Modelling Structured Data with Neural Nets

Chris Dyer
DeepMind
Carnegie Mellon University

LxMLS 2021

July 13, 2021

What can neural sequence models do?

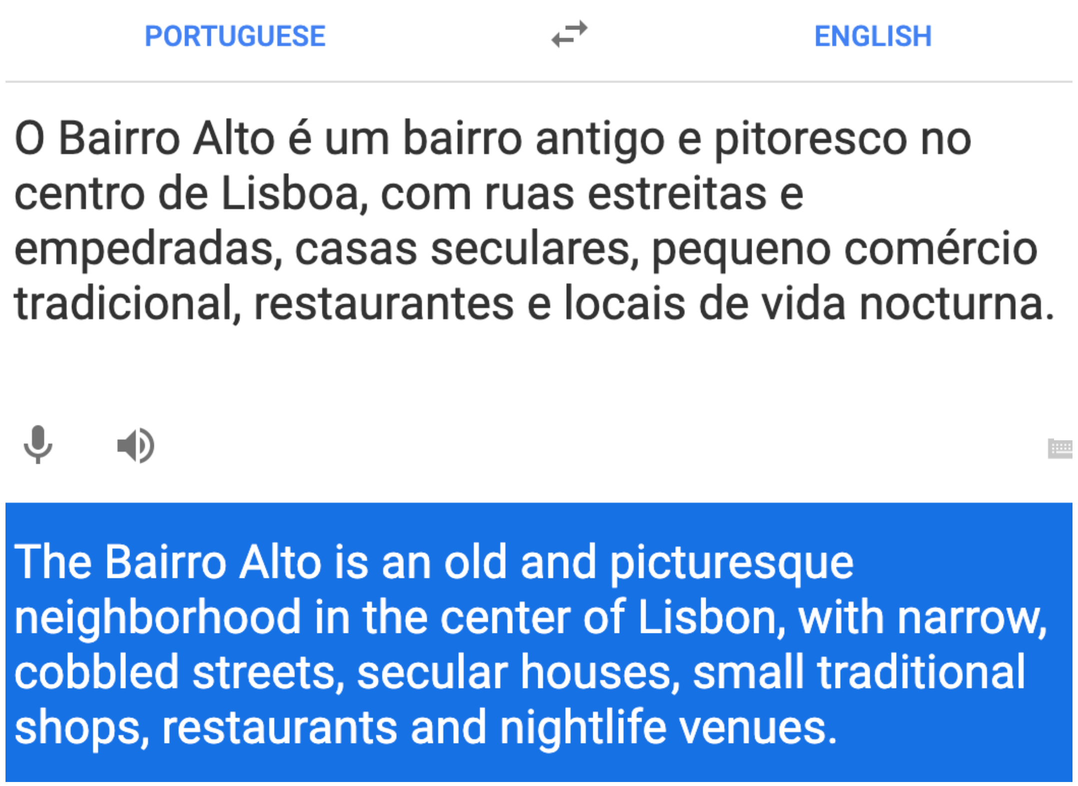
What can neural sequence models do?

PORTUGUESE ⇄ ENGLISH

O Bairro Alto é um bairro antigo e pitoresco no centro de Lisboa, com ruas estreitas e empedradas, casas seculares, pequeno comércio tradicional, restaurantes e locais de vida nocturna.

Google Translate

What can neural sequence models do?



Google Translate

What can neural sequence models do?

PORTUGUESE ⇄ ENGLISH

O Bairro Alto é um bairro antigo e pitoresco no centro de Lisboa, com ruas estreitas e empedradas, casas seculares, pequeno comércio tradicional, restaurantes e locais de vida nocturna.

The Bairro Alto is an old and picturesque neighborhood in the center of Lisbon, with narrow, cobbled streets, secular houses, small traditional shops, restaurants and nightlife venues.

Google Translate

In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

OpenAI GPT-2 Language Model

What can neural sequence models do?

PORTUGUESE ⇄ ENGLISH

O Bairro Alto é um bairro antigo e pitoresco no centro de Lisboa, com ruas estreitas e empedradas, casas seculares, pequeno comércio tradicional, restaurantes e locais de vida nocturna.

The Bairro Alto is an old and picturesque neighborhood in the center of Lisbon, with narrow, cobbled streets, secular houses, small traditional shops, restaurants and nightlife venues.

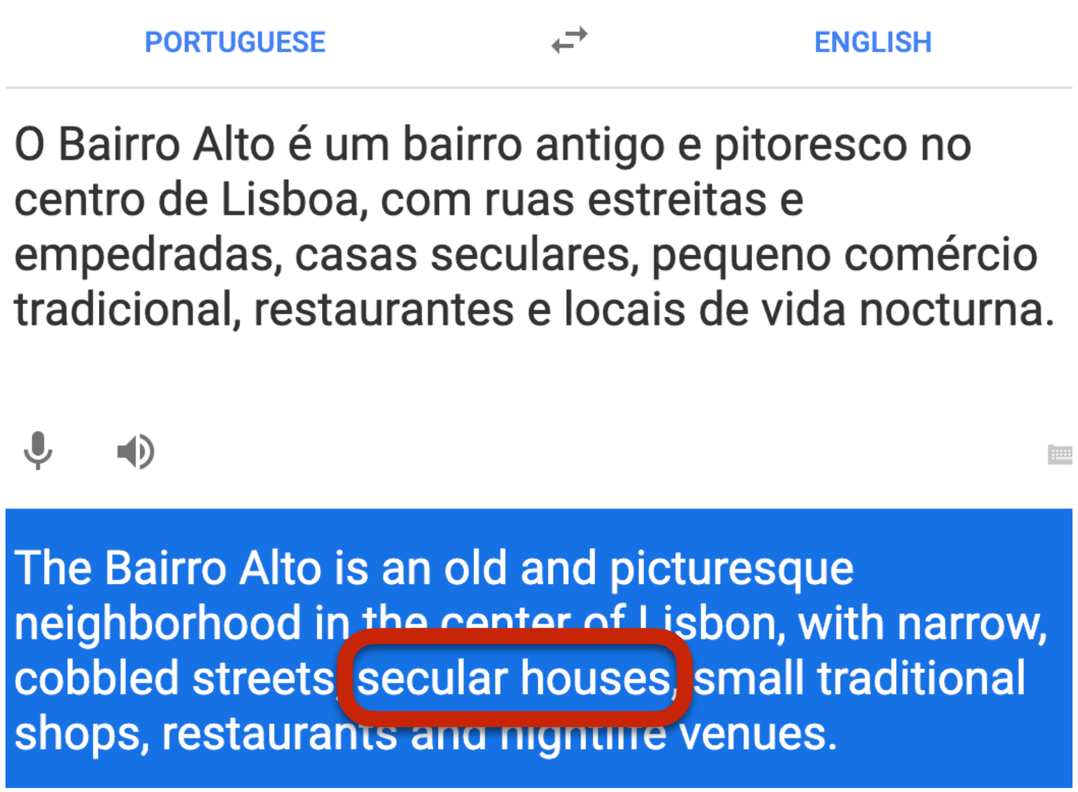
Google Translate

In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

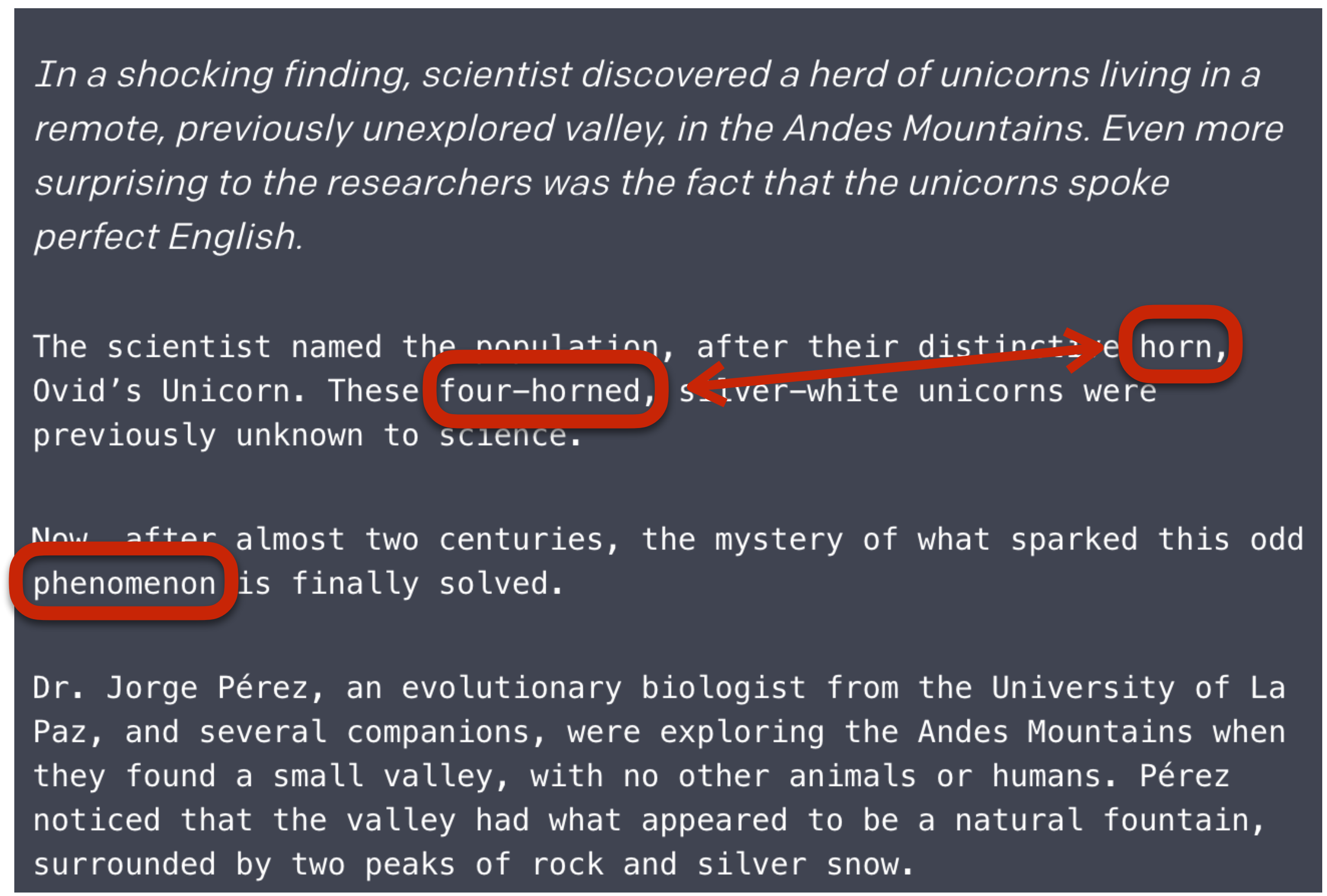
```
The scientist named the population, after their distinctive horn,
Ovid's Unicorn. These four-horned, silver-white unicorns were
previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd
phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La
Paz, and several companions, were exploring the Andes Mountains when
they found a small valley, with no other animals or humans. Pérez
noticed that the valley had what appeared to be a natural fountain,
surrounded by two peaks of rock and silver snow.
```

OpenAI GPT-2 Language Model

What can neural sequence models do?



Google Translate



OpenAI GPT-2 Language Model

What can neural sequence models do?

sentiment.ts | write_sql.go | parse_expenses.py | addresses.rb

```
#!/usr/bin/env ts-node

import { fetch } from "fetch-h2";

// Determine whether the sentiment of text is positive
// Use a web service
async function isPositive(text: string): Promise<boolean> {
```

What can neural sequence models do?

sentiment.ts | write_sql.go | parse_expenses.py | addresses.rb

```typescript
#!/usr/bin/env ts-node

import { fetch } from "fetch-h2";

// Determine whether the sentiment of text is positive
// Use a web service
async function isPositive(text: string): Promise<boolean> {
  const response = await fetch(`http://text-processing.com/api/sentiment/`, {
    method: "POST",
    body: `text=${text}`,
    headers: {
      "Content-Type": "application/x-www-form-urlencoded",
    },
  });
  const json = await response.json();
  return json.label === "pos";
}
```

Copilot

Outline: Part I

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

Outline: Part II

- Conditional language models
- Encode-decoder architectures
- Sequence-to-sequence with attention and RNNs
- Transformers and self-attention

Representing Sequential Data
Recurrent Neural Networks

- Lots of (most??) interesting data is sequential in nature
  - Words in sentences, documents, conversations
  - DNA, amino acids
  - Stock market returns
  - Tokens in a program, actions taken by an agent playing a game, sound amplitudes in an acoustic signal, the pixels in an image, … … … …

Sequence Modeling Tasks

- In some applications, we want to **condition** on sequential data and make a prediction
  - Examples: read a review and decide whether it is positive or negative; read a blog post and predict who wrote it
- In other applications, we want to **generate** sequential data
  - Examples: POS tagging, machine translation, summarization, “natural language generation”, image generation, text to speech, speech to text …
  - (in many of these, we need to do both)

Example: Language Models

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

It is convenient (but not necessary) to decompose this probability using the **chain rule**, as follows:

$$p(\boldsymbol{w}) = p(w_1) \times p(w_2 \mid w_1) \times p(w_3 \mid w_1, w_2) \times \cdots \times p(w_\ell \mid w_1, \ldots, w_{\ell-1})$$

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

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

Example: Language Models

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

Thus,

(i) For **conditioning problems**, we need to **represent a sequence**.

(ii) For **generation problems**, we need to **represent a sequence** — the *history* at each time step.

Example: Language Models

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

Thus,

How do we represent an arbitrarily long sequence?

(i) For **co** … **t a seque**

(ii) For **ge** … **a sequence** … the *history* at each time step.

Example: Language Models

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

Thus,

(i) For **co** … **a seque** …

(ii) For **ge** … **a sequence** … the *history* at each time step.

How do we represent an arbitrarily long sequence?

We will train a neural network to build a representation of sequences of unbounded length.

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

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

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

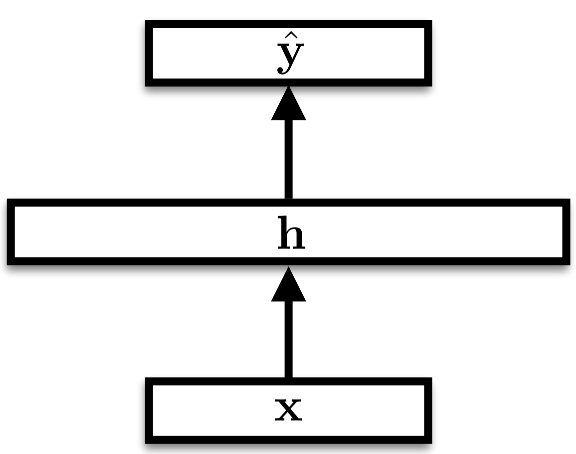
- What functions can this parametric form compute?
  - If **h** is big enough (i.e., enough dimensions), it can represent any vector-valued function to any degree of precision
- This is a much more powerful regression model!
- You can think of **h** as "induced features" in a linear classifier
  - The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

Recurrent Neural Networks

Feed-forward NN

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

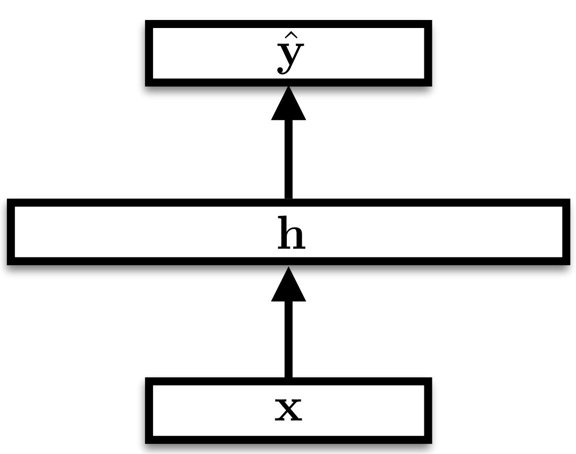
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$



Recurrent Neural Networks

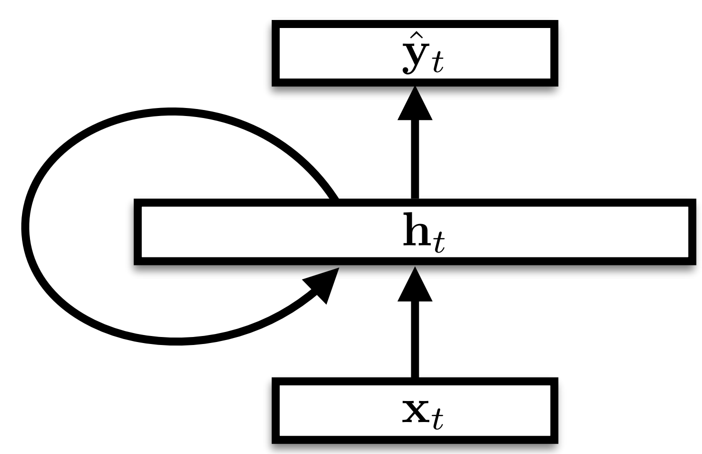
Feed-forward NN

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



Recurrent NN

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

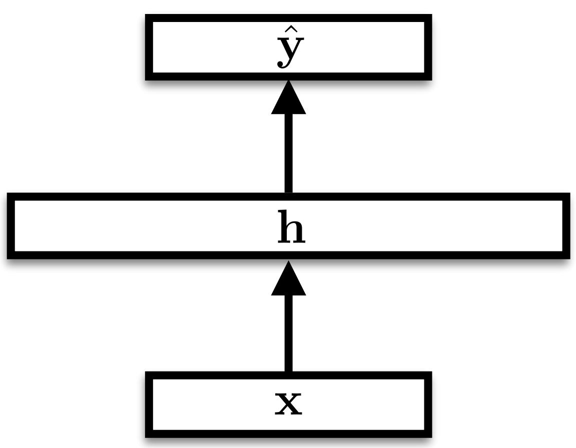


Recurrent Neural Networks

Feed-forward NN

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

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

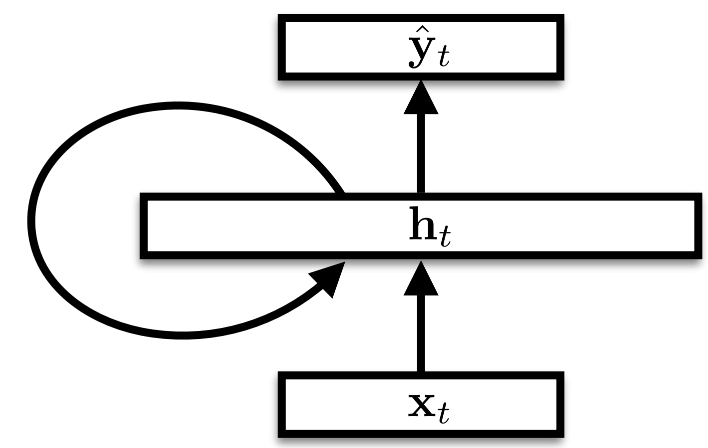


Recurrent NN

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

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

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

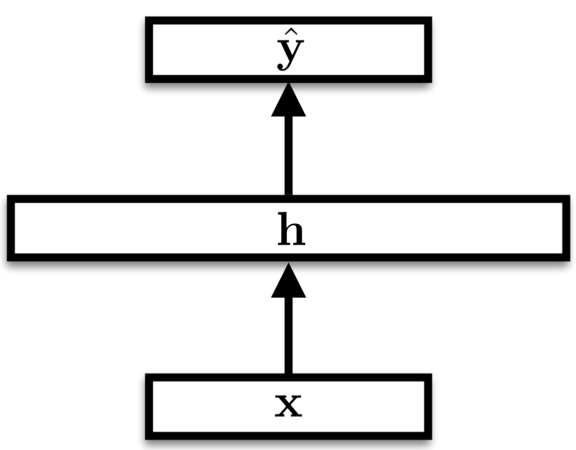


Recurrent Neural Networks

Feed-forward NN

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

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

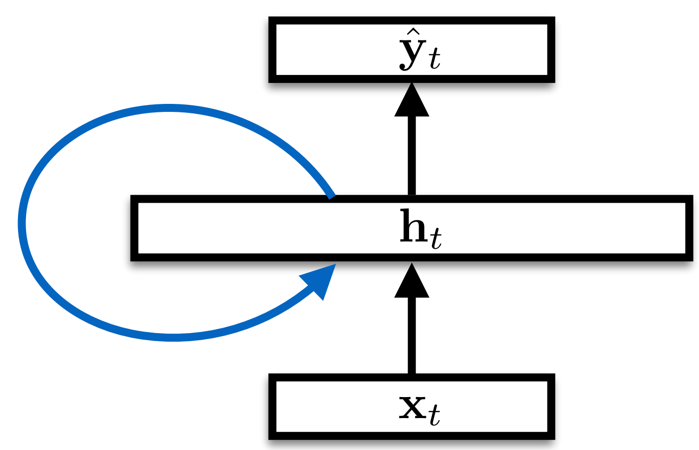


Recurrent NN

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

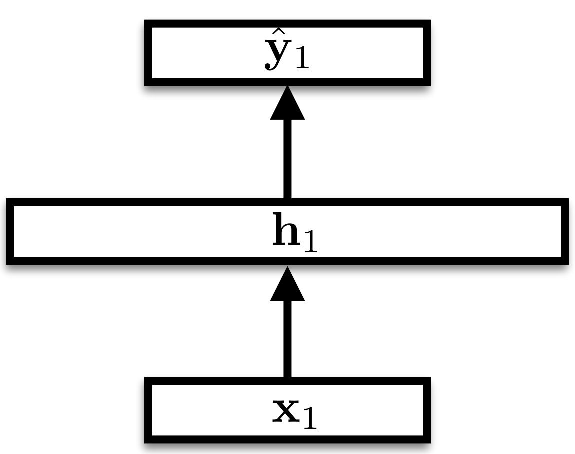
$$\mathbf{h}_t = g(\mathbf{V}[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{c})$$

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



Recurrent Neural Networks

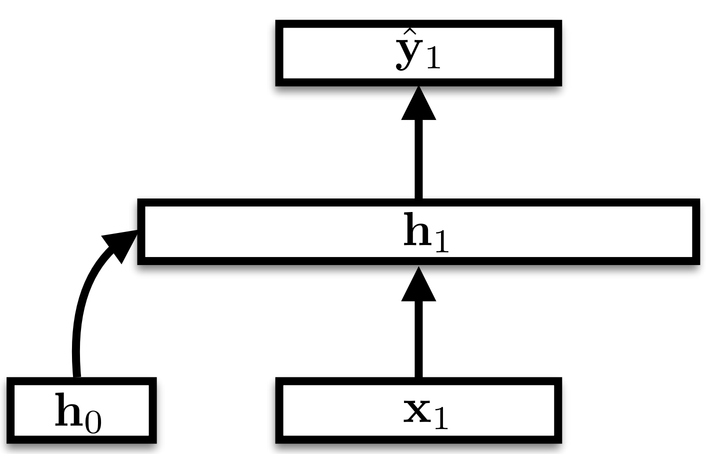
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

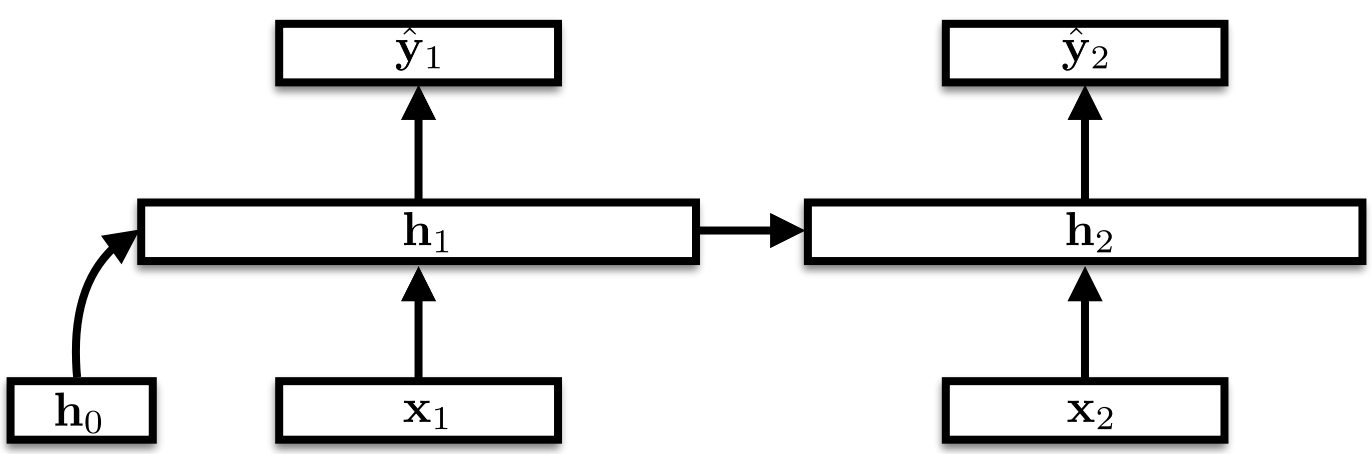
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

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



Recurrent Neural Networks

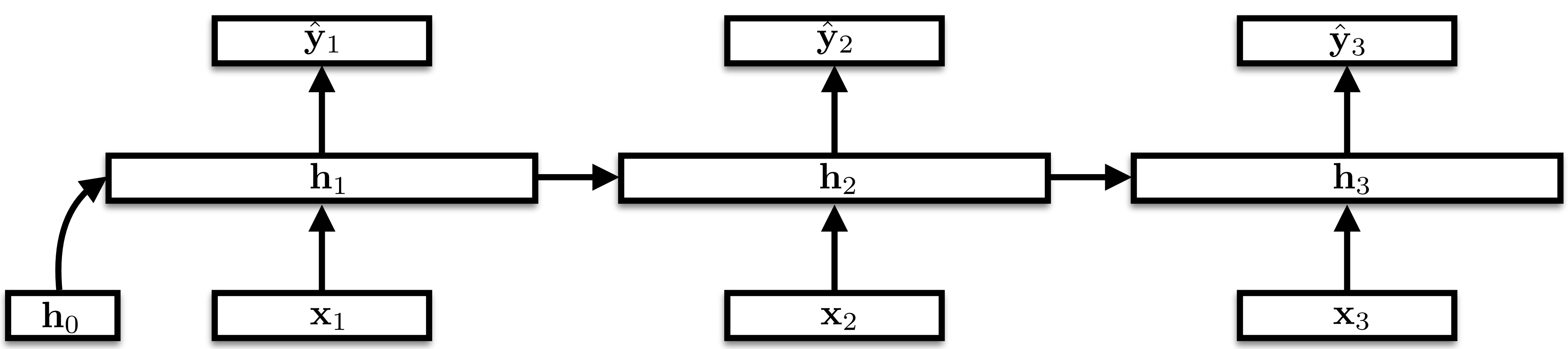
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

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

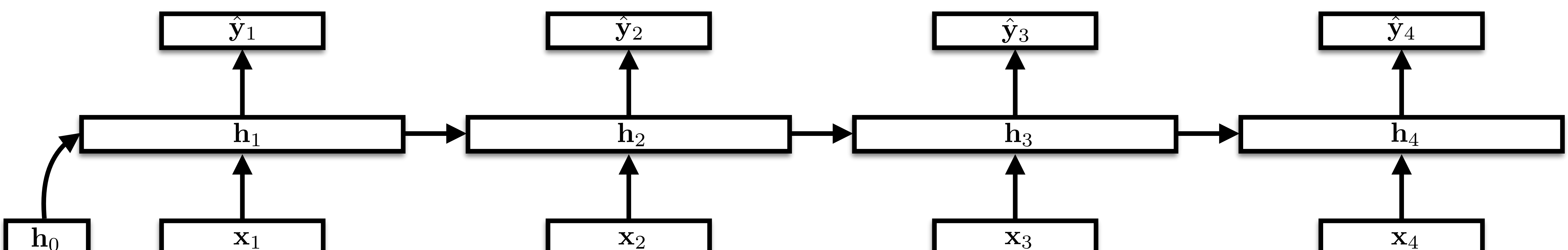
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

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

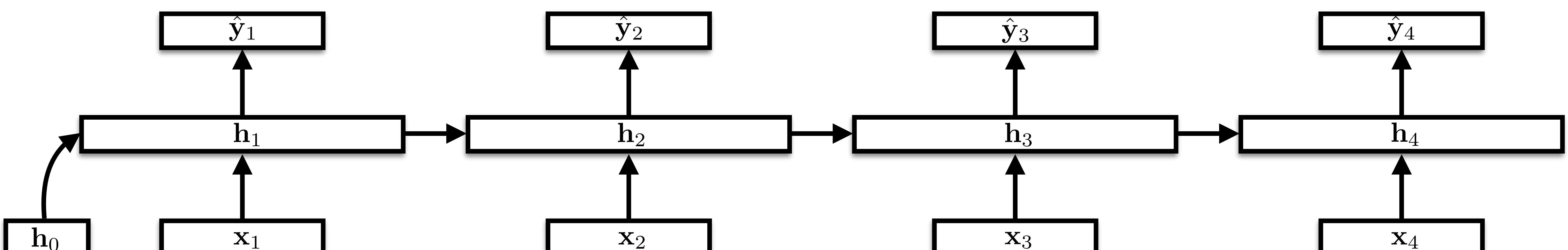
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

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

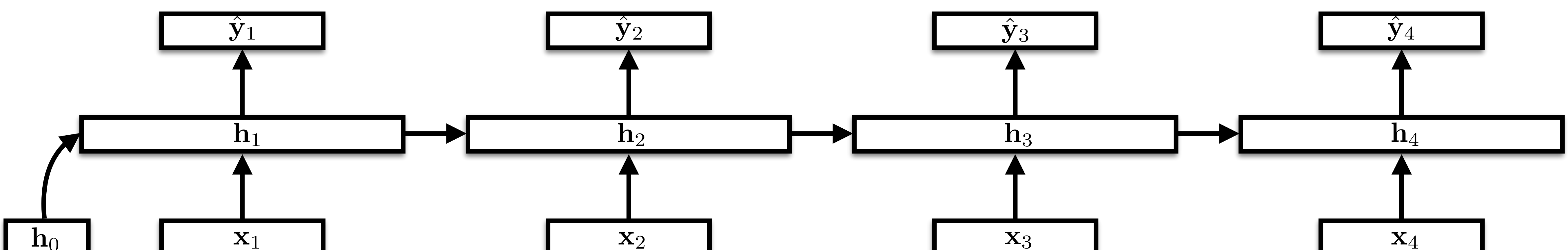
How do we train the RNN's parameters?



