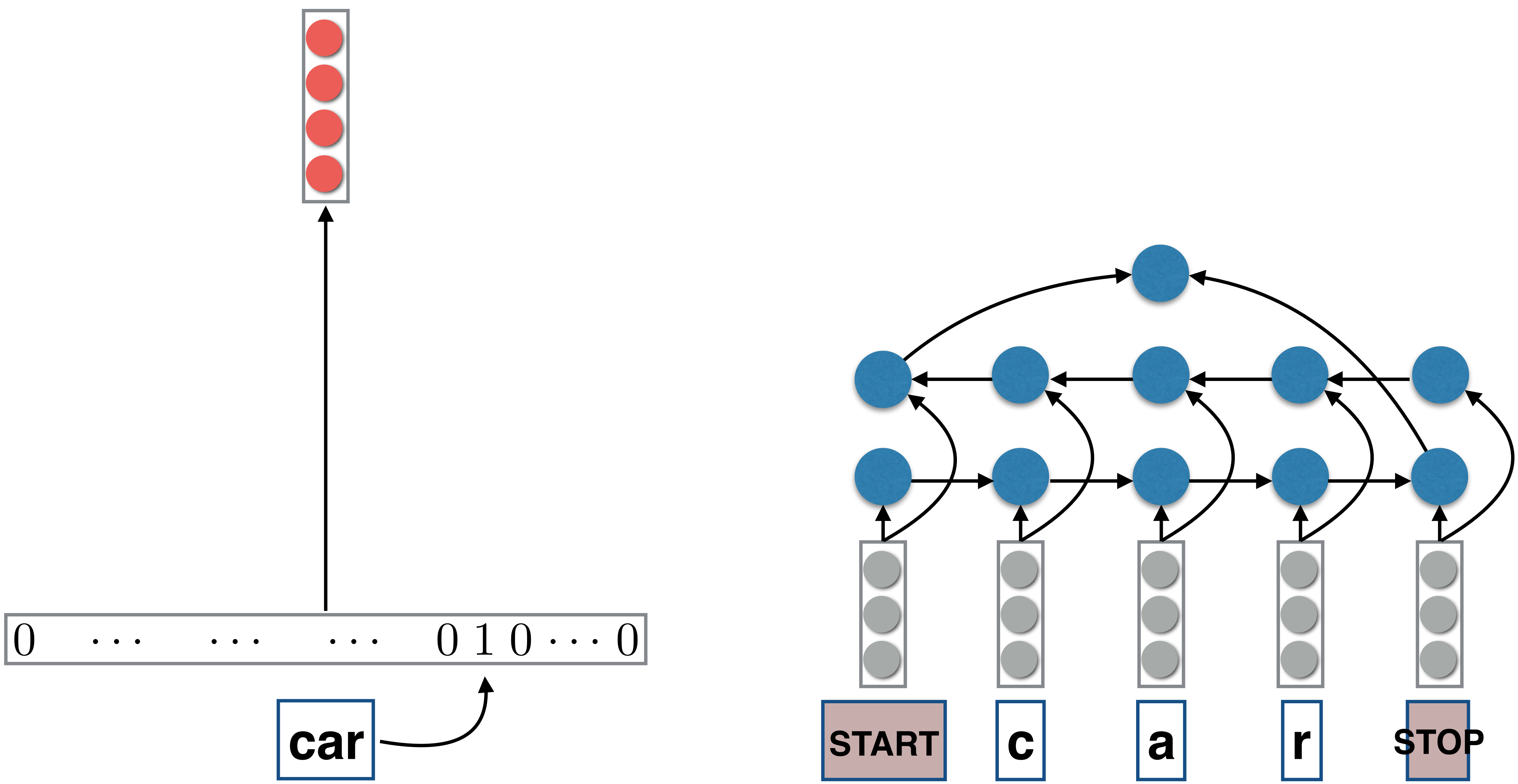


Word Embedding Models

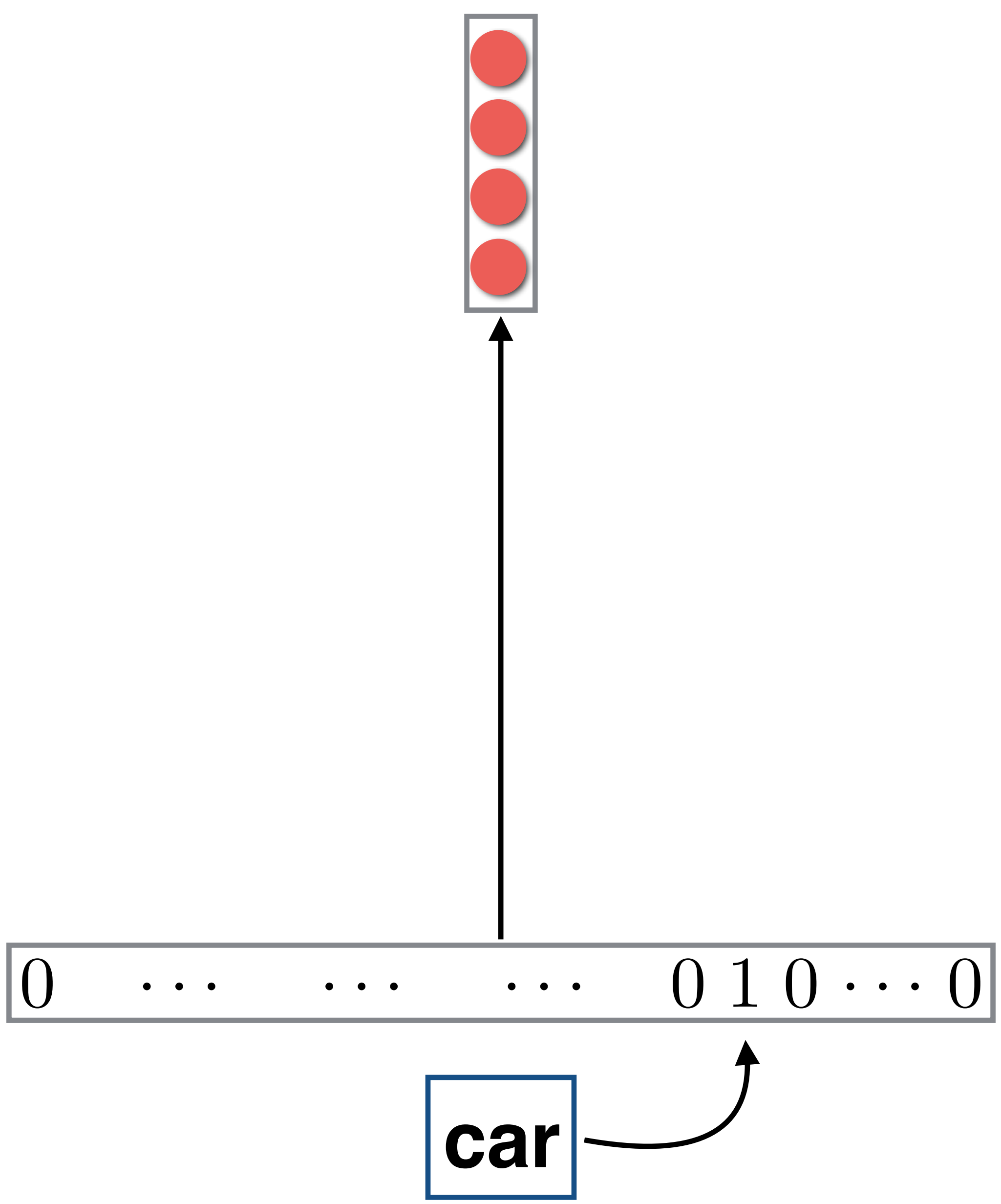


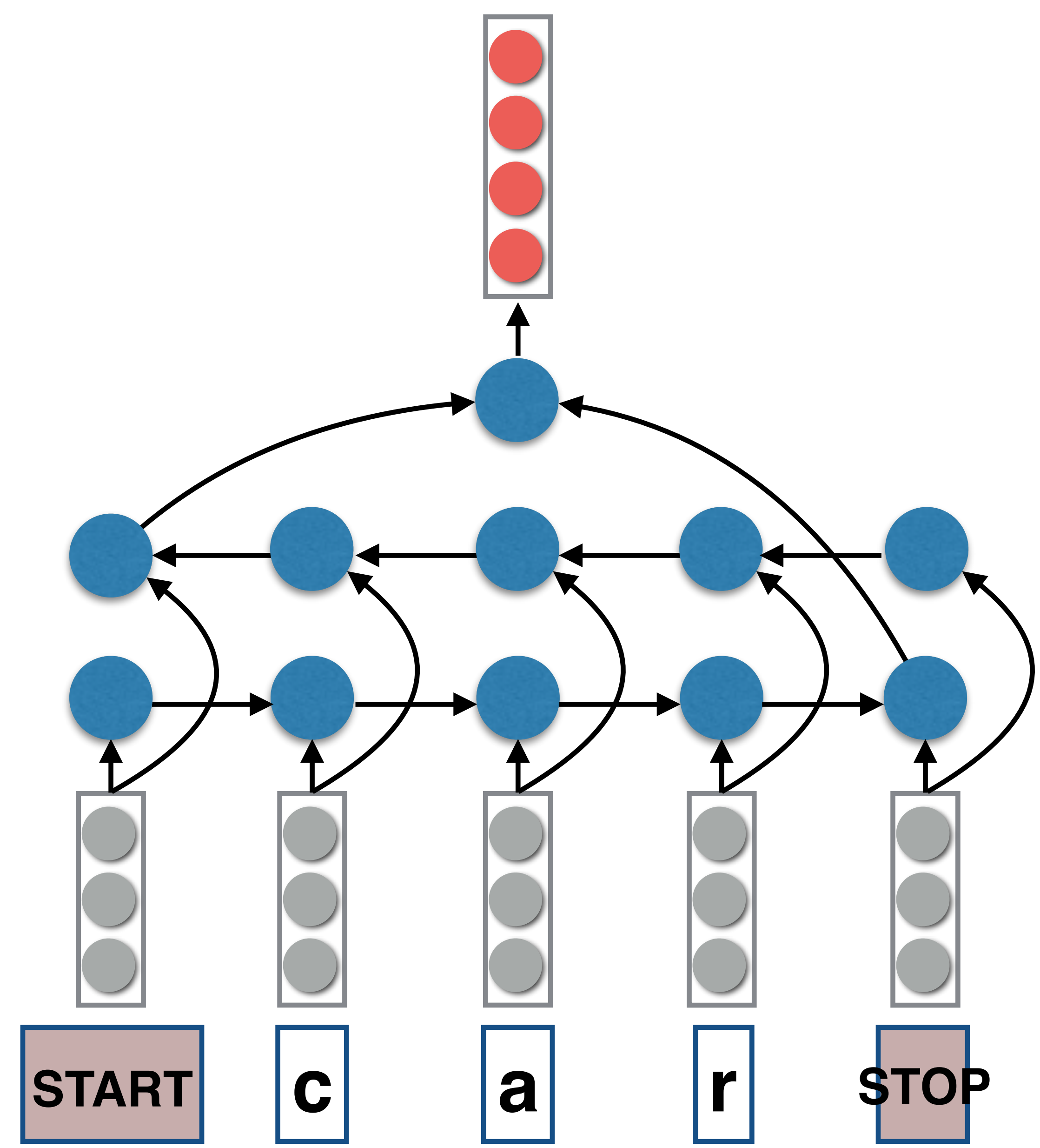


Word Embedding Models



Word Embedding Models





Memorize

Generalize

Language modeling
CharLSTM > Word Lookup

		ppl Words	ppl Chars	Δ
Analytic	**English**	59.4	57.4	-2.0

Language modeling
CharLSTM > Word Lookup

		ppl Words	ppl Chars	Δ
Analytic	**English**	59.4	57.4	-2.0
Agglutinative	**Turkish**	44.0	32.9	**-11.1**
	German	59.1	43.0	**-16.1**

Language modeling
CharLSTM > Word Lookup

		ppl Words	ppl Chars	Δ
Analytic	**English**	59.4	57.4	-2.0
Agglutinative	**Turkish**	44.0	32.9	**-11.1**
	German	59.1	43.0	**-16.1**
Fusional	**Portuguese**	46.2	40.9	-5.3
	Catalan	35.3	34.9	-0.4

Language modeling
CharLSTM > Word Lookup

		ppl Words	ppl Chars	Δ	lθl Words	lθl Chars
Analytic	**English**	59.4	57.4	-2.0	4.3M	**0.18M**
Agglutinative	**Turkish**	44.0	32.9	**-11.1**	5.7M	**0.17M**
	German	59.1	43.0	**-16.1**	6.3M	**0.18M**
Fusional	**Portuguese**	46.2	40.9	-5.3	4.2M	**0.18M**
	Catalan	35.3	34.9	-0.4	4.3M	**0.18M**

Language modeling
Word similarities

increased	John
reduced	Richard
improved	George
expected	James
decreased	Robert
targeted	Edward

Language modeling
Word similarities

increased	John	Noahshire	phding
reduced	Richard	Nottinghamshire	mixing
improved	George	Bucharest	modelling
expected	James	Saxony	styling
decreased	Robert	Johannesburg	blaming
targeted	Edward	Gloucestershire	christening

Questions?

Recurrent Neural Networks (RNNs)

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$ $\mathbf{0}$

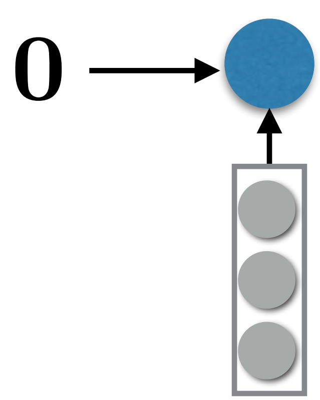
$\boldsymbol{x} =$ START $\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$

What is a vector representation of a sequence $\boldsymbol{x}$?

Recurrent Neural Networks (RNNs)

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$



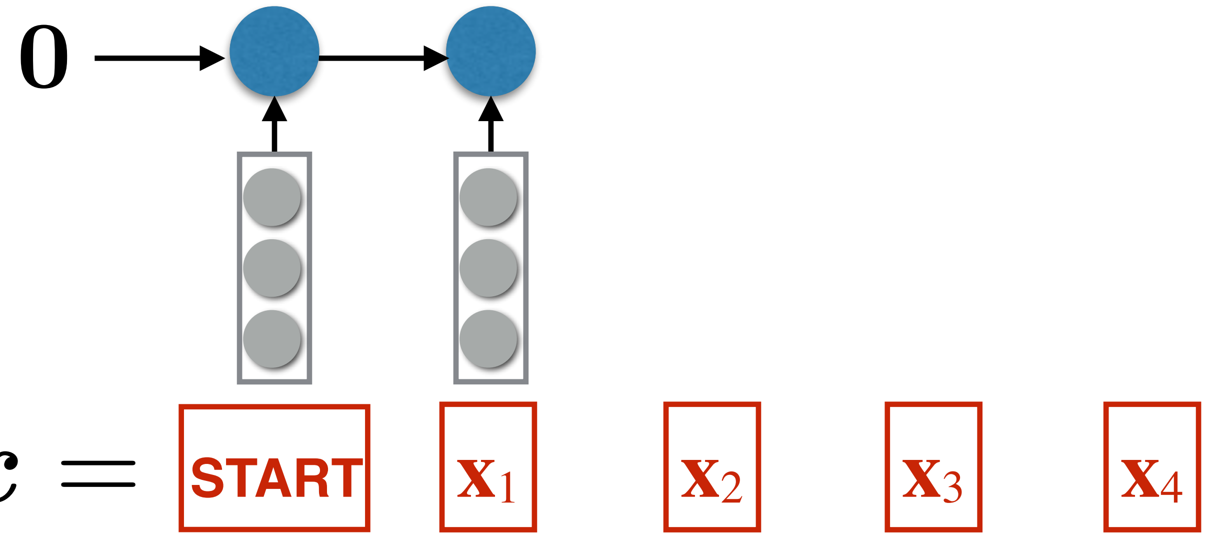
$\boldsymbol{x} =$ START $\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$

What is a vector representation of a sequence $\boldsymbol{x}$?

Recurrent Neural Networks (RNNs)

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$



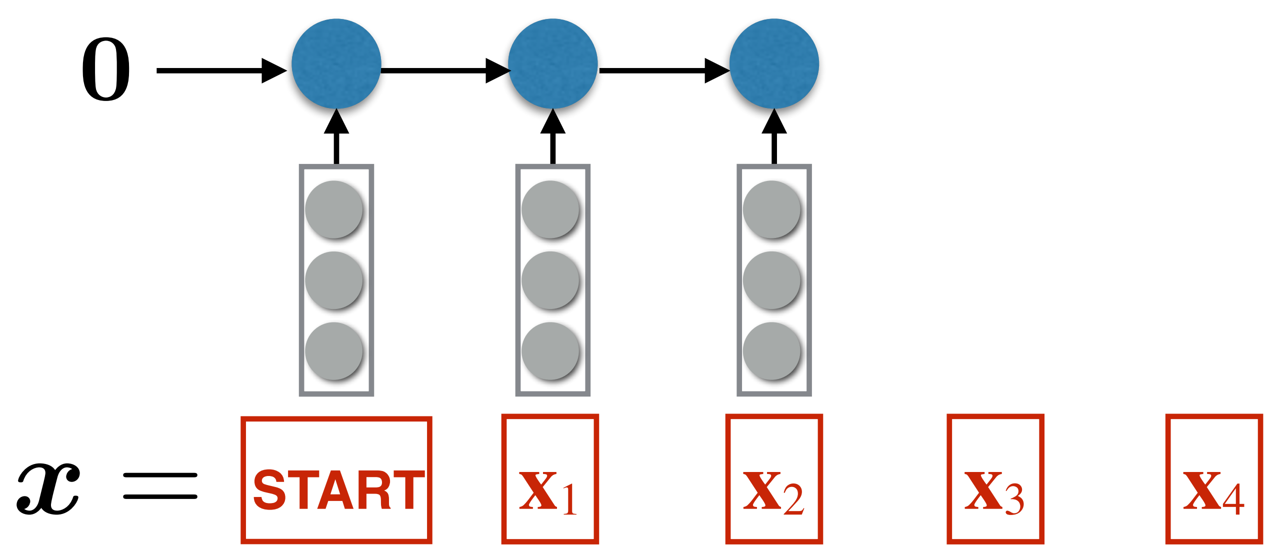
$\boldsymbol{x} =$ START $\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$

What is a vector representation of a sequence $\boldsymbol{x}$?

Recurrent Neural Networks (RNNs)

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$

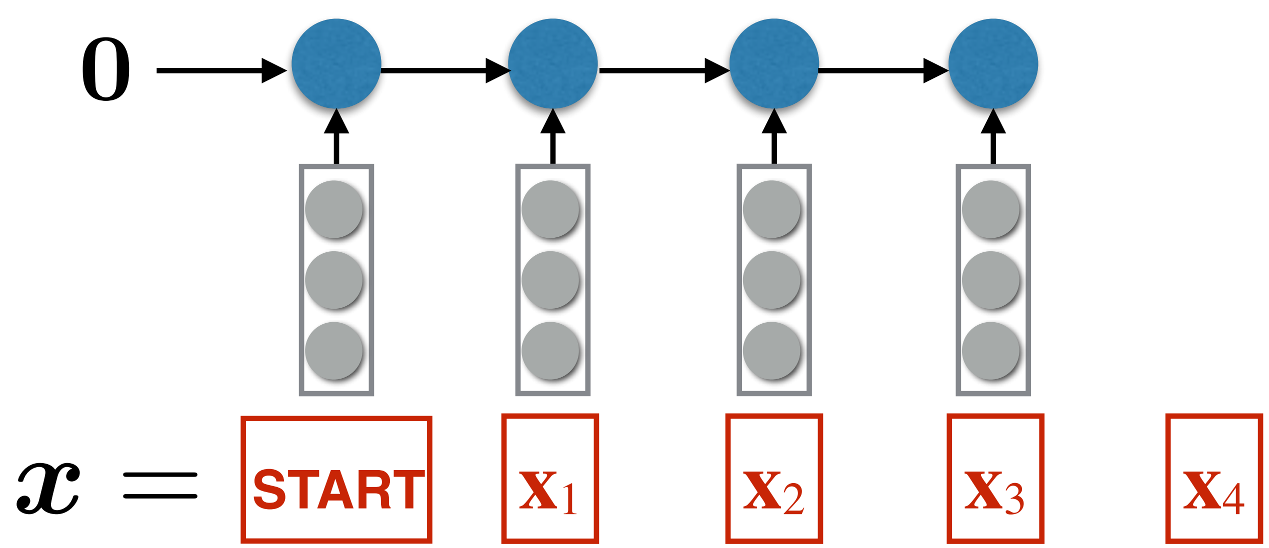


What is a vector representation of a sequence $\boldsymbol{x}$?

Recurrent Neural Networks (RNNs)

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

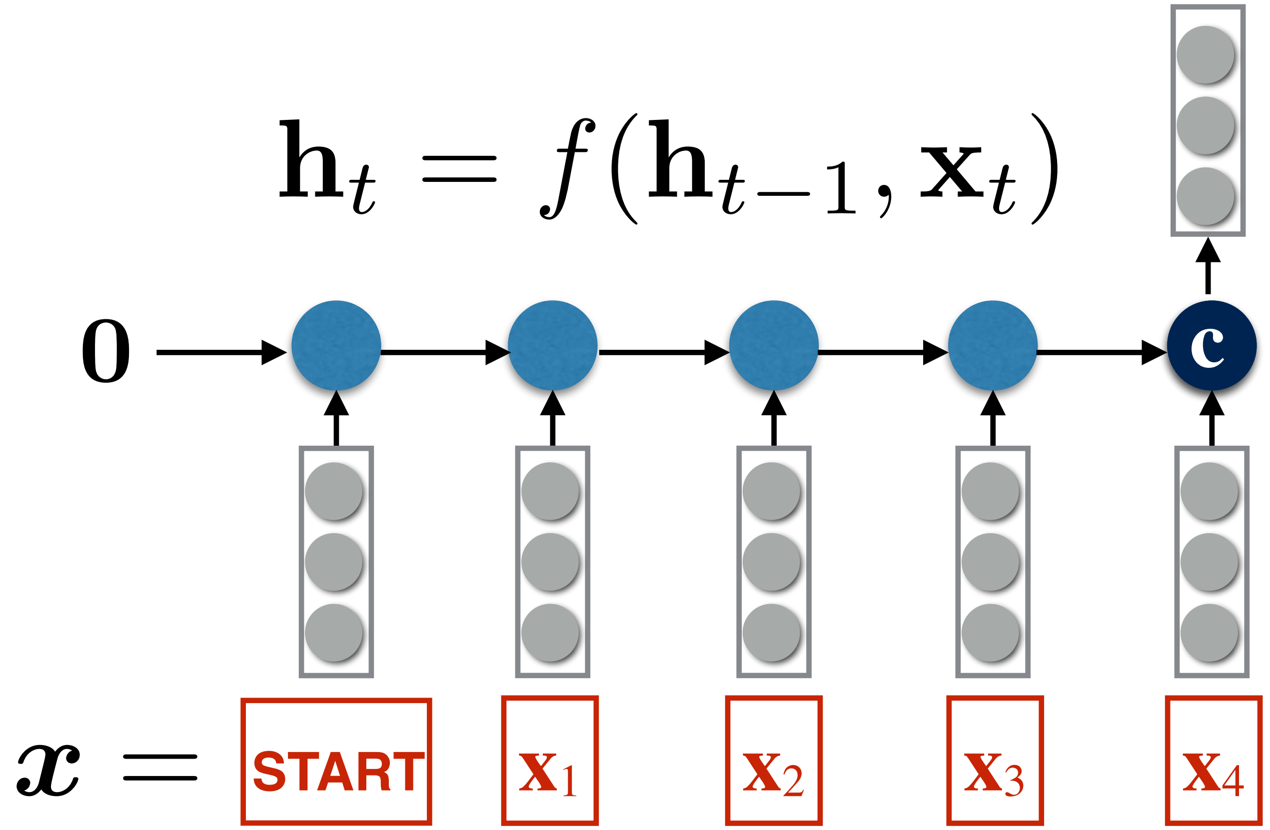
$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$



What is a vector representation of a sequence $\boldsymbol{x}$?

