Modeling Sequential Data with Recurrent Networks

Chris Dyer
DeepMind
Carnegie Mellon University

LxMLS 2017 July 27, 2017

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

- Conditional sequence models
- Applications
    - Translation
    - Image caption generation
- Better learning with attention
- Applications
    - Translation
    - Image caption generation
- Implementation tricks (time permitting)

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
    - Notes in a melody
    - Stock market returns
    - …
- **How do we represent an arbitrarily long history?**

Recurrent Neural Networks

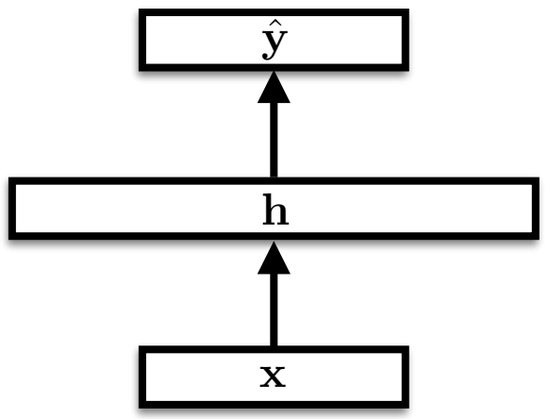
- Lots of interesting data is sequential in nature
    - Words in sentences
    - DNA
    - Notes in a melody
    - Stock market returns
    - …
- **How do we represent an arbitrarily long history?**
    - we will train neural networks to build a representation of these arbitrarily big sequences

Recurrent Neural Networks

Feed-forward NN

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

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

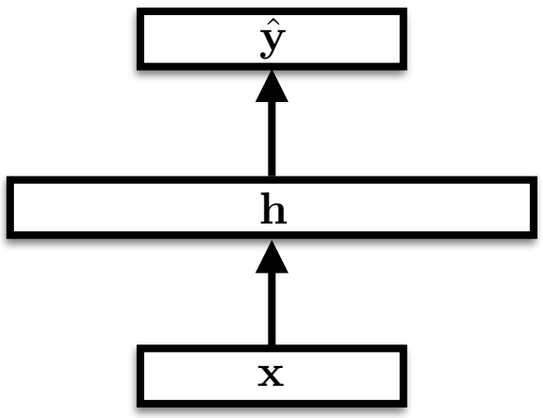


Recurrent Neural Networks

Feed-forward NN

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

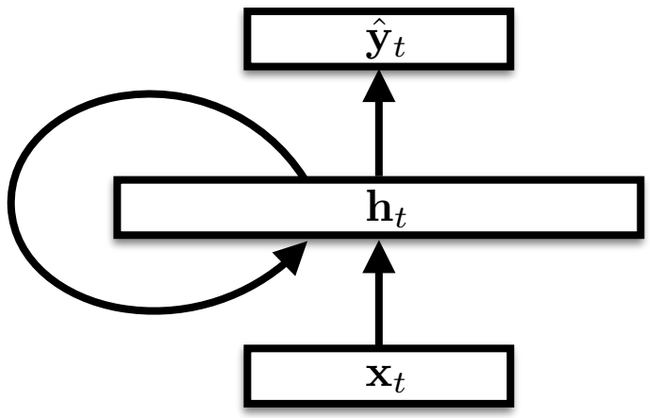
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$



Recurrent NN

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

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

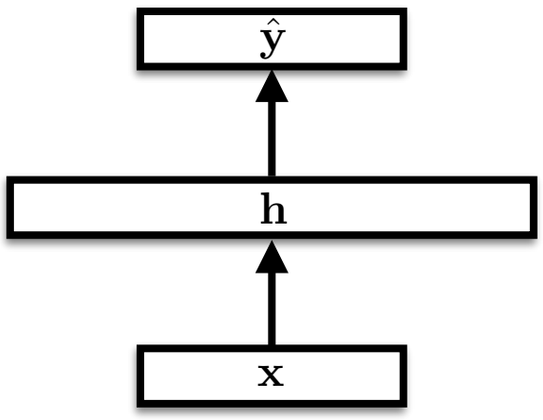


Recurrent Neural Networks

Feed-forward NN

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

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

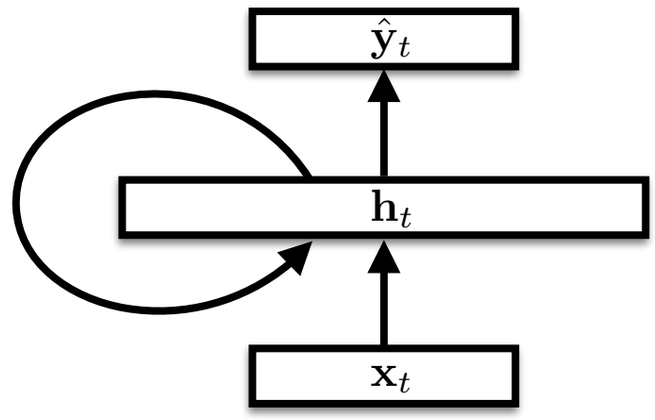


Recurrent NN

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

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

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

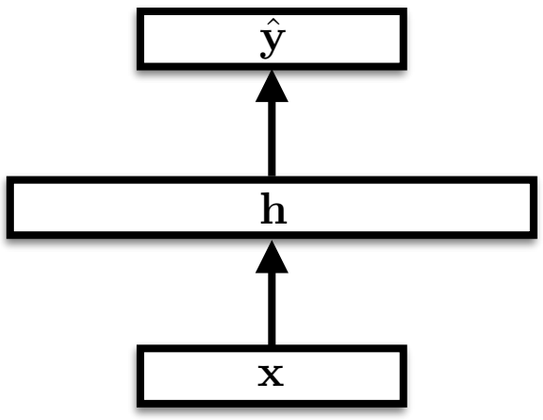


Recurrent Neural Networks

Feed-forward NN

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

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

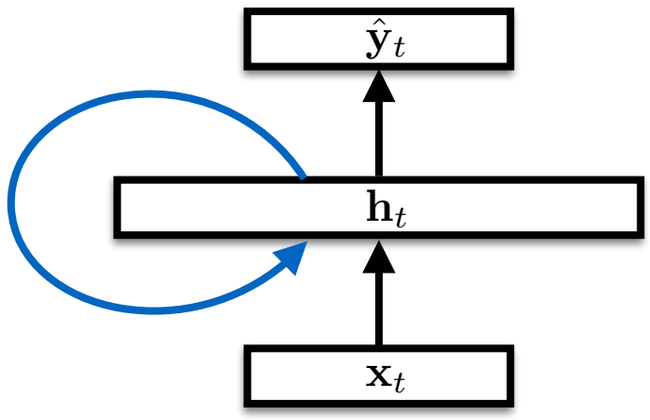


Recurrent NN

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

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

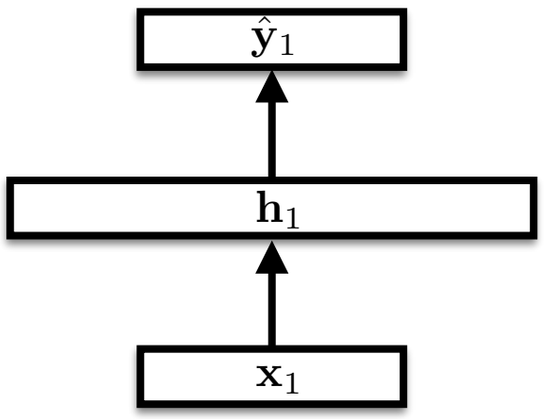
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

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

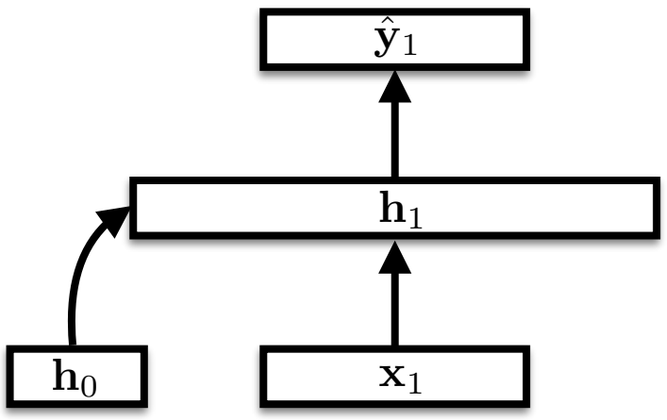
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

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

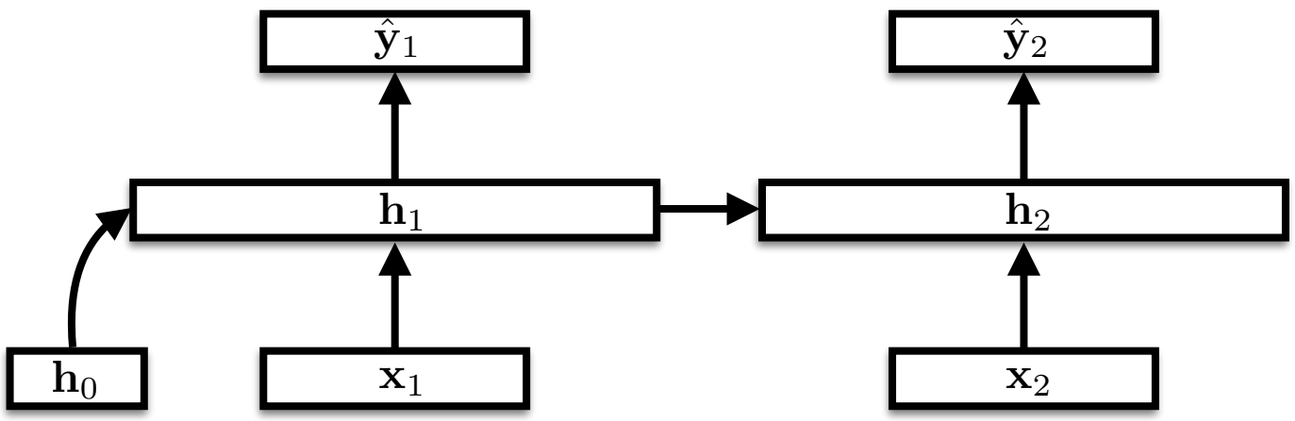
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

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

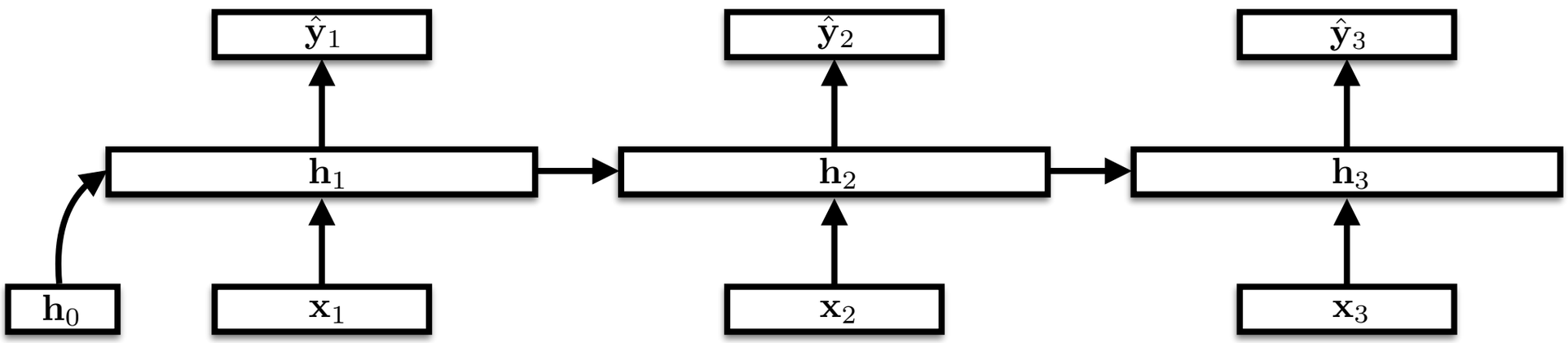
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

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

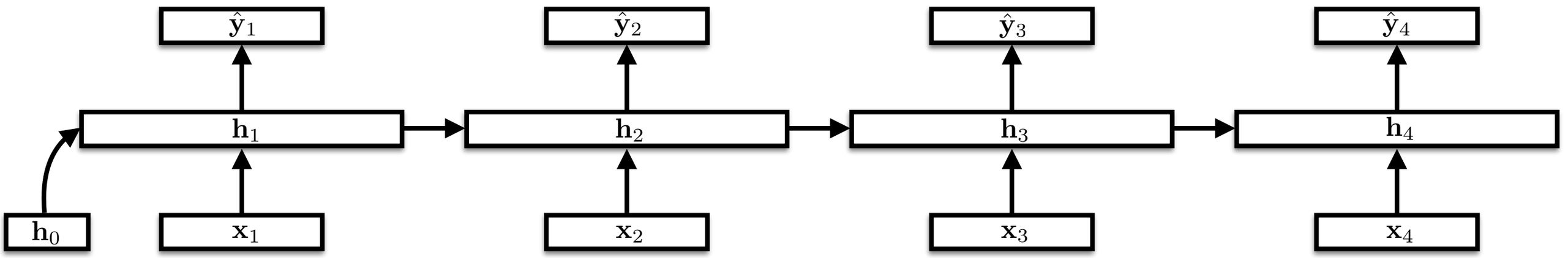
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

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

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

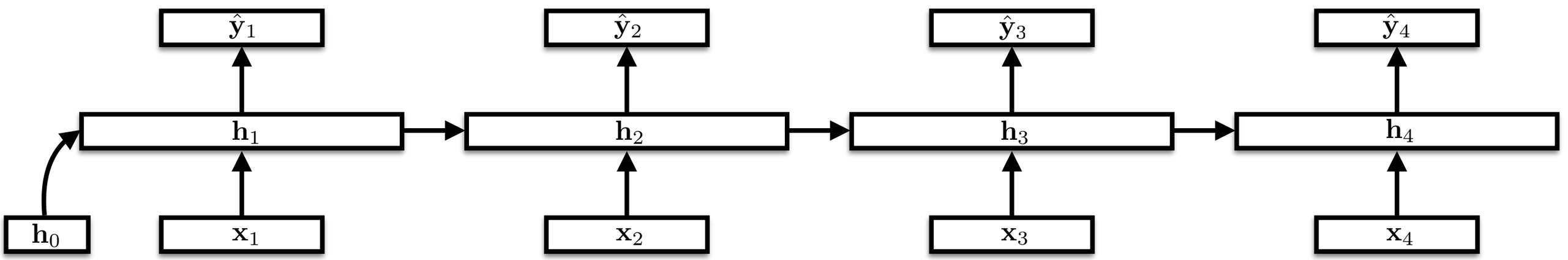


Recurrent Neural Networks

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

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

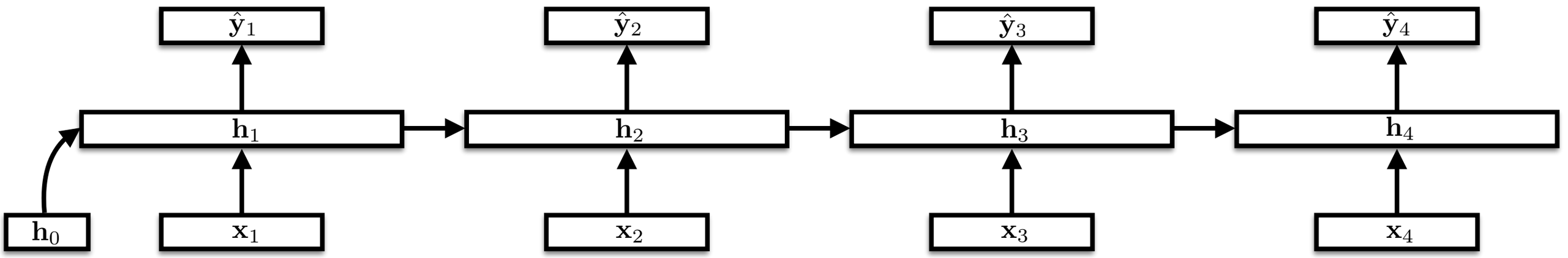
How do we train the RNN's parameters?



Recurrent Neural Networks

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

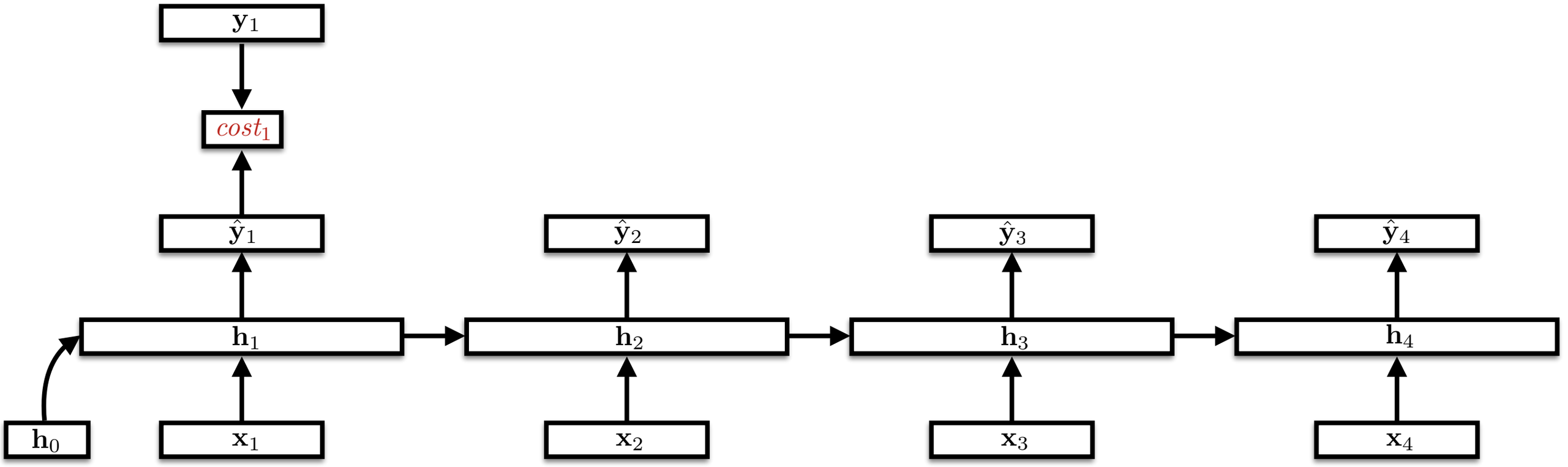
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

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

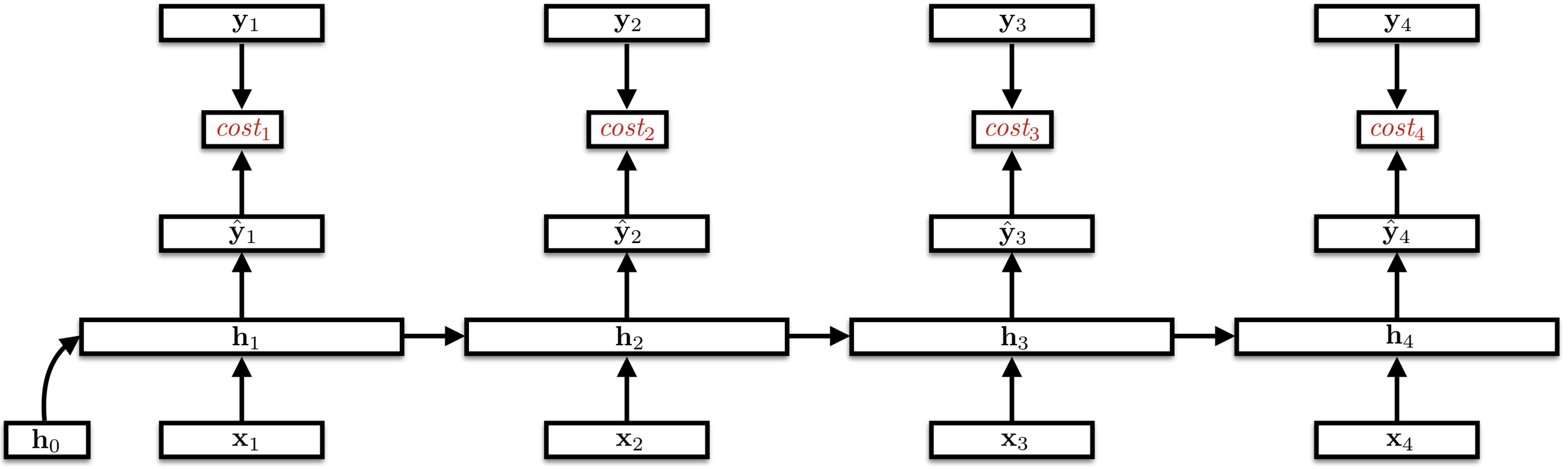
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

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

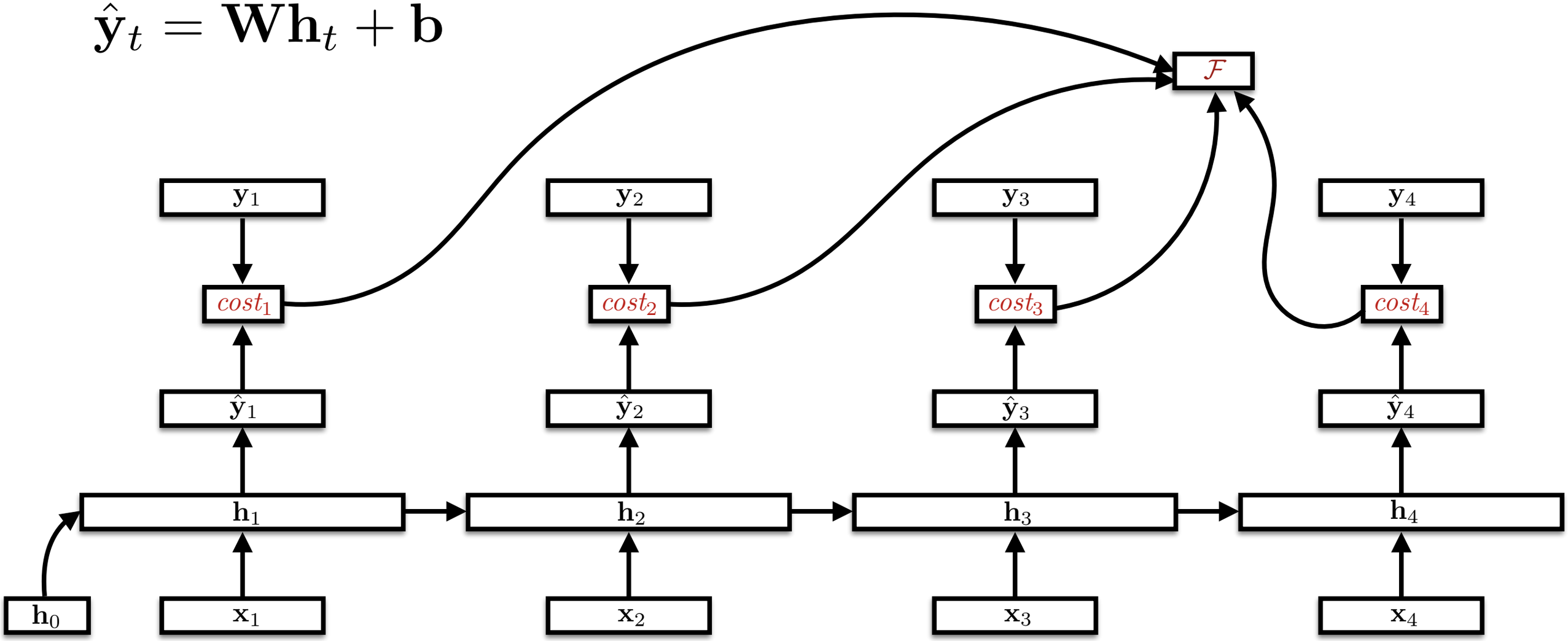
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



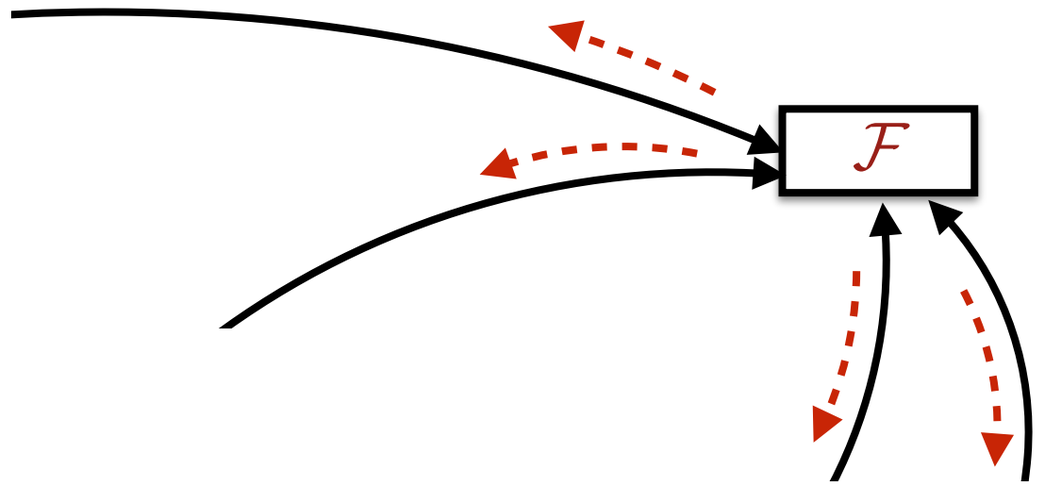
Recurrent Neural Networks

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

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



Recurrent Neural Networks

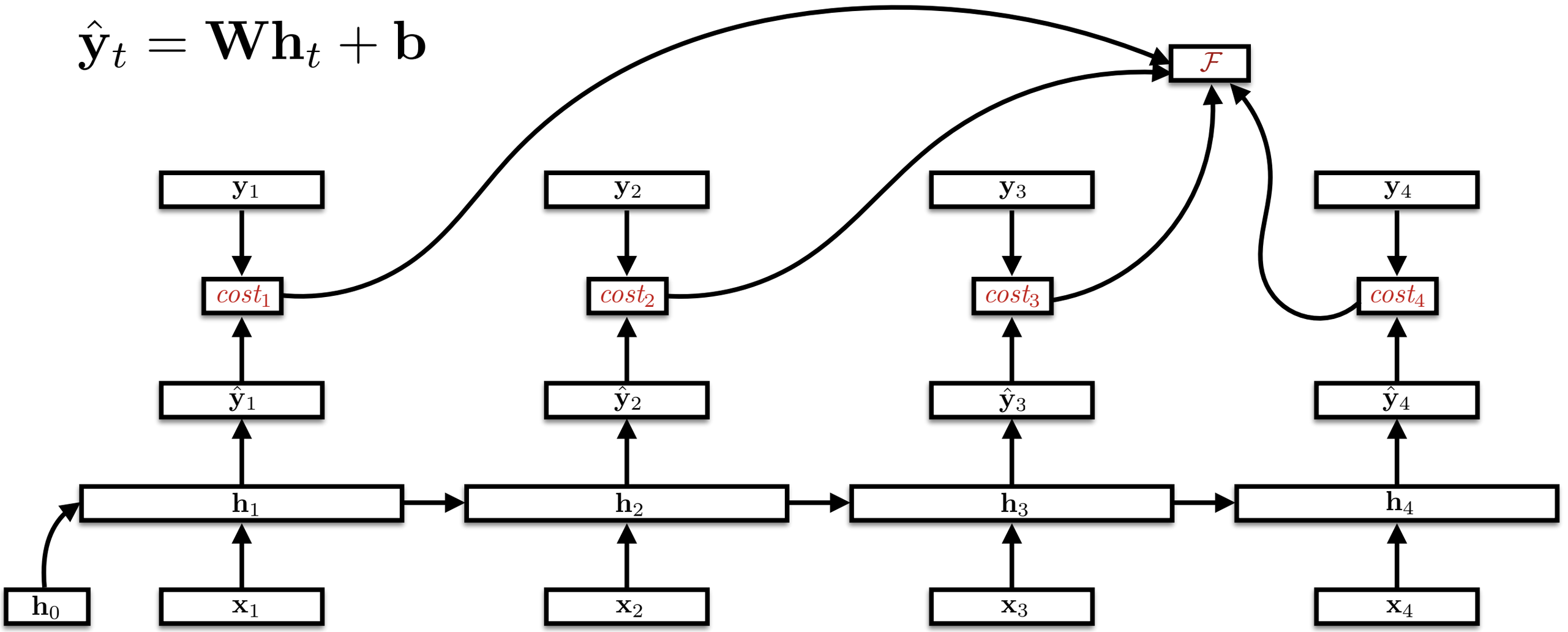


- The unrolled graph is a well-formed (DAG) computation graph—we can run backprop
  - Parameters are tied across time, derivatives are aggregated across all time steps
  - This is historically called “backpropagation through time” (BPTT)

Parameter Tying

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

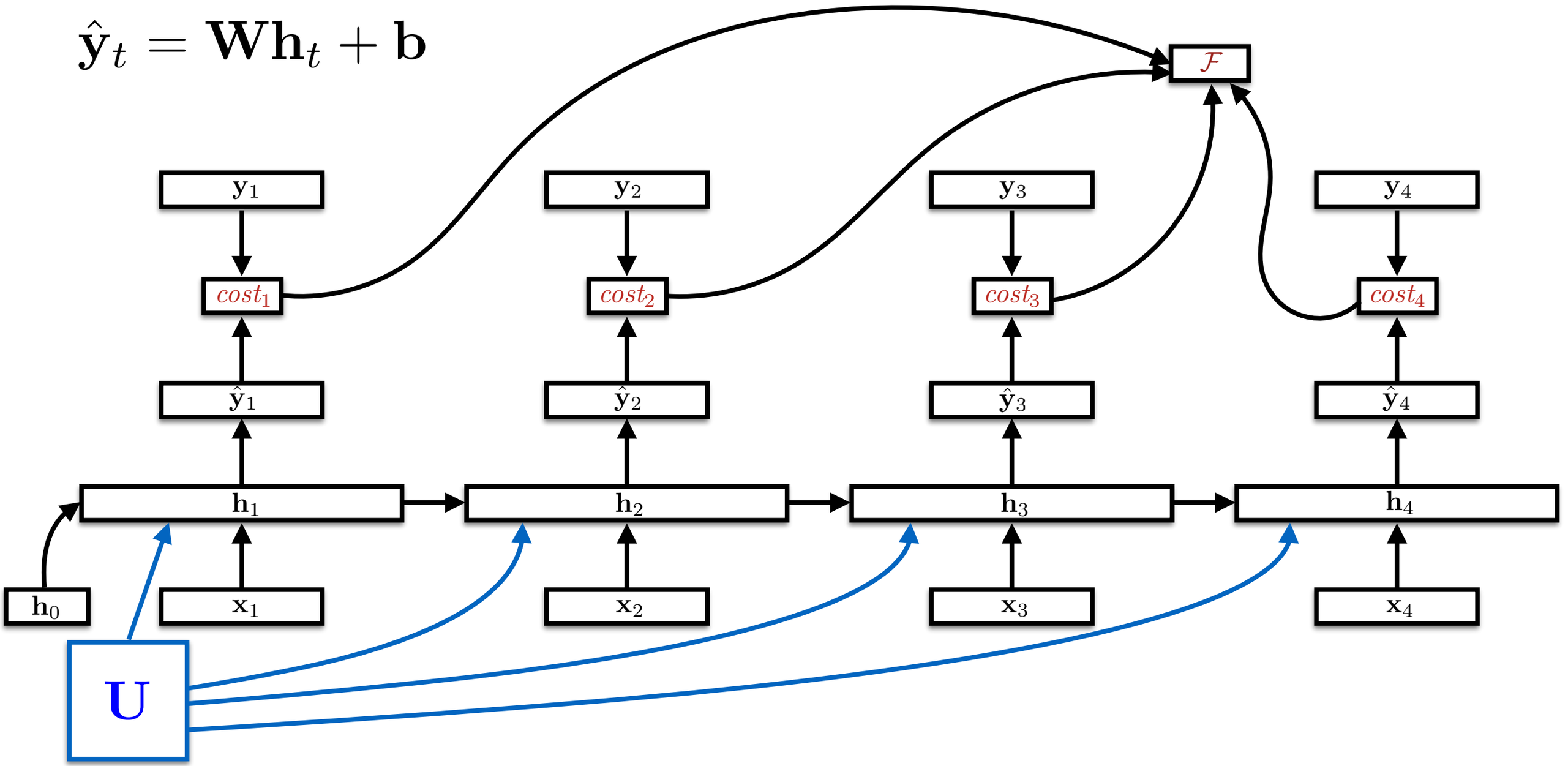
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



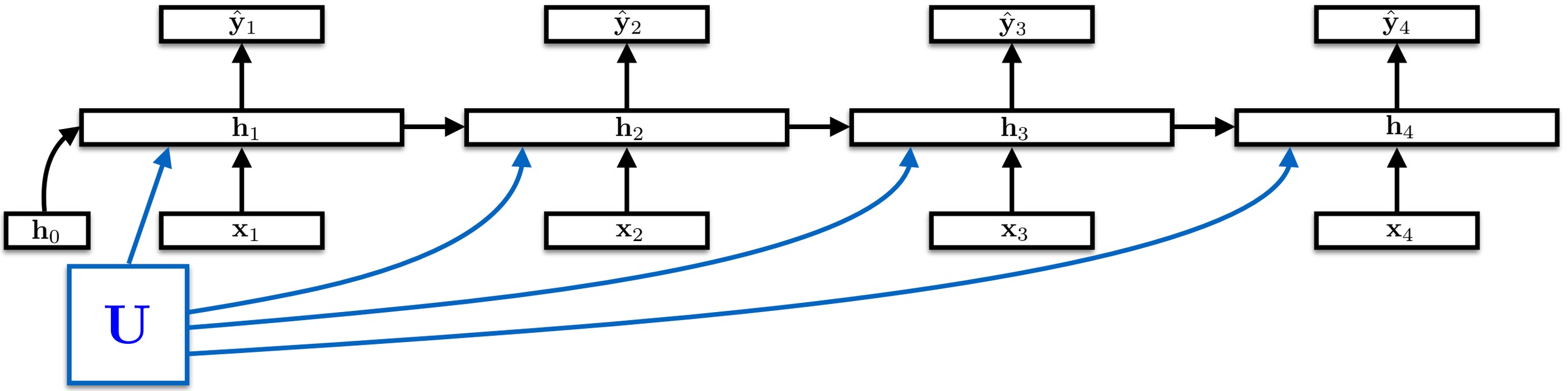
Parameter Tying

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

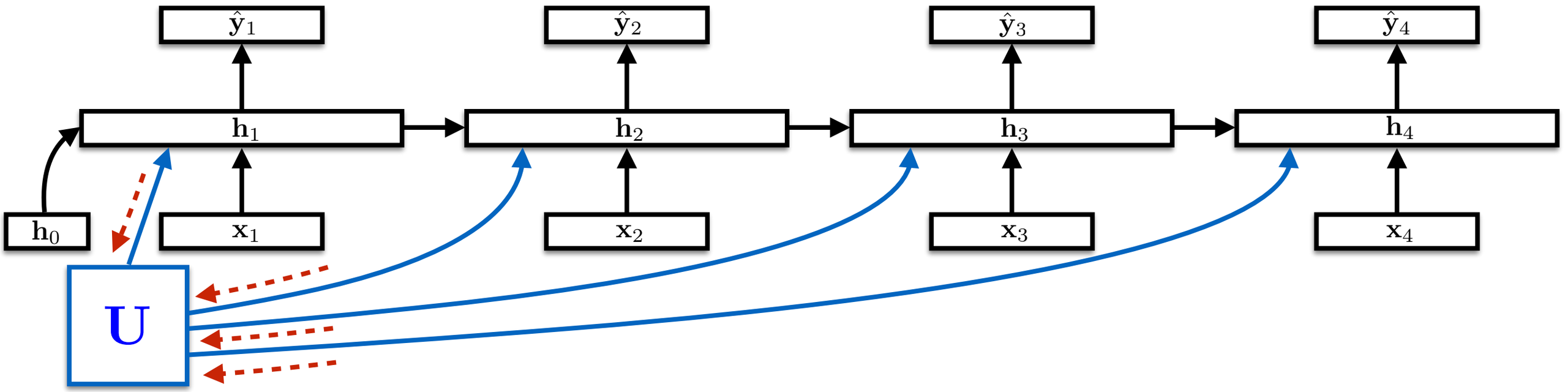
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Parameter Tying

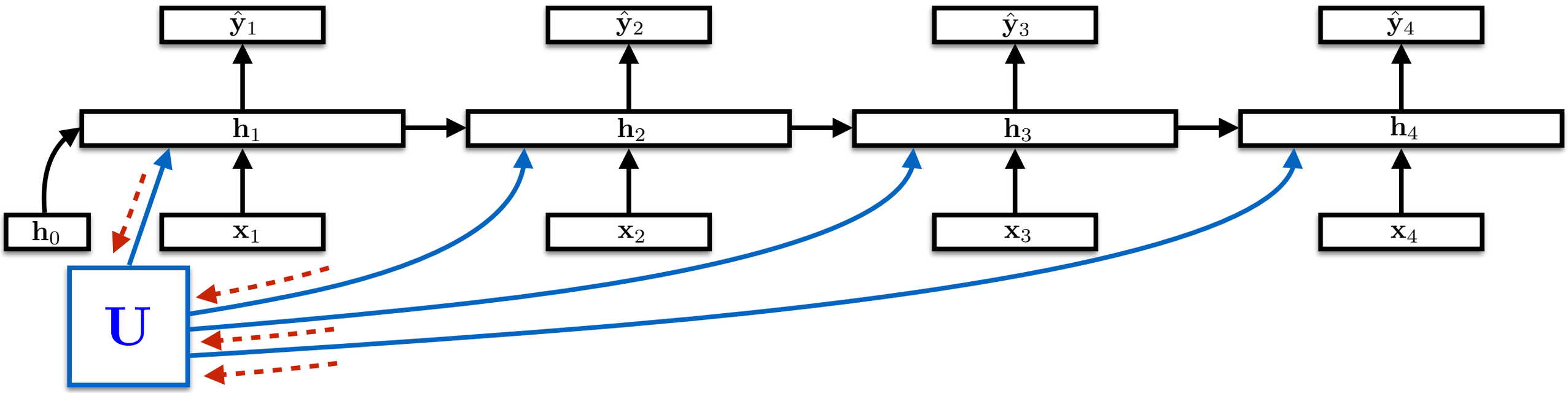


Parameter Tying



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

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!).

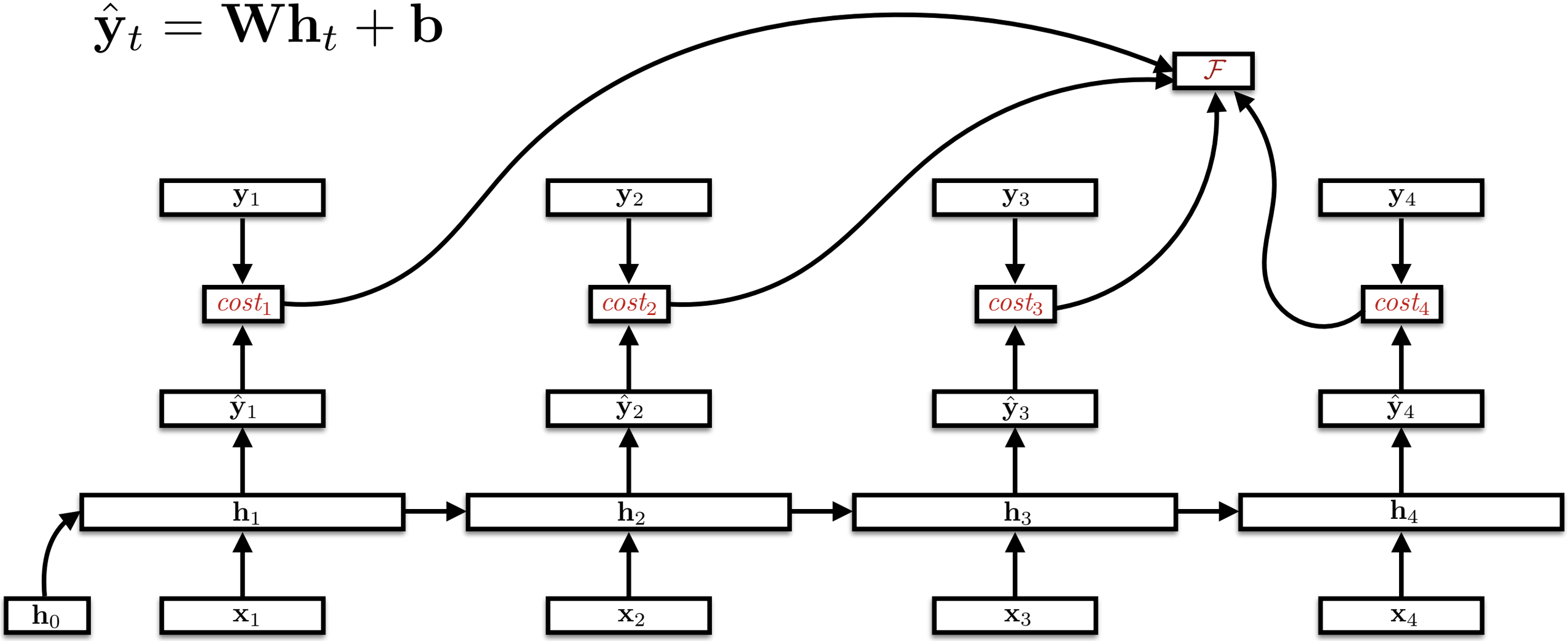
Parameter Tying

- 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?

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

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

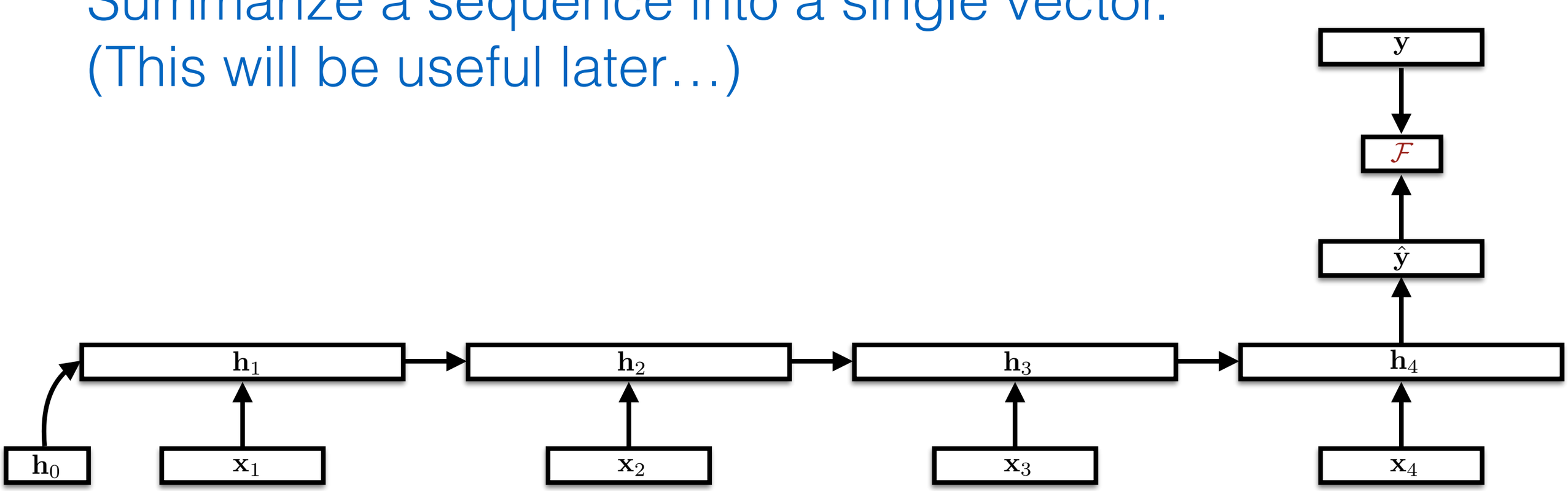


“Read and summarize”

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

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

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

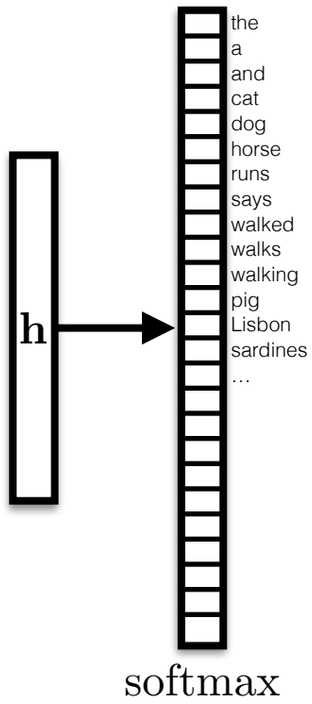
View 2: Recursive Definition

- 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.
  - **Learn** (or fix) 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



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

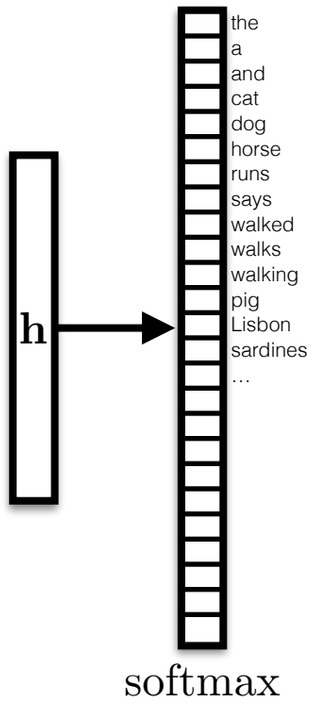
$$p_i = \frac{\exp u_i}{\sum_j \exp u_j}$$

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

$$|V| = 100,000$$

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

Example: Language Model



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

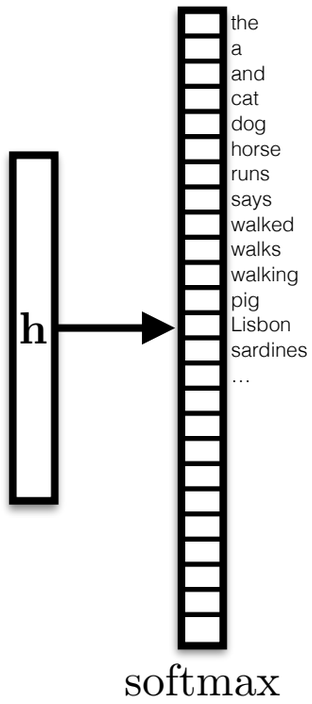
$$p_i = \frac{\exp u_i}{\sum_j \exp u_j}$$

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

$$|V| = 100,000$$

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

Example: Language Model



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

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

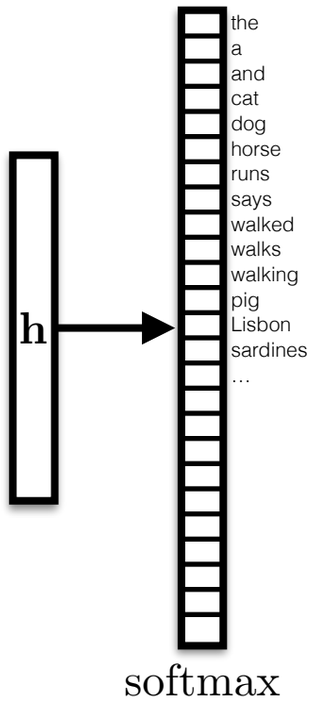
$$\mathbf{h} \in \mathbb{R}^d$$

$$|V| = 100,000$$

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

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

Example: Language Model



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

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

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

$$|V| = 100,000$$

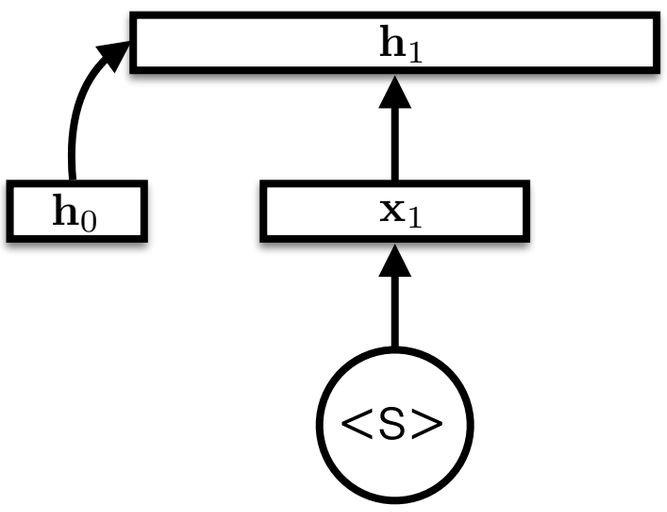
What are the dimensions of $\mathbf{b}$?

$$p(\boldsymbol{e}) = p(e_1) \times$$
$$p(e_2 \mid e_1) \times$$
$$p(e_3 \mid e_1, e_2) \times$$
$$p(e_4 \mid e_1, e_2, e_3) \times$$
$$\dots$$