Recurrent Neural Networks

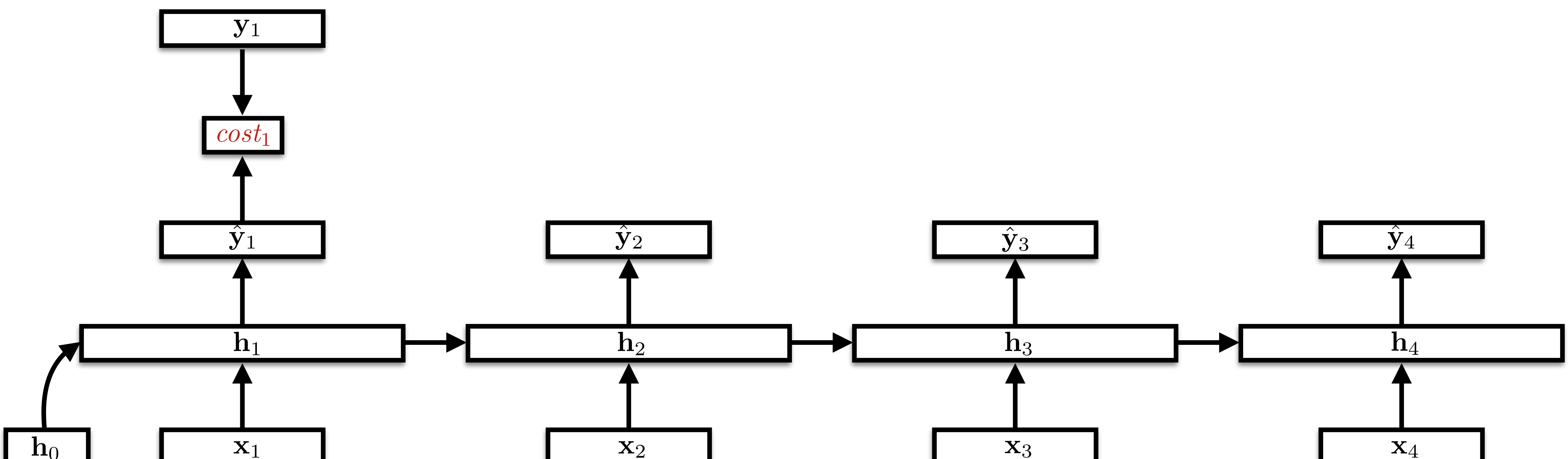
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

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



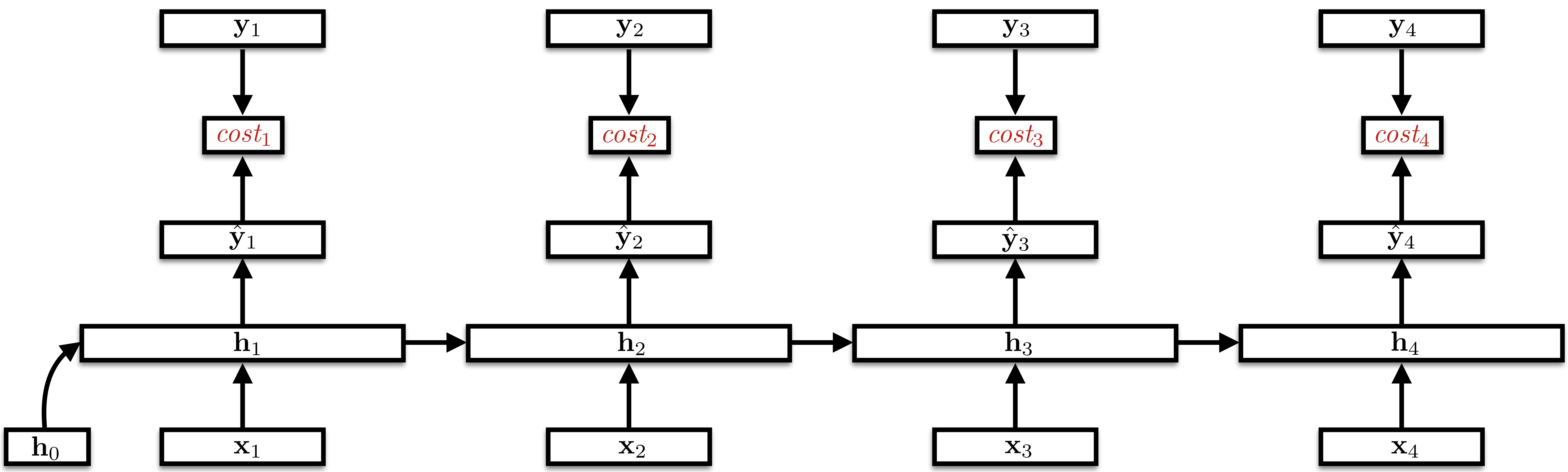
Recurrent Neural Networks

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



Recurrent Neural Networks

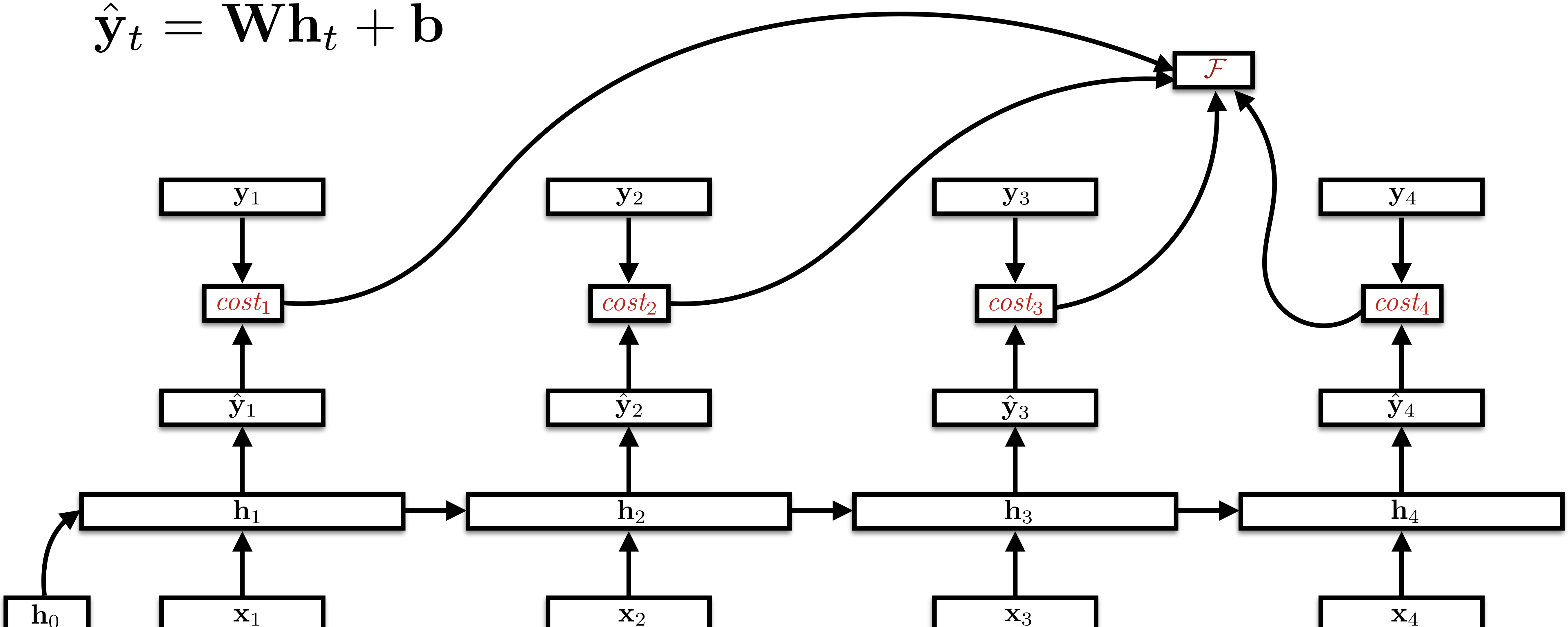
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



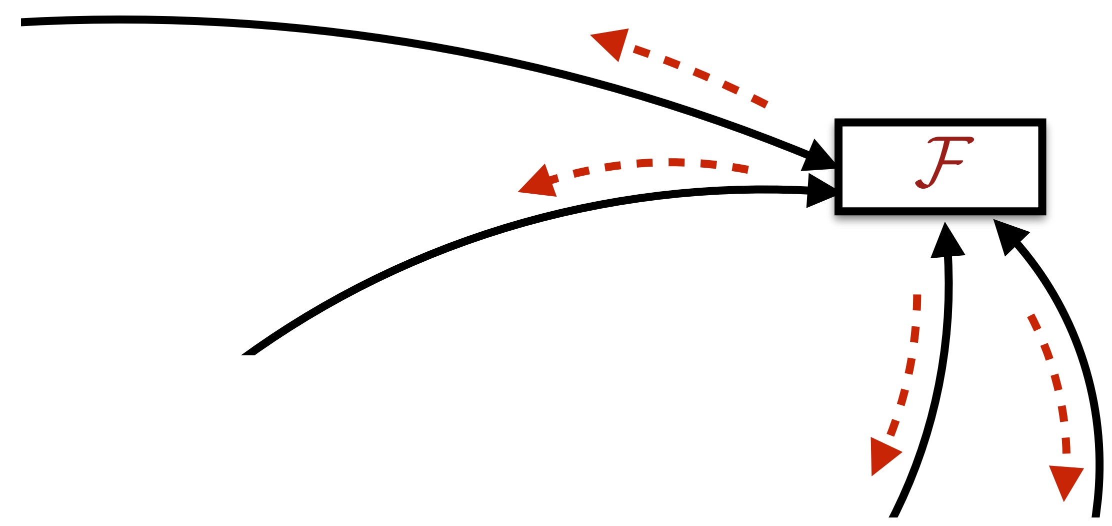
Recurrent Neural Networks

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

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



Recurrent Neural Networks

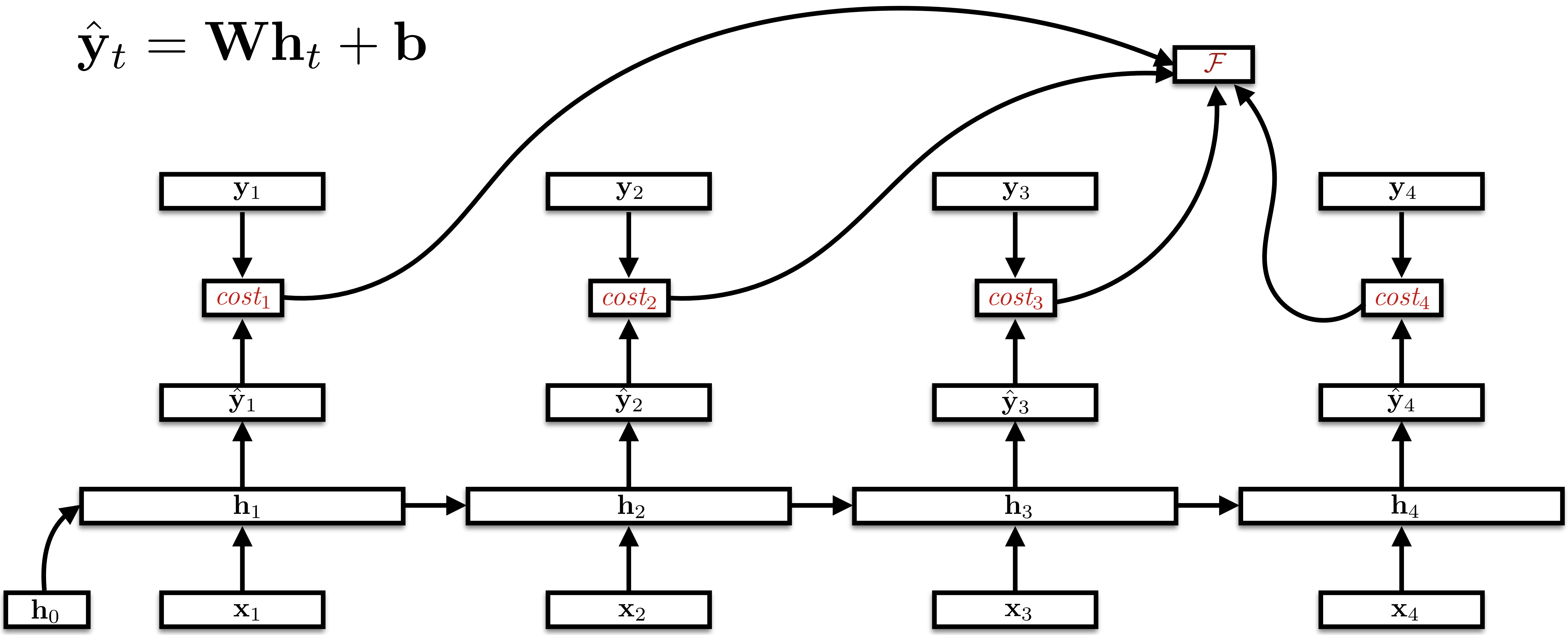


- The unrolled graph is a well-formed (DAG) computation graph—we can run backprop
  - Parameters are tied across time, derivatives are aggregated across all time steps
  - This is historically called “backpropagation through time” (BPTT)

Parameter Tying

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

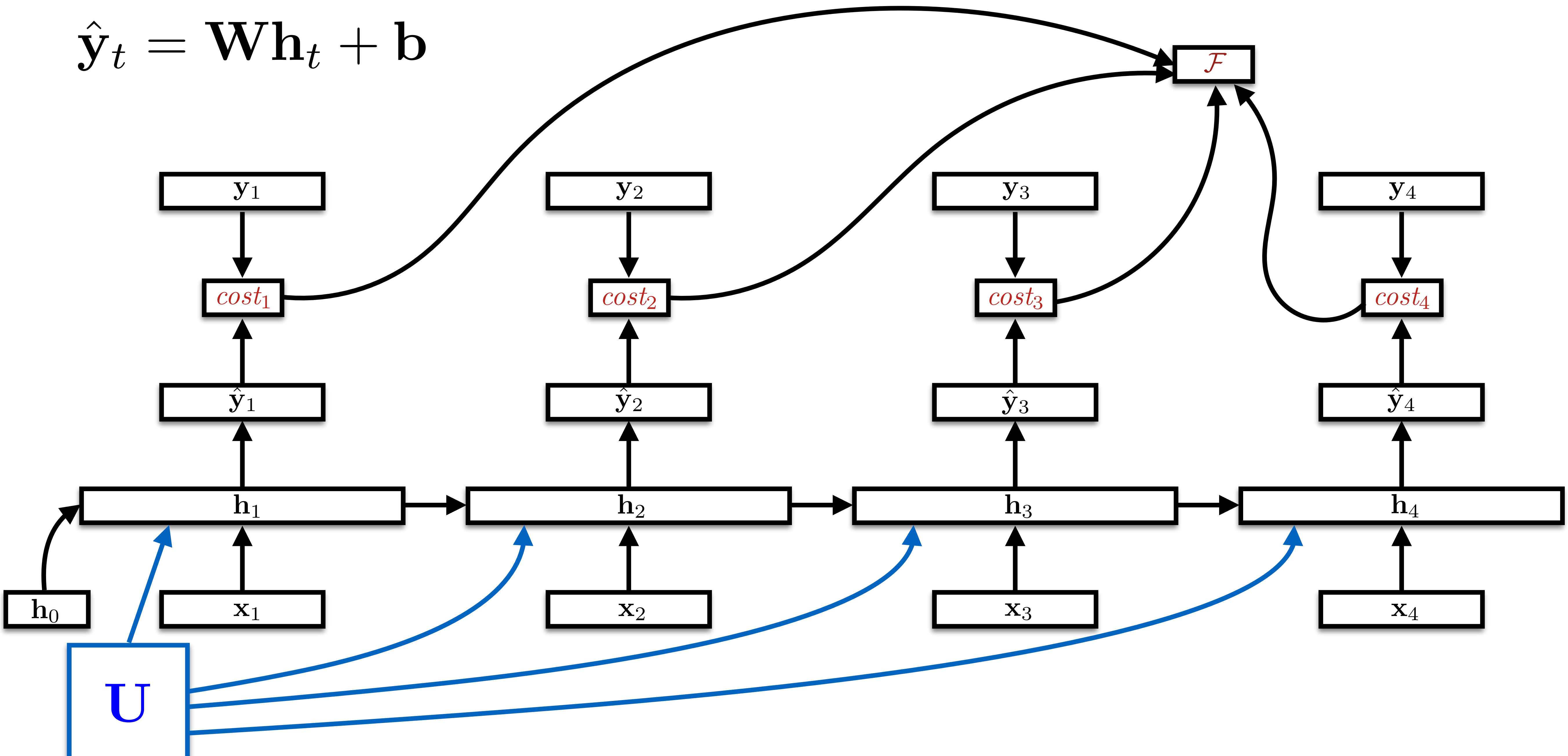
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



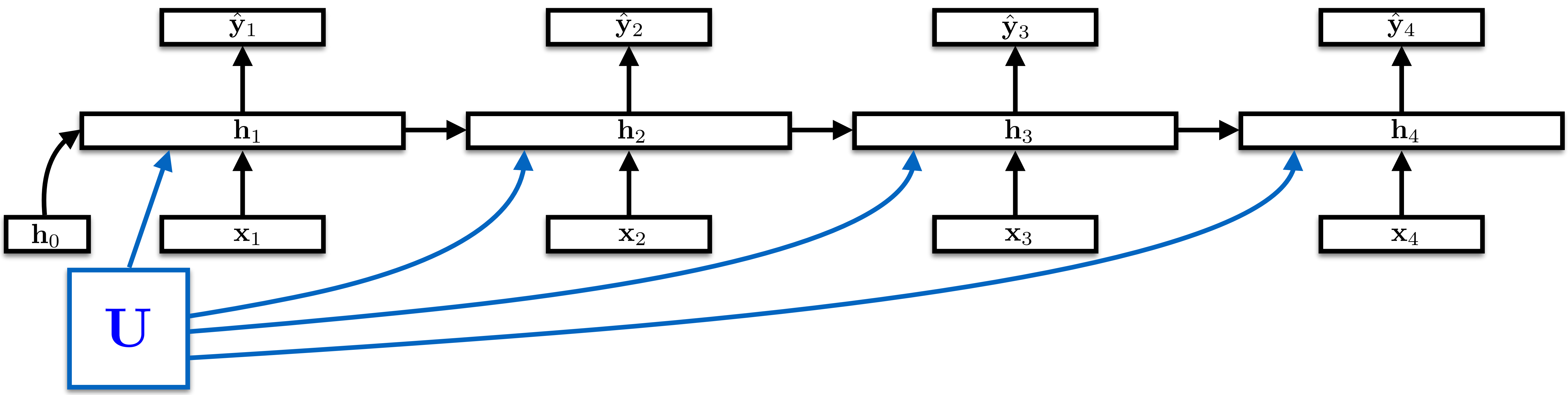
Parameter Tying

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

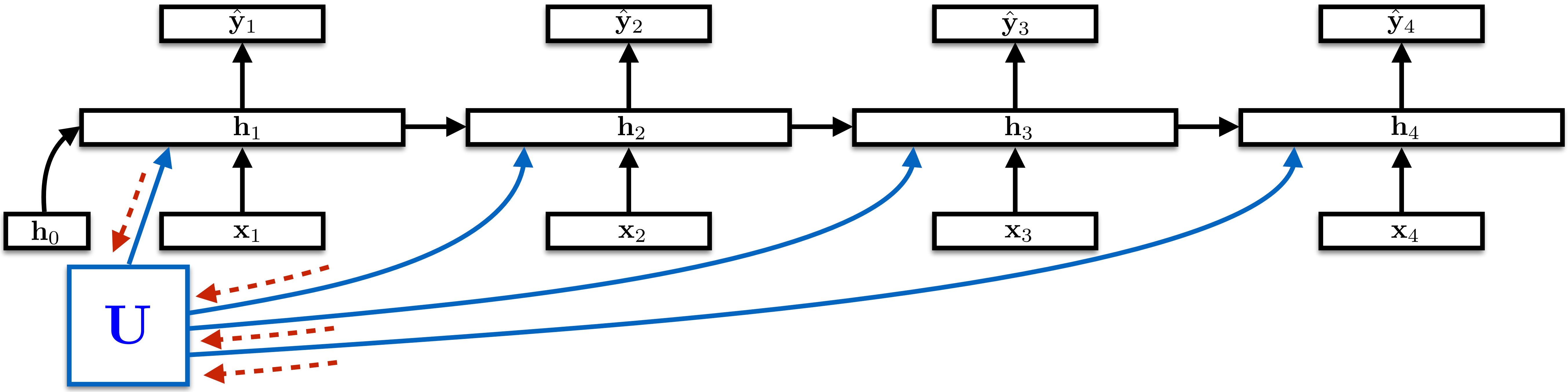
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Parameter Tying

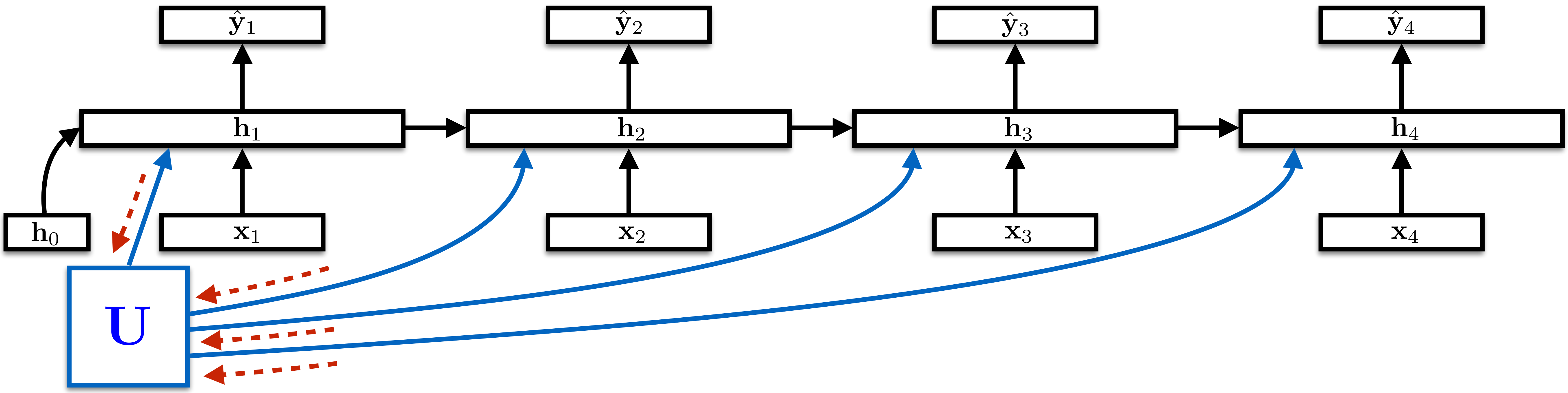


Parameter Tying



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

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 transition matrix for HMMs!

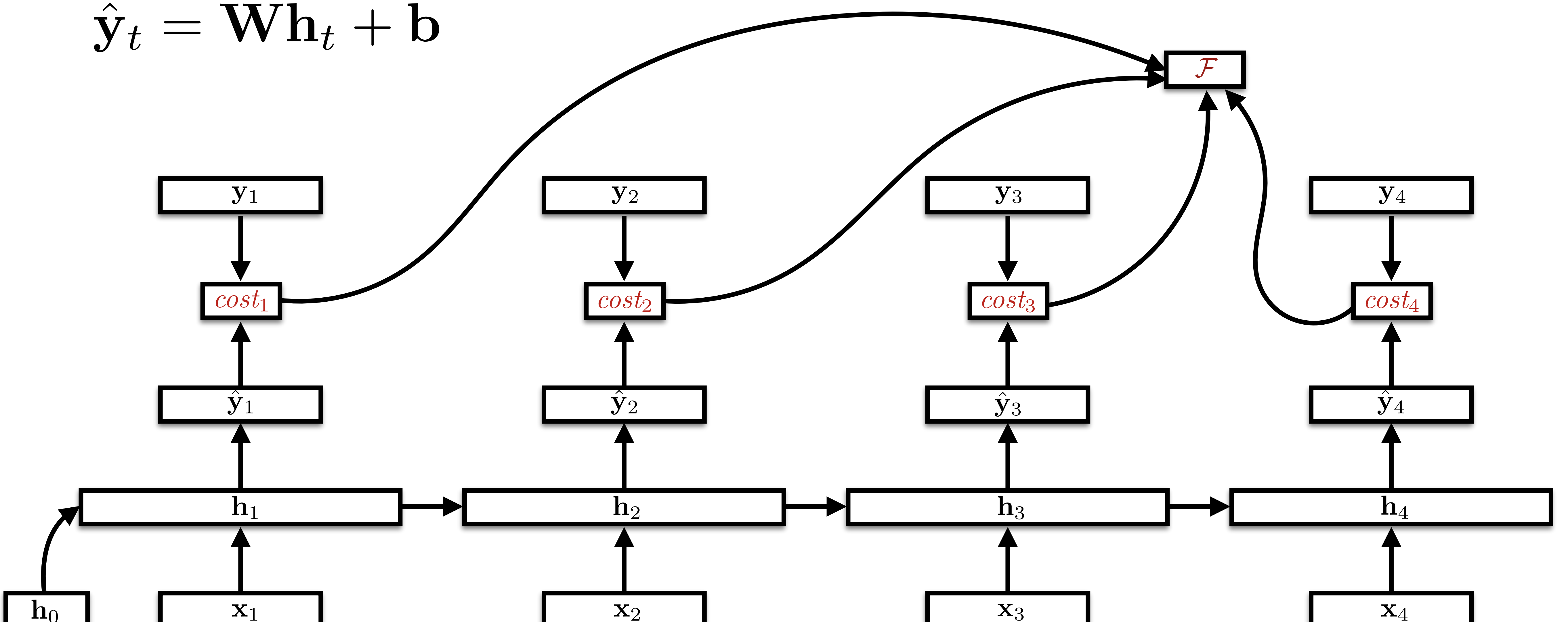
Parameter Tying

- 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?