Recurrent Neural Networks (RNNs)

$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$



What is a vector representation of a sequence $\boldsymbol{x}$?

RNN Encoder-Decoders

$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$

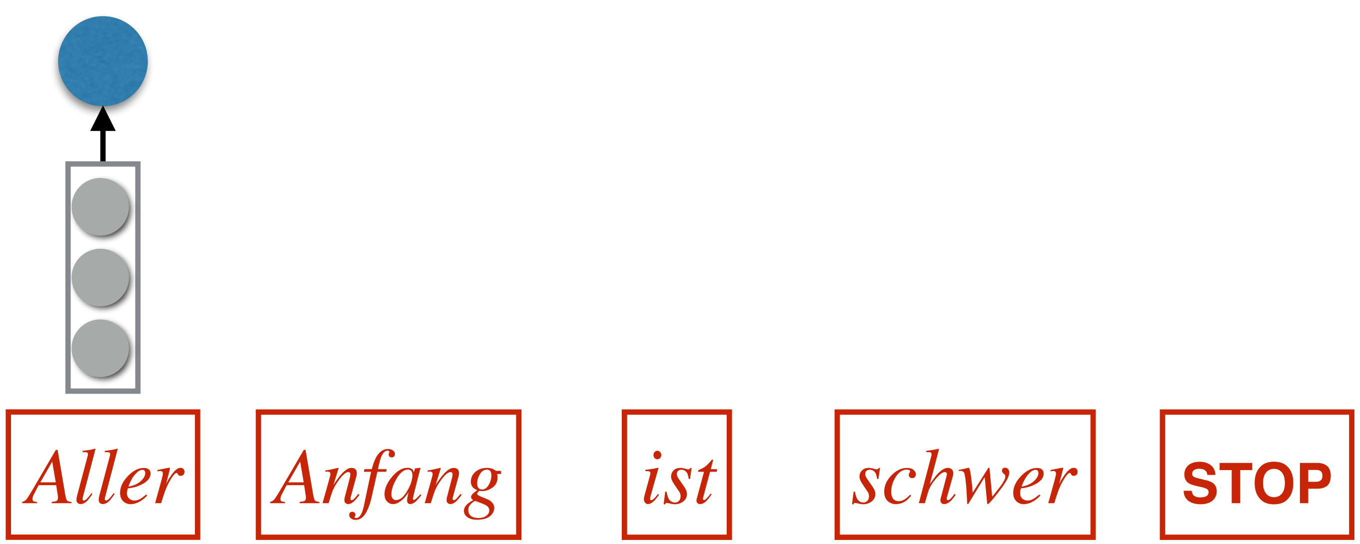
$\boxed{\textit{Aller}}$ $\boxed{\textit{Anfang}}$ $\boxed{\textit{ist}}$ $\boxed{\textit{schwer}}$ $\boxed{\textbf{STOP}}$

What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$?

Cho et al. (2014); Sutskever et al. (2014)

RNN Encoder-Decoders

$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$

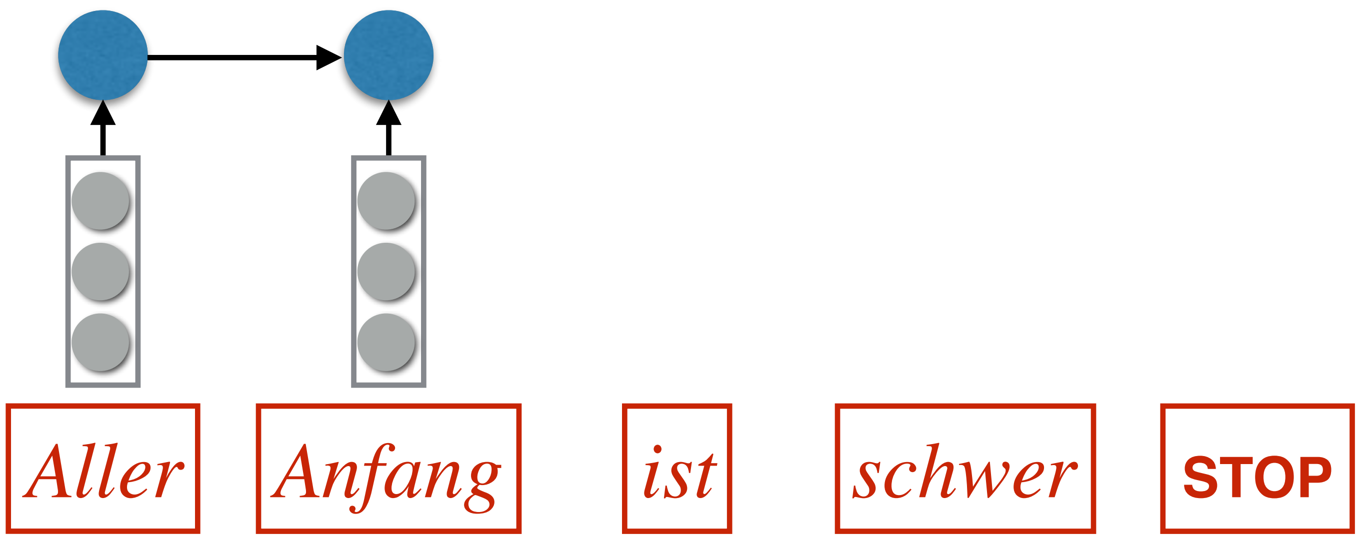


What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$ **?**

Cho et al. (2014); Sutskever et al. (2014)

RNN Encoder-Decoders

$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$

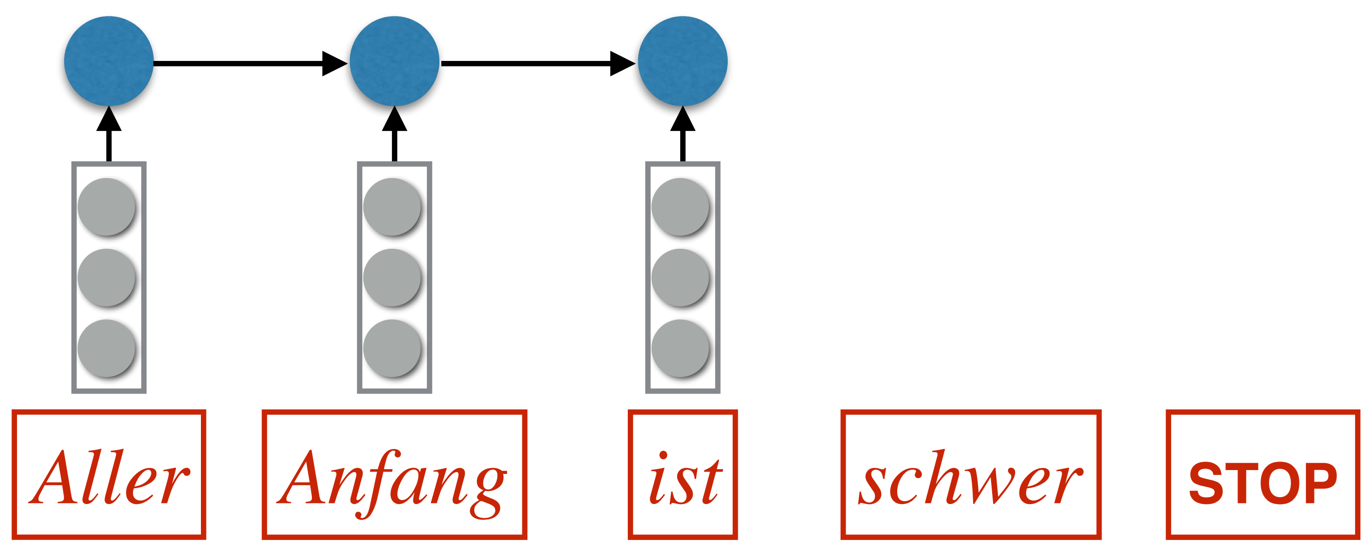


What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$?

Cho et al. (2014); Sutskever et al. (2014)

RNN Encoder-Decoders

$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$

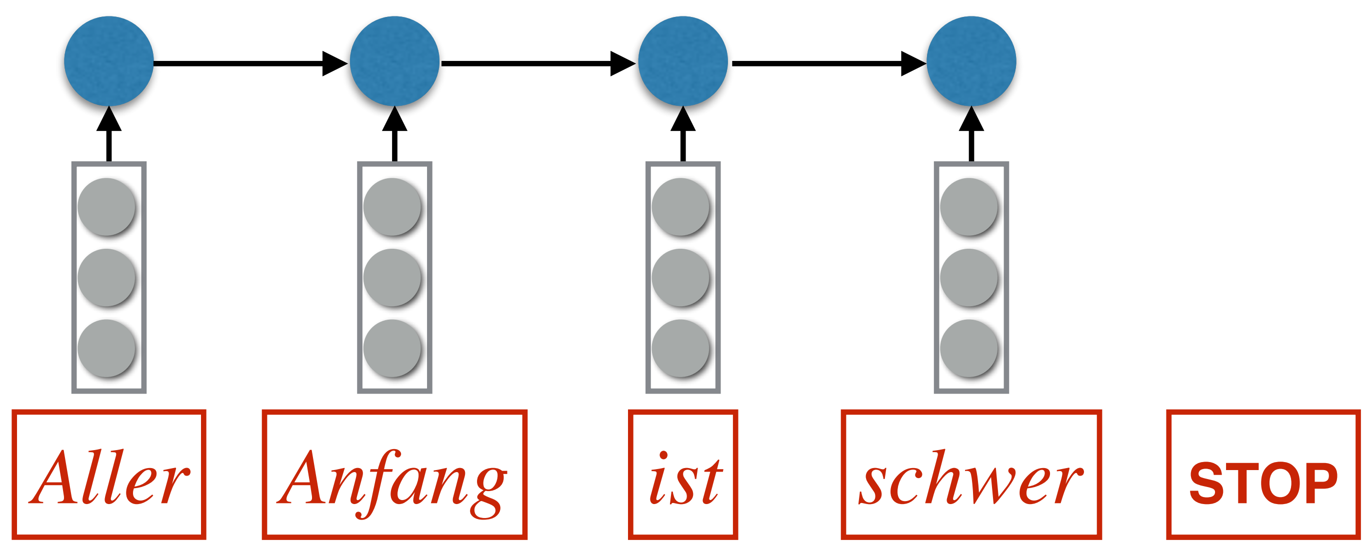


What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$?

Cho et al. (2014); Sutskever et al. (2014)

RNN Encoder-Decoders

$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$

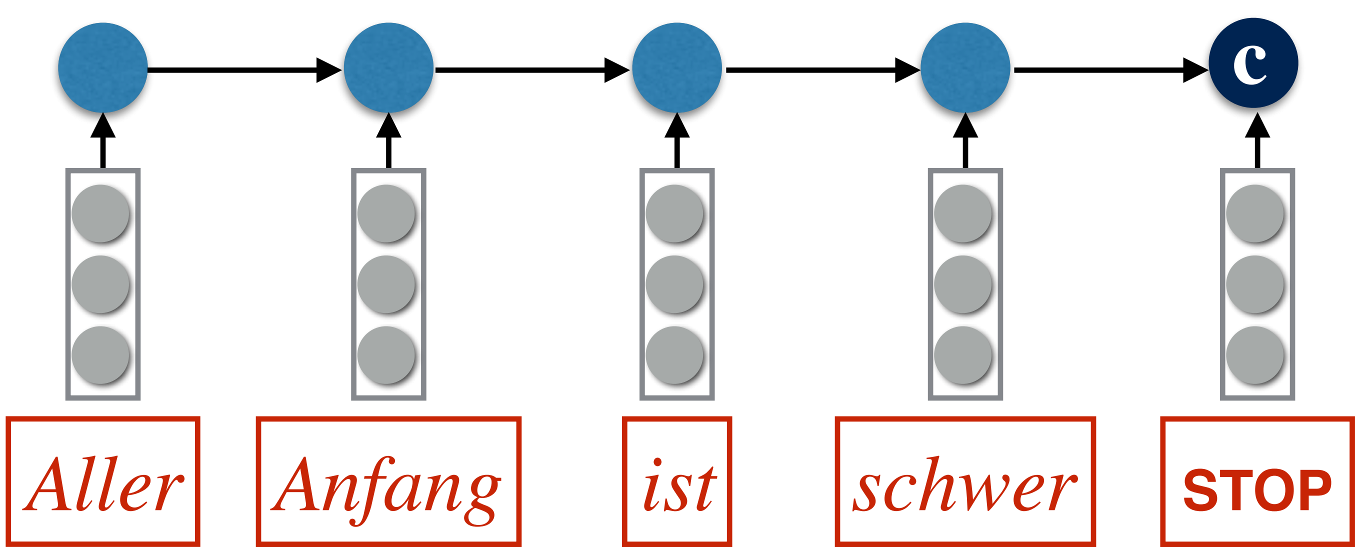


What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$**?**

Cho et al. (2014); Sutskever et al. (2014)

RNN Encoder-Decoders

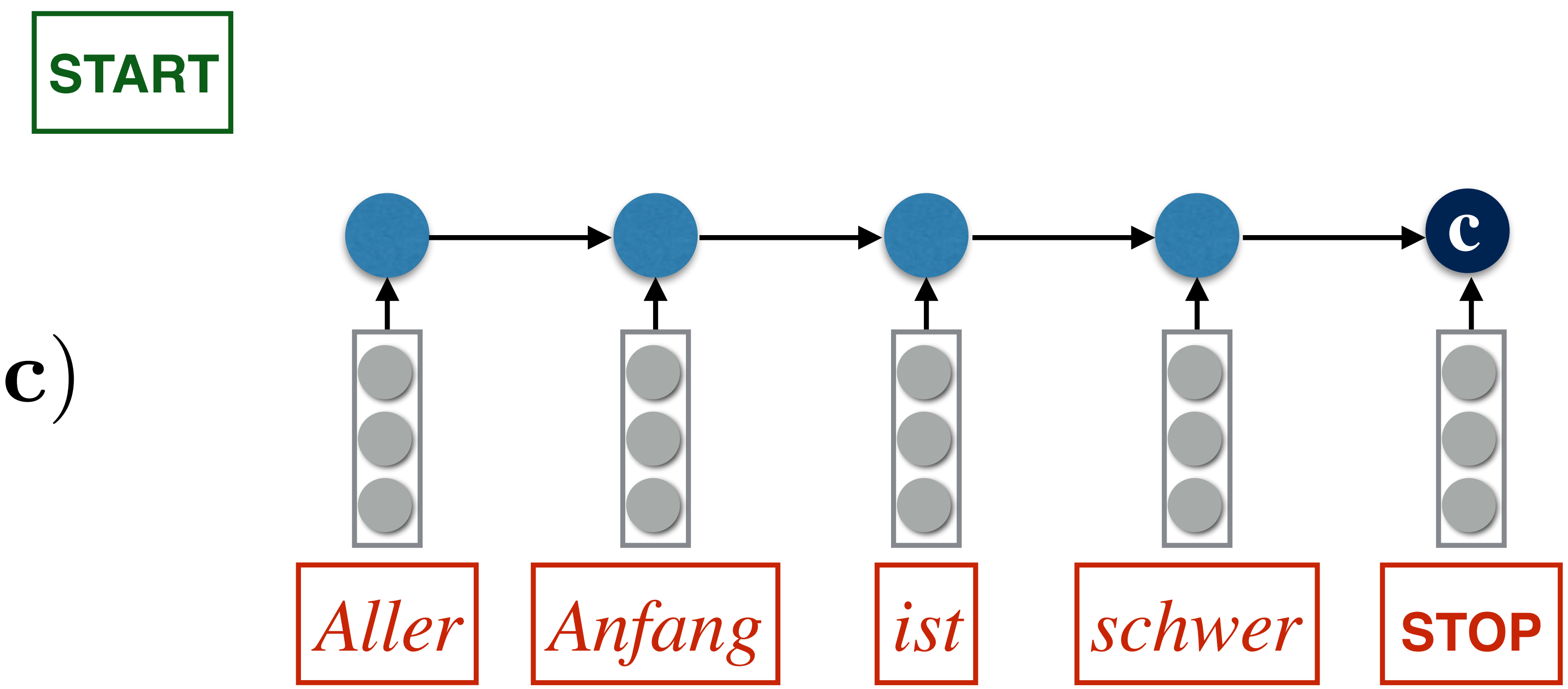
$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$



What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$?

Cho et al. (2014); Sutskever et al. (2014)

RNN Encoder-Decoders

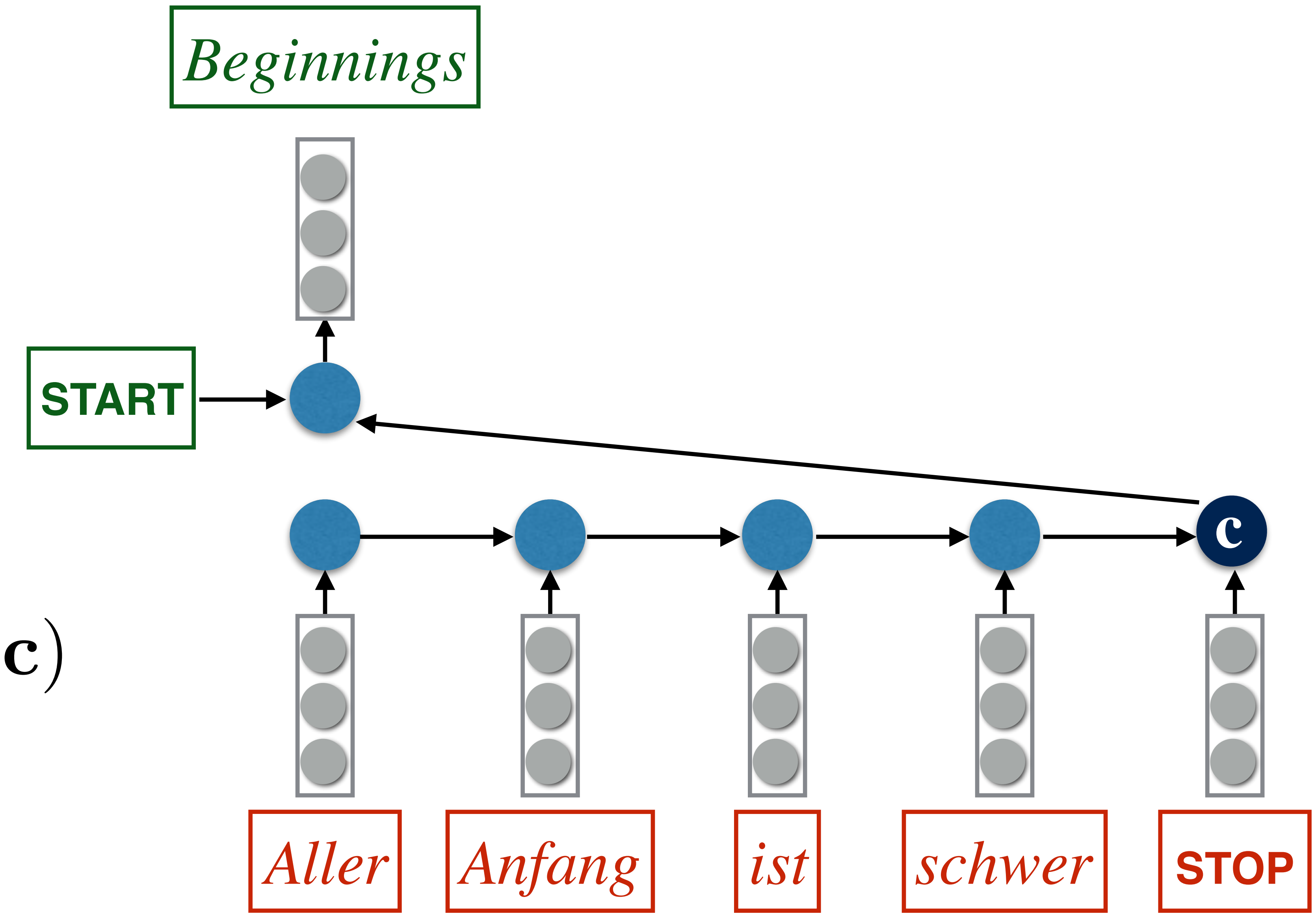


$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$
$$\boldsymbol{y} \mid \mathbf{c} \sim \mathrm{RNNLM}(\mathbf{c})$$

What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$?

Cho et al. (2014); Sutskever et al. (2014)

RNN Encoder-Decoders



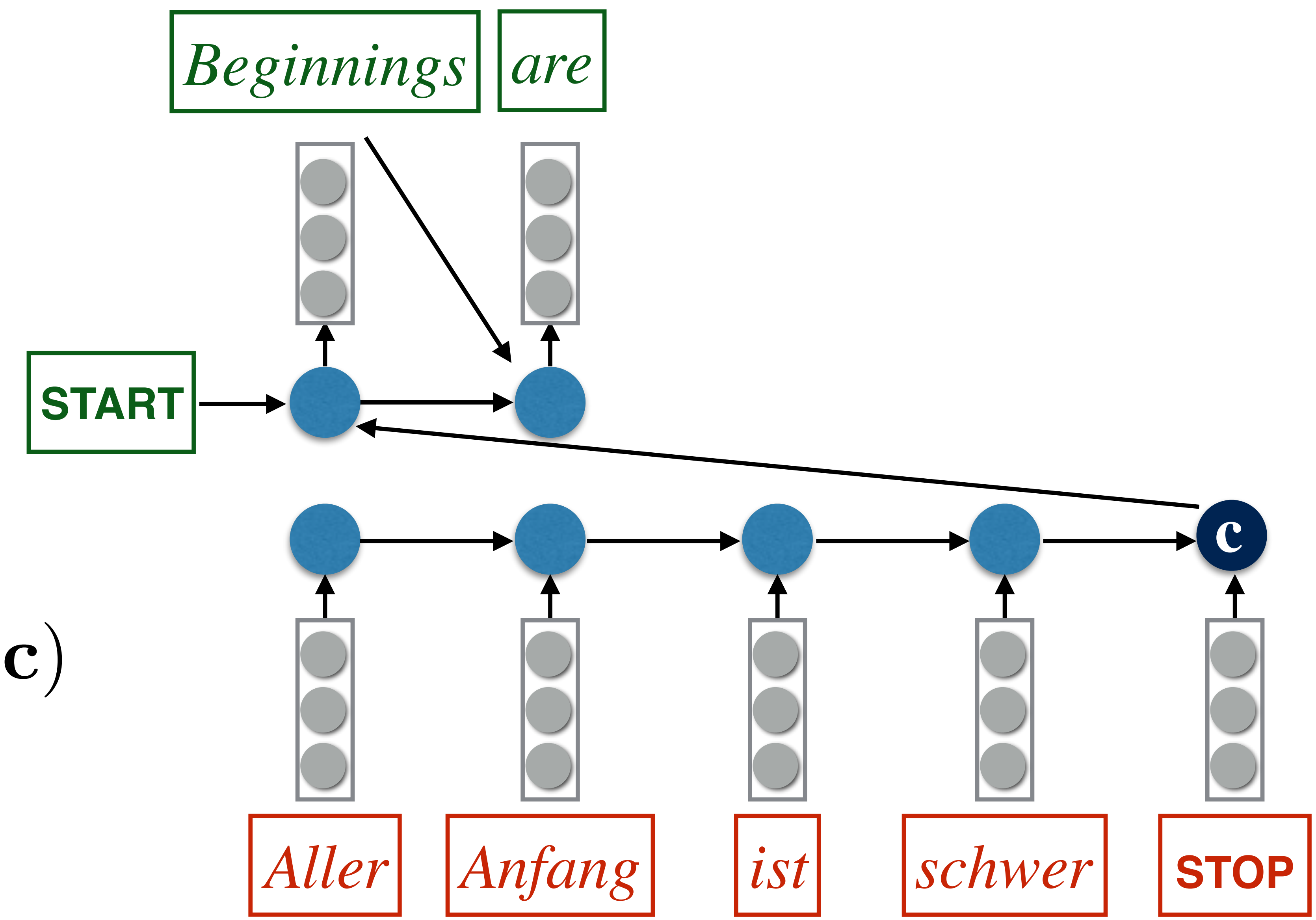
$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$

$$\boldsymbol{y} \mid \mathbf{c} \sim \mathrm{RNNLM}(\mathbf{c})$$

What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$?