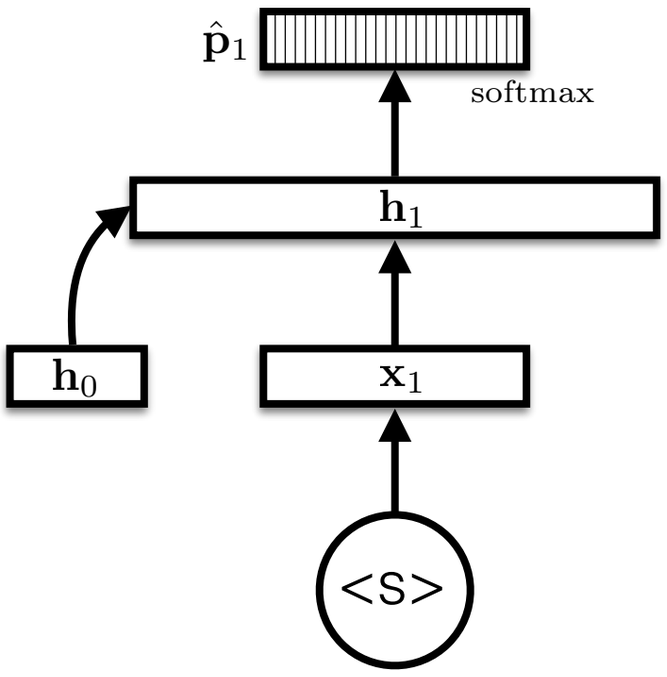
histories are sequences of words…

Example: Language Model

Example: Language Model

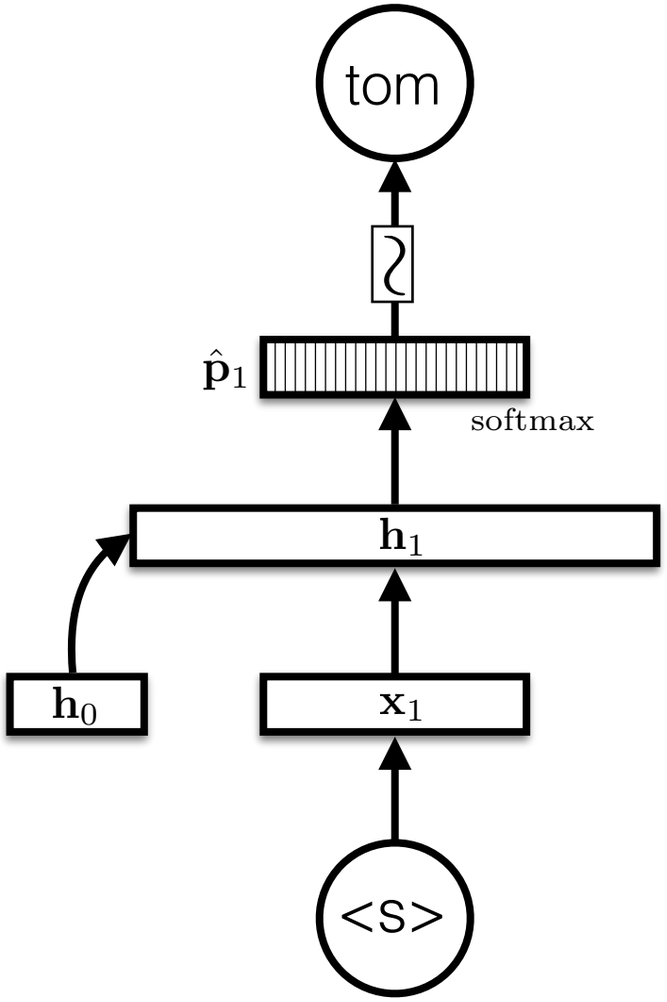


Example: Language Model



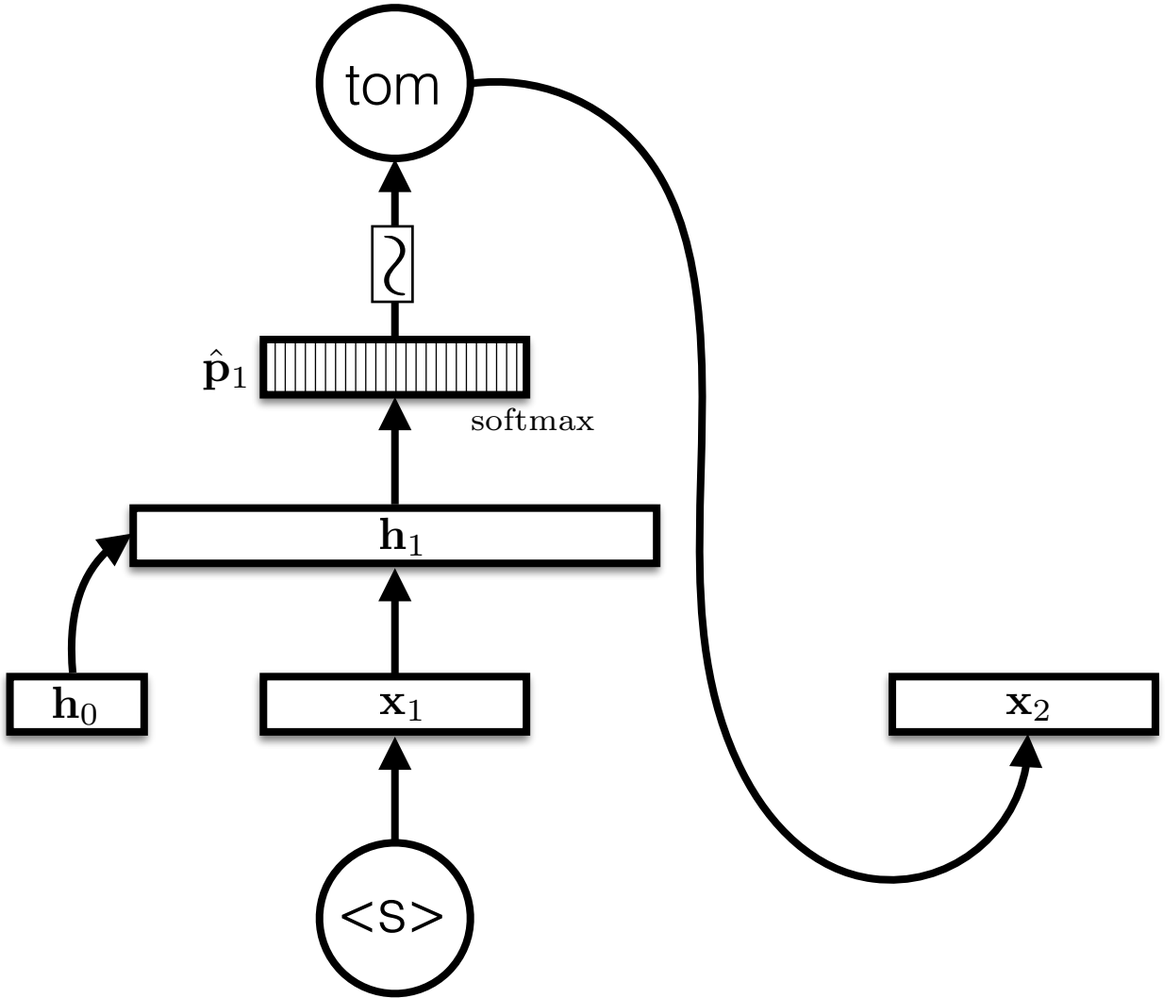
Example: Language Model

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



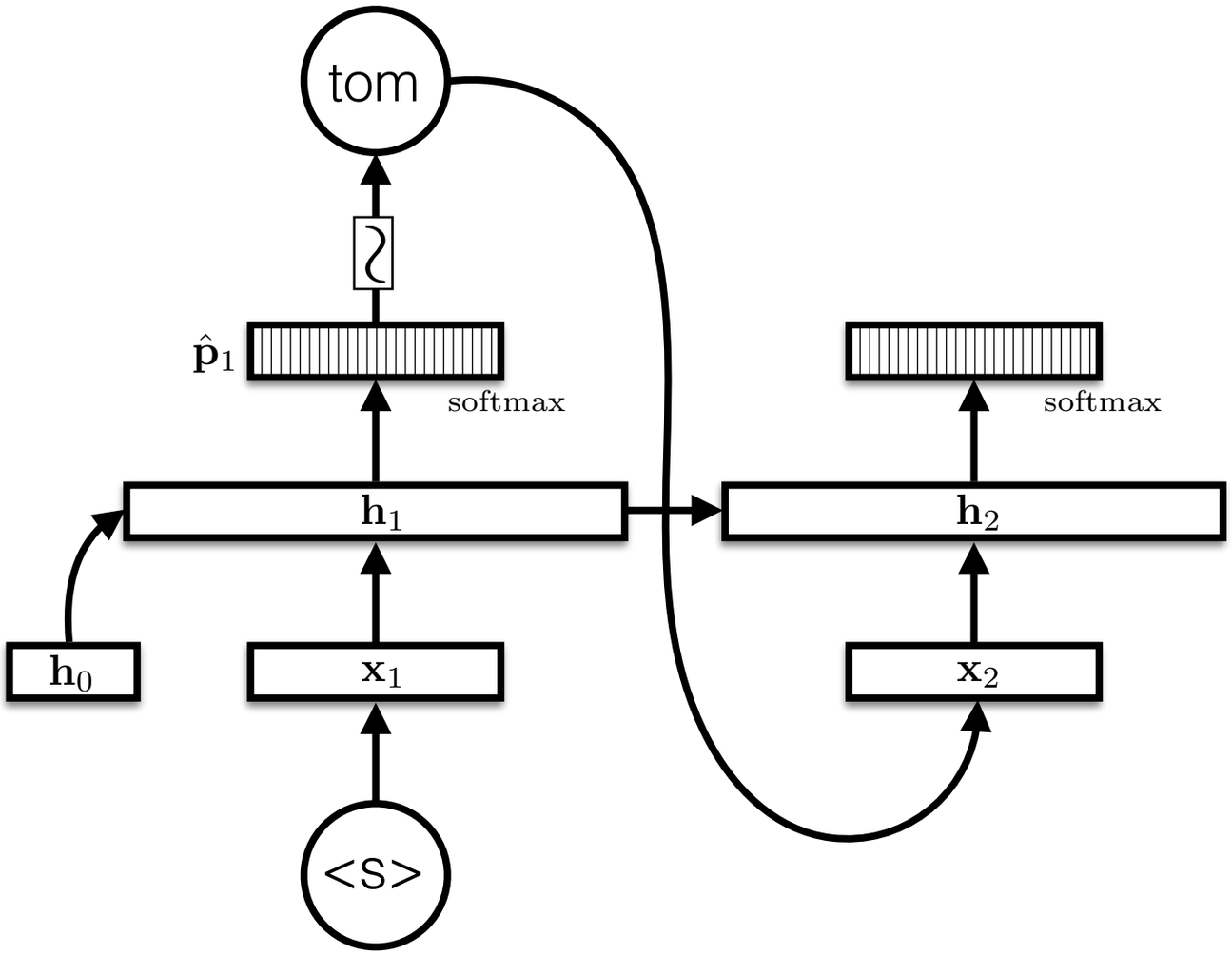
Example: Language Model

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



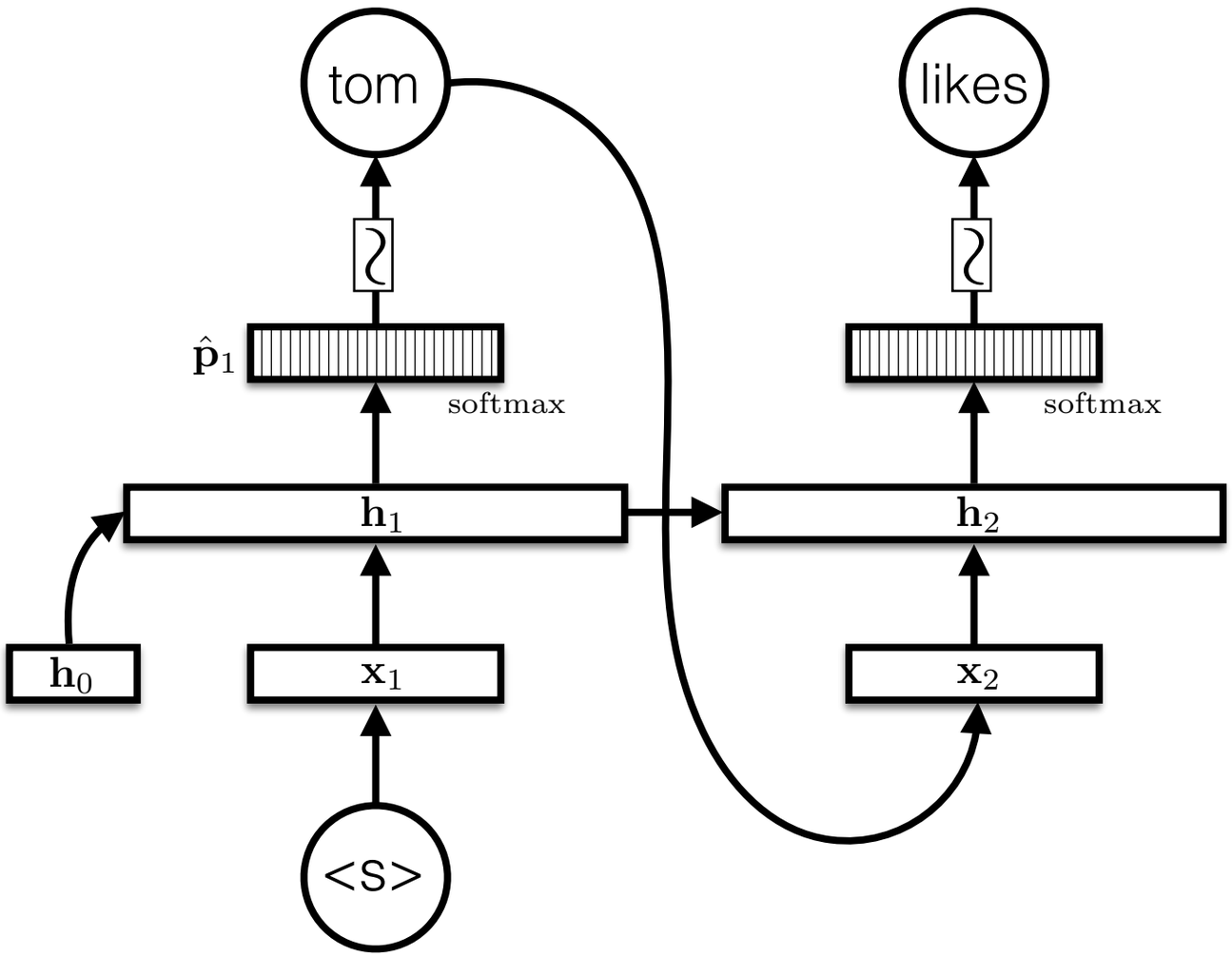
Example: Language Model

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



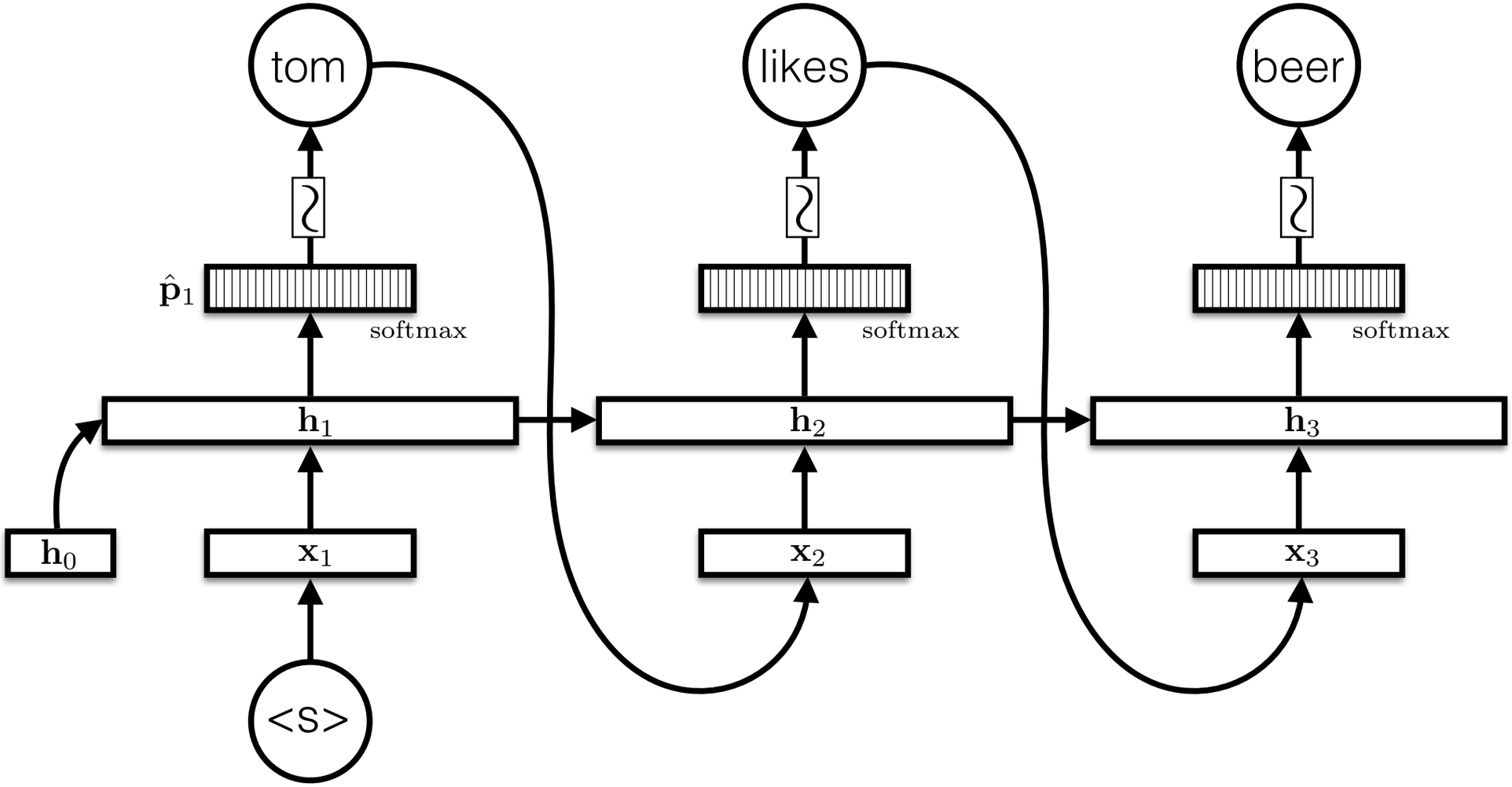
Example: Language Model

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



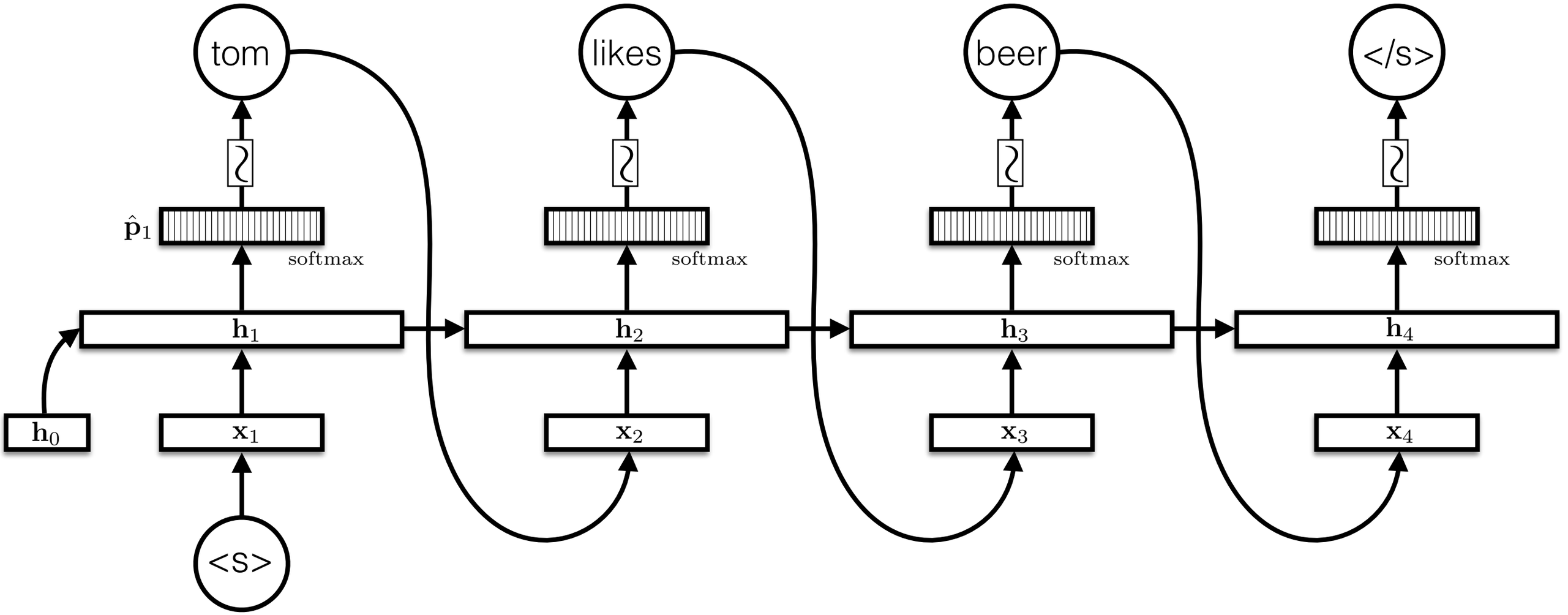
Example: Language Model

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

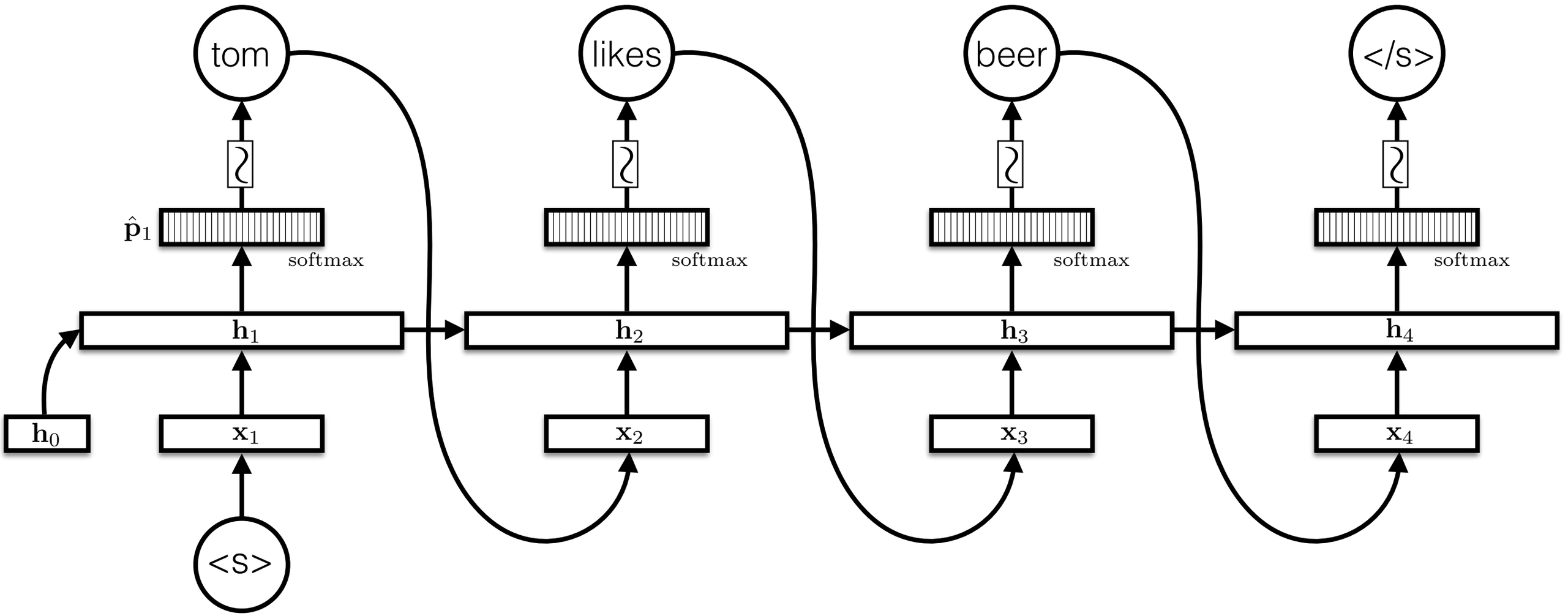


Example: Language Model

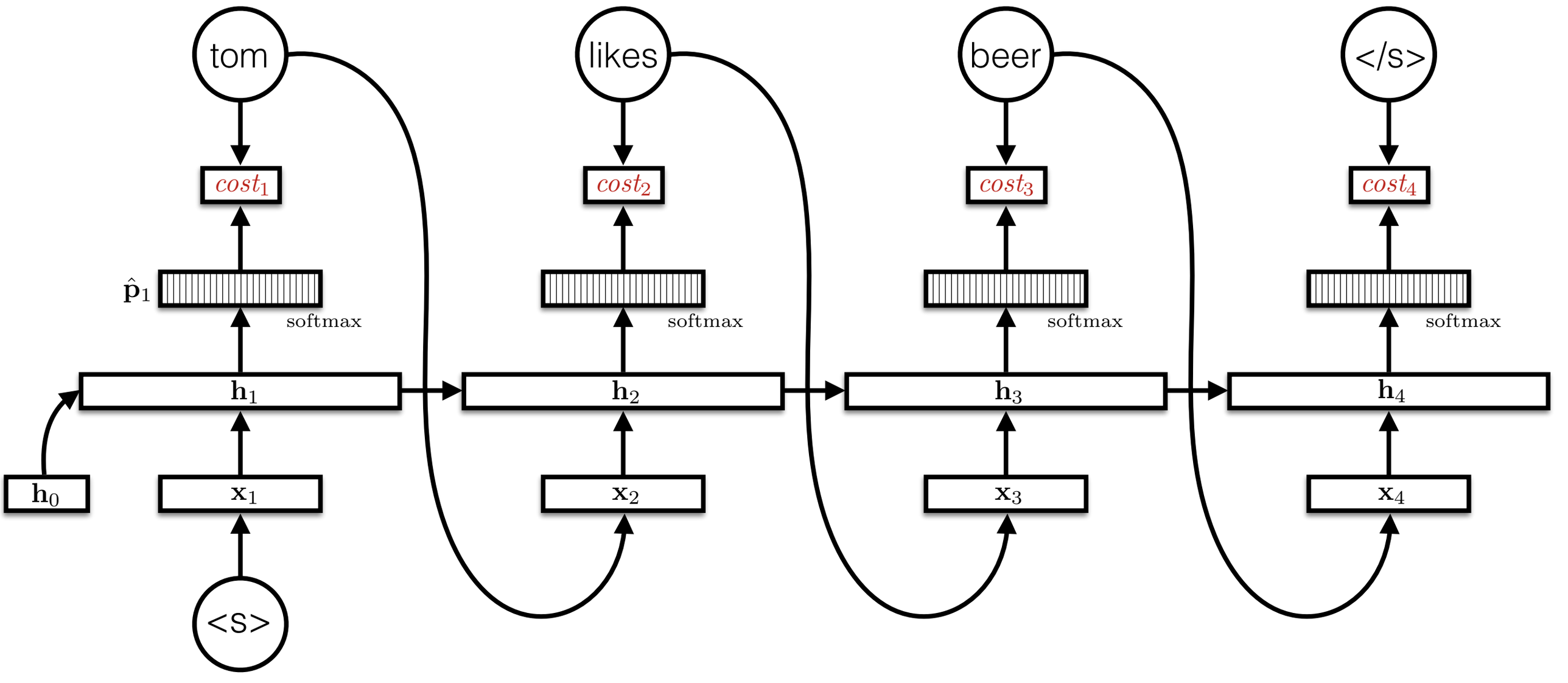
$$p(tom \mid \langle \texttt{s} \rangle) \times p(likes \mid \langle \texttt{s} \rangle, tom) \times p(beer \mid \langle \texttt{s} \rangle, tom, likes) \times p(\langle /\texttt{s} \rangle \mid \langle \texttt{s} \rangle, tom, likes, beer)$$



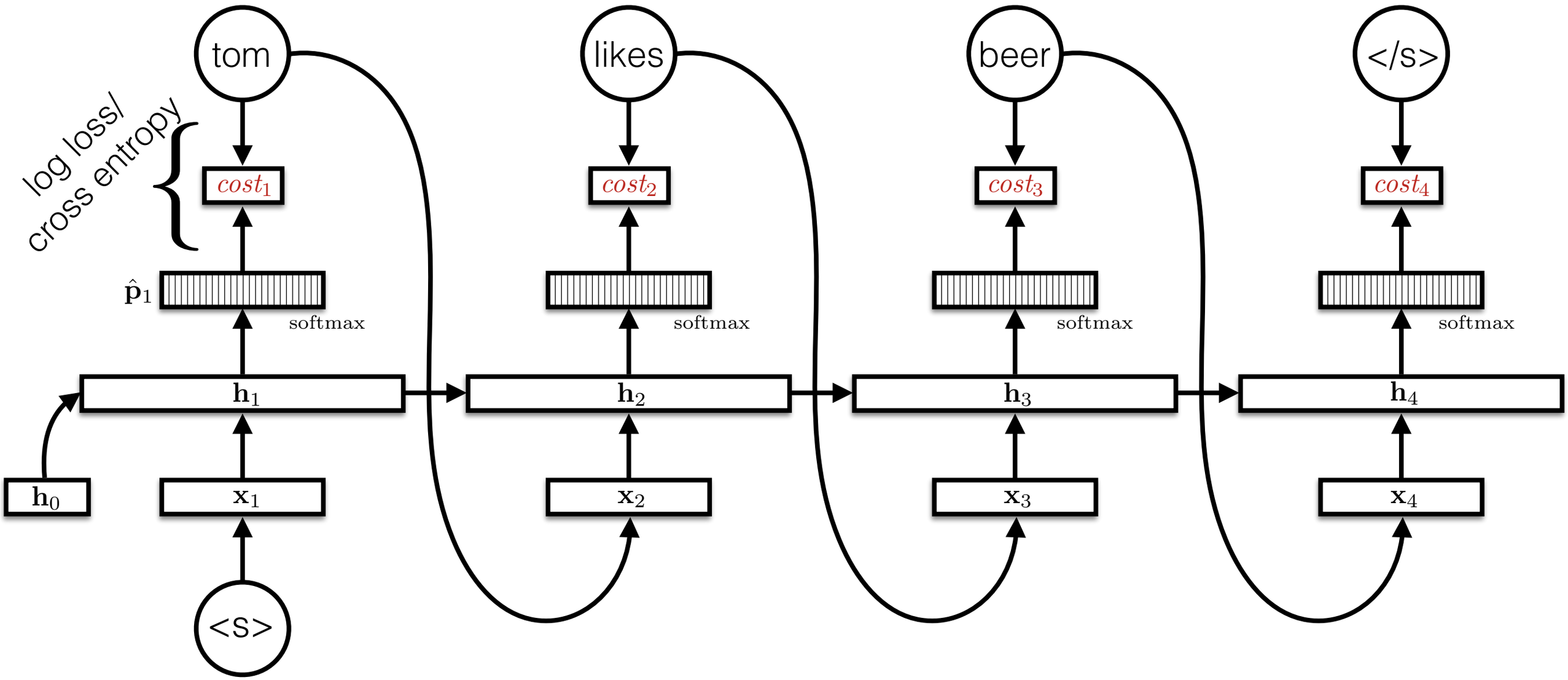
Language Model Training



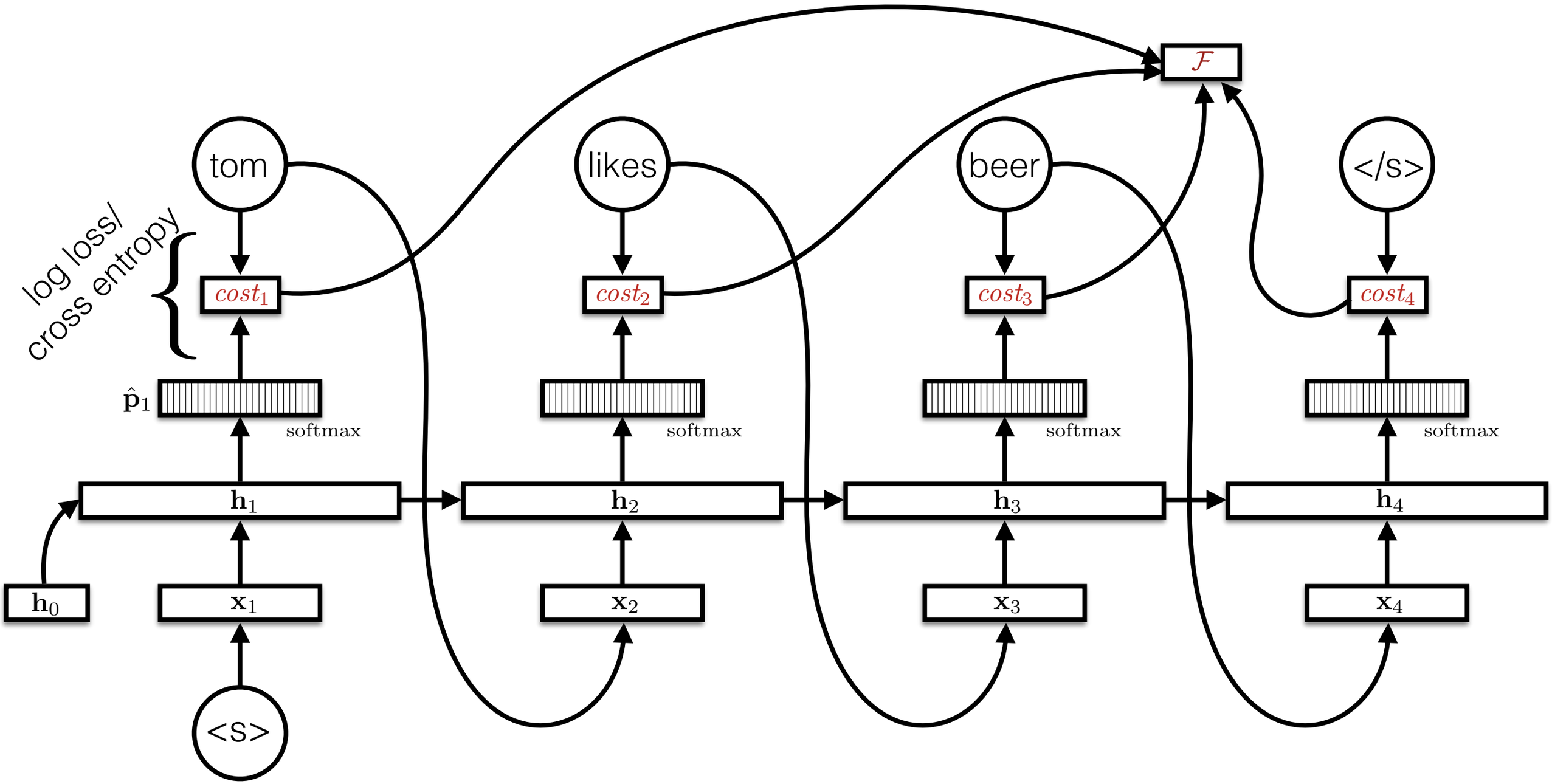
Language Model Training



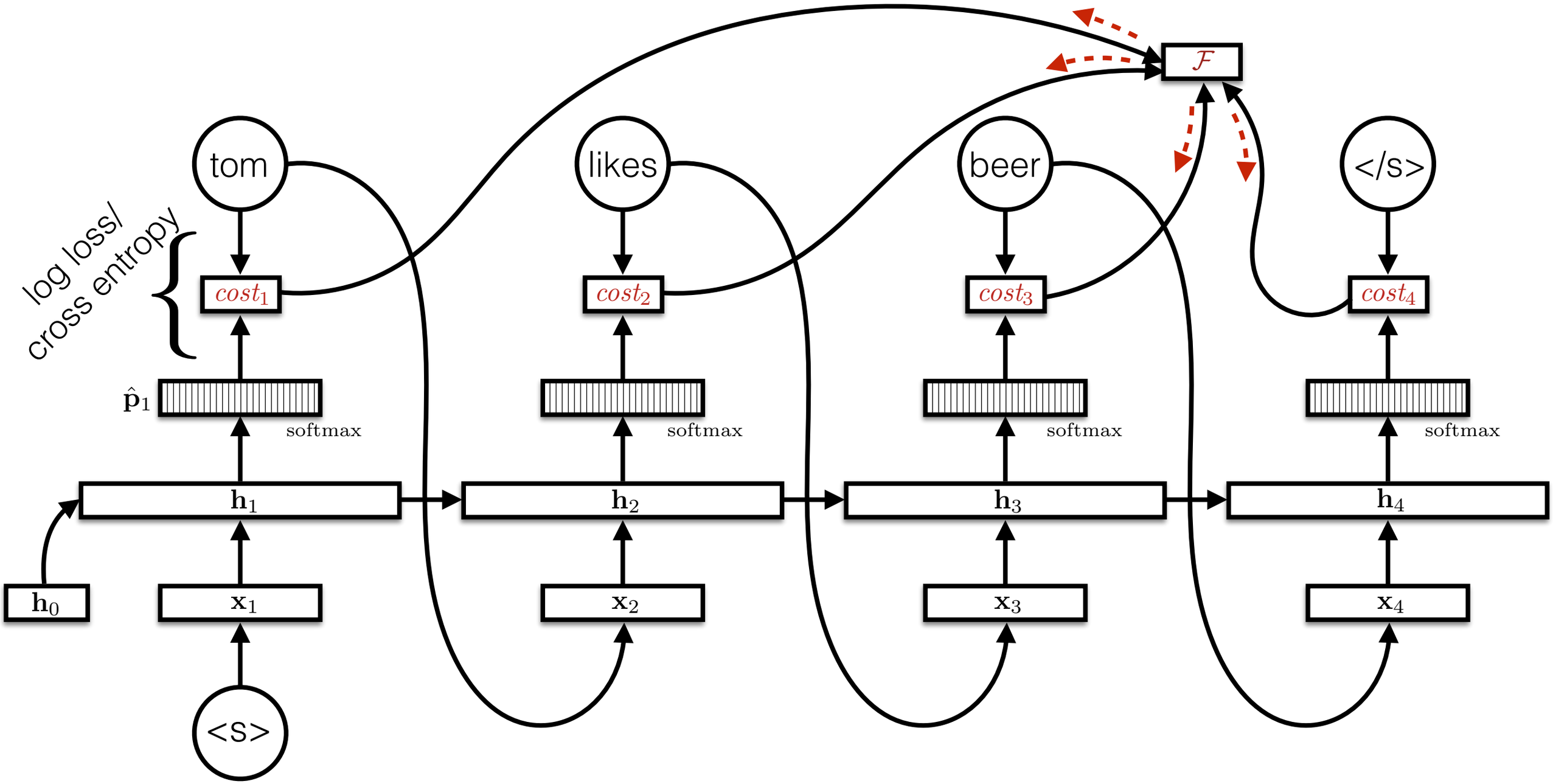
Language Model Training



Language Model Training



Language Model Training



RNN Language Models

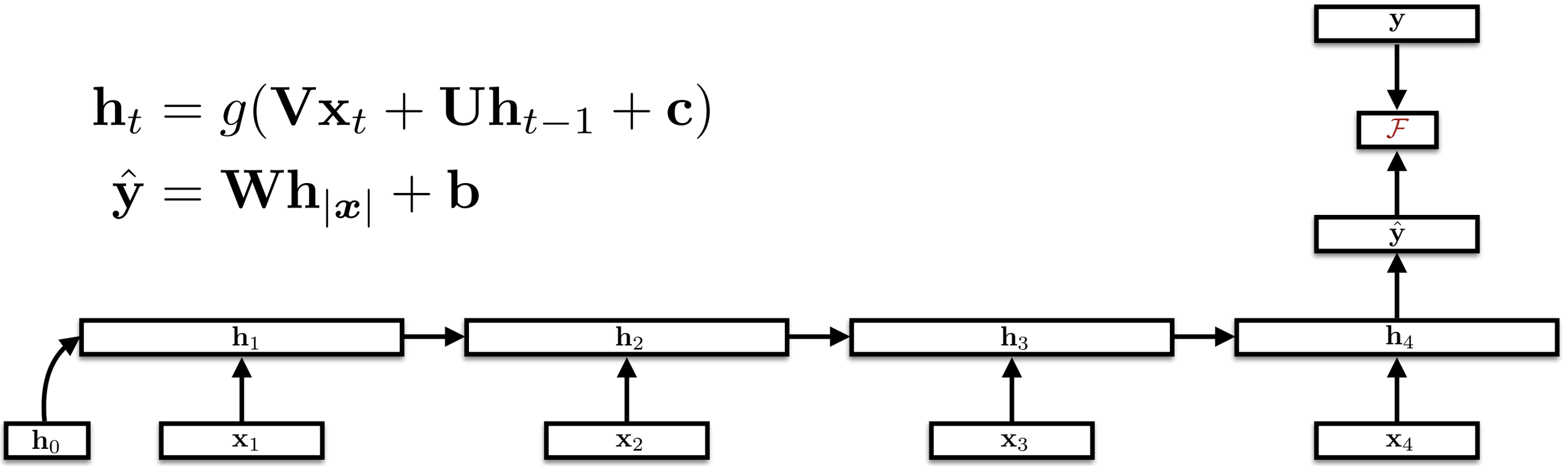
- Unlike Markov (*n*-gram) models, RNNs never forget
    - However we will see they might have trouble learning to use their memories (more soon…)
- Algorithms
    - Sample a sequence from the probability distribution defined by the RNN
    - Train the RNN to minimize cross entropy (aka MLE)
    - What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

Questions?

Training Challenges

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

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



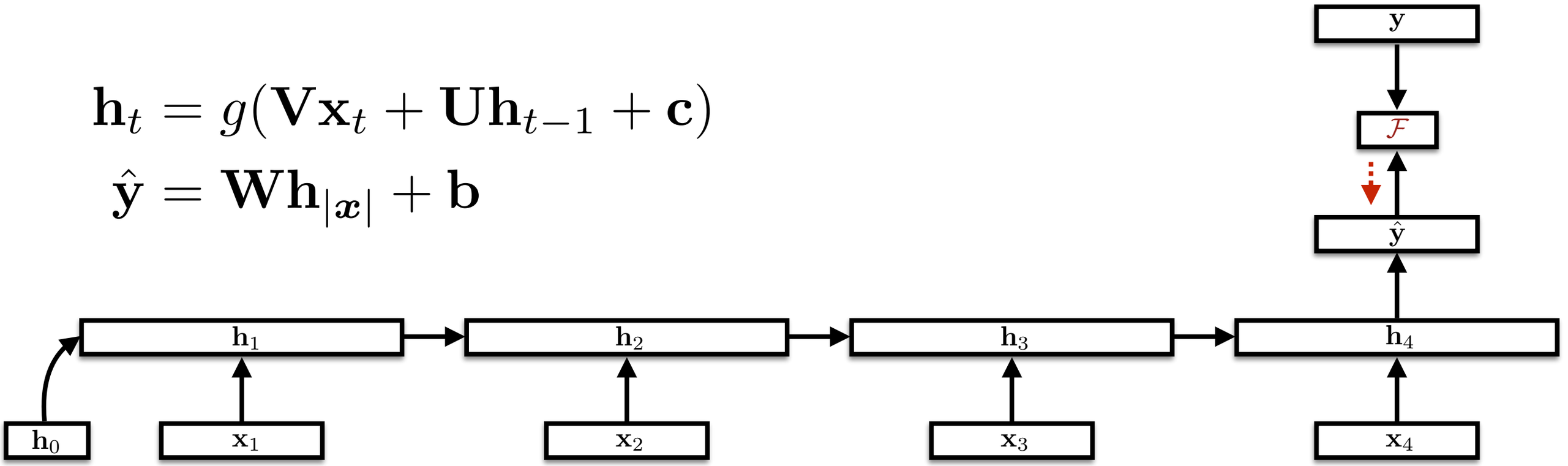
What happens to gradients as you go back in time?

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

Training Challenges

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

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



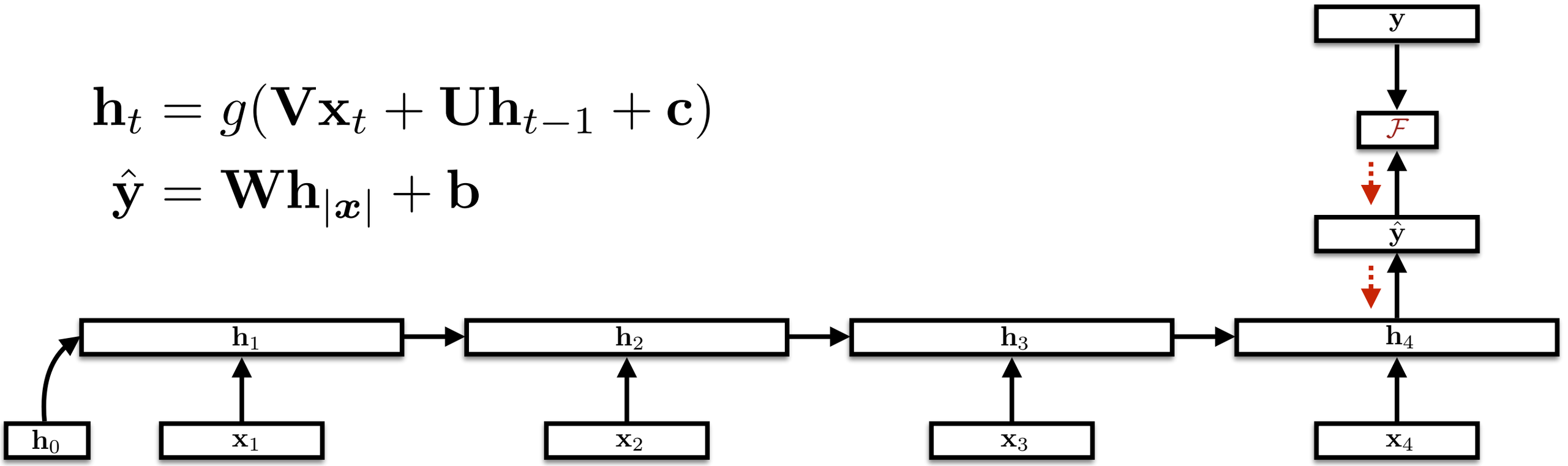
What happens to gradients as you go back in time?

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

Training Challenges

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

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



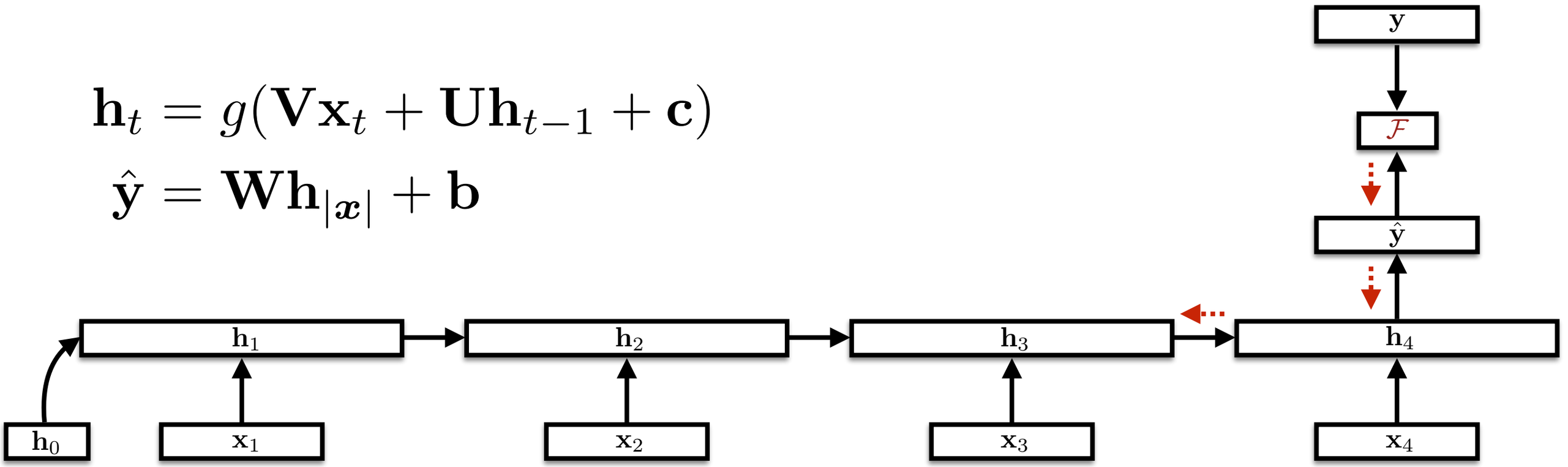
What happens to gradients as you go back in time?

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

Training Challenges

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

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



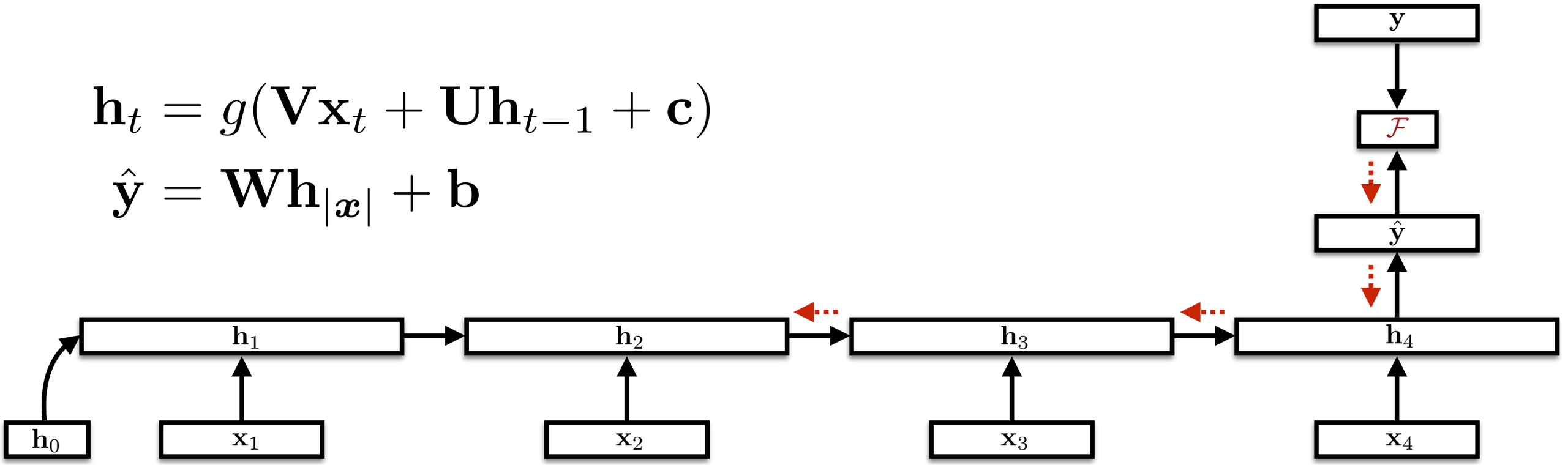
What happens to gradients as you go back in time?

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

Training Challenges

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

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



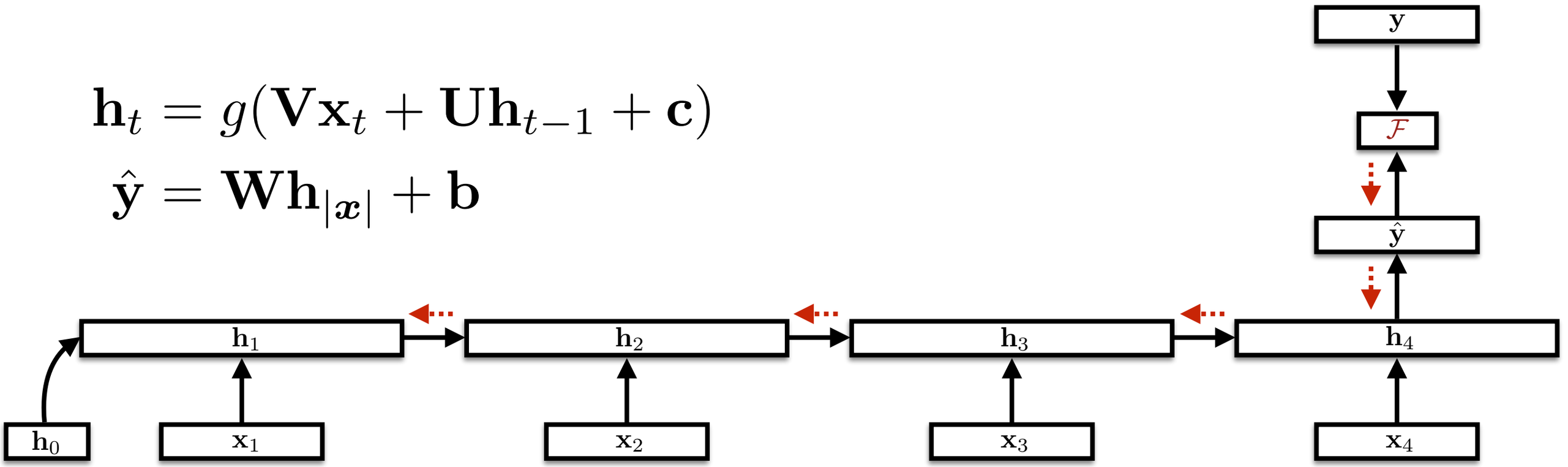
What happens to gradients as you go back in time?

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

Training Challenges

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

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



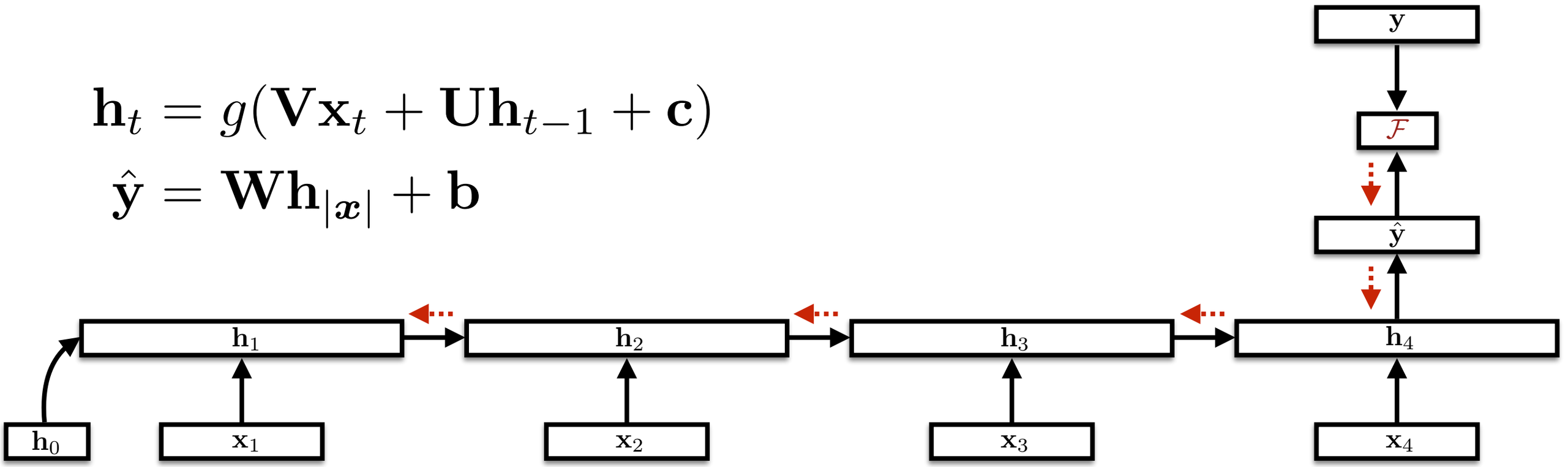
What happens to gradients as you go back in time?

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

Training Challenges

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

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



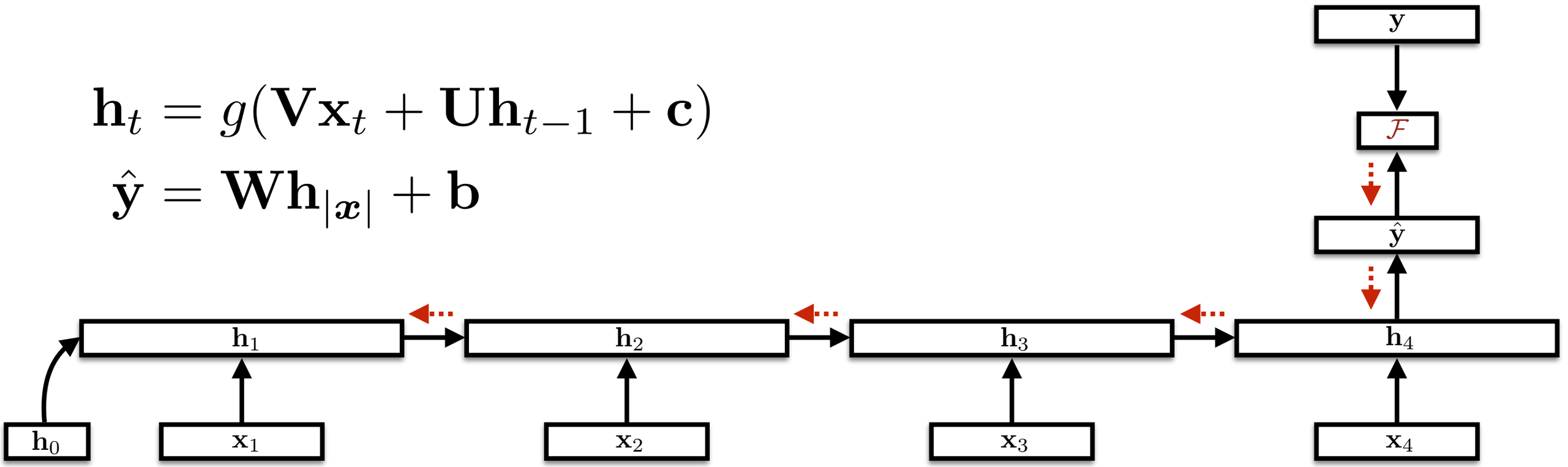
What happens to gradients as you go back in time?

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

Training Challenges

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

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



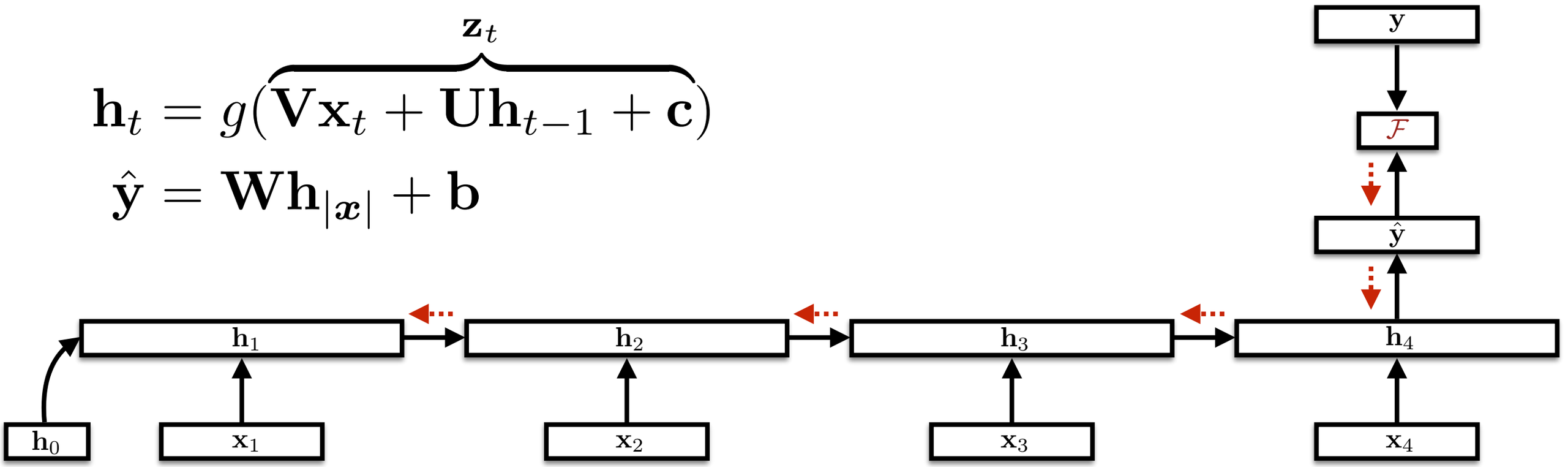
What happens to gradients as you go back in time?

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

Training Challenges

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

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



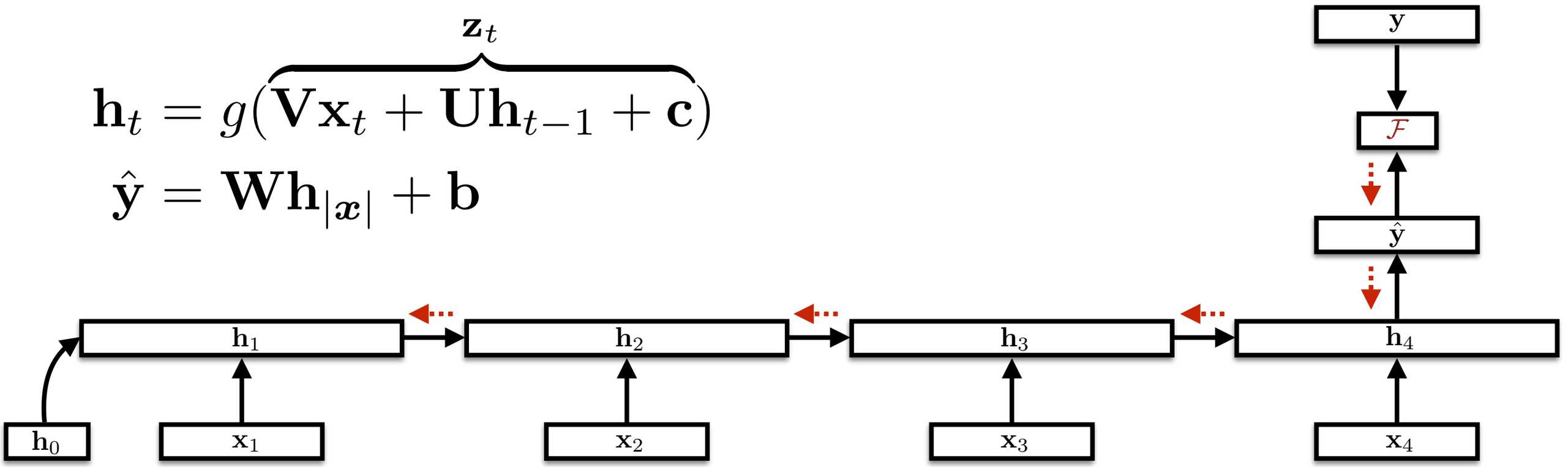
What happens to gradients as you go back in time?