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

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

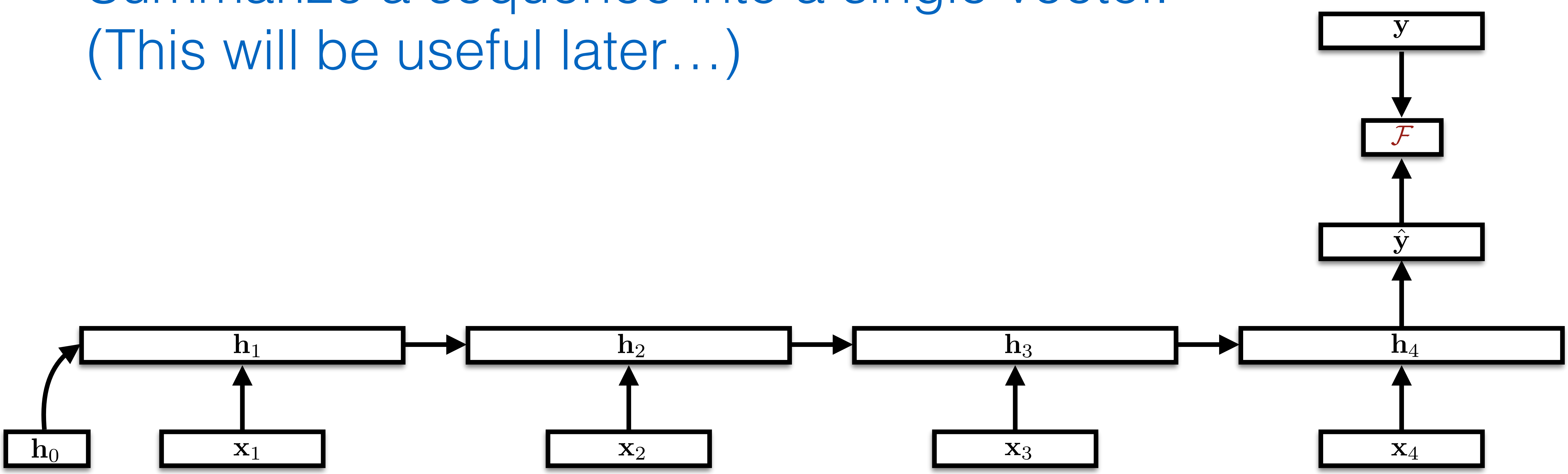


“Read and summarize”

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

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

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

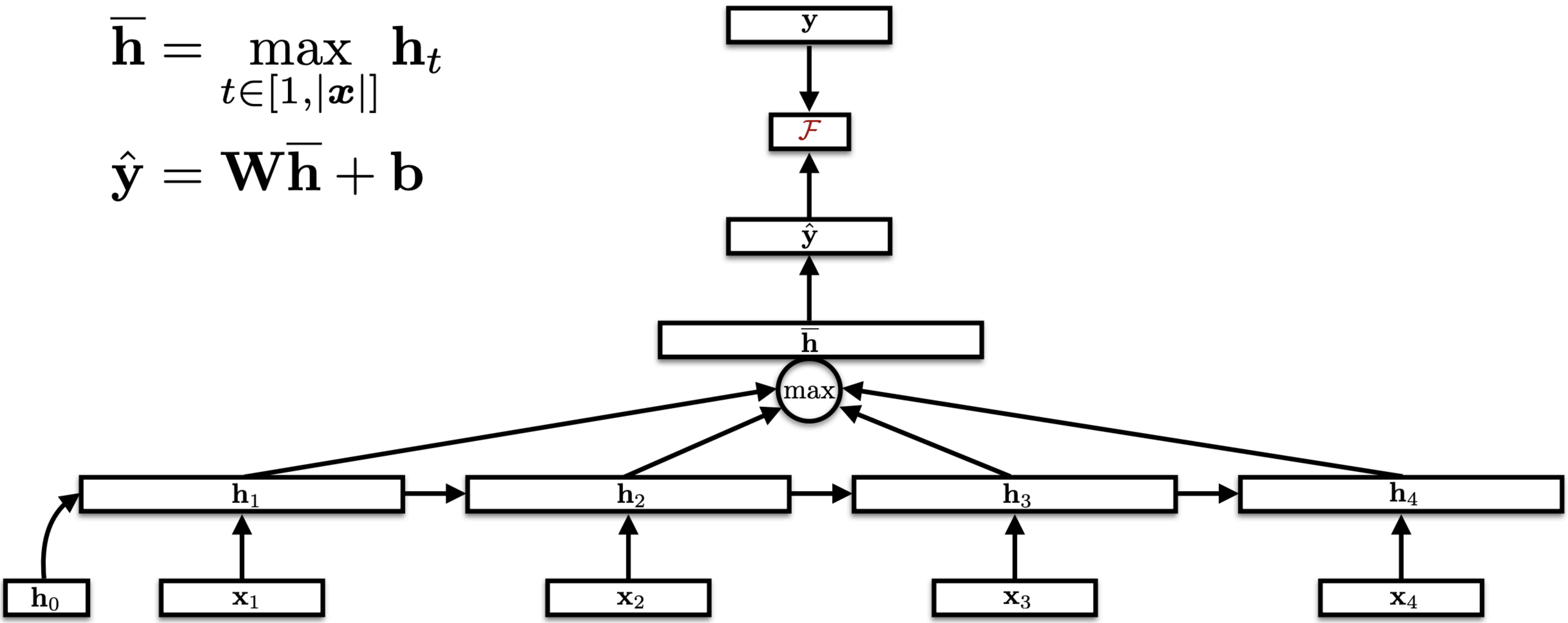


“Read and summarize”

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

$$\overline{\mathbf{h}} = \max_{t \in [1, |\boldsymbol{x}|]} \mathbf{h}_t$$

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



View 2: Recursive Definition

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

View 2: Recursive Definition

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

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

View 2: Recursive Definition

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

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

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

  - Define (learn) a representation of the base case

  - Learn a representation of the inductive step

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

History-based LMs

As Noah told us, a common strategy is in sequence modeling is to make a **Markov assumption.**

$$
\begin{aligned}
p(\boldsymbol{w}) = & p(w_1)\times \\
& p(w_2 \mid w_1)\times \\
& p(w_3 \mid w_1, w_2)\times \\
& p(w_4 \mid w_1, w_2, w_3)\times \\
& \ldots
\end{aligned}
$$

History-based LMs

As Noah told us, a common strategy is in sequence modeling is to make a **Markov assumption.**

$$
\begin{aligned}
p(\boldsymbol{w}) = & p(w_1)\times \\
& p(w_2 \mid w_1)\times \\
& p(w_3 \mid \cancel{w_1}, w_2)\times \\
& p(w_4 \mid \cancel{w_1, w_2}, w_3)\times \\
& \ldots
\end{aligned}
$$

Markov: forget the “distant” past.
Is this valid for language? No…
Is it practical? Often!

History-based LMs

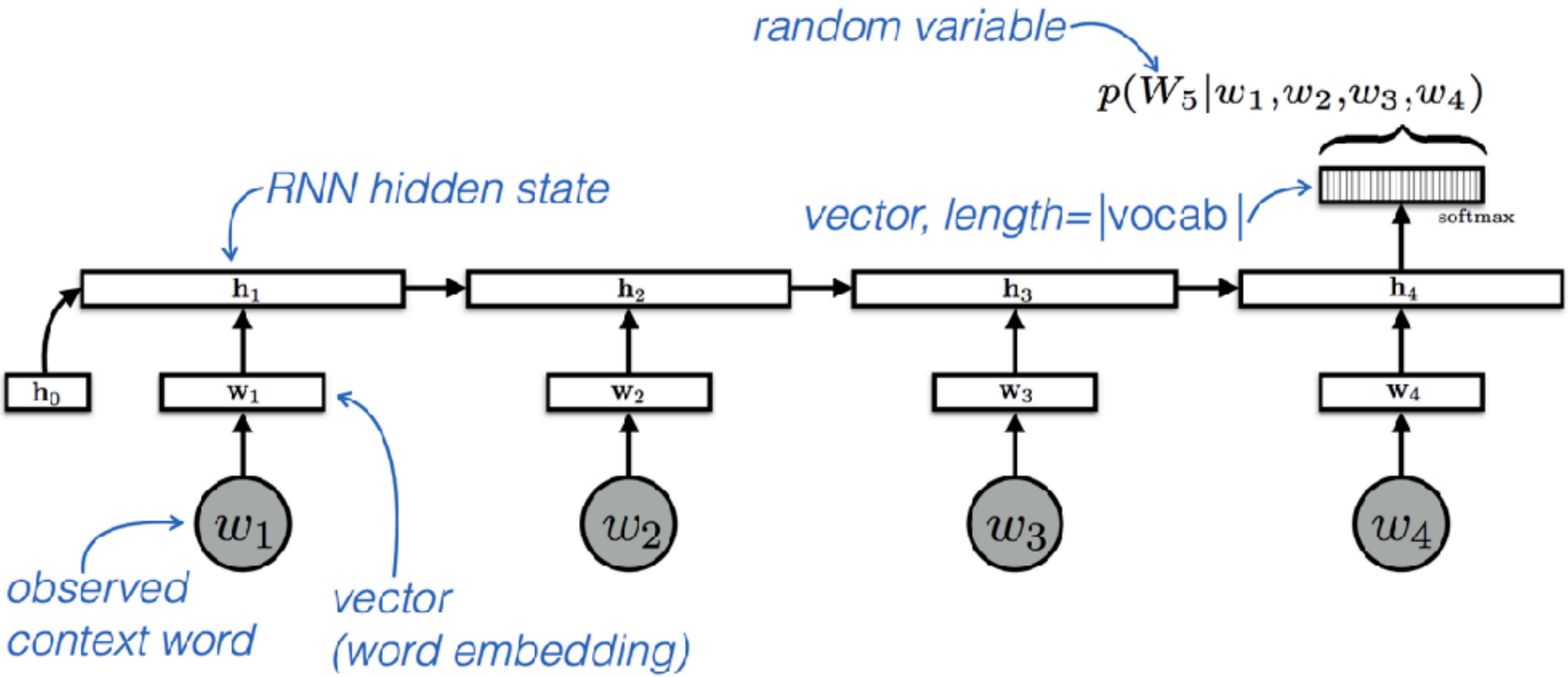
As Noah told us, a common strategy is in sequence modeling is to make a **Markov assumption.**

$$
\begin{aligned}
p(\boldsymbol{w}) = & p(w_1) \times \\
& p(w_2 \mid w_1) \times \\
& p(w_3 \mid \cancel{w_1}, w_2) \times \\
& p(w_4 \mid \cancel{w_1, w_2}, w_3) \times \\
& \ldots
\end{aligned}
$$

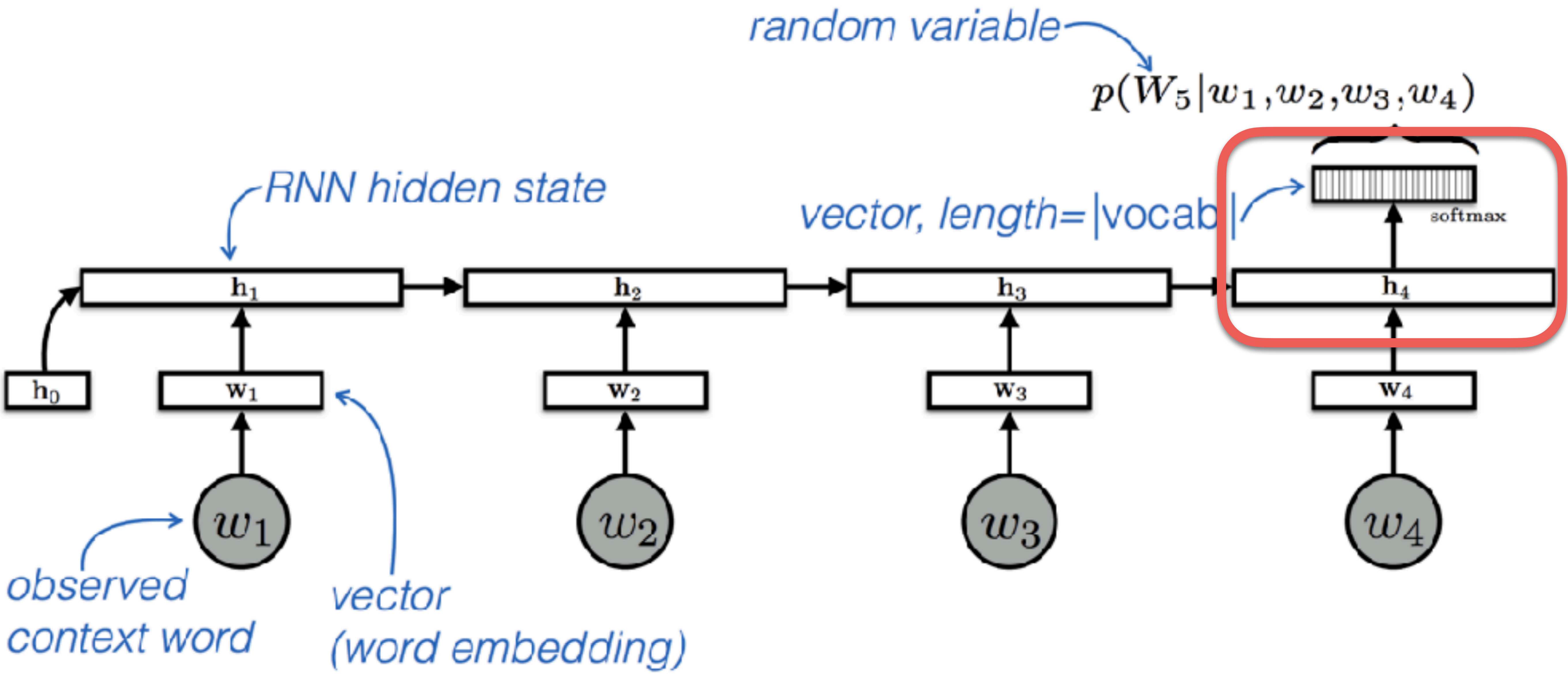
Markov: forget the “distant” past.
Is this valid for language? No…
Is it practical? Often!

Why RNNs are great for language:
no more Markov assumptions.

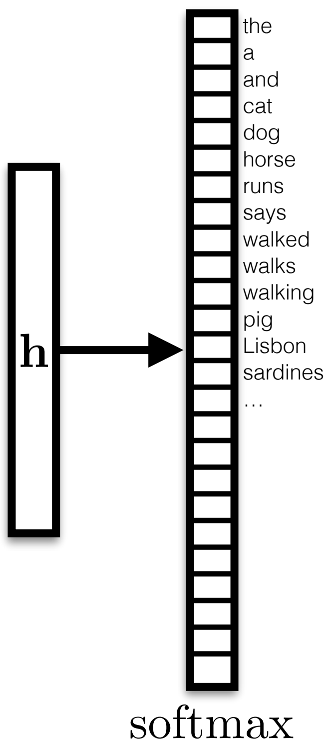
History-based LMs with RNNs



History-based LMs with RNNs



Example: Language Model



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

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

The p_i's form a distribution, i.e.

$$p_i > 0 \quad \forall i, \quad \sum p_i = 1$$

To enforce this stochastic constraint, we suggest a *normalised exponential* output non-linearity,

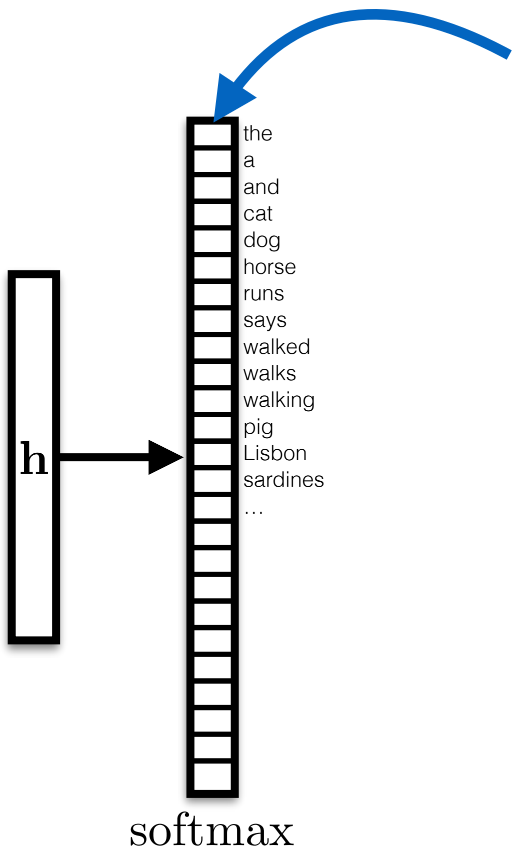
$$O_j = e^{I_j} / \sum_k e^{I_k}.$$

This **"softmax"** function is a generalisation of the logistic to multiple inputs. It also generalises maximum picking, or "Winner-Take-All", in the sense that that the outputs change smoothly, and equal inputs produce equal outputs. Although it looks rather cumbersome, and perhaps not really in the spirit of neural networks, those familiar with Markov random fields or statistical mechanics will know that it has convenient mathematical properties. Circuit designers will enjoy the simple transistor circuit which implements it.

Bridle. (1990) Probabilistic interpretation of feedforward classification

Example: Language Model

Each dimension corresponds to a word in a closed vocabulary, ***V***.

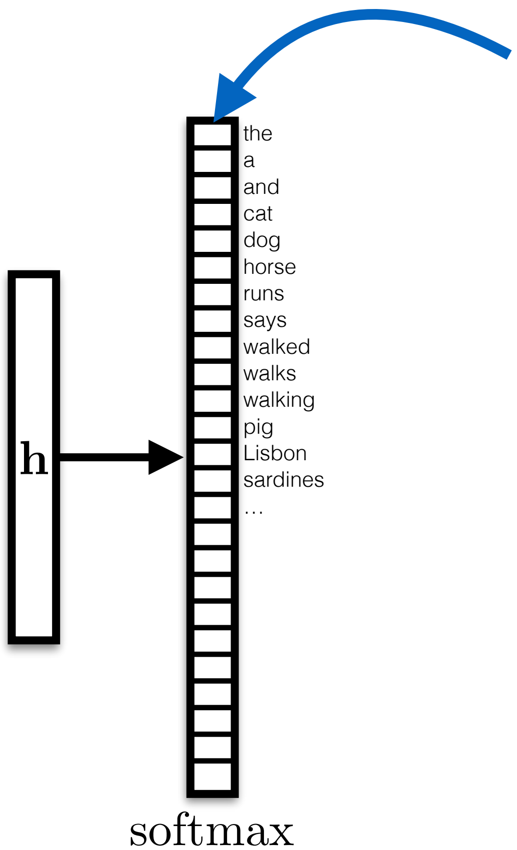


$$\mathbf{u} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

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

Example: Language Model

Each dimension corresponds to a word in a closed vocabulary, ***V***.

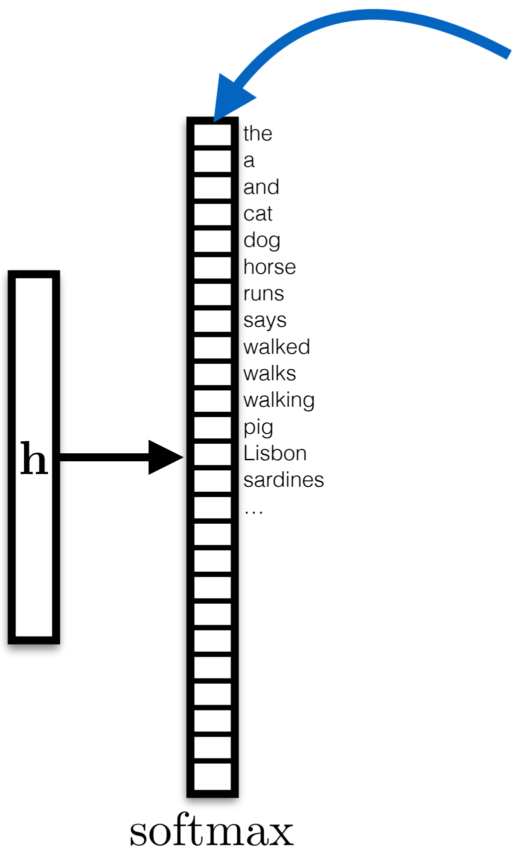


$$\mathbf{u} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

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

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

Example: Language Model



Each dimension corresponds to a word in a closed vocabulary, ***V***.

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

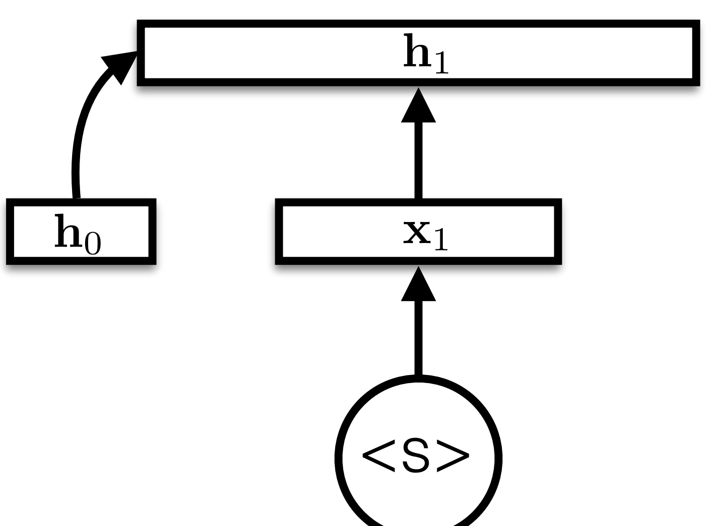
$$p_i = \frac{\exp u_i}{\sum_j \exp u_j}$$