Cho et al. (2014); Sutskever et al. (2014)

RNN Encoder-Decoders



$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$

$$\boldsymbol{y} \mid \mathbf{c} \sim \mathrm{RNNLM}(\mathbf{c})$$

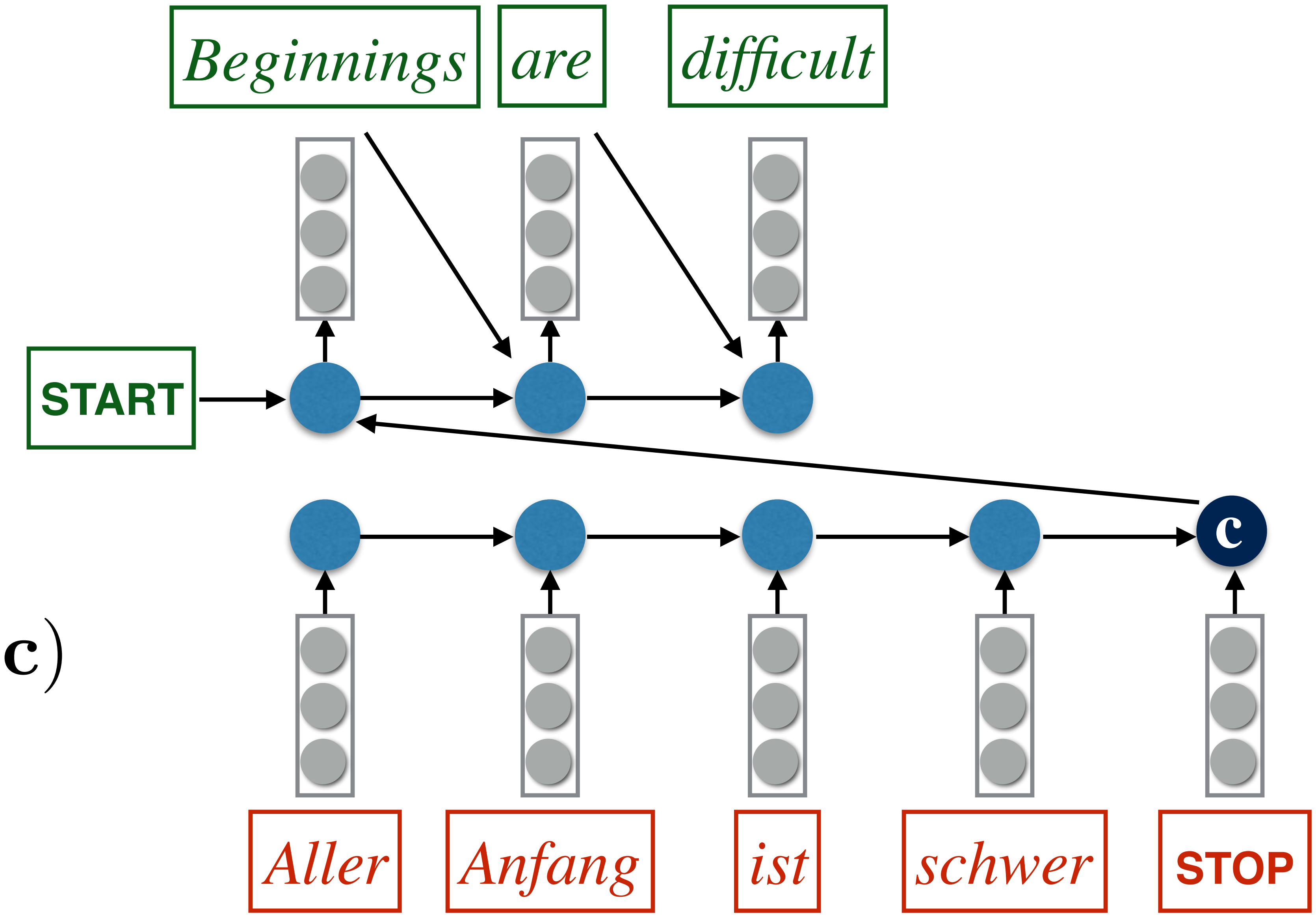
What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$?

Cho et al. (2014); Sutskever et al. (2014)

RNN Encoder-Decoders

$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$

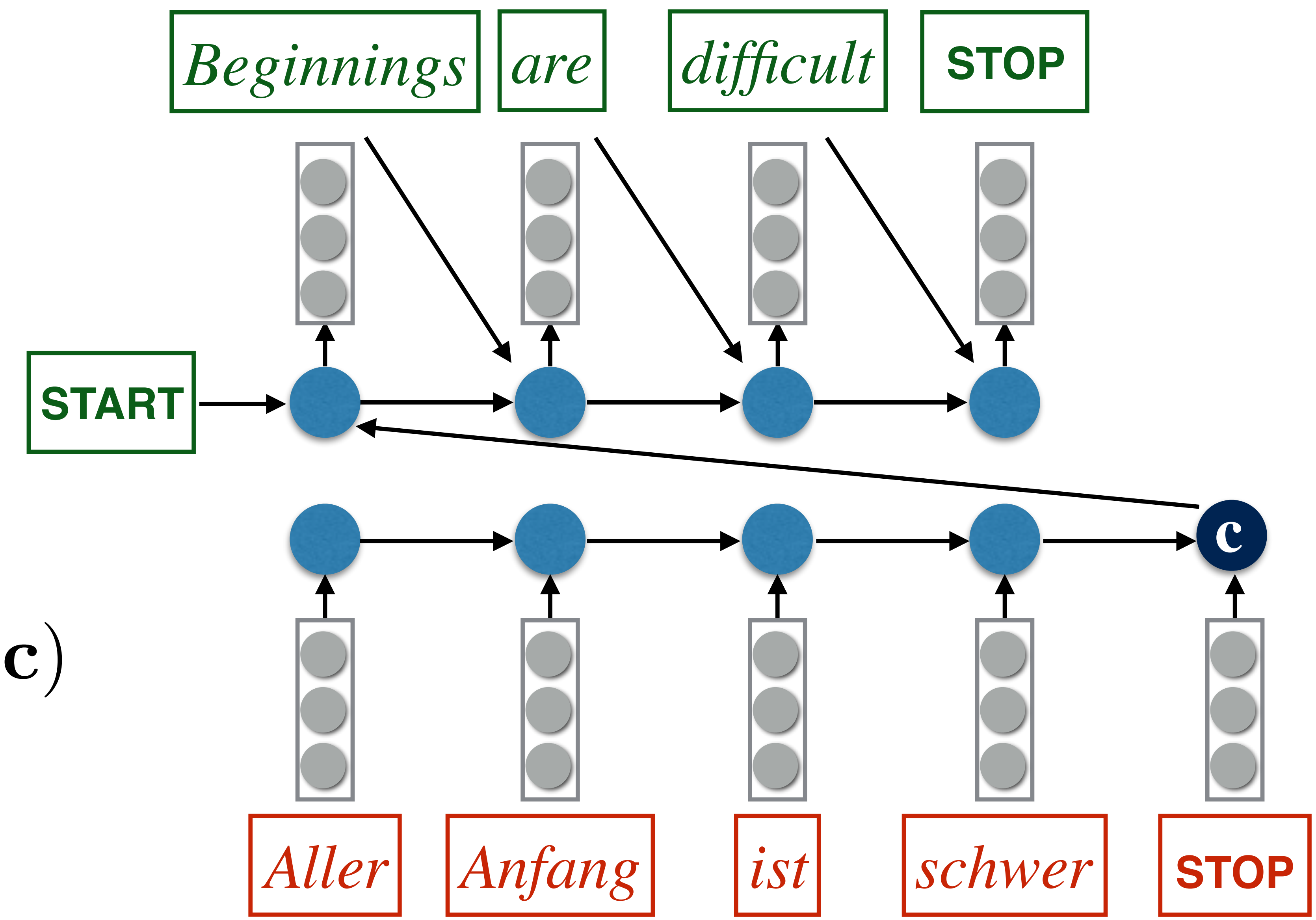
$$\boldsymbol{y} \mid \mathbf{c} \sim \mathrm{RNNLM}(\mathbf{c})$$



What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$**?**

Cho et al. (2014); Sutskever et al. (2014)

RNN Encoder-Decoders

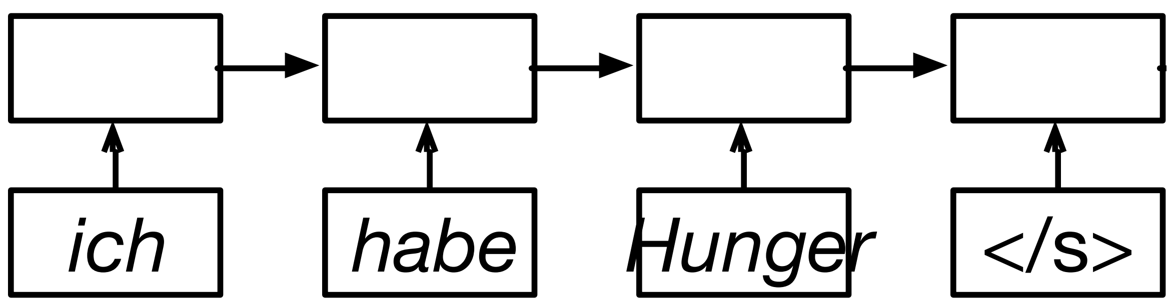


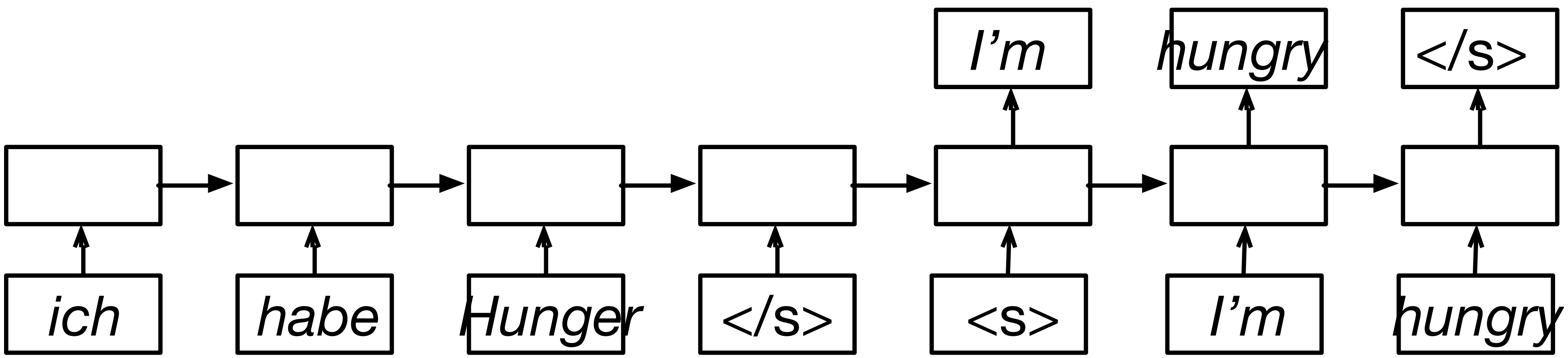
$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$

$$\boldsymbol{y} \mid \mathbf{c} \sim \mathrm{RNNLM}(\mathbf{c})$$

What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$?

Cho et al. (2014); Sutskever et al. (2014)


ich
habe
Hunger
</s>


I’m
hungry
</s>
ich
habe
Hunger
</s>
<s>
I’m
hungry

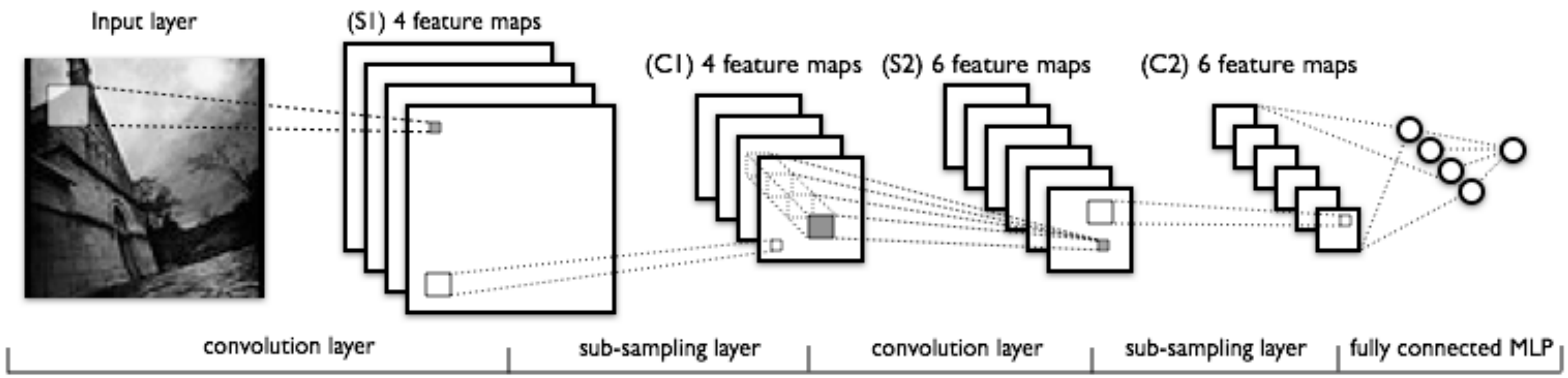
Sutskever et al. (2014)

Method	**test BLEU score (ntst14)**
Bahdanau et al. [2]	28.45
Baseline System [29]	33.30
Single forward LSTM, beam size 12	26.17
Single reversed LSTM, beam size 12	30.59
Ensemble of 5 reversed LSTMs, beam size 1	33.00
Ensemble of 2 reversed LSTMs, beam size 12	33.27
Ensemble of 5 reversed LSTMs, beam size 2	34.50
Ensemble of 5 reversed LSTMs, beam size 12	**34.81**

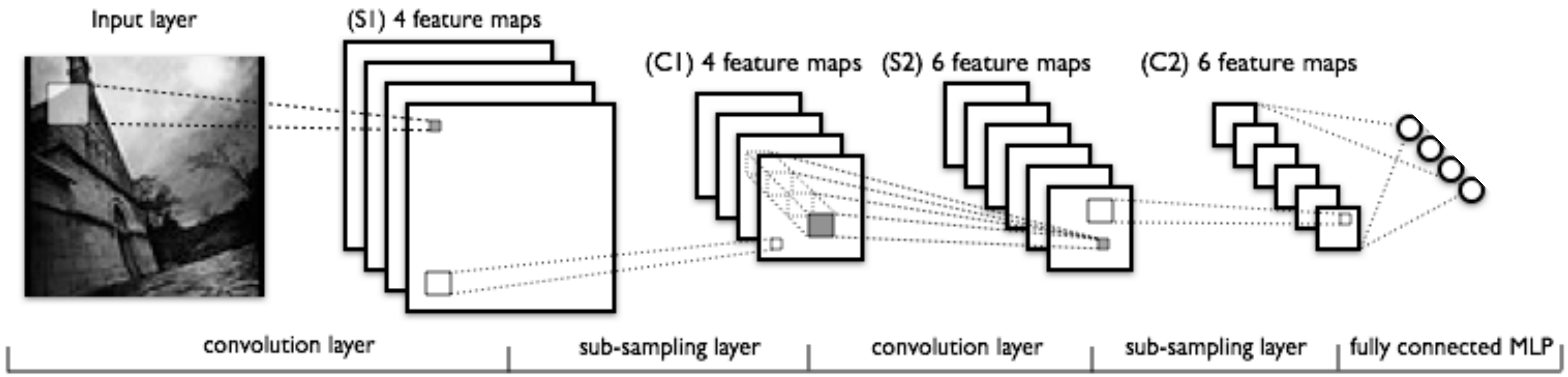
Ensembles of NNs

- Sutskever noticed that their single models did not work well
- But by combining N independently trained models and obtaining a "consensus", the performance could be improved a lot
- This is called **ensembling**.

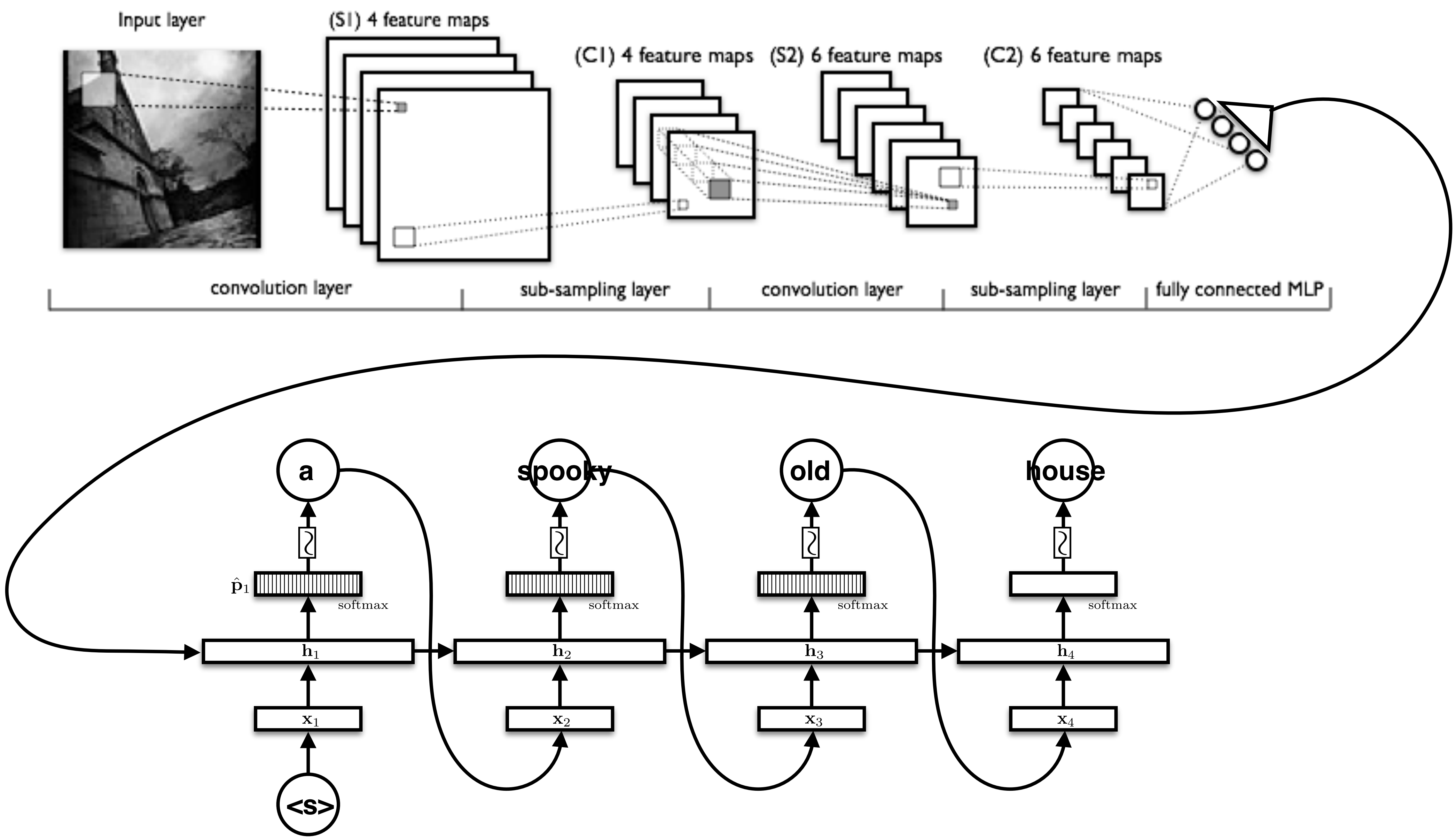
Encode anything as a vector!



Encode anything as a vector!



Encode anything as a vector!



Limitations

- A possible conceptual problem
  - Sentences have unbounded lengths
  - Vectors have finite capacity
- A possible practical problem
  - Distance between “translations” and their sources are distant- can LSTMs learn this?