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

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)

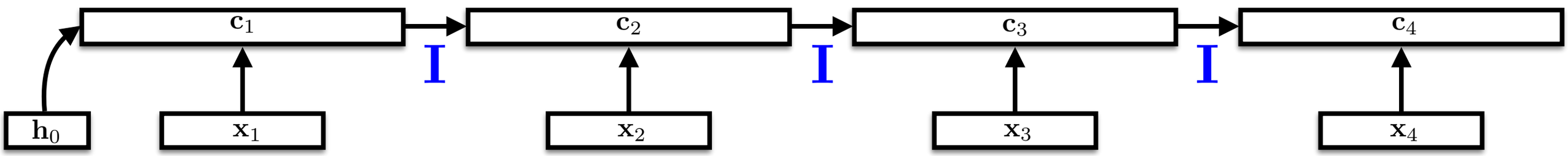
- Gated recurrent units (GRUs; Cho et al., 2014)

- 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

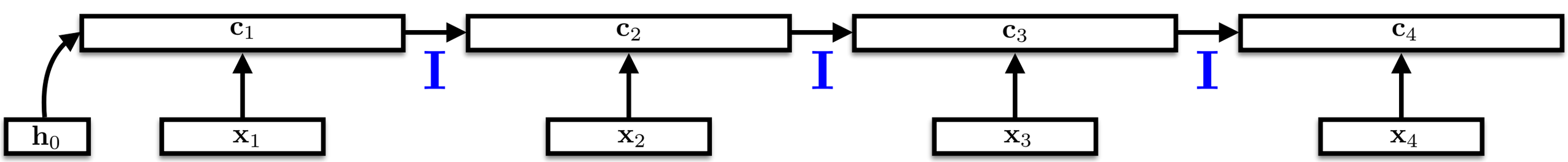
Memory cells

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



Memory cells

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

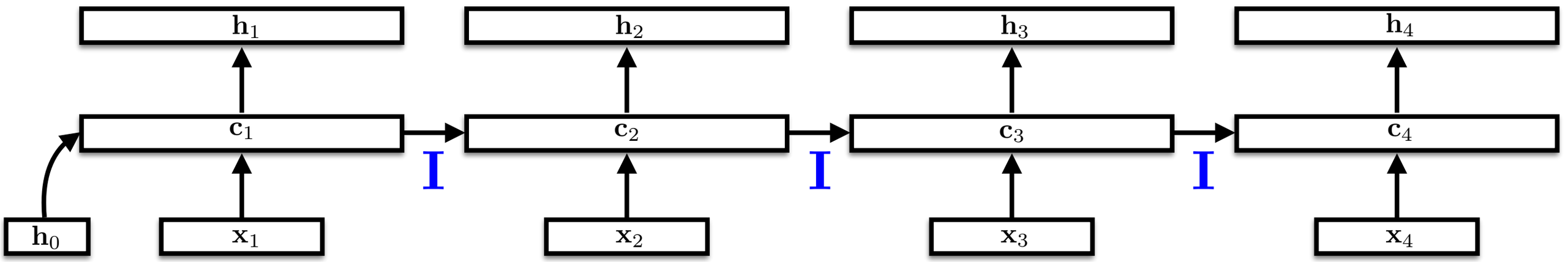


Memory cells

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

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

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

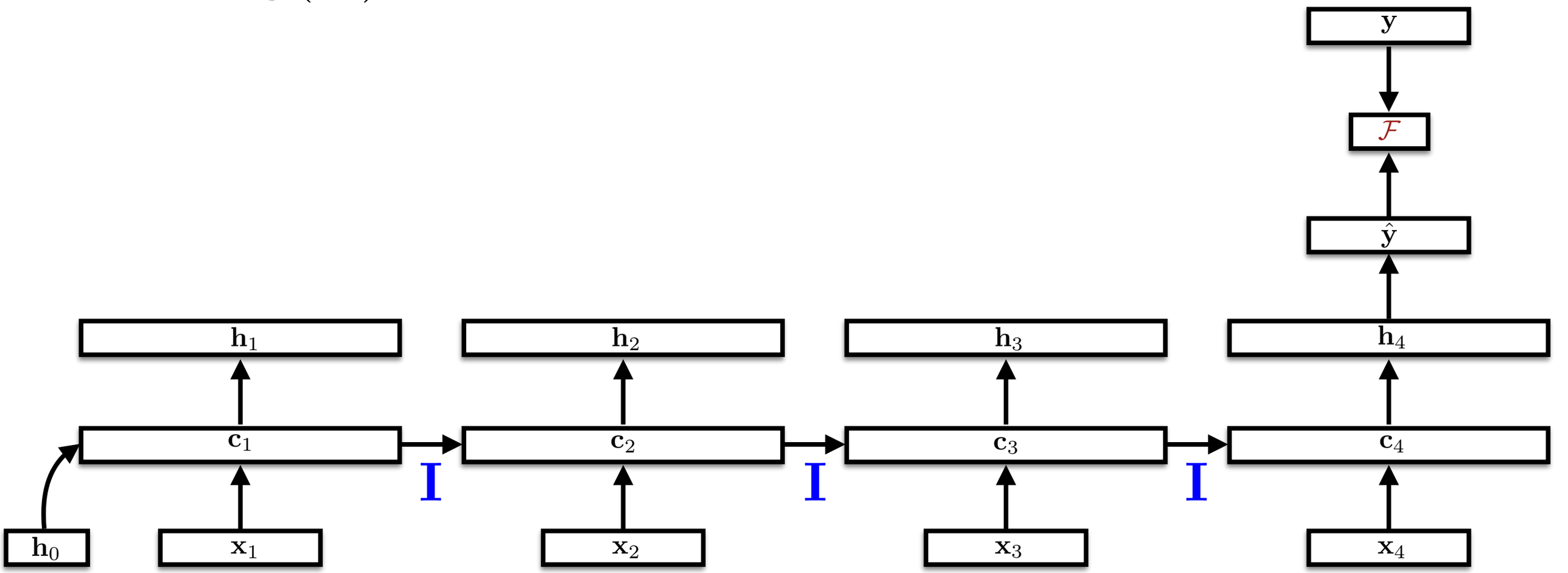


Memory cells

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

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

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



Memory cells

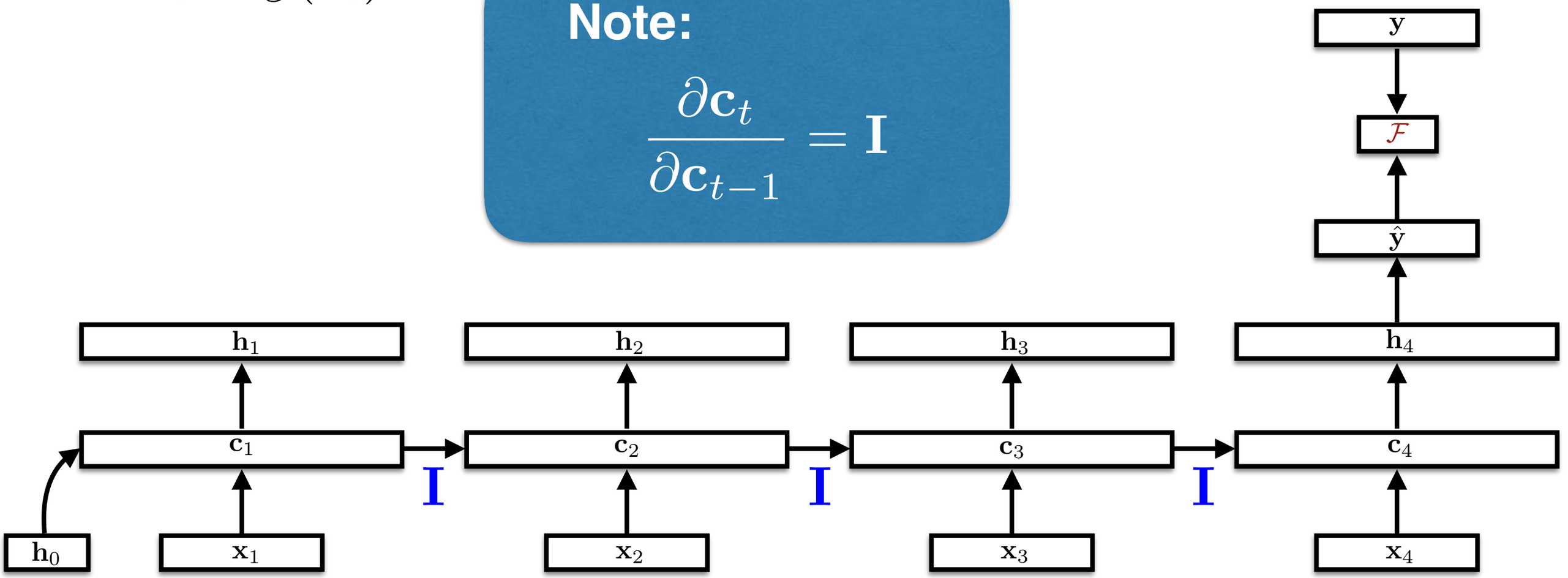
$$\mathbf{c}_t = \mathbf{c}_{t-1} + f(\mathbf{x}_t)$$

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

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

Note:

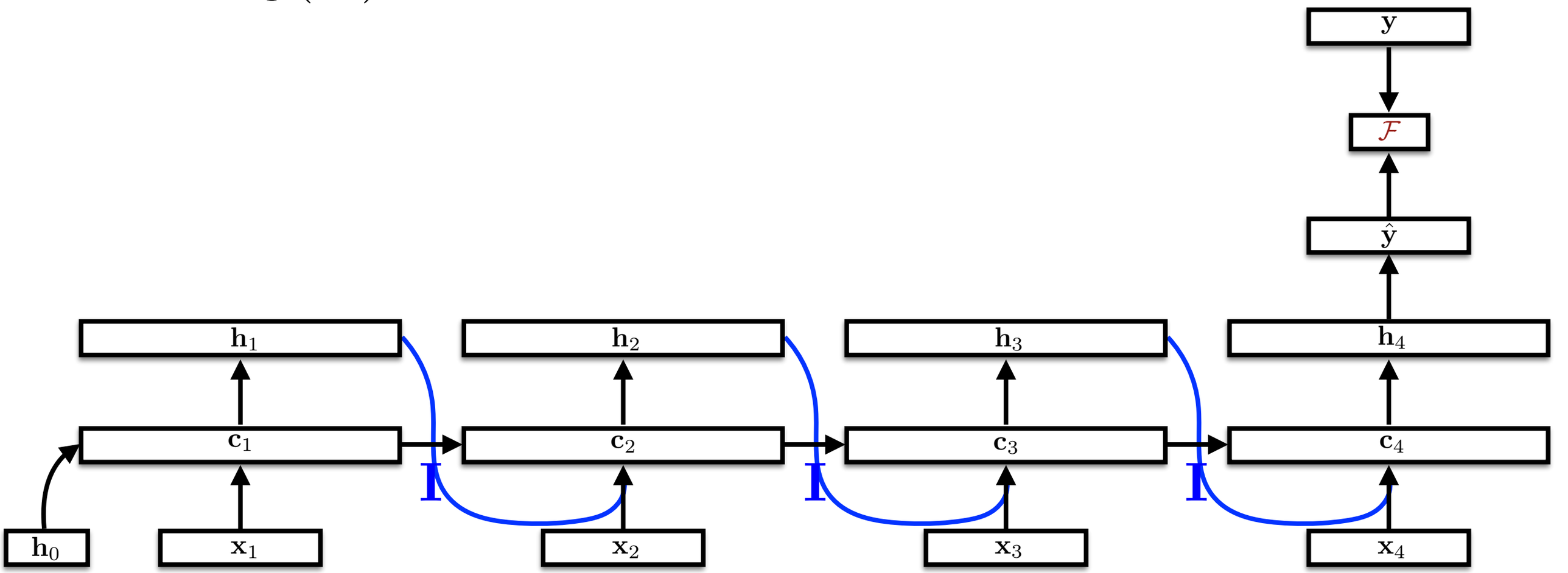
$$\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = \mathbf{I}$$



Memory cells

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

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



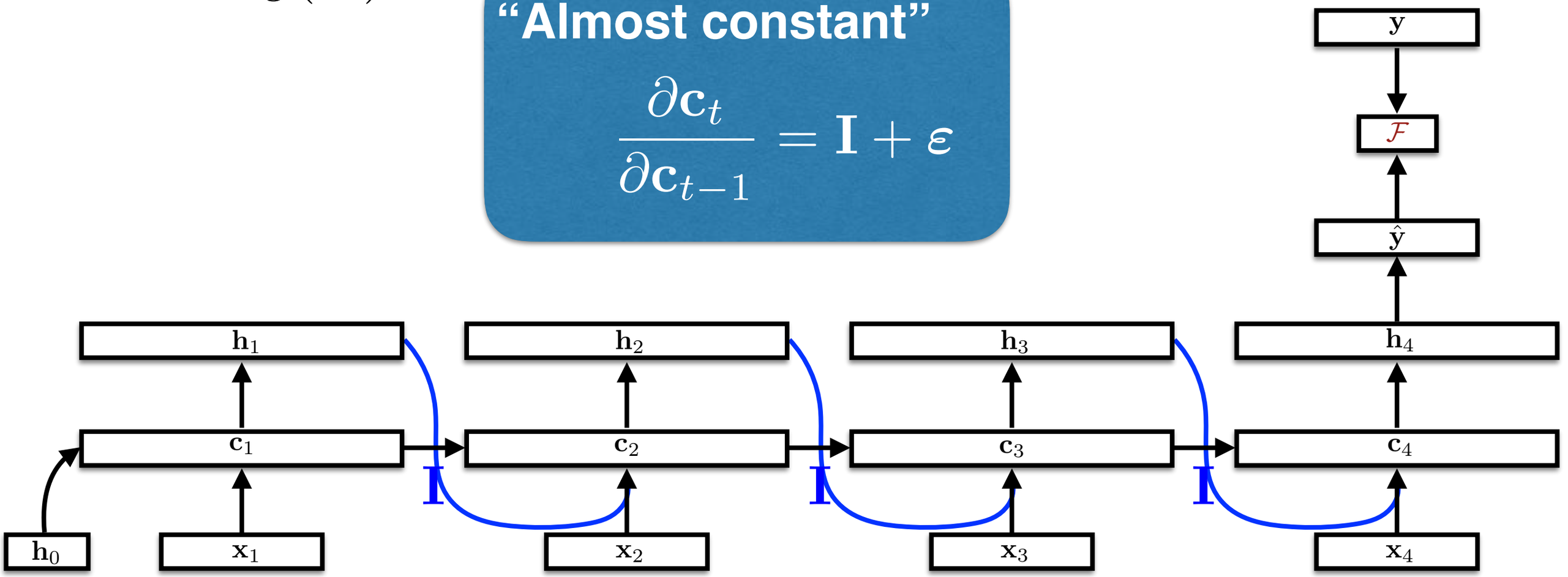
Memory cells

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

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

"Almost constant"

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



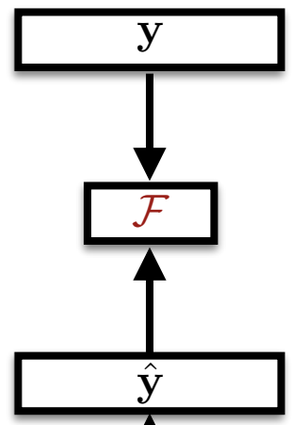
Memory cells

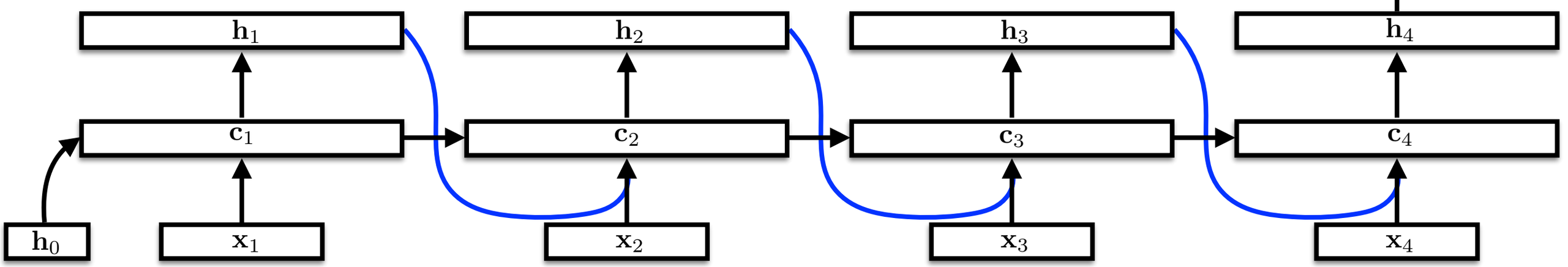
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

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

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

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





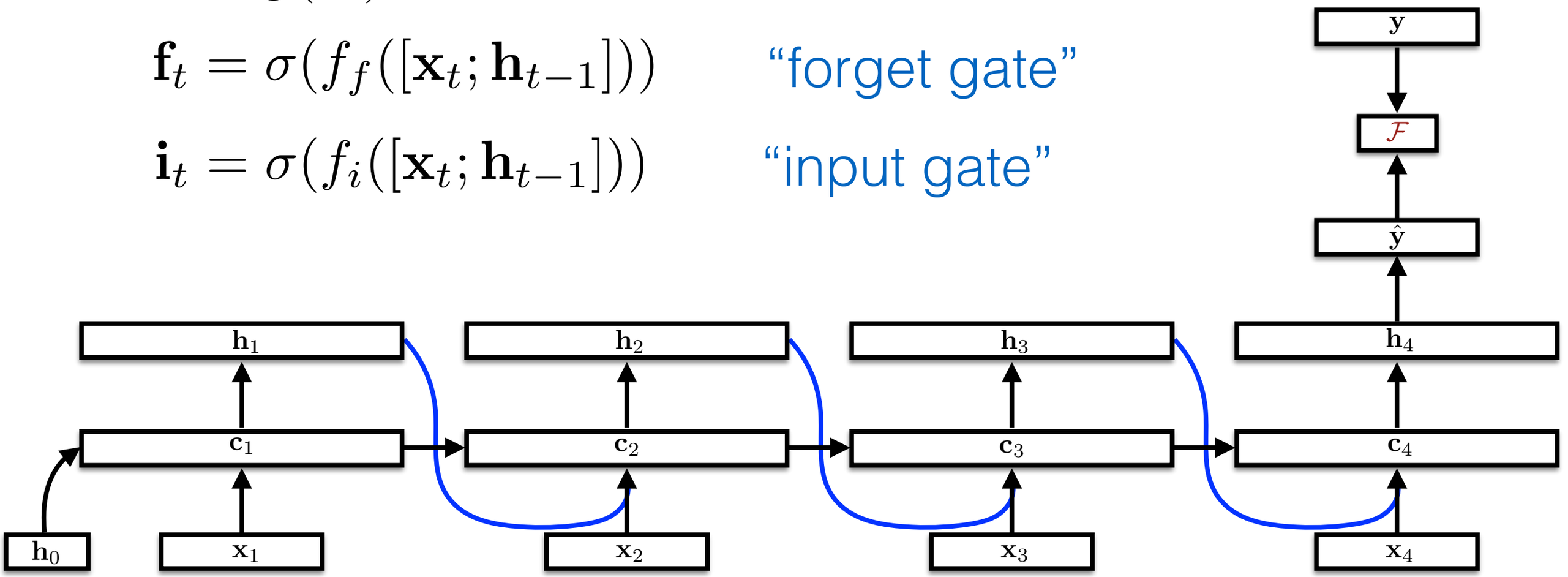
Memory cells

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

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

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

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



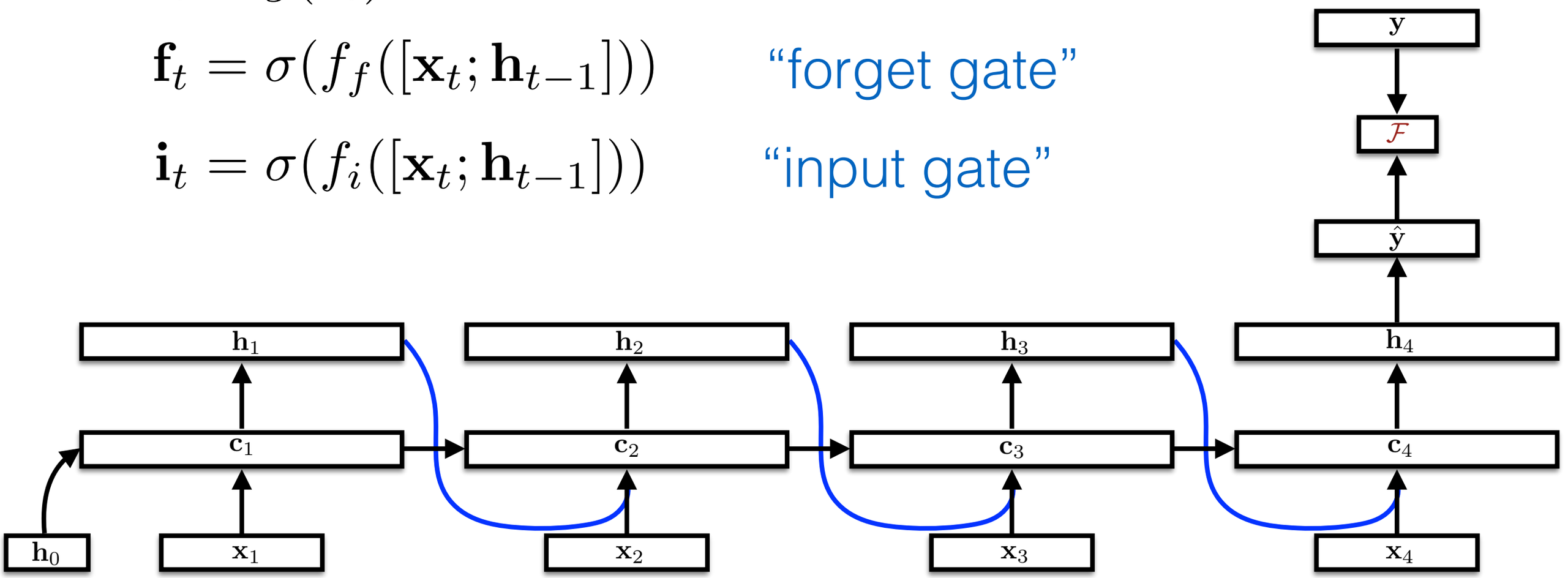
Memory cells

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

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

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

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



LSTM

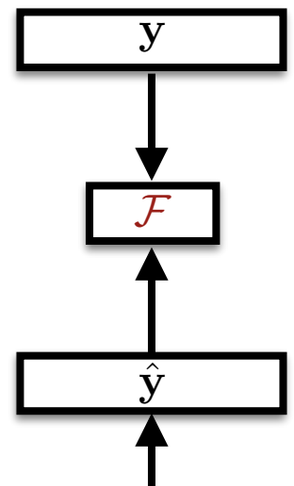
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

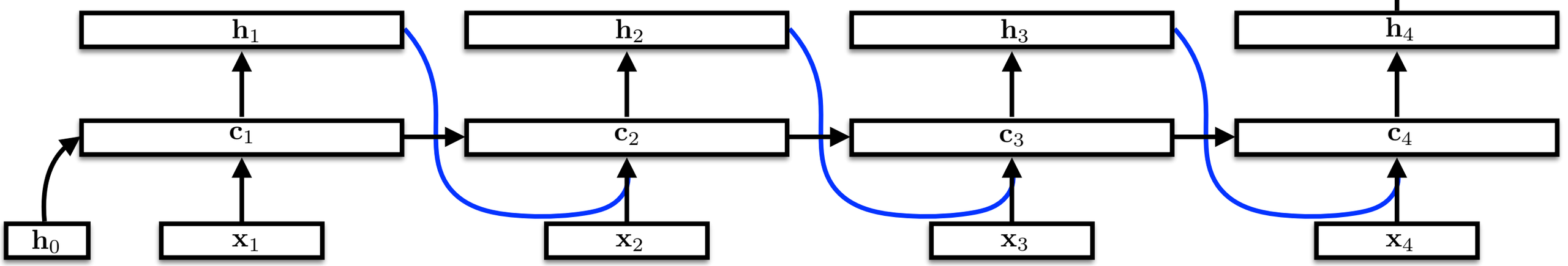
$$\mathbf{h}_t = \mathbf{o}_t \odot g(\mathbf{c}_t)$$

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

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

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





LSTM

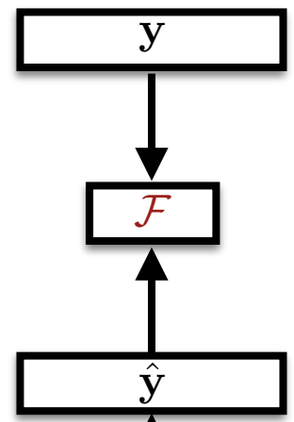
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

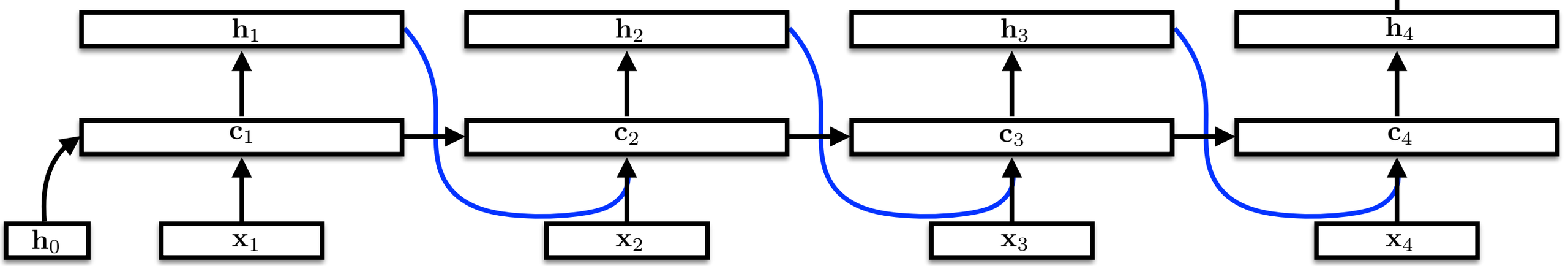
$$\mathbf{h}_t = \mathbf{o}_t \odot g(\mathbf{c}_t)$$

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

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

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





LSTM Variant

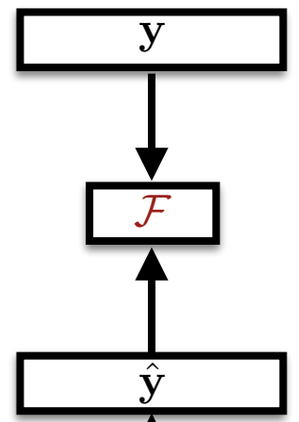
$$\mathbf{c}_t = (\mathbf{1} - \mathbf{i}_t) \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

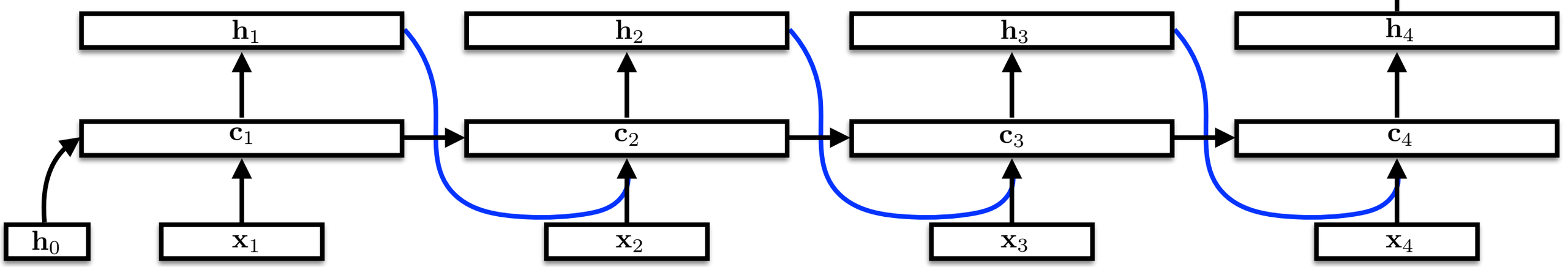
$$\mathbf{h}_t = \mathbf{o}_t \odot g(\mathbf{c}_t)$$

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

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

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





LSTM Variant

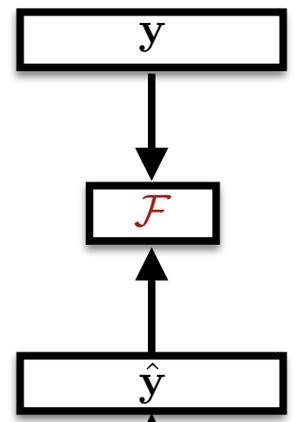
$$\mathbf{c}_t = (\mathbf{1} - \mathbf{i}_t) \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

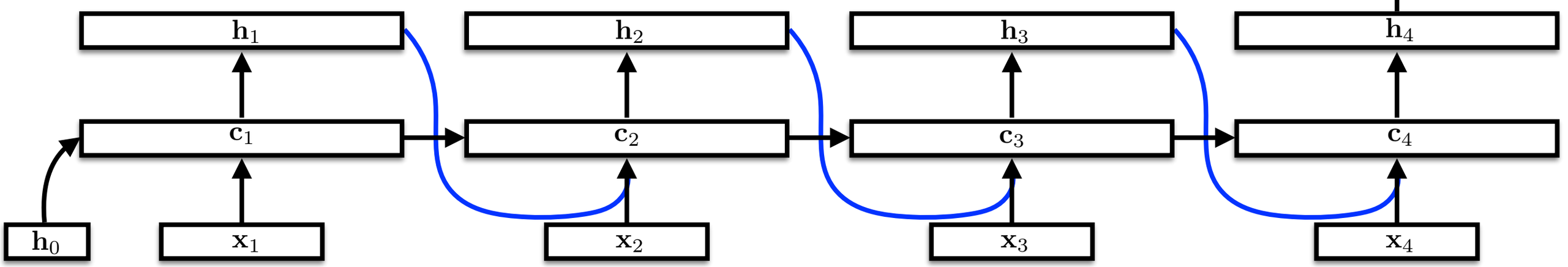
$$\mathbf{h}_t = \mathbf{o}_t \odot g(\mathbf{c}_t)$$

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

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

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





Another Visualization

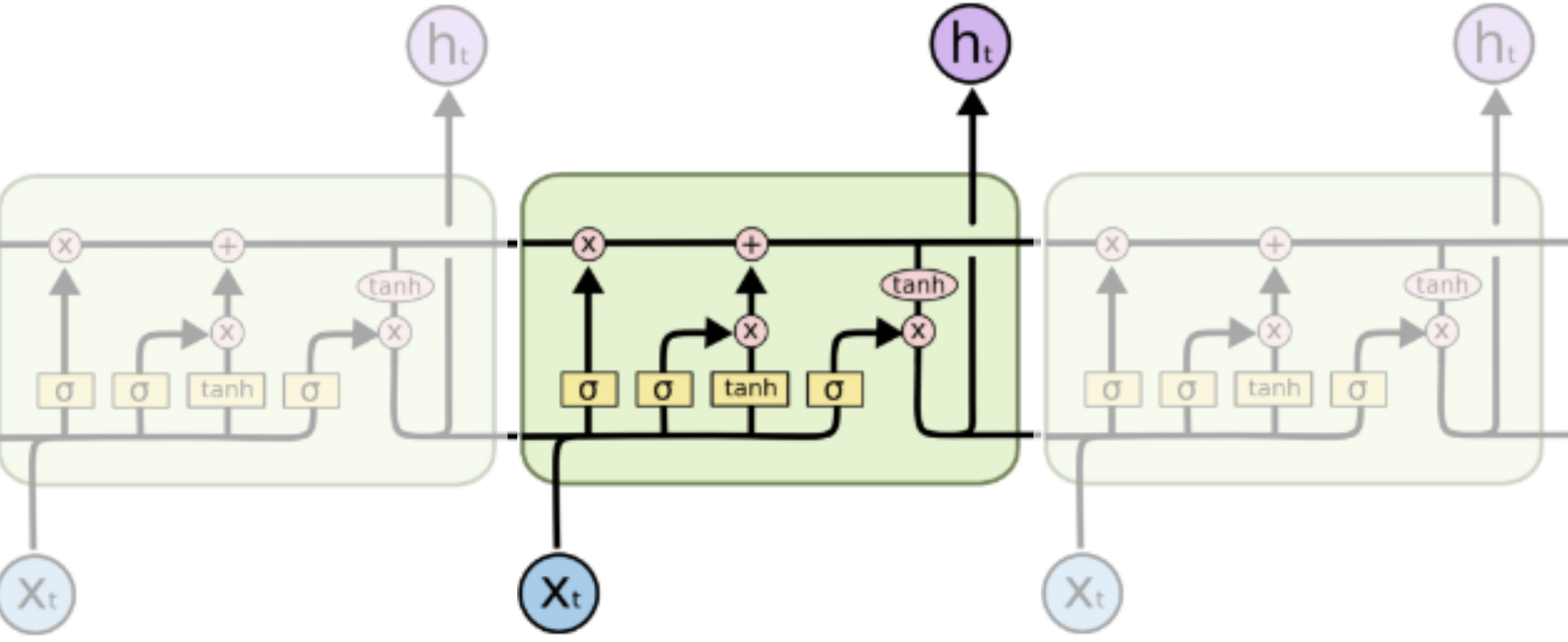


Figure credit: Christopher Olah

Another Visualization

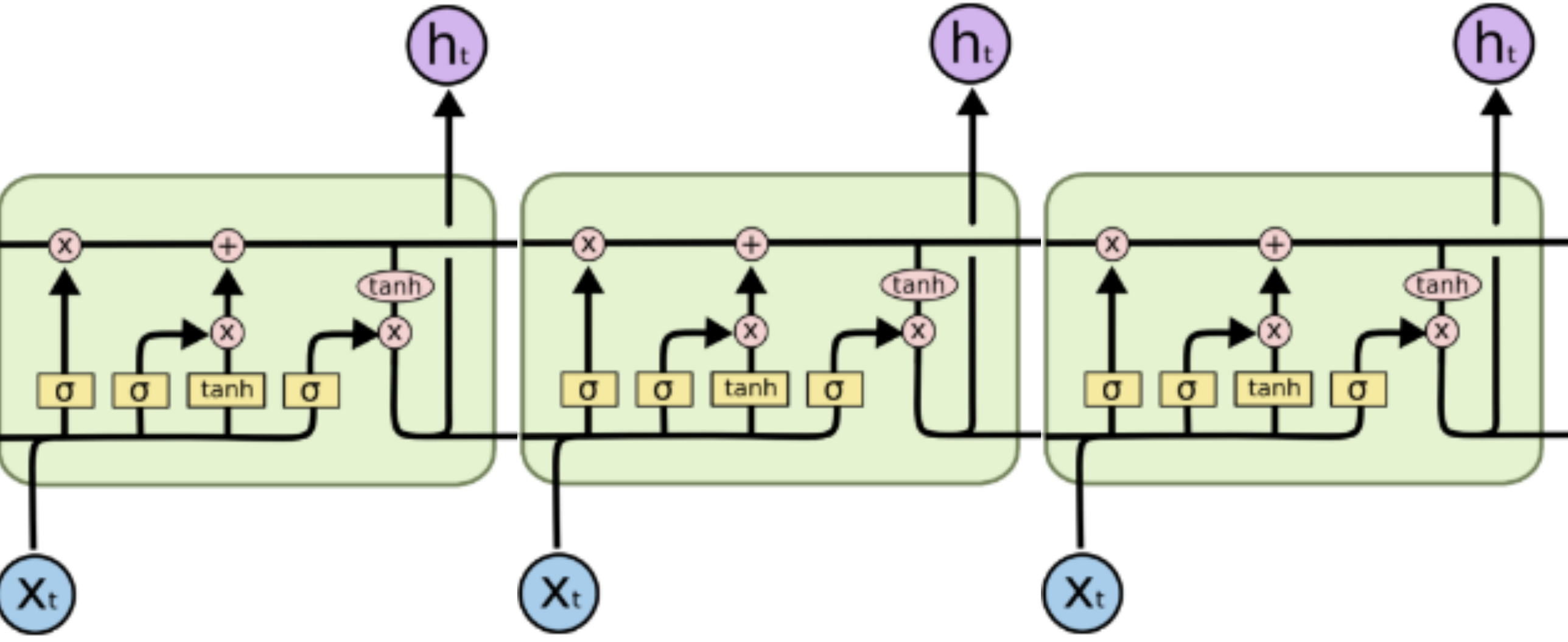


Figure credit: Christopher Olah

Another Visualization

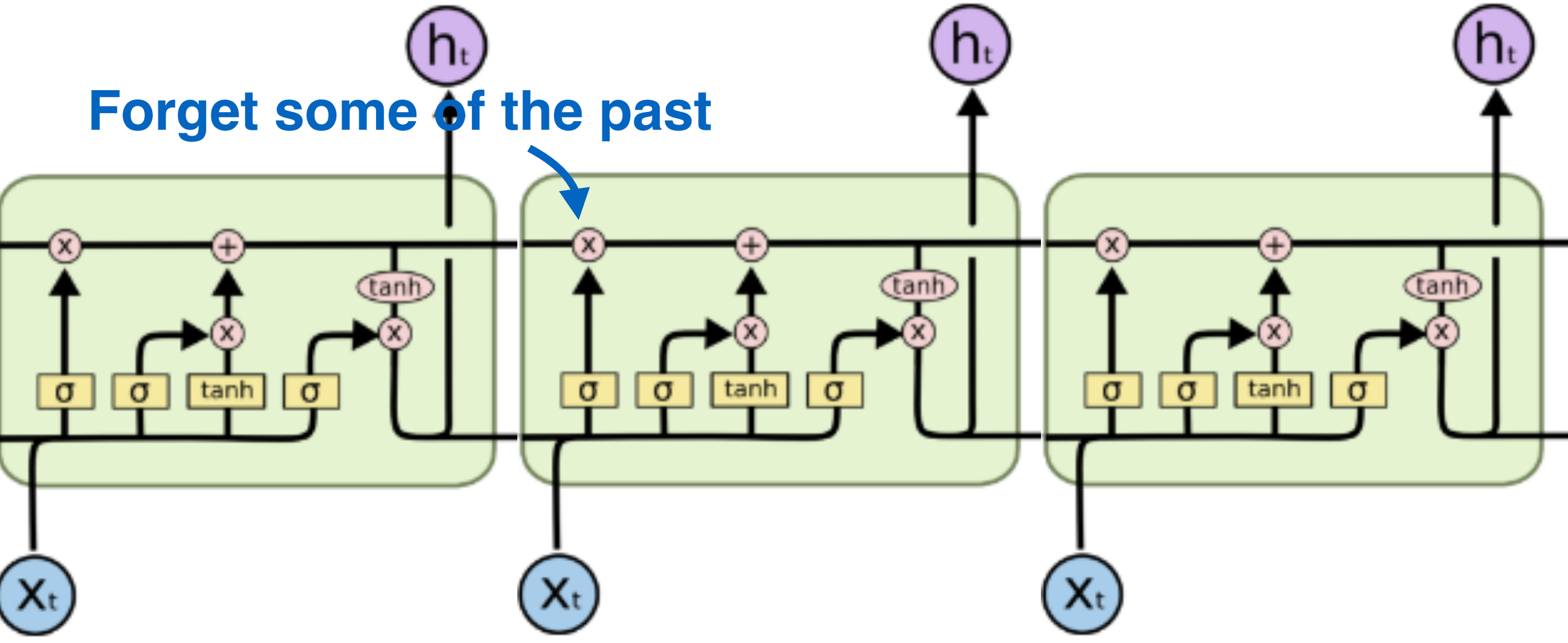


Figure credit: Christopher Olah

Another Visualization

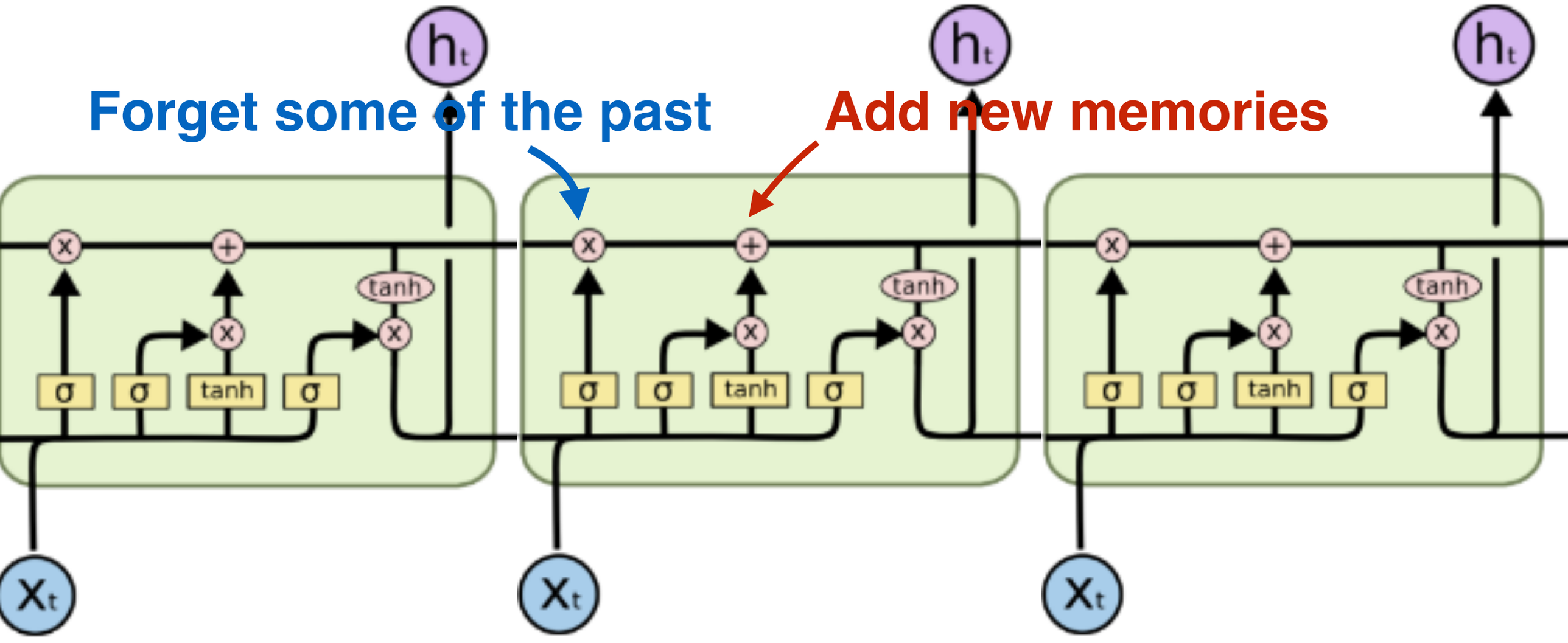


Figure credit: Christopher Olah

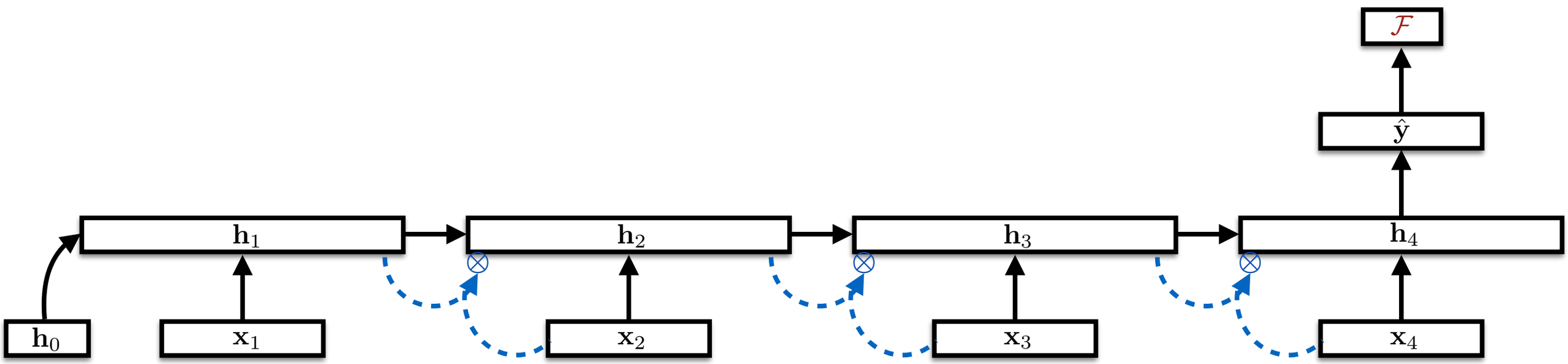
Gated Recurrent Units (GRUs)

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

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

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

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



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

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

- 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?

Review: Unconditional LMs

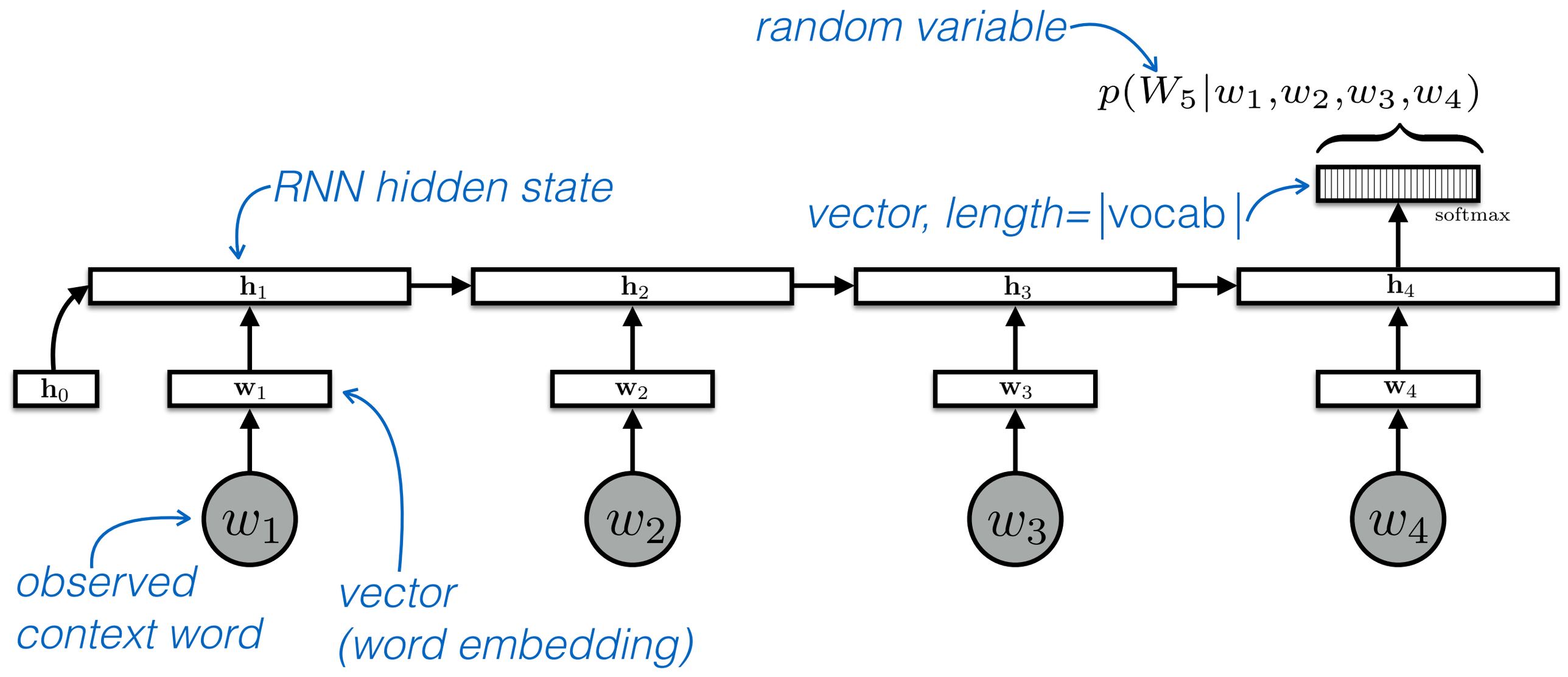
A language model assigns probabilities to sequences of words, $\boldsymbol{w} = (w_1, w_2, \ldots, w_\ell)$.

We saw that it is helpful to decompose this probability using the chain rule, as follows:

$$\begin{aligned} p(\boldsymbol{w}) &= p(w_1) \times p(w_2 \mid w_1) \times p(w_3 \mid w_1, w_2) \times \cdots \times \\ &\qquad p(w_\ell \mid w_1, \ldots, w_{\ell-1}) \\ &= \prod_{t=1}^{|\boldsymbol{w}|} p(w_t \mid w_1, \ldots, w_{t-1}) \end{aligned}$$

This reduces the language modeling problem to **modeling the probability of the next word**, given the history of preceding words.

Unconditional LMs with RNNs



Conditional LMs

A **conditional language model** assigns probabilities to sequences of words, $\boldsymbol{w} = (w_1, w_2, \ldots, w_\ell)$, given some conditioning context, $\boldsymbol{x}$.

As with unconditional models, it is again helpful to use the chain rule to decompose this probability:

$$p(\boldsymbol{w} \mid \boldsymbol{x}) = \prod_{t=1}^{\ell} p(w_t \mid \boldsymbol{x}, w_1, w_2, \ldots, w_{t-1})$$

*What is the probability of the next word, given the history of previously generated words **and** conditioning context $\boldsymbol{x}$?*

Conditional LMs

x "input"	w "**text** output"
An author	A document written by that author
A topic label	An article about that topic
{SPAM, NOT_SPAM}	An email
A sentence in French	Its English translation
A sentence in English	Its French translation
A sentence in English	Its Chinese translation
An image	A text description of the image
A document	Its summary
A document	Its translation
Meterological measurements	A weather report
Acoustic signal	Transcription of speech
Conversational history + database	Dialogue system response
A question + a document	Its answer
A question + an image	Its answer

Conditional LMs

x "input"	w "**text** output"
An author	A document written by that author
A topic label	An article about that topic
{SPAM, NOT_SPAM}	An email
A sentence in French	Its English translation
A sentence in English	Its French translation
A sentence in English	Its Chinese translation
An image	A text description of the image
A document	Its summary
A document	Its translation
Meterological measurements	A weather report
Acoustic signal	Transcription of speech
Conversational history + database	Dialogue system response
A question + a document	Its answer
A question + an image	Its answer

Conditional LMs

x "input"	w "**text** output"
An author	A document written by that author
A topic label	An article about that topic
{SPAM, NOT_SPAM}	An email
A sentence in French	Its English translation
A sentence in English	Its French translation
A sentence in English	Its Chinese translation
An image	A text description of the image
A document	Its summary
A document	Its translation
Meterological measurements	A weather report
Acoustic signal	Transcription of speech
Conversational history + database	Dialogue system response
A question + a document	Its answer
A question + an image	Its answer

this week

next week

two weeks

Data for training conditional LMs

To train conditional language models, we need *paired samples*, $\{(\boldsymbol{x}_i, \boldsymbol{w}_i)\}_{i=1}^{N}$.

Data availability varies. It's easy to think of tasks that could be solved by conditional language models, but the data just doesn't exist.

Relatively large amounts of data for:

Translation, summarisation, caption generation, speech recognition

Algorithmic challenges

We often want to find the most likely $\boldsymbol{w}$ given some $\boldsymbol{x}$. This is unfortunately generally an *intractable problem*.

$$\boldsymbol{w}^* = \arg\max_{\boldsymbol{w}} p(\boldsymbol{w} \mid \boldsymbol{x})$$

We therefore approximate it using a **beam search** or with Monte Carlo methods since $\boldsymbol{w}^{(i)} \sim p(\boldsymbol{w} \mid \boldsymbol{x})$ is often computationally easy.

Improving search/inference is an open research question.

How can we search more effectively?

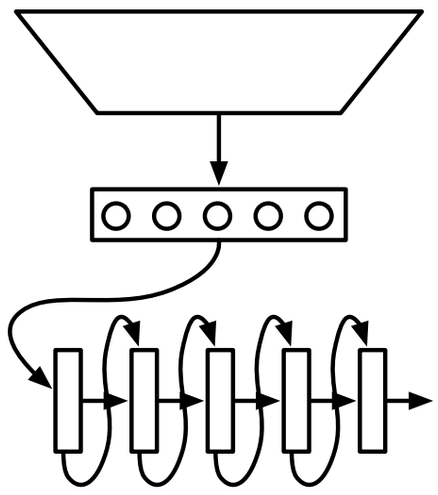
Can we get guarantees that we have found the max?

Can we limit the model a bit to make search easier?

Section overview

The rest of this section will look at “encoder-decoder” models that learn a function that maps $\boldsymbol{x}$ into a fixed-size vector and then uses a language model to “decode” that vector into a sequence of words, $\boldsymbol{w}$.

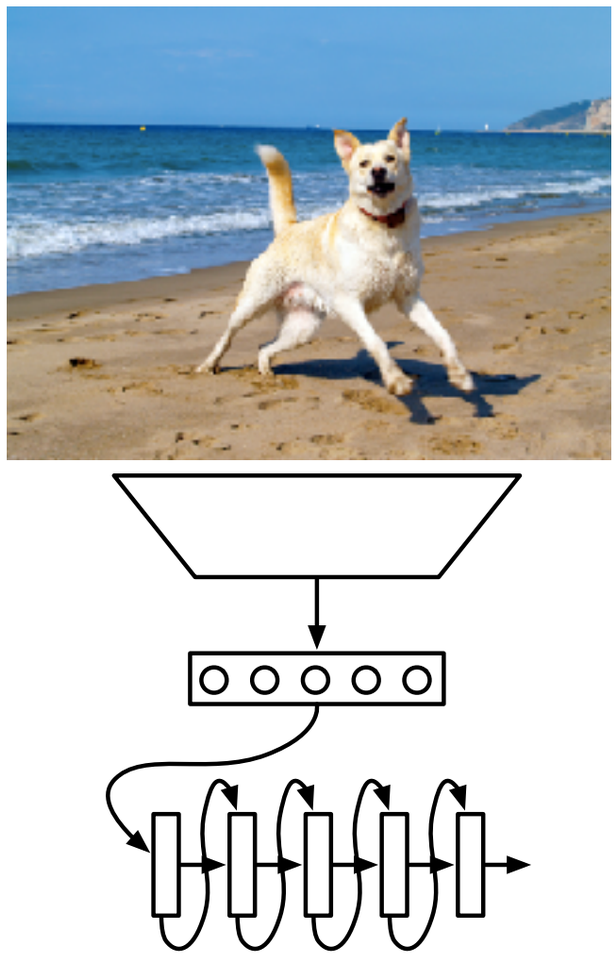
x ***Kunst kann nicht gelehrt werden…***



w ***Artistry can’t be taught…***

Section overview

The rest of this section will look at "encoder-decoder" models that learn a function that maps $\boldsymbol{x}$ into a fixed-size vector and then uses a language model to "decode" that vector into a sequence of words, $\boldsymbol{w}$.



$\boldsymbol{x}$

$\boldsymbol{w}$ ***A dog is playing on the beach.***

Section overview

- **Two questions**
  - How do we encode $\boldsymbol{x}$ as a fixed-size vector, $\mathbf{c}$?
    - Problem (or at least modality) specific
    - Think about assumptions
  - How do we condition on $\mathbf{c}$ in the decoding model?
    - Less problem specific
    - We will review solution/architectures

Kalchbrenner and Blunsom 2013

Encoder

$$\mathbf{c} = \text{embed}(\boldsymbol{x})$$

$$\mathbf{s} = \mathbf{V}\mathbf{c}$$

Recurrent decoder

Recurrent connection

Embedding of w_{t-1}

Source sentence

$$\mathbf{h}_t = g(\mathbf{W}[\mathbf{h}_{t-1}; \mathbf{w}_{t-1}] + \mathbf{s} + \mathbf{b}])$$

$$\mathbf{u}_t = \mathbf{P}\mathbf{h}_t + \mathbf{b}'$$

Learnt bias

$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{<t}) = \text{softmax}(\mathbf{u}_t)$$

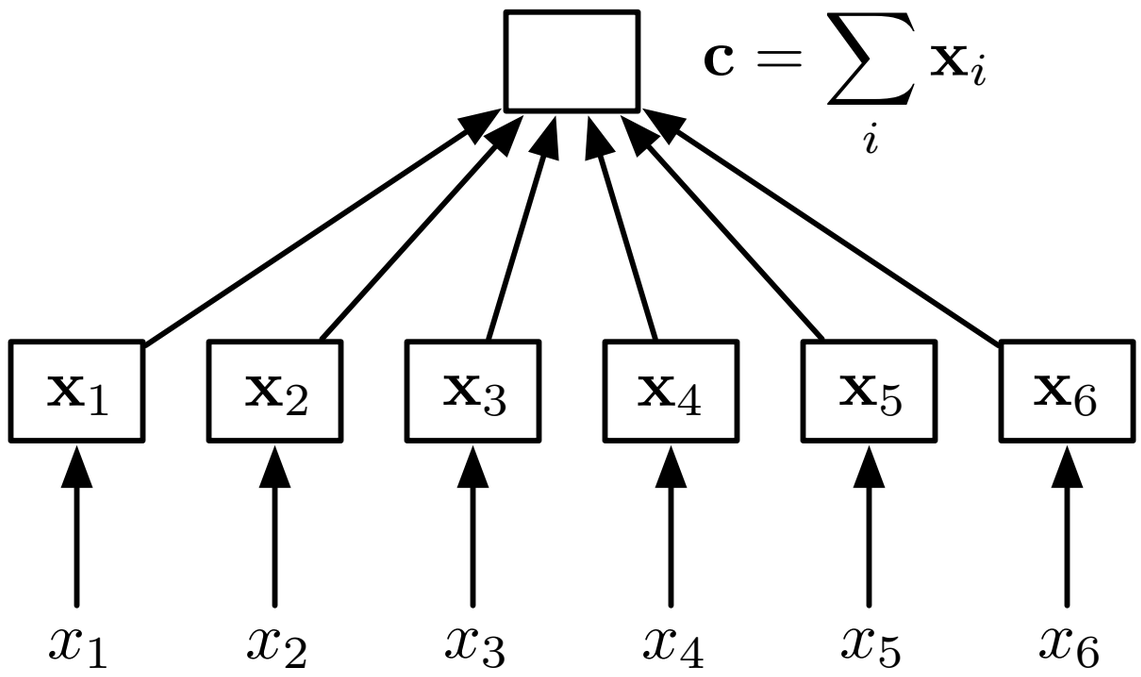
Recall unconditional RNN

$$\mathbf{h}_t = g(\mathbf{W}[\mathbf{h}_{t-1}; \mathbf{w}_{t-1}] + \mathbf{b}])$$

K&B 2013: Encoder

How should we define $\mathbf{c} = \text{embed}(\boldsymbol{x})$?

The simplest model possible:

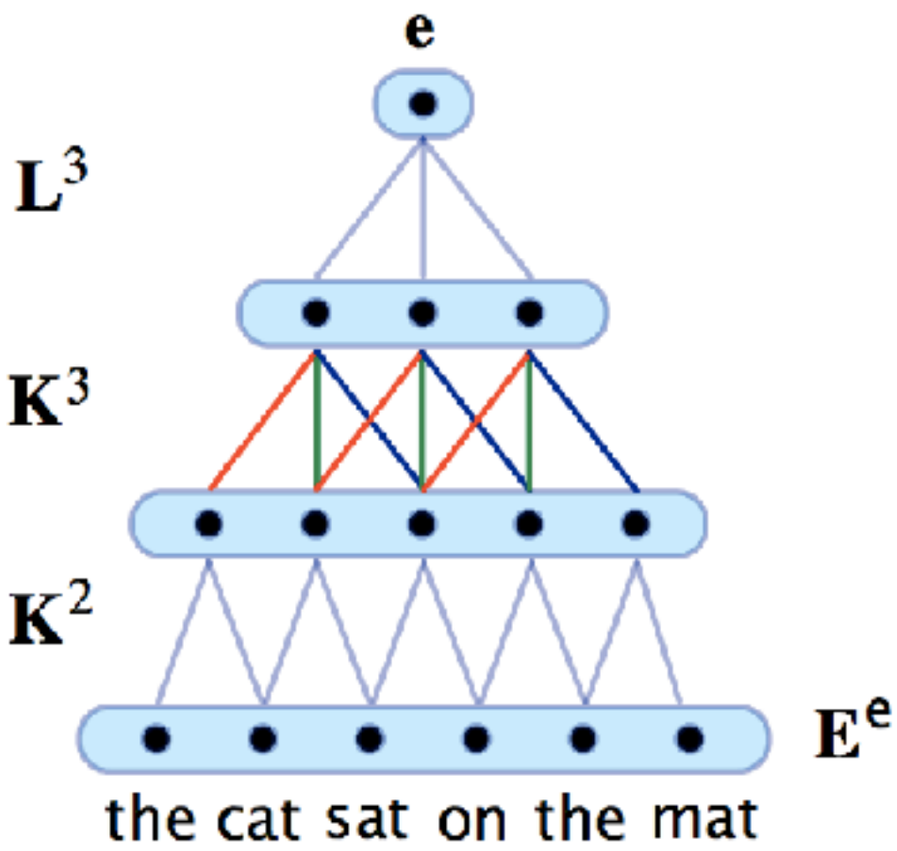


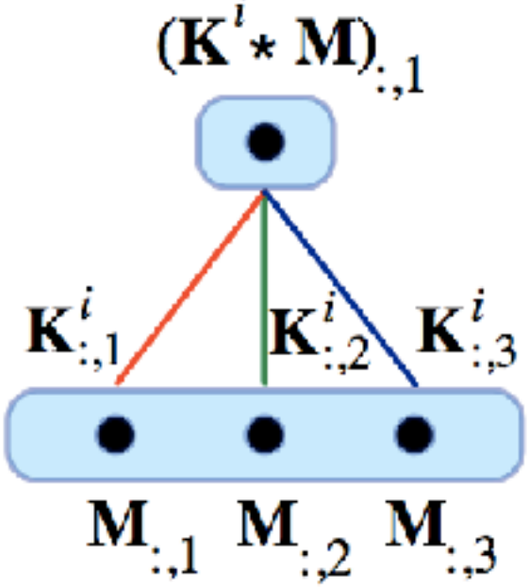
What do you think of this model?