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

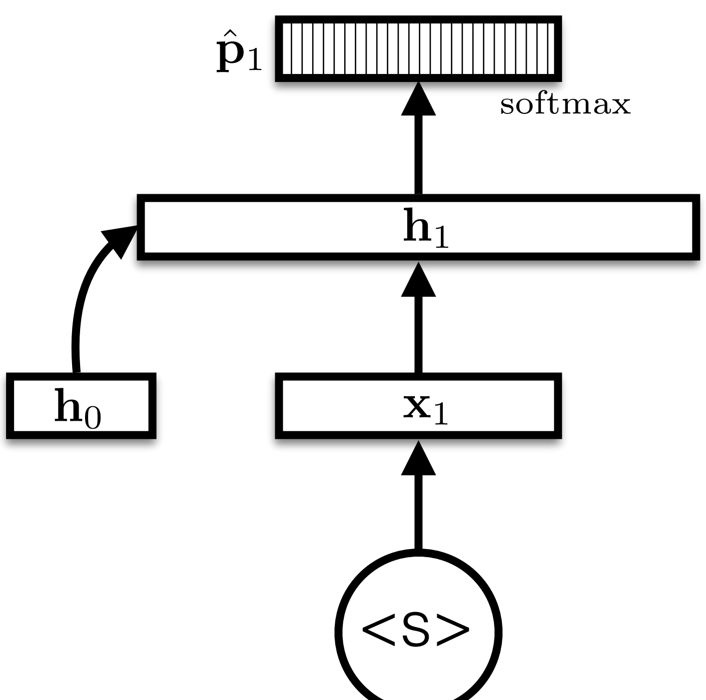
histories are sequences of words…

Example: Language Model

Example: Language Model

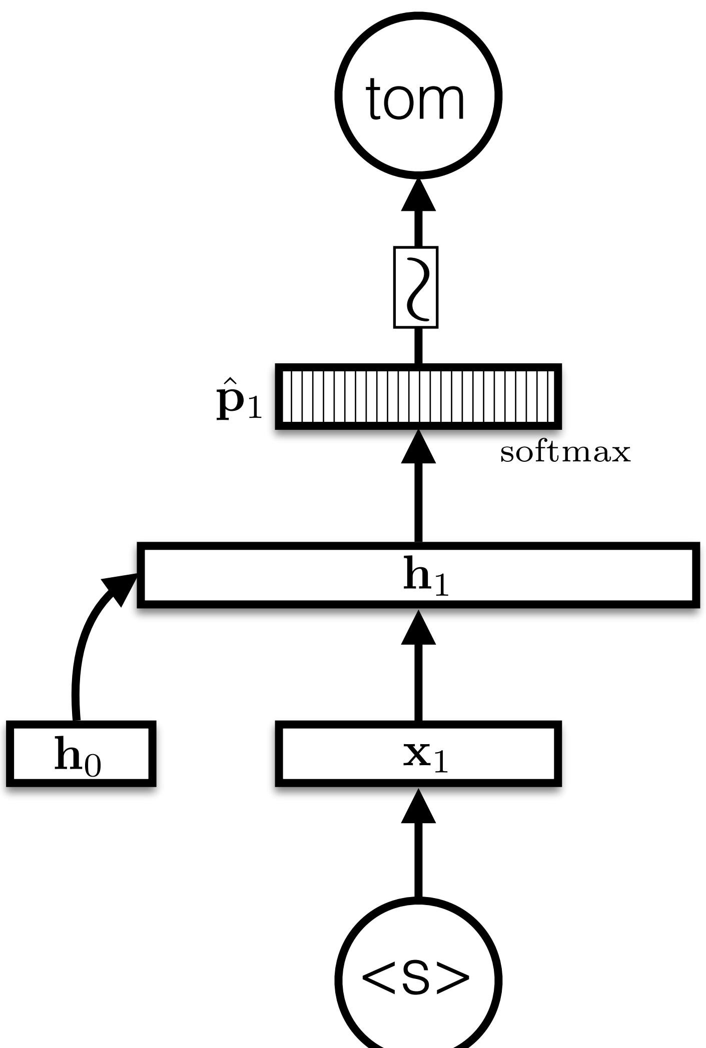


Example: Language Model



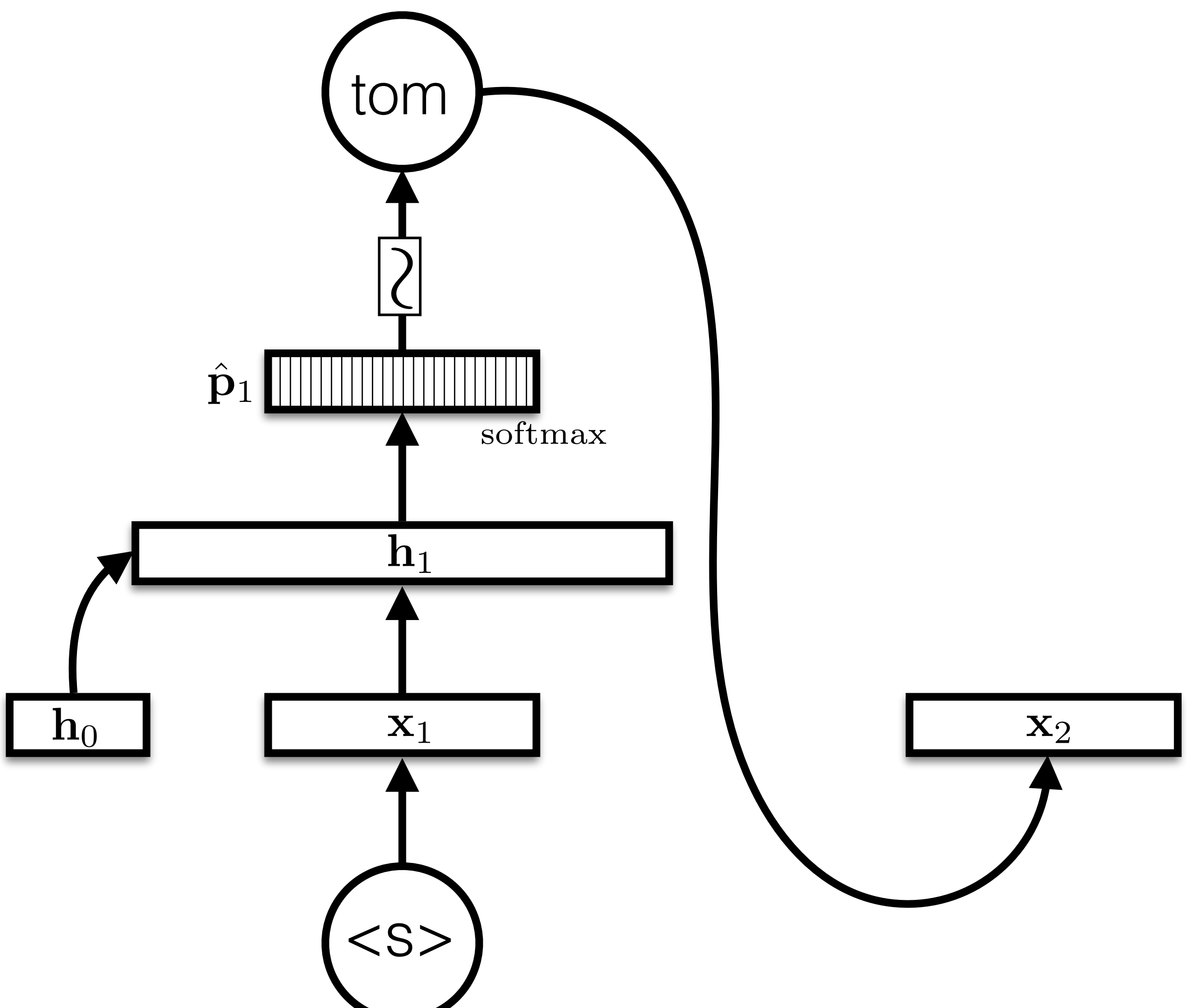
Example: Language Model

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



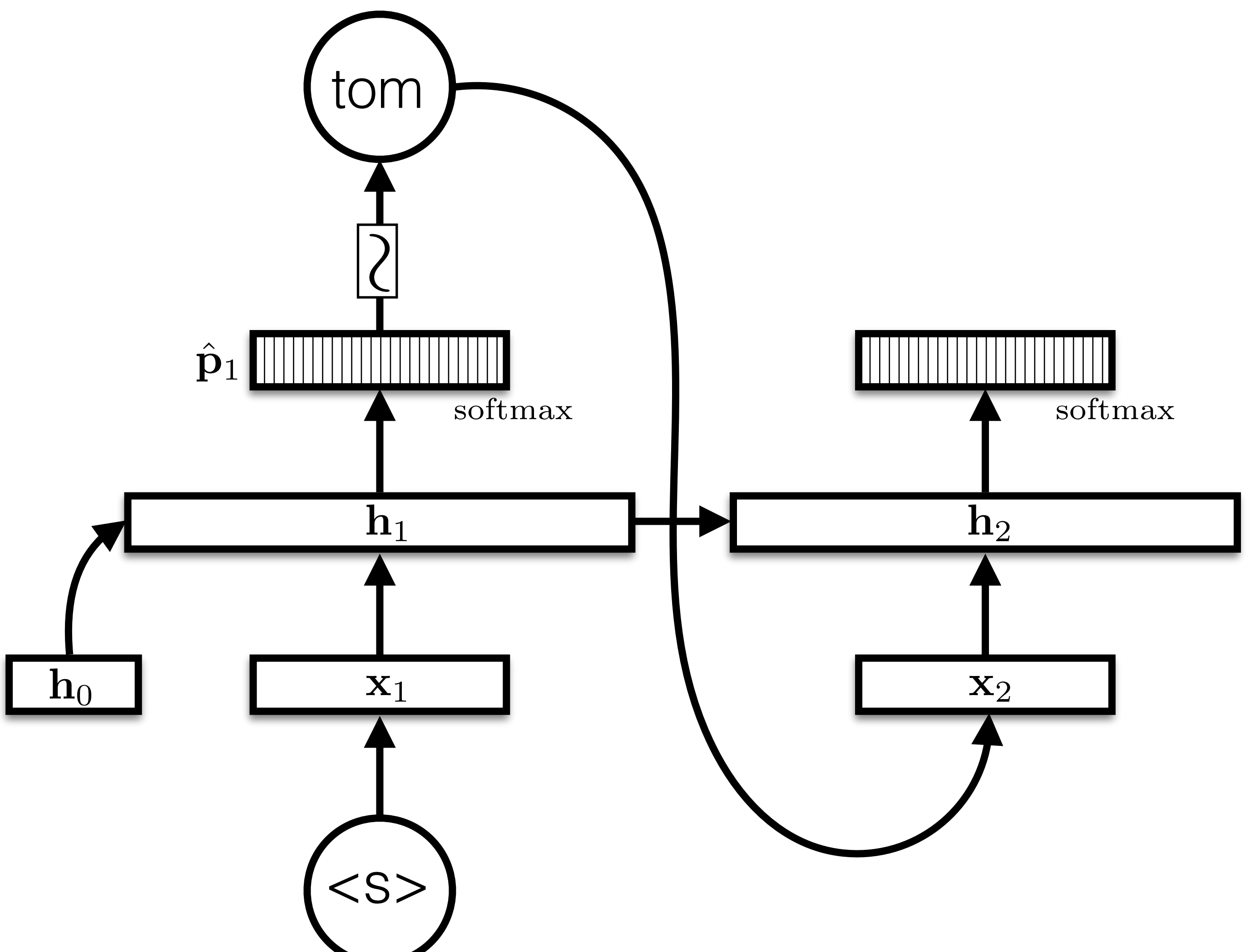
Example: Language Model

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



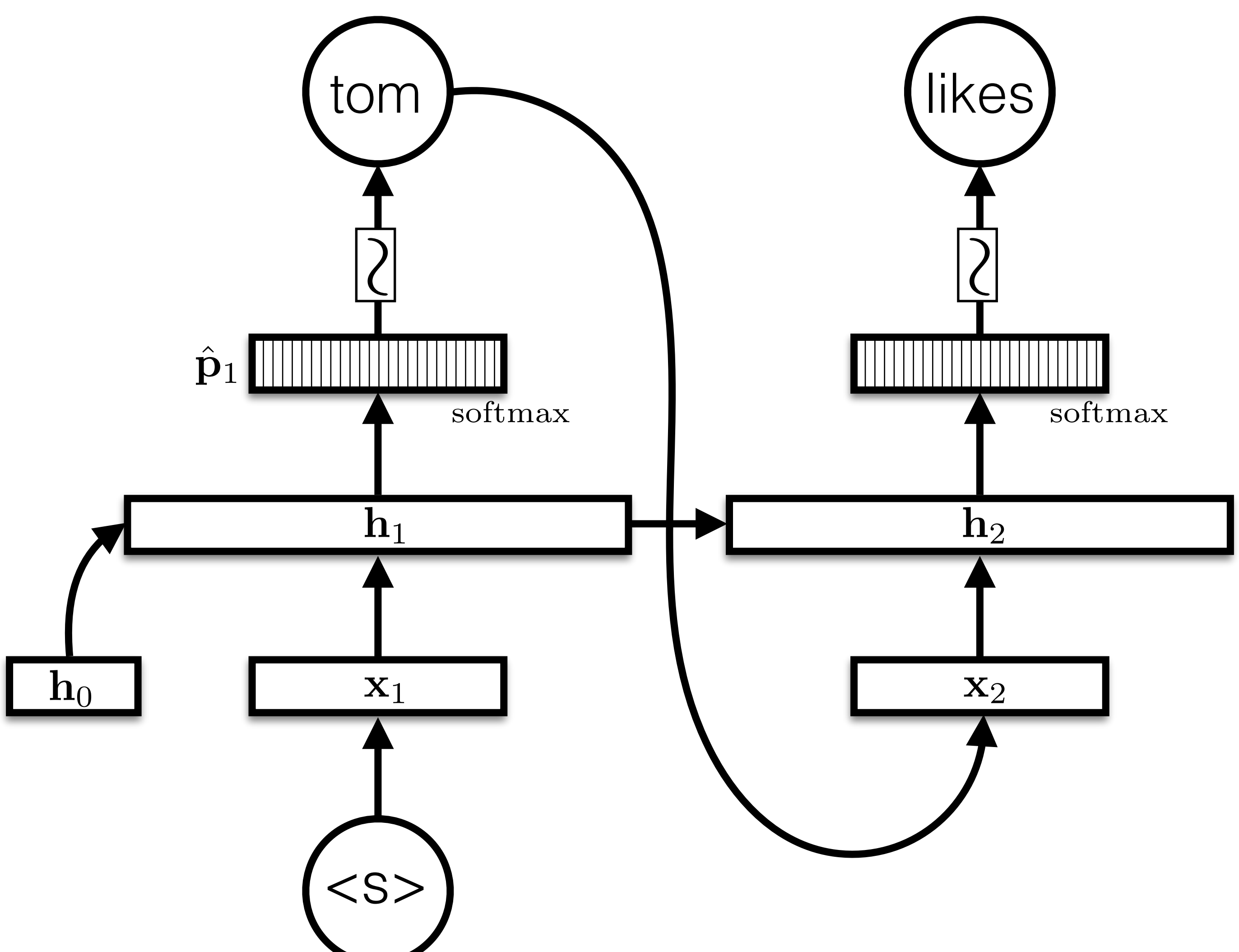
Example: Language Model

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



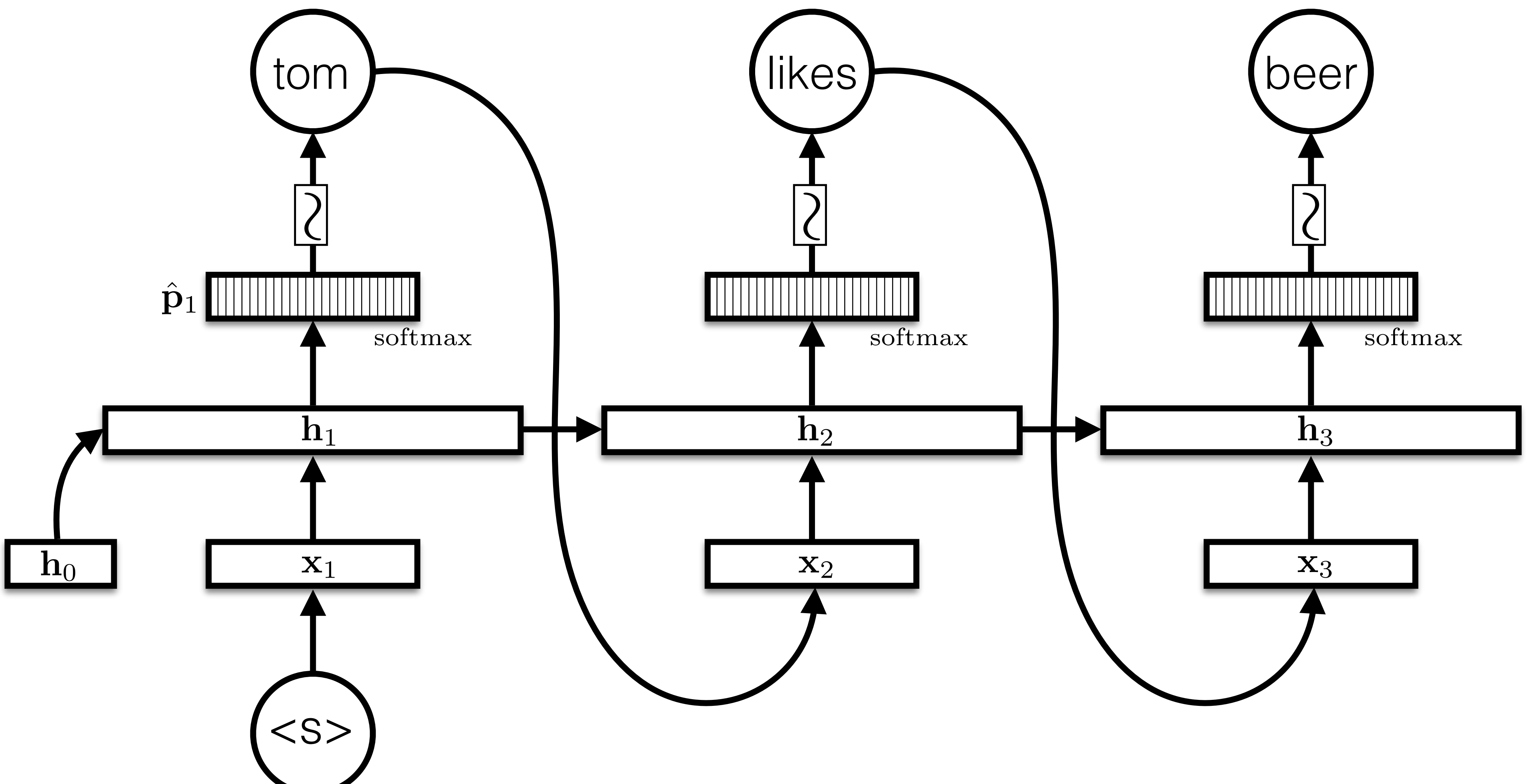
Example: Language Model

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



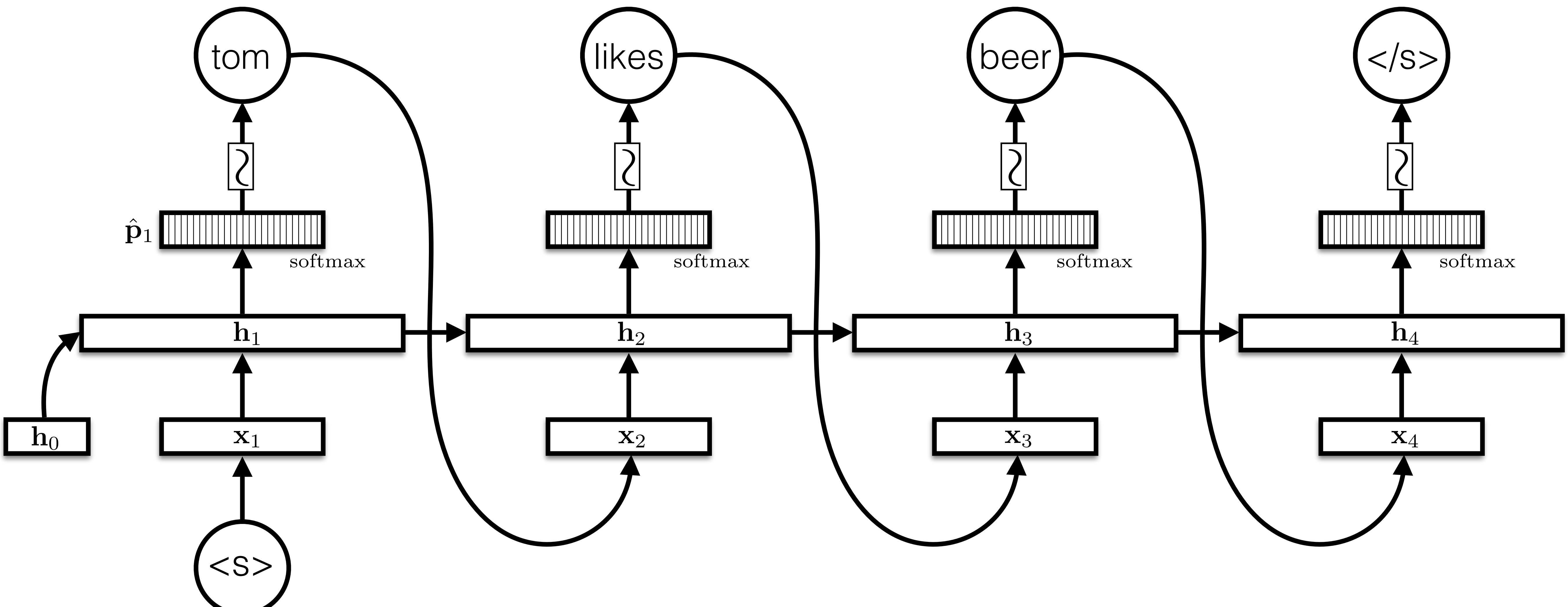
Example: Language Model

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

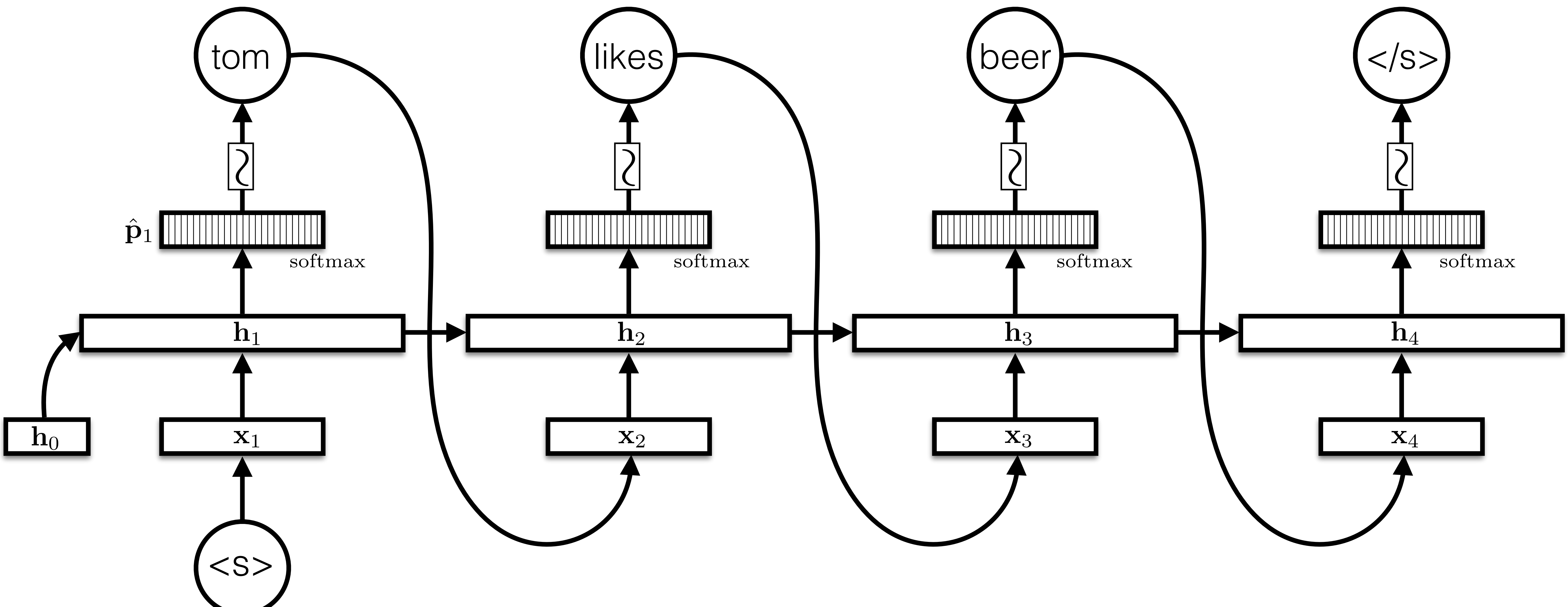


Example: Language Model

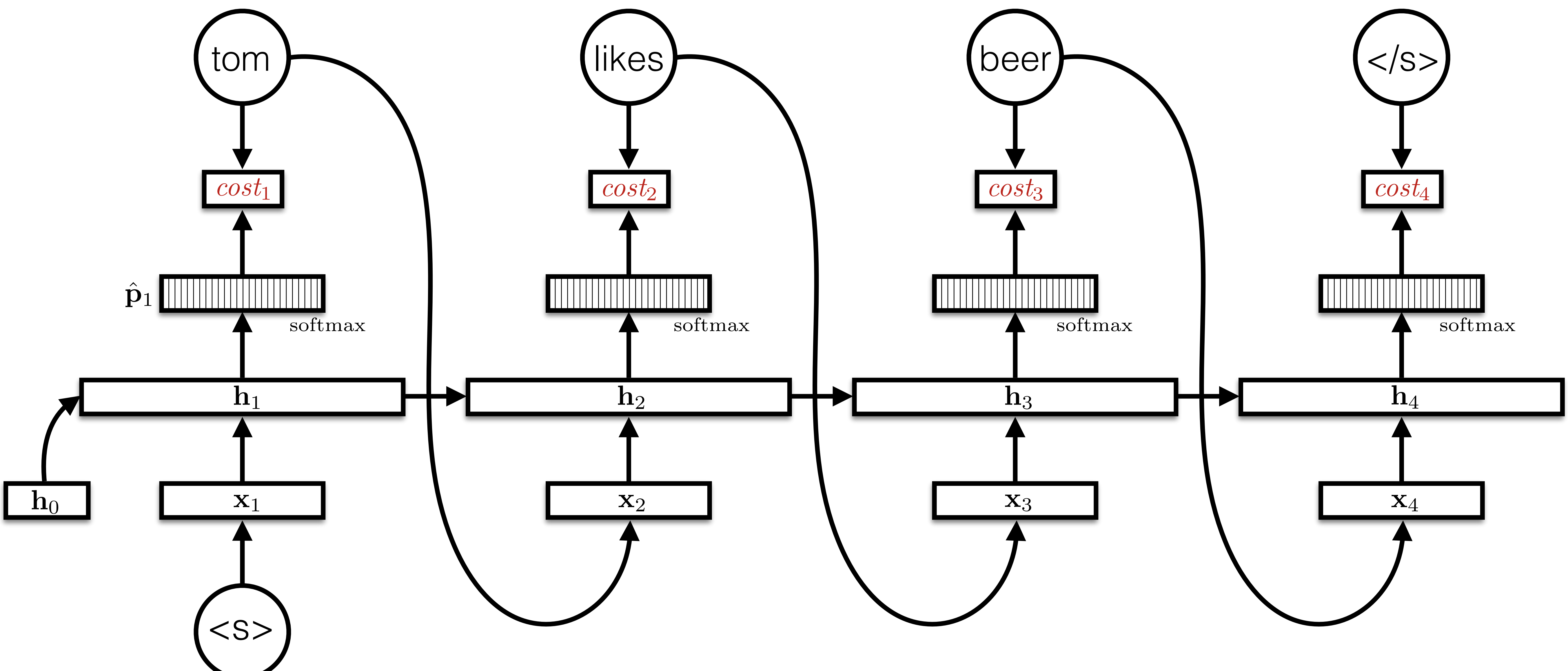
$$p(tom \mid \langle \mathbf{s} \rangle) \times p(likes \mid \langle \mathbf{s} \rangle, tom)$$
$$\times p(beer \mid \langle \mathbf{s} \rangle, tom, likes)$$
$$\times p(\langle /\mathbf{s} \rangle \mid \langle \mathbf{s} \rangle, tom, likes, beer)$$