Two Goals

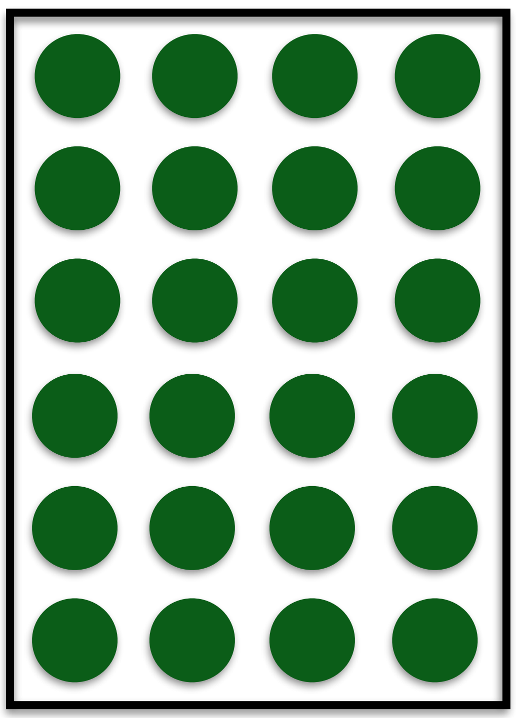
- Represent a source sentence as a matrix
- Generate a target sentence from a matrix

- These two steps are:
  - An algorithm for neural MT
  - A way of introducing **attention**

Sentences as Matrices

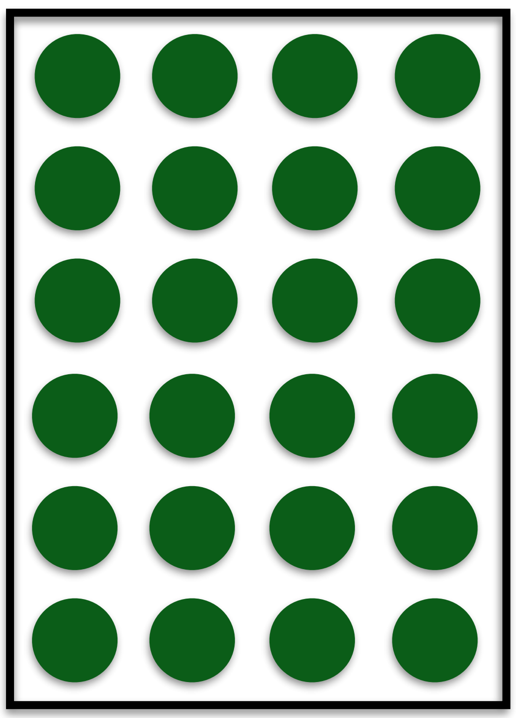
- Problem with the fixed-size vector model in translation (maybe in images?)
  - Sentences are of different sizes but vectors are of the same size
- Solution: use matrices instead
  - Fixed number of rows, but number of columns depends on the number of words
  - Usually $|\boldsymbol{f}|$ = #cols

Sentences as Matrices

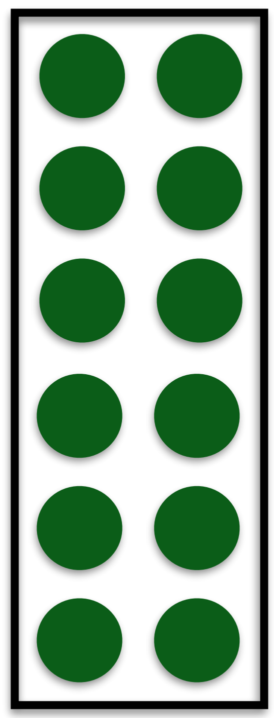


Ich möchte ein Bier

Sentences as Matrices

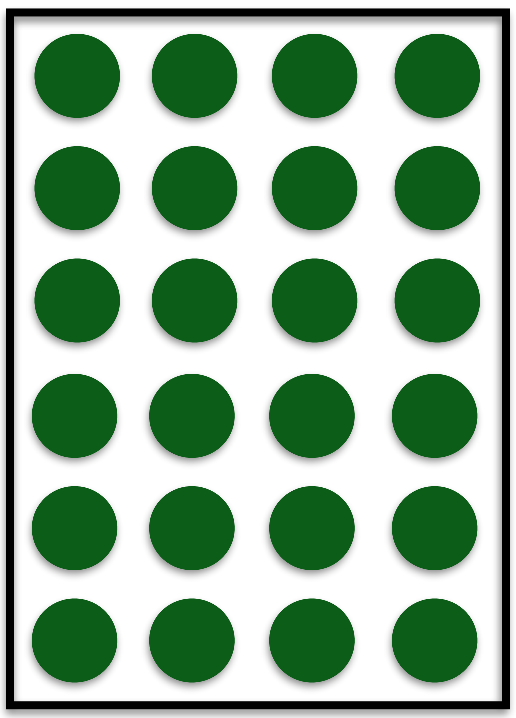


Ich möchte ein Bier

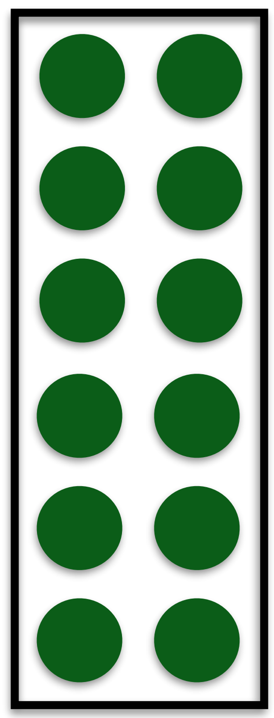


Mach's gut

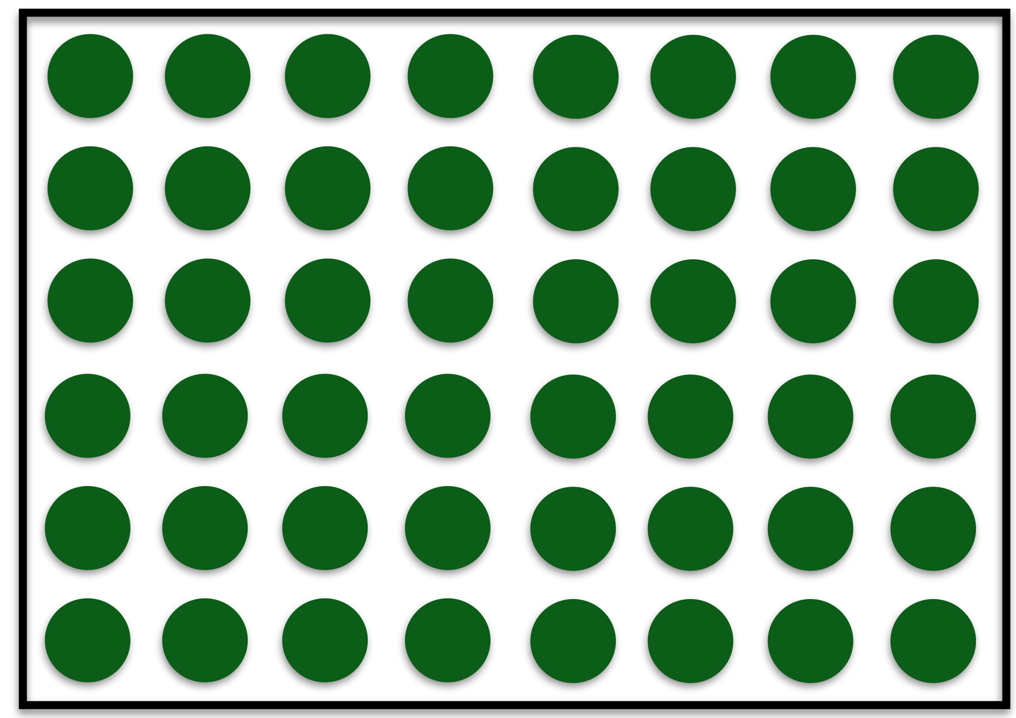
Sentences as Matrices



Ich möchte ein Bier

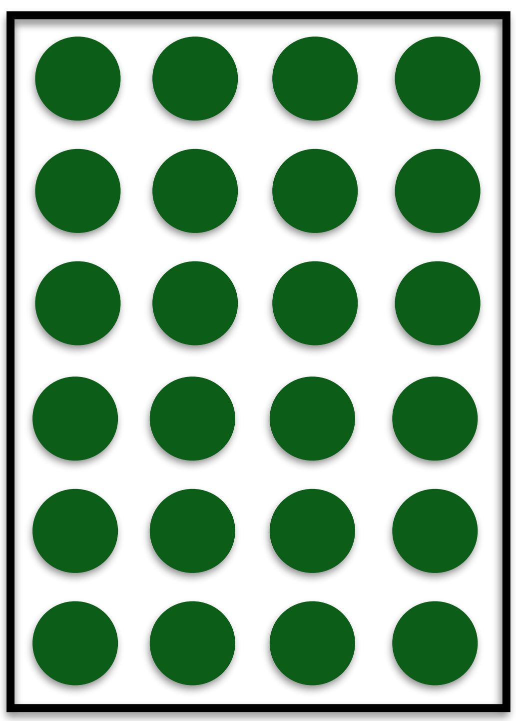


Mach's gut

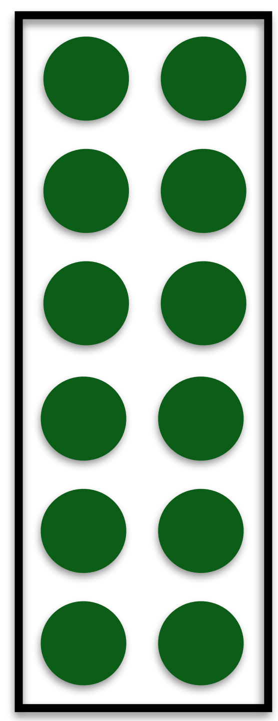


Die Wahrheiten der Menschen sind die unwiderlegbaren Irrtümer

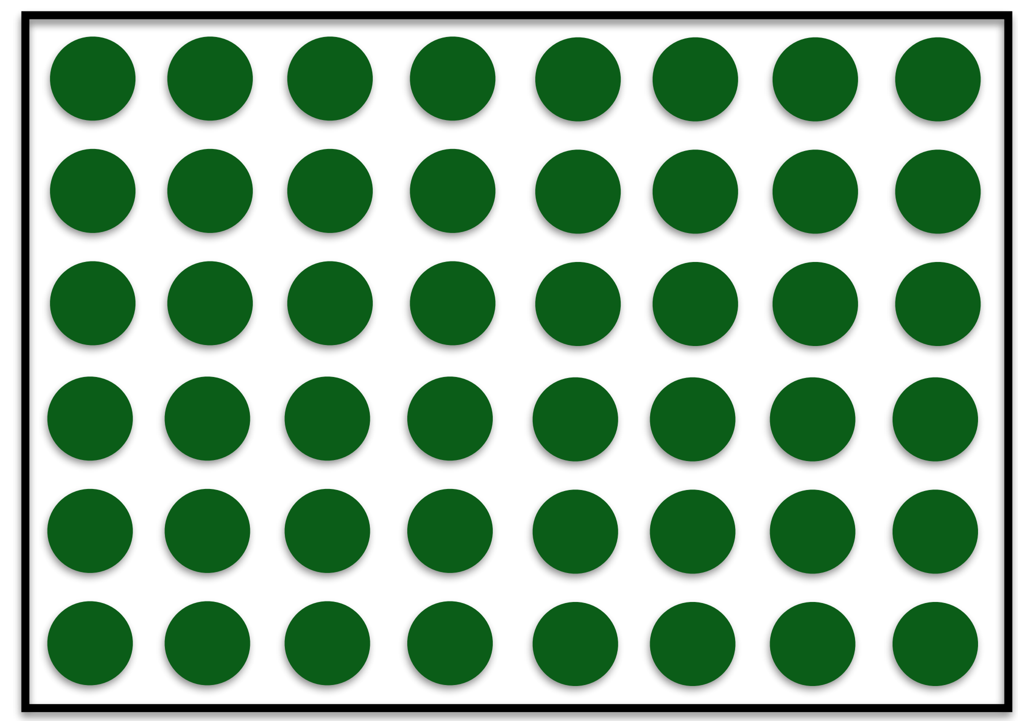
Sentences as Matrices



Ich möchte ein Bier



Mach's gut

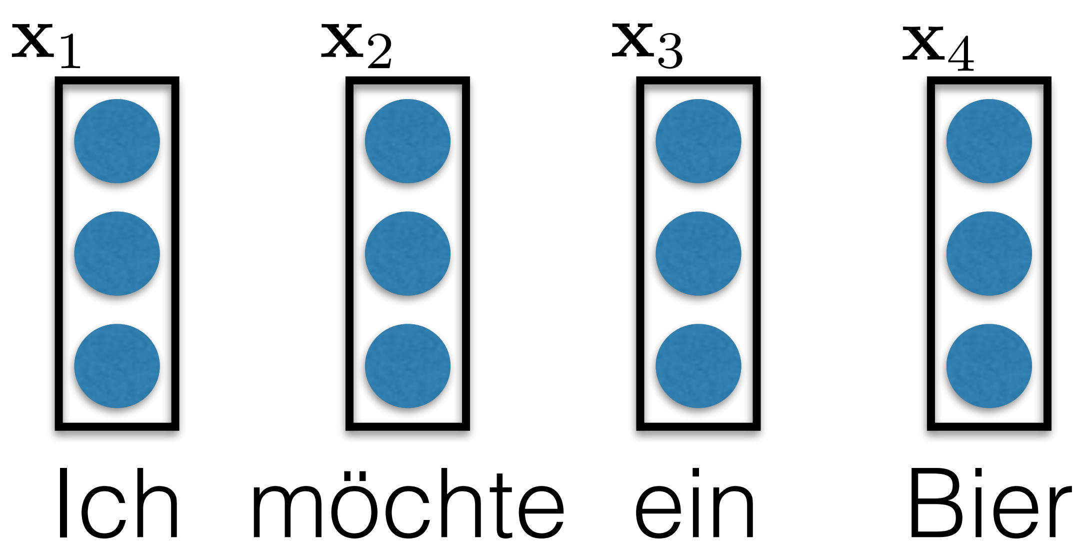


Die Wahrheiten der Menschen sind die unwiderlegbaren Irrtümer

Question: How do we build these matrices?

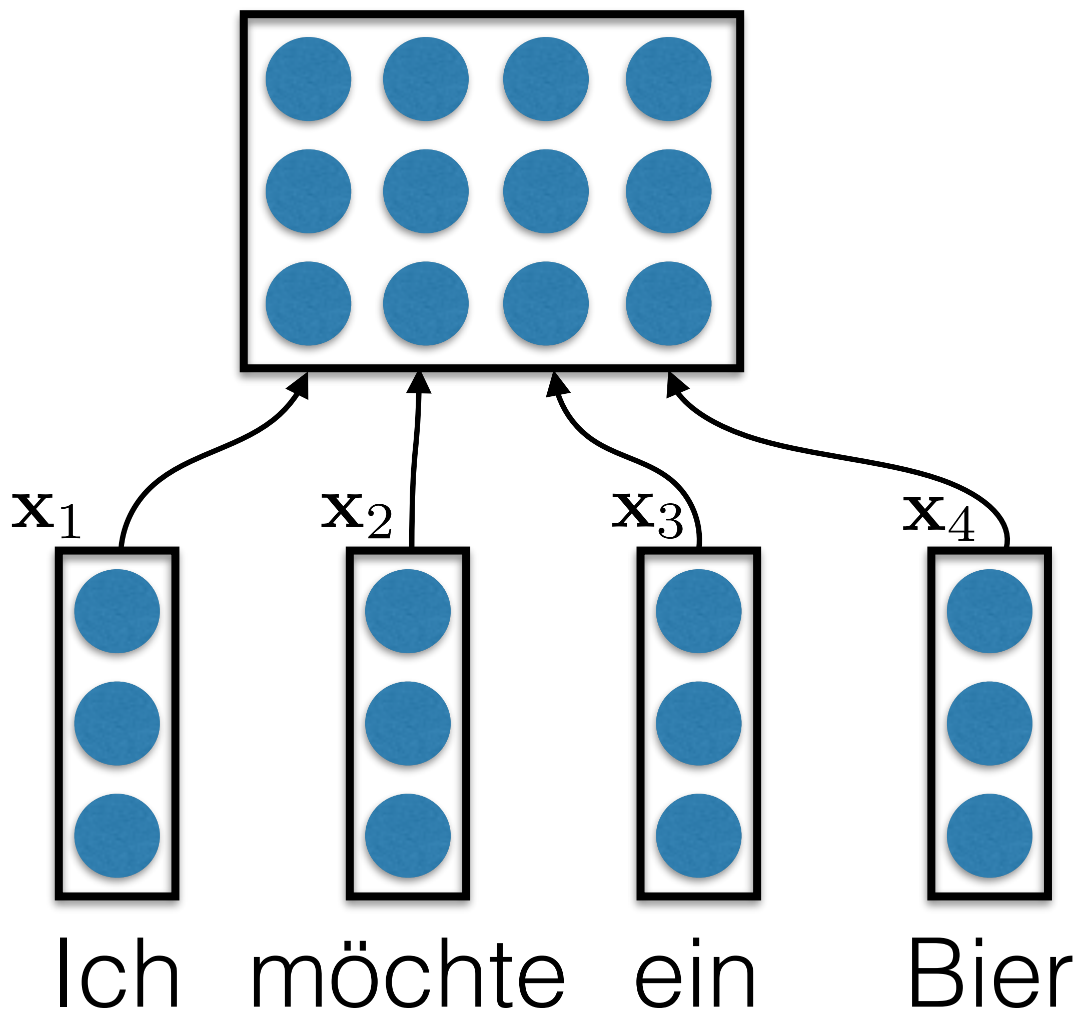
With Concatenation

- Each word type is represented by an n-dimensional vector
- Take all of the vectors for the sentence and concatenate them into a matrix
- Simplest possible model
    - So simple, no one has bothered to publish how well/badly it works!

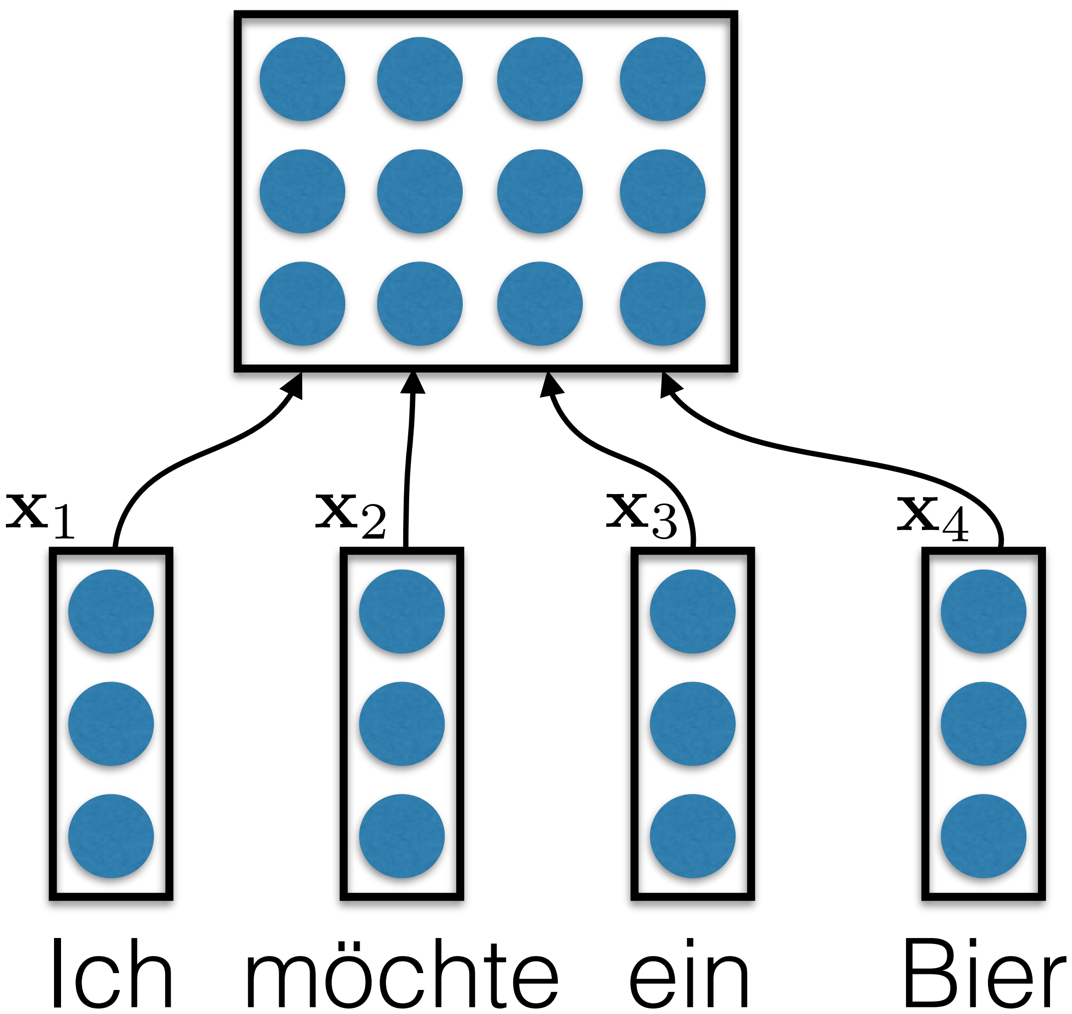


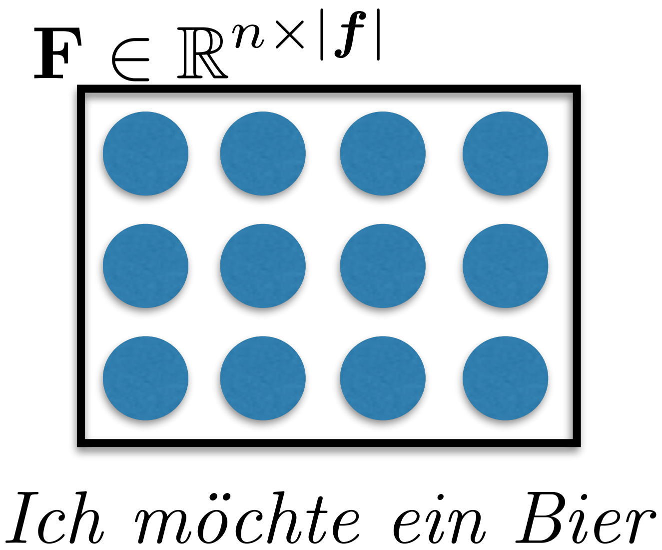
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier

$$\mathbf{f}_i = \mathbf{x}_i$$



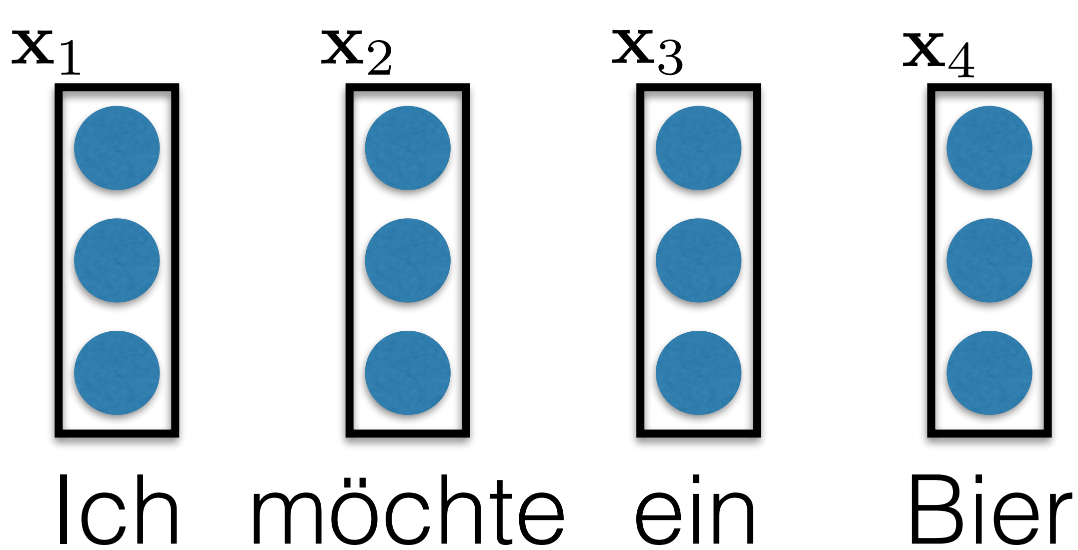
$$\mathbf{f}_i = \mathbf{x}_i$$

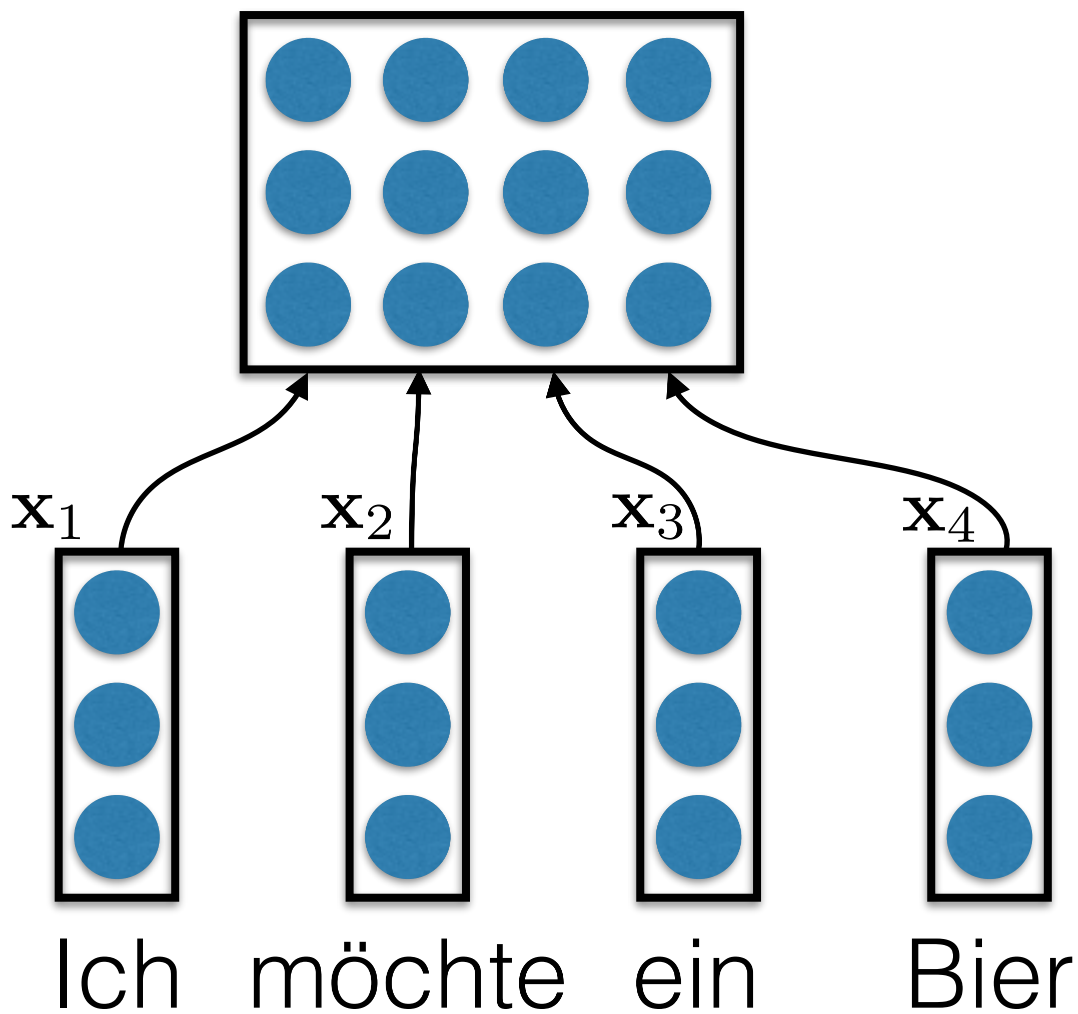


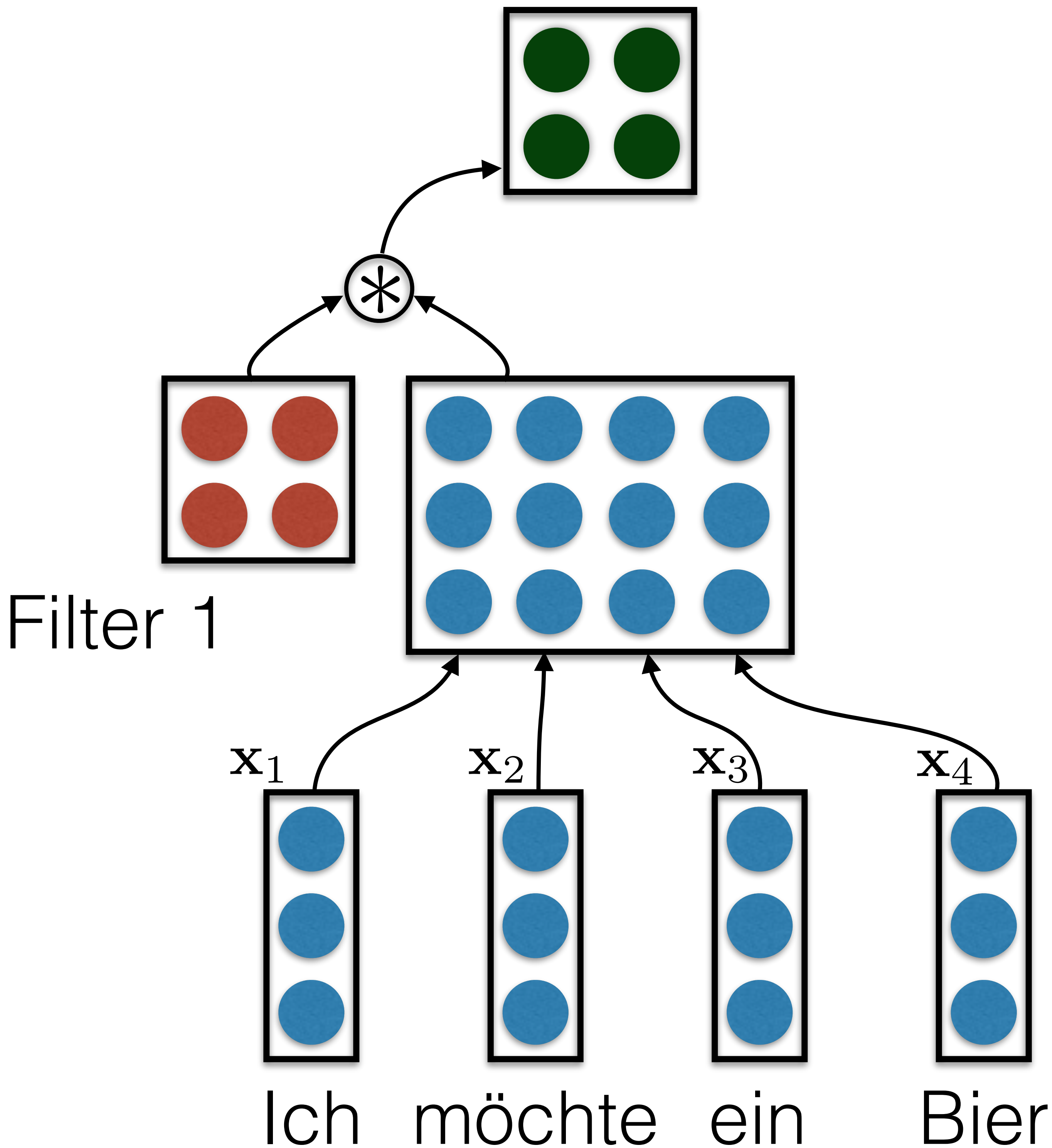


With Convolutional Nets

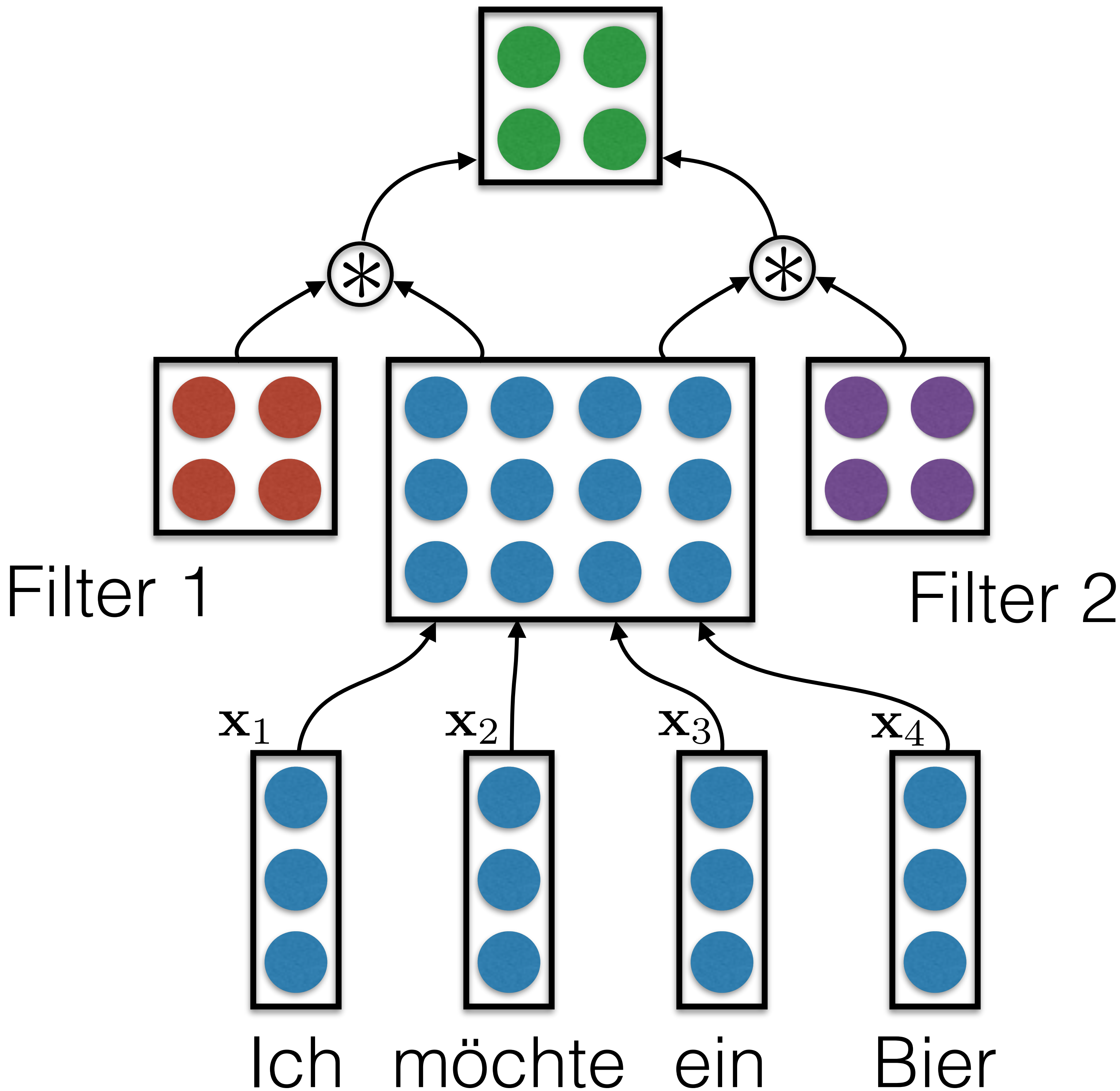
- Apply convolutional networks to transform the naive concatenated matrix to obtain a context-dependent matrix
- Closely related to the first “modern” neural translation model proposed (Kalchbrenner et al., 2013)
  - No one has been using convnets lately in MT (including Kalchbrenner et al, who are using BiLSTMs these days)
- Note: convnets usually have a “pooling” operation at the top level that results in a fixed-sized representation. For sentences, it is probably good to leave this out.


$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier

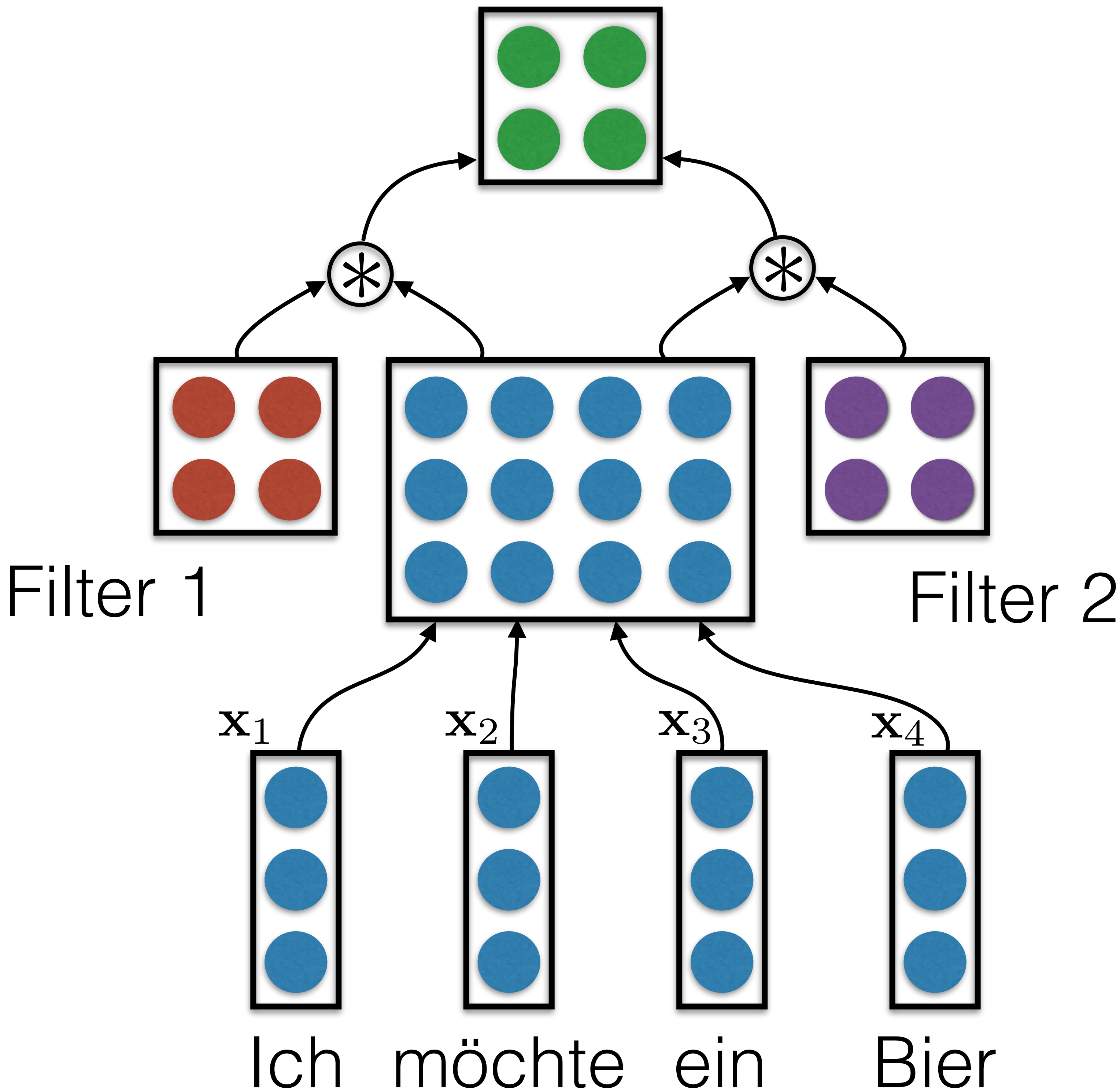

$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier



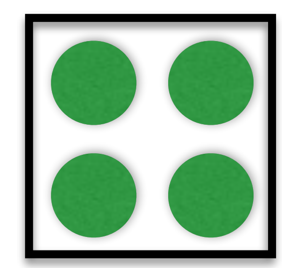
Filter 1
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier



Filter 1
Filter 2
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier



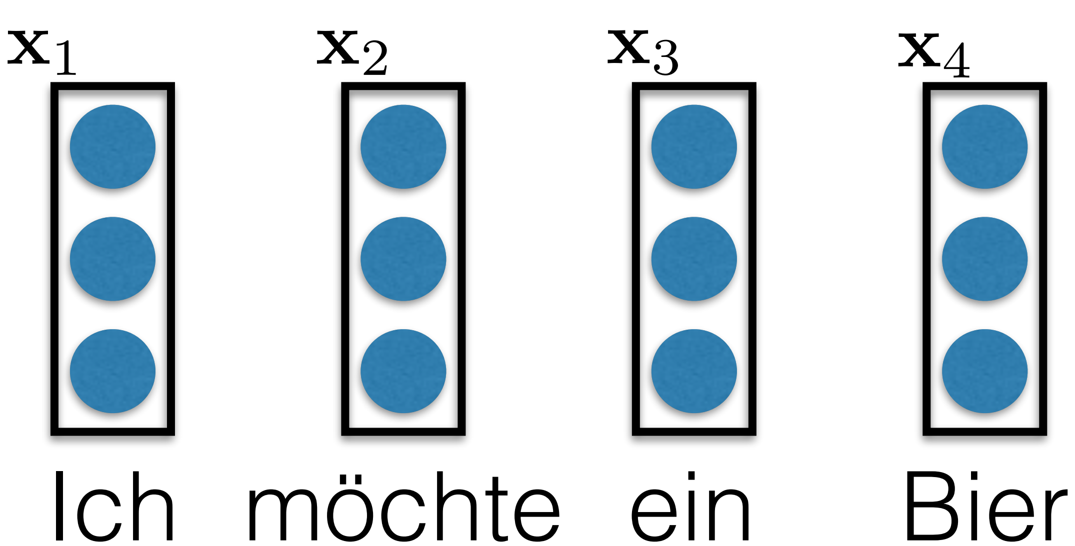
$\mathbf{F} \in \mathbb{R}^{f(n) \times g(|\boldsymbol{f}|)}$



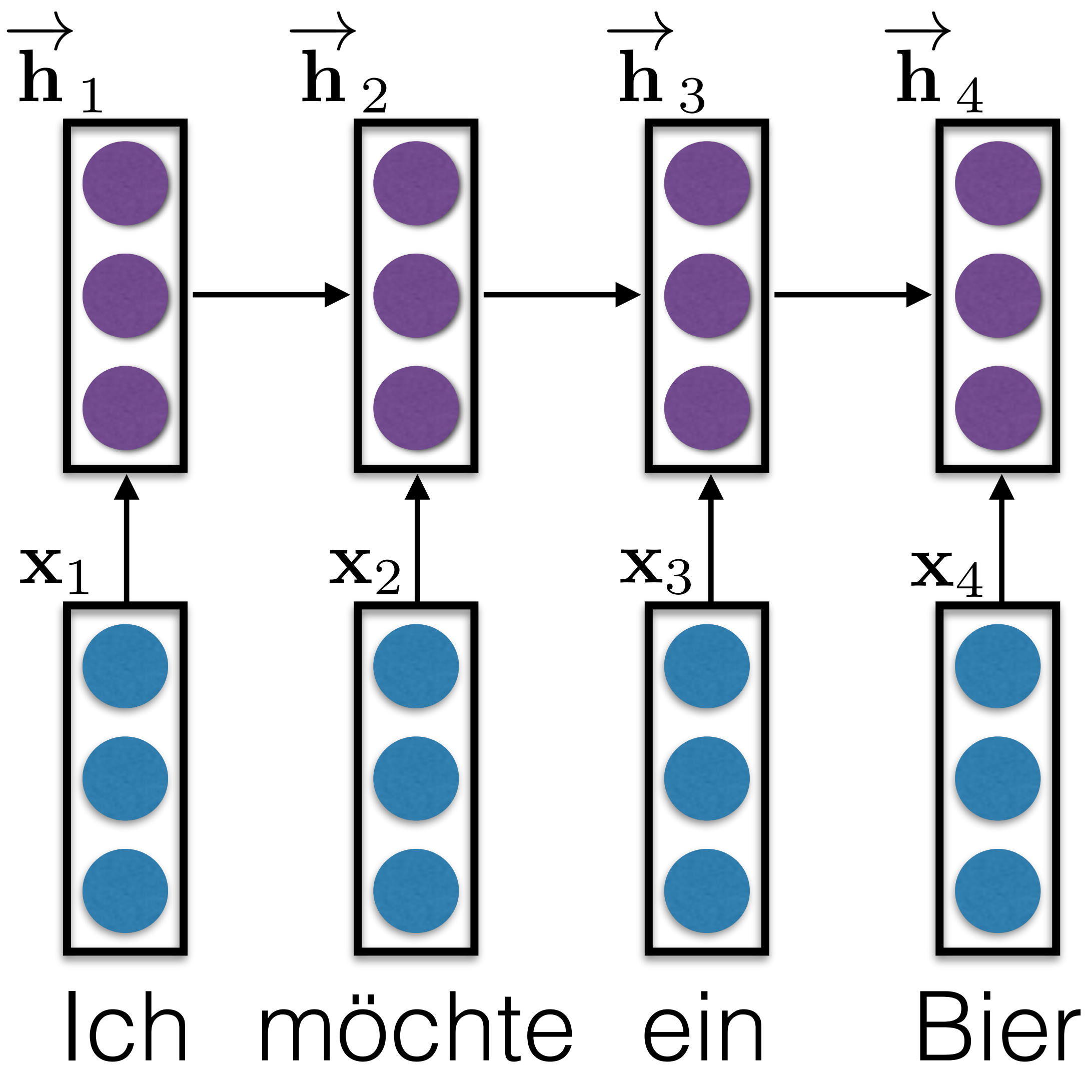
Ich möchte ein Bier

With Bidirectional RNNs

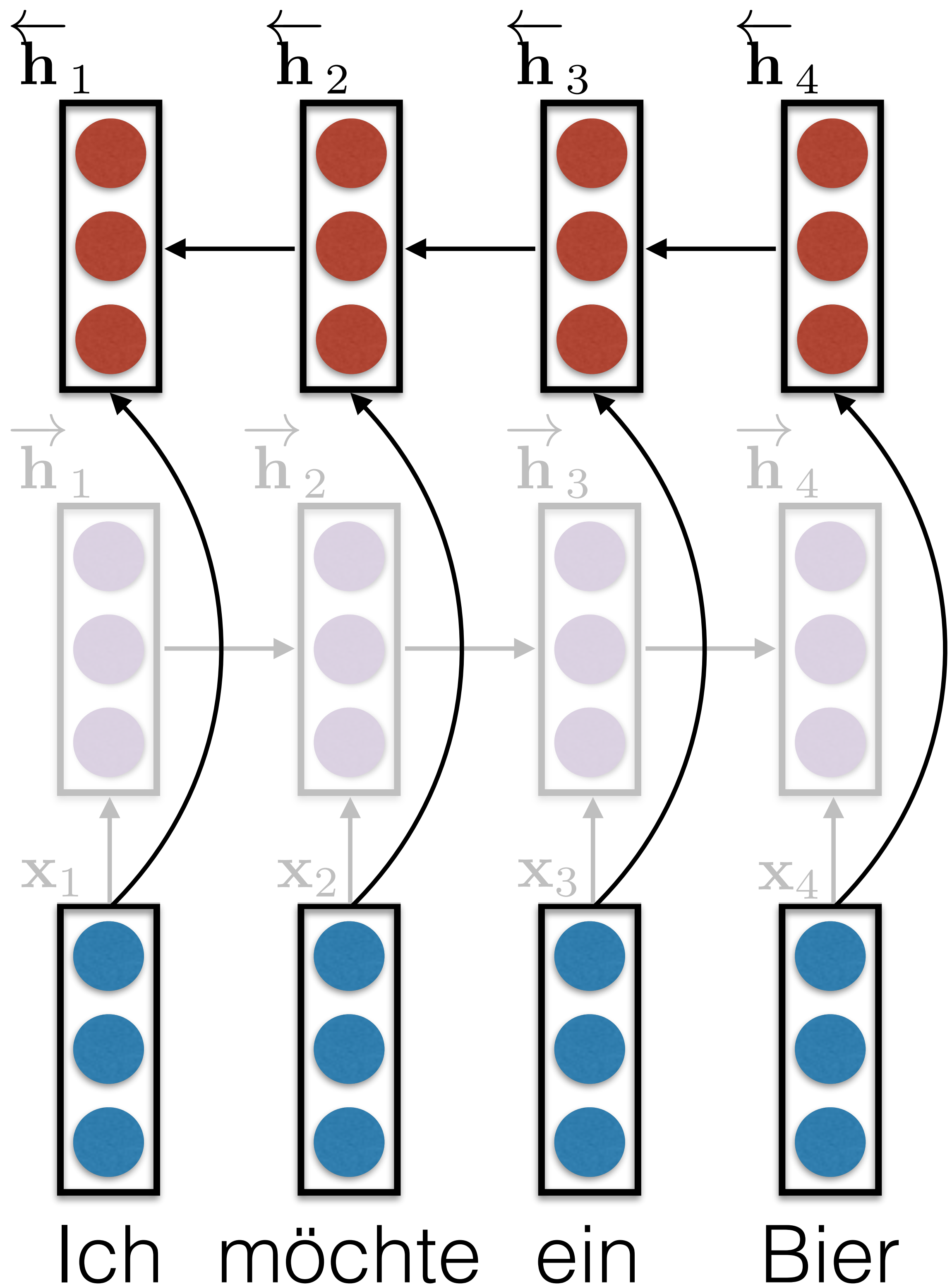
- By far the most widely used matrix representation, due to Bahdanau et al (2015)
- One column per word
- Each column (word) has two halves concatenated together:
    - a “forward representation”, i.e., a word and its left context
    - a “reverse representation”, i.e., a word and its right context
- Implementation: bidirectional RNNs (GRUs or LSTMs) to read $\boldsymbol{f}$ from left to right and right to left, concatenate representations