K&B 2013: CSM Encoder

How should we define $\mathbf{c} = \mathrm{embed}(\boldsymbol{x})$?

Convolutional sentence model (CSM)





K&B 2013: CSM Encoder

- **Good**
  - Convolutions learn interactions among features in a local context
  - By stacking them, longer range dependencies can be learnt
  - Deep ConvNets have a branching structure similar to trees, but no parser is required
- **Bad**
  - Sentences have different lengths, need different depth trees; convnets are not usually so dynamic, but see*

* Kalchbrenner et al. (2014). A convolutional neural network for modelling sentences. In *Proc. ACL*.

K&B 2013: RNN Decoder

Encoder

$$\mathbf{c} = \text{embed}(\boldsymbol{x})$$

$$\mathbf{s} = \mathbf{V}\mathbf{c}$$

Recurrent decoder

Recurrent connection

Embedding of w_{t-1}

Source sentence

$$\mathbf{h}_t = g(\mathbf{W}[\mathbf{h}_{t-1}; \mathbf{w}_{t-1}] + \mathbf{s} + \mathbf{b}])$$

$$\mathbf{u}_t = \mathbf{P}\mathbf{h}_t + \mathbf{b}'$$

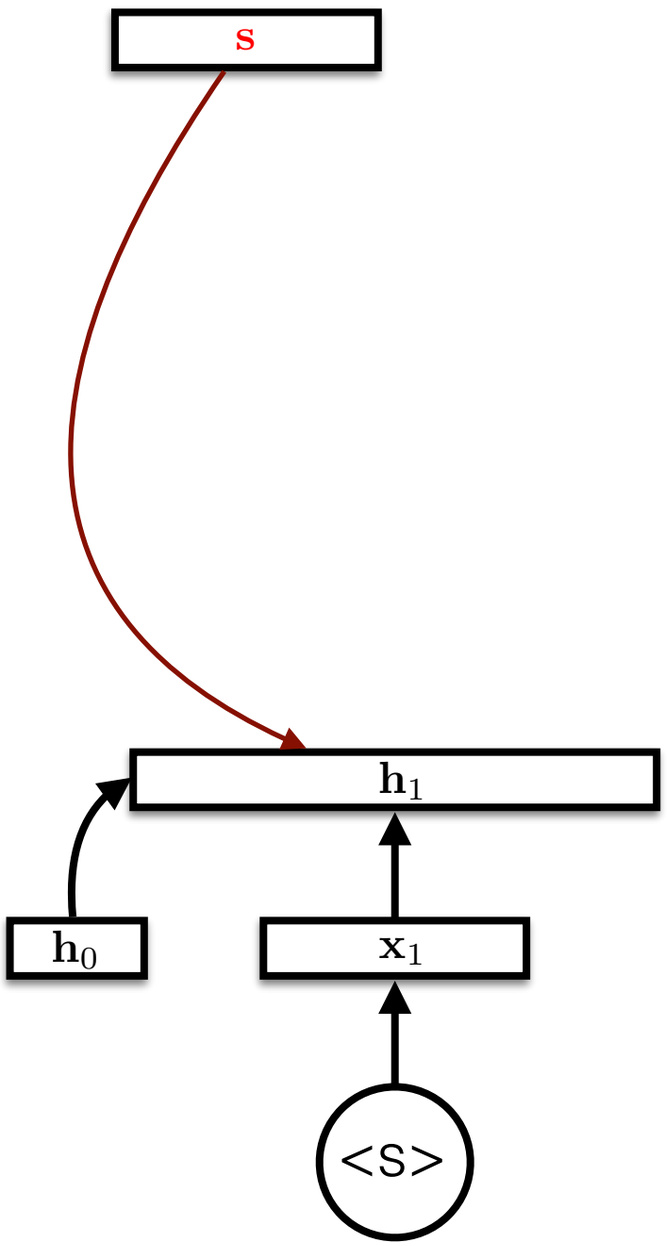
Learnt bias

$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{<t}) = \text{softmax}(\mathbf{u}_t)$$

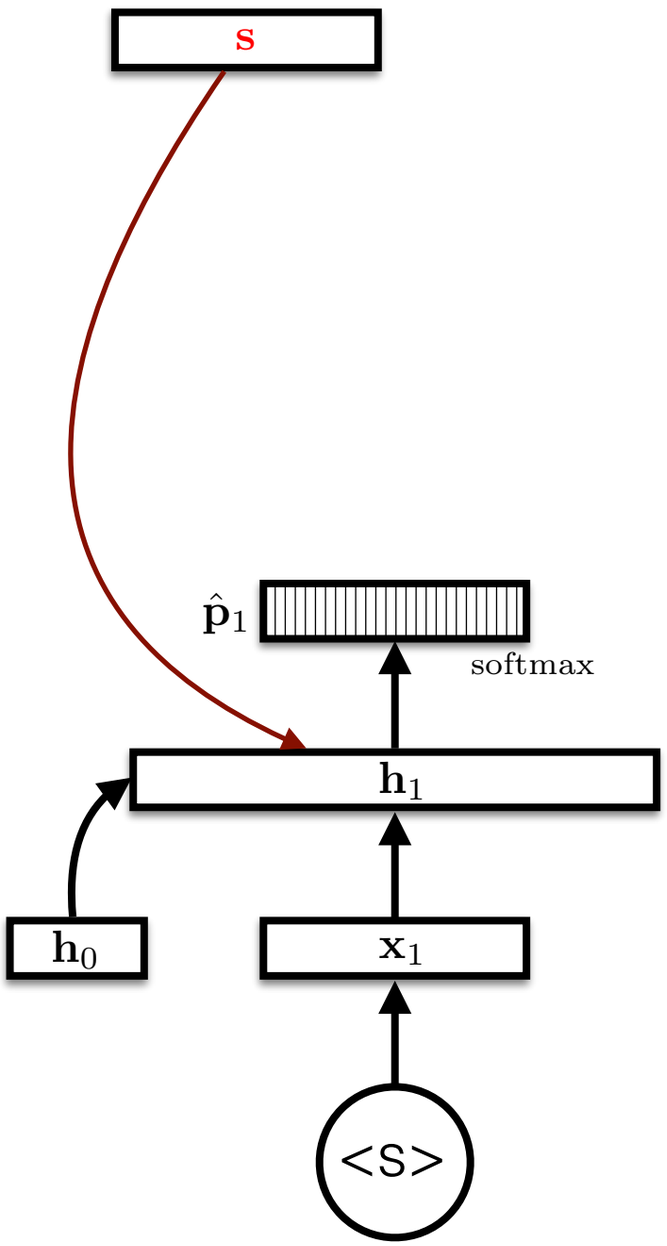
Recall unconditional RNN

$$\mathbf{h}_t = g(\mathbf{W}[\mathbf{h}_{t-1}; \mathbf{w}_{t-1}] + \mathbf{b}])$$

K&B 2013: RNN Decoder

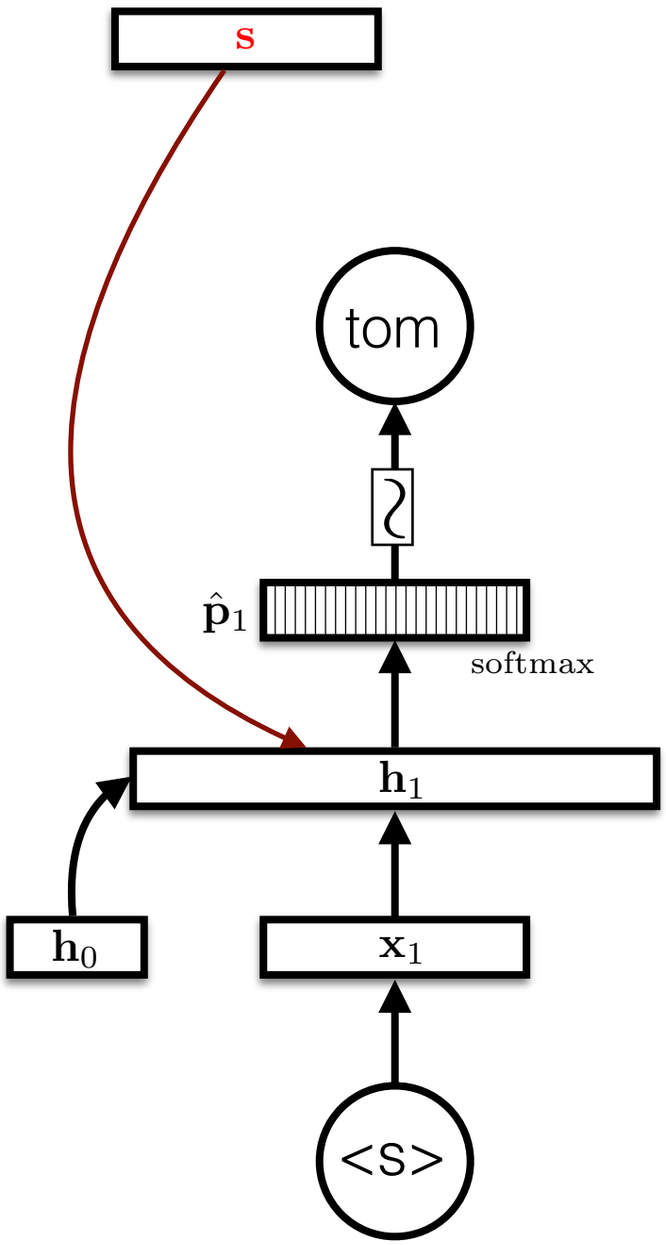


K&B 2013: RNN Decoder



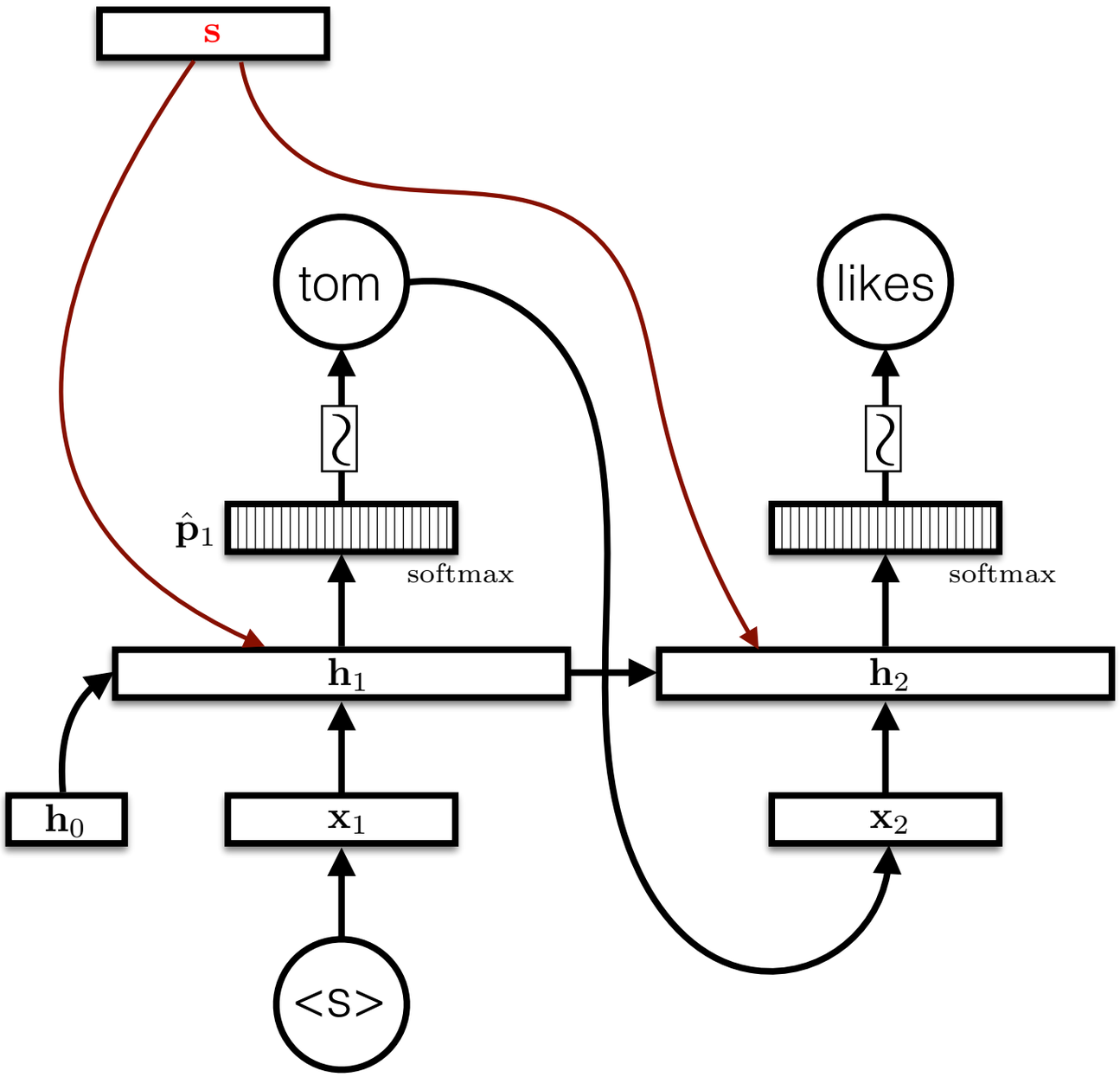
K&B 2013: RNN Decoder

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



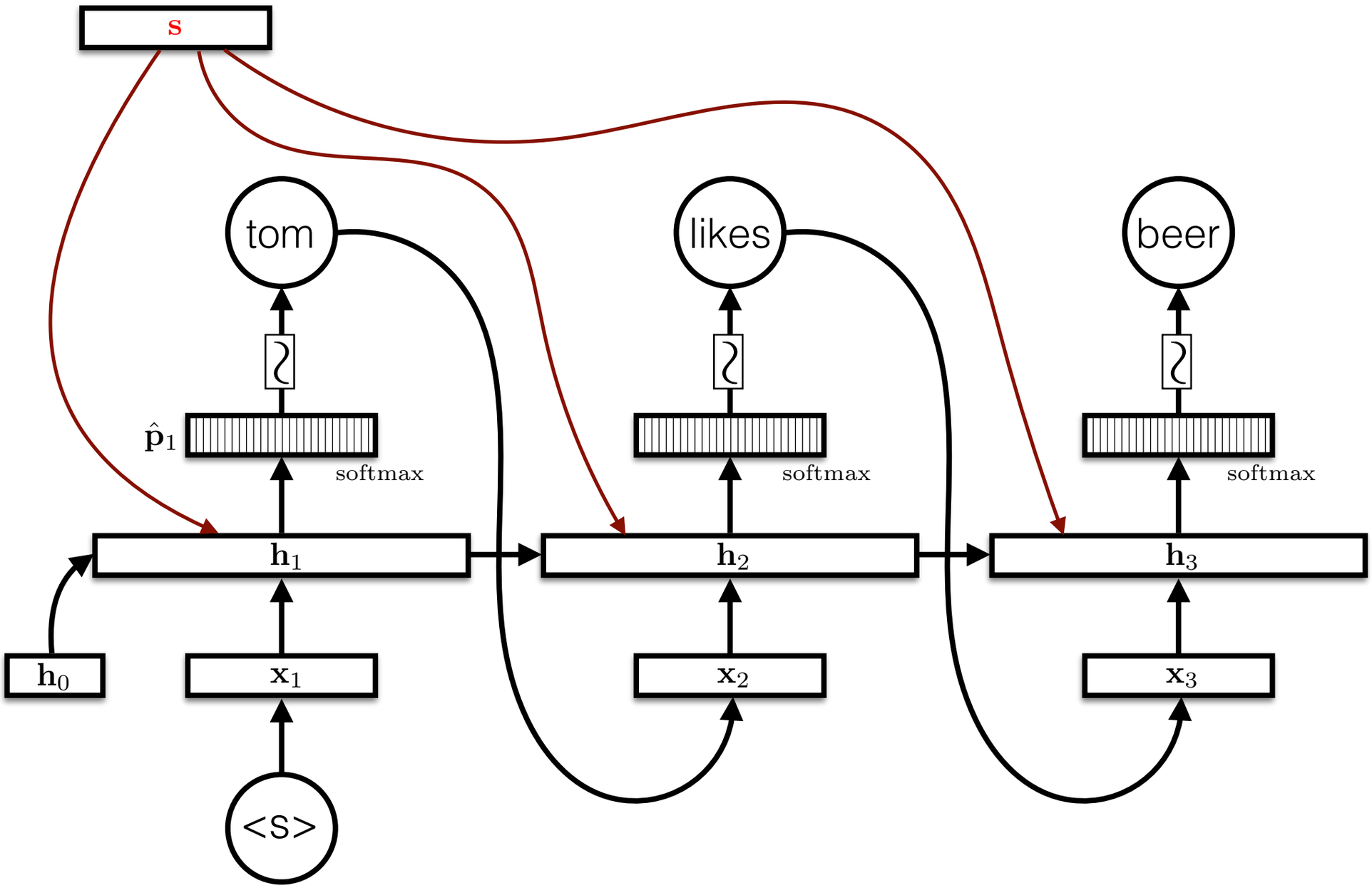
K&B 2013: RNN Decoder

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



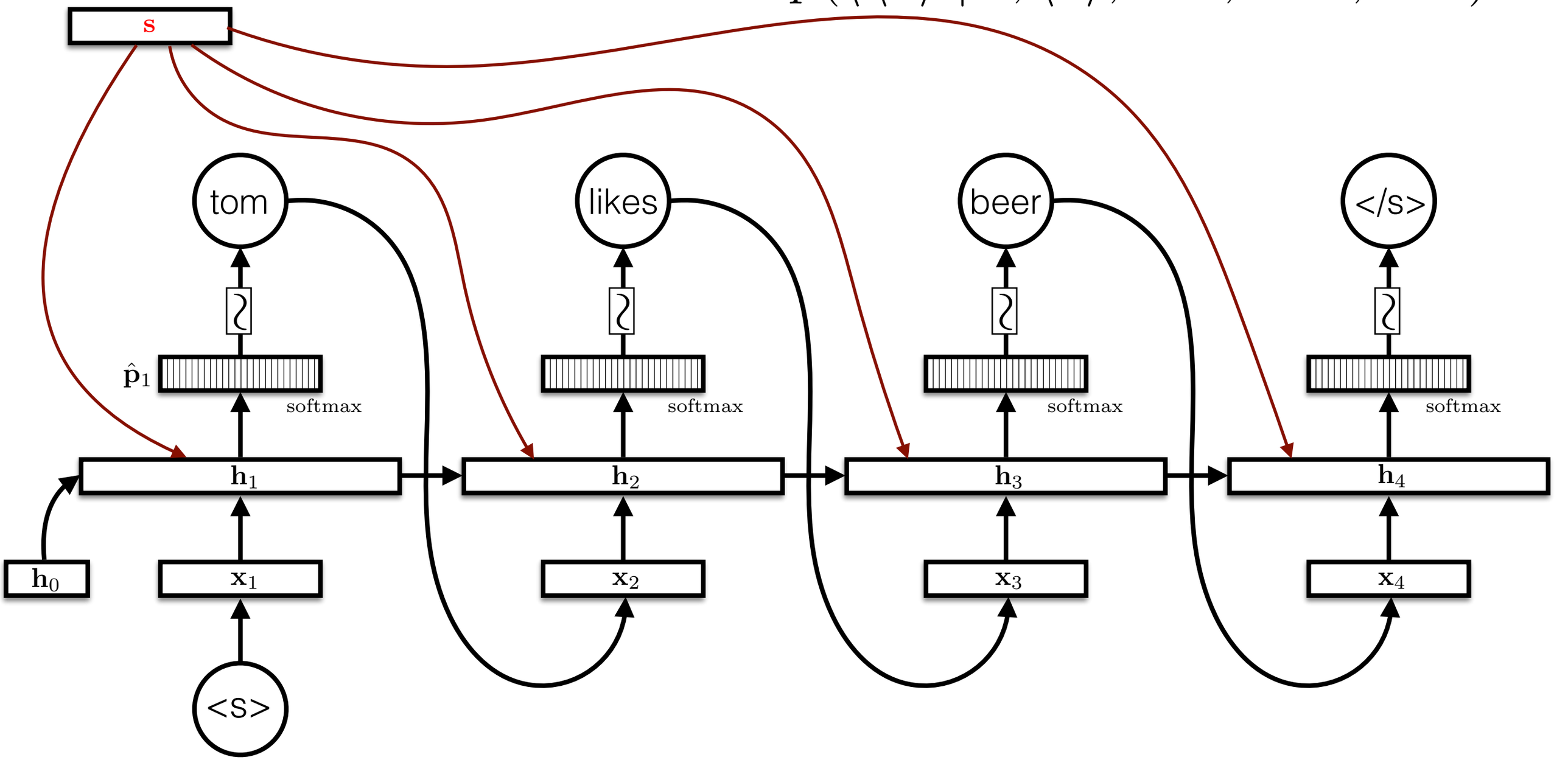
K&B 2013: RNN Decoder

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



K&B 2013: RNN Decoder

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



Sutskever et al. (2014)

LSTM encoder

$$(\mathbf{c}_0, \mathbf{h}_0) \text{ are parameters}$$

$$(\mathbf{c}_i, \mathbf{h}_i) = \mathrm{LSTM}(x_i, \mathbf{c}_{i-1}, \mathbf{h}_{i-1})$$

The encoding is $(\mathbf{c}_\ell, \mathbf{h}_\ell)$ where $\ell = |\boldsymbol{x}|$.

LSTM decoder

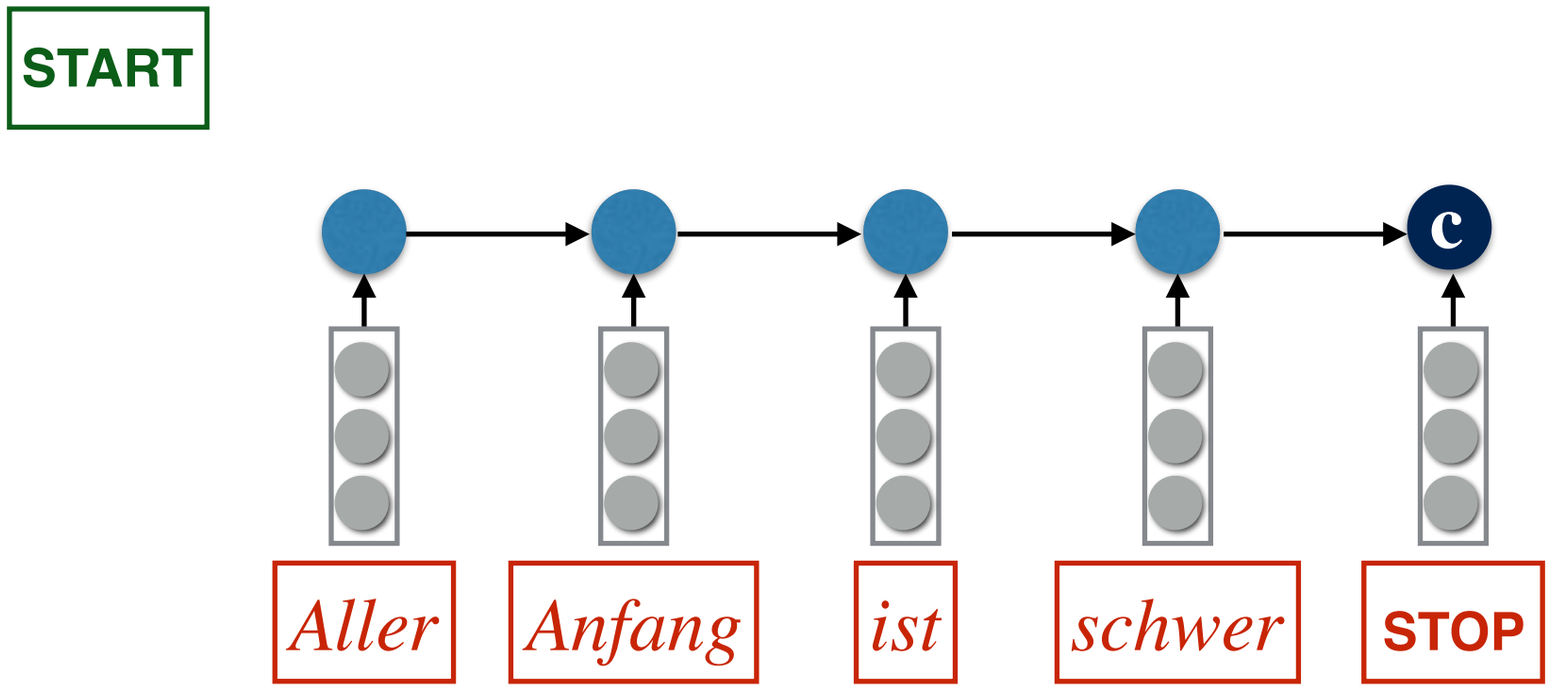
$$w_0 = \langle \mathbf{s} \rangle$$

$$(\mathbf{c}_{t+\ell}, \mathbf{h}_{t+\ell}) = \mathrm{LSTM}(w_{t-1}, \mathbf{c}_{t+\ell-1}, \mathbf{h}_{t+\ell-1})$$

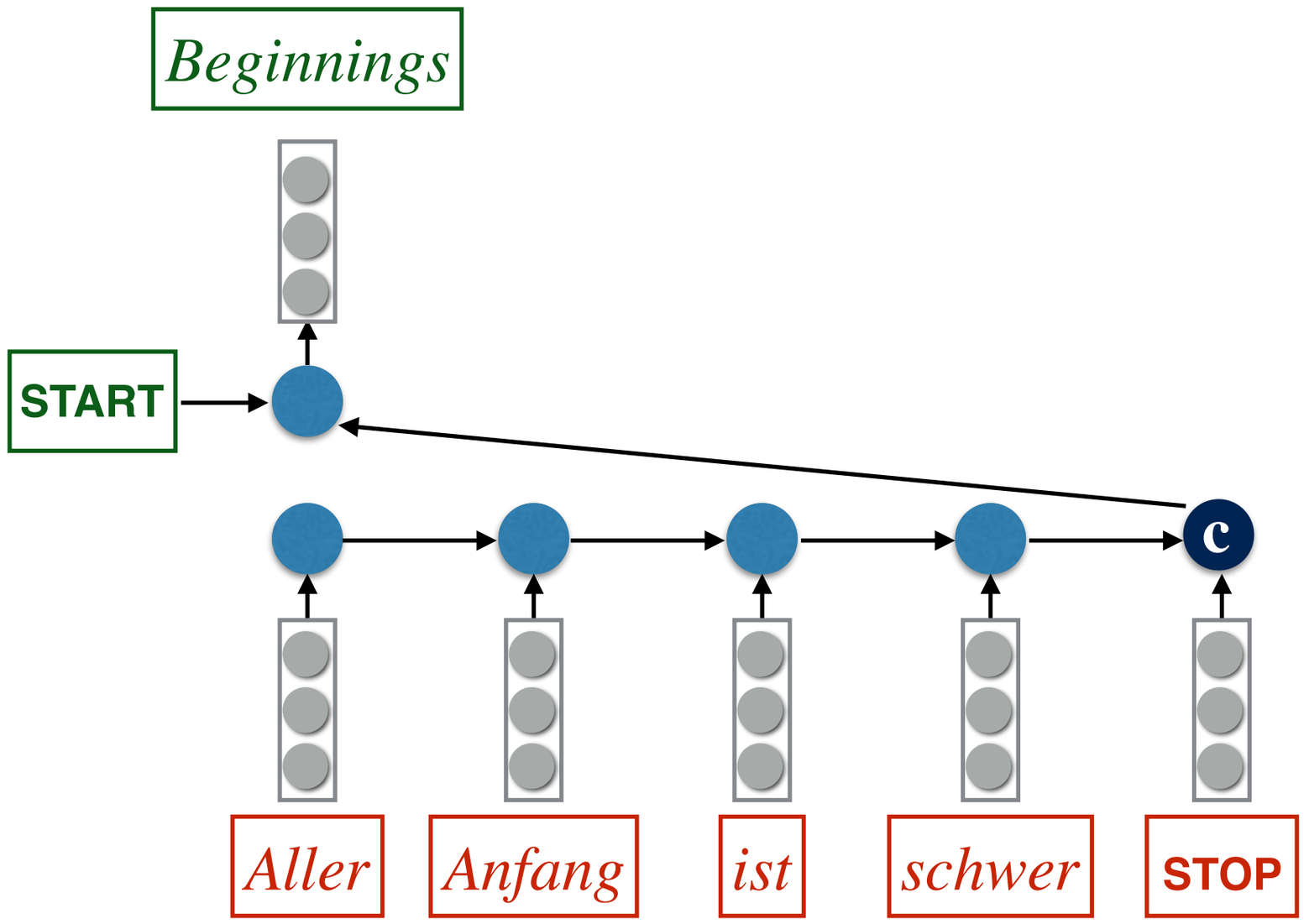
$$\mathbf{u}_t = \mathbf{P}\mathbf{h}_{t+\ell} + \mathbf{b}$$

$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{<t}) = \mathrm{softmax}(\mathbf{u}_t)$$

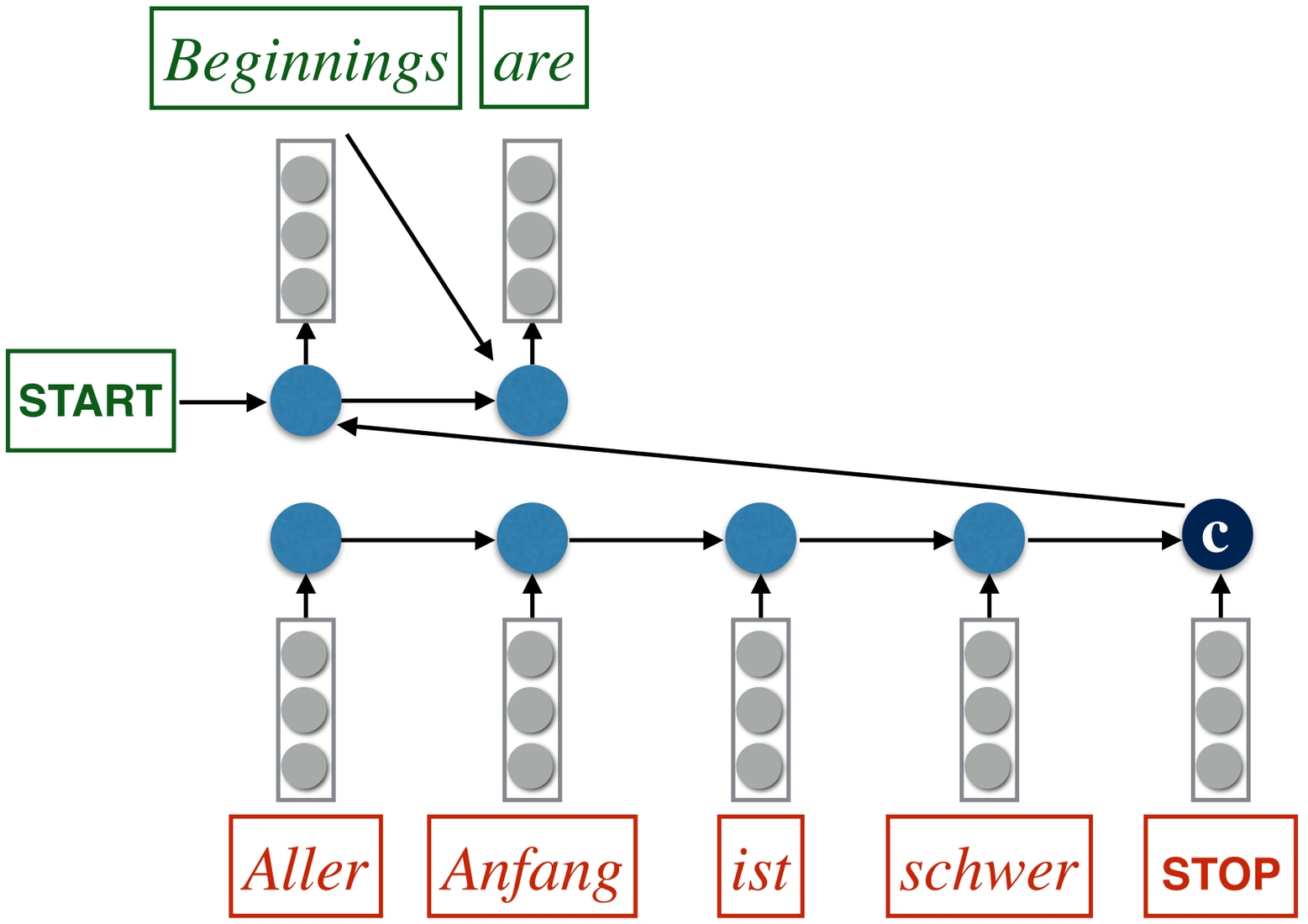
Sutskever et al. (2014)



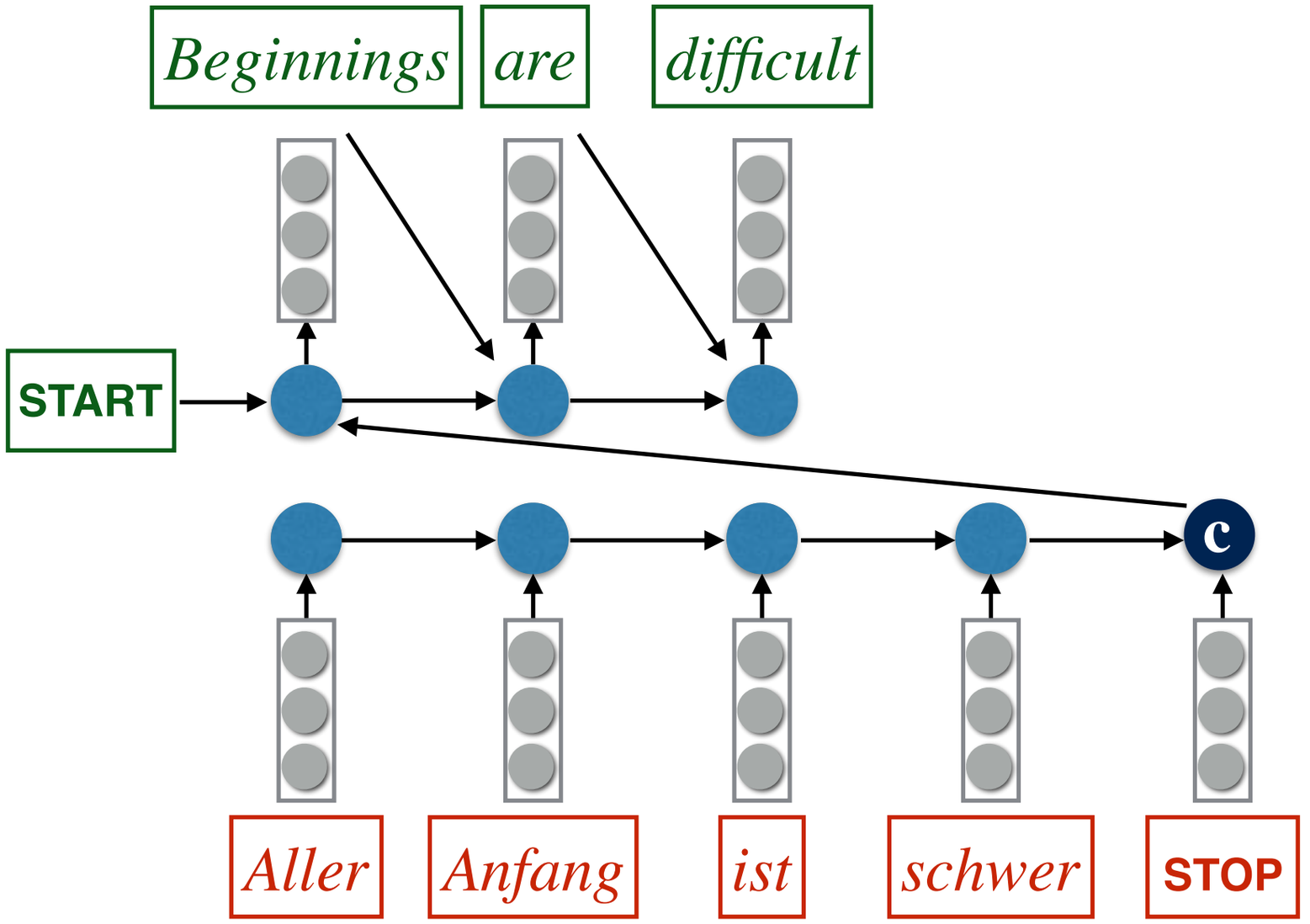
Sutskever et al. (2014)



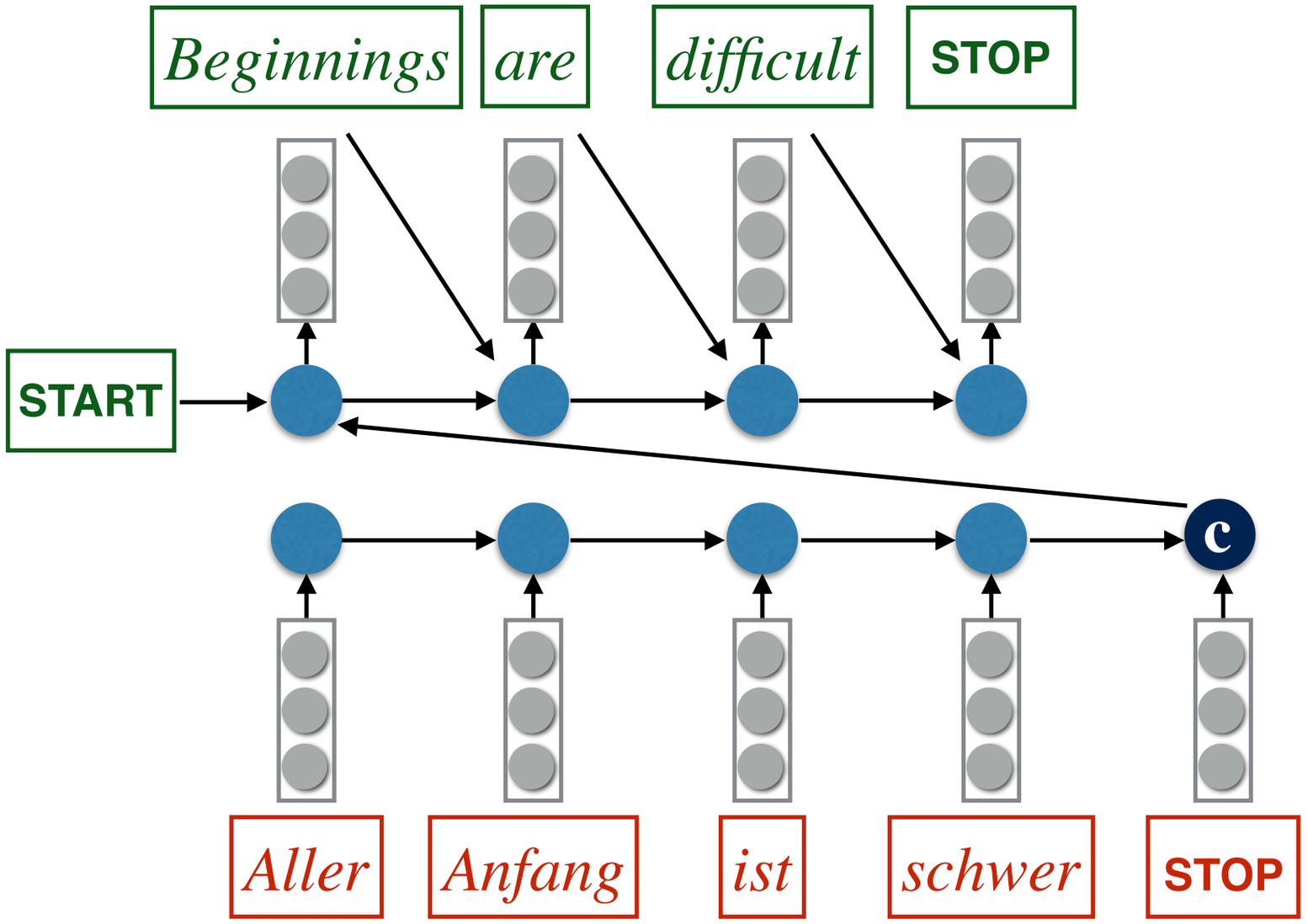
Sutskever et al. (2014)



Sutskever et al. (2014)



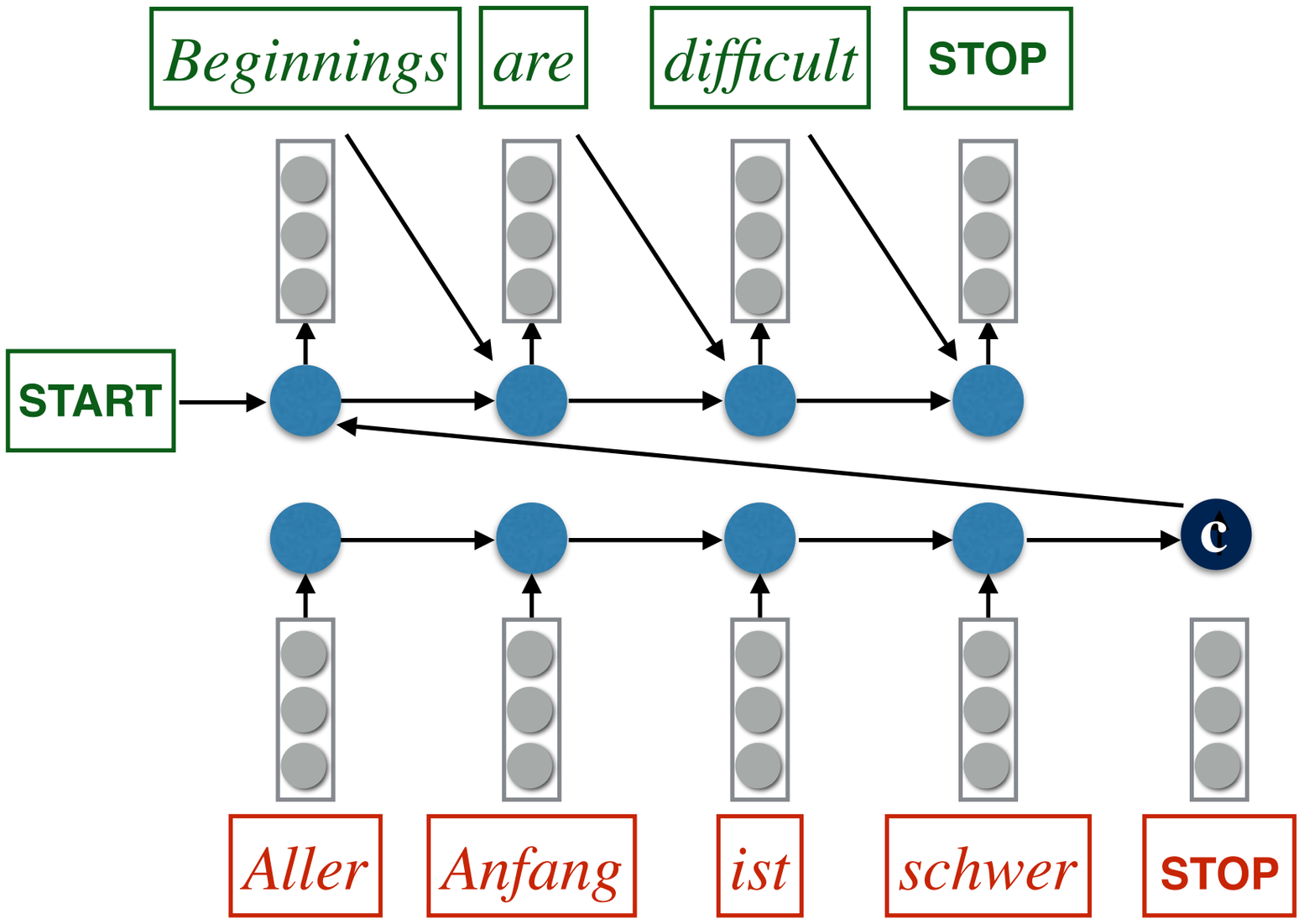
Sutskever et al. (2014)



Sutskever et al. (2014)

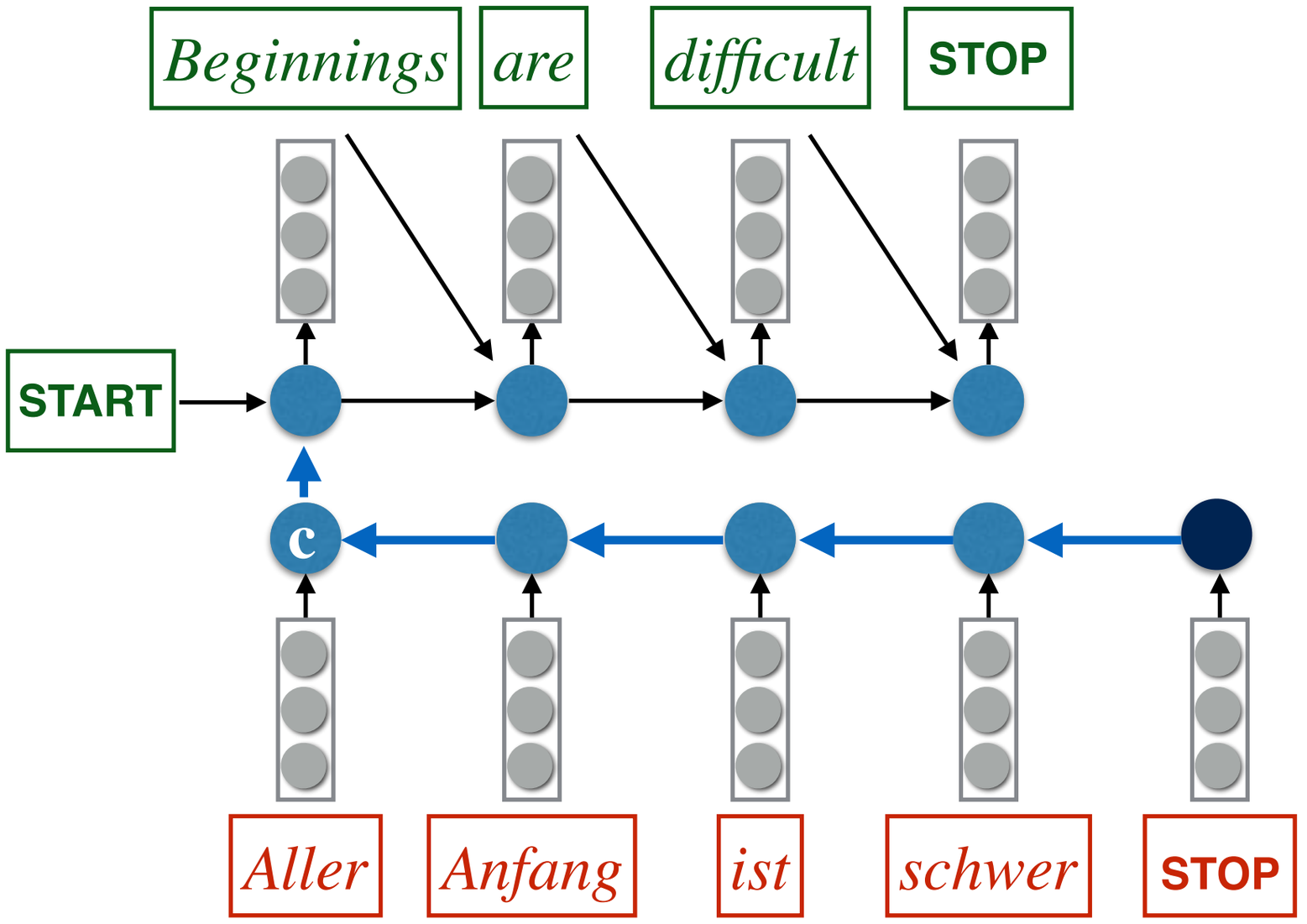
- **Good**
  - RNNs deal naturally with sequences of various lengths
  - LSTMs in principle can propagate gradients a long distance
  - Very simple architecture!
- **Bad**
  - The hidden state has to remember a lot of information!

Sutskever et al. (2014): Tricks



Sutskever et al. (2014): Tricks

Read the input sequence “backwards”: **+4 BLEU**



Sutskever et al. (2014): Tricks

Use an ensemble of J **independently trained** models.

Ensemble of 2 models: **+3 BLEU**

Ensemble of 5 models: **+4.5 BLEU**

Decoder:

$$(\mathbf{c}_{t+\ell}^{(j)}, \mathbf{h}_{t+\ell}^{(j)}) = \mathrm{LSTM}^{(j)}(w_{t-1}, \mathbf{c}_{t+\ell-1}^{(j)}, \mathbf{h}_{t+\ell-1}^{(j)})$$

$$\mathbf{u}_t^{(j)} = \mathbf{P}\mathbf{h}_t^{(j)} + \mathbf{b}^{(j)}$$

$$\mathbf{u}_t = \frac{1}{J}\sum_{j'=1}^{J} \mathbf{u}^{(j')}$$

$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{<t}) = \mathrm{softmax}(\mathbf{u}_t)$$

A word about decoding

In general, we want to find the most probable (MAP) output given the input, i.e.

$$\begin{aligned}\boldsymbol{w}^* &= \arg\max_{\boldsymbol{w}} p(\boldsymbol{w} \mid \boldsymbol{x}) \\ &= \arg\max_{\boldsymbol{w}} \sum_{t=1}^{|\boldsymbol{w}|} \log p(w_t \mid \boldsymbol{x}, \boldsymbol{w}_{<t})\end{aligned}$$

A word about decoding

In general, we want to find the most probable (MAP) output given the input, i.e.

$$\begin{aligned}\boldsymbol{w}^* &= \arg\max_{\boldsymbol{w}} p(\boldsymbol{w} \mid \boldsymbol{x}) \\ &= \arg\max_{\boldsymbol{w}} \sum_{t=1}^{|\boldsymbol{w}|} \log p(w_t \mid \boldsymbol{x}, \boldsymbol{w}_{<t})\end{aligned}$$

This is, for general RNNs, a hard problem. We therefore approximate it with a **greedy search**:

$$\begin{aligned}w_1^* &= \arg\max_{w_1} p(w_1 \mid \boldsymbol{x}) \\ w_2^* &= \arg\max_{w_2} p(w_2 \mid \boldsymbol{x}, w_1^*) \\ &\vdots \\ w_t^* &= \arg\max_{w_2} p(w_t \mid \boldsymbol{x}, \boldsymbol{w}_{<t}^*)\end{aligned}$$

A word about decoding

In general, we want to find the most probable (MAP) output given the input, i.e.

$$\boldsymbol{w}^* = \arg\max_{\boldsymbol{w}} p(\boldsymbol{w} \mid \boldsymbol{x})$$

$$= \arg\max_{\boldsymbol{w}} \sum_{t=1}^{|\boldsymbol{w}|} \log p(w_t \mid \boldsymbol{x}, \boldsymbol{w}_{<t})$$

undecidable :(

This is, for general RNNs, a ~~hard~~ problem. We therefore approximate it with a **greedy search**:

$$w_1^* = \arg\max_{w_1} p(w_1 \mid \boldsymbol{x})$$

$$w_2^* = \arg\max_{w_2} p(w_2 \mid \boldsymbol{x}, w_1^*)$$

$$\vdots$$

$$w_t^* = \arg\max_{w_2} p(w_t \mid \boldsymbol{x}, \boldsymbol{w}_{<t}^*)$$

A word about decoding

A slightly better approximation is to use a **beam search** with beam size *b*. Key idea: keep track of top b hypothesis.

E.g., for *b*=2:

$\boldsymbol{x} = Bier\ trinke\ ich$

beer drink I

$\langle s \rangle$
logprob=0

w_0 w_1 w_2 w_3

A word about decoding

A slightly better approximation is to use a **beam search** with beam size *b*. Key idea: keep track of top b hypothesis.

E.g., for *b*=2:

$\boldsymbol{x} = Bier\ trinke\ ich$

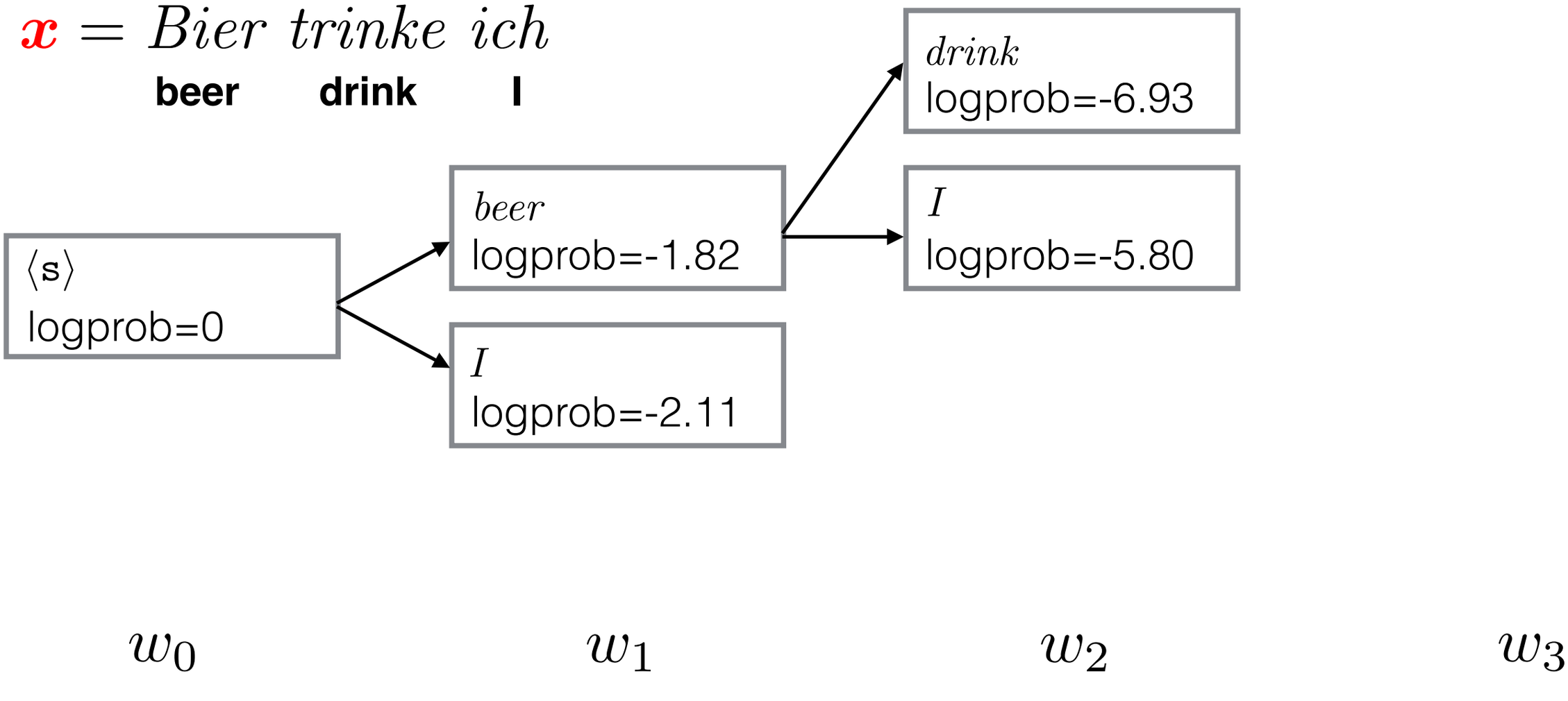
beer **drink** **I**

w_0 w_1 w_2 w_3

A word about decoding

A slightly better approximation is to use a **beam search** with beam size *b*. Key idea: keep track of top b hypothesis.