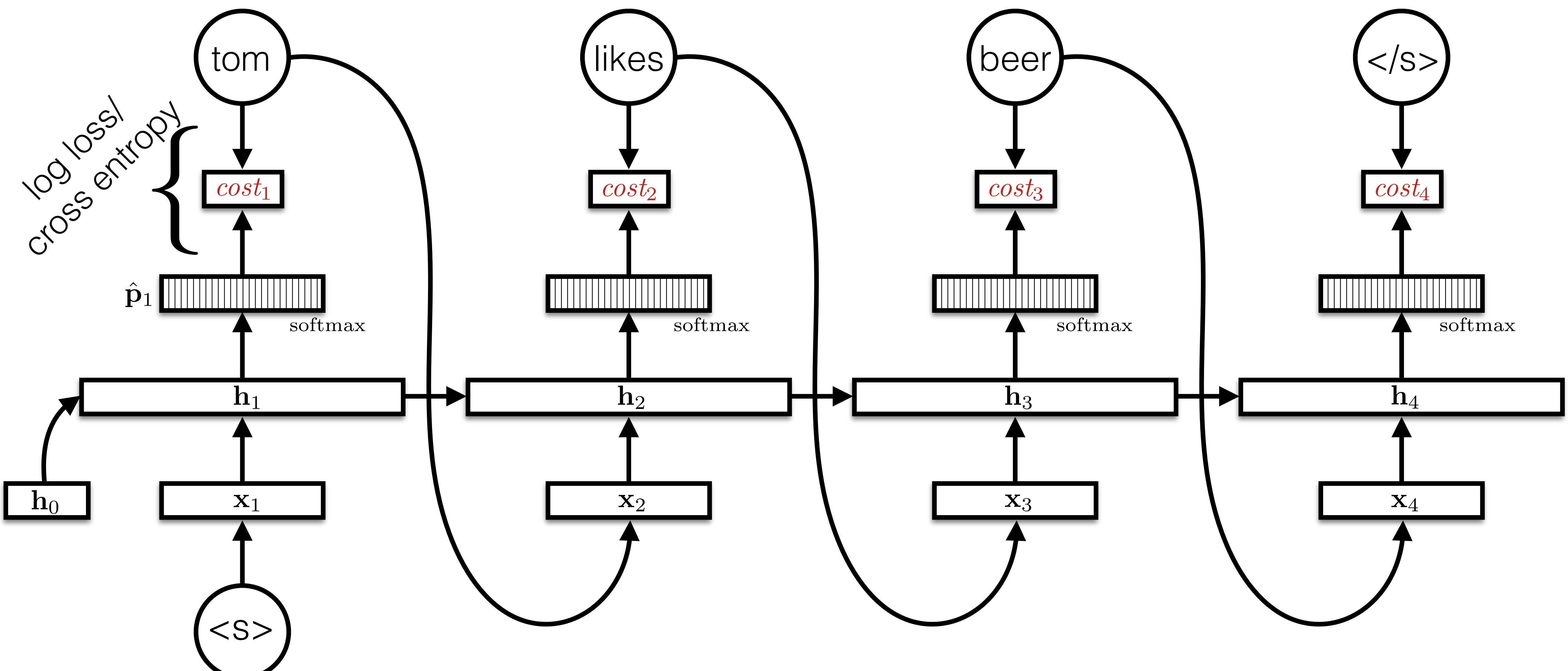
Language Model Training



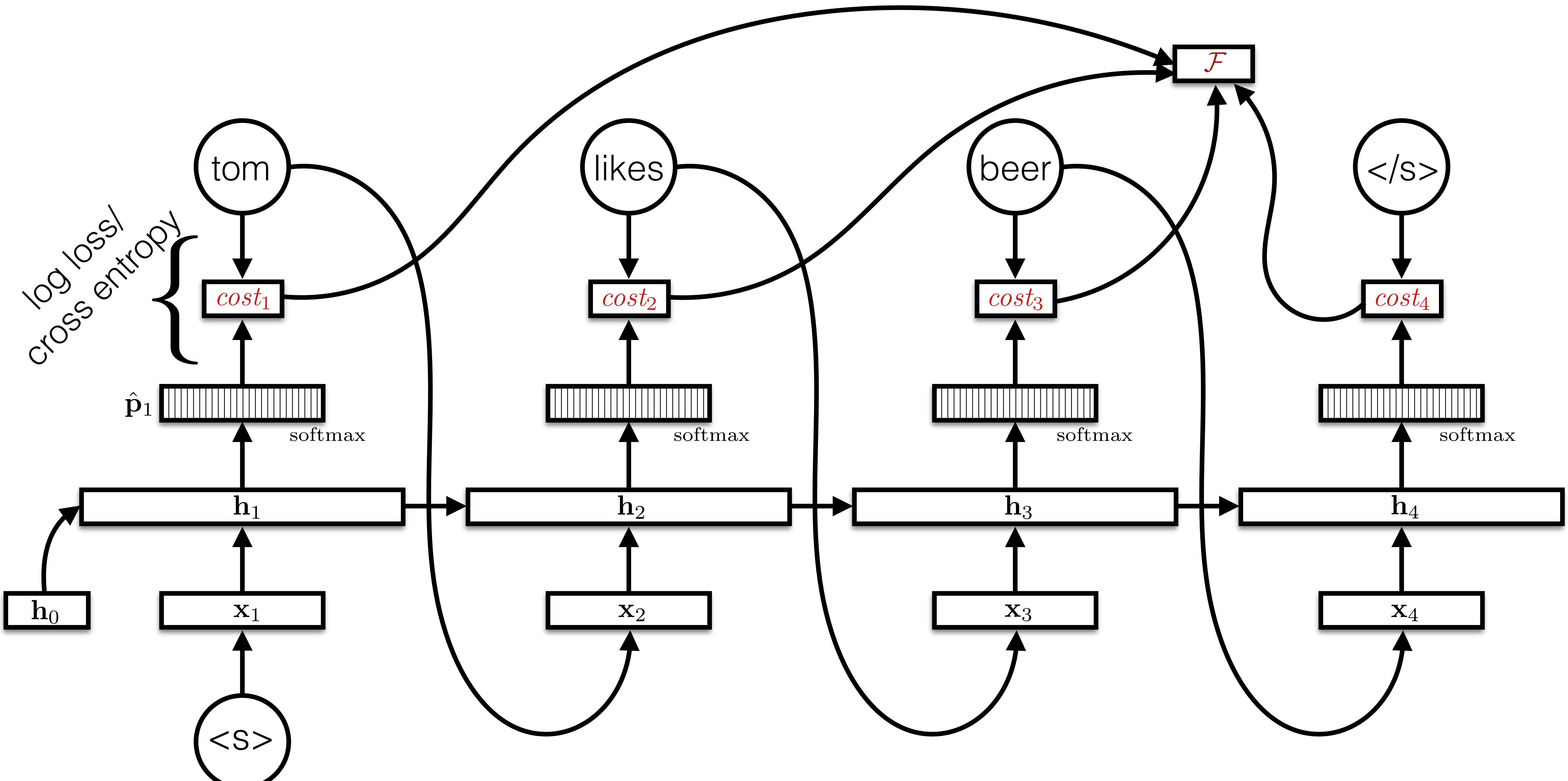
Language Model Training



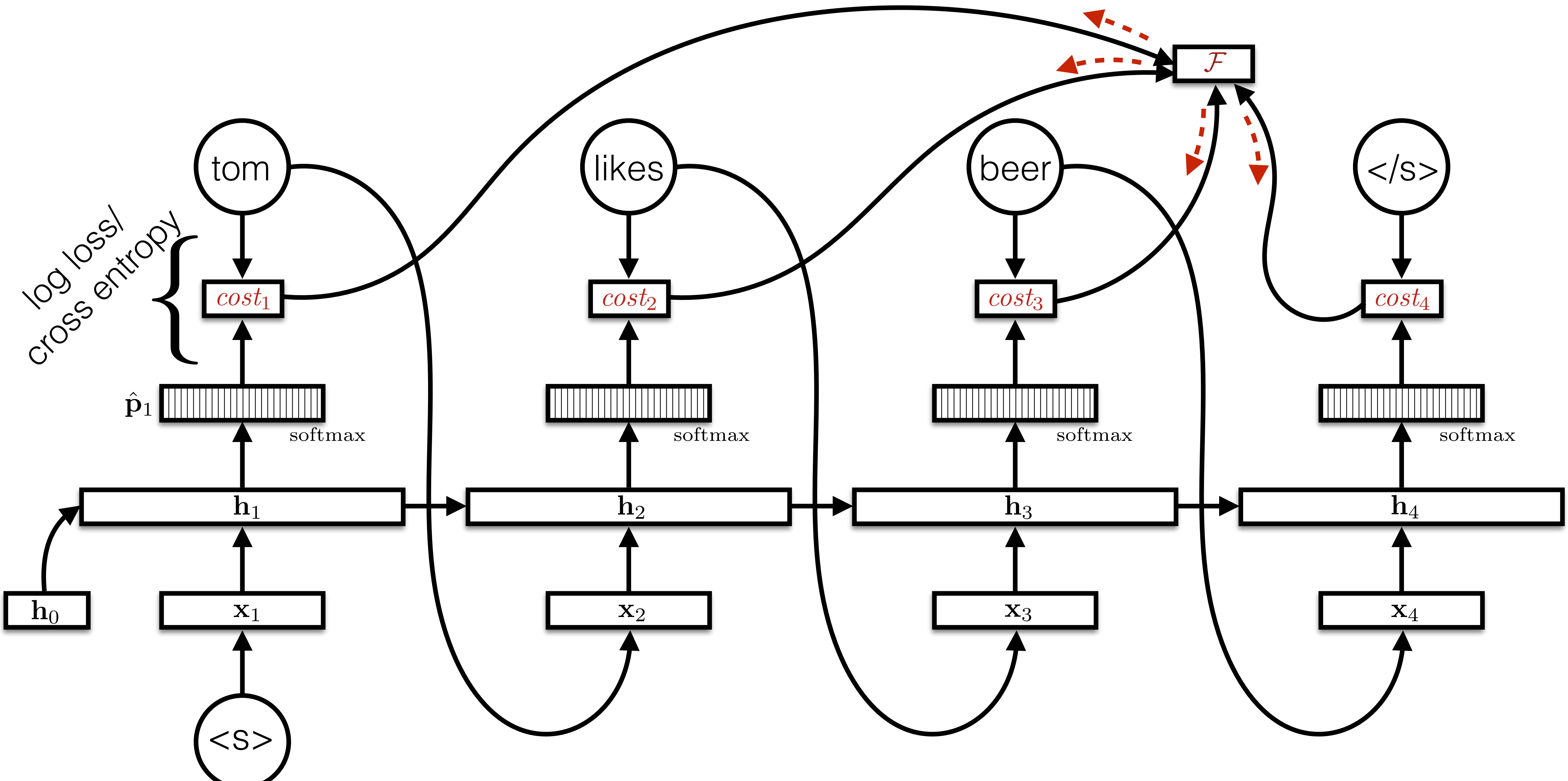
Language Model Training



Language Model Training



Language Model Training



Language Model Training

- The cross-entropy objective seeks the **maximum likelihood** (MLE) parameters.

  "Find the parameters that make the training data most likely."

- You will overfit:

  - Stop training early, based on a validation set

  - Weight decay / other weight regularizers

  - Dropout variants during training

- In contrast to count-based models, RNNs don't have problems with "zeros".

RNN Language Models

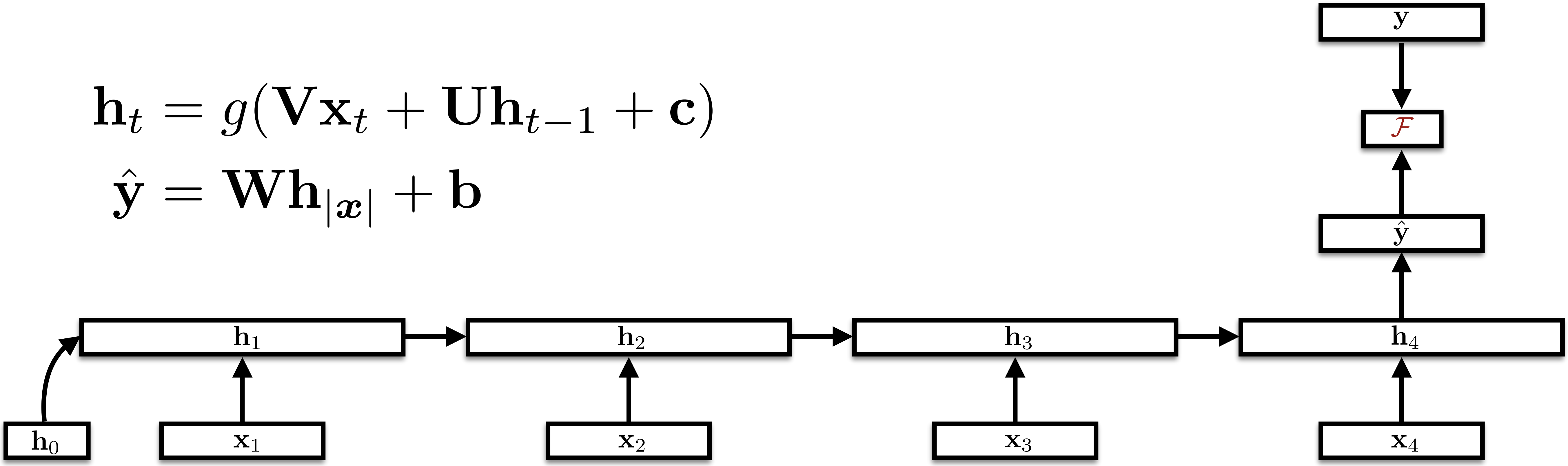
- Unlike Markov (*n*-gram) models, RNNs never forget
  - However we will see they might have trouble learning to use their memories (more soon…)
- Algorithms
  - Sample a sequence from the probability distribution defined by the RNN
  - Train the RNN to minimize cross entropy (aka MLE)
  - What about: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

Questions?

Training Challenges

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

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



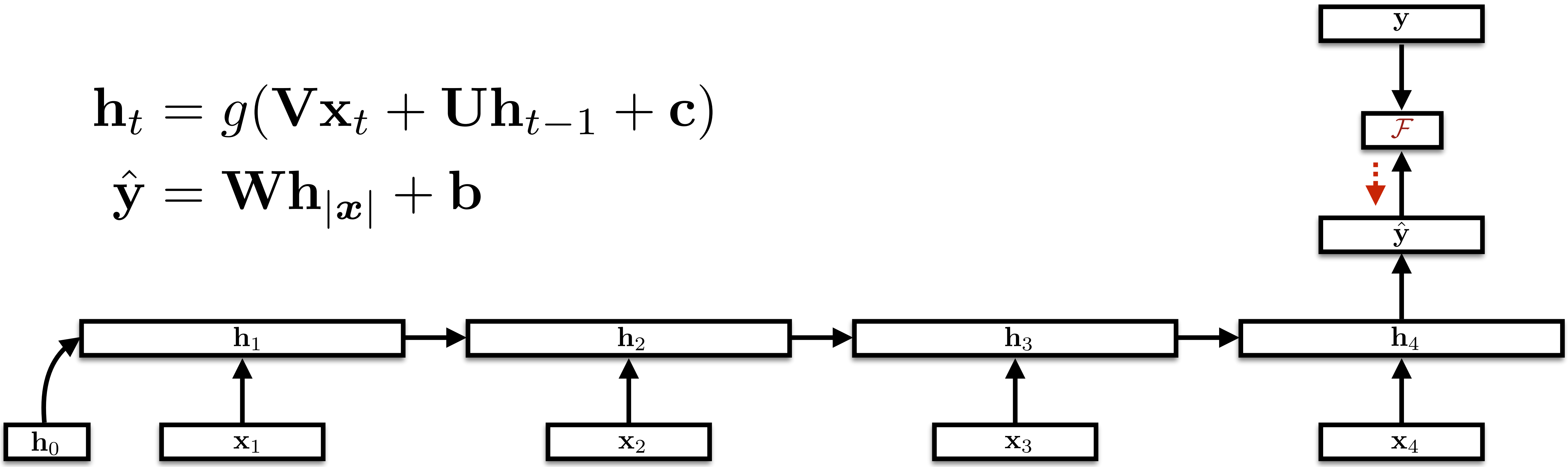
What happens to gradients as you go back in time?

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

Training Challenges

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

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



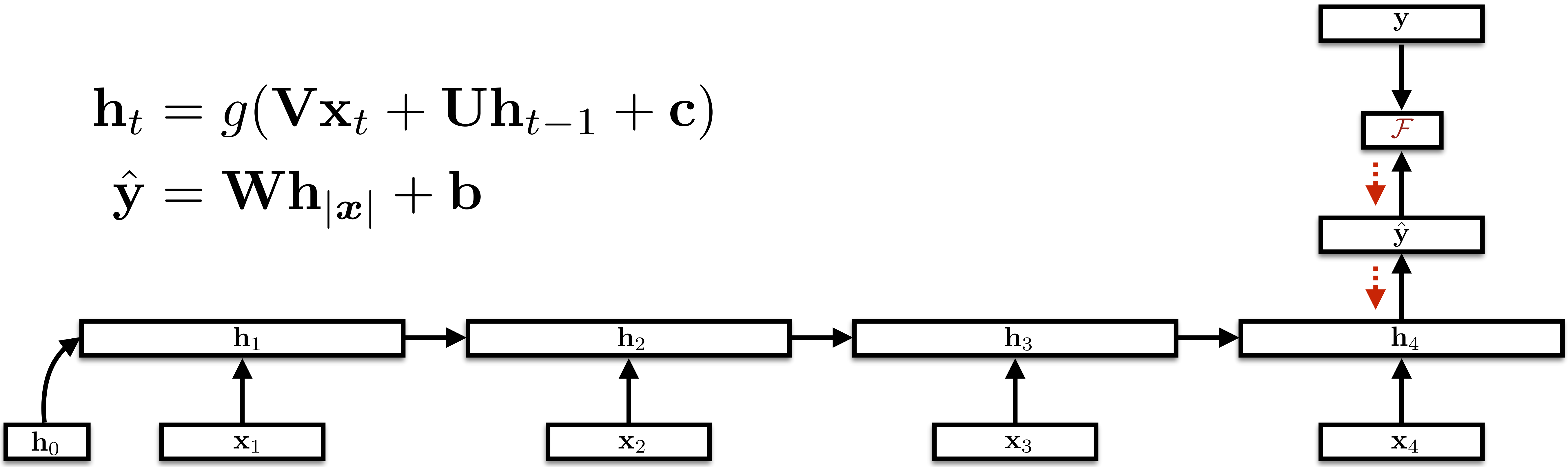
What happens to gradients as you go back in time?

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

Training Challenges

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

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

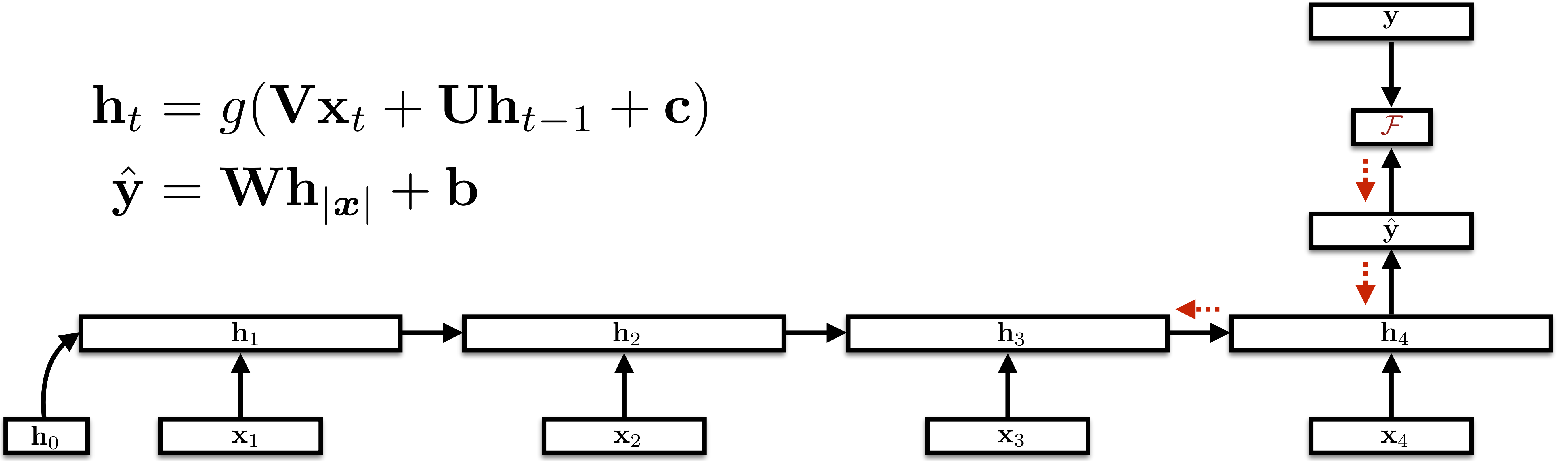


What happens to gradients as you go back in time?

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

Training Challenges

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

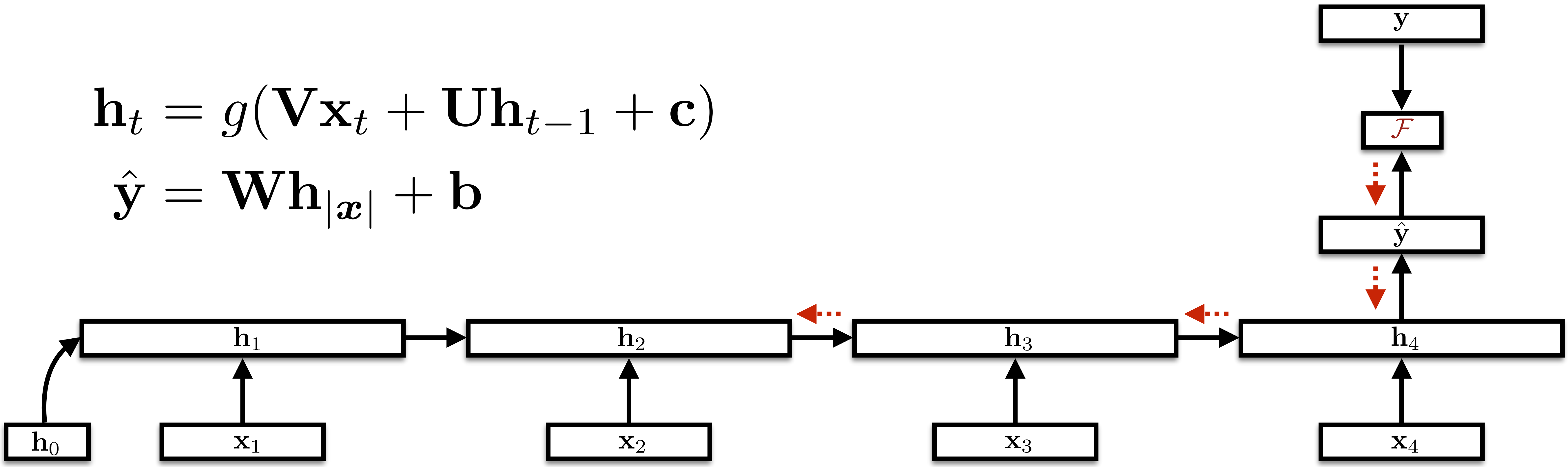


What happens to gradients as you go back in time?

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

Training Challenges

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



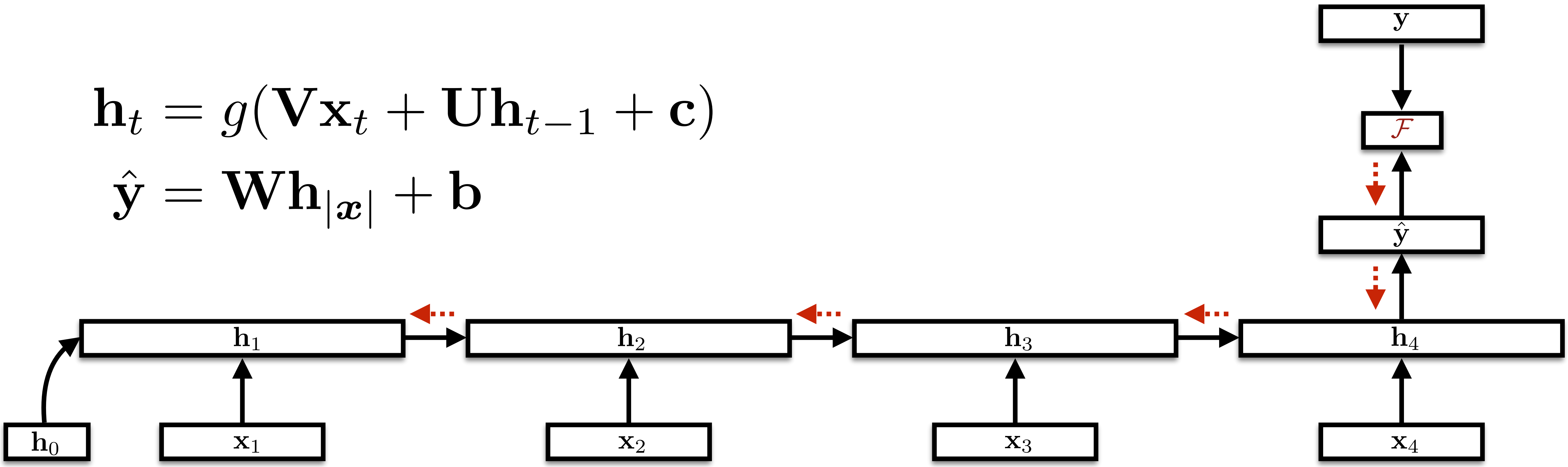
What happens to gradients as you go back in time?

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

Training Challenges

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

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



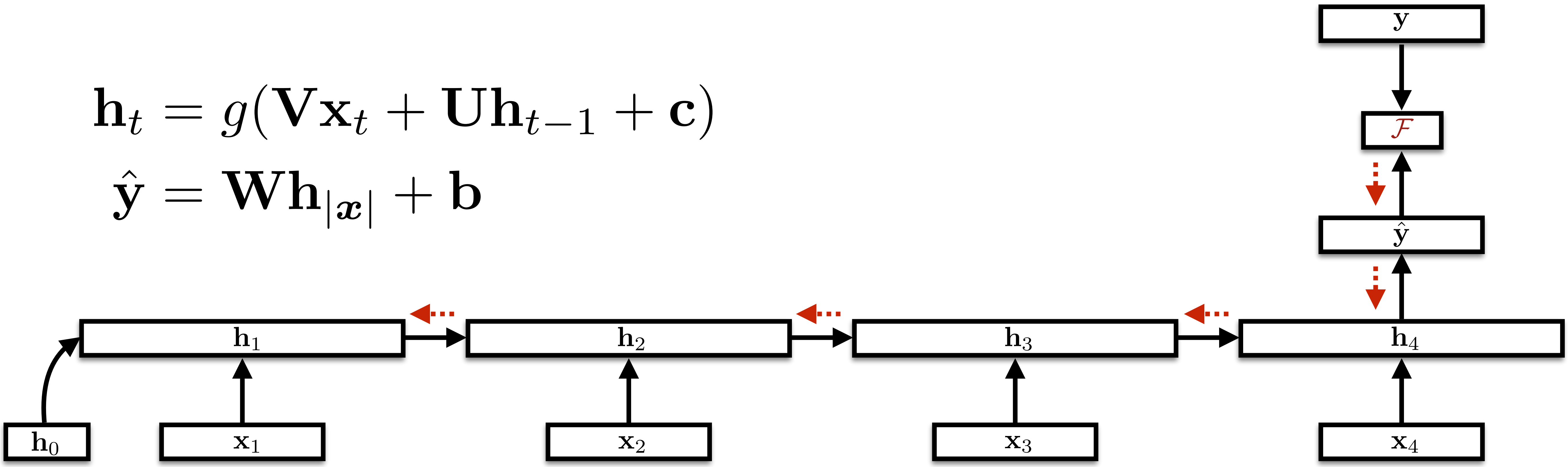
What happens to gradients as you go back in time?

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

Training Challenges

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

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

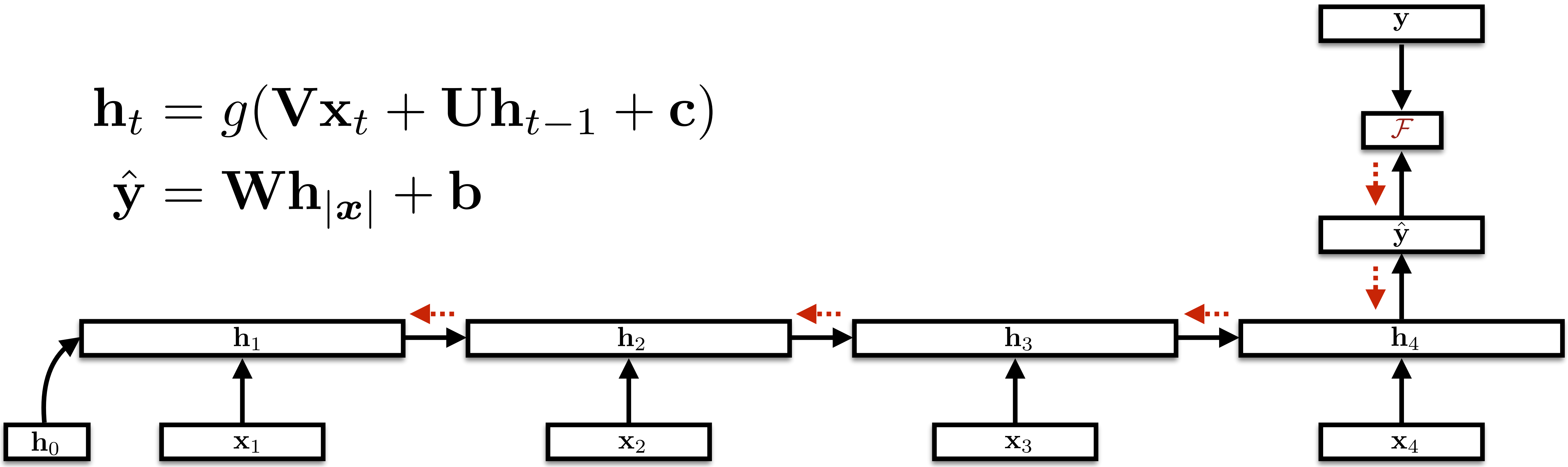


What happens to gradients as you go back in time?

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

Training Challenges

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



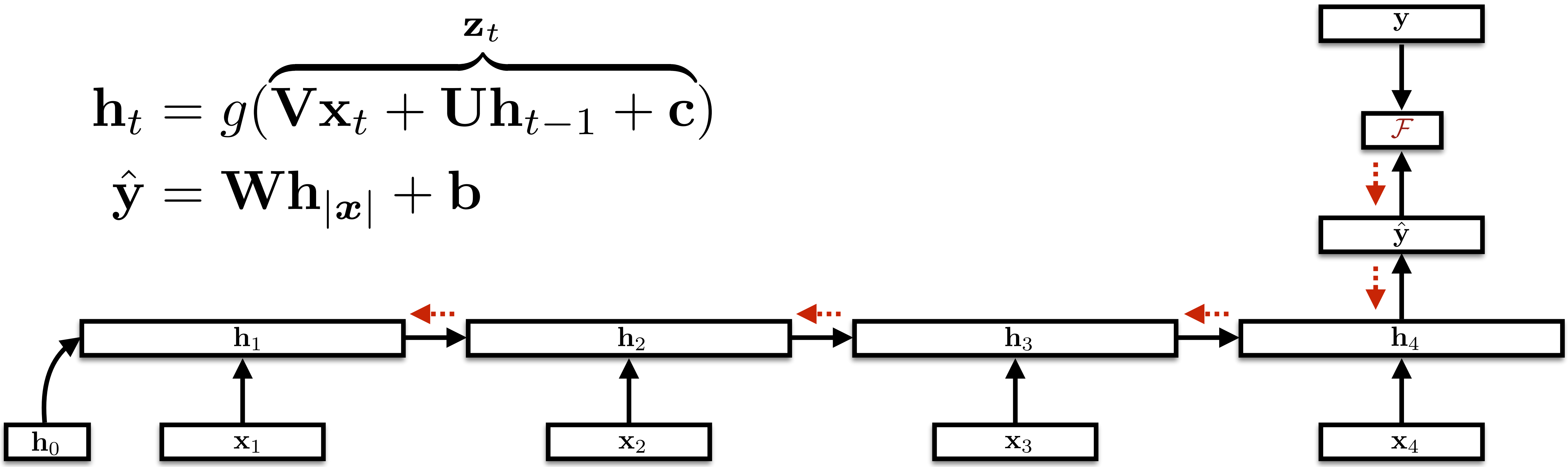
What happens to gradients as you go back in time?