$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier

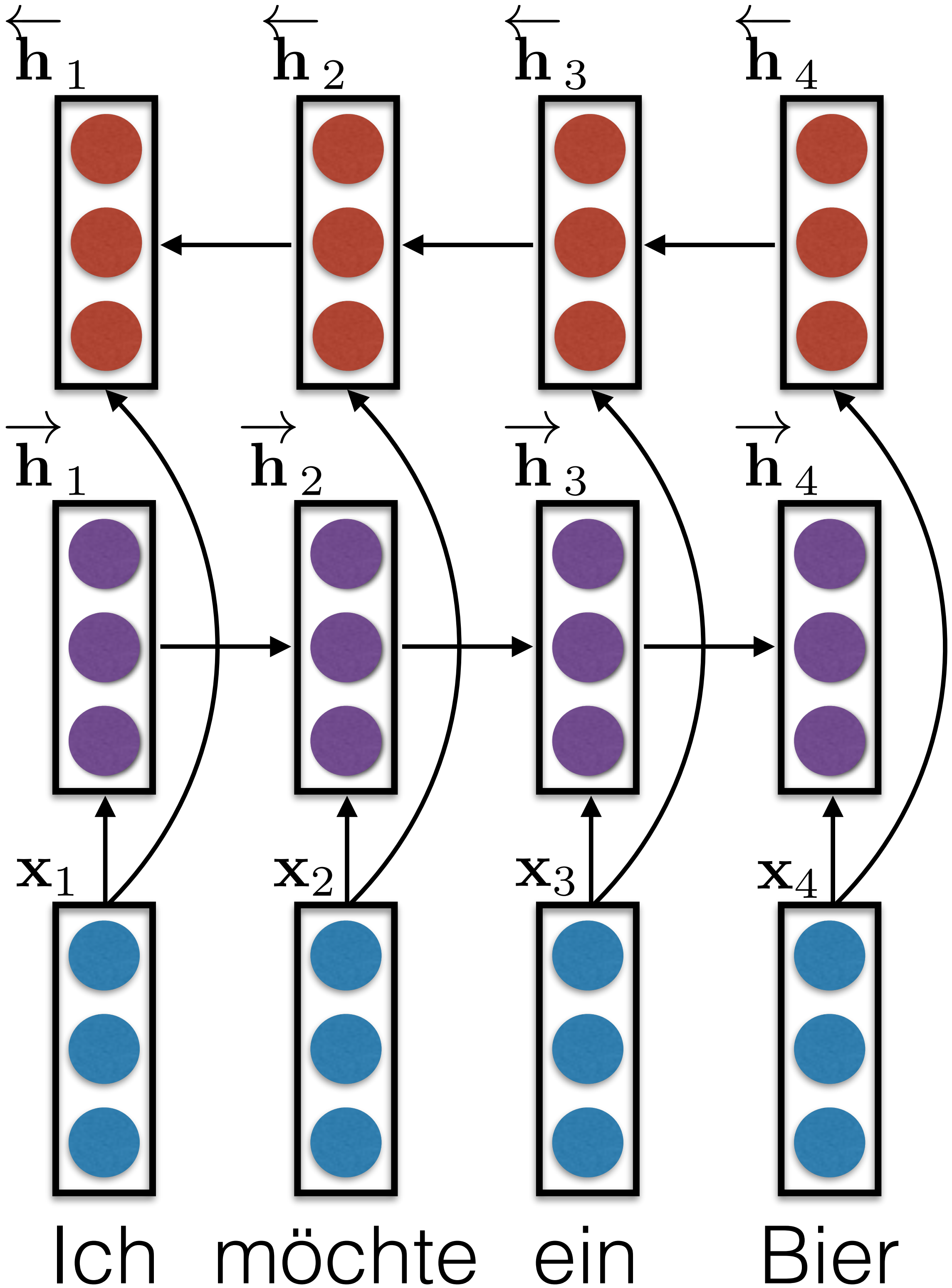


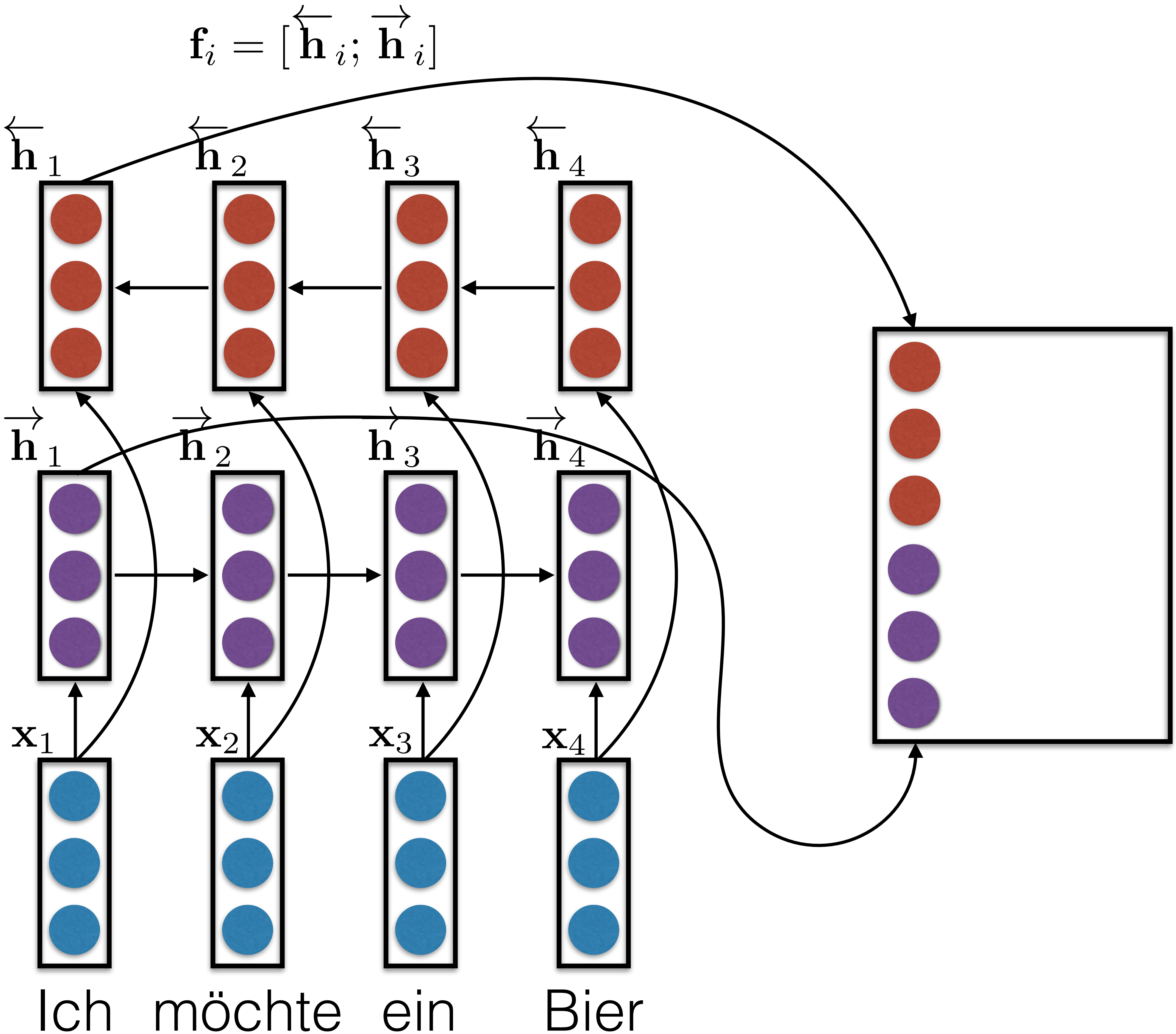
$\overrightarrow{\mathbf{h}}_1$
$\overrightarrow{\mathbf{h}}_2$
$\overrightarrow{\mathbf{h}}_3$
$\overrightarrow{\mathbf{h}}_4$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier

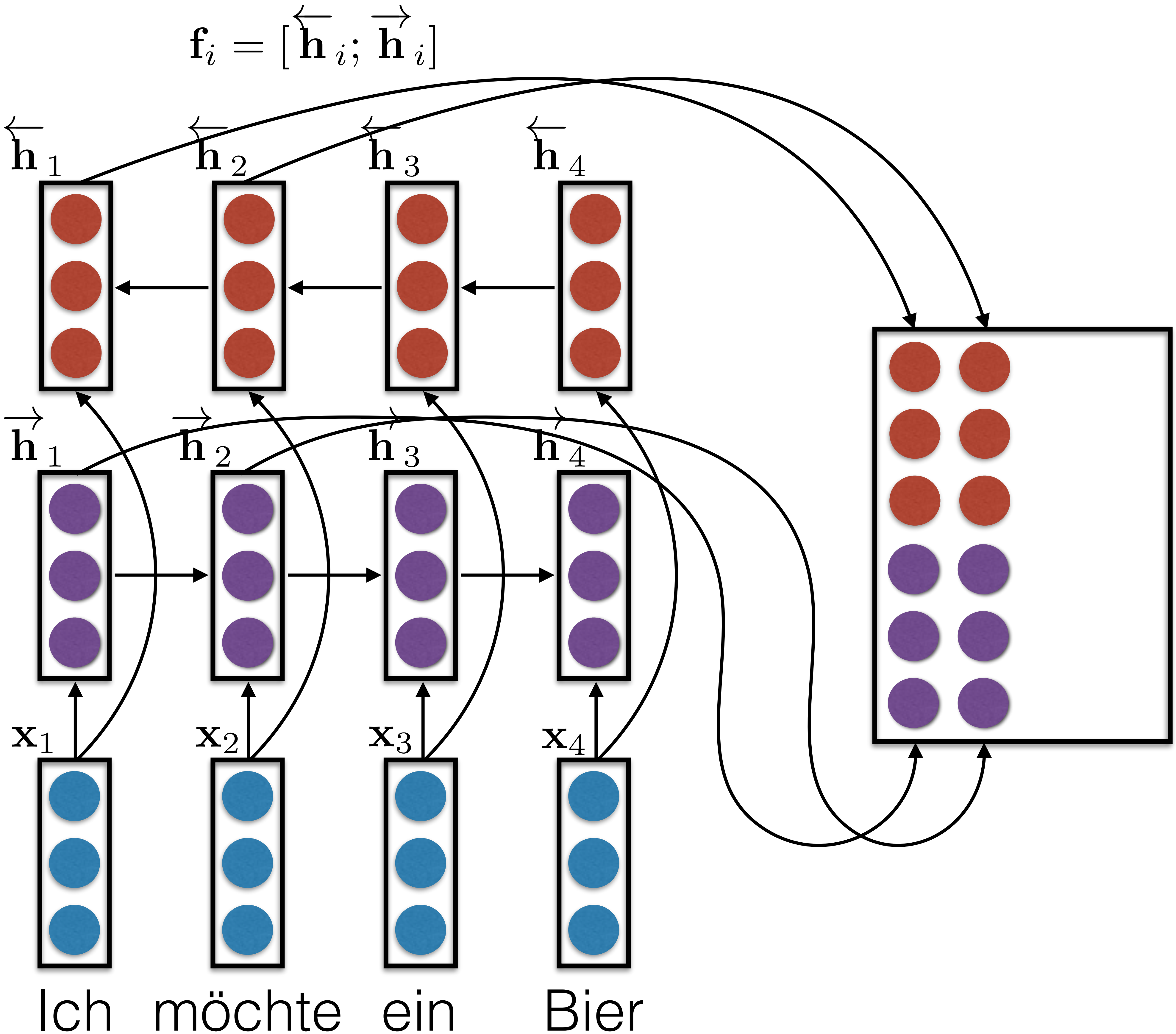


$\overleftarrow{\mathbf{h}}_1$ $\overleftarrow{\mathbf{h}}_2$ $\overleftarrow{\mathbf{h}}_3$ $\overleftarrow{\mathbf{h}}_4$
$\overrightarrow{\mathbf{h}}_1$ $\overrightarrow{\mathbf{h}}_2$ $\overrightarrow{\mathbf{h}}_3$ $\overrightarrow{\mathbf{h}}_4$
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$
Ich möchte ein Bier

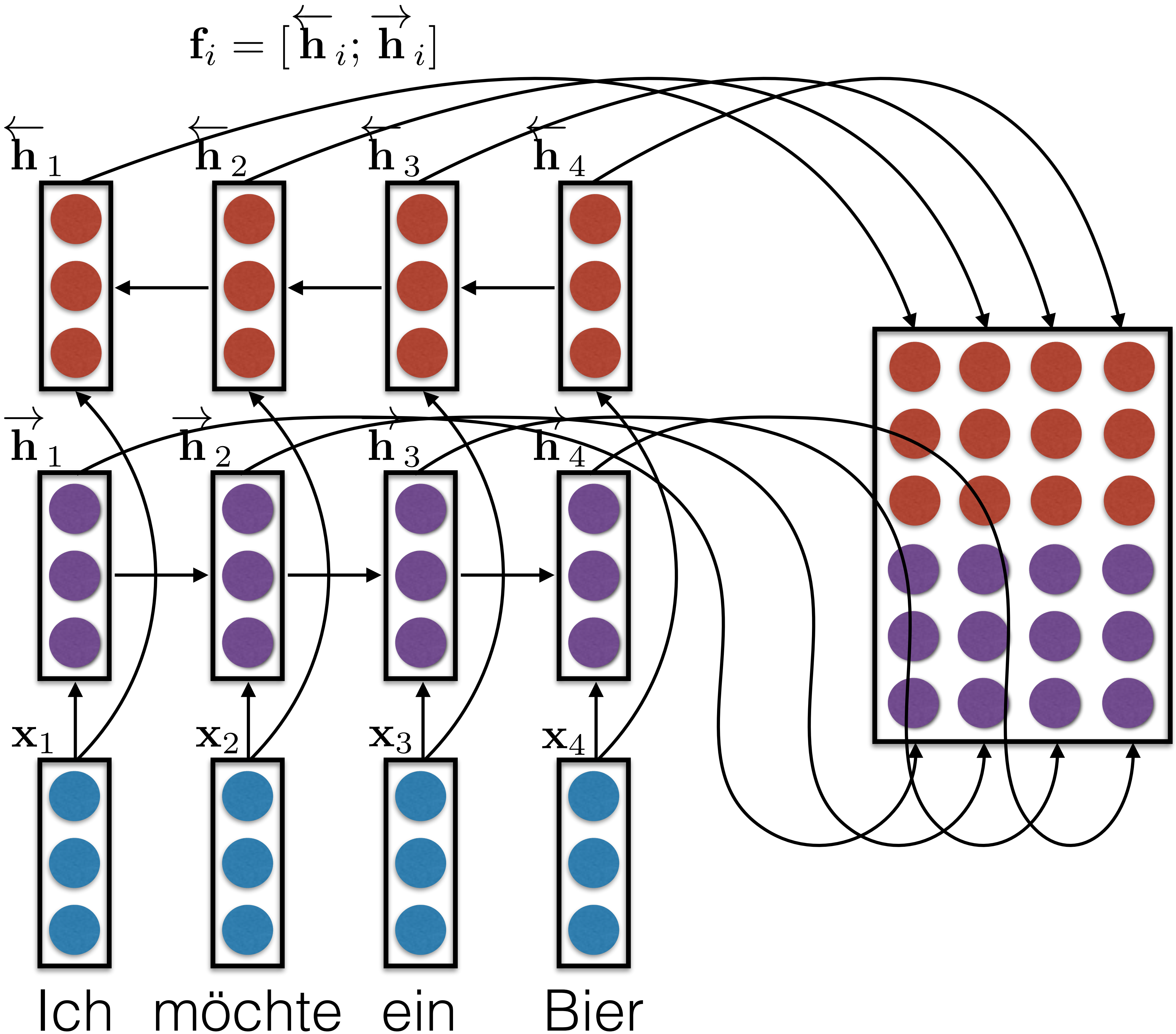
$$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$$




$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$
$\overleftarrow{\mathbf{h}}_1$
$\overleftarrow{\mathbf{h}}_2$
$\overleftarrow{\mathbf{h}}_3$
$\overleftarrow{\mathbf{h}}_4$
$\overrightarrow{\mathbf{h}}_1$
$\overrightarrow{\mathbf{h}}_2$
$\overrightarrow{\mathbf{h}}_3$
$\overrightarrow{\mathbf{h}}_4$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier

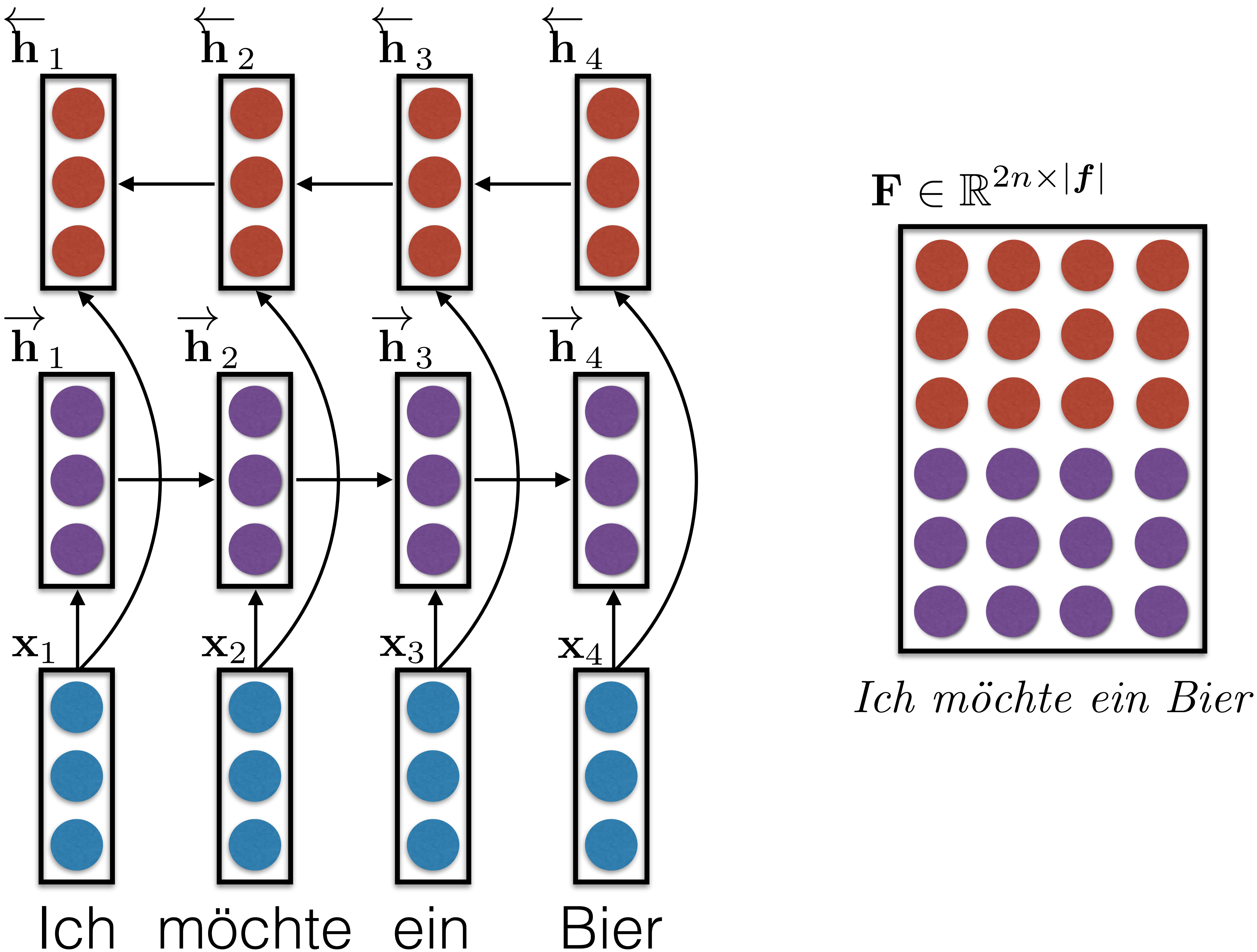


$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$
$\overleftarrow{\mathbf{h}}_1$
$\overleftarrow{\mathbf{h}}_2$
$\overleftarrow{\mathbf{h}}_3$
$\overleftarrow{\mathbf{h}}_4$
$\overrightarrow{\mathbf{h}}_1$
$\overrightarrow{\mathbf{h}}_2$
$\overrightarrow{\mathbf{h}}_3$
$\overrightarrow{\mathbf{h}}_4$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier



$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$
$\overleftarrow{\mathbf{h}}_1$
$\overleftarrow{\mathbf{h}}_2$
$\overleftarrow{\mathbf{h}}_3$
$\overleftarrow{\mathbf{h}}_4$
$\overrightarrow{\mathbf{h}}_1$
$\overrightarrow{\mathbf{h}}_2$
$\overrightarrow{\mathbf{h}}_3$
$\overrightarrow{\mathbf{h}}_4$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier

$$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$$



Where are we in 2016?

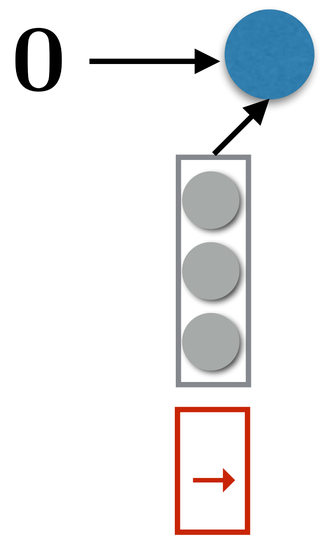
- There are lots of ways to construct **F**
  - Very little (published?) work comparing them
  - There are many more undiscovered things out there
    - convolutions are particularly interesting and under-explored
    - syntactic information could help
    - My intuition is simpler/faster models will work well for the matrix encoding part—context dependencies are limited in language.
    - try something with phrase types instead of word types?