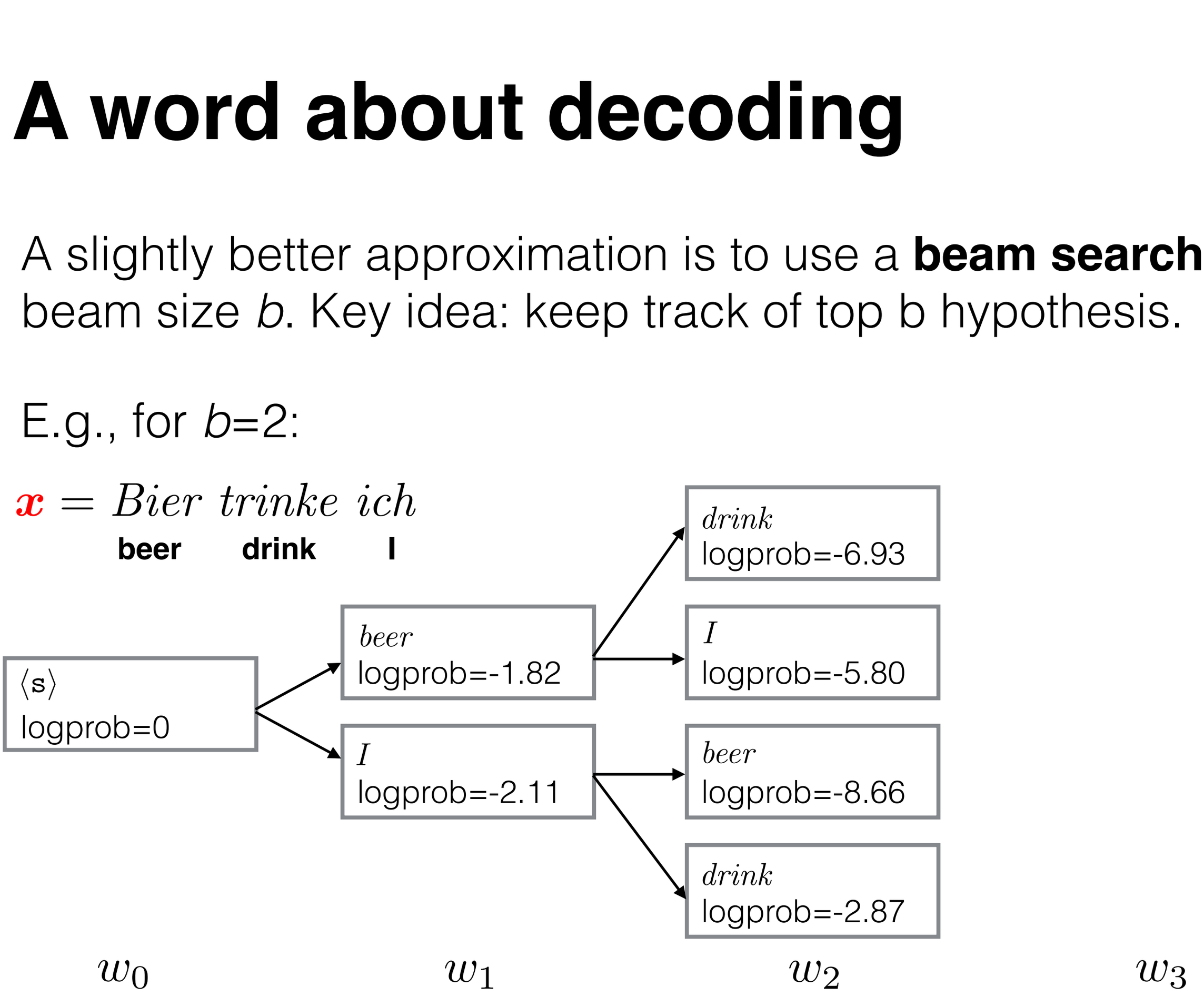
E.g., for *b*=2:



A word about decoding

A slightly better approximation is to use a **beam search** with beam size *b*. Key idea: keep track of top b hypothesis.

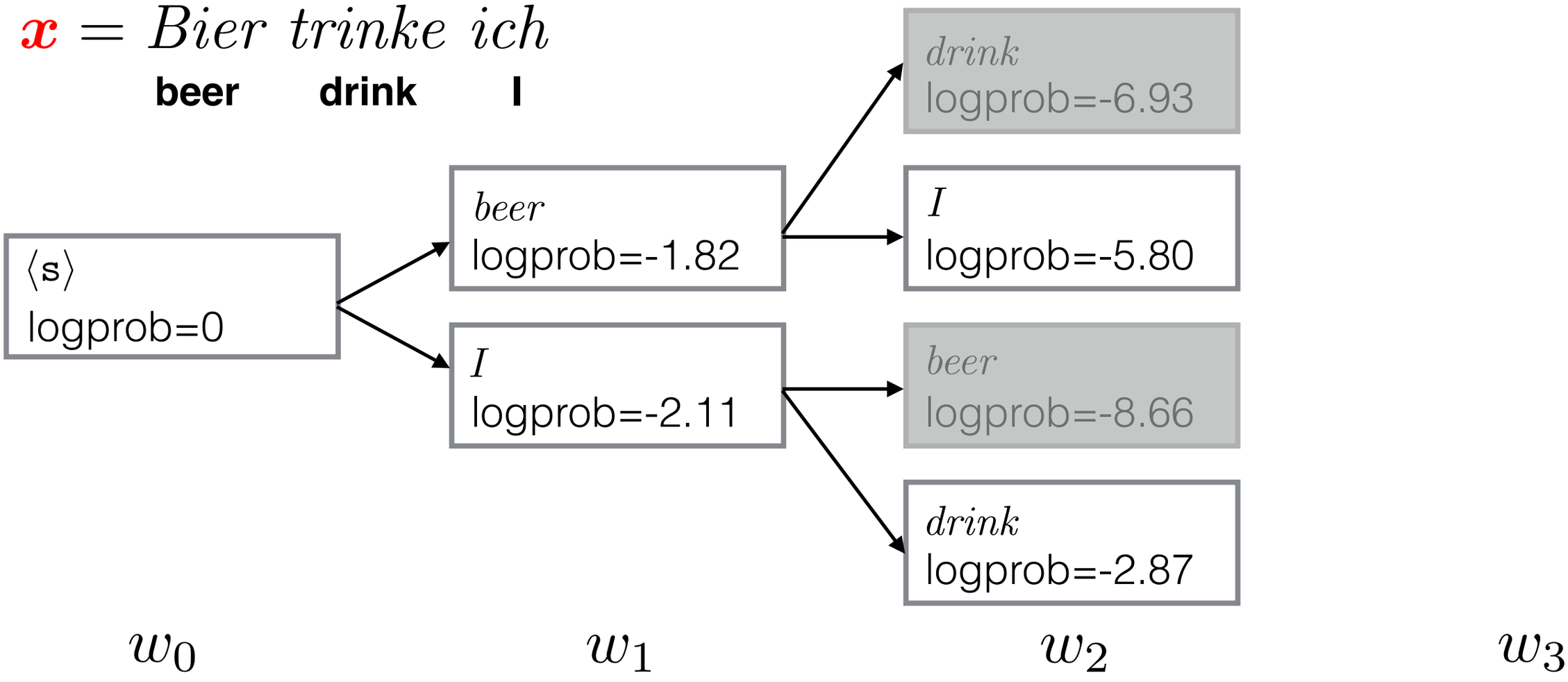
E.g., for *b*=2:



A word about decoding

A slightly better approximation is to use a **beam search** with beam size *b*. Key idea: keep track of top b hypothesis.

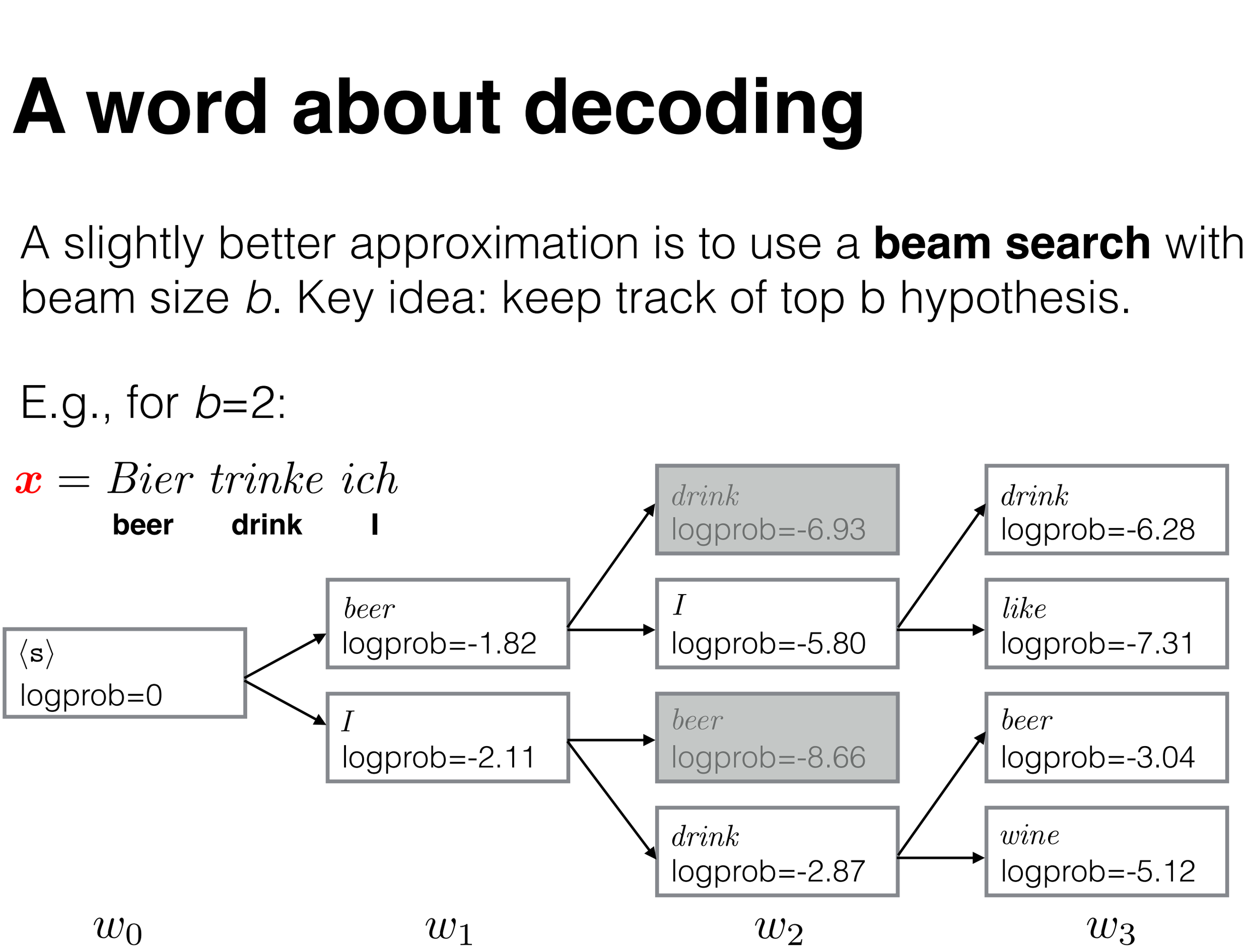
E.g., for *b*=2:



A word about decoding

A slightly better approximation is to use a **beam search** with beam size *b*. Key idea: keep track of top b hypothesis.

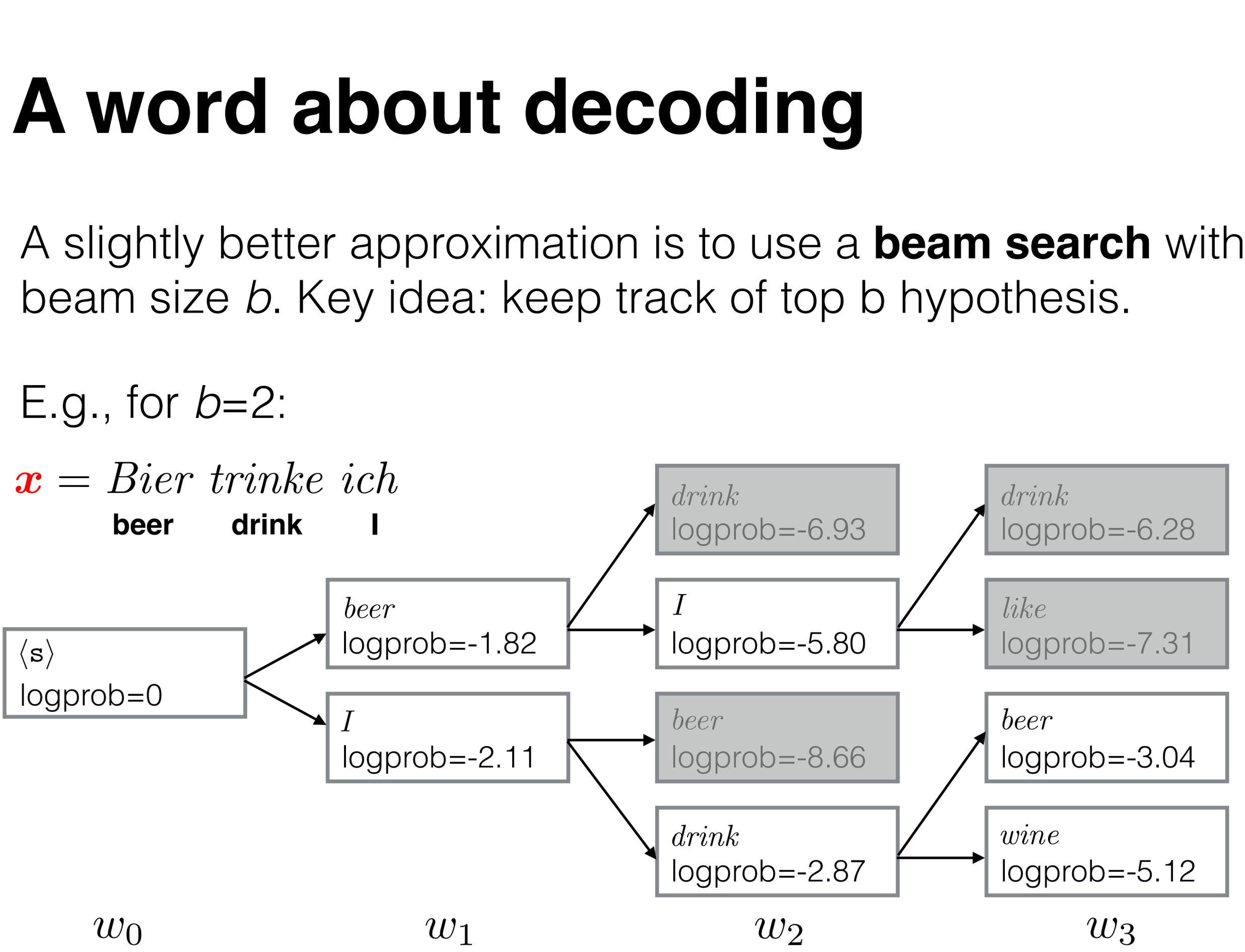
E.g., for *b*=2:



A word about decoding

A slightly better approximation is to use a **beam search** with beam size *b*. Key idea: keep track of top b hypothesis.

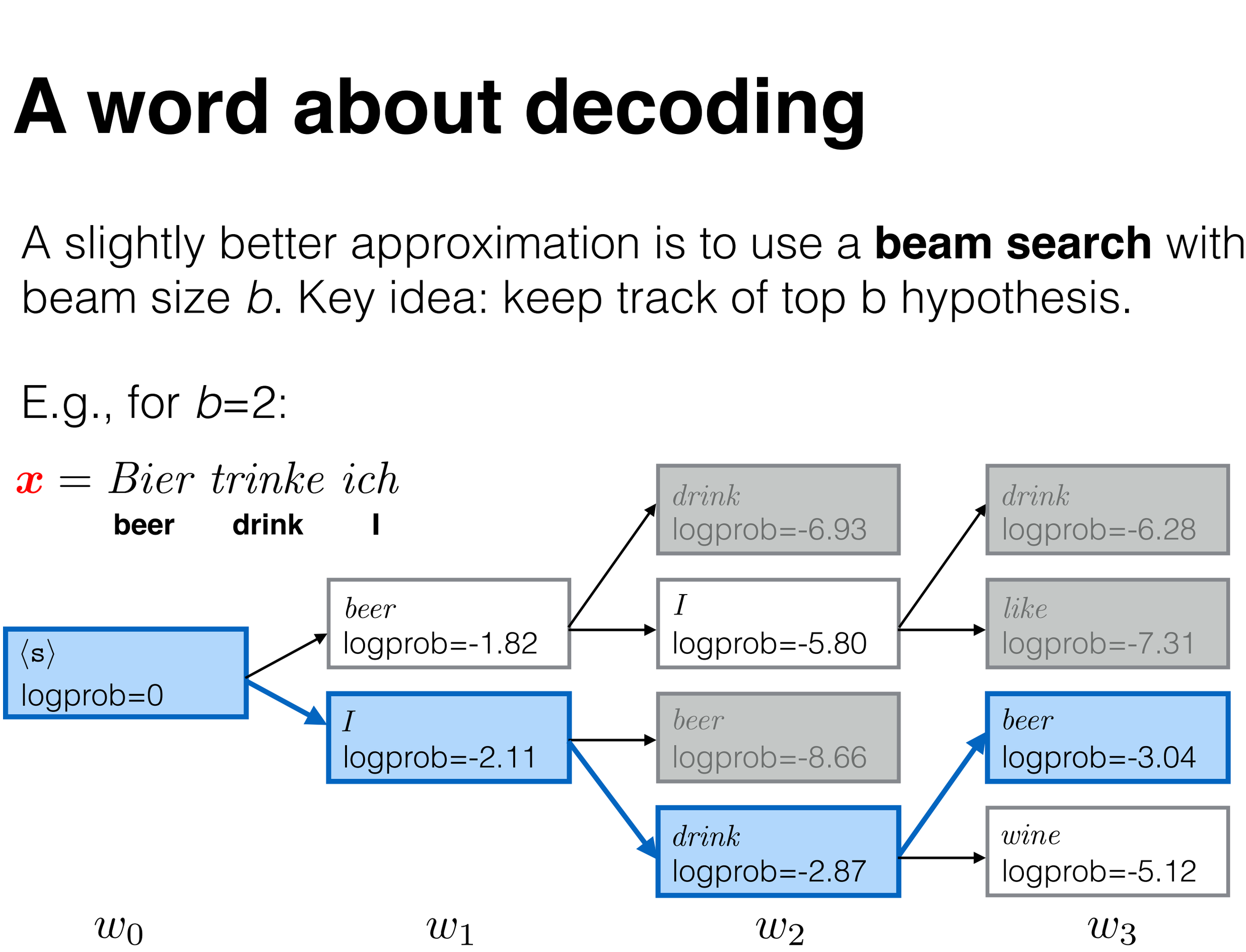
E.g., for *b*=2:



A word about decoding

A slightly better approximation is to use a **beam search** with beam size *b*. Key idea: keep track of top b hypothesis.

E.g., for *b*=2:



Sutskever et al. (2014): Tricks

Use beam search: **+1 BLEU**

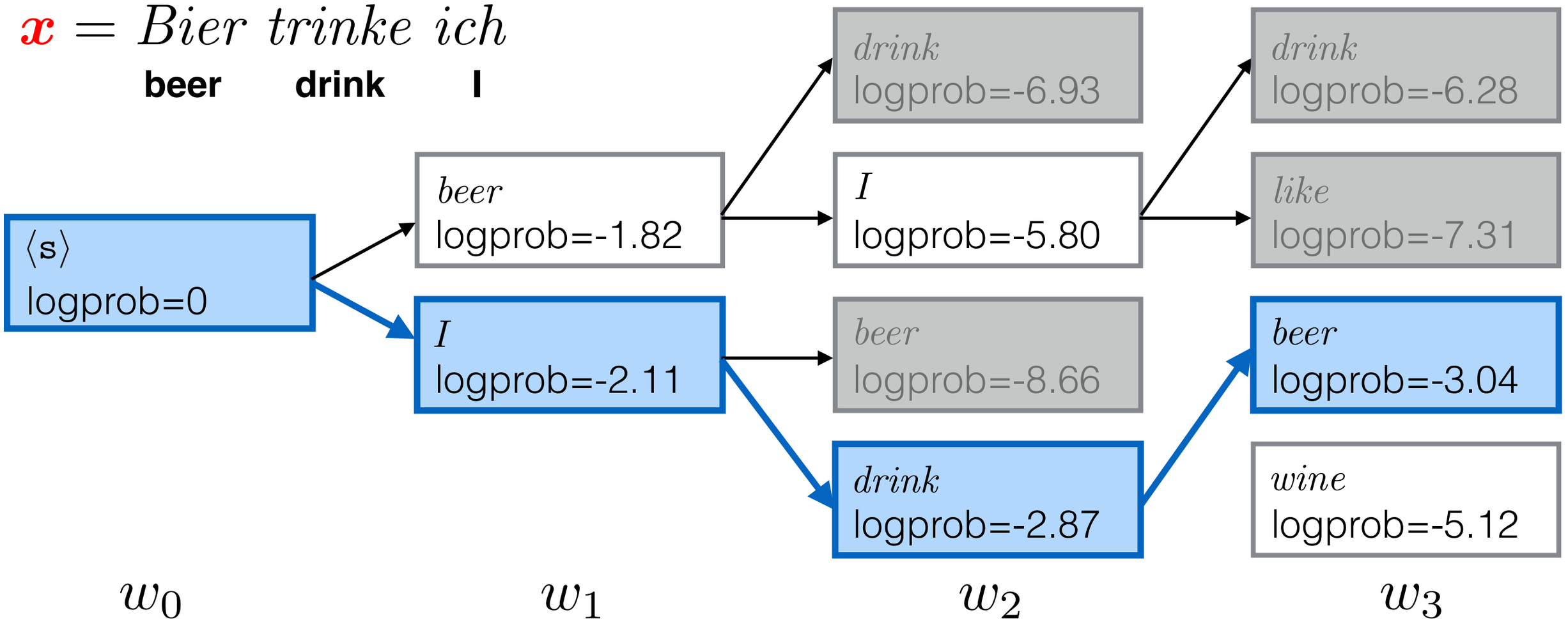


Image caption generation

- Neural networks are great for working with multiple modalities—**everything is a vector**!
- Image caption generation can therefore use the same techniques as translation modeling
- A word about data
  - Relatively few captioned images are available
  - Pre-train image embedding model using another task, like image identification (e.g., ImageNet)

Kiros et al. (2013)

- Looks a lot like Kalchbrenner and Blunsom (2013)
  - convolutional network on the input
  - n-gram language model on the output
- Innovation: **multiplicative interactions** in the decoder n-gram model

Kiros et al. (2013)

Encoder $\mathbf{x} = \text{embed}(\boldsymbol{x})$

Kiros et al. (2013)

Encoder $\mathbf{x} = \text{embed}(\boldsymbol{x})$

Unconditional *n*-gram LM:

Embedding of w_{t-1}

$$\mathbf{h}_t = \mathbf{W}[\mathbf{w}_{t-n+1}; \mathbf{w}_{t-n+2}; \ldots; \mathbf{w}_{t-1}]$$

$$\mathbf{u}_t = \mathbf{P}\mathbf{h}_t + \mathbf{b}$$

$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{t-n+1}^{t-1}) = \text{softmax}(\mathbf{u}_t)$$

Kiros et al. (2013)

Encoder $\mathbf{x} = \mathrm{embed}(\boldsymbol{x})$

Simple conditional *n*-gram LM:

$$\mathbf{h}_t = \mathbf{W}[\mathbf{w}_{t-n+1}; \mathbf{w}_{t-n+2}; \ldots; \mathbf{w}_{t-1}] + \mathbf{C}\mathbf{x}$$

$$\mathbf{u}_t = \mathbf{P}\mathbf{h}_t + \mathbf{b}$$

$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{t-n+1}^{t-1}) = \mathrm{softmax}(\mathbf{u}_t)$$

Kiros et al. (2013)

Encoder $\mathbf{x} = \mathrm{embed}(\boldsymbol{x})$

Simple conditional *n*-gram LM:

$$\mathbf{h}_t = \mathbf{W}[\mathbf{w}_{t-n+1}; \mathbf{w}_{t-n+2}; \ldots; \mathbf{w}_{t-1}] + \mathbf{C}\mathbf{x}$$

$$\mathbf{u}_t = \mathbf{P}\mathbf{h}_t + \mathbf{b}$$

$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{t-n+1}^{t-1}) = \mathrm{softmax}(\mathbf{u}_t)$$

Multiplicative *n*-gram LM:

$$w_i = r_{i,w}$$

Kiros et al. (2013)

Encoder $\mathbf{x} = \mathrm{embed}(\boldsymbol{x})$

Simple conditional *n*-gram LM:

$$\mathbf{h}_t = \mathbf{W}[\mathbf{w}_{t-n+1}; \mathbf{w}_{t-n+2}; \ldots; \mathbf{w}_{t-1}] + \mathbf{C}\mathbf{x}$$

$$\mathbf{u}_t = \mathbf{P}\mathbf{h}_t + \mathbf{b}$$

$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{t-n+1}^{t-1}) = \mathrm{softmax}(\mathbf{u}_t)$$

Multiplicative *n*-gram LM:

~~$w_i = r_{i,w}$~~

$$w_i = r_{i,j,w} x_j$$

Kiros et al. (2013)

Encoder $\mathbf{x} = \text{embed}(\boldsymbol{x})$

Simple conditional *n*-gram LM:

$$\mathbf{h}_t = \mathbf{W}[\mathbf{w}_{t-n+1}; \mathbf{w}_{t-n+2}; \ldots; \mathbf{w}_{t-1}] + \mathbf{C}\mathbf{x}$$

$$\mathbf{u}_t = \mathbf{P}\mathbf{h}_t + \mathbf{b}$$

$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{t-n+1}^{t-1}) = \text{softmax}(\mathbf{u}_t)$$

Multiplicative *n*-gram LM:

~~$w_i = r_{i,w}$~~

$w_i = r_{i,j,w} x_j$

how big is this tensor?

Kiros et al. (2013)

Encoder $\mathbf{x} = \mathrm{embed}(\boldsymbol{x})$

Simple conditional *n*-gram LM:

$$\mathbf{h}_t = \mathbf{W}[\mathbf{w}_{t-n+1}; \mathbf{w}_{t-n+2}; \ldots; \mathbf{w}_{t-1}] + \mathbf{C}\mathbf{x}$$

$$\mathbf{u}_t = \mathbf{P}\mathbf{h}_t + \mathbf{b}$$

$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{t-n+1}^{t-1}) = \mathrm{softmax}(\mathbf{u}_t)$$

Multiplicative *n*-gram LM:

$$w_i = r_{i,w}$$

Kiros et al. (2013)

Encoder $\mathbf{x} = \mathrm{embed}(\boldsymbol{x})$

Simple conditional *n*-gram LM:

$$\mathbf{h}_t = \mathbf{W}[\mathbf{w}_{t-n+1}; \mathbf{w}_{t-n+2}; \ldots; \mathbf{w}_{t-1}] + \mathbf{C}\mathbf{x}$$

$$\mathbf{u}_t = \mathbf{P}\mathbf{h}_t + \mathbf{b}$$

$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{t-n+1}^{t-1}) = \mathrm{softmax}(\mathbf{u}_t)$$

Multiplicative *n*-gram LM:

~~$w_i = r_{i,w}$~~

$$w_i = r_{i,j,w} x_j$$

Kiros et al. (2013)

Encoder $\mathbf{x} = \text{embed}(\boldsymbol{x})$

Simple conditional *n*-gram LM:

$$\mathbf{h}_t = \mathbf{W}[\mathbf{w}_{t-n+1}; \mathbf{w}_{t-n+2}; \ldots; \mathbf{w}_{t-1}] + \mathbf{C}\mathbf{x}$$

$$\mathbf{u}_t = \mathbf{P}\mathbf{h}_t + \mathbf{b}$$

$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{t-n+1}^{t-1}) = \text{softmax}(\mathbf{u}_t)$$

Multiplicative *n*-gram LM:

~~$w_i = r_{i,w}$~~

$$w_i = r_{i,j,w} x_j$$

what's the intuition here?

Kiros et al. (2013)

Encoder $\mathbf{x} = \text{embed}(\boldsymbol{x})$

Simple conditional *n*-gram LM:

$$\mathbf{h}_t = \mathbf{W}[\mathbf{w}_{t-n+1}; \mathbf{w}_{t-n+2}; \ldots; \mathbf{w}_{t-1}] + \mathbf{C}\mathbf{x}$$

$$\mathbf{u}_t = \mathbf{P}\mathbf{h}_t + \mathbf{b}$$

$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{t-n+1}^{t-1}) = \text{softmax}(\mathbf{u}_t)$$

Multiplicative *n*-gram LM:

~~$w_i = r_{i,w}$~~

$w_i = r_{i,j,w} x_j$

how big is this tensor?

Kiros et al. (2013)

Encoder $\mathbf{x} = \mathrm{embed}(\boldsymbol{x})$

Simple conditional *n*-gram LM:

$$\mathbf{h}_t = \mathbf{W}[\mathbf{w}_{t-n+1}; \mathbf{w}_{t-n+2}; \ldots; \mathbf{w}_{t-1}] + \mathbf{C}\mathbf{x}$$

$$\mathbf{u}_t = \mathbf{P}\mathbf{h}_t + \mathbf{b}$$

$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{t-n+1}^{t-1}) = \mathrm{softmax}(\mathbf{u}_t)$$

Multiplicative *n*-gram LM:

~~$w_i = r_{i,w}$~~

$$w_i = r_{i,j,w} x_j$$

Kiros et al. (2013)

Encoder $\mathbf{x} = \text{embed}(\boldsymbol{x})$

Simple conditional *n*-gram LM:

$$\mathbf{h}_t = \mathbf{W}[\mathbf{w}_{t-n+1}; \mathbf{w}_{t-n+2}; \ldots; \mathbf{w}_{t-1}] + \mathbf{C}\mathbf{x}$$

$$\mathbf{u}_t = \mathbf{P}\mathbf{h}_t + \mathbf{b}$$

$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{t-n+1}^{t-1}) = \text{softmax}(\mathbf{u}_t)$$

Multiplicative *n*-gram LM:

~~$w_i = r_{i,w}$~~

~~$w_i = r_{i,j,w} x_j$~~

$$w_i = u_{w,i} v_{i,j} \qquad (\mathbf{U} \in \mathbb{R}^{|V| \times d}, \quad \mathbf{V} \in \mathbb{R}^{d \times k})$$

$$\mathbf{r}_t = \mathbf{W}[\mathbf{w}_{t-n+1}; \mathbf{w}_{t-n+2}; \ldots; \mathbf{w}_{t-1}] + \mathbf{C}\mathbf{x}$$

Kiros et al. (2013)

Encoder $\mathbf{x} = \mathrm{embed}(\boldsymbol{x})$

Simple conditional *n*-gram LM:

$$\mathbf{h}_t = \mathbf{W}[\mathbf{w}_{t-n+1}; \mathbf{w}_{t-n+2}; \ldots; \mathbf{w}_{t-1}] + \mathbf{Cx}$$

$$\mathbf{u}_t = \mathbf{P}\mathbf{h}_t + \mathbf{b}$$

$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{t-n+1}^{t-1}) = \mathrm{softmax}(\mathbf{u}_t)$$

Multiplicative *n*-gram LM:

$$w_i = u_{w,i} v_{i,j} \qquad (\mathbf{U} \in \mathbb{R}^{|V| \times d}, \quad \mathbf{V} \in \mathbb{R}^{d \times k})$$

$$\mathbf{r}_t = \mathbf{W}[\mathbf{w}_{t-n+1}; \mathbf{w}_{t-n+2}; \ldots; \mathbf{w}_{t-1}] + \mathbf{Cx}$$

$$\mathbf{h}_t = (\mathbf{W}^{fr}\mathbf{r}_t) \odot (\mathbf{W}^{fx}\mathbf{x})$$

$$\mathbf{u}_t = \mathbf{P}\mathbf{h}_t + \mathbf{b}$$

$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{<t}) = \mathrm{softmax}(\mathbf{u}_t)$$

Kiros et al. (2013)

- Two take-home messages:
  - Feed-forward *n*-gram models can be used in place of RNNs in conditional models
  - Modeling interactions between input modalities holds a lot of promise
    - Although MLP-type models can approximate higher order tensors, multiplicative models appear to make learning interactions easier

Conditioning with vectors

We are compressing a lot of information in a finite-sized vector.

Conditioning with vectors

We are compressing a lot of information in a finite-sized vector.

Prof. Ray Mooney

"You can't cram the meaning of a whole %&!$# sentence into a single $&!#* vector!"

Conditioning with vectors

We are compressing a lot of information in a finite-sized vector.

Gradients have a long way to travel. Even LSTMs forget!

Conditioning with vectors

We are compressing a lot of information in a finite-sized vector.

Gradients have a long way to travel. Even LSTMs forget!

What is to be done?

Outline of Final Section

- Machine translation with attention
- Image caption generation with attention

Outline of Final Section

- Machine translation with attention
- Image caption generation with attention

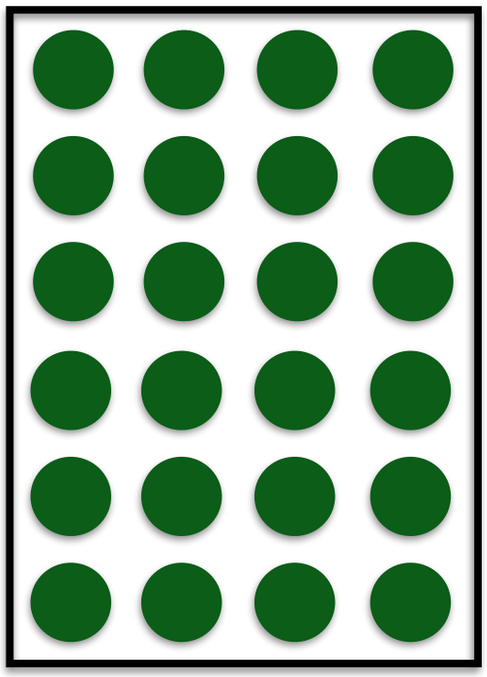
Solving the Vector Problem in Translation

- Represent a source sentence as a matrix
- Generate a target sentence from a matrix
- This will
  - Solve the capacity problem
  - Solve the gradient flow problem

Sentences as Matrices

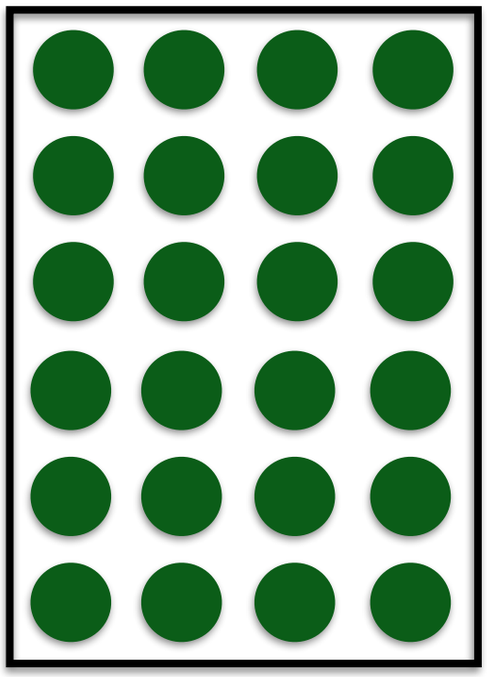
- Problem with the fixed-size vector model
  - Sentences are of different sizes but vectors are of the same size
- Solution: use matrices instead
  - Fixed number of rows, but number of columns depends on the number of words
  - Usually $|\boldsymbol{f}|$ = #cols

Sentences as Matrices

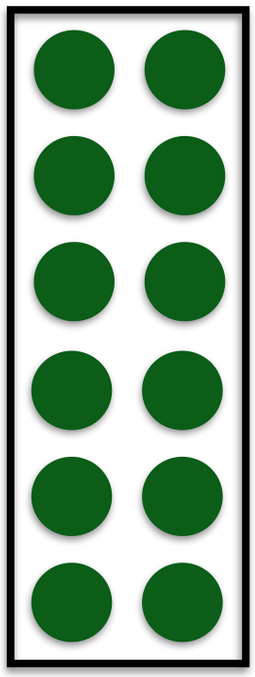


Ich möchte ein Bier

Sentences as Matrices

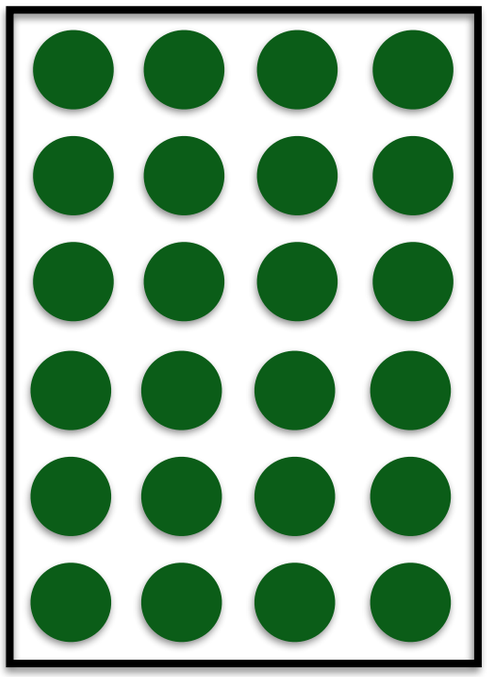


Ich möchte ein Bier

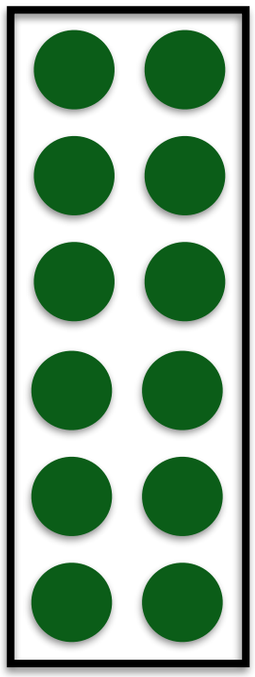


Mach's gut

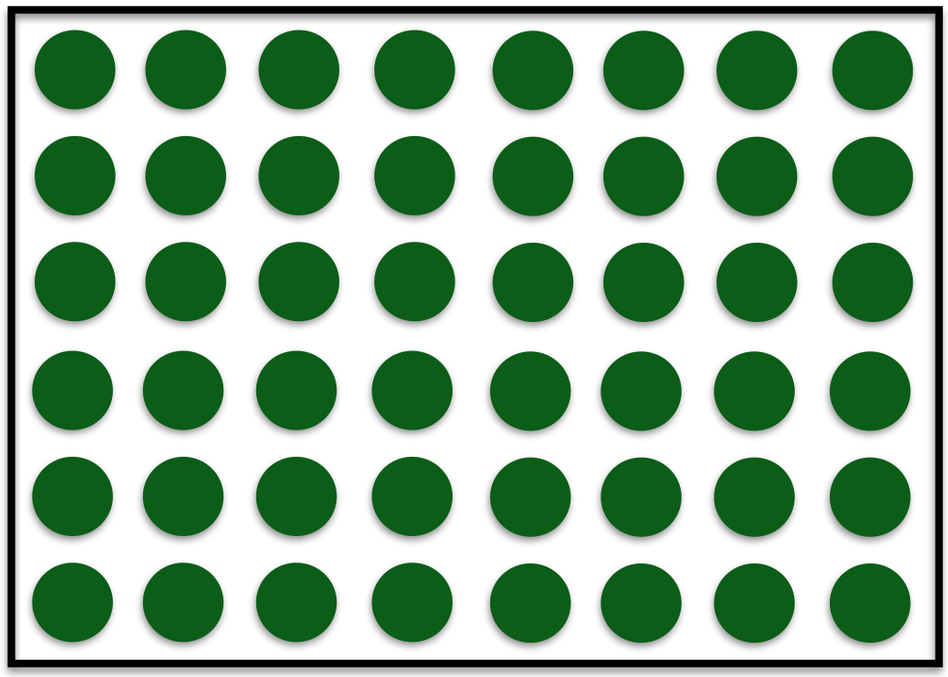
Sentences as Matrices



Ich möchte ein Bier

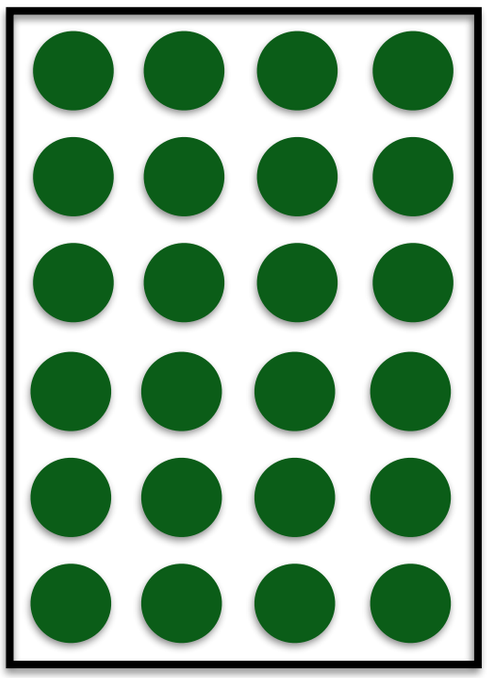


Mach's gut

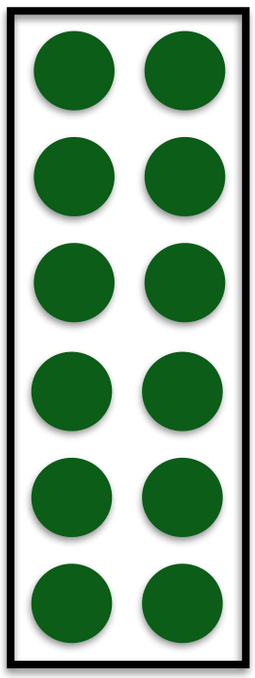


Die Wahrheiten der Menschen sind die unwiderlegbaren Irrtümer

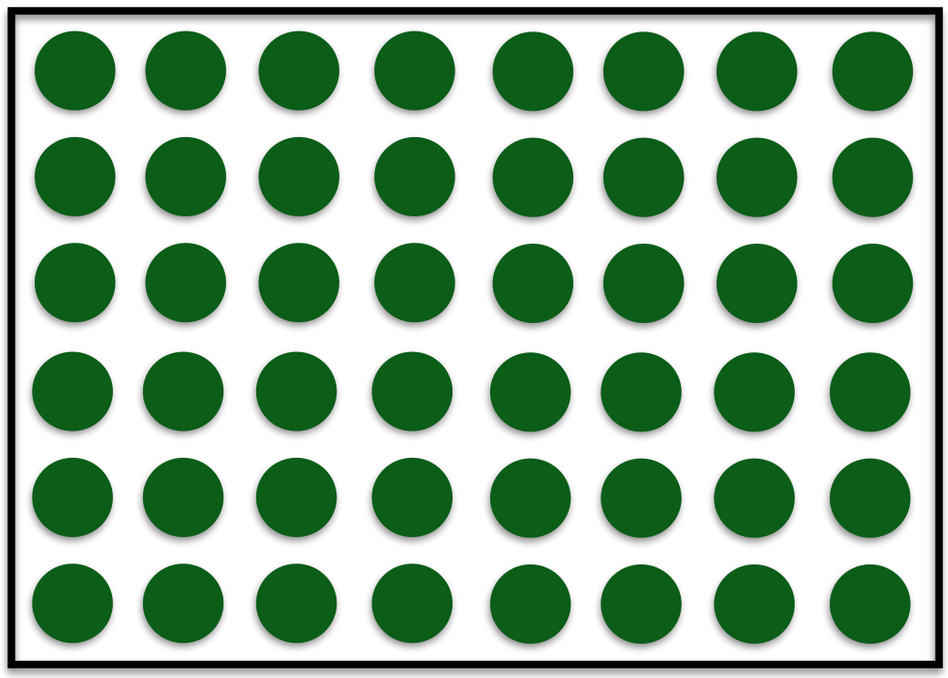
Sentences as Matrices



Ich möchte ein Bier



Mach's gut



Die Wahrheiten der Menschen sind die unwiderlegbaren Irrtümer

Question: How do we build these matrices?

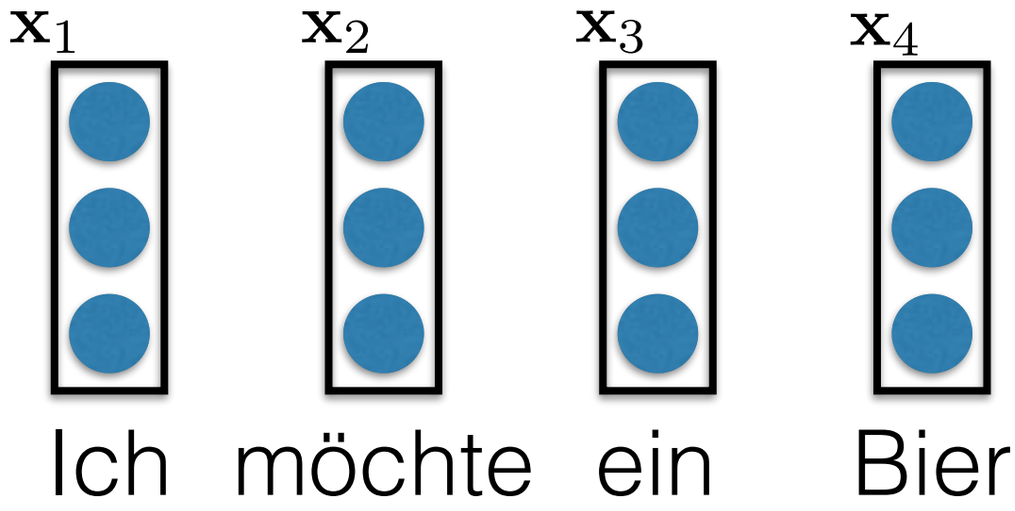
With Concatenation

- Each word type is represented by an n-dimensional vector

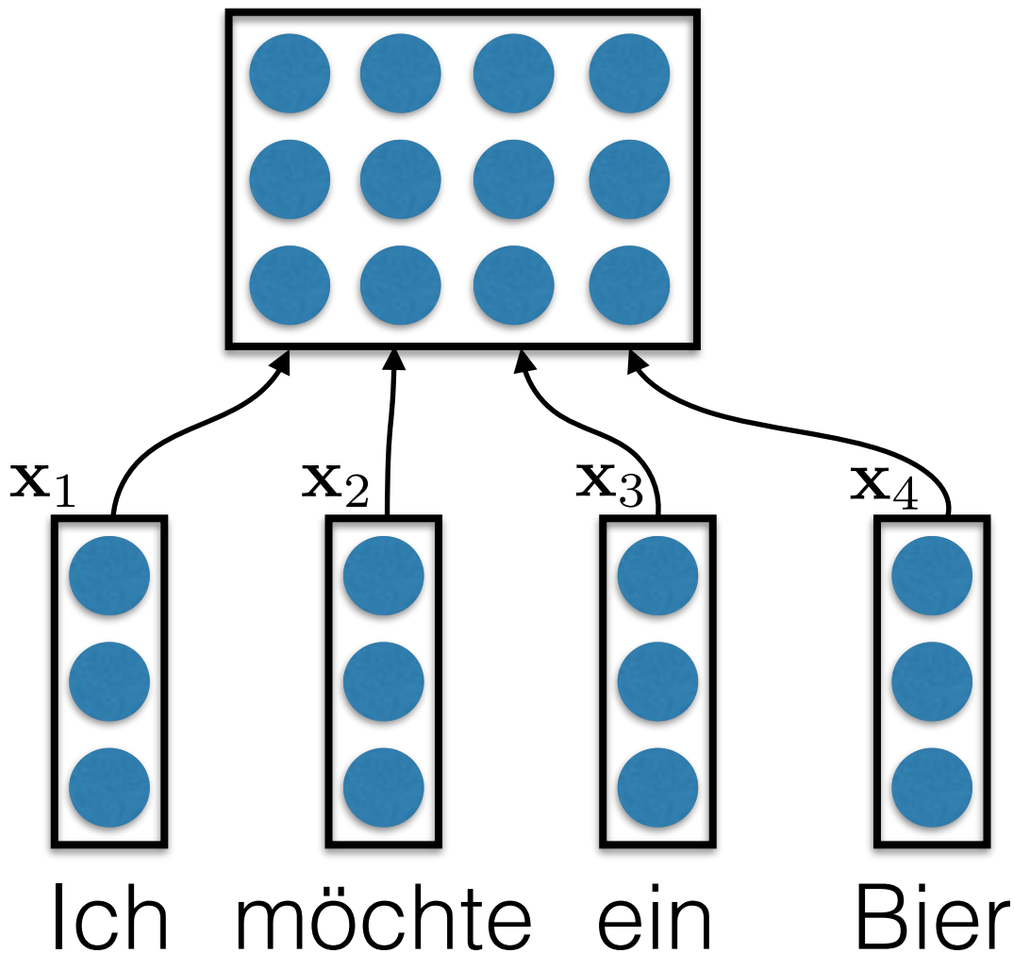
- Take all of the vectors for the sentence and concatenate them into a matrix

- Simplest possible model

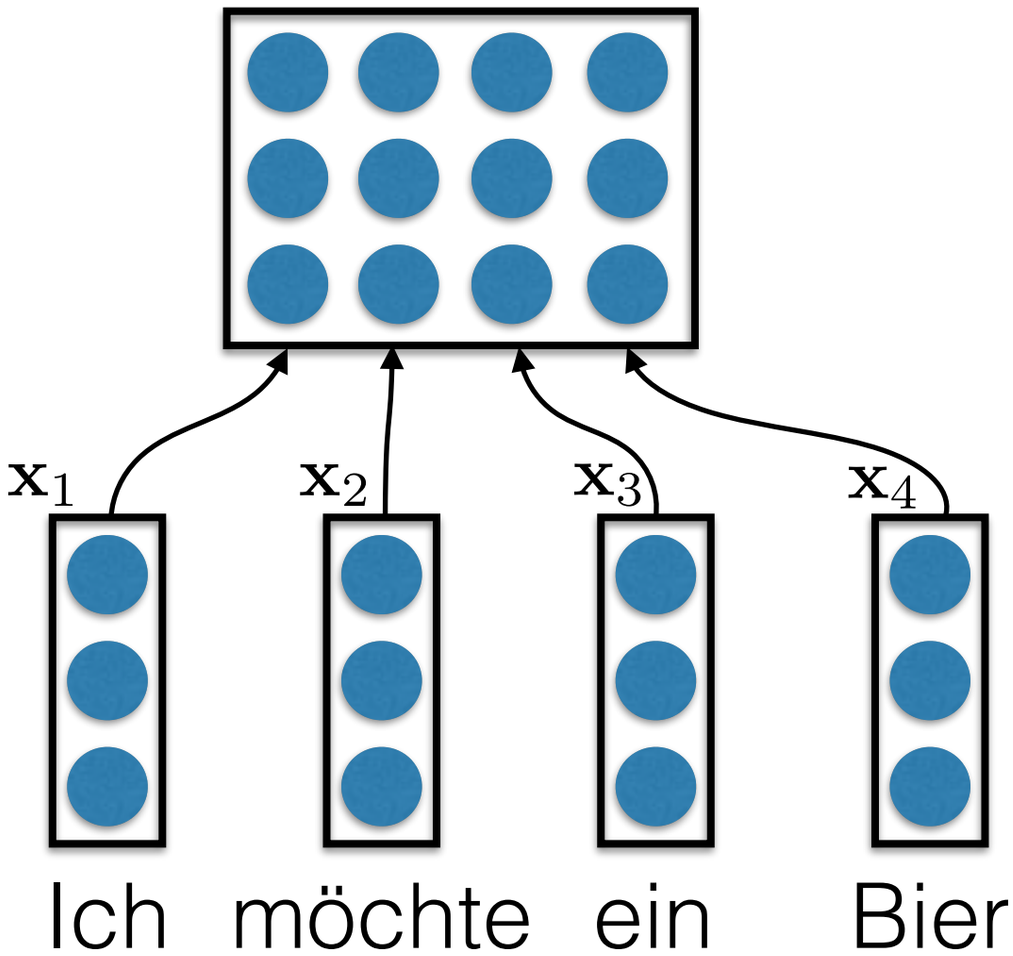
    - So simple, no one has bothered to publish how well/badly it works!