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

Training Challenges

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

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



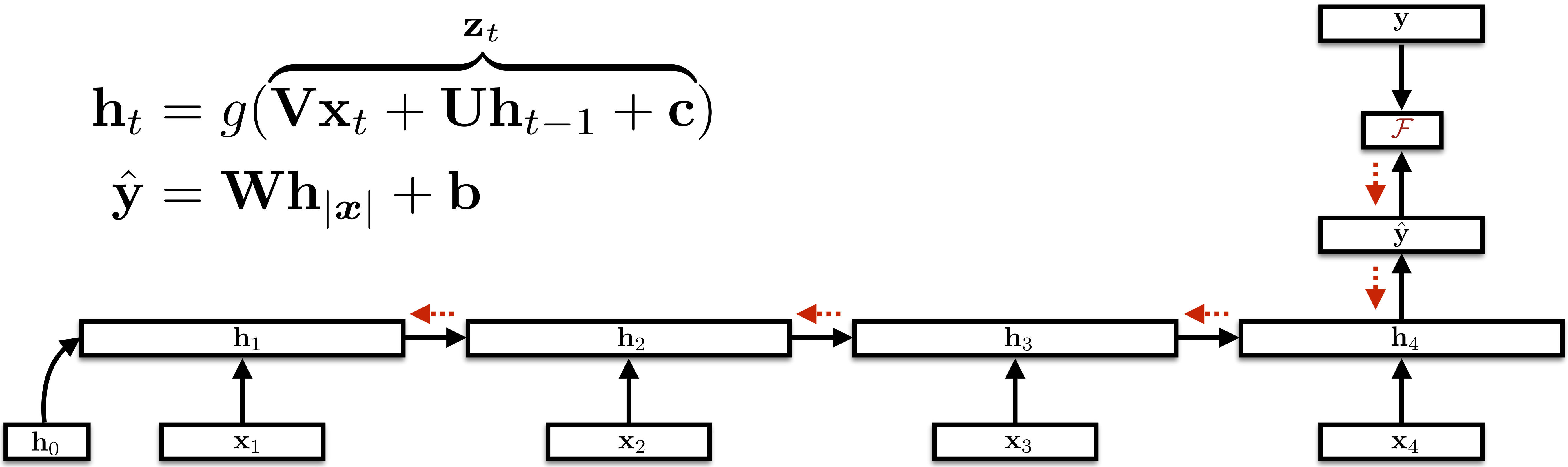
What happens to gradients as you go back in time?

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

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

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

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

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

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

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

Training Challenges

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

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

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

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

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

Training Challenges

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

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

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

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

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

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

Training Challenges

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

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

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

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

Training Challenges

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

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

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

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

Three cases: largest eigenvalue is
exactly 1; gradient propagation is stable
<1; gradient vanishes (exponential decay)
>1; gradient explodes (exponential growth)

Vanishing Gradients

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

Alternative RNNs

- Long short-term memories (LSTMs; Hochreiter and Schmidthuber, 1997)

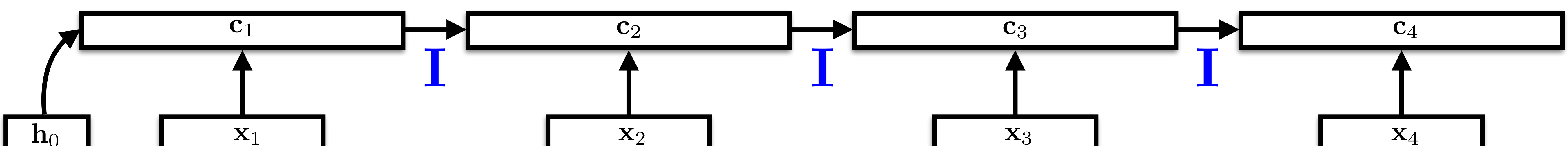
- Gated recurrent units (GRUs; Cho et al., 2014)

- Intuition instead of **multiplying** across time (which leads to exponential growth), we want the error to be **constant**.

    - What is a function whose Jacobian has a spectral radius of exactly **I**: the identity function

Memory cells

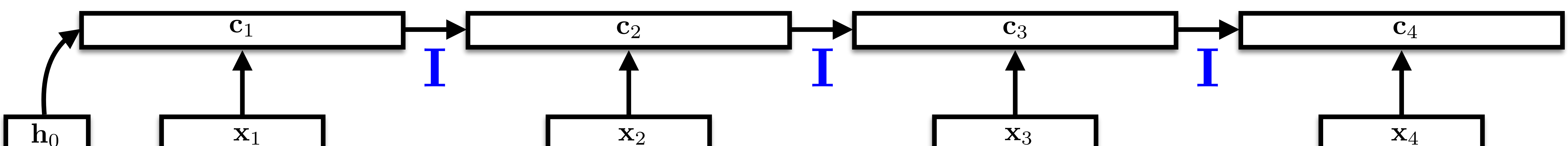
$$\mathbf{c}_t = \mathbf{c}_{t-1} + f(\mathbf{x}_t)$$



Memory cells

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

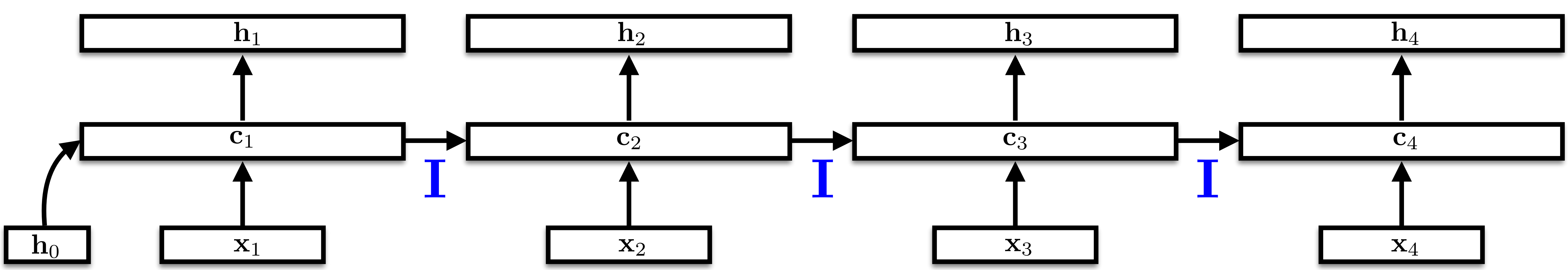
$$f(\mathbf{v}) = \tanh(\mathbf{W}\mathbf{v} + \mathbf{b})$$



Memory cells

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

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

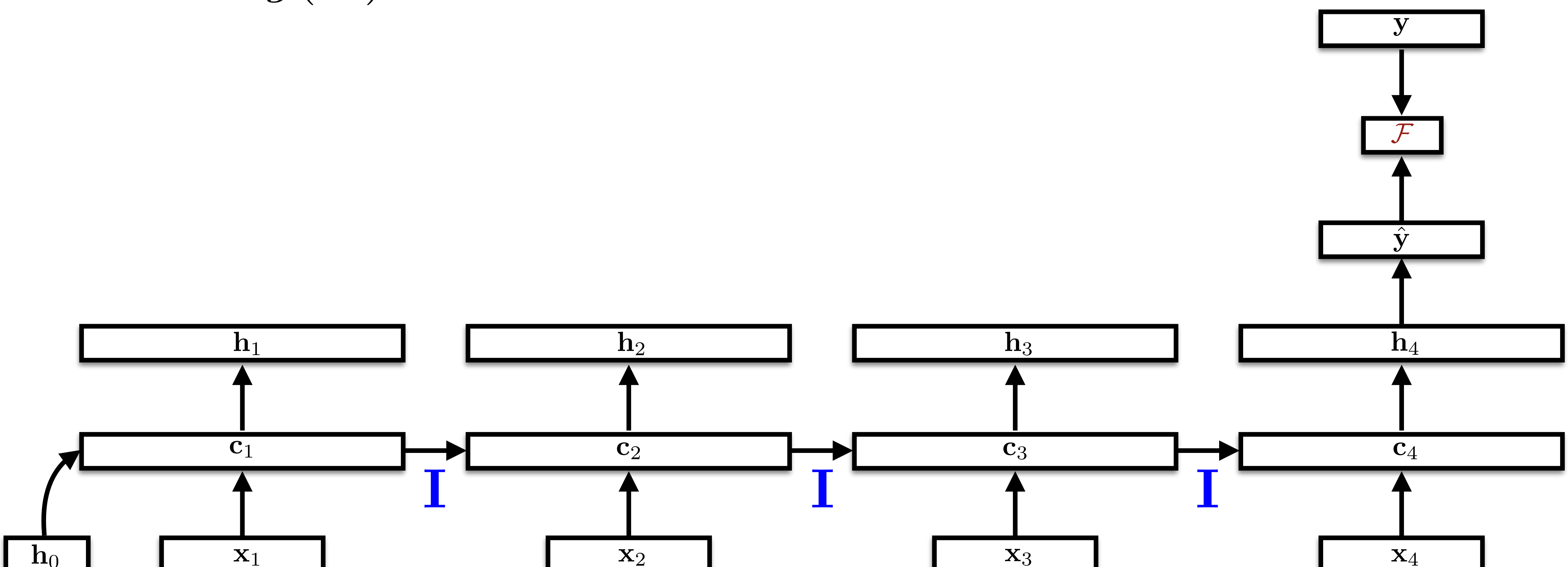


Memory cells

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

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

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



Memory cells

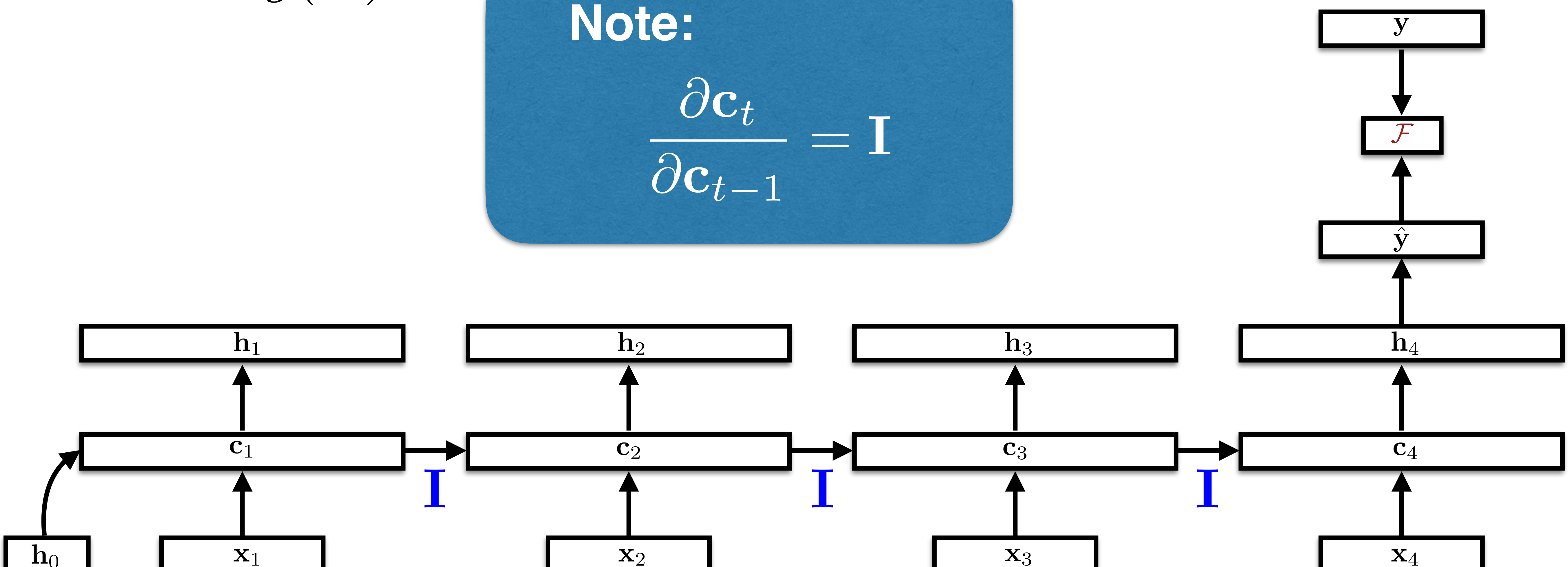
$$\mathbf{c}_t = \mathbf{c}_{t-1} + f(\mathbf{x}_t)$$

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

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

Note:

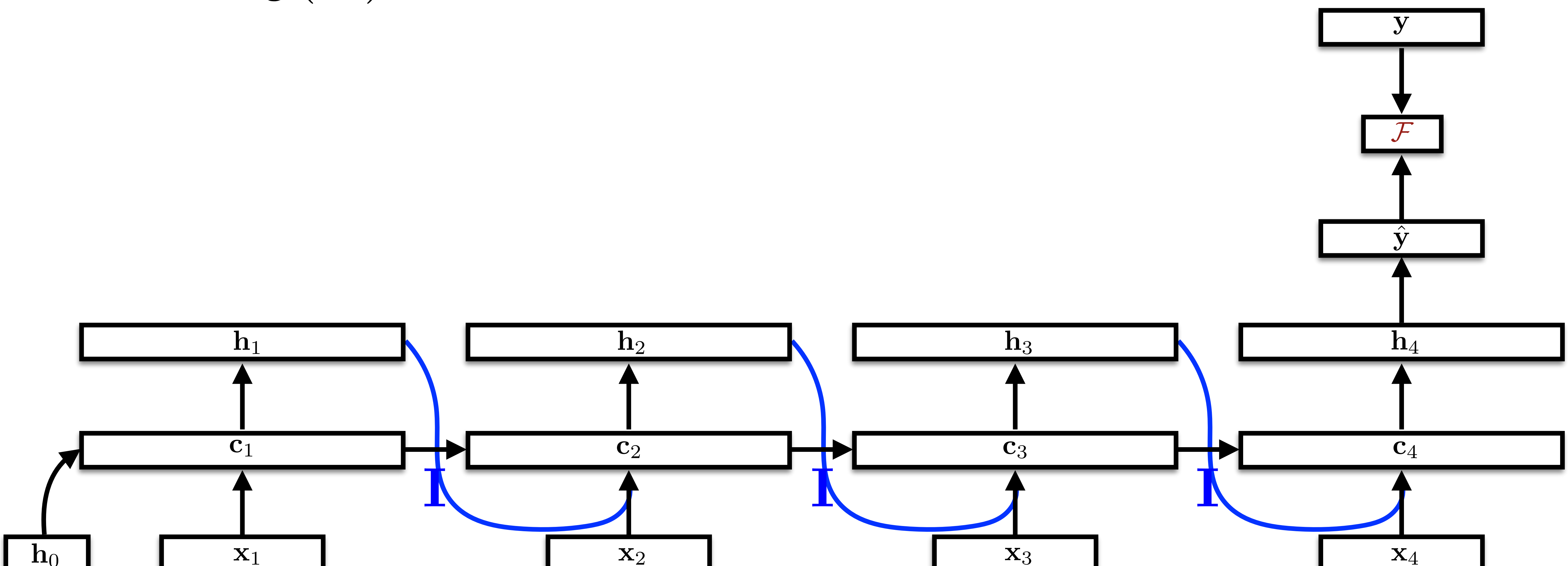
$$\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = \mathbf{I}$$



Memory cells

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

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



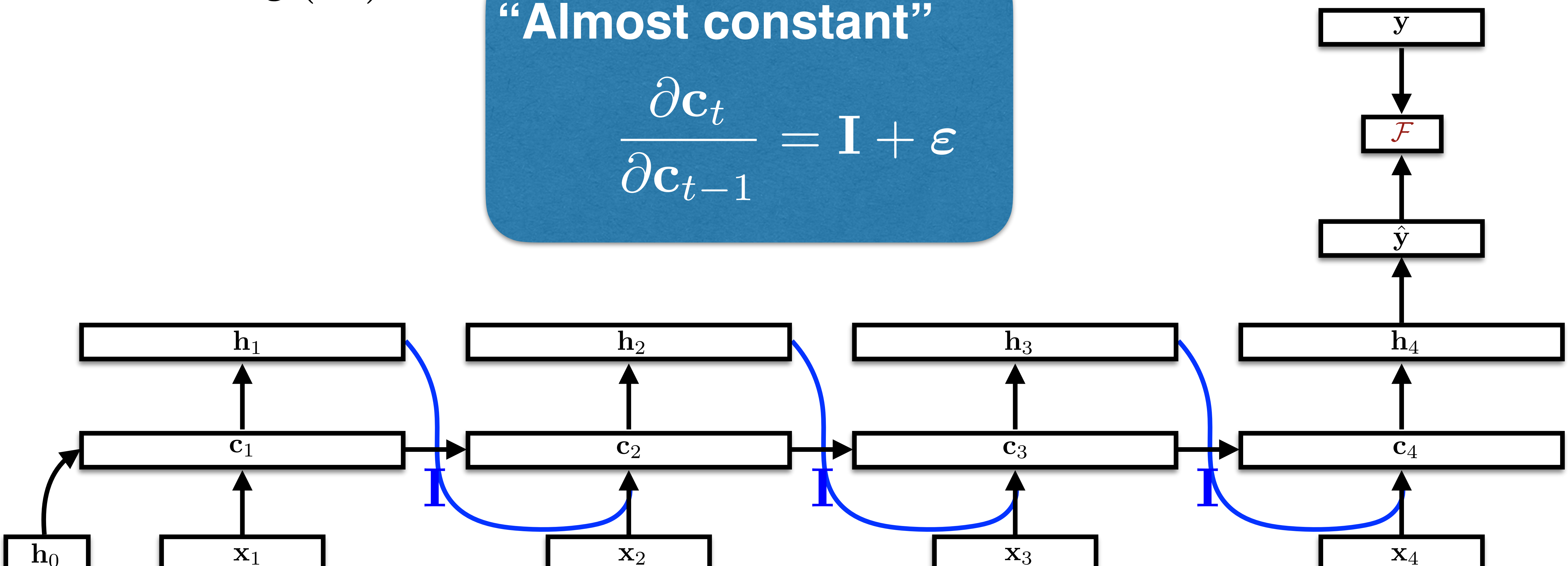
Memory cells

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

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

"Almost constant"

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



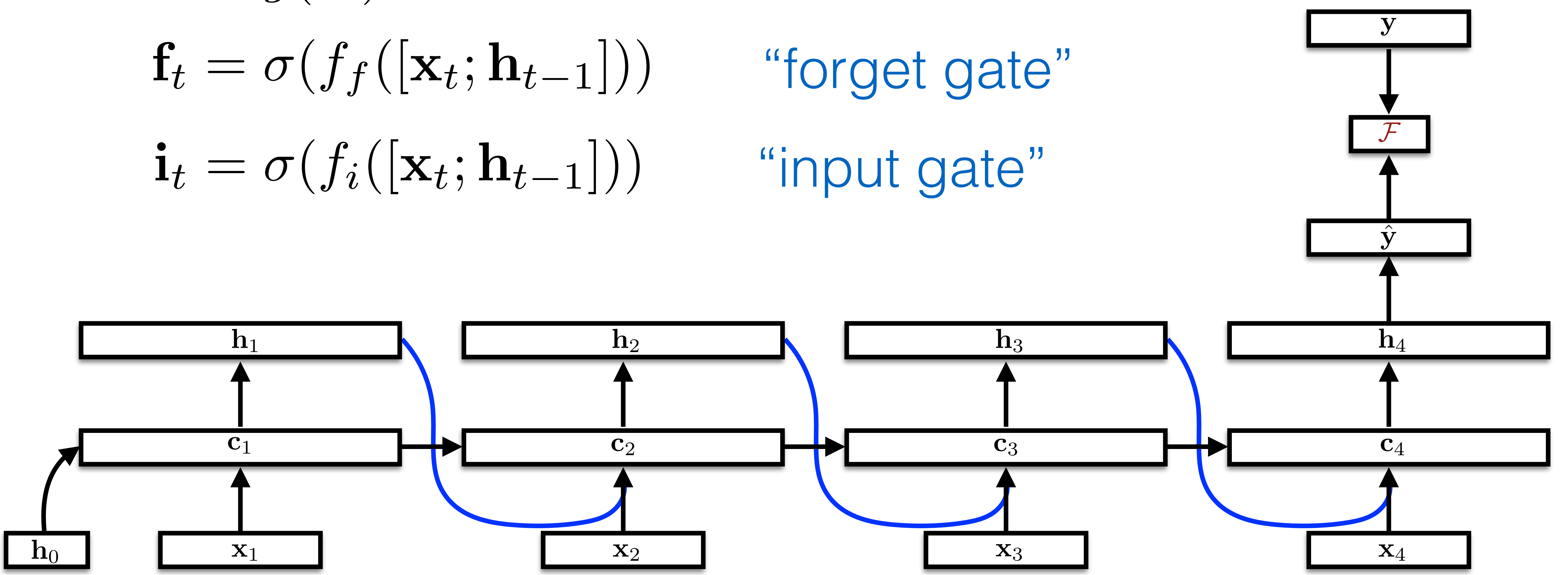
Memory cells

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

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

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

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



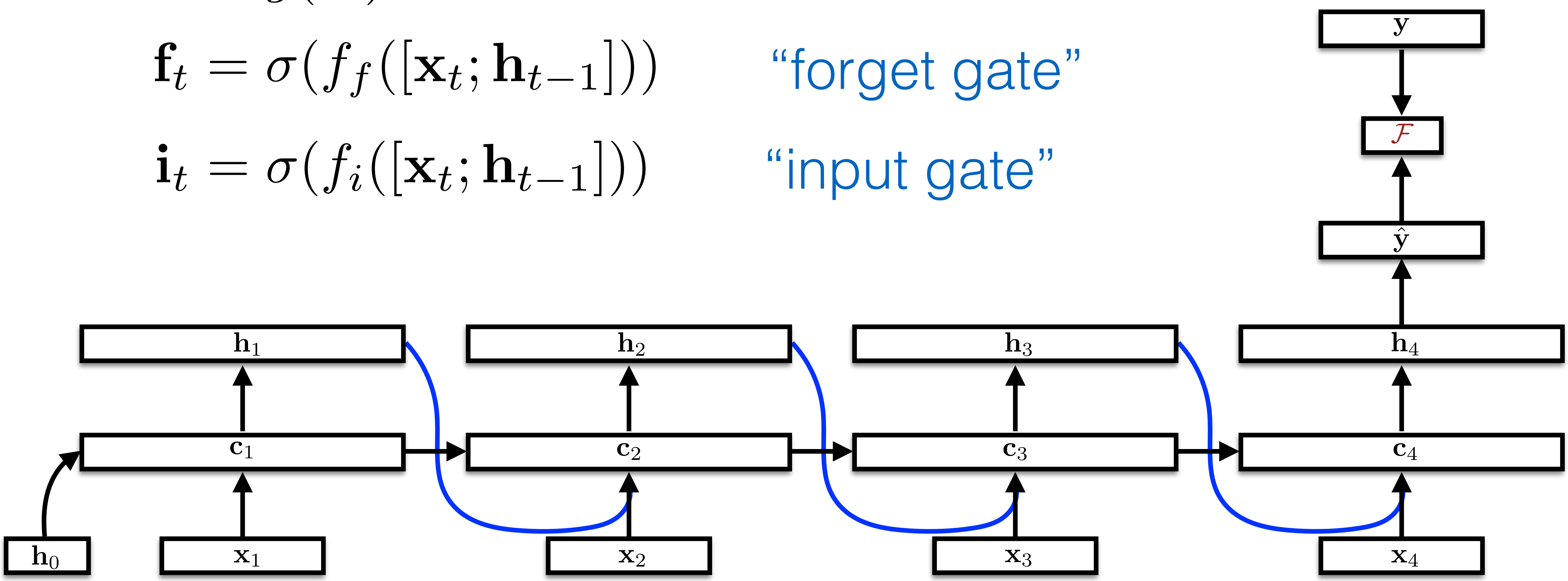
Memory cells

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

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

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

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



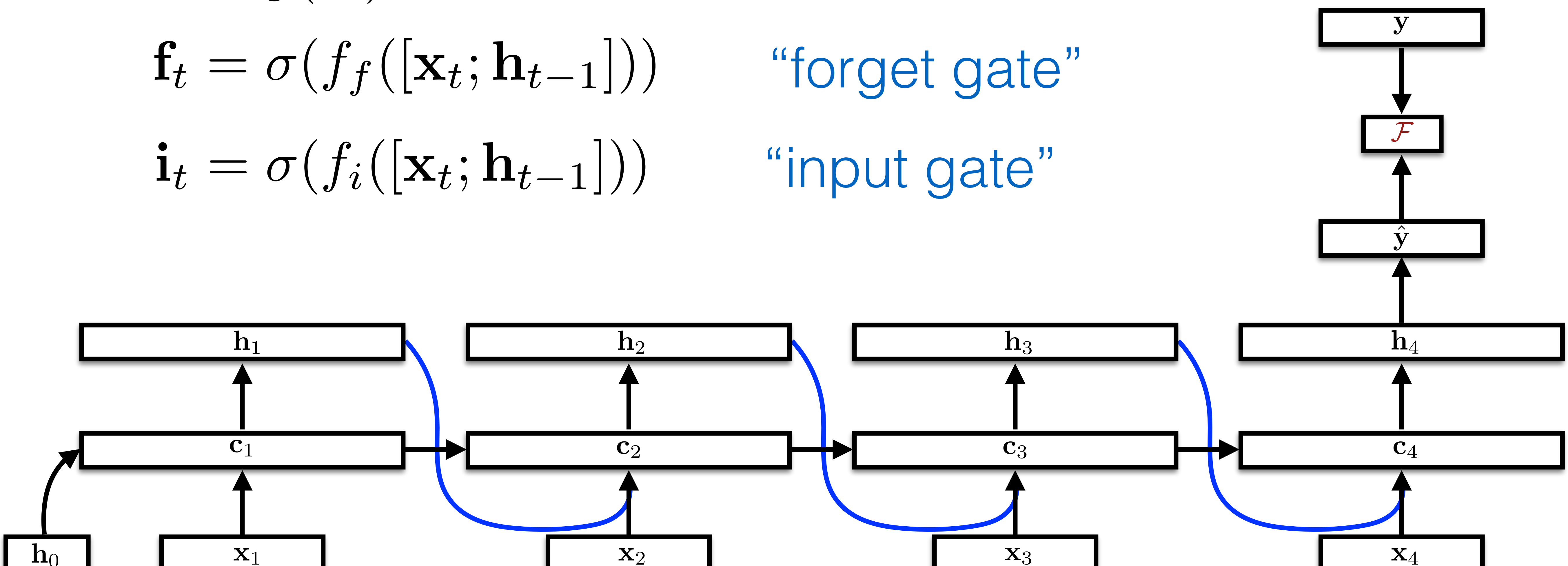
Memory cells

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

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

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

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



LSTM

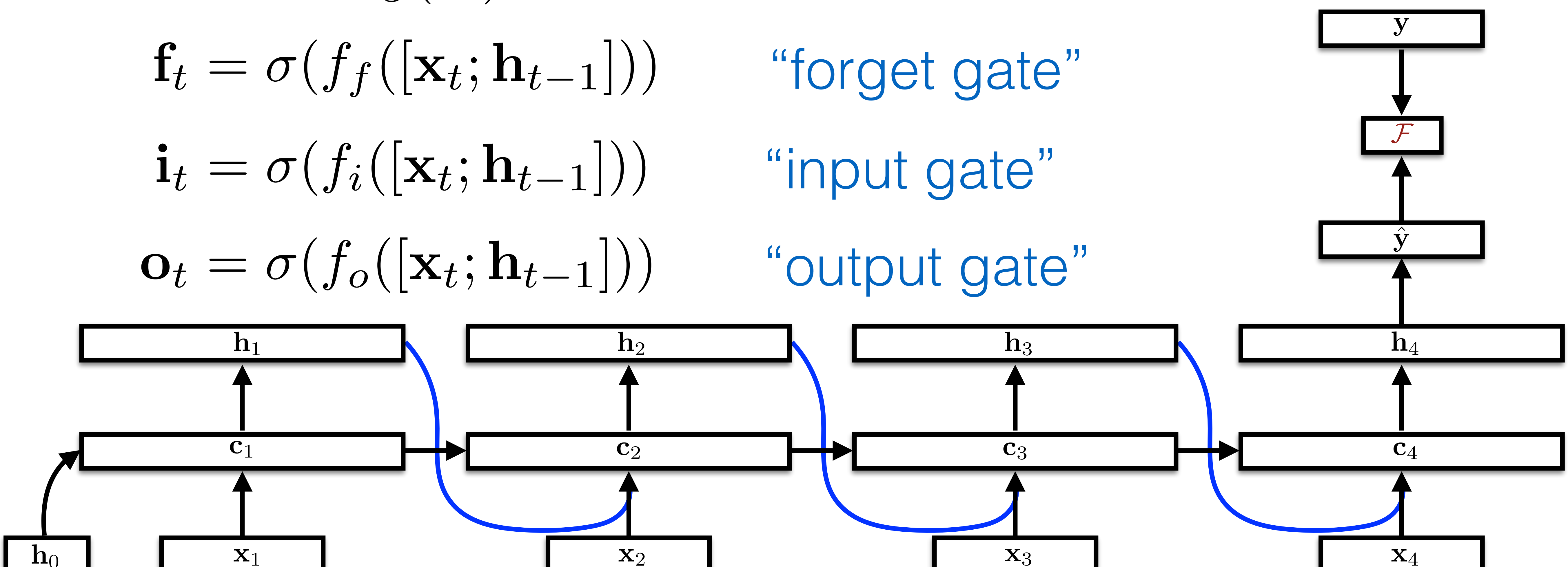
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