Generation from Matrices

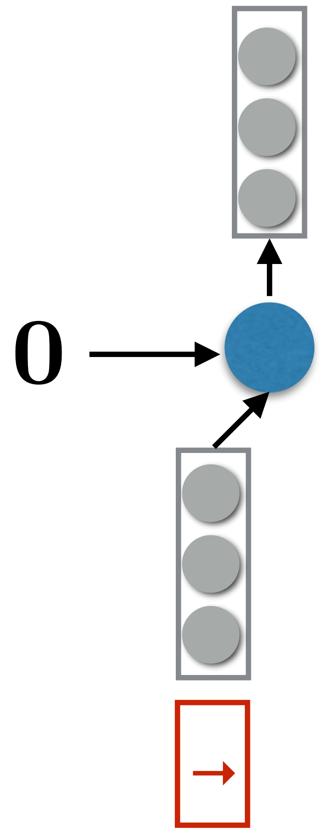
- We have a matrix **F** representing the input, now we need to generate from it
- Bahdanau et al. (2015) were the first to propose using ***attention*** for translating from matrix-encoded sentences
- High-level idea
  - Generate the output sentence word by word using an RNN
  - At each output position t, the RNN receives **two** inputs (in addition to any recurrent inputs)
    - a fixed-size vector embedding of the previously generated output symbol e_{t-1}
    - a fixed-size vector encoding a "view" of the input matrix
  - How do we get a fixed-size vector from a matrix that changes over time?
    - Bahdanau et al: do a weighted sum of the columns of **F** (i.e., words) based on how important they are *at the current time step*. (i.e., just a matrix-vector product $\mathbf{F}\mathbf{a}_t$)
    - The weighting of the input columns at each time-step ($\mathbf{a}_t$) is called ***attention***

0

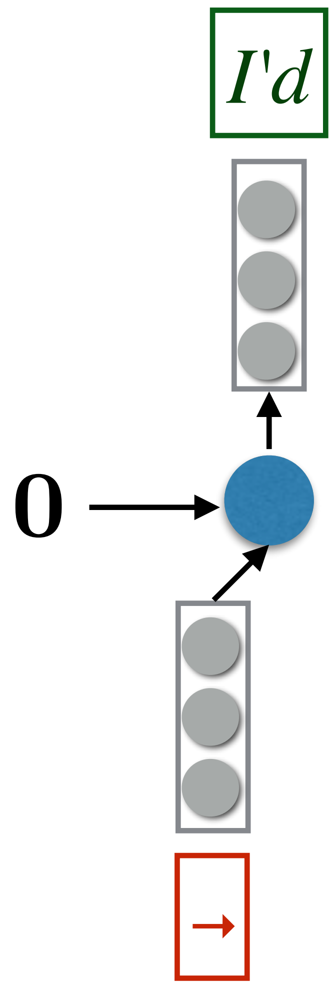
Recall RNNs…



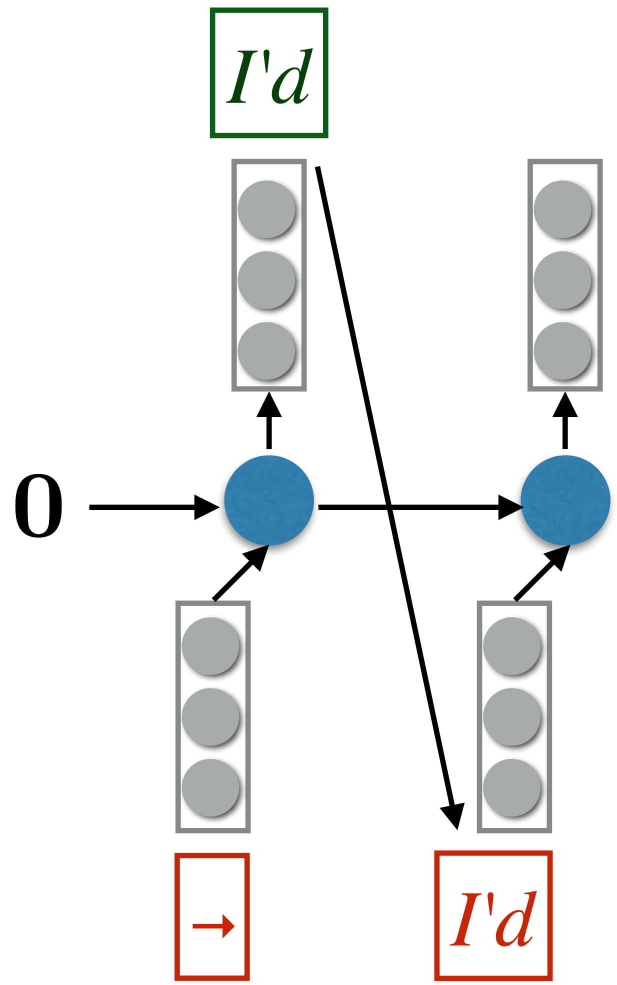
Recall RNNs…



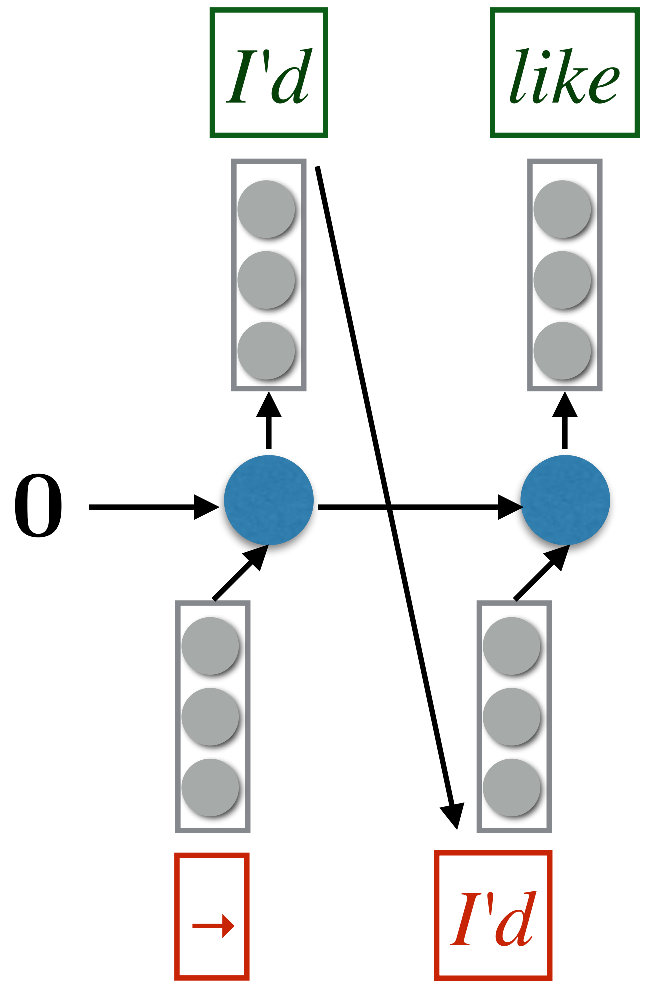
Recall RNNs…



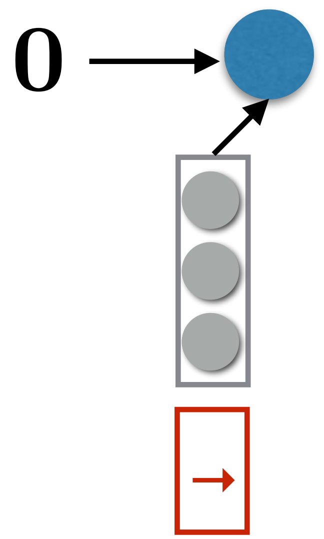
Recall RNNs…

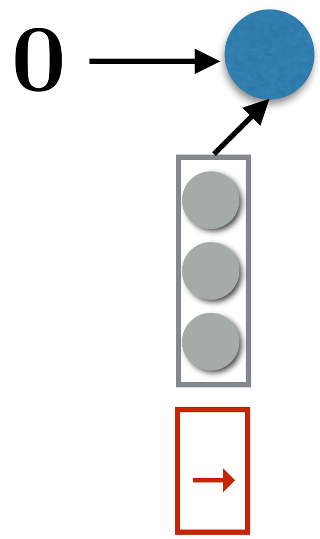


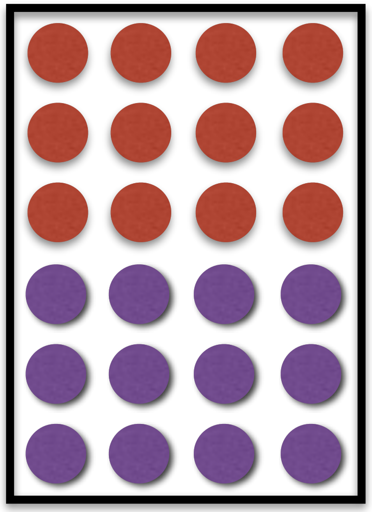
Recall RNNs…



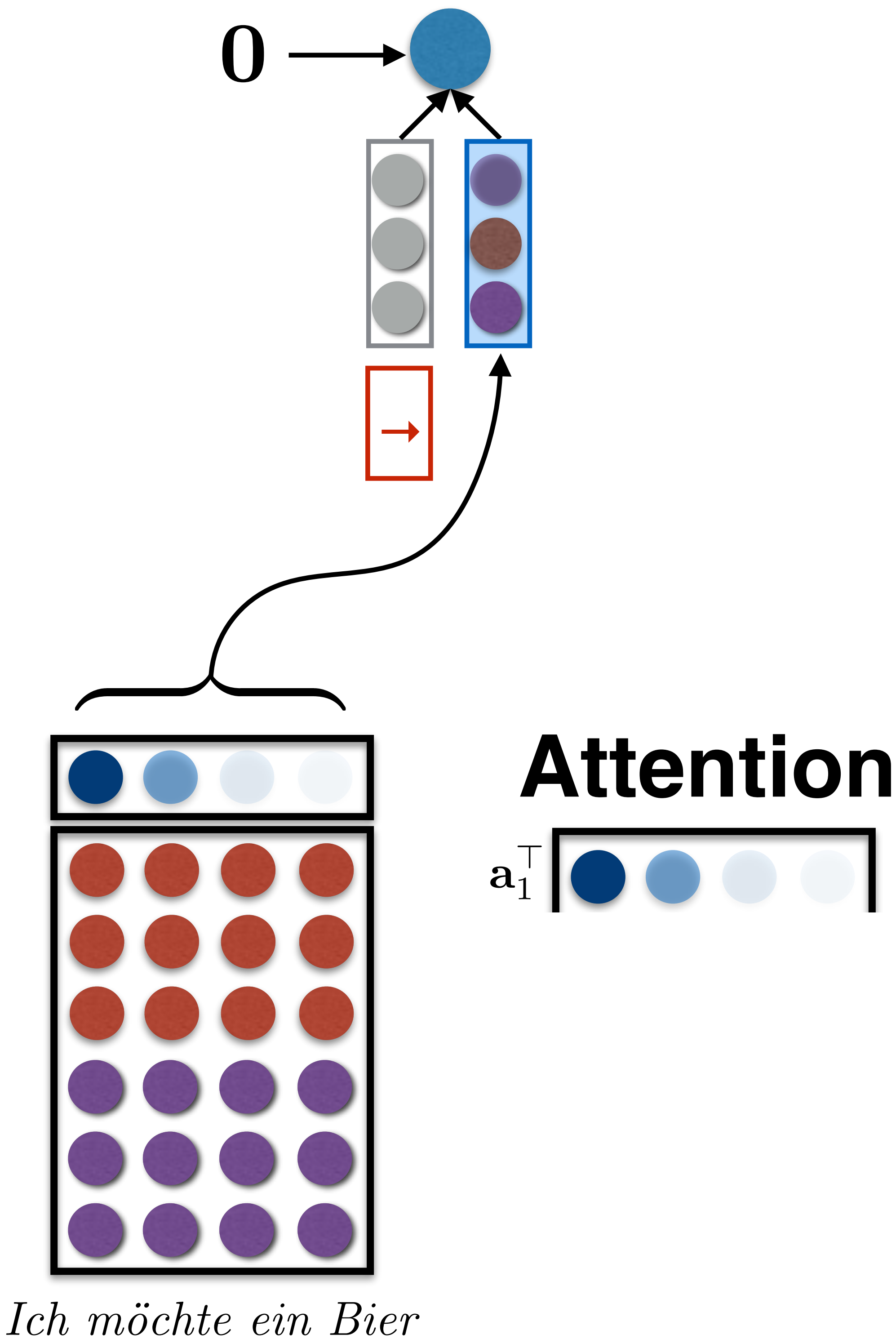
Recall RNNs…


0

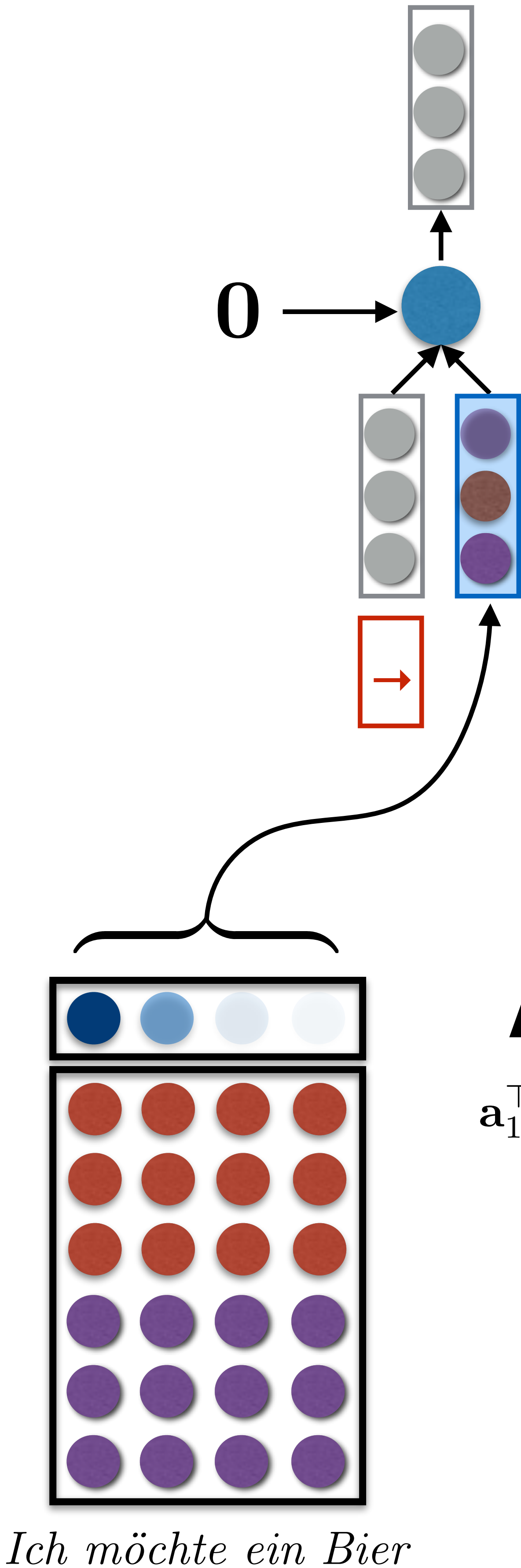




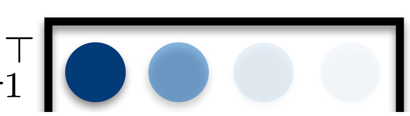
Ich möchte ein Bier

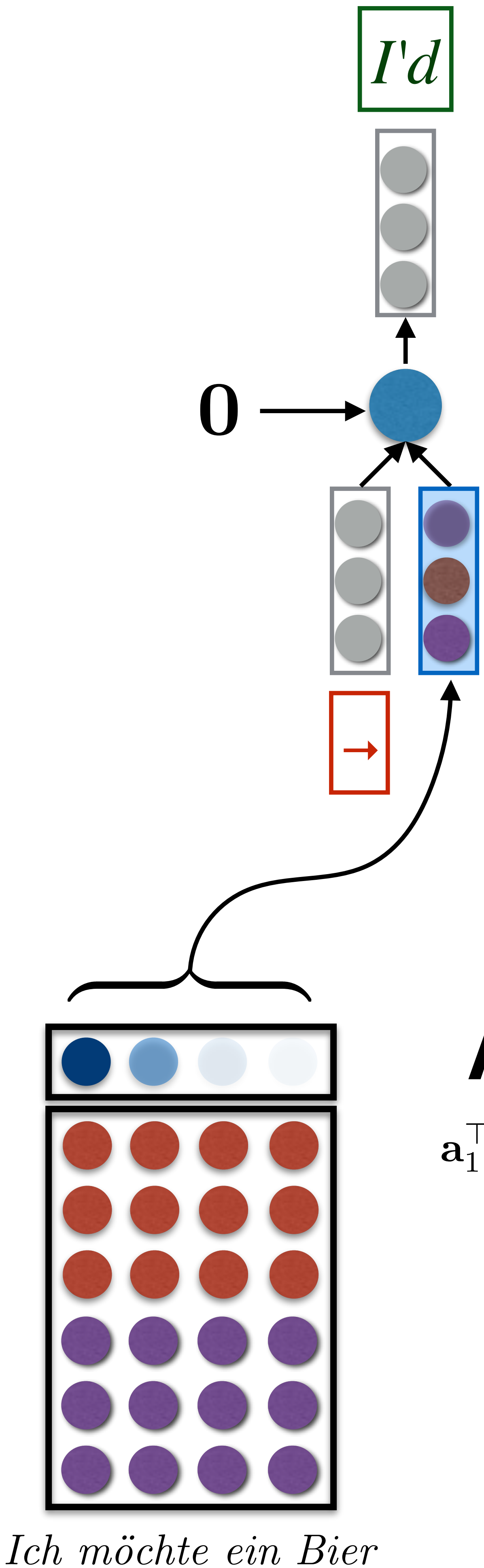


Attention history:



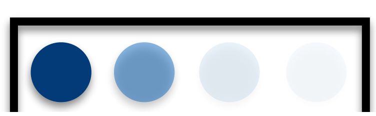
Attention history:

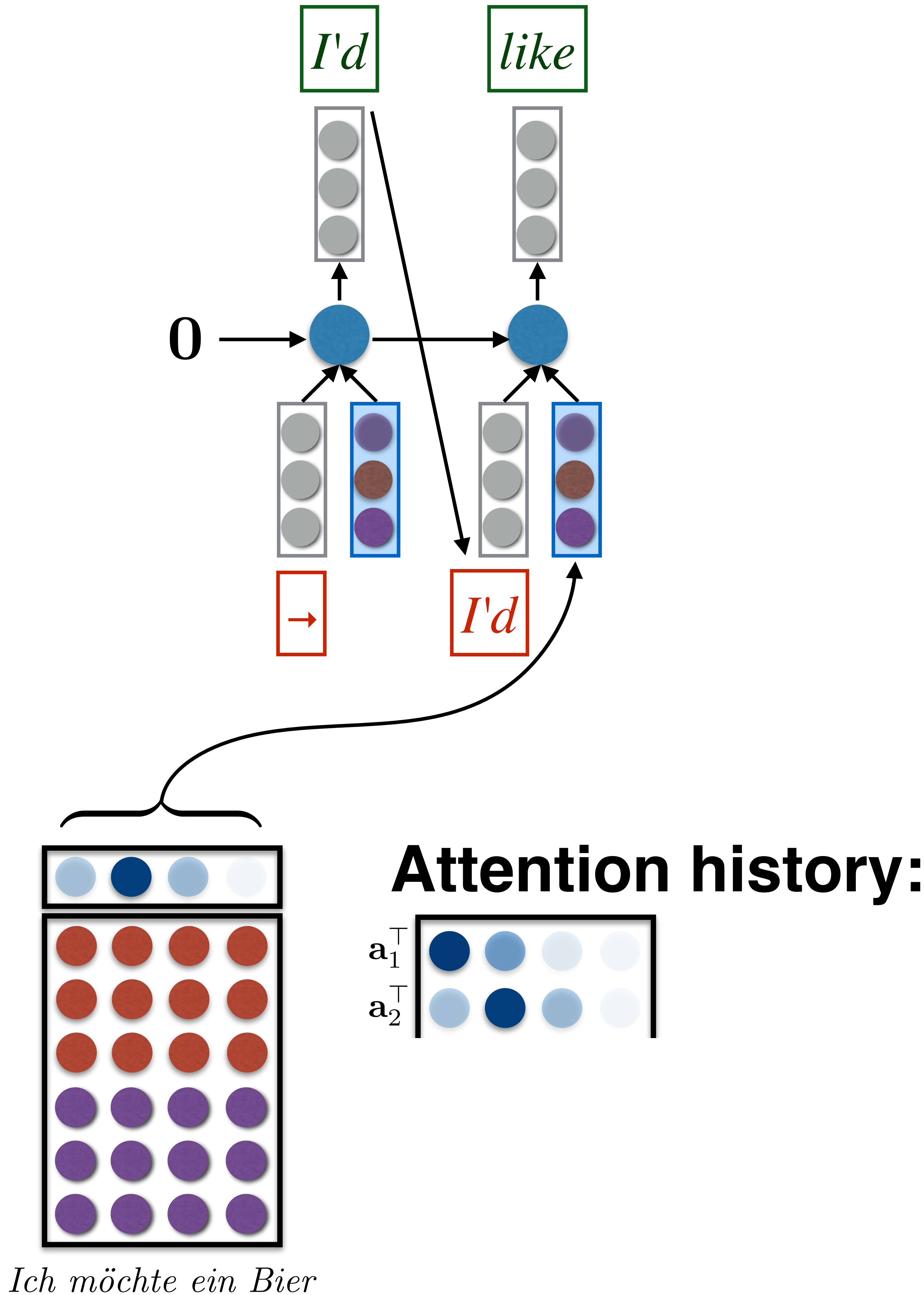
$\mathbf{a}_1^\top$ 

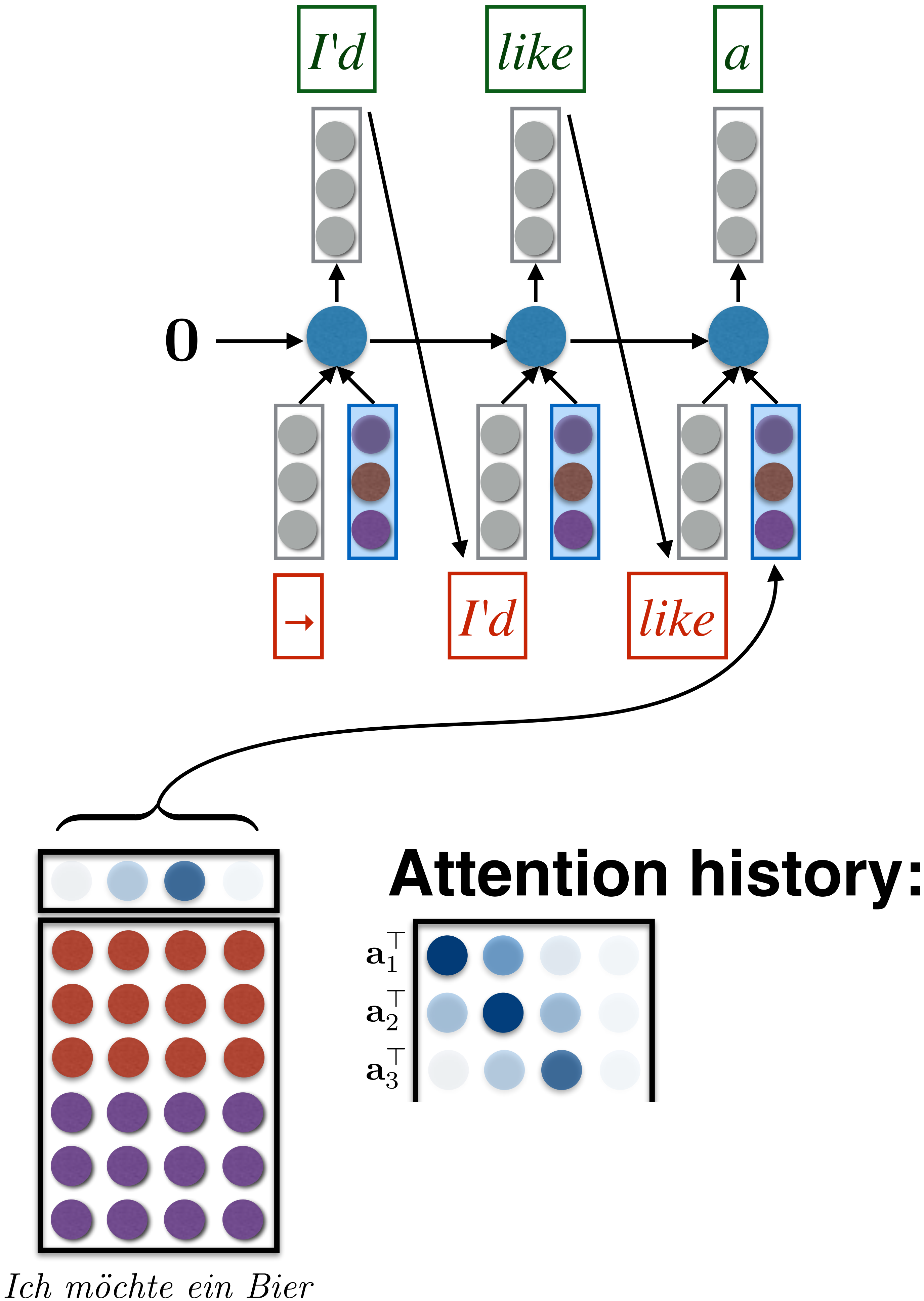


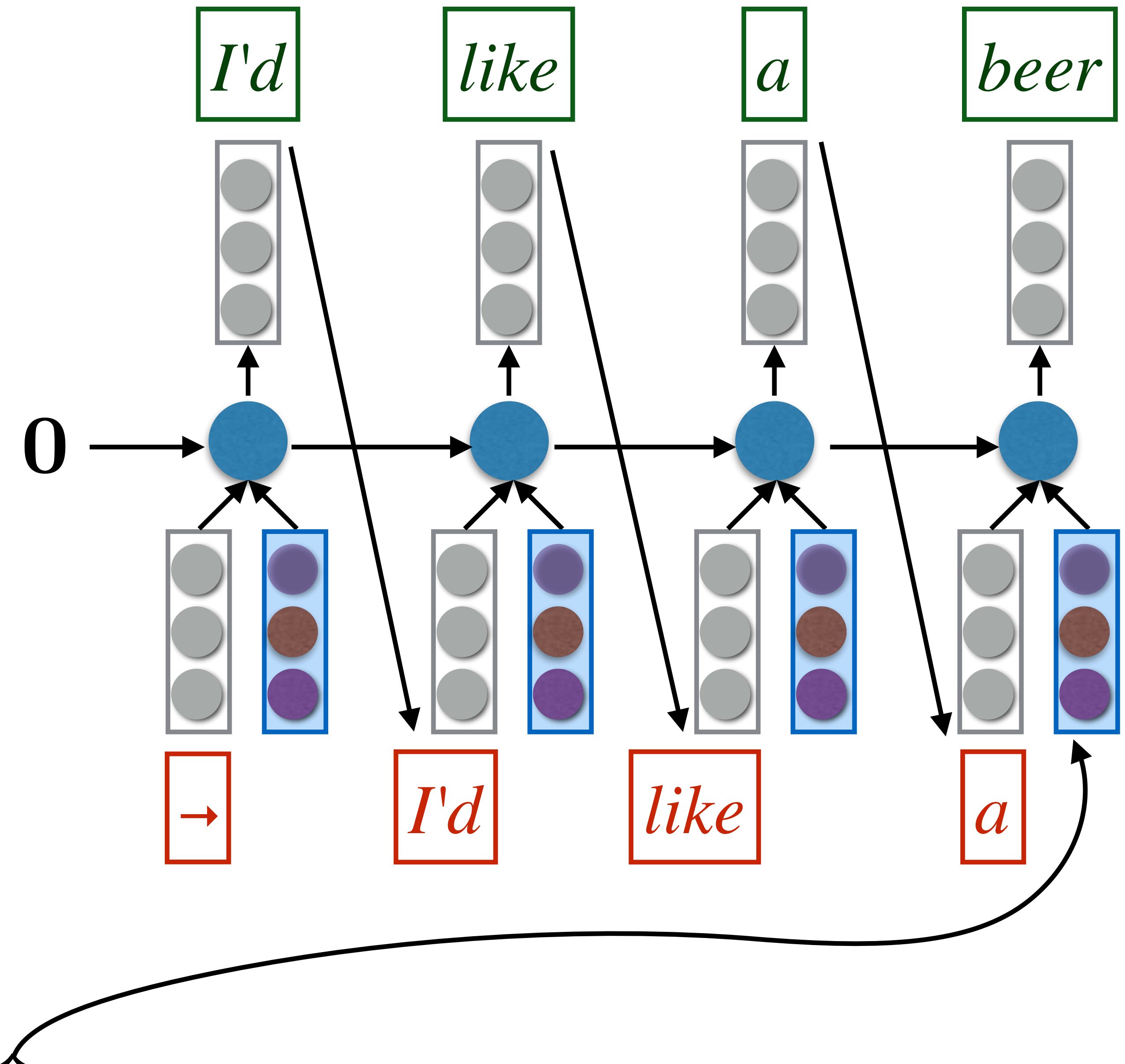
Attention history:

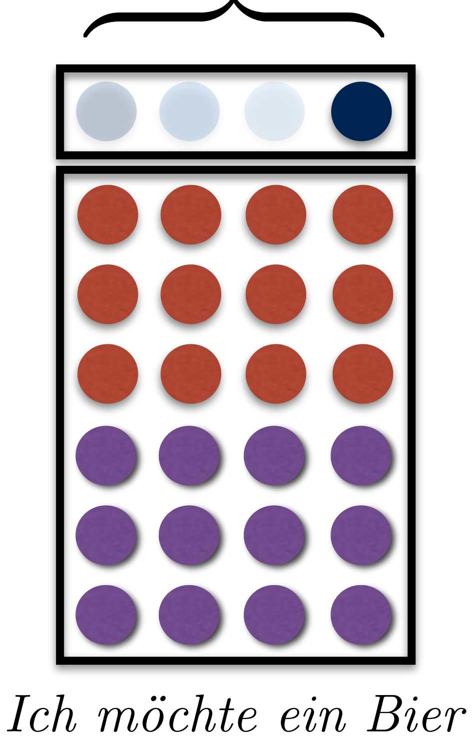
$\mathbf{a}_1^\top$



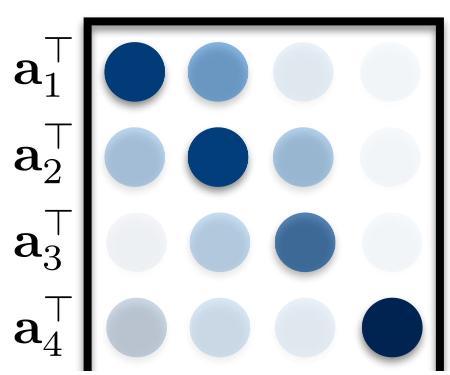

I'd
like
0
→
I'd
Attention history:
$\mathbf{a}_1^\top$
$\mathbf{a}_2^\top$
Ich möchte ein Bier

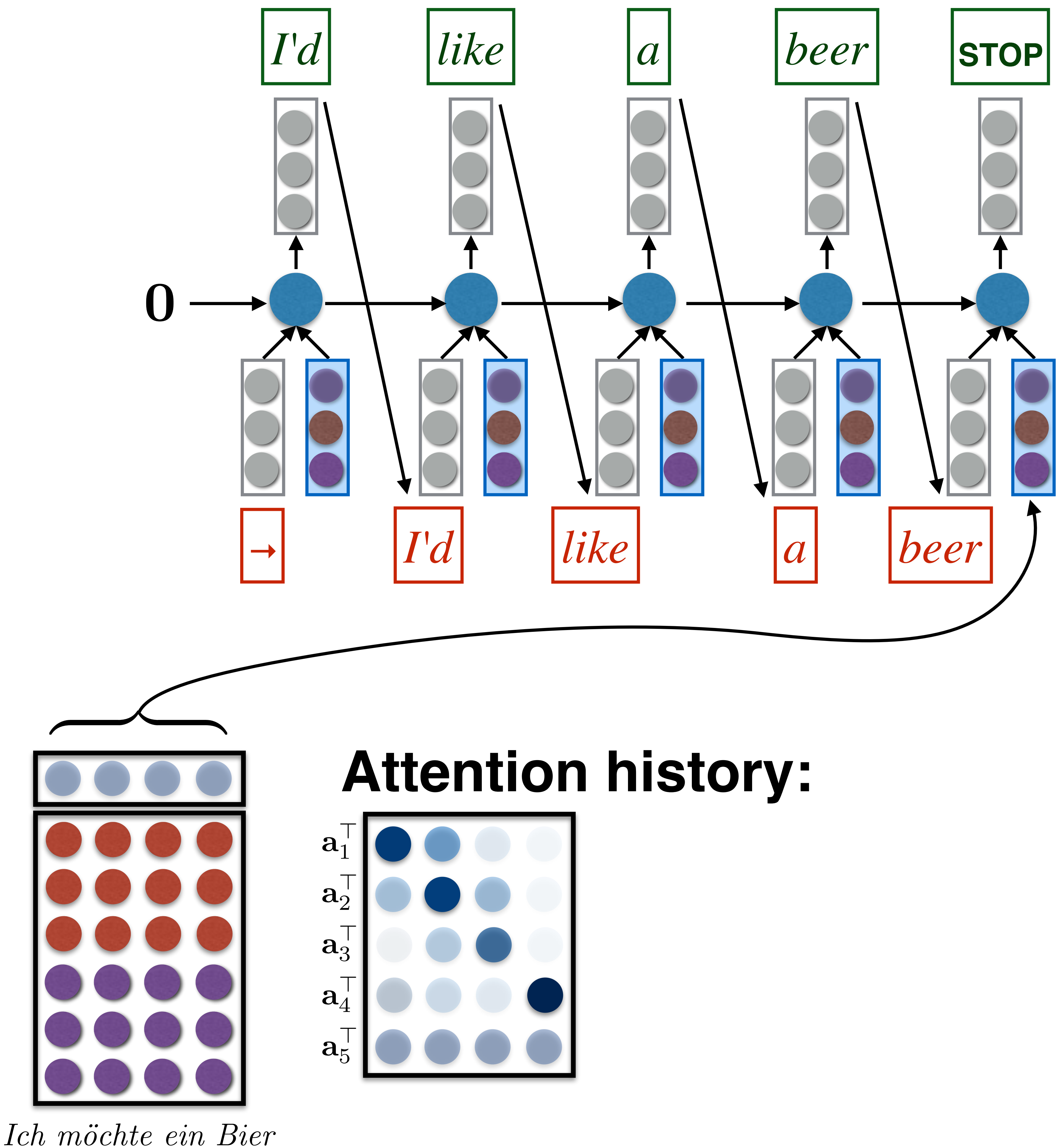

I'd
like
a
0
→
I'd
like
Attention history:
$\mathbf{a}_1^\top$
$\mathbf{a}_2^\top$
$\mathbf{a}_3^\top$
Ich möchte ein Bier





Attention history:





I'd
like
a
beer
STOP
0
→
I'd
like
a
beer
Attention history:
$\mathbf{a}_1^\top$
$\mathbf{a}_2^\top$
$\mathbf{a}_3^\top$
$\mathbf{a}_4^\top$
$\mathbf{a}_5^\top$
Ich möchte ein Bier

Attention

- How do we know what to attend to at each time-step?
- That is, how do we compute $\mathbf{a}_t$?

Computing Attention

- At each time step (one time step = one output word), we want to be able to "attend" to different words in the source sentence
  - We need a weight for every word: this is an $|\boldsymbol{f}|$-length vector $\mathbf{a}_t$
  - Here is a simplified version of Bahdanau et al.'s solution
    - Use an RNN to predict model output, call the hidden states $\mathbf{s}_t$
      ($\mathbf{s}_t$ has a fixed dimensionality, call it m)

Computing Attention

- At each time step (one time step = one output word), we want to be able to "attend" to different words in the source sentence
  - We need a weight for every word: this is an |***f***|-length vector $\mathbf{a}_t$
  - Here is a simplified version of Bahdanau et al.'s solution
    - Use an RNN to predict model output, call the hidden states $\mathbf{s}_t$
      ($\mathbf{s}_t$ has a fixed dimensionality, call it *m*)
    - At time *t* compute the ***expected input embedding*** $\mathbf{r}_t = \mathbf{V}\mathbf{s}_{t-1}$
      ($\mathbf{V}$ is a learned parameter)

Computing Attention

- At each time step (one time step = one output word), we want to be able to "attend" to different words in the source sentence
  - We need a weight for every word: this is an $|\boldsymbol{f}|$-length vector $\mathbf{a}_t$
  - Here is a simplified version of Bahdanau et al.'s solution
    - Use an RNN to predict model output, call the hidden states $\mathbf{s}_t$
      ($\mathbf{s}_t$ has a fixed dimensionality, call it m)
    - At time t compute the ***expected input embedding*** $\mathbf{r}_t = \mathbf{V}\mathbf{s}_{t-1}$
      ($\mathbf{V}$ is a learned parameter)
    - Take the dot product with every column in the source matrix to compute the ***attention energy***. $\mathbf{u}_t = \mathbf{F}^\top \mathbf{r}_t$ (called $\mathbf{e}_t$ in the paper)
      (Since $\mathbf{F}$ has $|\boldsymbol{f}|$ columns, $\mathbf{u}_t$ has $|\boldsymbol{f}|$ rows)

Computing Attention

- At each time step (one time step = one output word), we want to be able to "attend" to different words in the source sentence
  - We need a weight for every word: this is an |***f***|-length vector $\mathbf{a}_t$
  - Here is a simplified version of Bahdanau et al.'s solution
    - Use an RNN to predict model output, call the hidden states $\mathbf{s}_t$
      ($\mathbf{s}_t$ has a fixed dimensionality, call it *m*)
    - At time *t* compute the ***expected input embedding*** $\mathbf{r}_t = \mathbf{V}\mathbf{s}_{t-1}$
      ($\mathbf{V}$ is a learned parameter)
    - Take the dot product with every column in the source matrix to compute the ***attention energy***. $\mathbf{u}_t = \mathbf{F}^\top \mathbf{r}_t$ (called $\mathbf{e}_t$ in the paper)
      (Since **F** has |***f***| columns, $\mathbf{u}_t$ has |***f***| rows)
    - Exponentiate and normalize to 1: $\mathbf{a}_t = \mathrm{softmax}(\mathbf{u}_t)$
      (called $\boldsymbol{\alpha}_t$ in the paper)

Computing Attention

- At each time step (one time step = one output word), we want to be able to "attend" to different words in the source sentence
  - We need a weight for every word: this is an |***f***|-length vector $\mathbf{a}_t$
  - Here is a simplified version of Bahdanau et al.'s solution
    - Use an RNN to predict model output, call the hidden states $\mathbf{s}_t$
      ($\mathbf{s}_t$ has a fixed dimensionality, call it *m*)
    - At time *t* compute the ***expected input embedding*** $\mathbf{r}_t = \mathbf{V}\mathbf{s}_{t-1}$
      ($\mathbf{V}$ is a learned parameter)
    - Take the dot product with every column in the source matrix to compute the ***attention energy***. $\mathbf{u}_t = \mathbf{F}^\top \mathbf{r}_t$ (called $\mathbf{e}_t$ in the paper)
      (Since **F** has |***f***| columns, $\mathbf{u}_t$ has |***f***| rows)
    - Exponentiate and normalize to 1: $\mathbf{a}_t = \mathrm{softmax}(\mathbf{u}_t)$
      (called $\boldsymbol{\alpha}_t$ in the paper)
    - Finally, the ***input source vector*** for time *t* is $\mathbf{c}_t = \mathbf{F}\mathbf{a}_t$

Nonlinear Attention-Energy Model

- In the actual model, Bahdanau et al. replace the dot product between the columns of **F** and $\mathbf{r}_t$ with an MLP:

$$\mathbf{u}_t = \mathbf{F}^\top \mathbf{r}_t \qquad \text{(simple model)}$$

Nonlinear Attention-Energy Model

- In the actual model, Bahdanau et al. replace the dot product between the columns of **F** and $\mathbf{r}_t$ with an MLP:

$$\cancel{\mathbf{u}_t = \mathbf{F}^\top \mathbf{r}_t \qquad \text{(simple model)}}$$

$$\mathbf{u}_t = \mathbf{v}^\top \tanh(\mathbf{W}\mathbf{F} + \mathbf{r}_t) \qquad \text{(Bahdanau et al)}$$

Nonlinear Attention-Energy Model

- In the actual model, Bahdanau et al. replace the dot product between the columns of **F** and **r**$_t$ with an MLP:

$$\cancel{\mathbf{u}_t = \mathbf{F}^\top \mathbf{r}_t \qquad \text{(simple model)}}$$

$$\mathbf{u}_t = \mathbf{v}^\top \tanh(\mathbf{W}\mathbf{F} + \mathbf{r}_t) \qquad \text{(Bahdanau et al)}$$

- Here, **W** and **v** are learned parameters of appropriate dimension and + "broadcasts" over the |***f***| columns in **WF**
- This can learn more complex interactions
  - It is unclear if the added complexity is necessary for good performance

Putting it all together

$\mathbf{F} = \mathrm{EncodeAsMatrix}(\boldsymbol{f})$

$e_0 = \langle \mathbf{s} \rangle$

$\mathbf{s}_0 = \mathbf{w}$ (Learned initial state; Bahdanau uses $\mathbf{U}\overleftarrow{\mathbf{h}}_1$)

$t = 0$

$\textbf{while } e_t \neq \langle / \mathbf{s} \rangle :$

$$t = t + 1$$

$$\left.\begin{array}{l} \mathbf{r}_t = \mathbf{V}\mathbf{s}_{t-1} \\ \mathbf{u}_t = \mathbf{v}^\top \tanh(\mathbf{W}\mathbf{F} + \mathbf{r}_t) \\ \mathbf{a}_t = \mathrm{softmax}(\mathbf{u}_t) \end{array}\right\} \text{(Compute attention)}$$

$$\mathbf{c}_t = \mathbf{F}\mathbf{a}_t$$

$$\mathbf{s}_t = \mathrm{RNN}(\mathbf{s}_{t-1}, [\mathbf{e}_{t-1}; \mathbf{c}_t])$$ ($\mathbf{e}_{t-1}$ is a learned embedding of e_t)

$$\mathbf{y}_t = \mathrm{softmax}(\mathbf{P}\mathbf{s}_t + \mathbf{b})$$ ($\mathbf{P}$ and $\mathbf{b}$ are learned parameters)

$$e_t \mid \boldsymbol{e}_{<t} \sim \mathrm{Categorical}(\mathbf{y}_t)$$

Putting it all together

$\mathbf{F} = \mathrm{EncodeAsMatrix}(\boldsymbol{f})$

$e_0 = \langle \mathbf{s} \rangle$

$\mathbf{s}_0 = \mathbf{w}$ (Learned initial state; Bahdanau uses $\mathbf{U}\overleftarrow{\mathbf{h}}_1$)

$t = 0$

$\mathbf{while}\ e_t \neq \langle /\mathbf{s} \rangle :$

$\quad t = t + 1$

doesn't depend on output decisions

$$\left.\begin{aligned} \mathbf{r}_t &= \mathbf{V}\mathbf{s}_{t-1} \\ \mathbf{u}_t &= \mathbf{v}^\top \tanh(\mathbf{WF} + \mathbf{r}_t) \\ \mathbf{a}_t &= \mathrm{softmax}(\mathbf{u}_t) \end{aligned}\right\}$$ (Compute attention)

$\quad \mathbf{c}_t = \mathbf{F}\mathbf{a}_t$

$\quad \mathbf{s}_t = \mathrm{RNN}(\mathbf{s}_{t-1}, [\mathbf{e}_{t-1}; \mathbf{c}_t])$ ($\mathbf{e}_{t-1}$ is a learned embedding of e_t)

$\quad \mathbf{y}_t = \mathrm{softmax}(\mathbf{P}\mathbf{s}_t + \mathbf{b})$ ($\mathbf{P}$ and $\mathbf{b}$ are learned parameters)

$\quad e_t \mid \boldsymbol{e}_{<t} \sim \mathrm{Categorical}(\mathbf{y}_t)$

Putting it all together

$\mathbf{F} = \mathrm{EncodeAsMatrix}(\boldsymbol{f})$ (Part 1 of lecture)

$e_0 = \langle \mathbf{s} \rangle$

$\mathbf{s}_0 = \mathbf{w}$ (Learned initial state; Bahdanau uses $\mathbf{U}\overleftarrow{\mathbf{h}}_1$)

$t = 0$

$\mathbf{X} = \mathbf{WF}$

while $e_t \neq \langle /\mathbf{s} \rangle$:

$$t = t + 1$$

$$\left.\begin{aligned}
\mathbf{r}_t &= \mathbf{V}\mathbf{s}_{t-1} \\
\mathbf{u}_t &= \mathbf{v}^\top \tanh(\overset{\mathbf{X}}{\cancel{\mathbf{WF}}} + \mathbf{r}_t) \\
\mathbf{a}_t &= \mathrm{softmax}(\mathbf{u}_t)
\end{aligned}\right\} \text{(Compute attention)}$$

$$\mathbf{c}_t = \mathbf{F}\mathbf{a}_t$$

$$\mathbf{s}_t = \mathrm{RNN}(\mathbf{s}_{t-1}, [\mathbf{e}_{t-1}; \mathbf{c}_t]) \quad (\mathbf{e}_{t-1} \text{ is a learned embedding of } e_t)$$

$$\mathbf{y}_t = \mathrm{softmax}(\mathbf{P}\mathbf{s}_t + \mathbf{b}) \quad (\mathbf{P} \text{ and } \mathbf{b} \text{ are learned parameters})$$

$$e_t \mid \boldsymbol{e}_{<t} \sim \mathrm{Categorical}(\mathbf{y}_t)$$

Putting it all together

$\mathbf{F} = \text{EncodeAsMatrix}(\boldsymbol{f})$ (Part 1 of lecture)

$e_0 = \langle \mathbf{s} \rangle$

$\mathbf{s}_0 = \mathbf{w}$ (Learned initial state; Bahdanau uses $\mathbf{U}\overleftarrow{\mathbf{h}}_1$)

$t = 0$

$\mathbf{X} = \mathbf{WF}$

$\textbf{while } e_t \neq \langle /\mathbf{s} \rangle :$

$$t = t + 1$$

$$\left.\begin{aligned} \mathbf{r}_t &= \mathbf{V}\mathbf{s}_{t-1} \\ \mathbf{u}_t &= \mathbf{v}^\top \tanh(\mathbf{X} + \mathbf{r}_t) \\ \mathbf{a}_t &= \text{softmax}(\mathbf{u}_t) \end{aligned}\right\} \text{(Compute attention)}$$

$$\mathbf{c}_t = \mathbf{F}\mathbf{a}_t$$

$\mathbf{s}_t = \text{RNN}(\mathbf{s}_{t-1}, [\mathbf{e}_{t-1}; \mathbf{c}_t])$ ($\mathbf{e}_{t-1}$ is a learned embedding of e_t)

$\mathbf{y}_t = \text{softmax}(\mathbf{P}\mathbf{s}_t + \mathbf{b})$ ($\mathbf{P}$ and $\mathbf{b}$ are learned parameters)

$$e_t \mid \boldsymbol{e}_{<t} \sim \text{Categorical}(\mathbf{y}_t)$$

Summary

- Attention is closely related to “pooling” operations in convnets (and other architectures)
- Bahdanau’s attention model seems to only cares about “content”
  - No obvious bias in favor of diagonals, short jumps, fertility, etc.
  - Some work has begun to add other “structural” biases (Luong et al., 2015; Cohn et al., 2016), but there are lots more opportunities
- Attention is similar to **alignment**, but there are important differences
  - alignment makes stochastic but hard decisions. Even if the alignment probability distribution is “flat”, the model picks one word or phrase at a time
  - attention is “soft” (you add together all the words). Big difference between “flat” and “peaked” attention weights

Attention and Translation

- Cho's question: does a translator read and memorize the input sentence/document and then generate the output?
    - Compressing the entire input sentence into a vector basically says "memorize the sentence"
    - Common sense experience says translators refer back and forth to the input. (also backed up by eye-tracking studies)
- Should humans be a model for machines?

Questions?