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

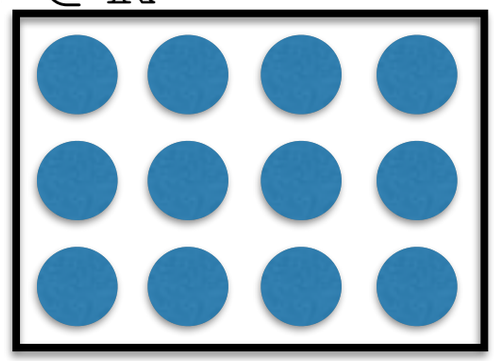
$$\mathbf{f}_i = \mathbf{x}_i$$



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



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



Ich möchte ein Bier

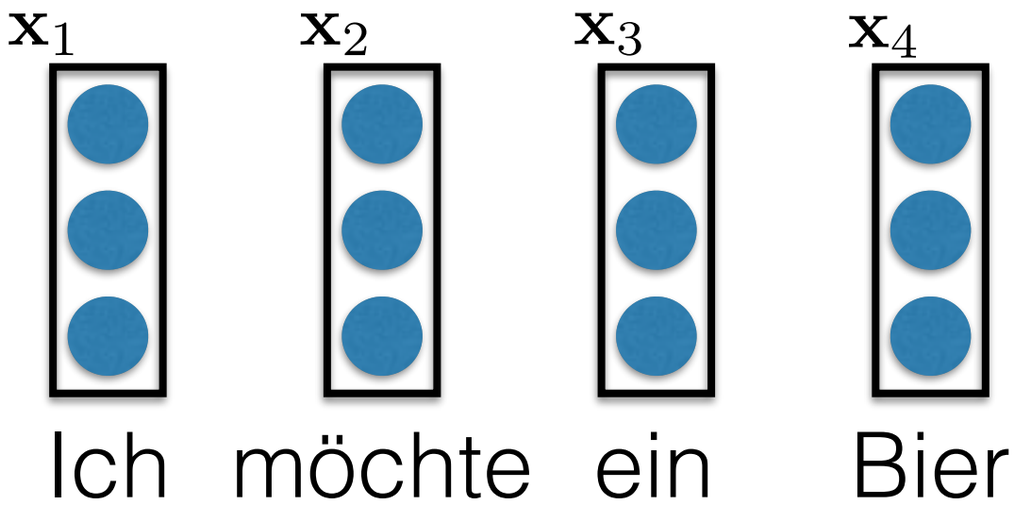
With Convolutional Nets

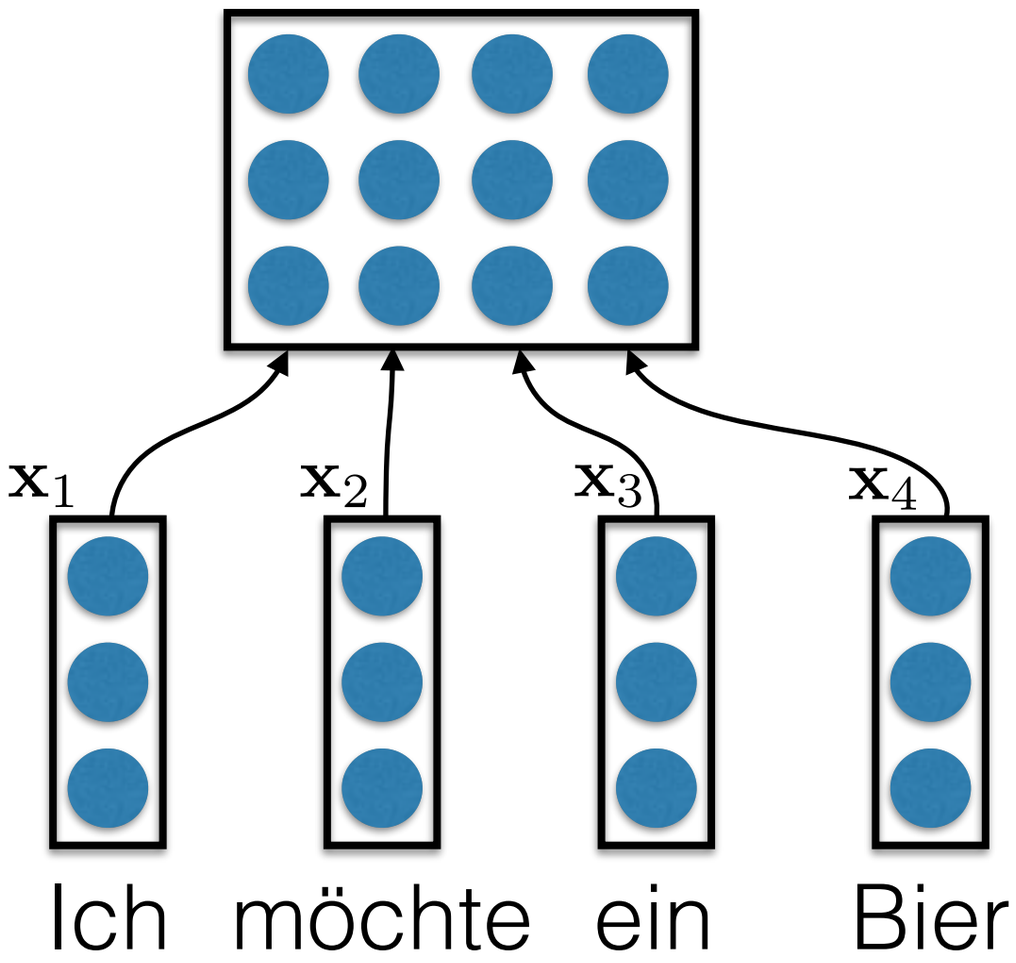
- Apply convolutional networks to transform the naive concatenated matrix to obtain a context-dependent matrix

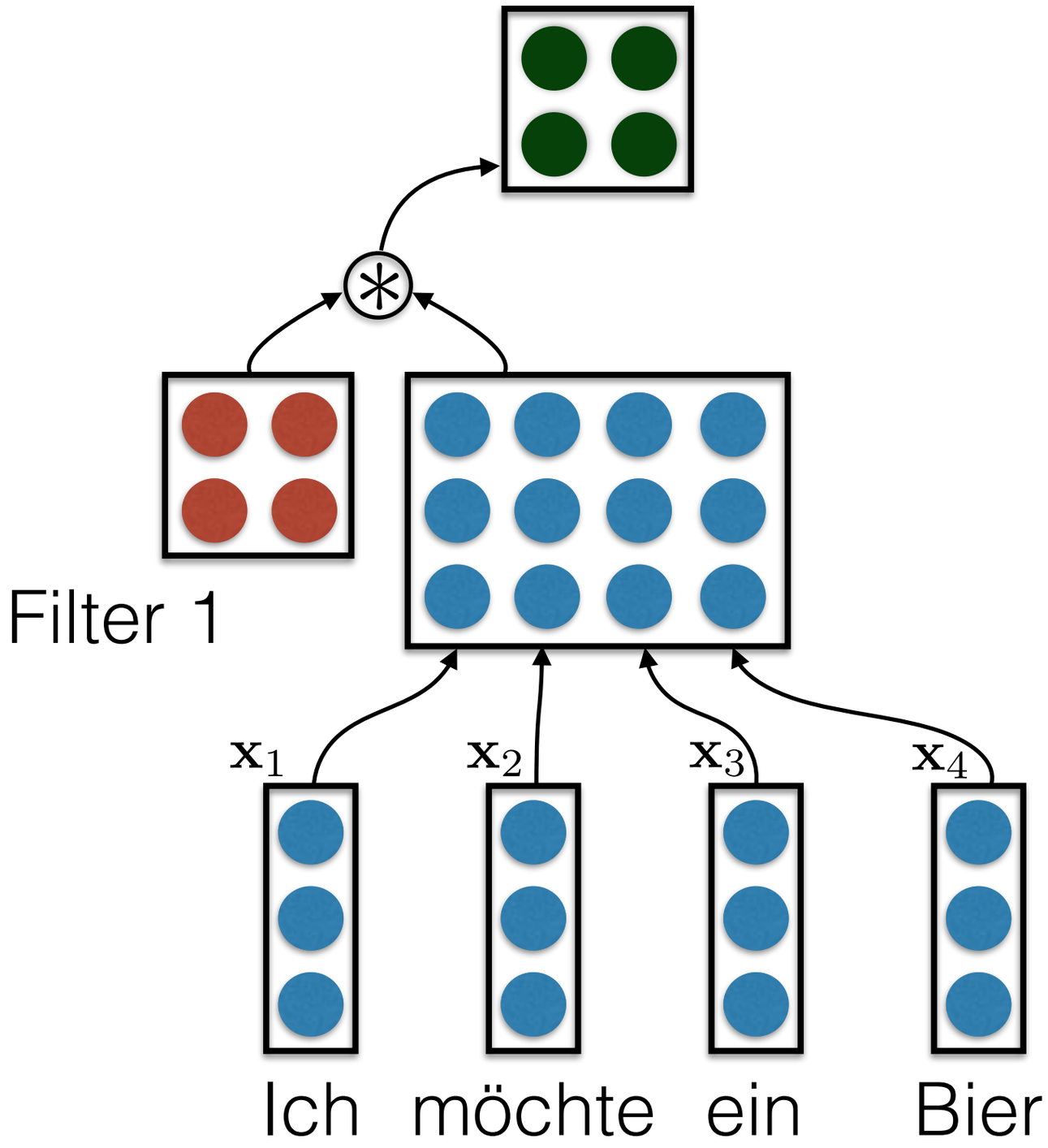
- Explored in a recent ICLR submission by Gehring et al., 2016 (from FAIR)

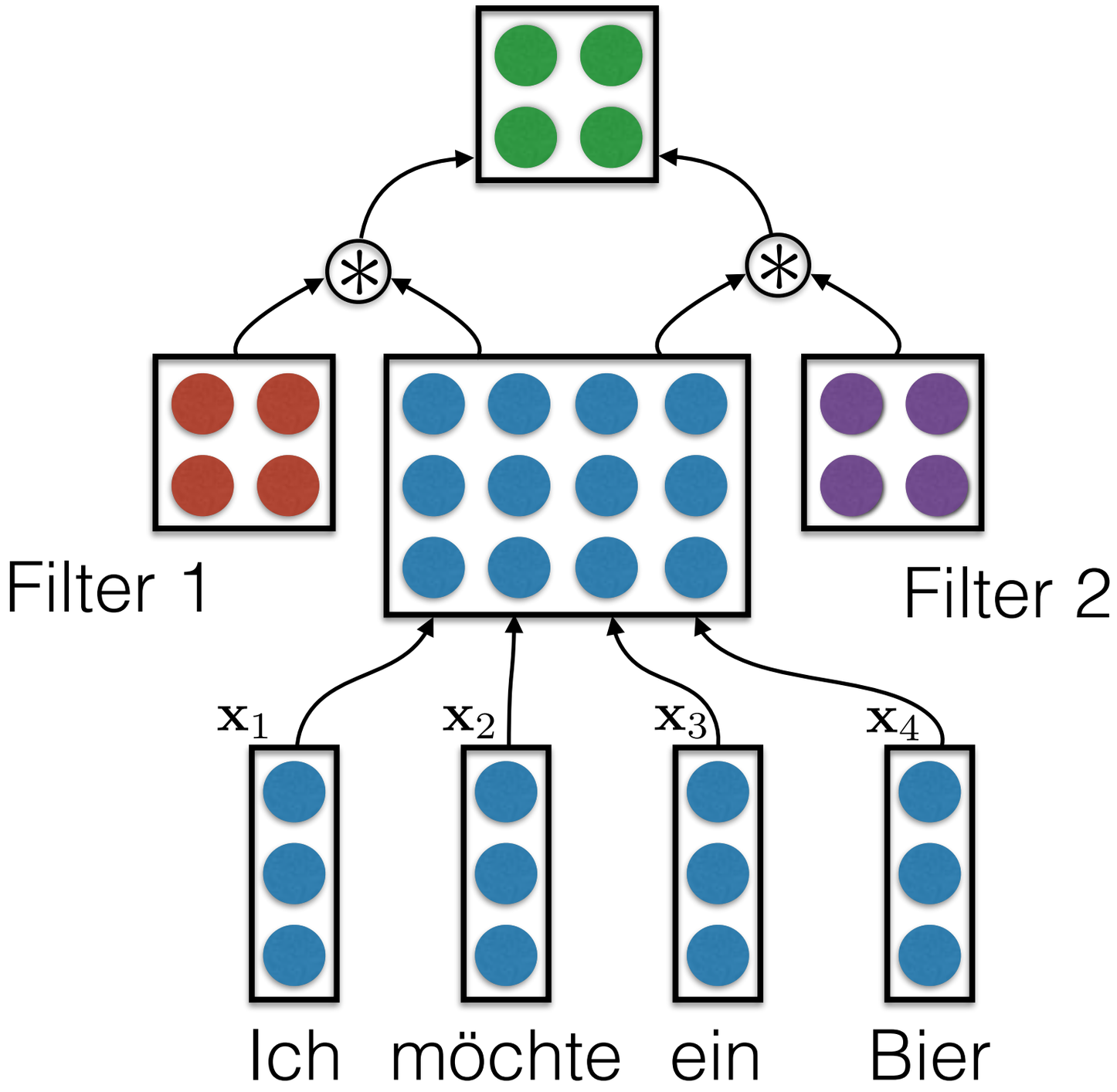
    - Closely related to the neural translation model proposed by Kalchbrenner and Blunsom, 2013

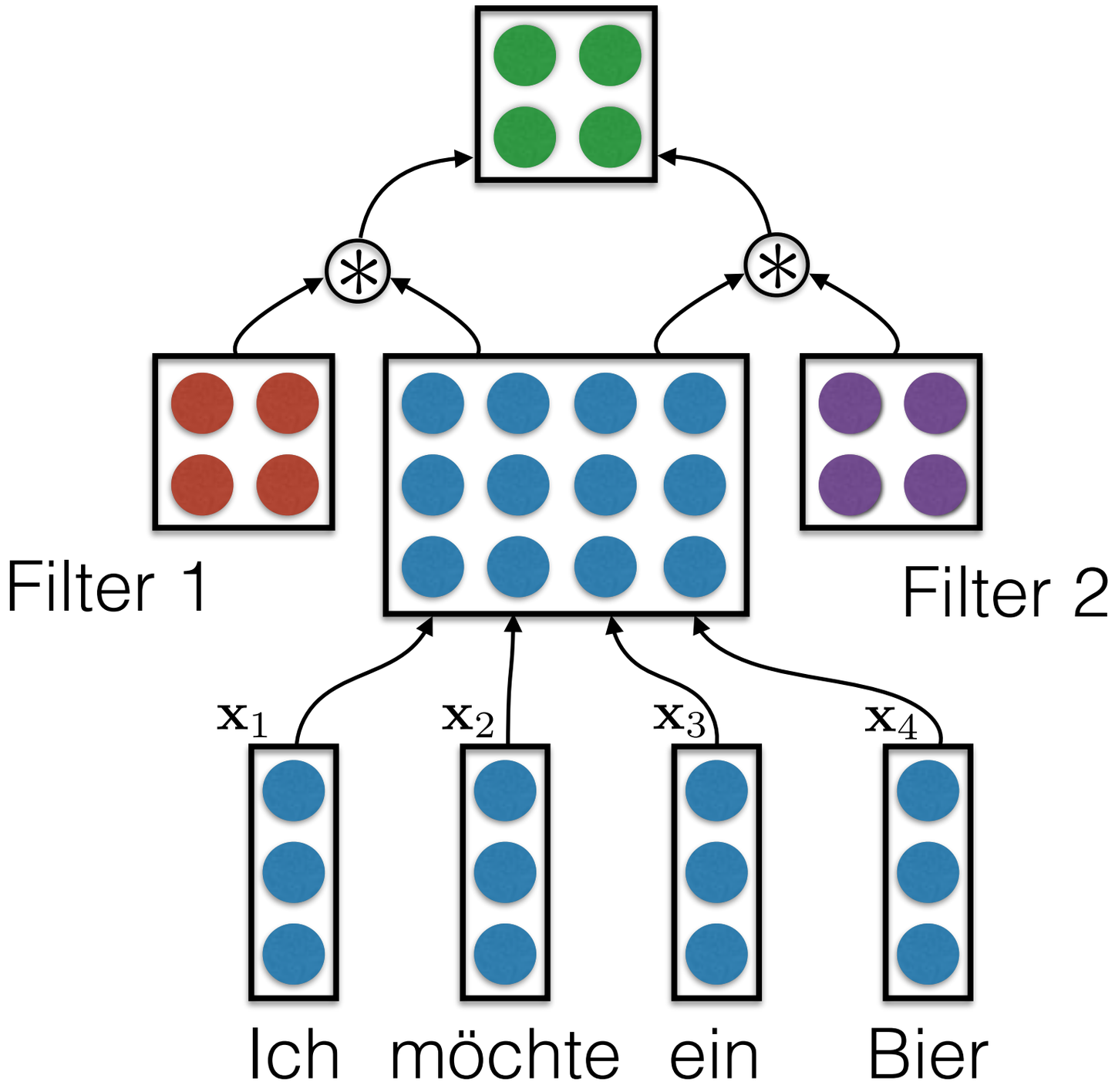
- Note: convnets usually have a “pooling” operation at the top level that results in a fixed-sized representation. For sentences, leave this out.


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

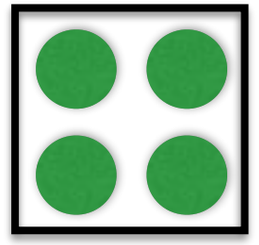

$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier


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)

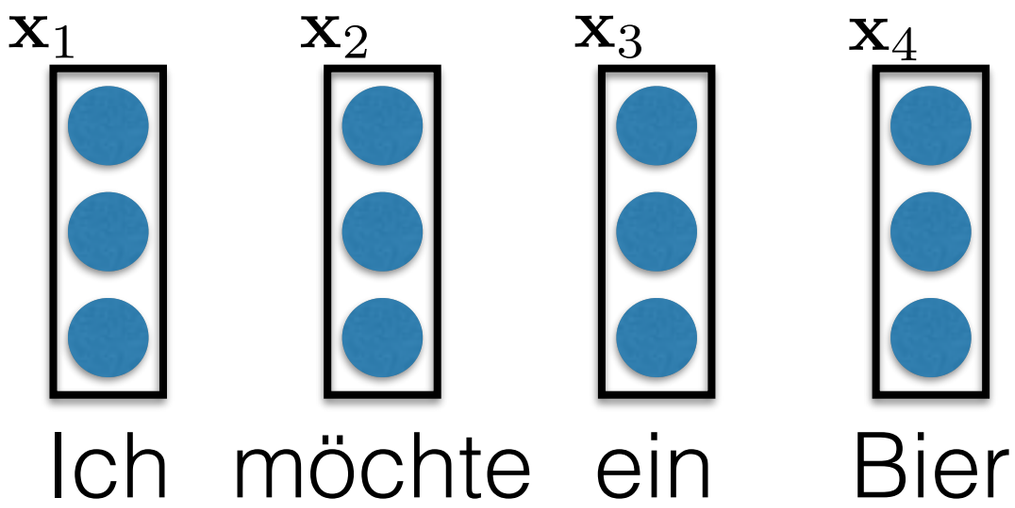
- One column per word

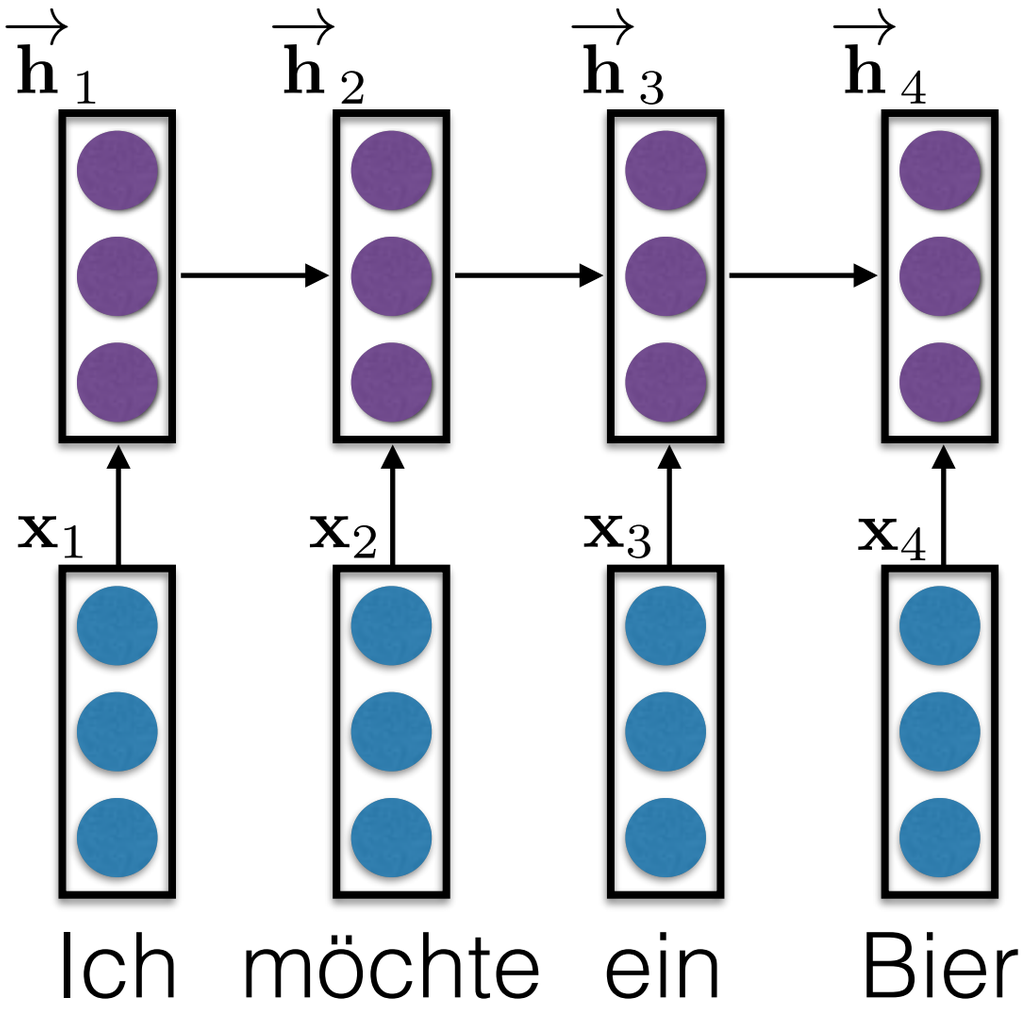
- Each column (word) has two halves concatenated together:

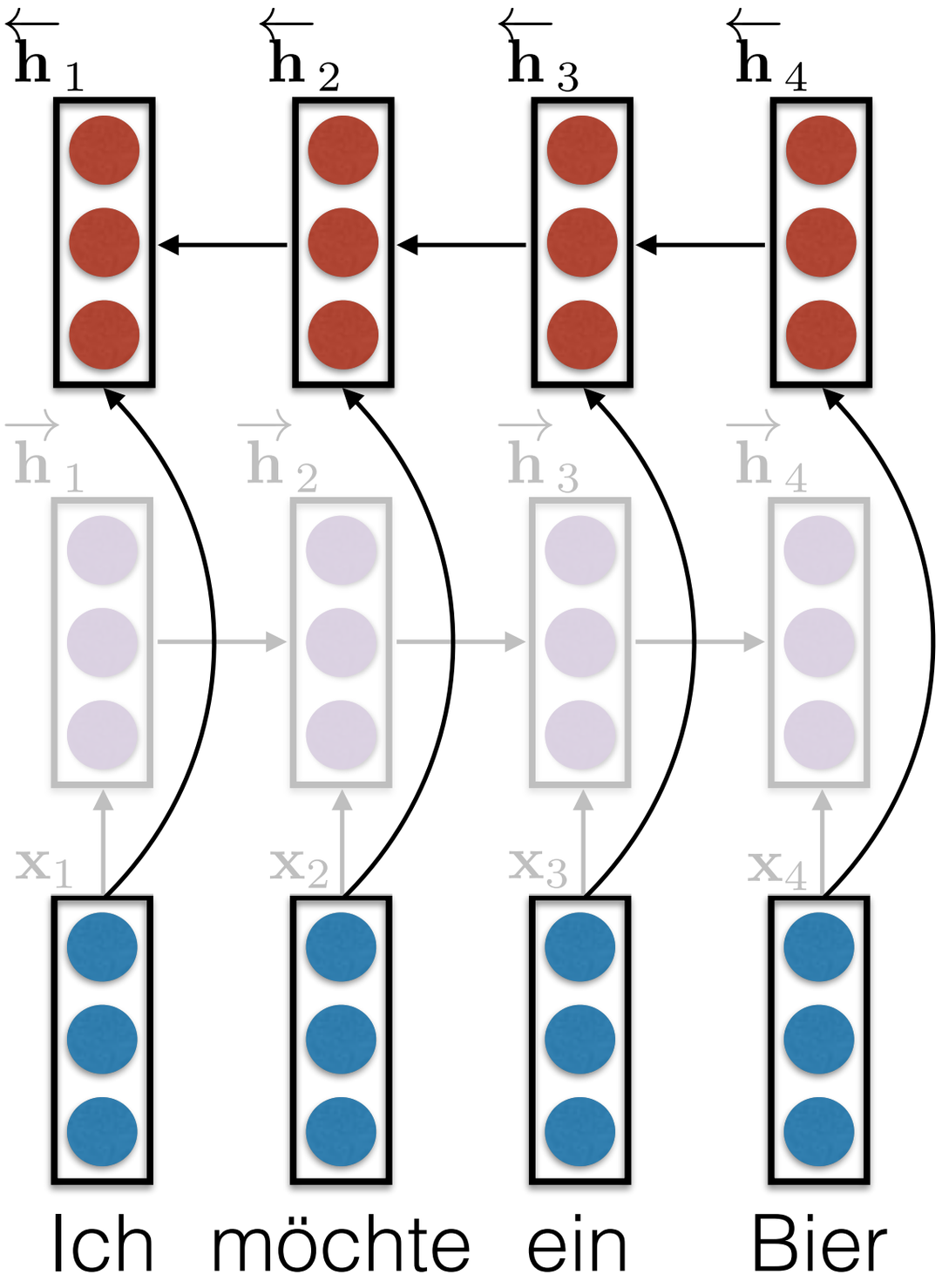
  - a “forward representation”, i.e., a word and its left context

  - a “reverse representation”, i.e., a word and its right context

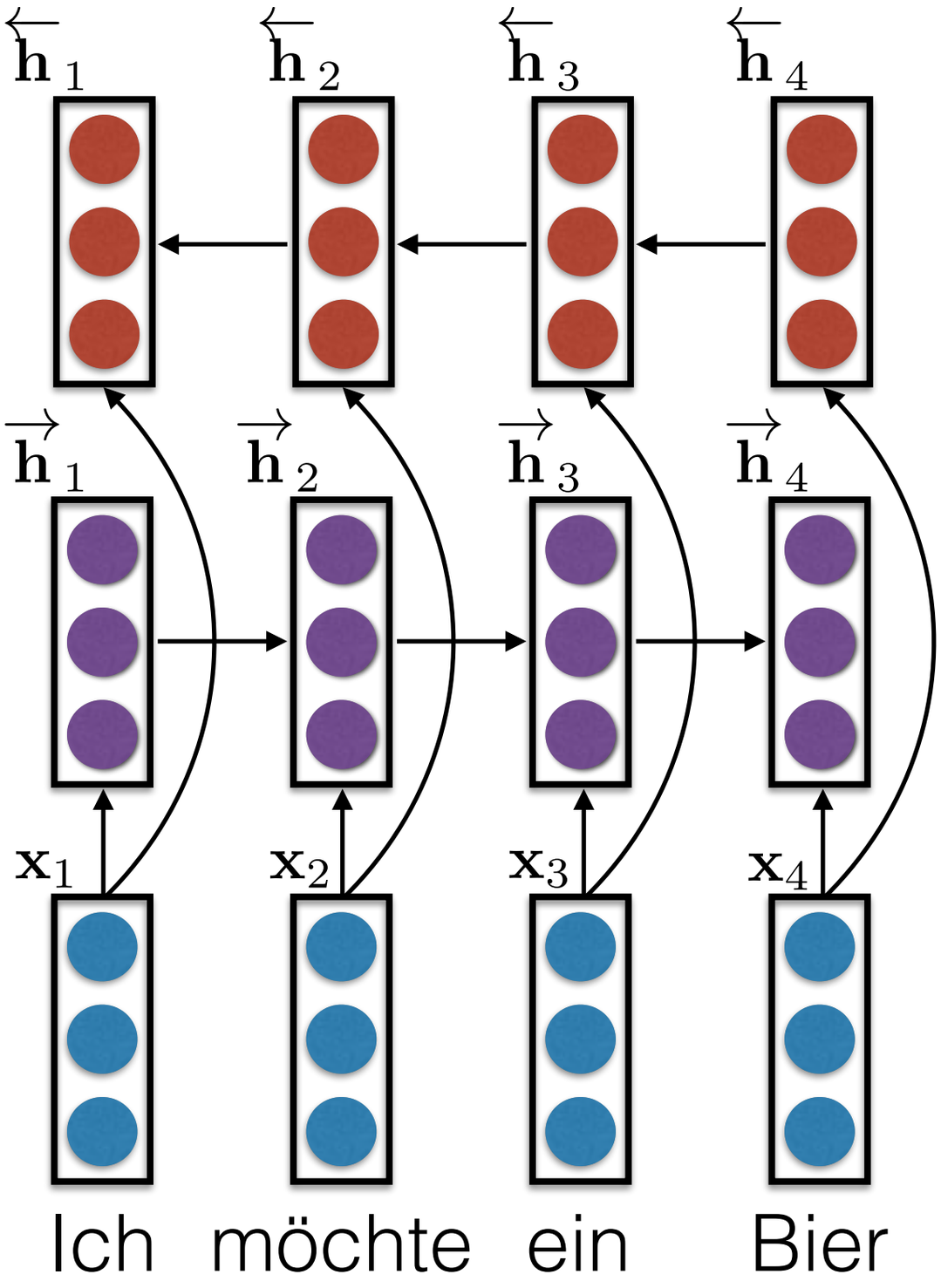
- Implementation: **bidirectional RNNs** (GRUs or LSTMs) to read ***f*** from left to right and right to left, concatenate representations

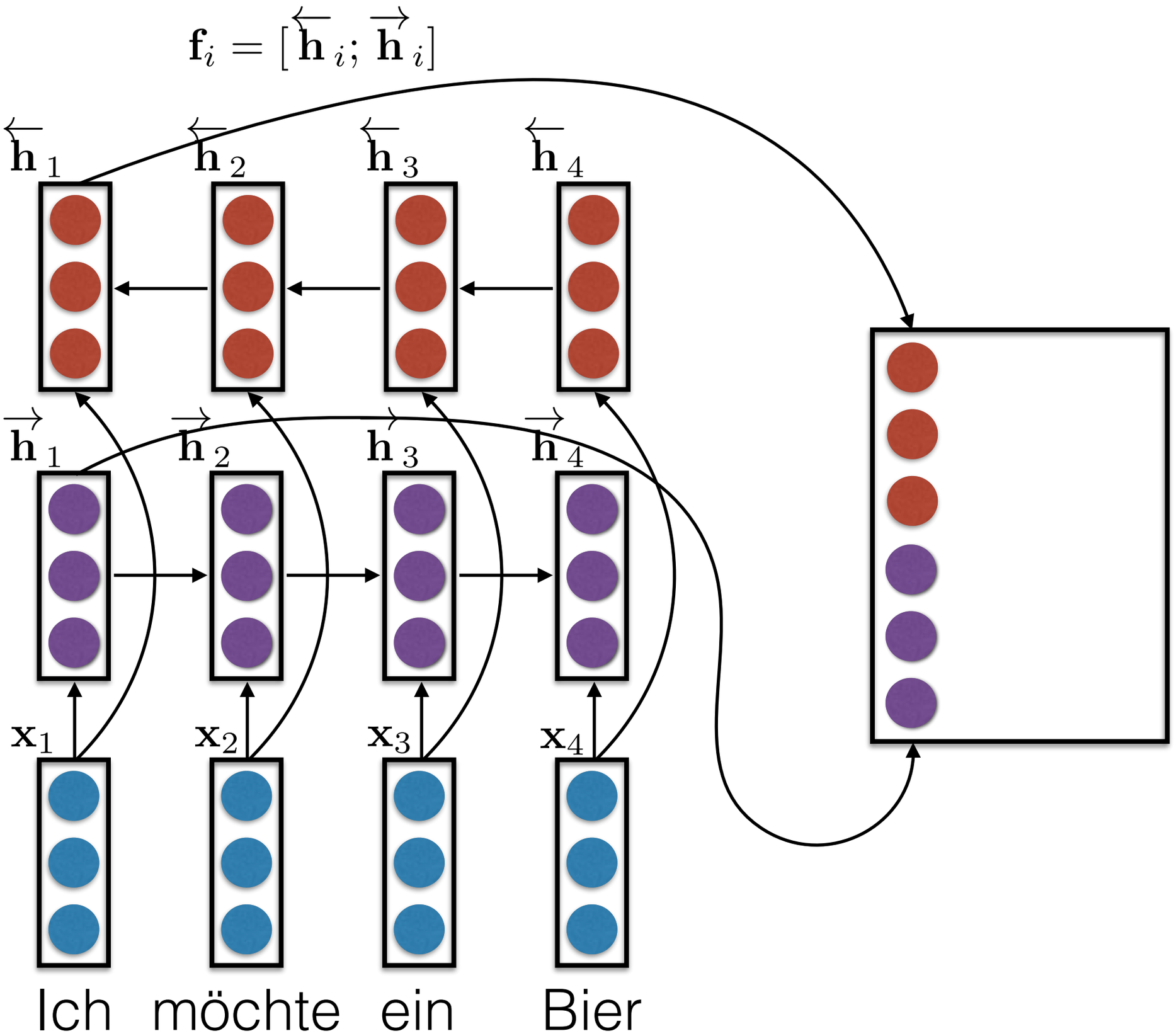

$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier

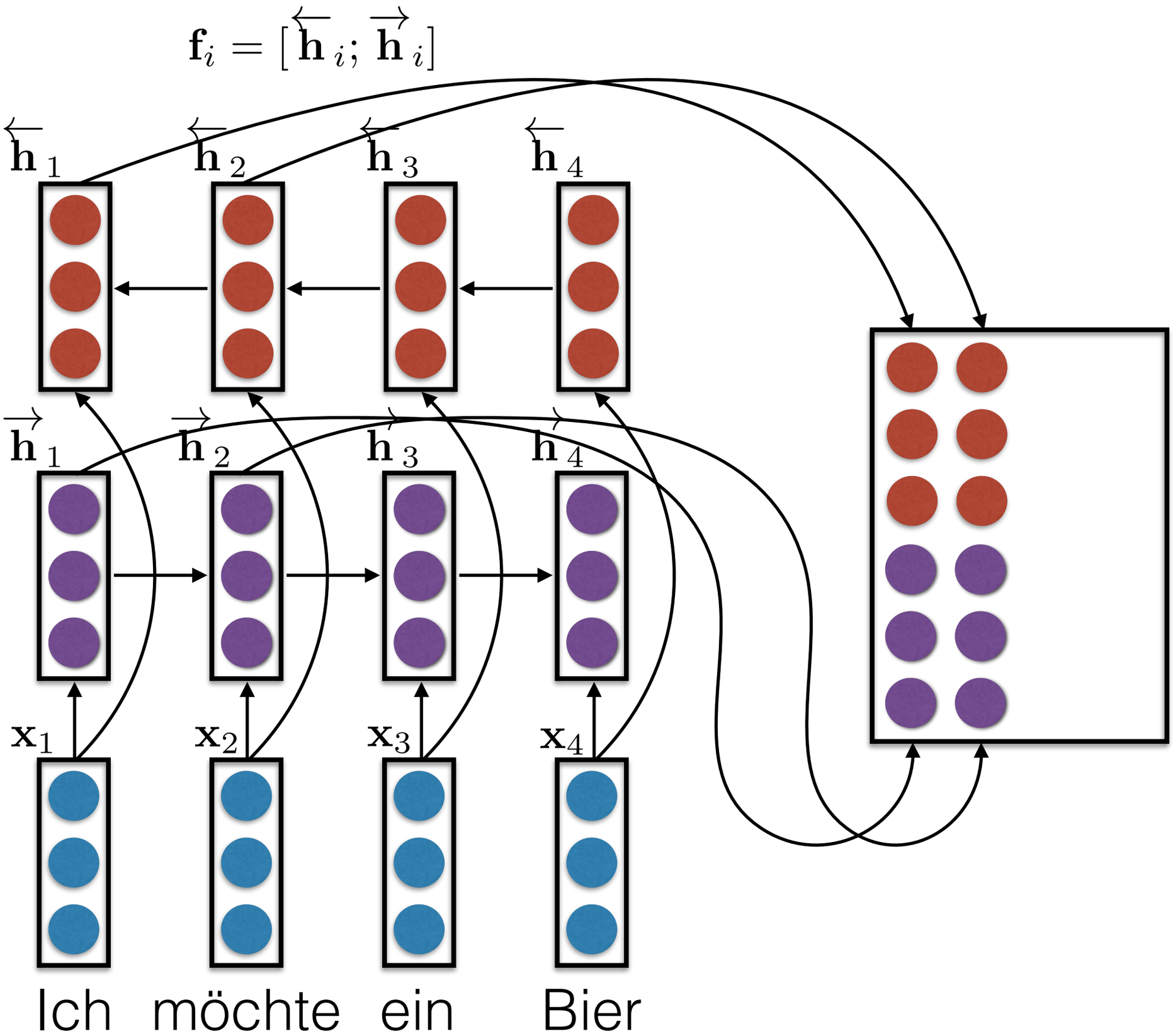

$\overrightarrow{\mathbf{h}}_1$
$\overrightarrow{\mathbf{h}}_2$
$\overrightarrow{\mathbf{h}}_3$
$\overrightarrow{\mathbf{h}}_4$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier


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

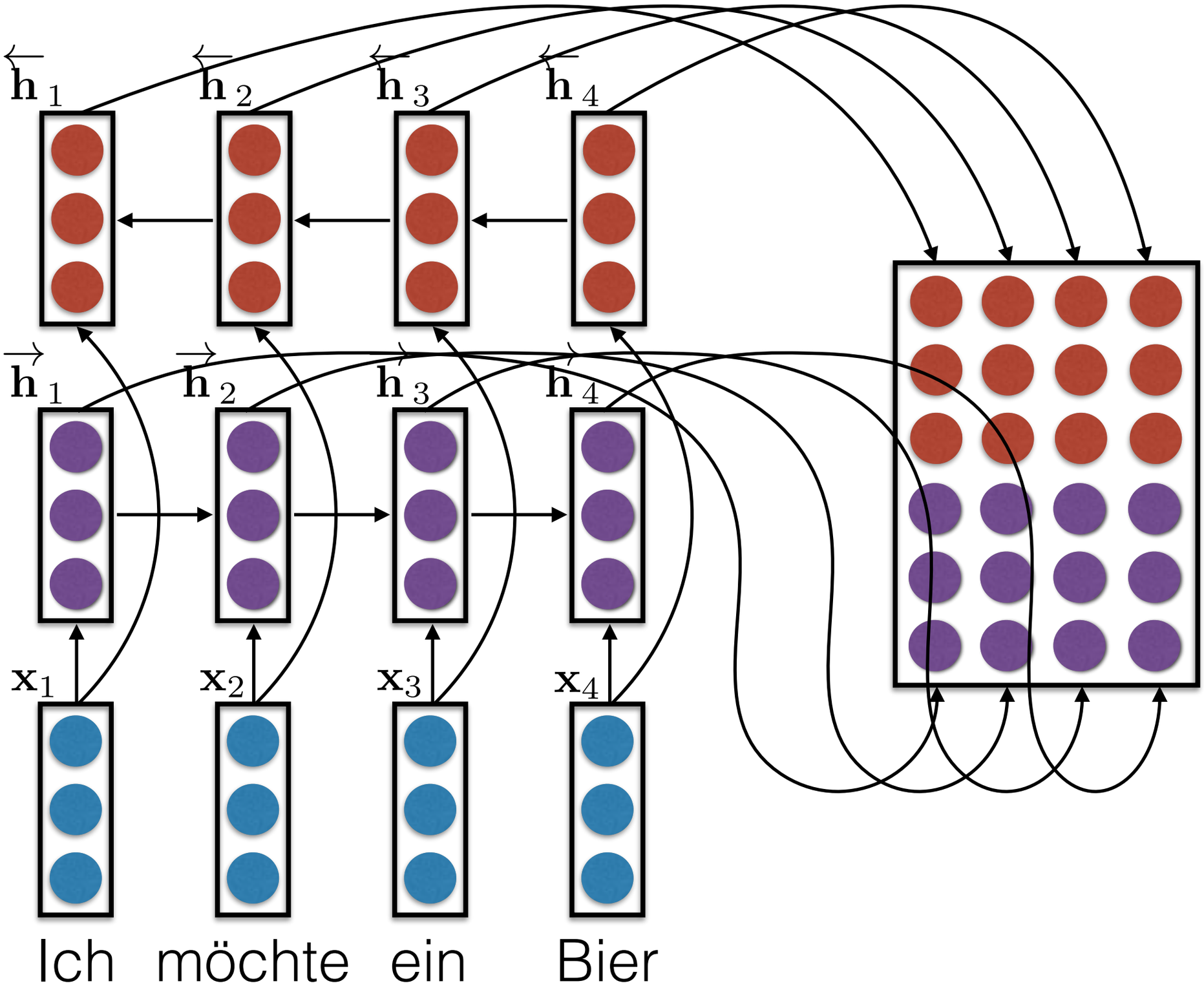
$$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$$



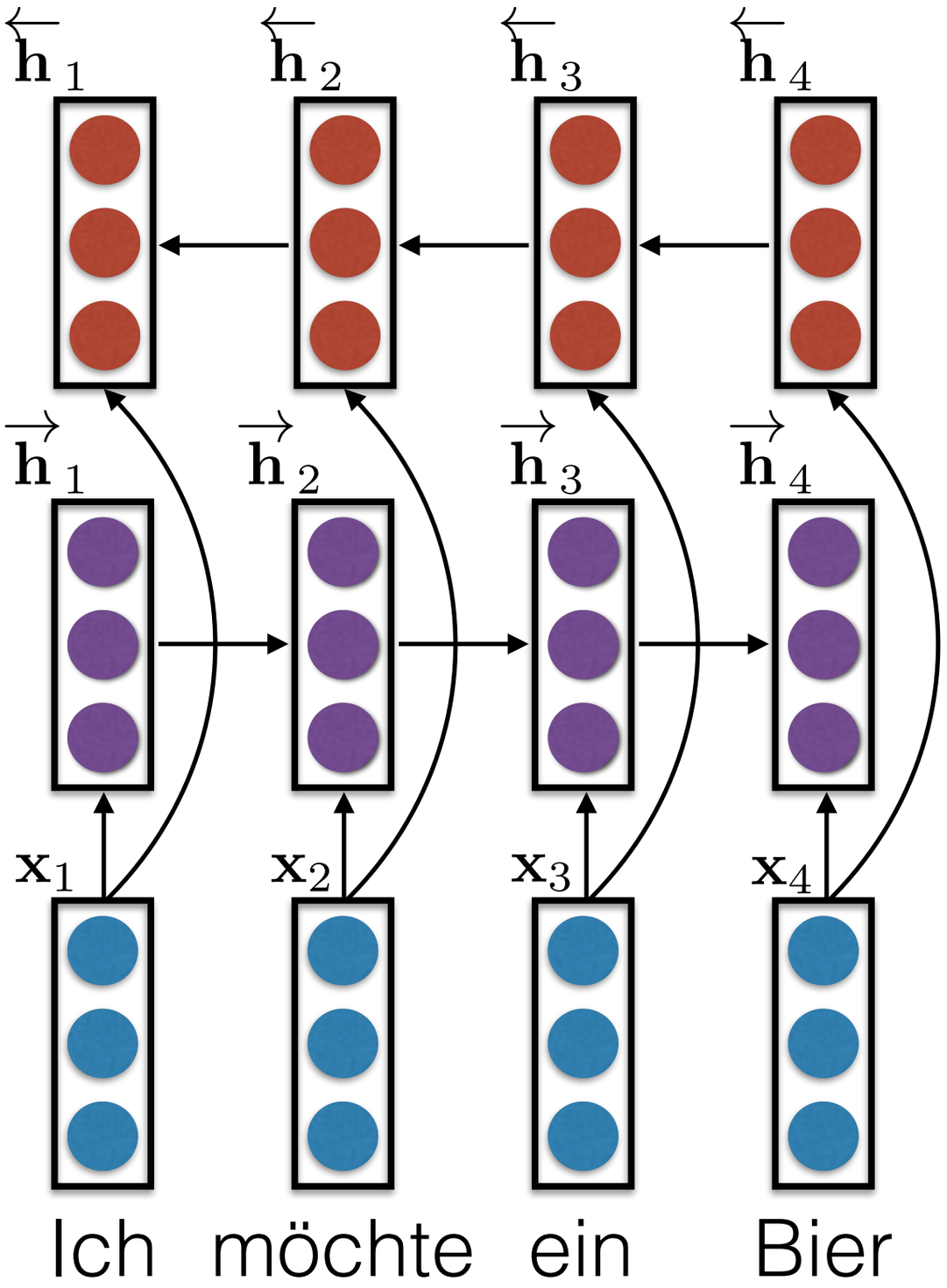

$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$
$\overleftarrow{\mathbf{h}}_1$
$\overleftarrow{\mathbf{h}}_2$
$\overleftarrow{\mathbf{h}}_3$
$\overleftarrow{\mathbf{h}}_4$
$\overrightarrow{\mathbf{h}}_1$
$\overrightarrow{\mathbf{h}}_2$
$\overrightarrow{\mathbf{h}}_3$
$\overrightarrow{\mathbf{h}}_4$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

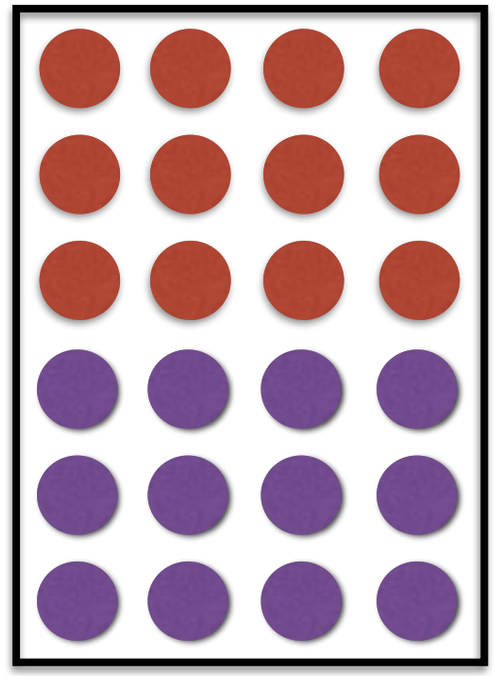
$$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$$



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



$$\mathbf{F} \in \mathbb{R}^{2n \times |\boldsymbol{f}|}$$



Ich möchte ein Bier

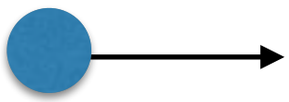
Where are we in 2017?

- There are lots of ways to construct **F**
  - Very little systematic work comparing them
  - There are many more undiscovered things out there
    - convolutions are particularly interesting and under-explored
    - syntactic information can help (Sennrich & Haddow, 2016; Nadejde et al., 2017), but many more integration strategies are possible
    - try something with phrase types instead of word types?

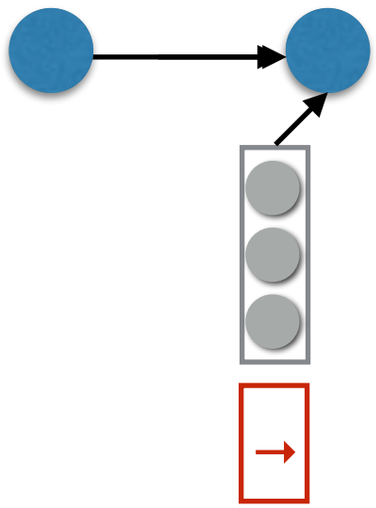
Multi-word expressions are a pain in the neck .

Generation from Matrices

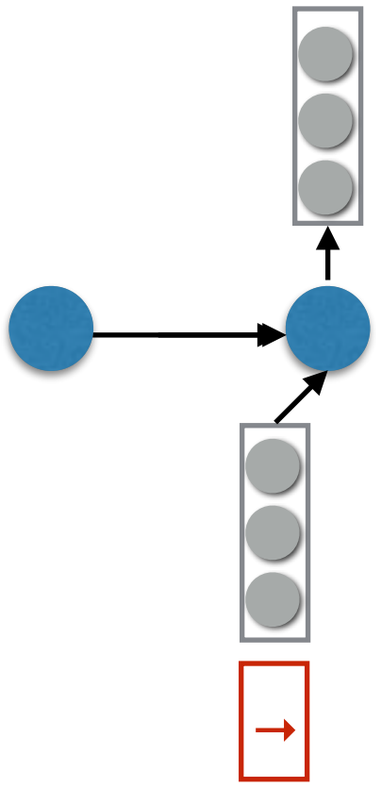
- We have a matrix **F** representing the input, now we need to generate from it
- Bahdanau et al. (2015) were the first to propose using ***attention*** for translating from matrix-encoded sentences
- High-level idea
  - Generate the output sentence word by word using an RNN
  - At each output position *t*, the RNN receives **two** inputs (in addition to any recurrent inputs)
    - a fixed-size vector embedding of the previously generated output symbol $e_{t\text{-}1}$
    - 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 $\mathbf{Fa}_t$)
    - The weighting of the input columns at each time-step ($\mathbf{a}_t$) is called ***attention***