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

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

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

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



LSTM

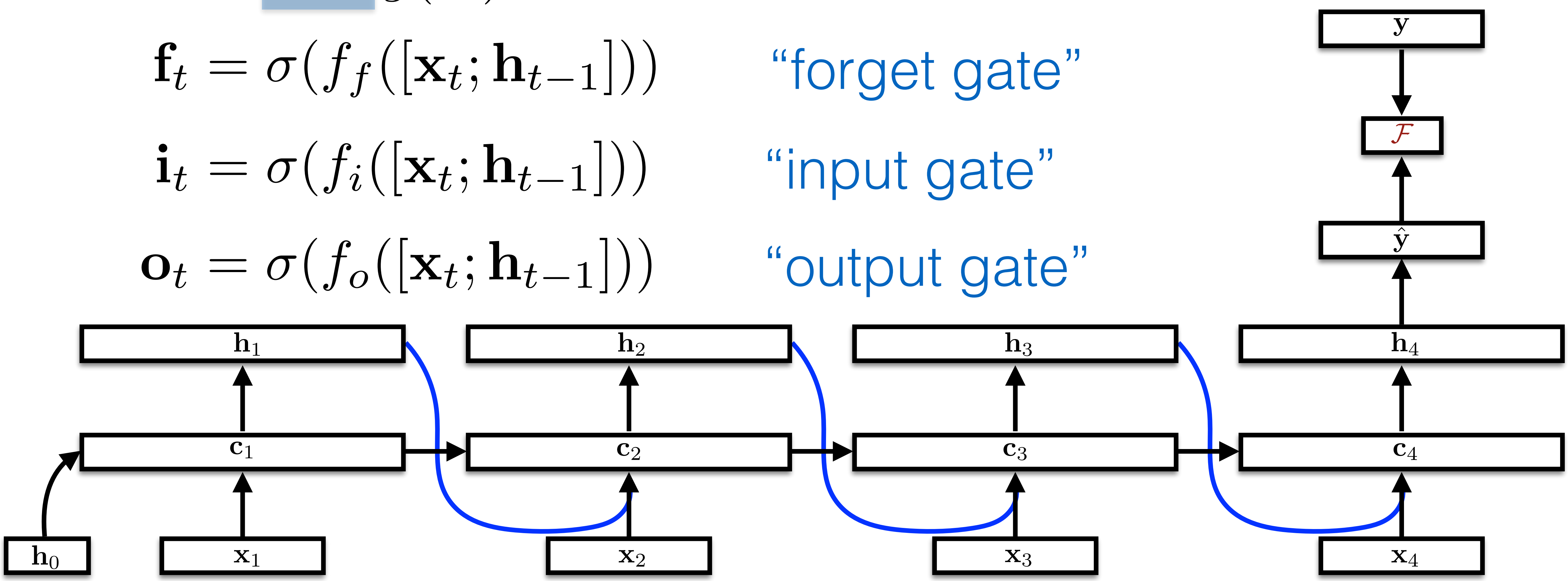
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

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

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

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

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



LSTM Variant

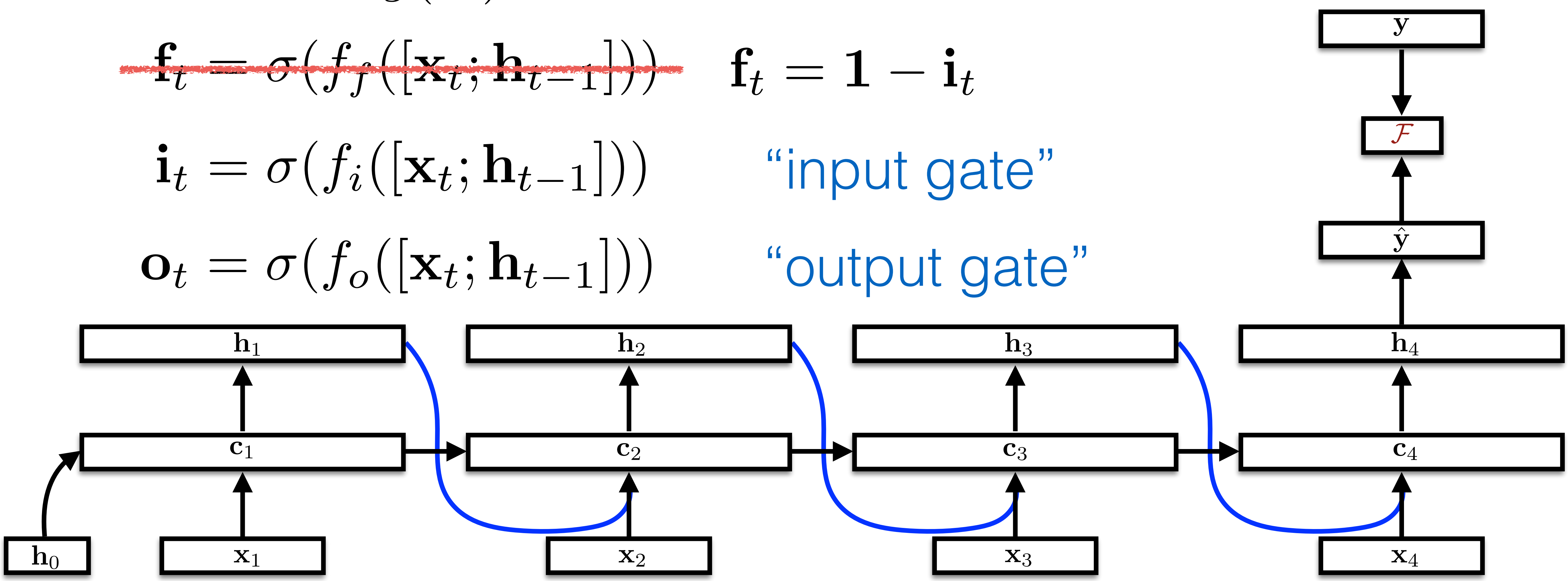
$$\mathbf{c}_t = (\mathbf{1} - \mathbf{i}_t) \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

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

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

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

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



LSTM Variant

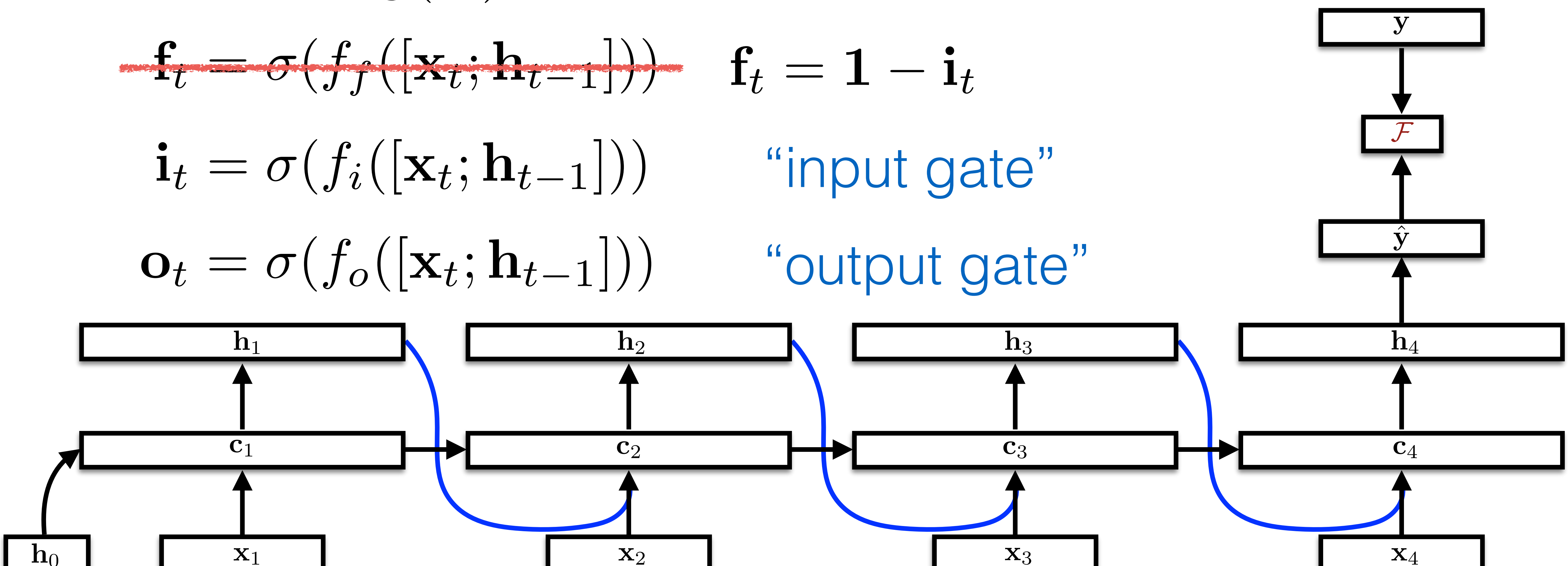
$$\mathbf{c}_t = (\mathbf{1} - \mathbf{i}_t) \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

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

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

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

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



Another Visualization

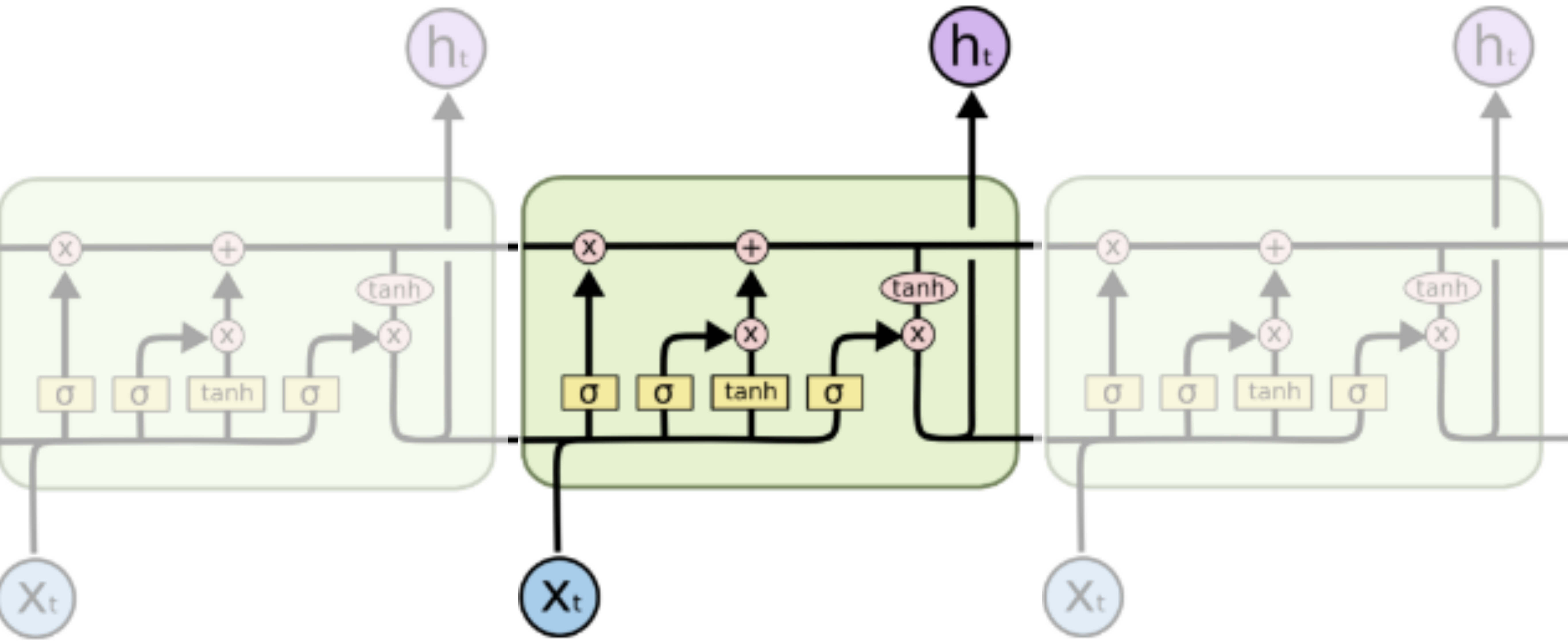


Figure credit: Christopher Olah

Another Visualization

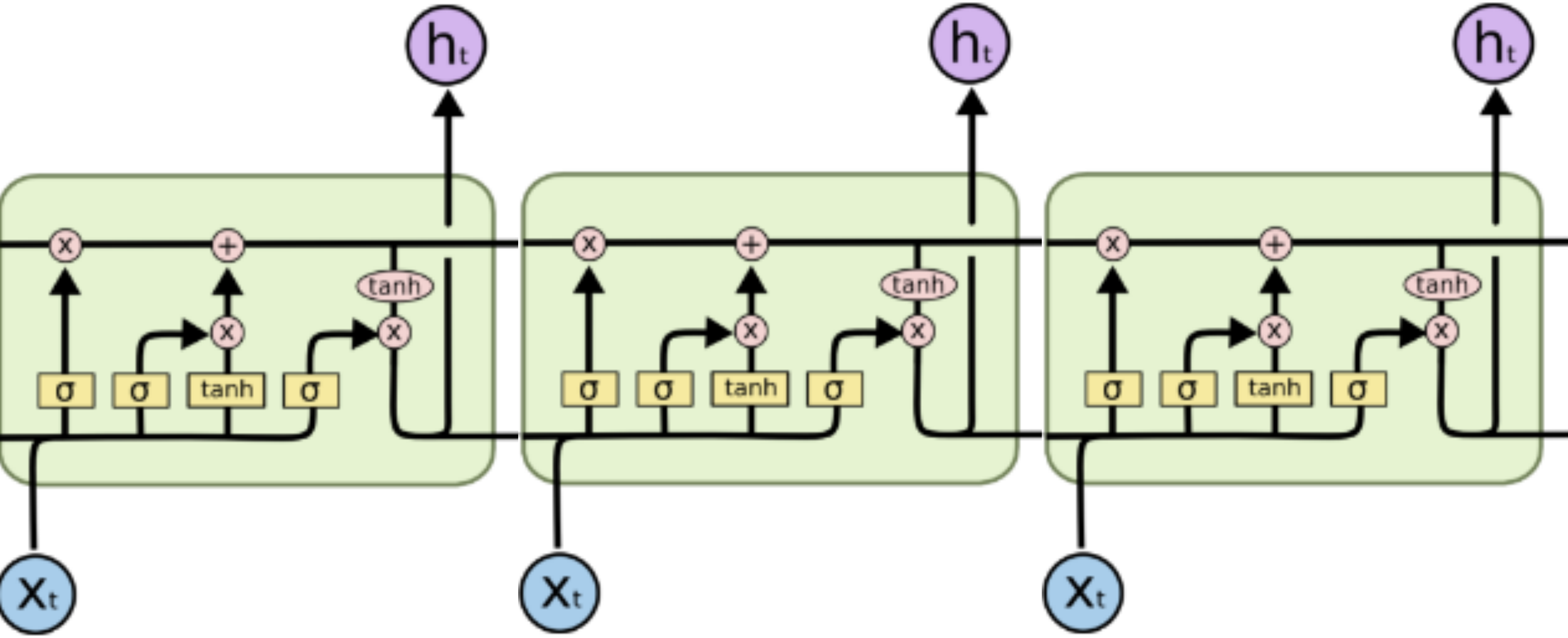


Figure credit: Christopher Olah

Another Visualization

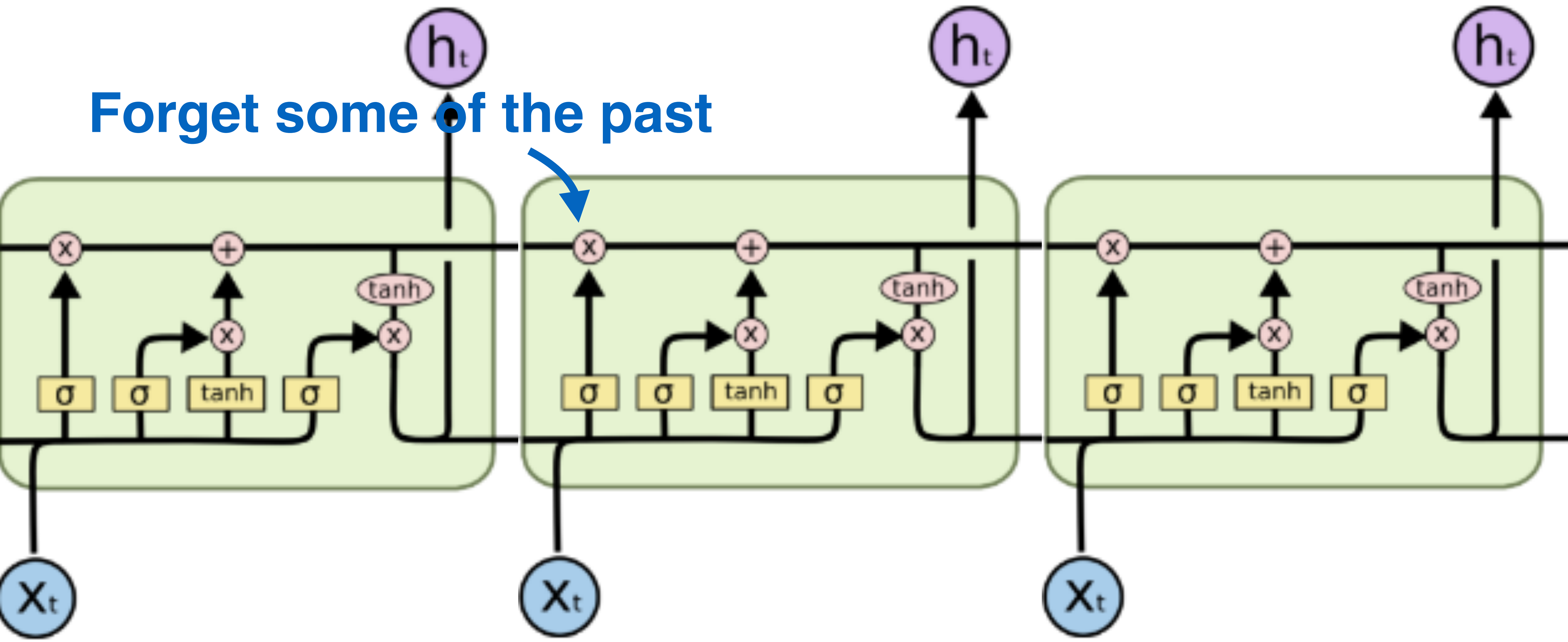


Figure credit: Christopher Olah

Another Visualization

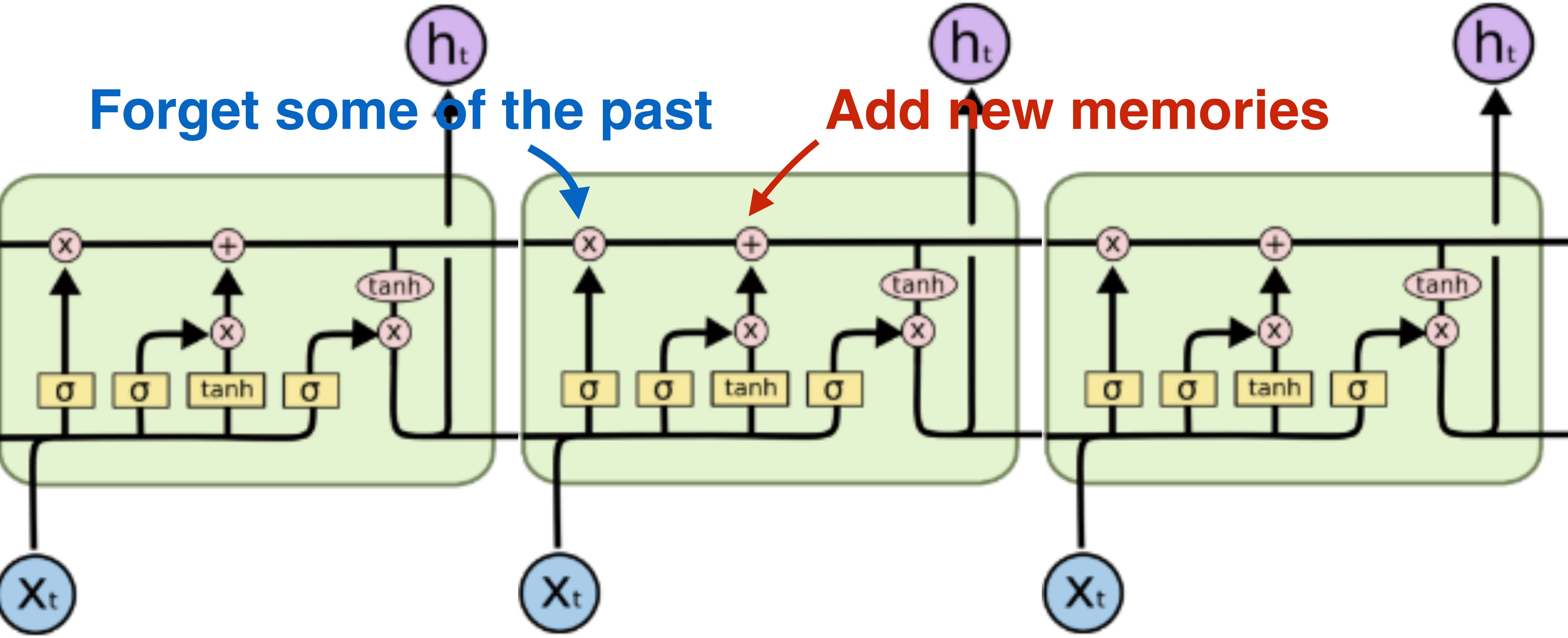


Figure credit: Christopher Olah

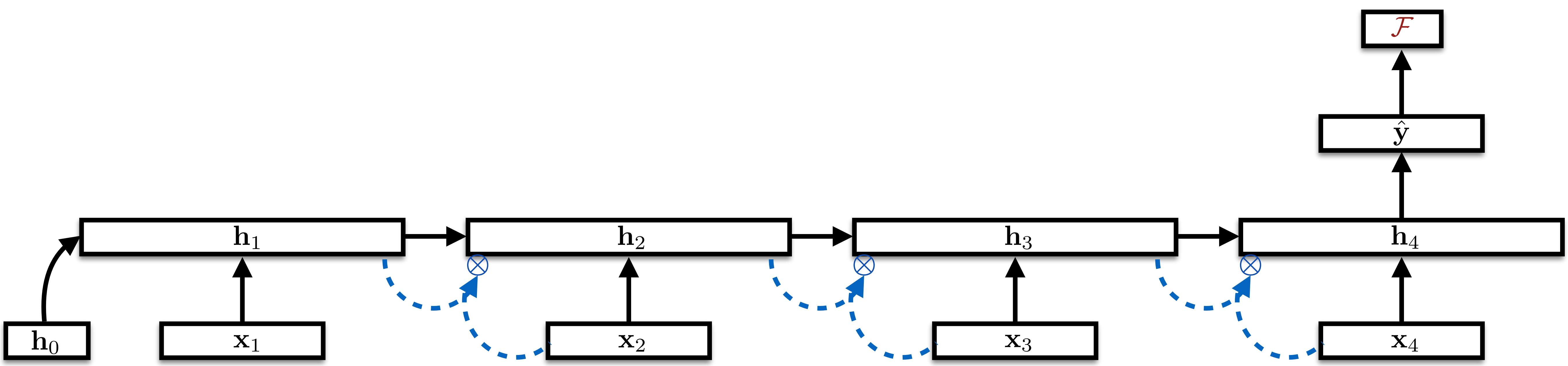
Gated Recurrent Units (GRUs)

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

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

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

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



Summary

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

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

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

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

Summary

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

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

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

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

Questions?

Break?

Conditional LMs

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

As with unconditional models, it helpful to use the chain rule to decompose the 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

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

Relatively large amounts of data for:
Translation, summarization, caption generation, speech recognition

Evaluating Conditional LMs

How good is our conditional language model?

These are language models, we can use **cross-entropy** or **perplexity**. okay to implement, hard to interpret

Task specific evaluation. Compare the model's most likely output to a human-generated reference output using a task-specific evaluation metric L.

$$\boldsymbol{w}^* = \arg\max_{\boldsymbol{w}} p(\boldsymbol{w} \mid \boldsymbol{x}) \qquad L(\boldsymbol{w}^*, \boldsymbol{w}_{ref})$$

Examples of L: BLEU, METEOR, ROUGE, WER

easy to implement, okay to interpret

Human evaluation.

hard to implement, easy to interpret

Evaluating Conditional LMs

How good is our conditional language model?