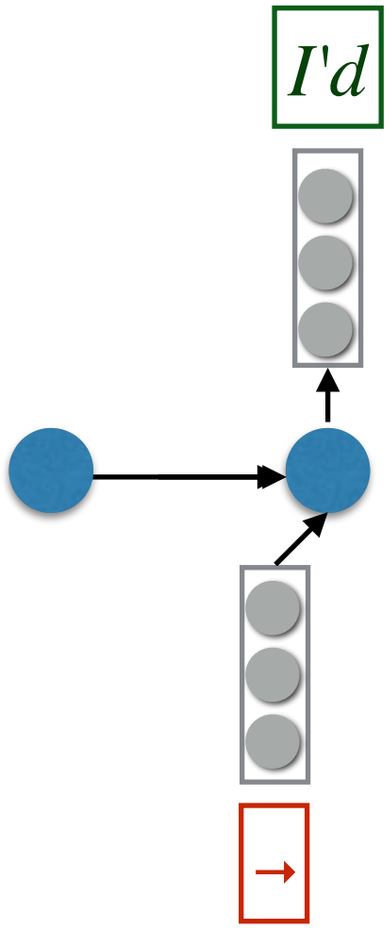
Recall RNNs…



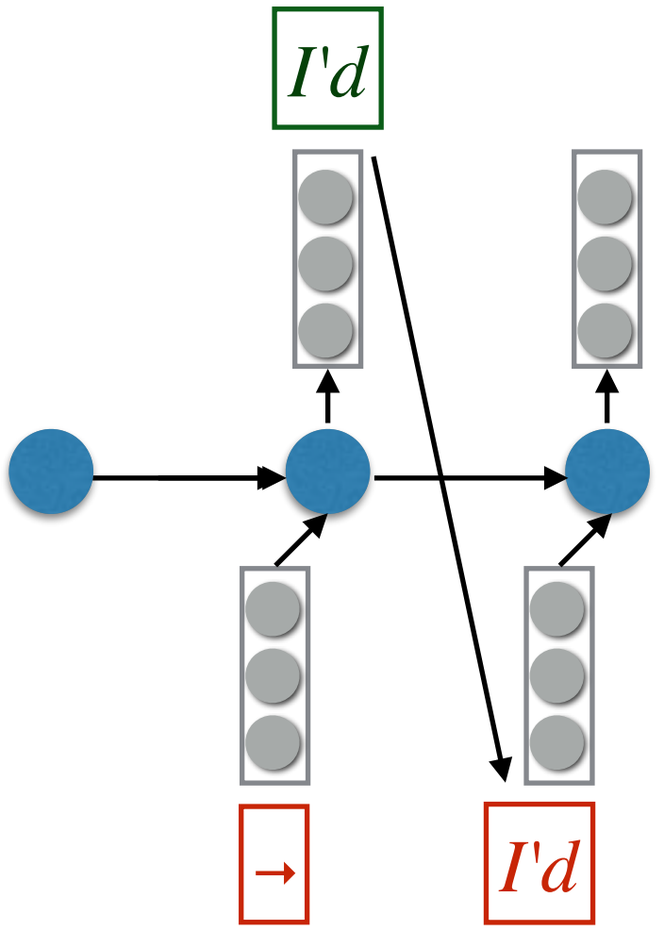
Recall RNNs…



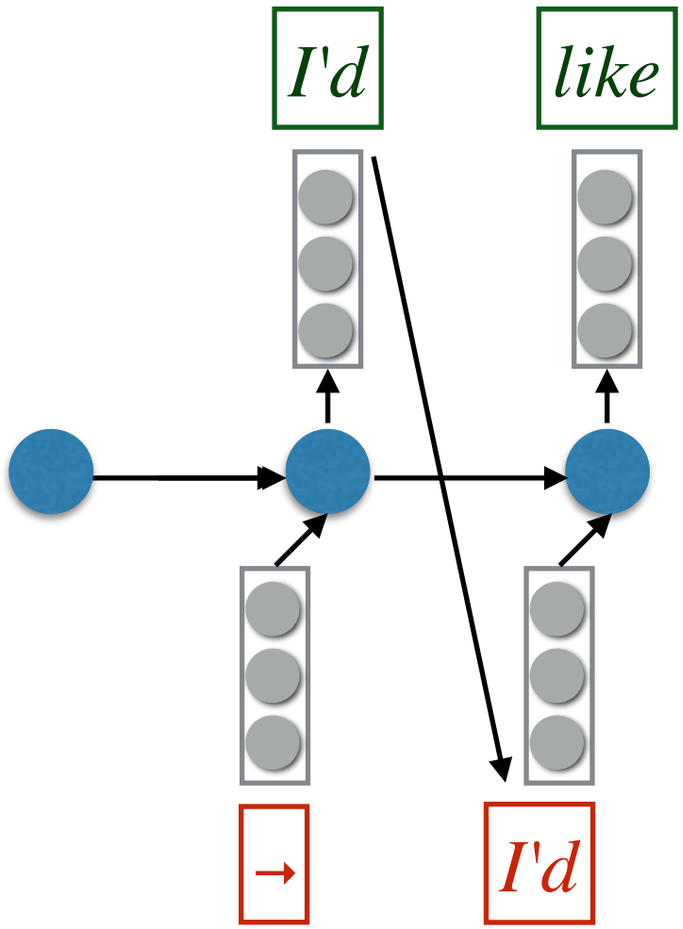
Recall RNNs…



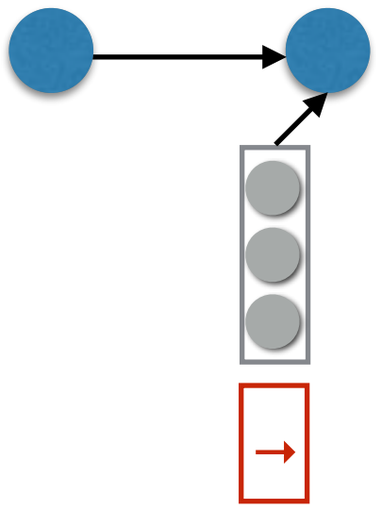
Recall RNNs…

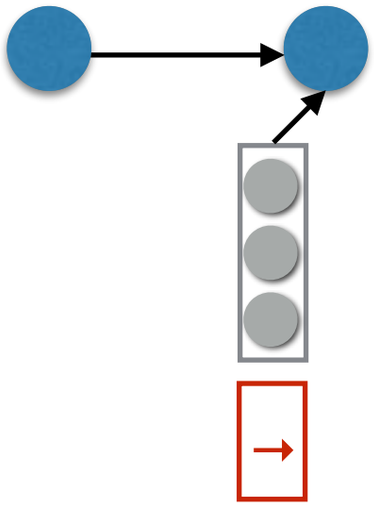


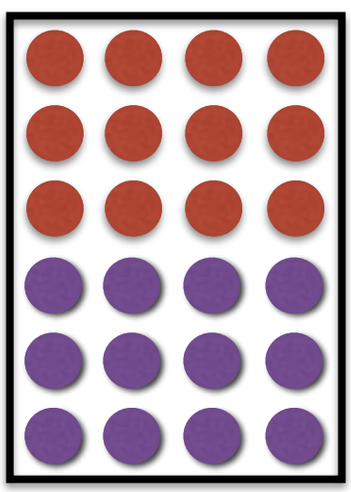
Recall RNNs…



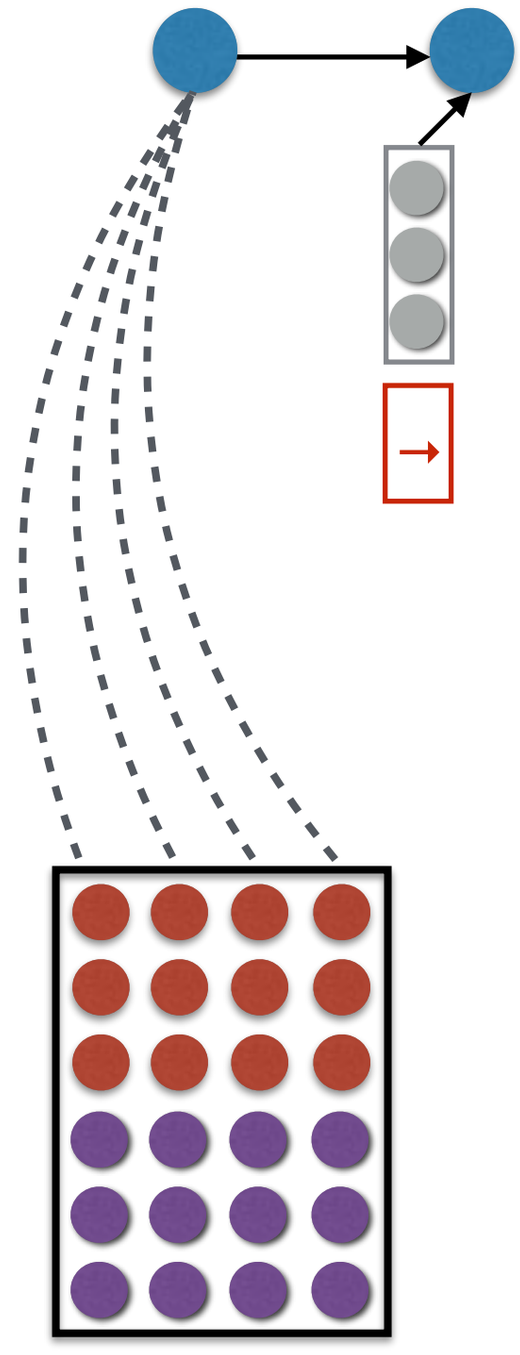
Recall RNNs…



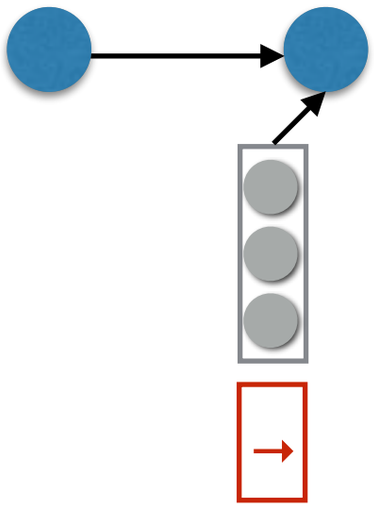


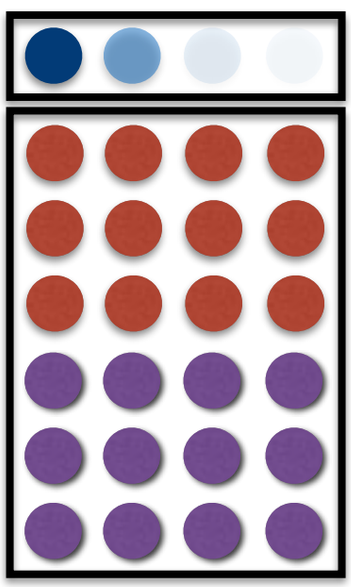


Ich möchte ein Bier



Ich möchte ein Bier

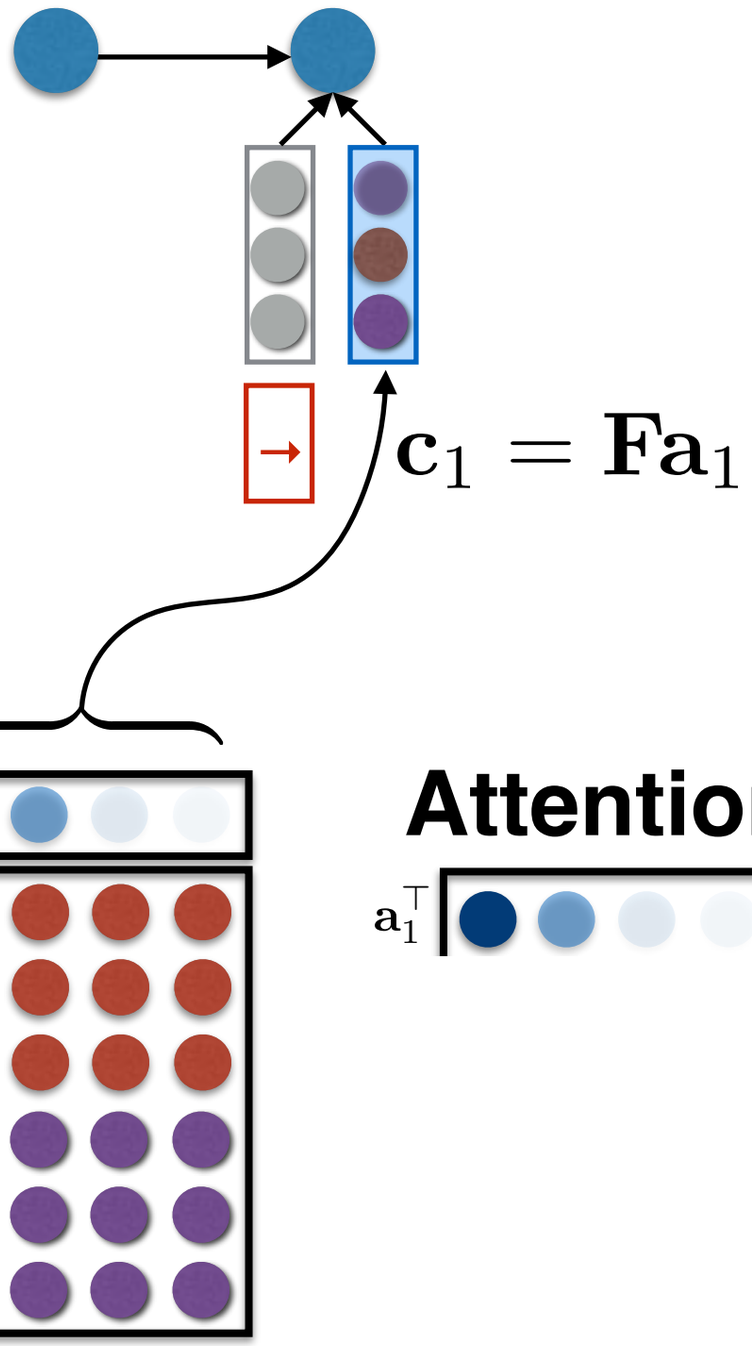


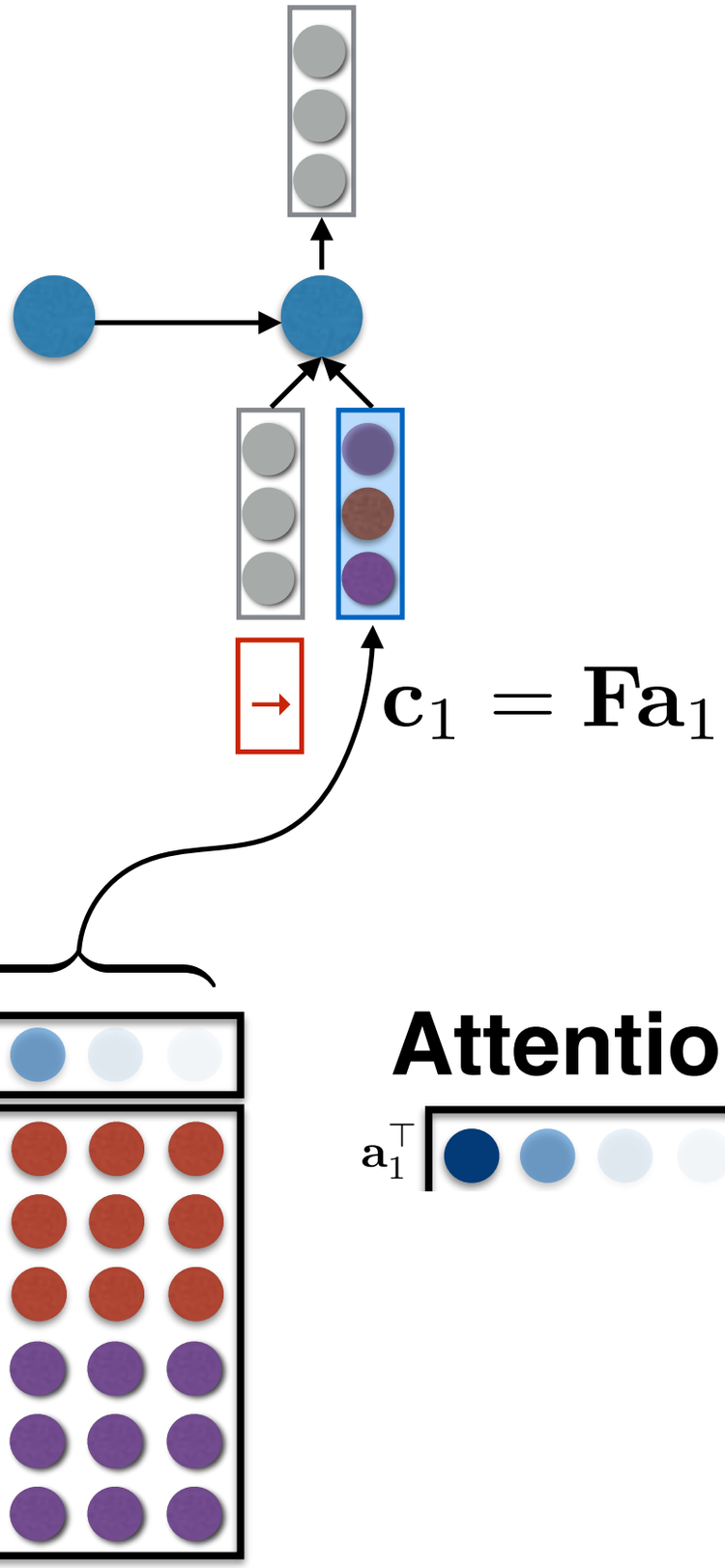


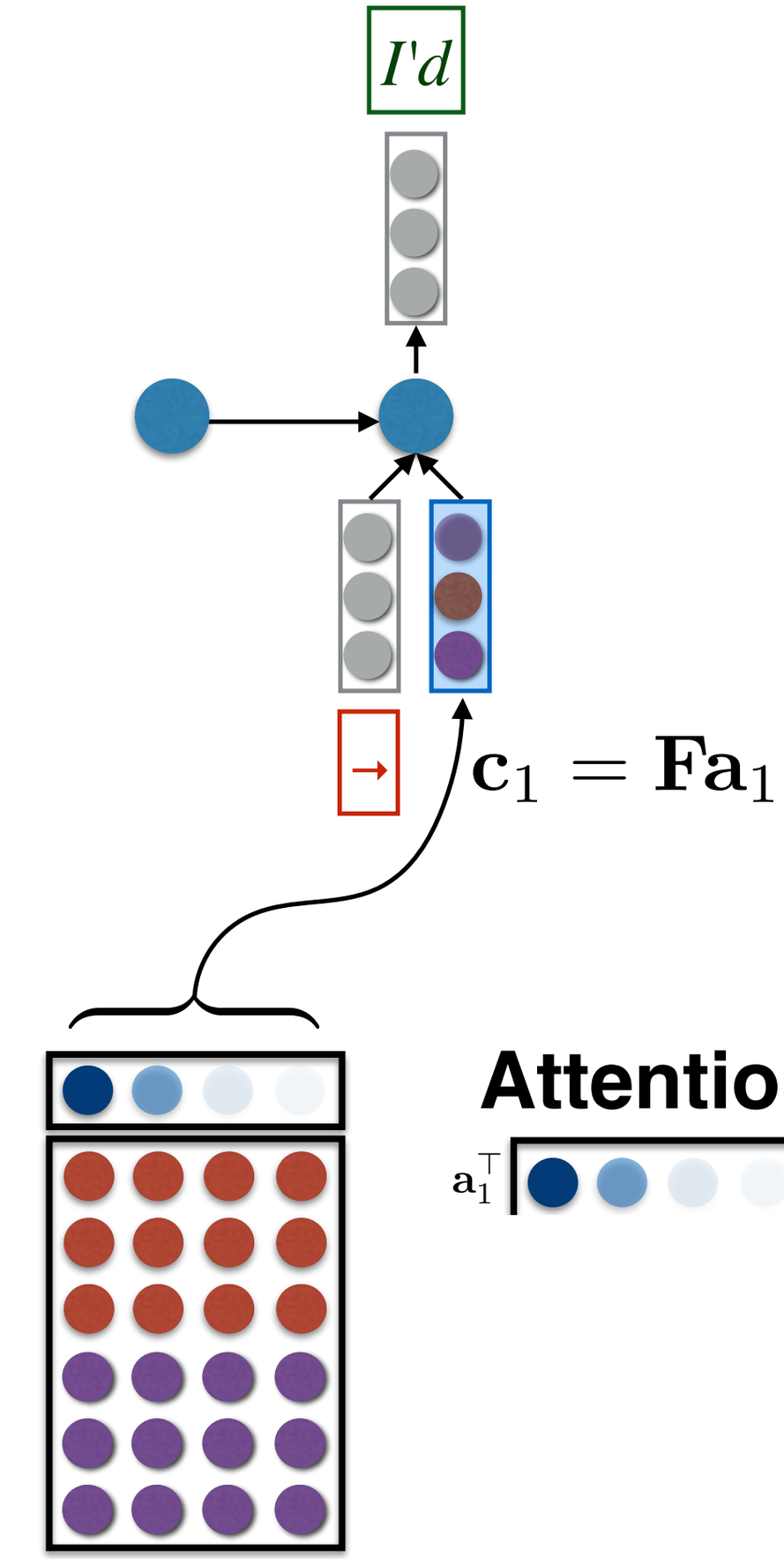
Ich möchte ein Bier

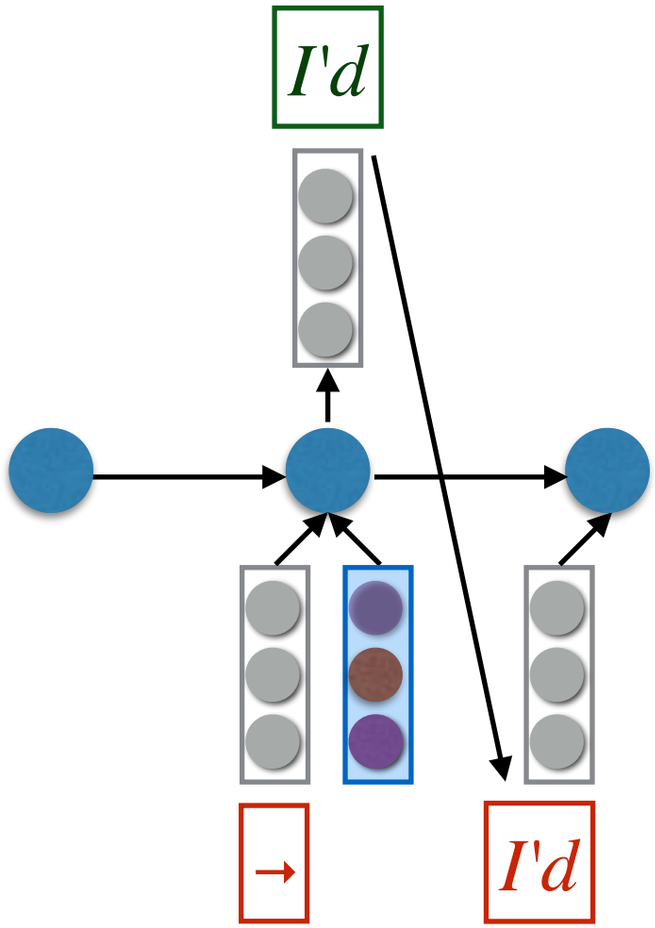
Attention history:

$\mathbf{a}_1^\top$

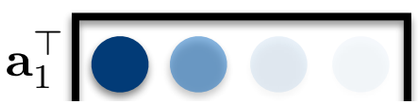

$\mathbf{c}_1 = \mathbf{F}\mathbf{a}_1$
Attention history:
$\mathbf{a}_1^\top$
Ich möchte ein Bier

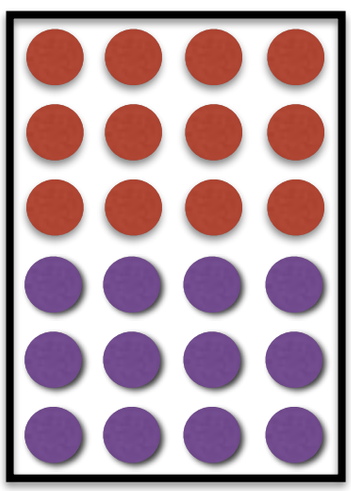

$\mathbf{c}_1 = \mathbf{F}\mathbf{a}_1$
Attention history:
$\mathbf{a}_1^\top$
Ich möchte ein Bier


I'd
→
$\mathbf{c}_1 = \mathbf{F}\mathbf{a}_1$
Attention history:
$\mathbf{a}_1^\top$
Ich möchte ein Bier

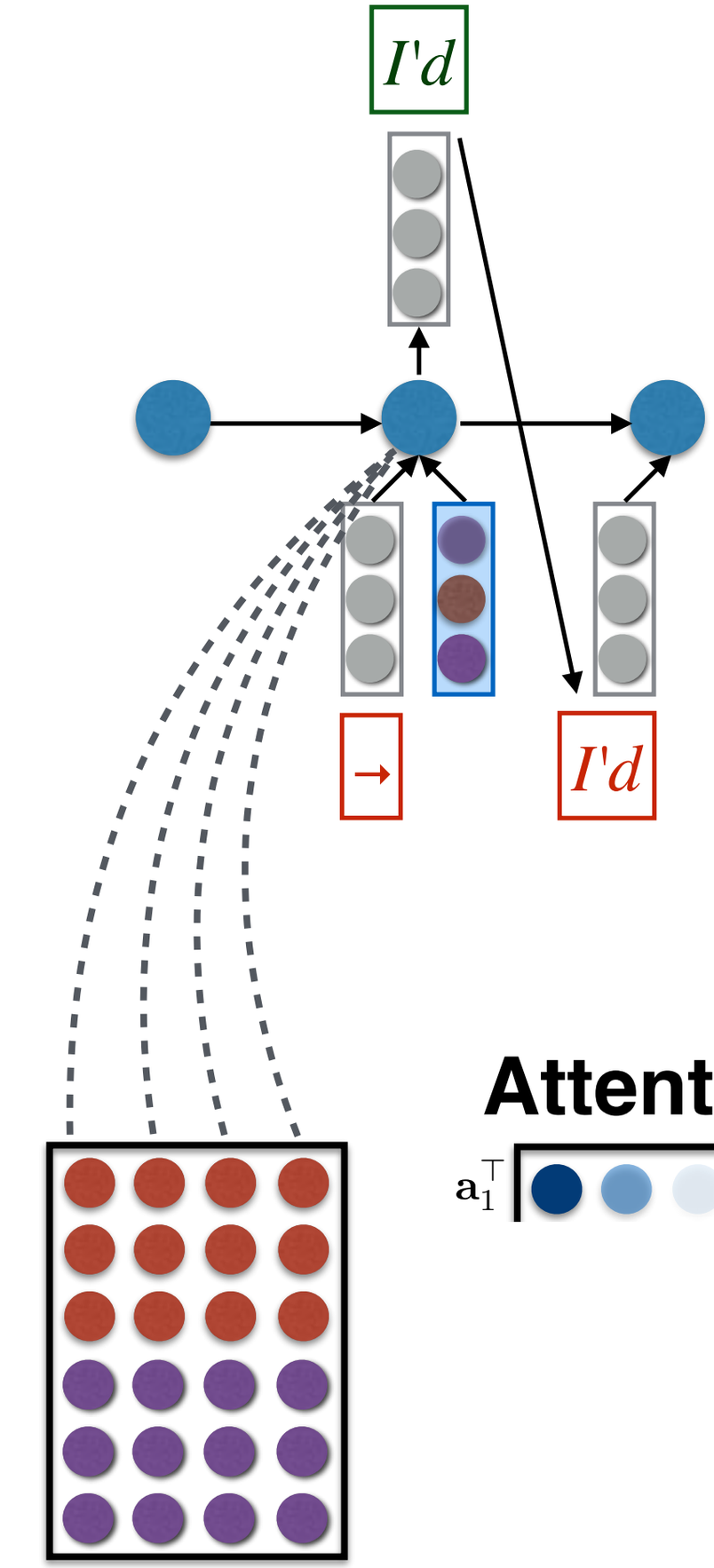


Attention history:

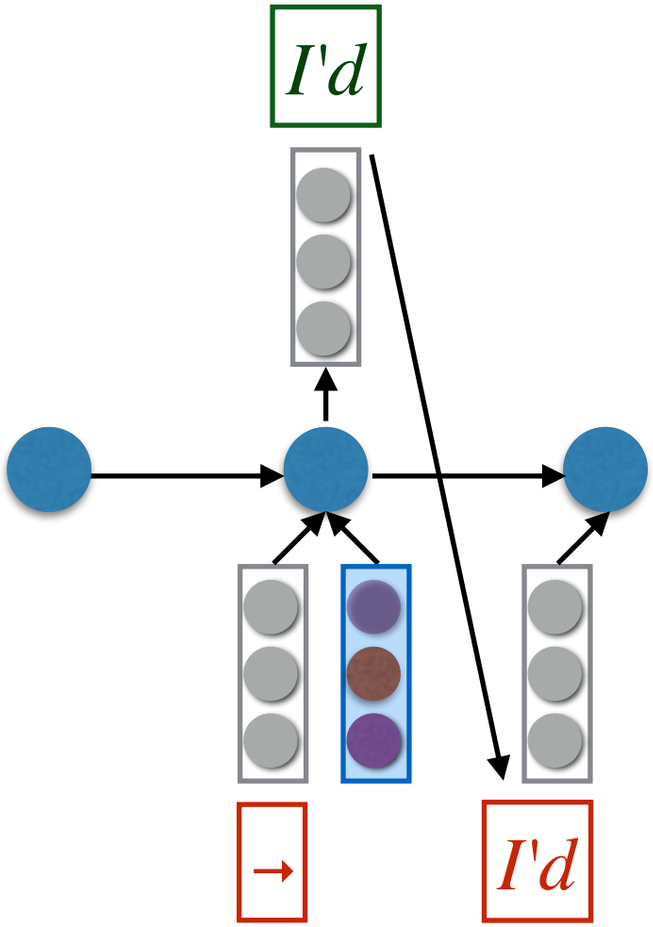


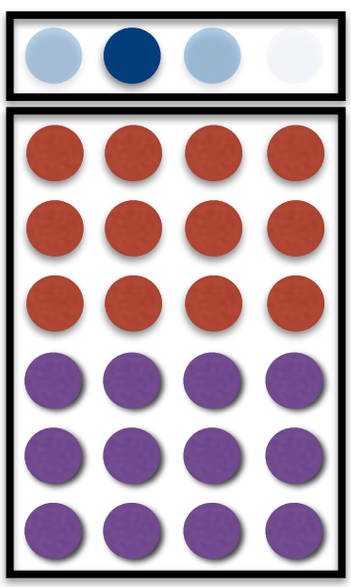


Ich möchte ein Bier



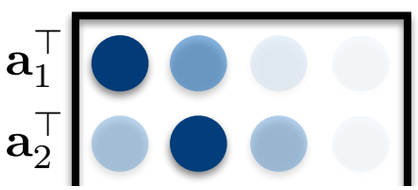
Attention history:

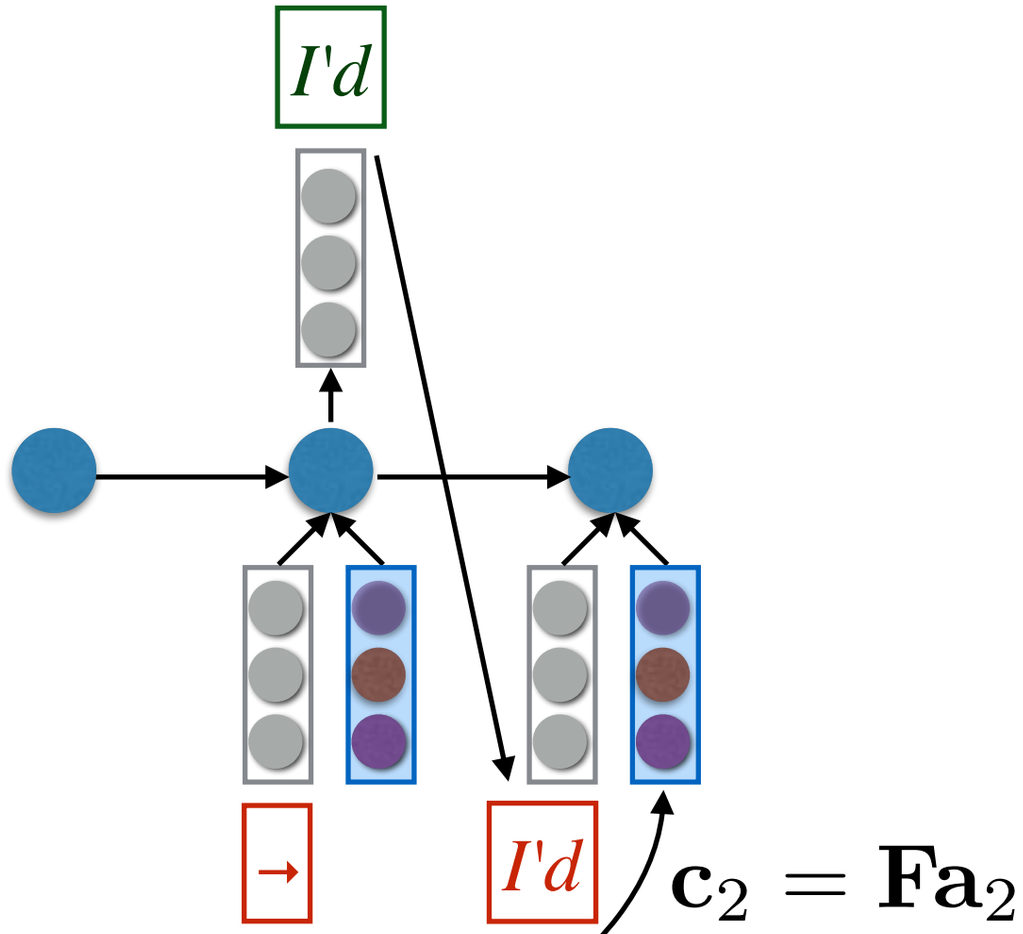


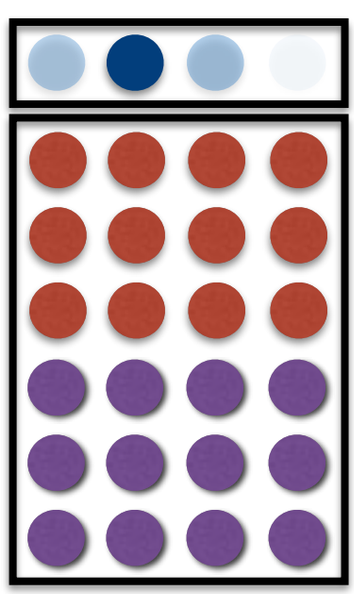


Ich möchte ein Bier

Attention history:

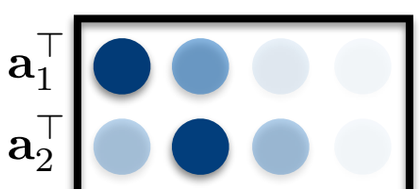


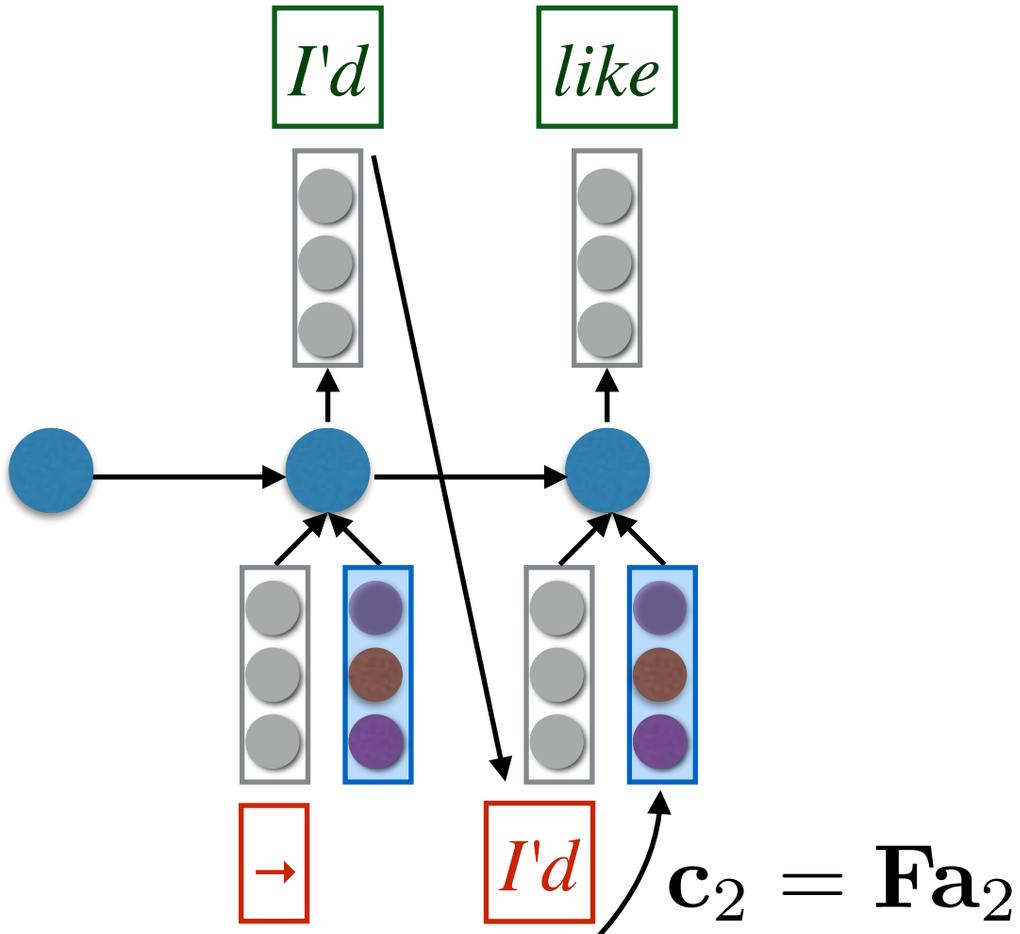


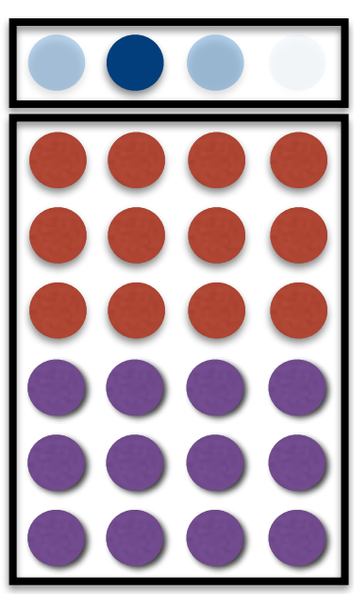


Ich möchte ein Bier

Attention history:

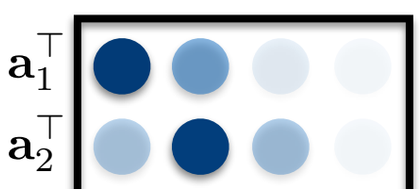


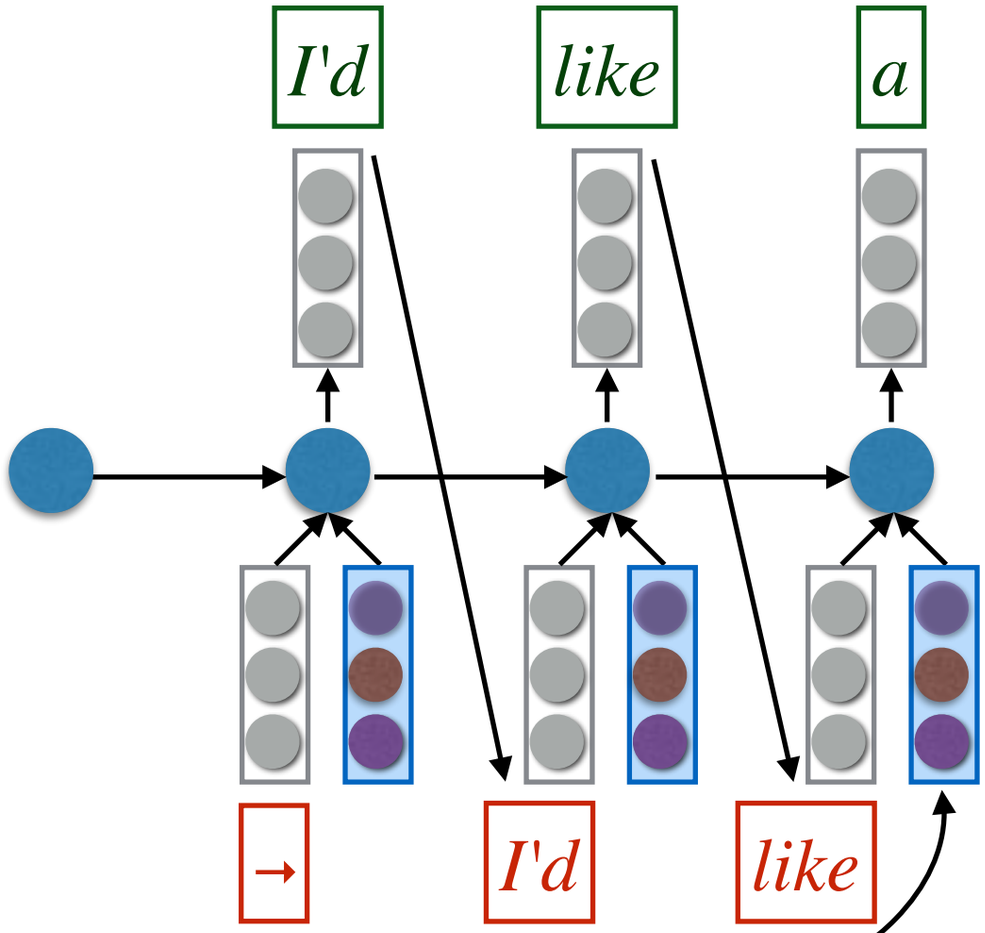


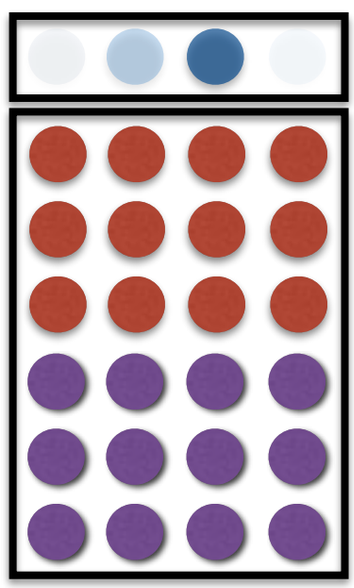


Ich möchte ein Bier

Attention history:

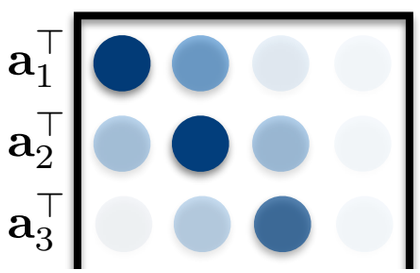


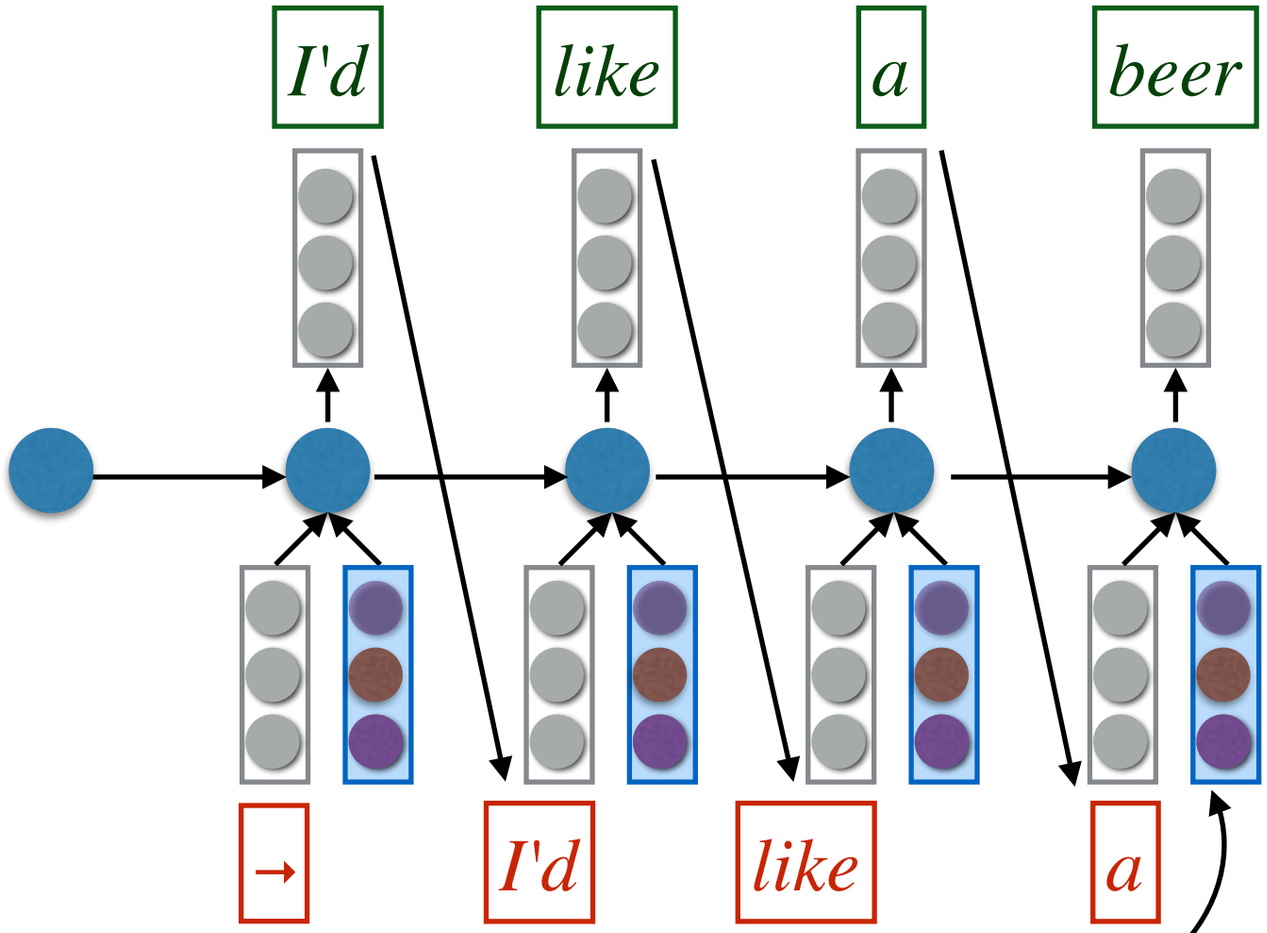


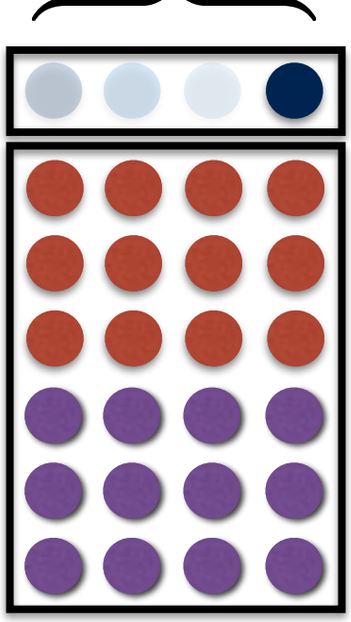


Ich möchte ein Bier

Attention history:

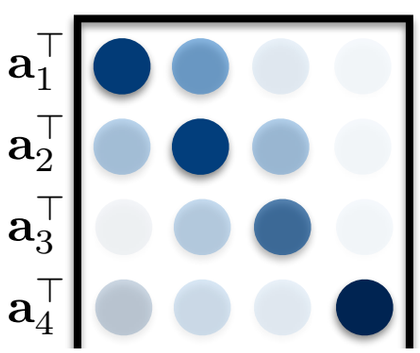


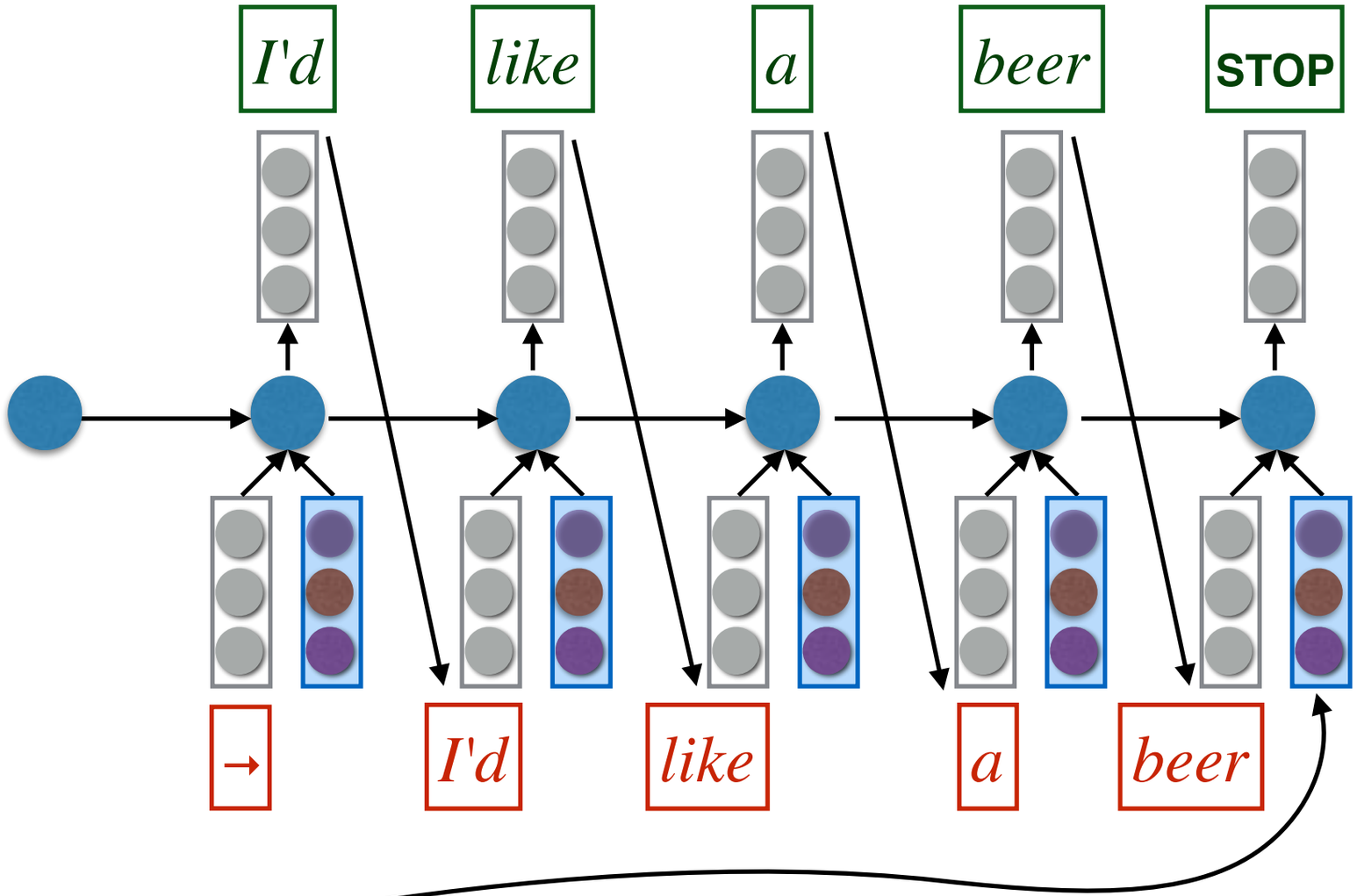


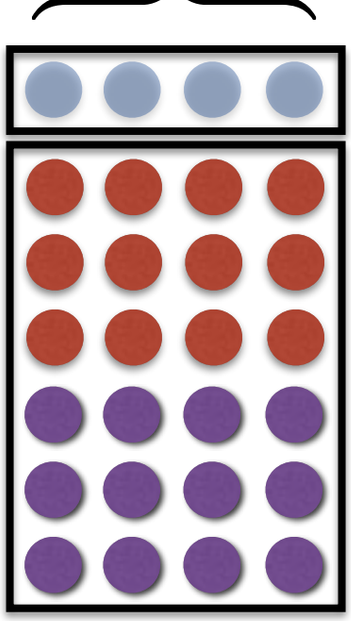


Ich möchte ein Bier

Attention history:

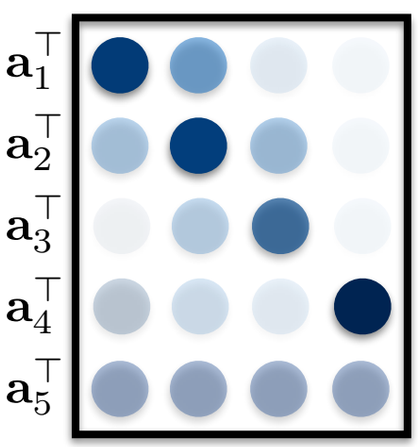






Ich möchte ein Bier

Attention history:



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 column: 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 column: 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 ***query key 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 column: 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 ***query key 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 $|\boldsymbol{f}|$ 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 column: 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 ***query key 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 column: 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 ***query key 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 $|\boldsymbol{f}|$ columns, $\mathbf{u}_t$ has $|\boldsymbol{f}|$ 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:

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

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

Nonlinear Attention-Energy Model

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

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

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

- Here, **W** and **v** are learned parameters of appropriate dimension and + "broadcasts" over the $|\boldsymbol{f}|$ 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

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

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

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

$t = 0$

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

$t = t + 1$

$\mathbf{r}_t = \mathbf{V}\mathbf{s}_{t-1}$

$\mathbf{u}_t = \mathbf{v}^\top \tanh(\mathbf{W}\mathbf{F} + \mathbf{r}_t)$

$\mathbf{a}_t = \text{softmax}(\mathbf{u}_t)$

} (Compute attention; part 2 of lecture)

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

$\mathbf{s}_t = \text{RNN}(\mathbf{s}_{t-1}, [\mathbf{e}_{t-1}; \mathbf{c}_t])$ ($\mathbf{e}_{t-1}$ 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)

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

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 $\mathbf{U}\overleftarrow{\mathbf{h}}_1$)

$t = 0$

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

$t = t + 1$

doesn't depend on output decisions

$\mathbf{r}_t = \mathbf{V}\mathbf{s}_{t-1}$

$\mathbf{u}_t = \mathbf{v}^\top \tanh(\mathbf{WF} + \mathbf{r}_t)$

$\mathbf{a}_t = \mathrm{softmax}(\mathbf{u}_t)$

} (Compute attention; part 2 of lecture)

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

$\mathbf{s}_t = \mathrm{RNN}(\mathbf{s}_{t-1}, [\mathbf{e}_{t-1}; \mathbf{c}_t])$ ($\mathbf{e}_{t-1}$ 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

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

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

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

$t = 0$

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

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

$t = t + 1$

$\mathbf{r}_t = \mathbf{V}\mathbf{s}_{t-1}$

$\mathbf{u}_t = \mathbf{v}^\top \tanh(\mathbf{WF} + \mathbf{r}_t)$

$\mathbf{a}_t = \text{softmax}(\mathbf{u}_t)$

} (Compute attention; part 2 of lecture)

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

$\mathbf{s}_t = \text{RNN}(\mathbf{s}_{t-1}, [\mathbf{e}_{t-1}; \mathbf{c}_t])$ ($\mathbf{e}_{t-1}$ 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)

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

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 $\mathbf{U}\overleftarrow{\mathbf{h}}_1$)

$t = 0$

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

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

$t = t + 1$

$\mathbf{r}_t = \mathbf{V}\mathbf{s}_{t-1}$

$\mathbf{u}_t = \mathbf{v}^\top \tanh(\mathbf{X} + \mathbf{r}_t)$

$\mathbf{a}_t = \text{softmax}(\mathbf{u}_t)$

} (Compute attention; part 2 of lecture)

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

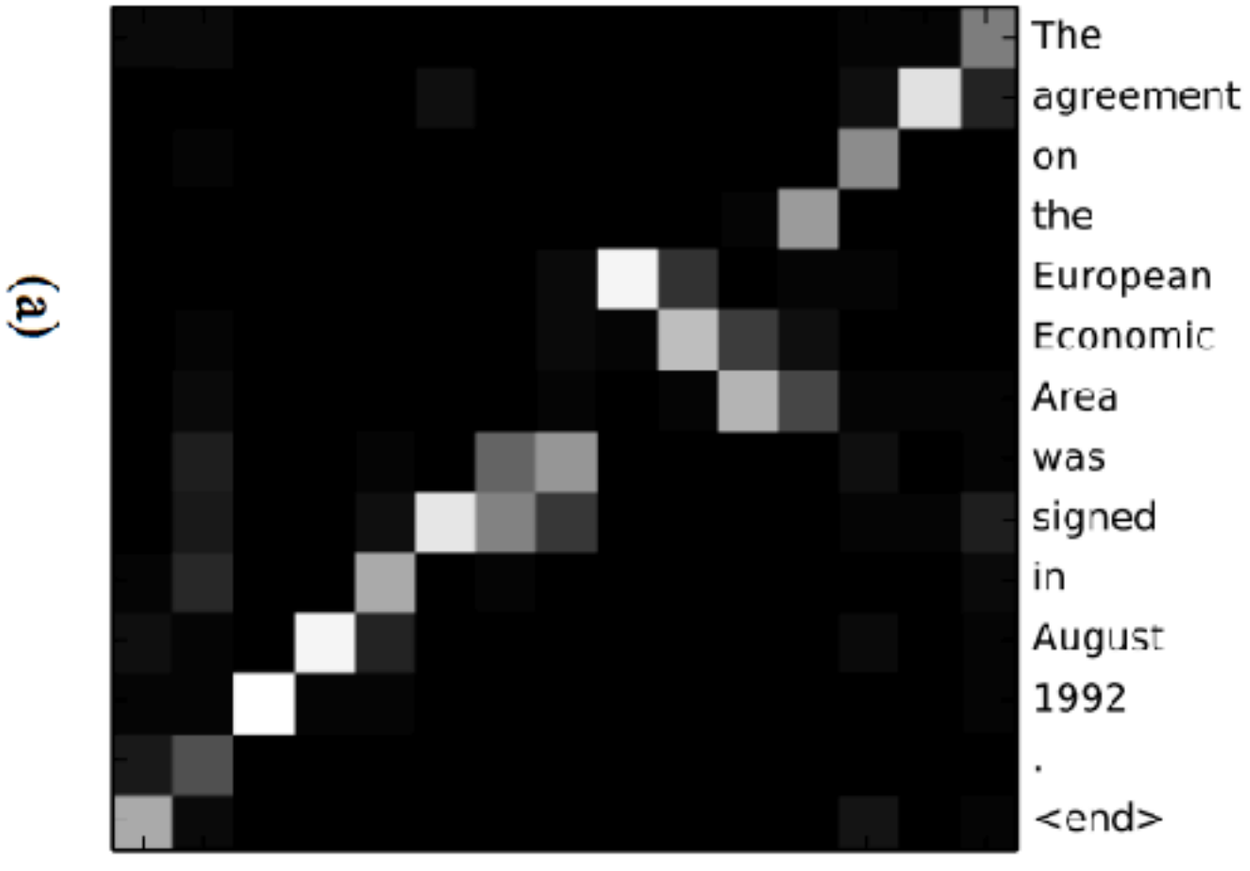
$\mathbf{s}_t = \text{RNN}(\mathbf{s}_{t-1}, [\mathbf{e}_{t-1}; \mathbf{c}_t])$ ($\mathbf{e}_{t-1}$ is a learned embedding of e_t)

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

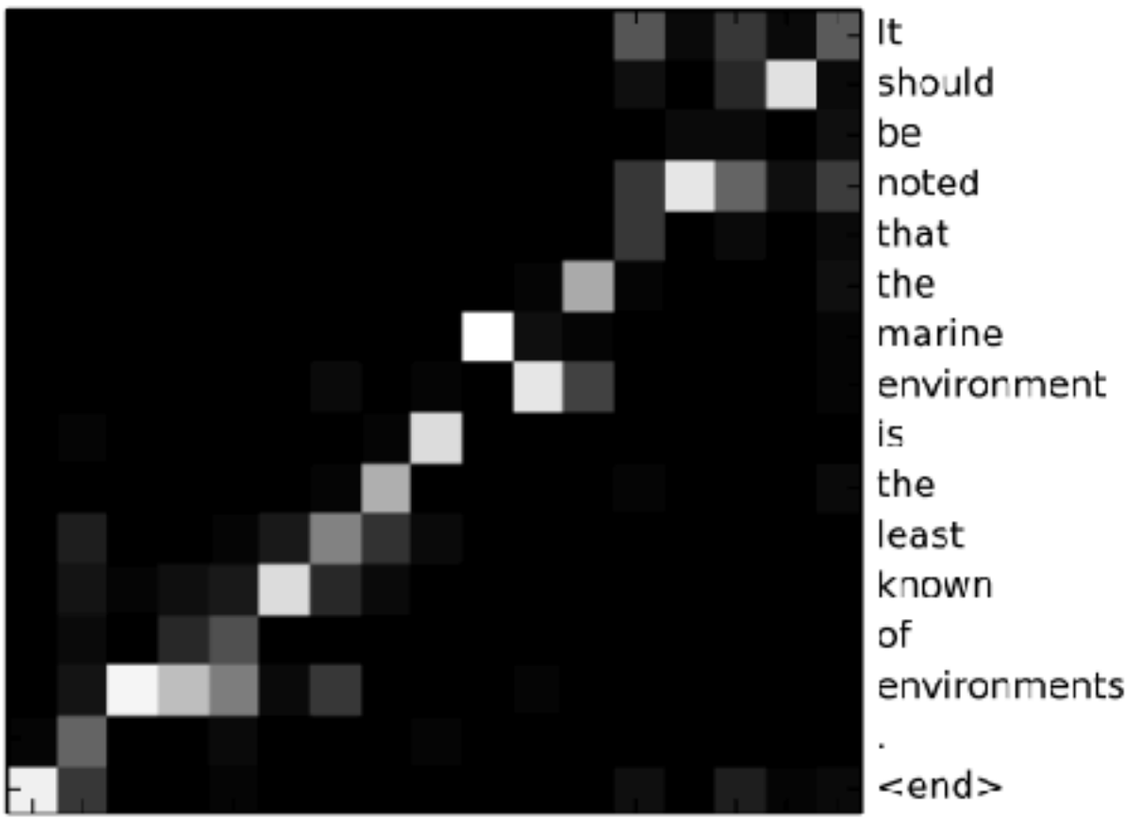
$e_t \mid \boldsymbol{e}_{<t} \sim \text{Categorical}(\mathbf{y}_t)$

Attention in MT

Add attention to seq2seq translation: **+11 BLEU**

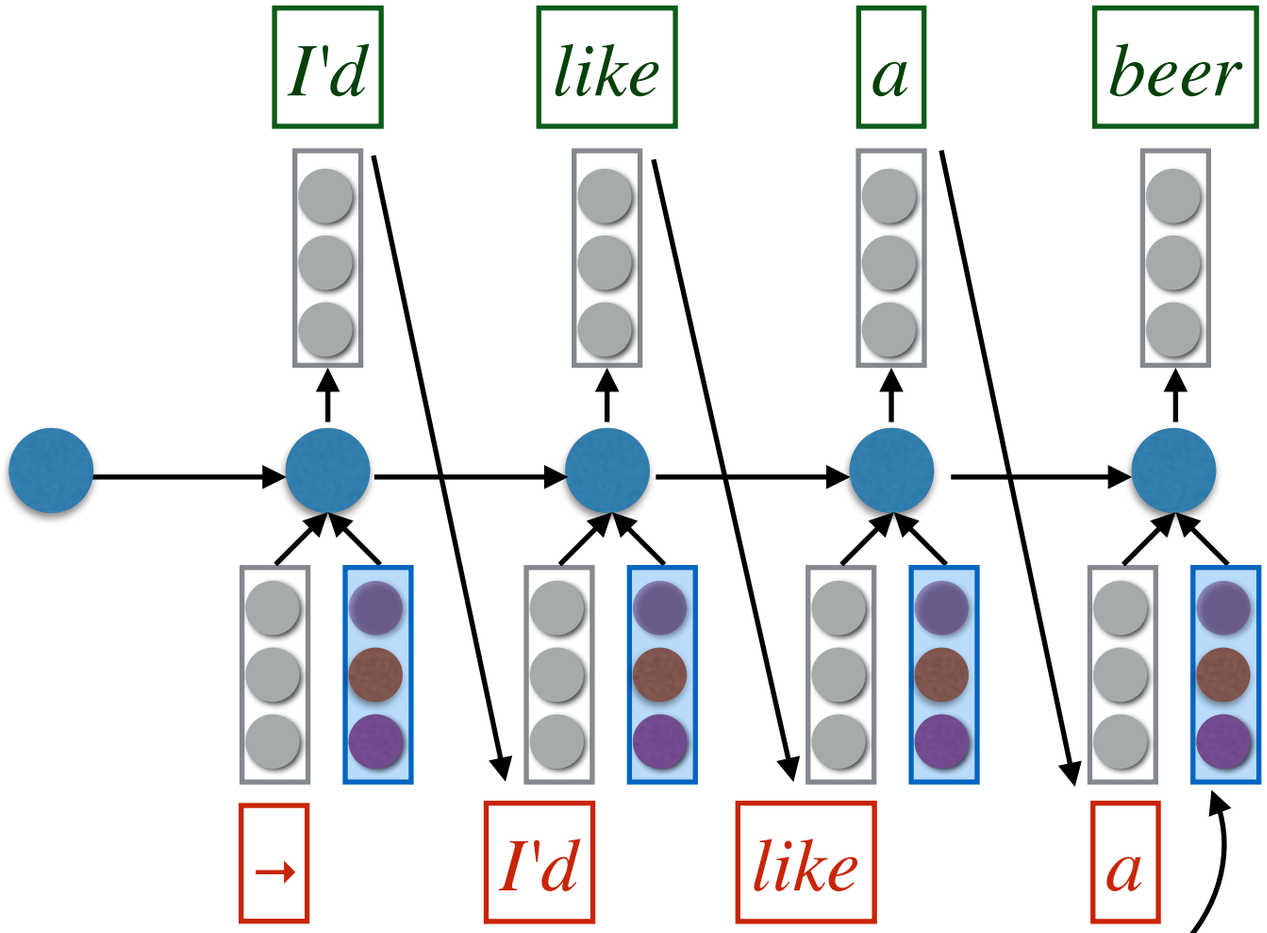

L'
accord
sur
la
zone
économique
européenne
a
été
signé
en
août
1992
.
<end>
The
agreement
on
the
European
Economic
Area
was
signed
in
August
1992
.
<end>

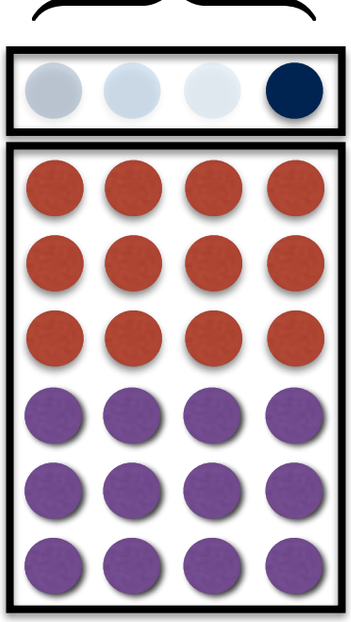
(a)


Il
convient
de
noter
que
l'
environnement
marin
est
le
moins
connu
de
l'
environnement
.
<end>
It
should
be
noted
that
the
marine
environment
is
the
least
known
of
environments
.
<end>

(b)

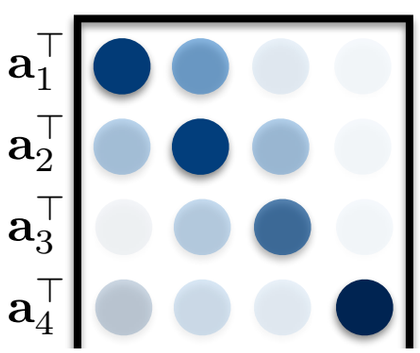
A word about gradients

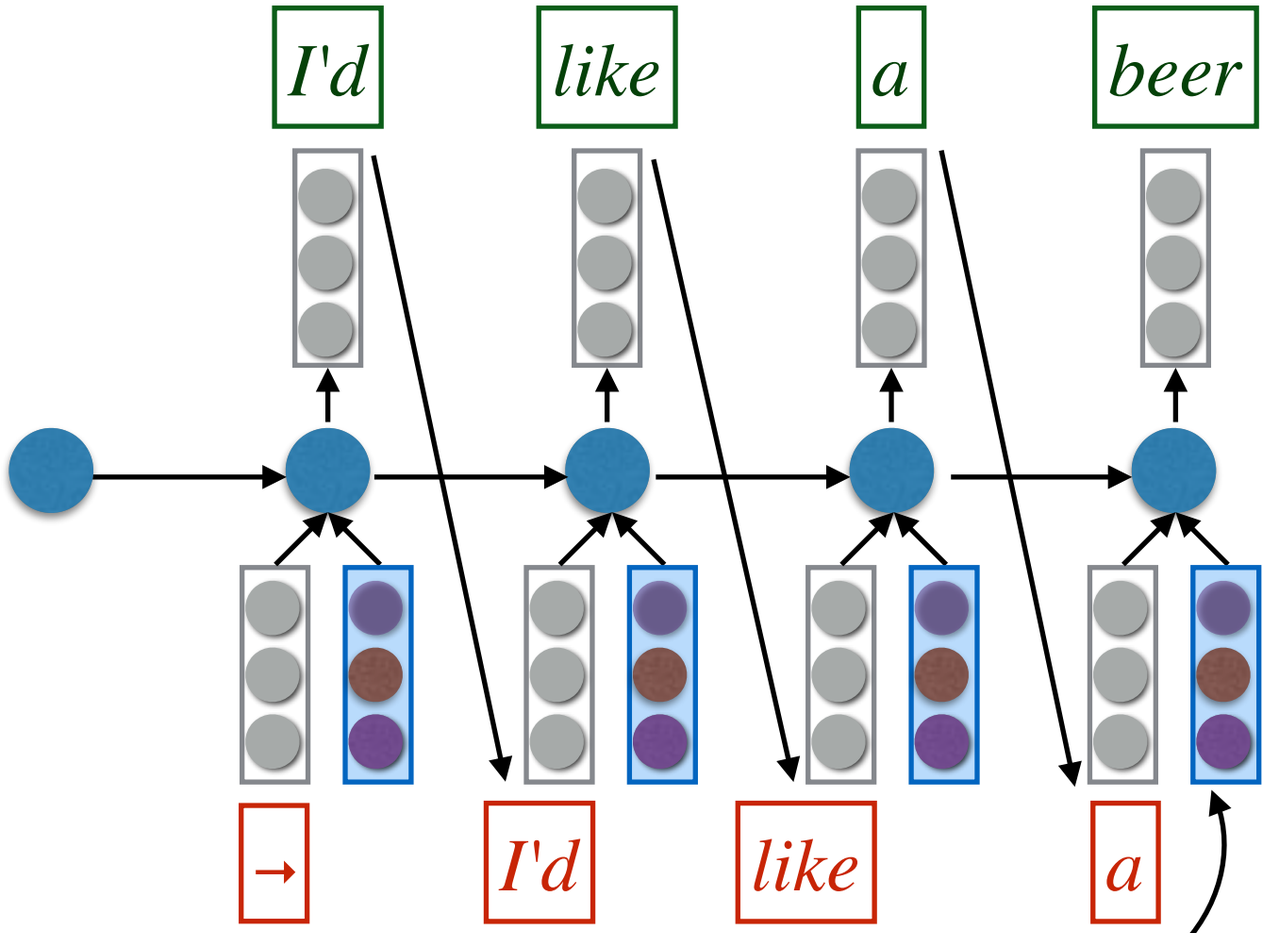


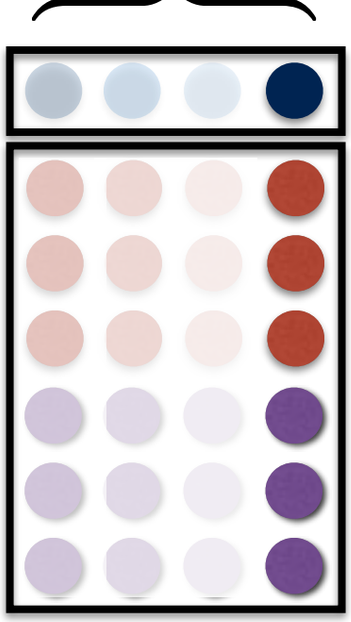


Ich möchte ein Bier

Attention history:

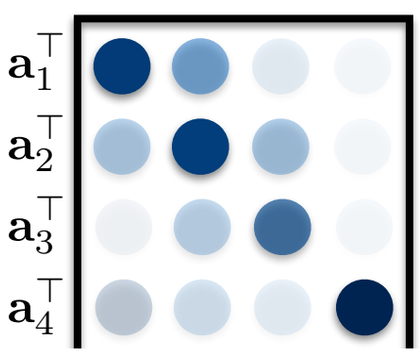


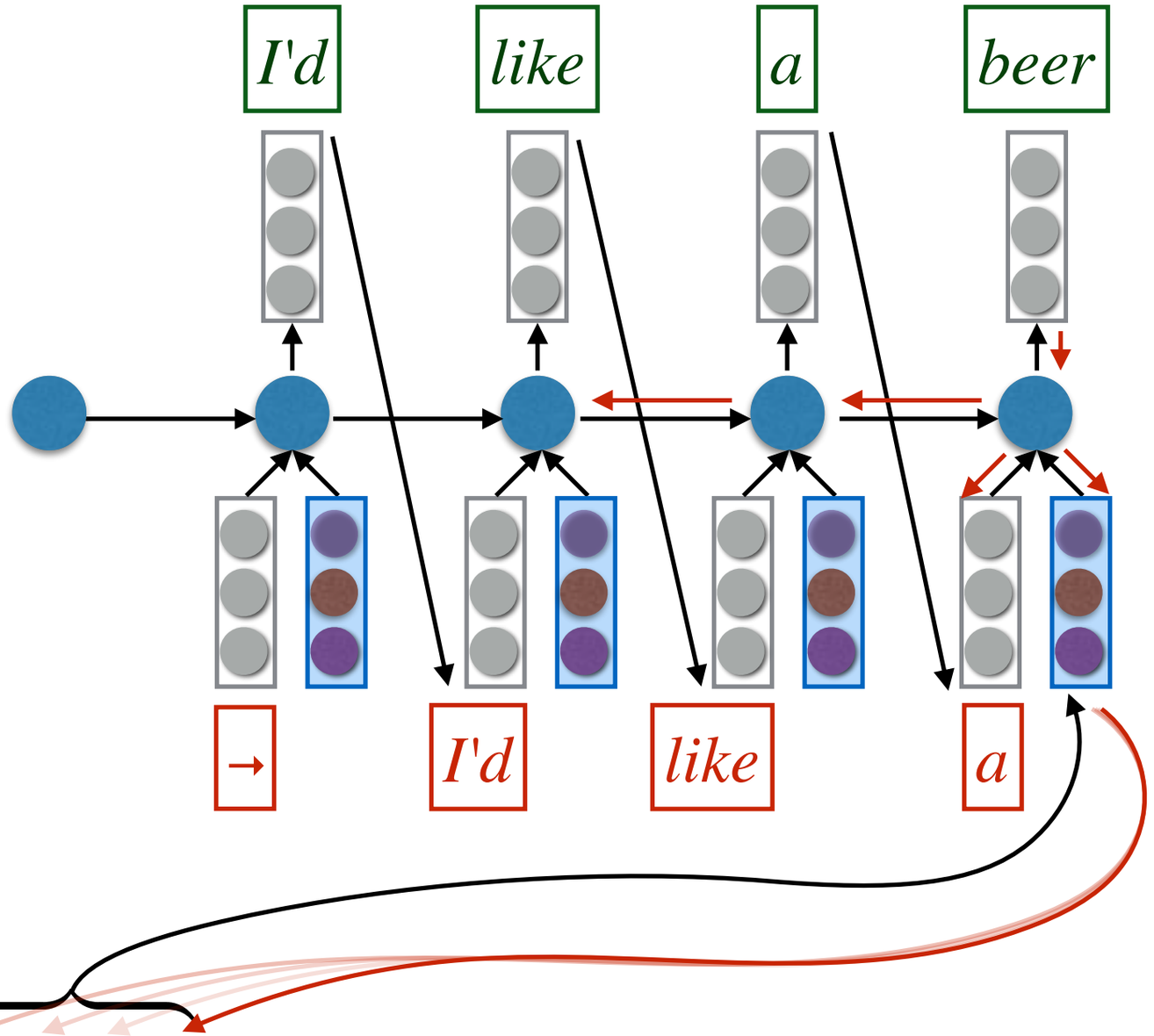


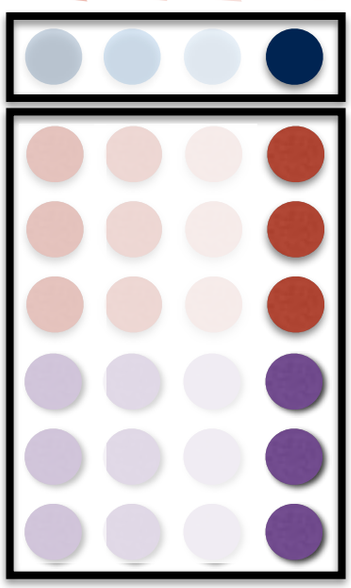


Ich möchte ein Bier

Attention history:

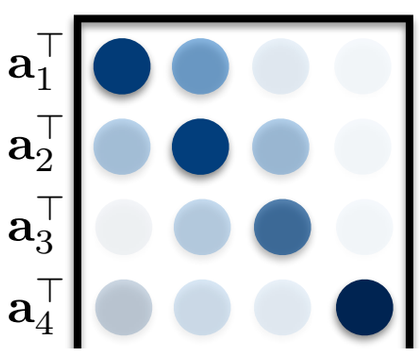






Ich möchte ein Bier

Attention history:



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?

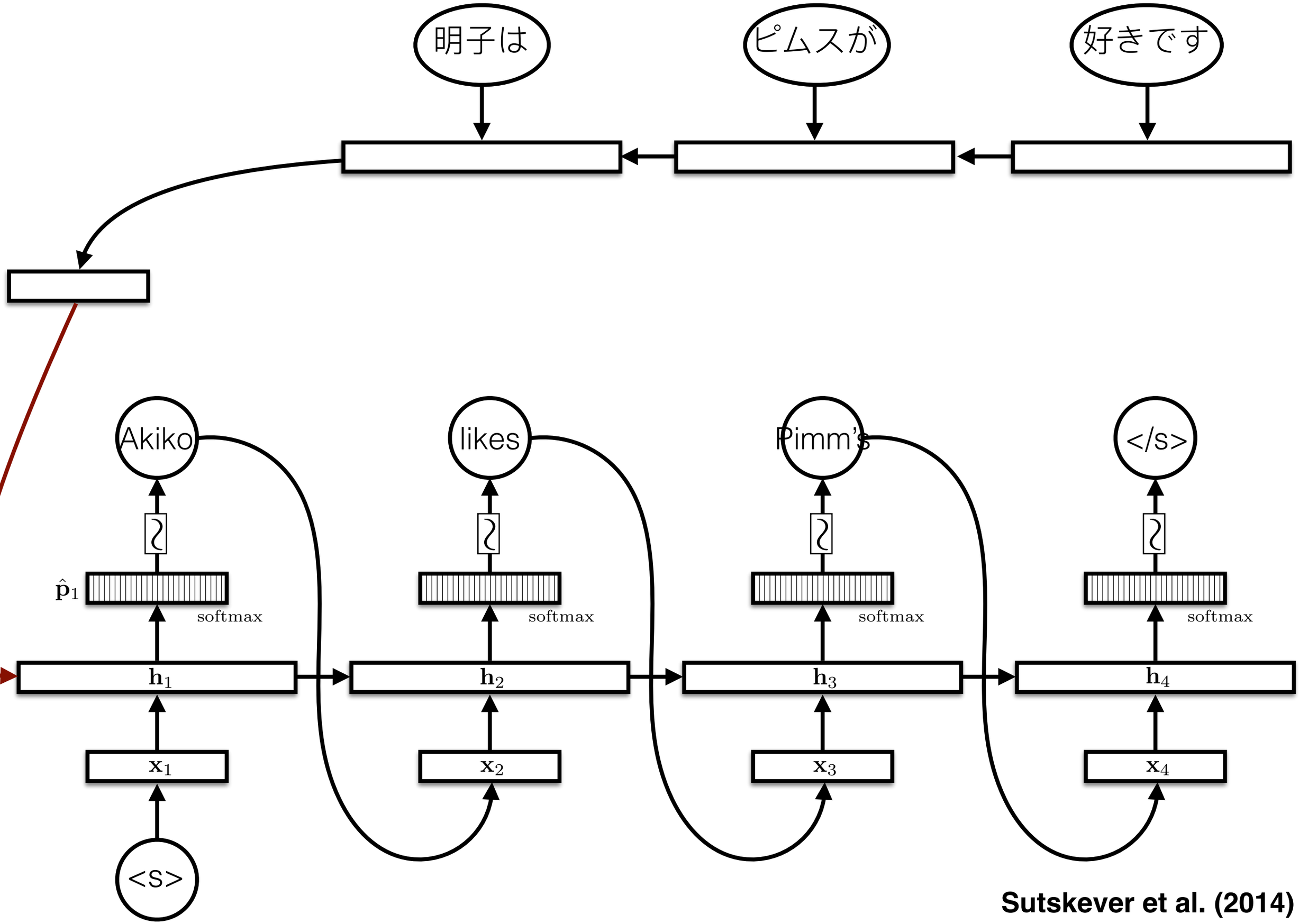
Summary

- Attention
  - provides the ability to establish information flow directly from distant
  - closely related to “pooling” operations in convnets (and other architectures)
- Traditional 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
  - Factorization into keys and values (Miller et al., 2016; Ba et al., 2016, Gulcehre et al., 2016)
- Attention weights provide interpretation you can look at

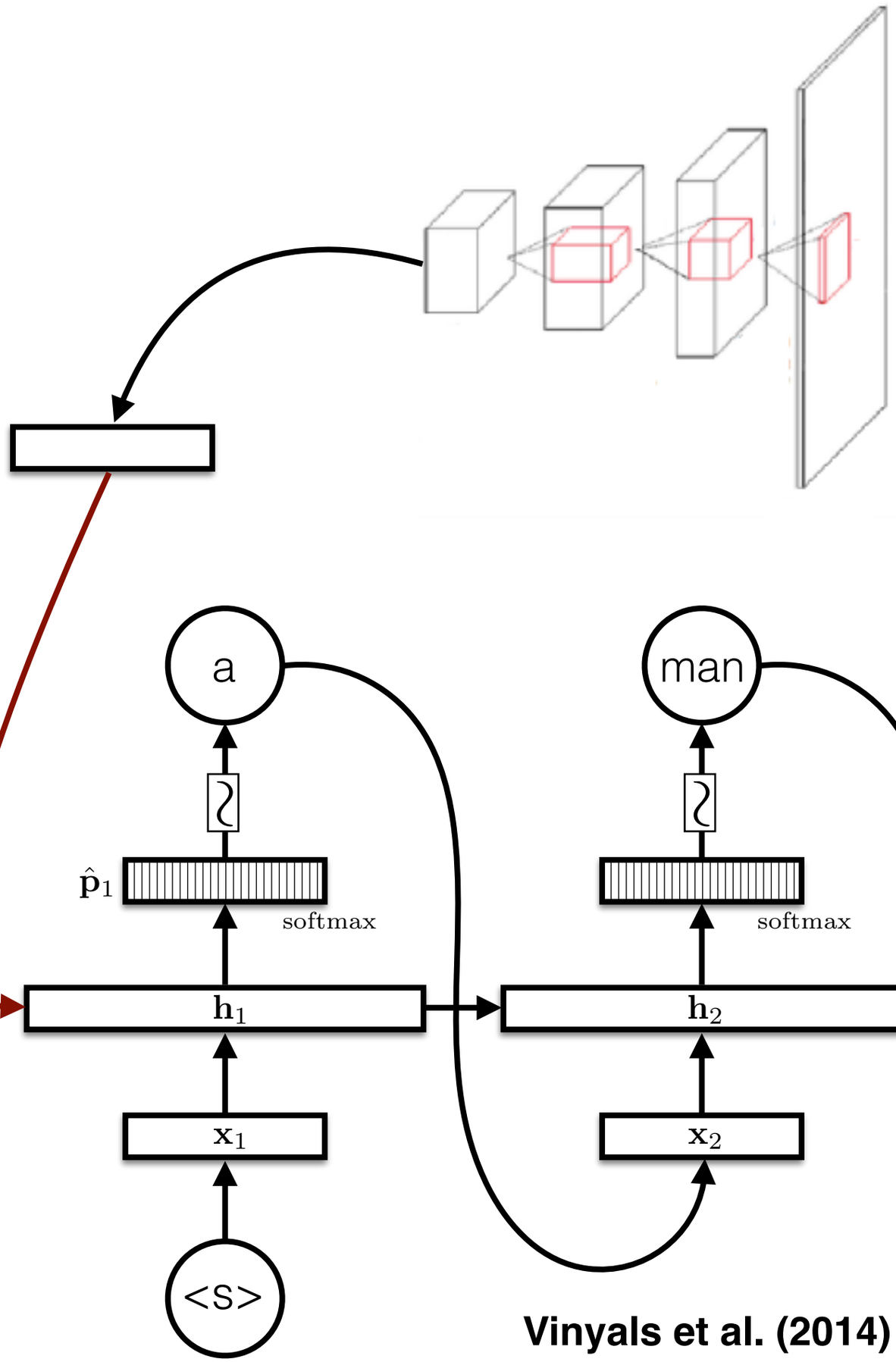
Outline of Lecture

- Machine translation with attention

- Image caption generation with attention



Sutskever et al. (2014)



Vinyals et al. (2014) Show and Tell: A Neural Image Caption Generator

Image Caption Generation

- Can attention help caption modeling?

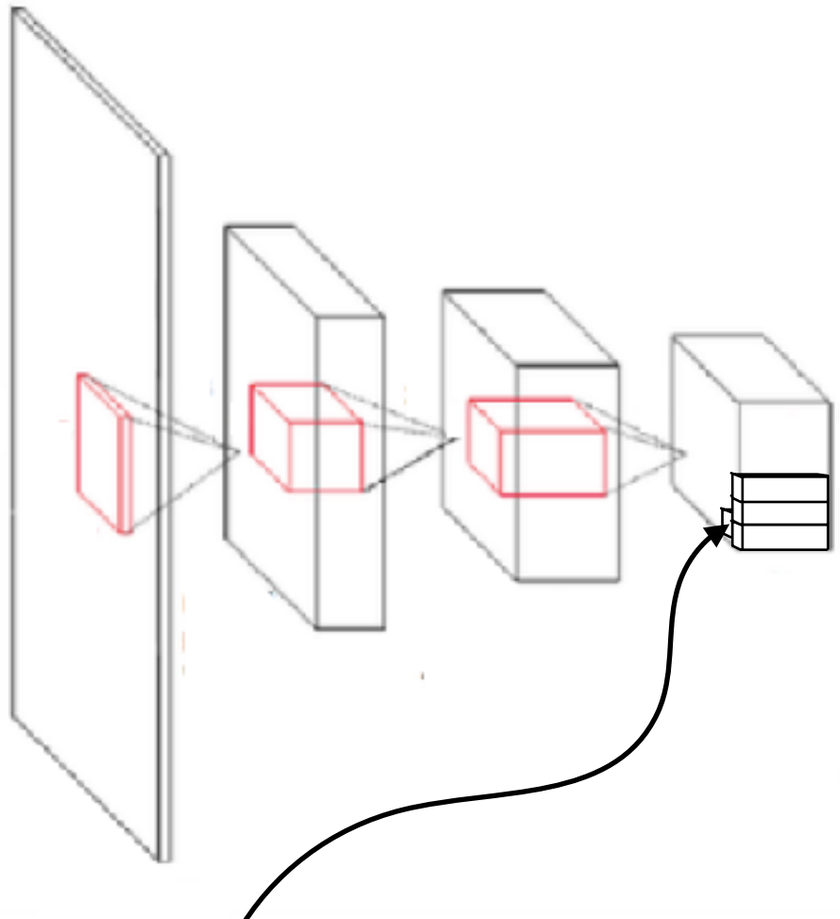
Show, Attend and Tell: Neural Image Caption Generation with Visual Attention

---

Kelvin Xu KELVIN.XU@UMONTREAL.CA
Jimmy Lei Ba JIMMY@PSI.UTORONTO.CA
Ryan Kiros RKIROS@CS.TORONTO.EDU
Kyunghyun Cho KYUNGHYUN.CHO@UMONTREAL.CA
Aaron Courville AARON.COURVILLE@UMONTREAL.CA
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Richard S. Zemel ZEMEL@CS.TORONTO.EDU
Yoshua Bengio FIND-ME@THE.WEB

Xu et al. (2015, ICML)

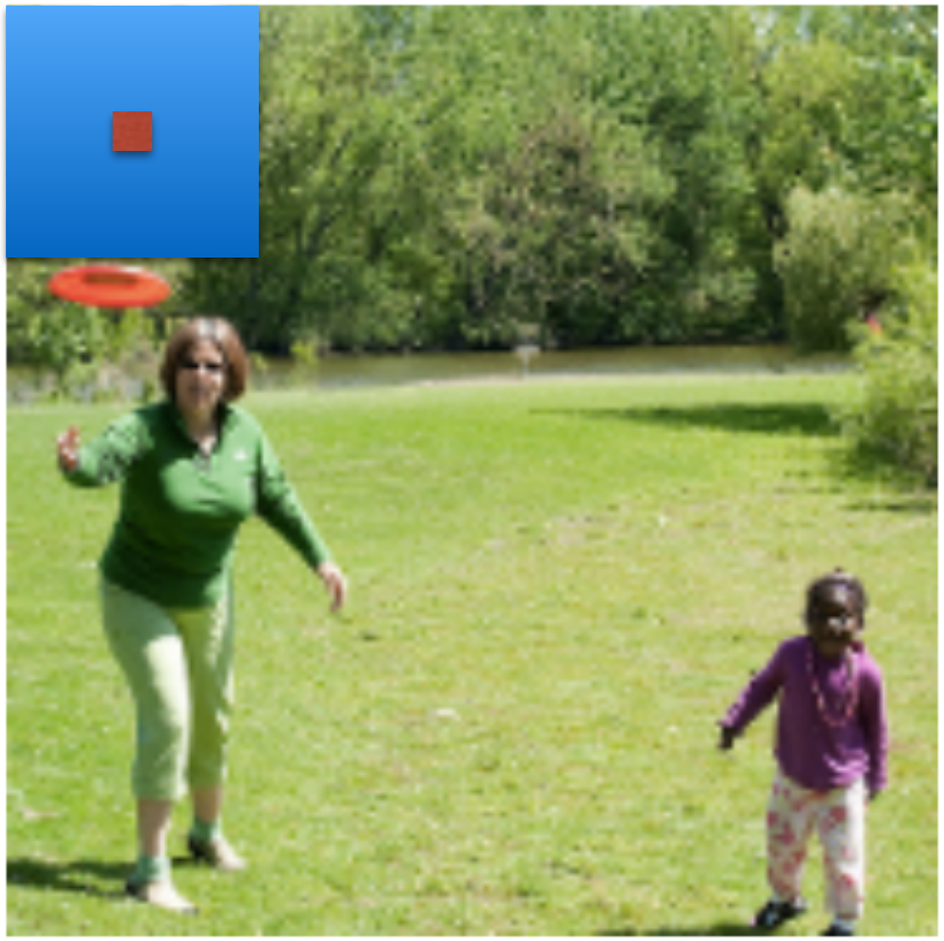
Regions in ConvNets



Each point in a “higher” level of a convnet
defines spatially localised feature vectors(/matrices).

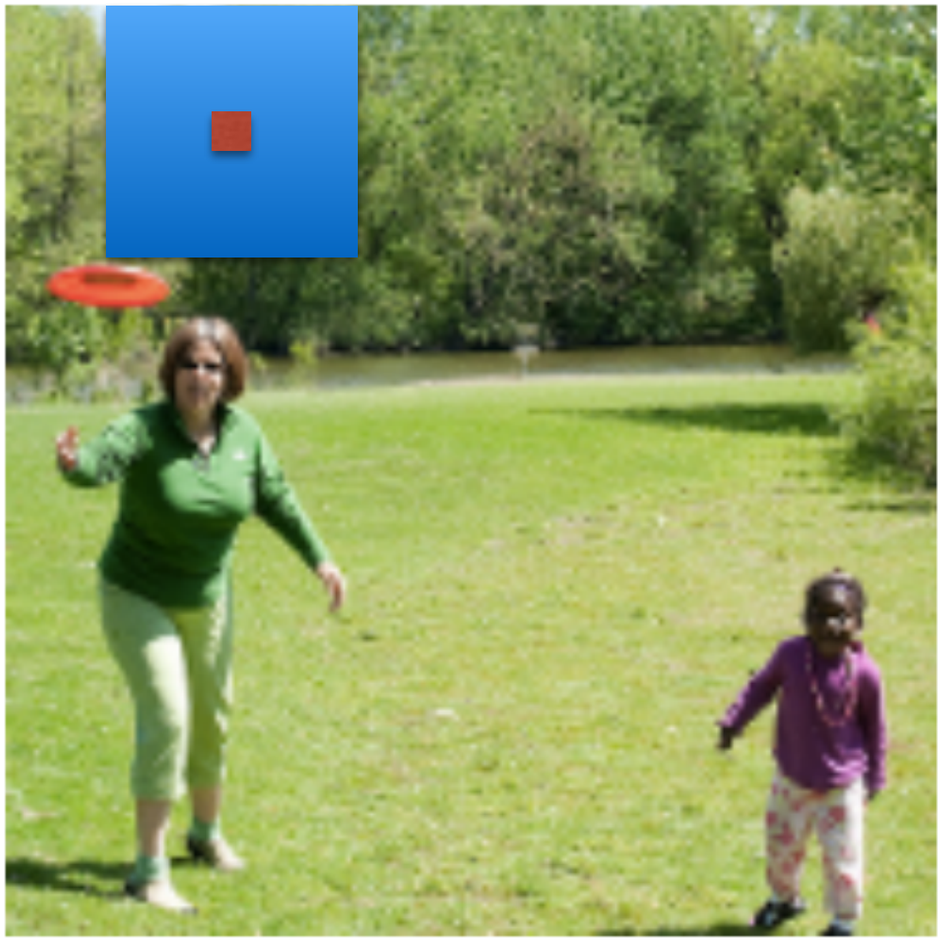
Xu et al. calls these “annotation vectors”, $\mathbf{a}_i,\ i \in \{1, \ldots, L\}$

$\mathbf{a}_1$



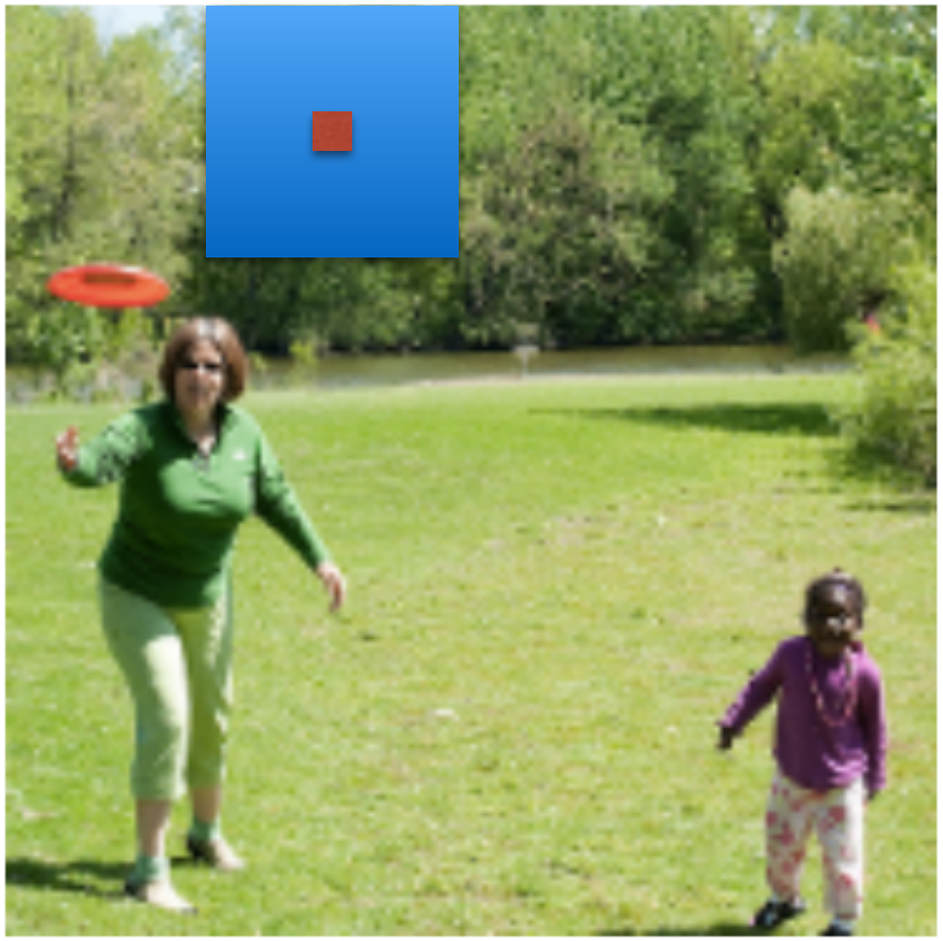
$$\mathbf{F} = \begin{bmatrix} | \\ \mathbf{a}_1 \\ | \end{bmatrix}$$

$\mathbf{a}_2$



$$\mathbf{F} = \begin{bmatrix} | & | \\ \mathbf{a}_1 & \mathbf{a}_2 \\ | & | \end{bmatrix}$$

$\mathbf{a}_3$



$$\mathbf{F} = \begin{bmatrix} | & | & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 & \cdots \\ | & | & | \end{bmatrix}$$

Attention

- Attention "weights" ($\mathbf{a}_t$) are computed using exactly the same technique as discussed above

Attention

- Attention "weights" ($\mathbf{a}_t$) are computed using exactly the same technique as discussed above

- Deterministic soft attention (Bahdanau et al., 2014)

$$\mathbf{c}_t = \mathbf{F}\mathbf{a}_t$$ (weighted average)

Attention

- Attention "weights" ($\mathbf{a}_t$) are computed using exactly the same technique as discussed above

- Deterministic soft attention (Bahdanau et al., 2014)

$$\mathbf{c}_t = \mathbf{F}\mathbf{a}_t \qquad \text{(weighted average)}$$

- Stochastic hard attention (Xu et al., 2015)

$$s_t \sim \text{Categorical}(\mathbf{a}_t)$$

$$\mathbf{c}_t = \mathbf{F}_{:,s_t} \qquad \text{(sample a column)}$$

  - What are the benefits of this model?

  - What are the challenges of learning the parameters of this model?

Learning Hard Attention

$$\mathcal{L} = -\log p(\boldsymbol{w} \mid \boldsymbol{x})$$

$$= -\log \sum_{\boldsymbol{s}} p(\boldsymbol{w}, \boldsymbol{s} \mid \boldsymbol{x})$$

$$= -\log \sum_{\boldsymbol{s}} p(\boldsymbol{s} \mid \boldsymbol{x}) p(\boldsymbol{w} \mid \boldsymbol{x}, \boldsymbol{s})$$

Learning Hard Attention

$$\begin{aligned}
\mathcal{L} &= -\log p(\boldsymbol{w} \mid \boldsymbol{x}) \\
&= -\log \sum_{\boldsymbol{s}} p(\boldsymbol{w}, \boldsymbol{s} \mid \boldsymbol{x}) \\
&= -\log \sum_{\boldsymbol{s}} p(\boldsymbol{s} \mid \boldsymbol{x}) p(\boldsymbol{w} \mid \boldsymbol{x}, \boldsymbol{s}) \\
&\leq -\sum p(\boldsymbol{s} \mid \boldsymbol{x}) \log p(\boldsymbol{w} \mid \boldsymbol{x}, \boldsymbol{s}) \quad \text{(Jensen's inequality)}
\end{aligned}$$

Learning Hard Attention

$$
\begin{aligned}
\mathcal{L} &= -\log p(\boldsymbol{w} \mid \boldsymbol{x}) \\
&= -\log \sum_{\boldsymbol{s}} p(\boldsymbol{w}, \boldsymbol{s} \mid \boldsymbol{x}) \\
&= -\log \sum_{\boldsymbol{s}} p(\boldsymbol{s} \mid \boldsymbol{x}) p(\boldsymbol{w} \mid \boldsymbol{x}, \boldsymbol{s}) \\
&\leq -\sum_{\boldsymbol{s}} p(\boldsymbol{s} \mid \boldsymbol{x}) \log p(\boldsymbol{w} \mid \boldsymbol{x}, \boldsymbol{s}) \quad \text{(Jensen's inequality)} \\
&\overset{\text{MC}}{\approx} -\frac{1}{N} \sum_{i=1}^{N} p(\boldsymbol{s}^{(i)} \mid \boldsymbol{x}) \log p(\boldsymbol{w} \mid \boldsymbol{x}, \boldsymbol{s})
\end{aligned}
$$

Learning Hard Attention

- Sample *N* sequences of attention decisions from the model

- The gradient is the probability of the gradient of the probability of this sequence scaled by the log probability of generating the target words using that sequence of attention decisions

- This is equivalent to using the REINFORCE algorithm (Williams, 1992) using the log probability of the observed words as a “reward function”. REINFORCE a policy gradient algorithm used for reinforcement learning.

A(0.98)

woman(0.54)

is(0.37)

throwing(0.33)

a(0.28)

frisbee(0.37)

in(0.21)

a(0.18)

park(0.35)

.(0.33)

A woman holding a clock in her hand.

A large white bird standing in a forest.

Attention in Captioning

Add soft attention to image captioning: **+2 BLEU**

Add hard attention to image captioning: **+4 BLEU**

Summary

- Significant performance improvements
  - Better performance over vector-based encodings
  - Better performance with smaller training data sets
- Model interpretability
- Better gradient flow
- Better capacity (especially obvious for translation)

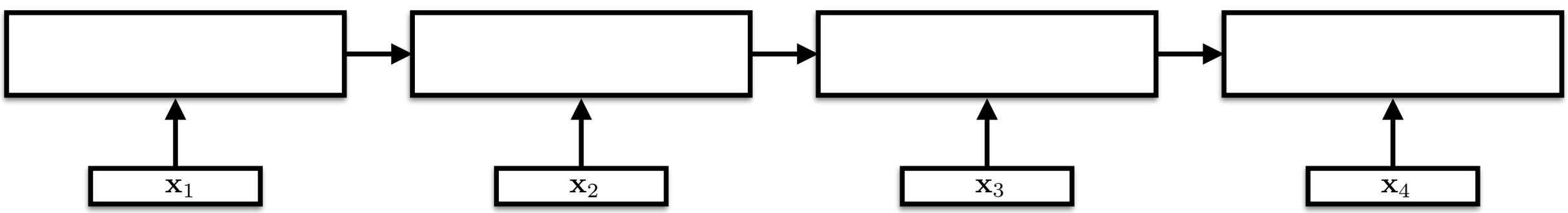
Questions?

A Few Tricks of the Trade

- Depth
- Dropout/recurrent dropout
- Minibatching

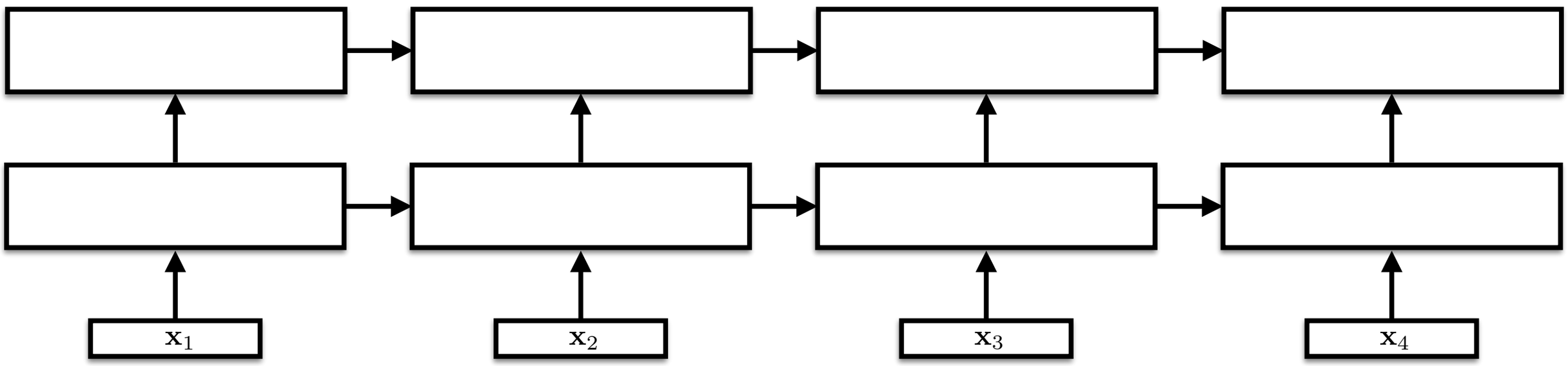
“Deep” LSTMs

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



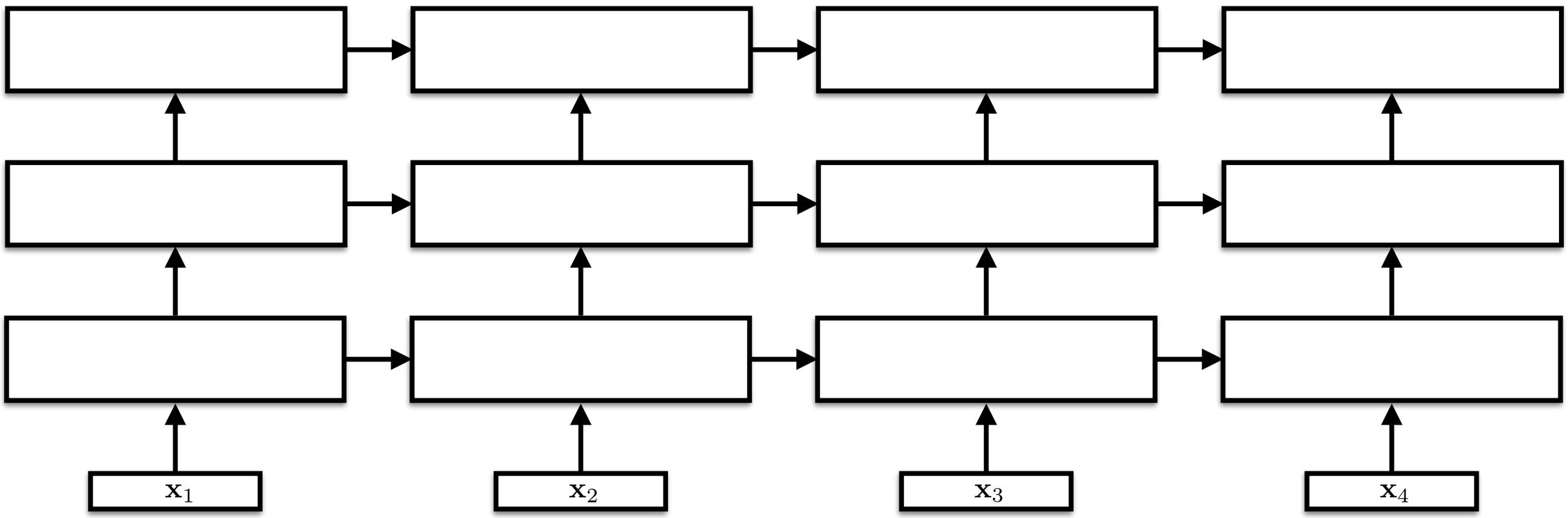
“Deep” LSTMs

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



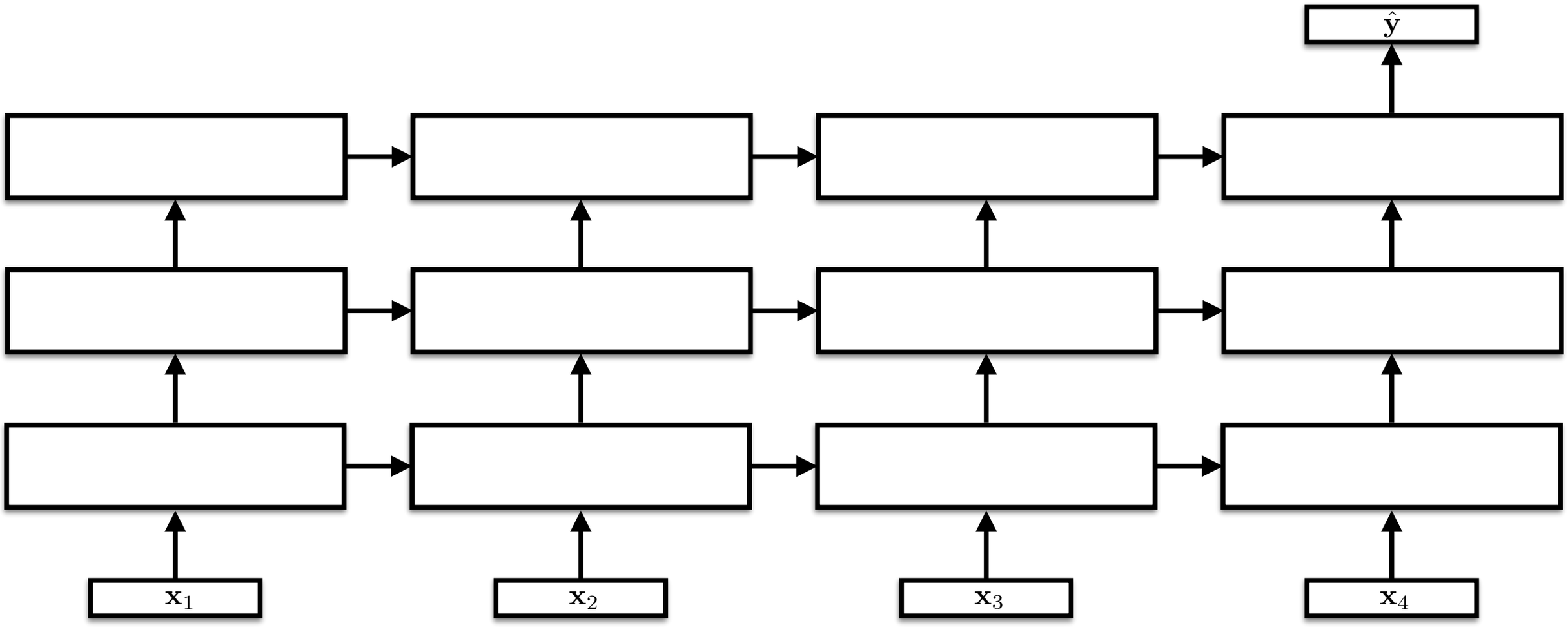
“Deep” LSTMs

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



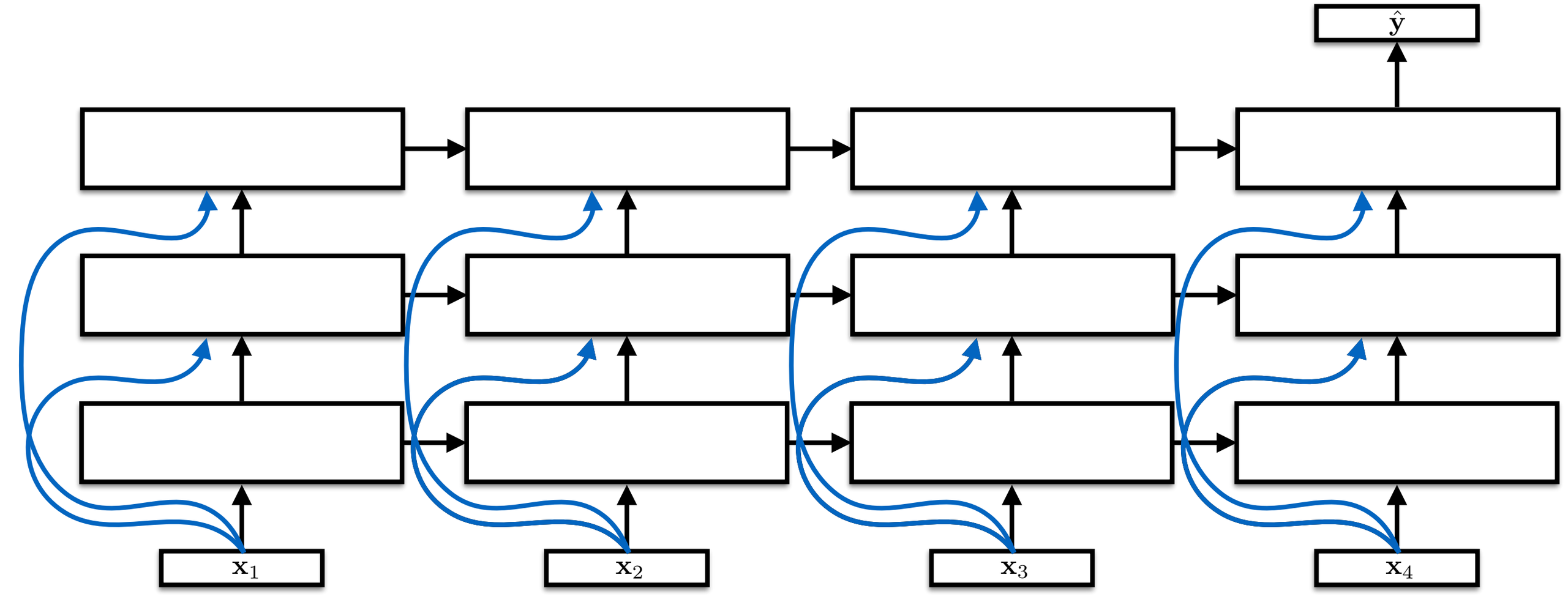
“Deep” LSTMs

- 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

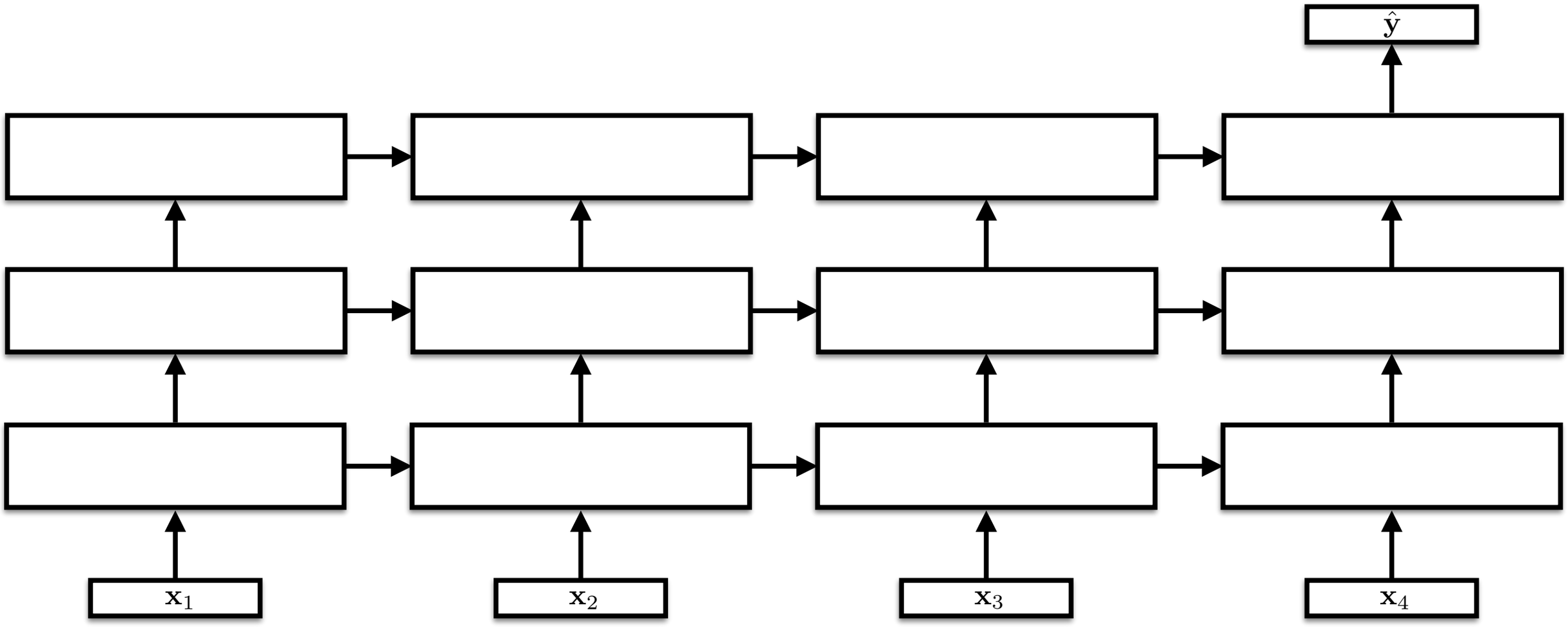


Does Depth Matter?

- 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

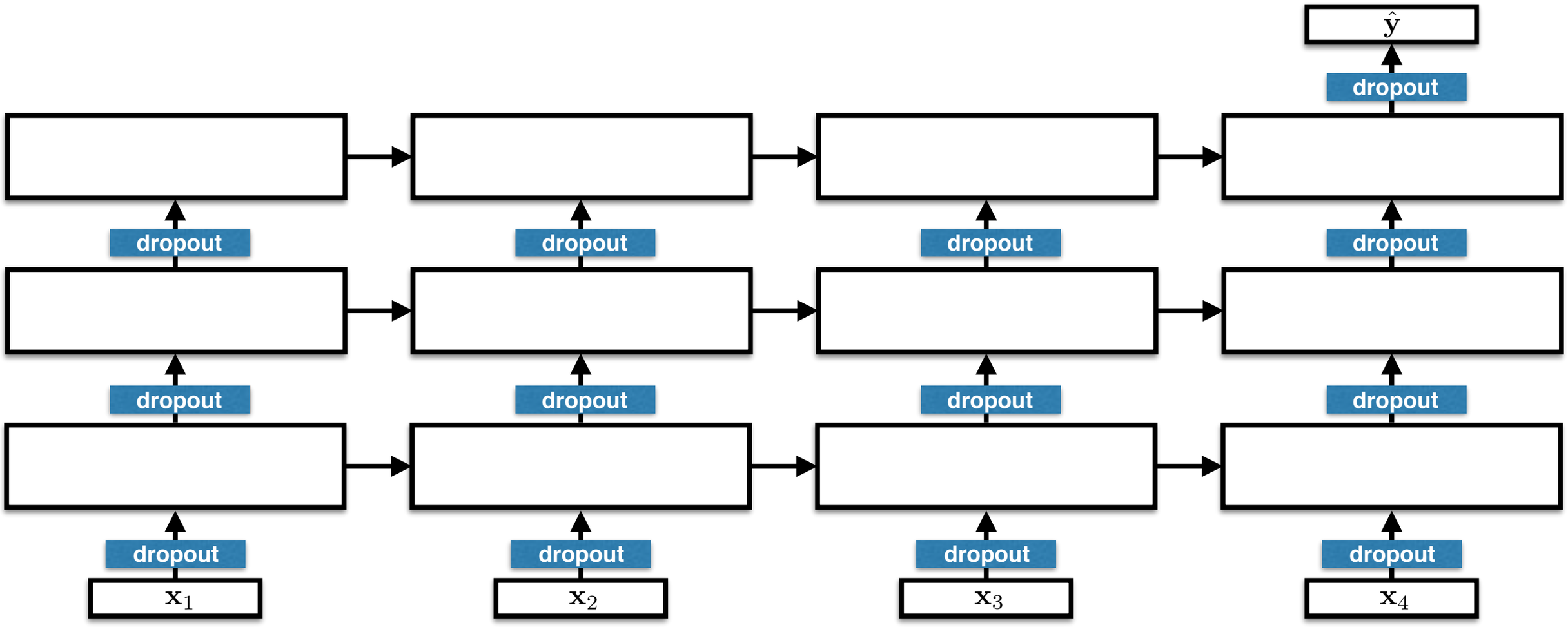
Dropout and Deep LSTMs

- Applying dropout layers requires some care



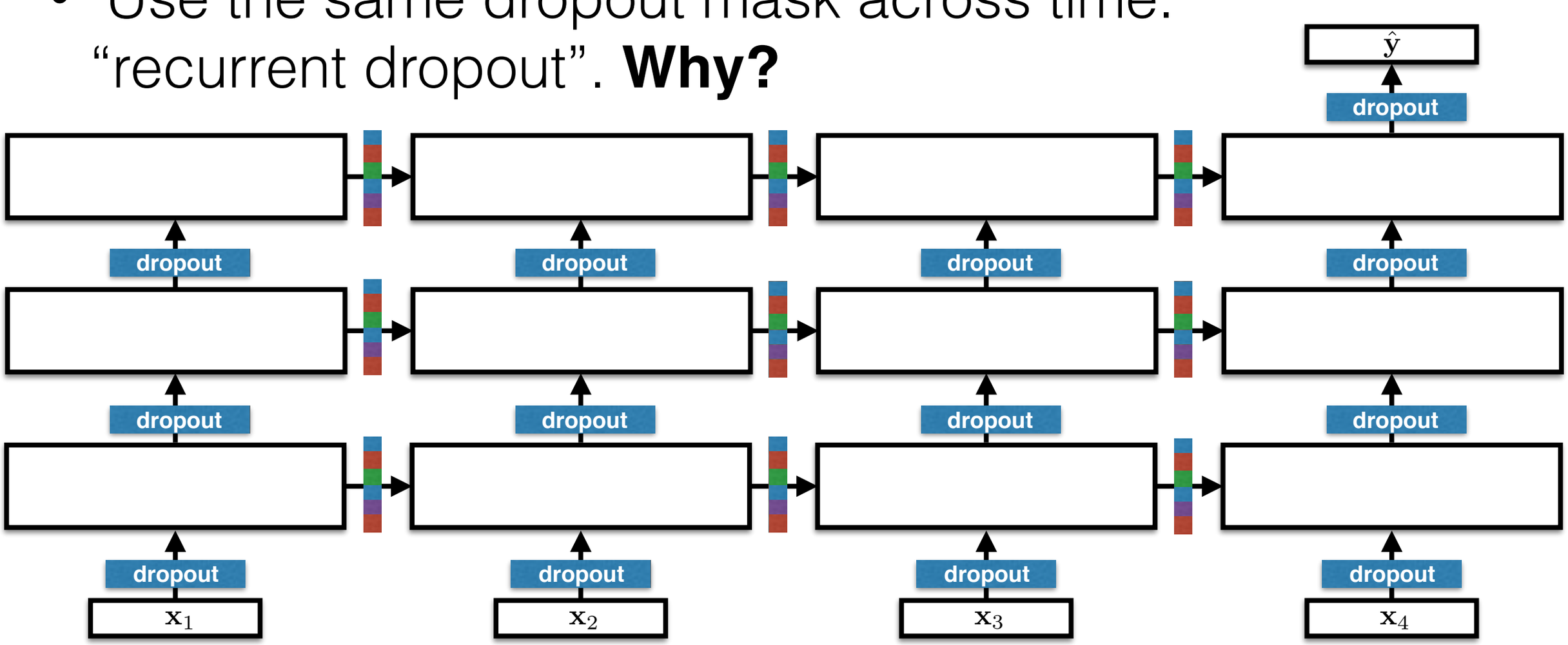
Dropout and Deep LSTMs

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



Dropout and Deep LSTMs

- Apply dropout between layers, but not on the recurrent connections
- Use the same dropout mask across time. “recurrent dropout”. **Why?**



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

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

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

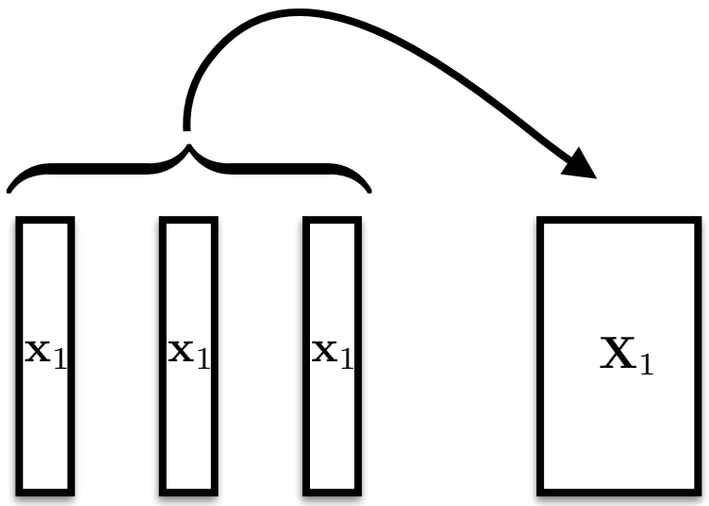
Minibatching

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

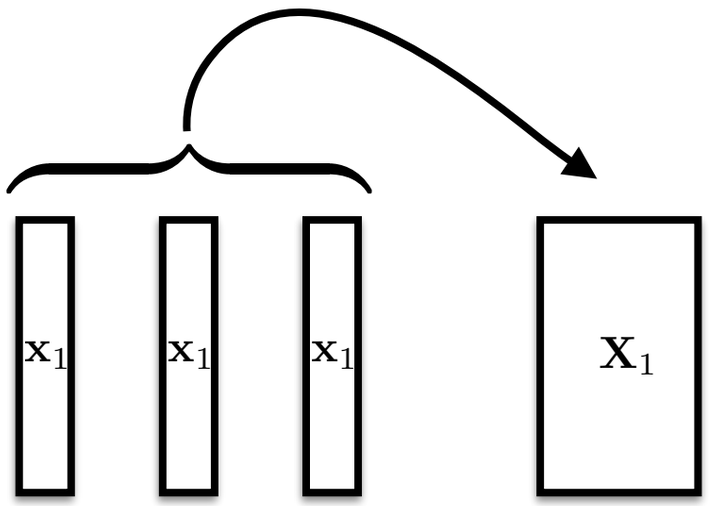
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

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

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

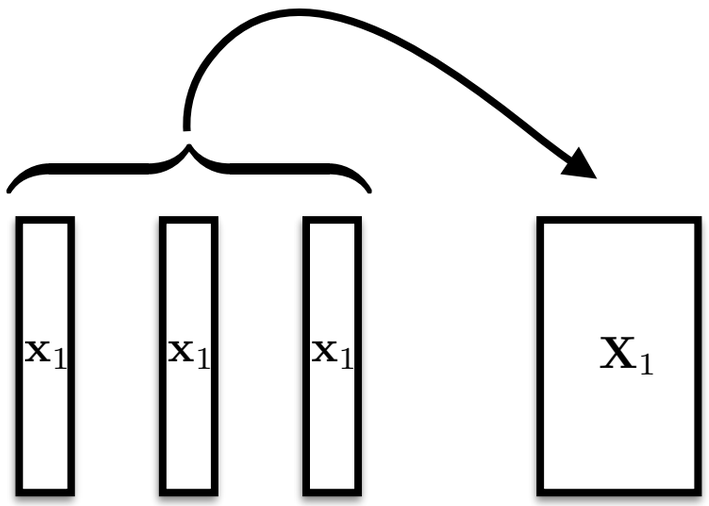
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

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

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.

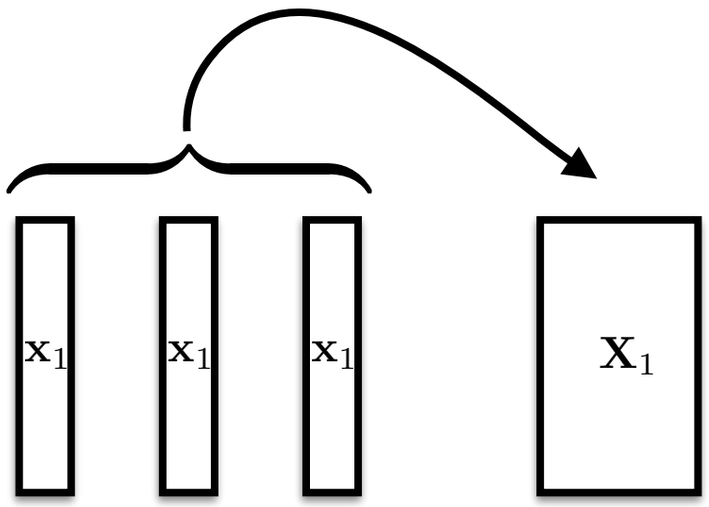
Minibatching

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.

Questions?

Há perguntas?

Obrigado!