These are language models, we can use **cross-entropy** or **perplexity**.

okay to implement, hard to interpret

Task specific evaluation. Compare the model's most likely output to a human-generated reference output using a task-specific evaluation metric L.

$$\boldsymbol{w}^* = \arg\max_{\boldsymbol{w}} p(\boldsymbol{w} \mid \boldsymbol{x}) \qquad L(\boldsymbol{w}^*, \boldsymbol{w}_{ref})$$

Examples of L: BLEU, METEOR, ROUGE, WER

easy to implement, okay to interpret

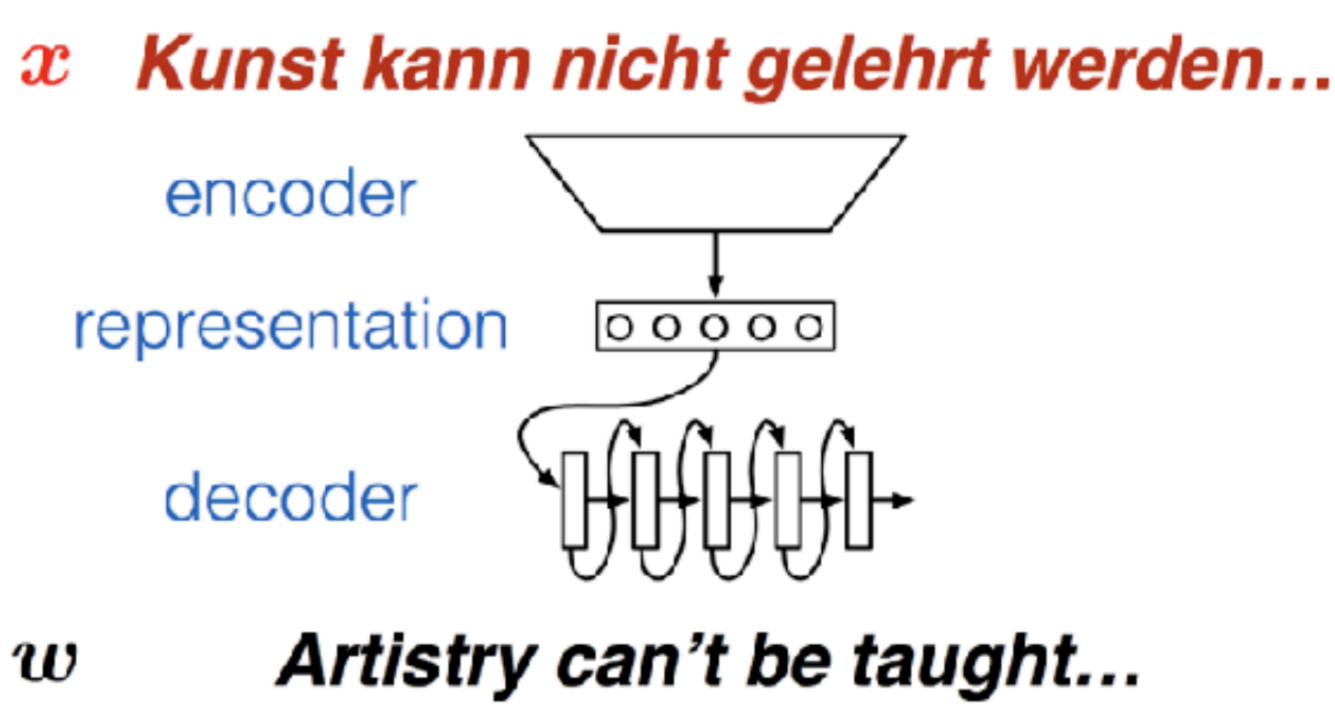
Human evaluation.

hard to implement, easy to interpret

Encoder-Decoder Models

Encoder-decoder models are a very simple class of conditional LMs that are nevertheless extremely powerful.

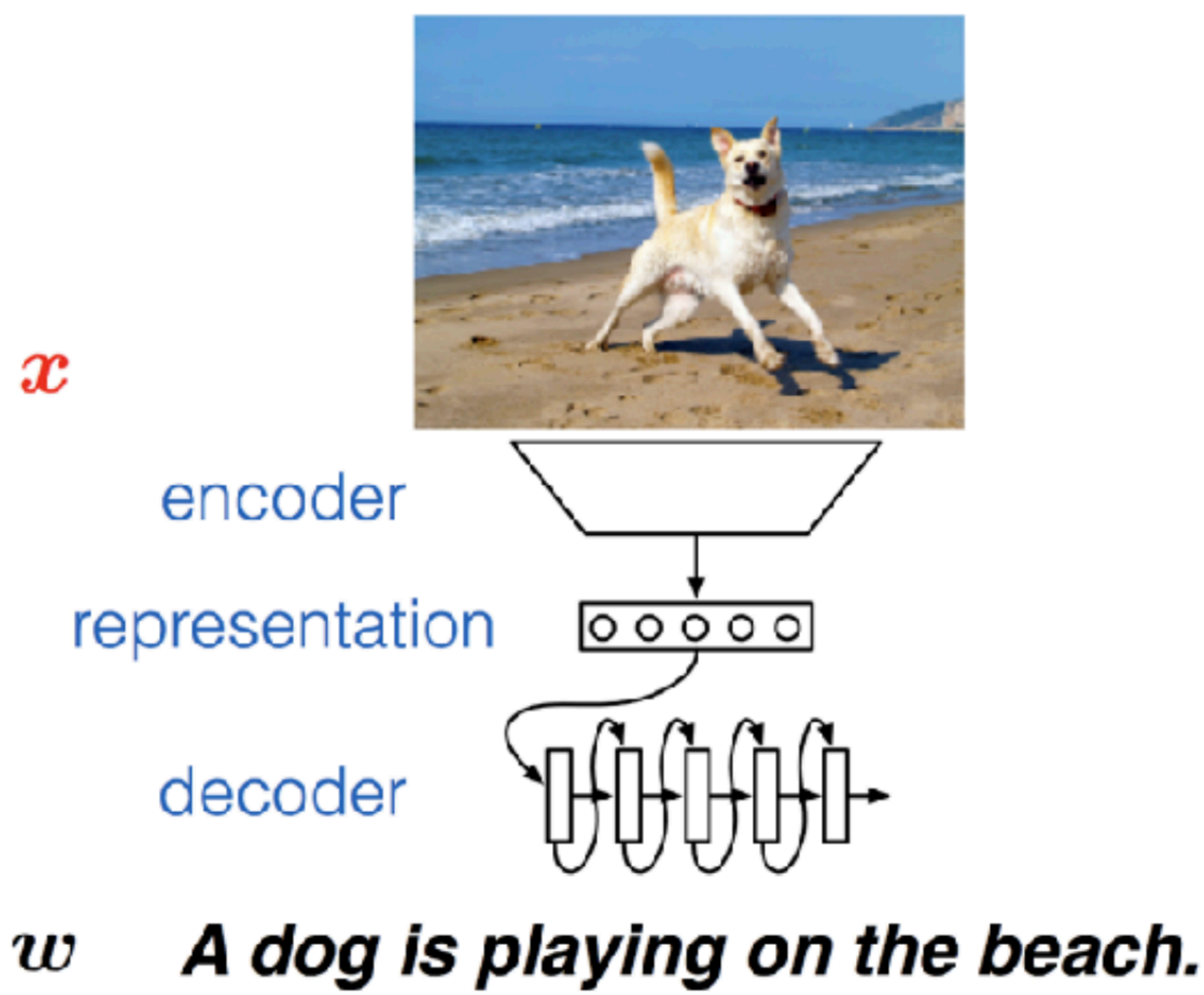
These “encode” $\boldsymbol{x}$ into a fixed-sized vector and “decode” that into a sequence of words $\boldsymbol{w}$.



Encoder-Decoder Models

Encoder-decoder models are a very simple class of conditional LMs that are nevertheless extremely powerful.

These "encode" $\boldsymbol{x}$ into a fixed-sized vector and "decode" that into a sequence of words $\boldsymbol{w}$.



Encoder-Decoder Models

Two questions

- How do we encode $\boldsymbol{x}$ into a fixed-sized vector?
  - Problem/modality specific
  - Think about assumptions!
- How do we decode that vector into a sequence of words $\boldsymbol{w}$?
  - Less problem specific (general decoders?)
  - We now describe a solution using RNNs.

Recurrent Neural Networks (RNNs)

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

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

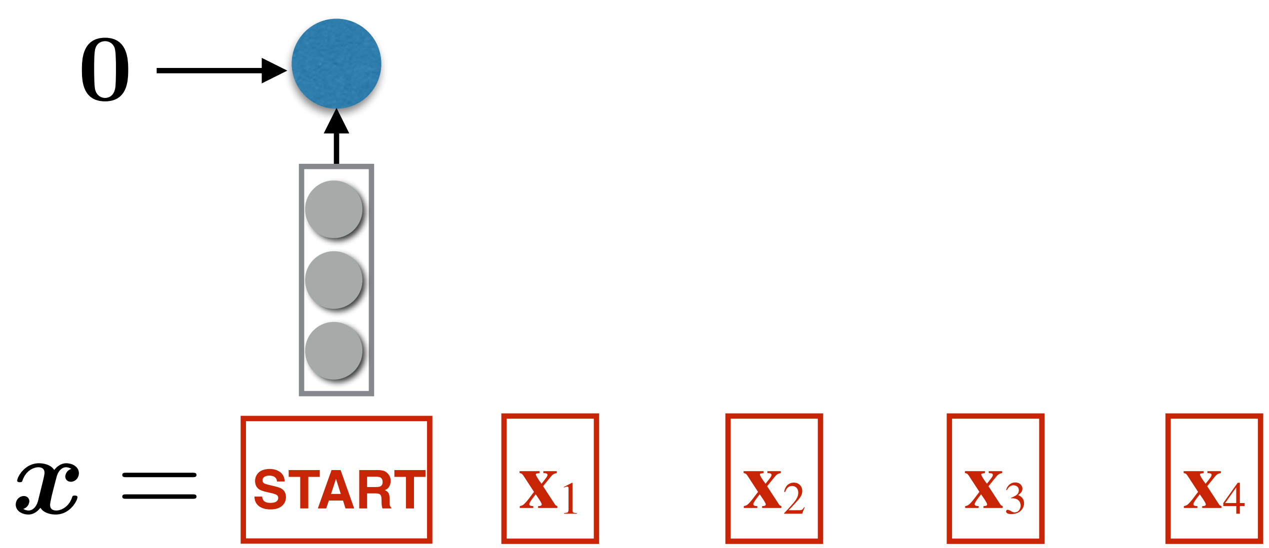
$$\boldsymbol{x} = \boxed{\text{START}} \quad \boxed{\mathbf{x}_1} \quad \boxed{\mathbf{x}_2} \quad \boxed{\mathbf{x}_3} \quad \boxed{\mathbf{x}_4}$$

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

Recurrent Neural Networks (RNNs)

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

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

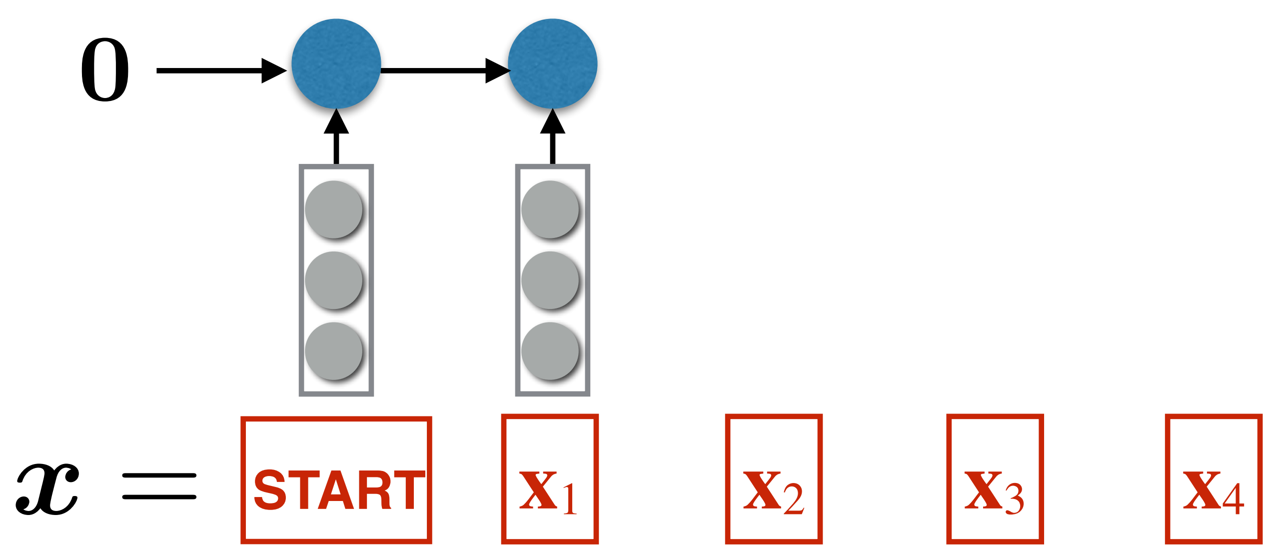


What is a vector representation of a sequence $\boldsymbol{x}$?

Recurrent Neural Networks (RNNs)

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

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

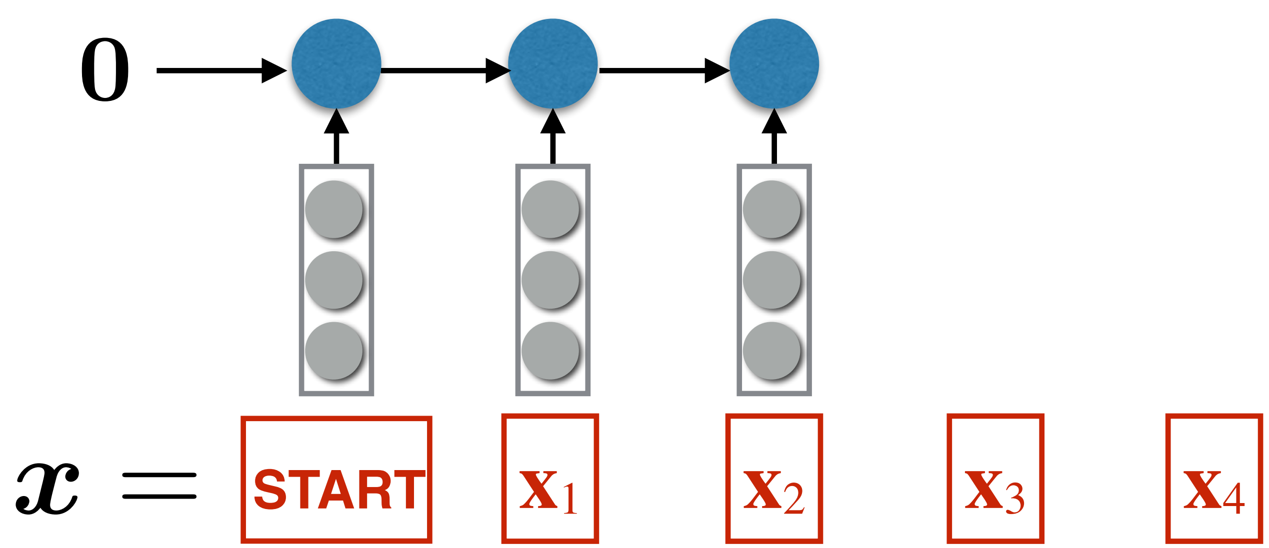


What is a vector representation of a sequence $\boldsymbol{x}$?

Recurrent Neural Networks (RNNs)

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

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

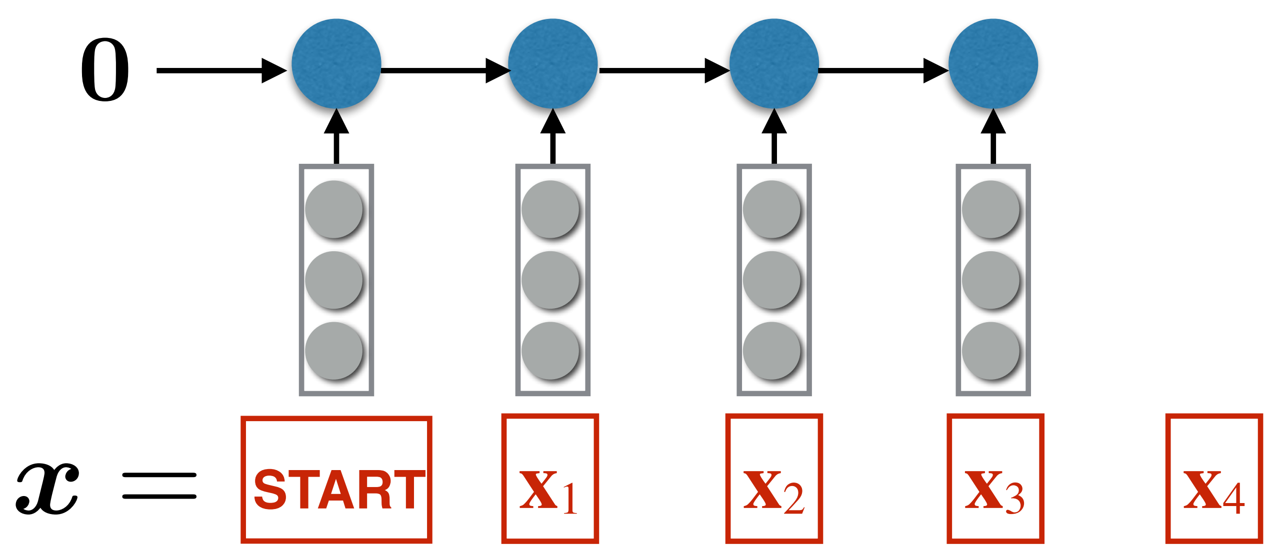


What is a vector representation of a sequence $\boldsymbol{x}$?

Recurrent Neural Networks (RNNs)

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

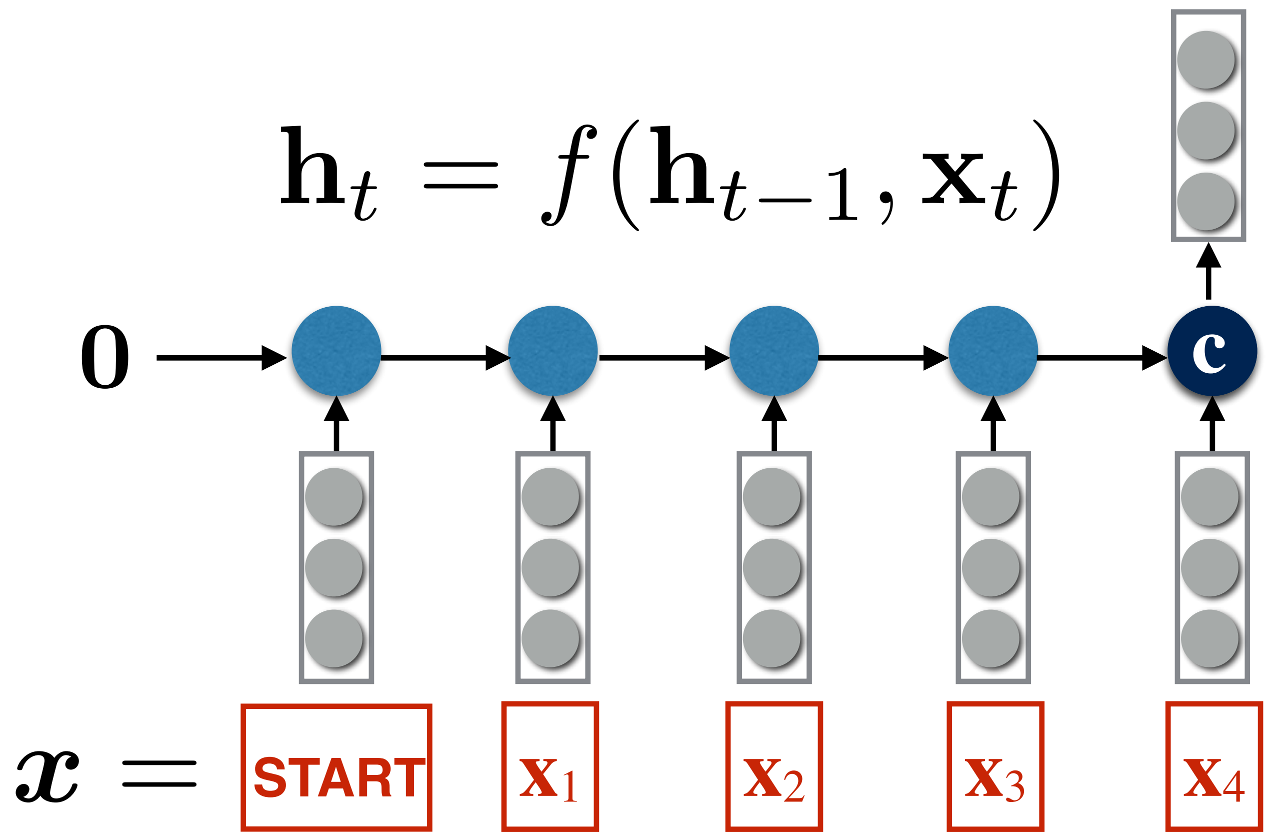
$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$



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

Recurrent Neural Networks (RNNs)

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



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

RNN Encoder-Decoders

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

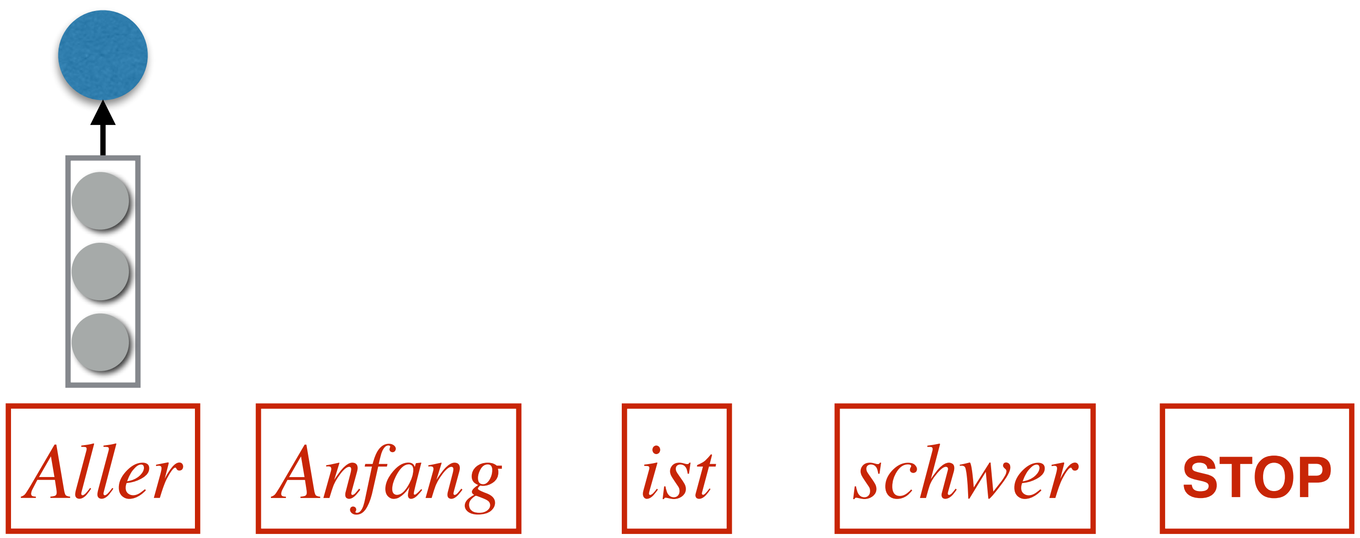
Aller *Anfang* *ist* *schwer* **STOP**

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

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

RNN Encoder-Decoders

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

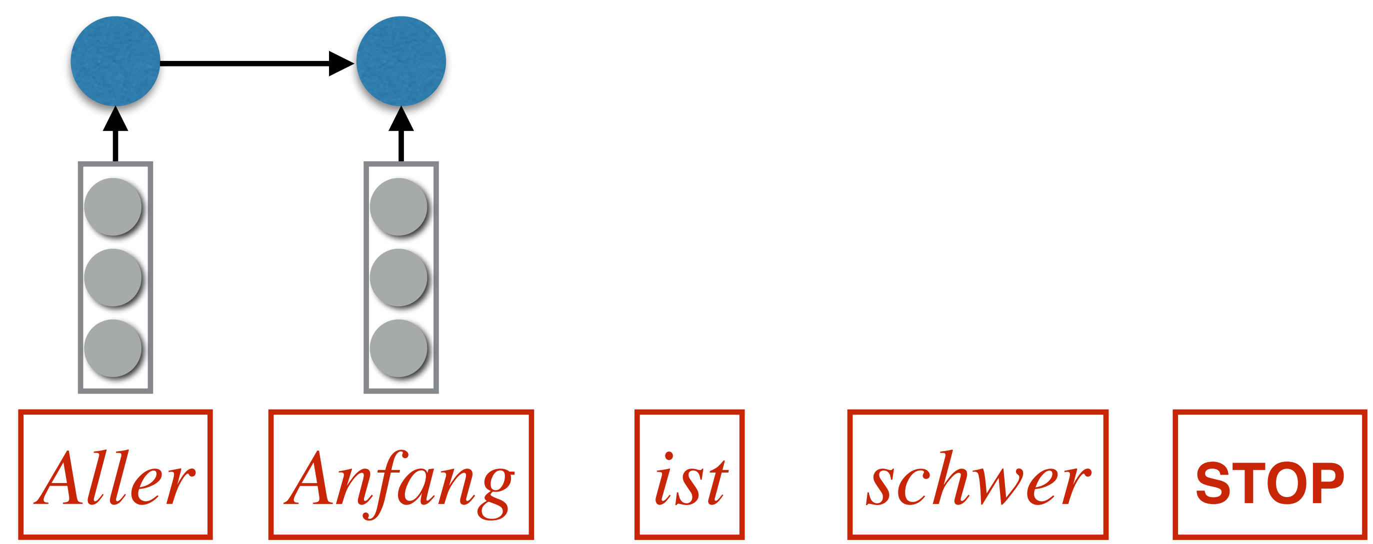


What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$**?**

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

RNN Encoder-Decoders

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

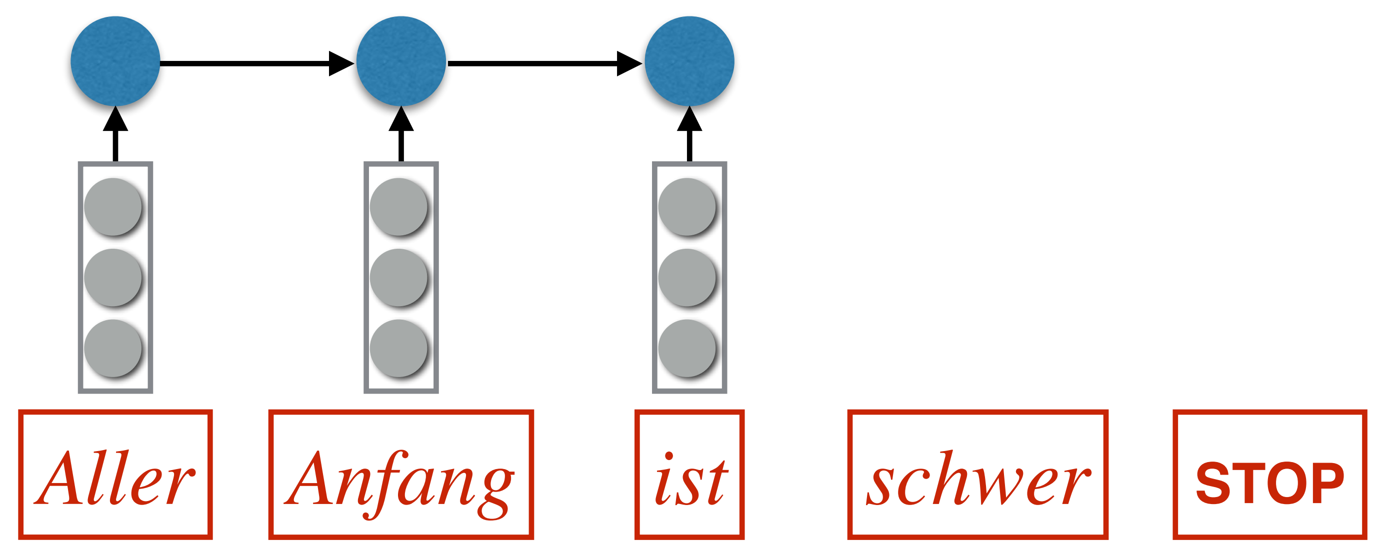


What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$?

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

RNN Encoder-Decoders

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

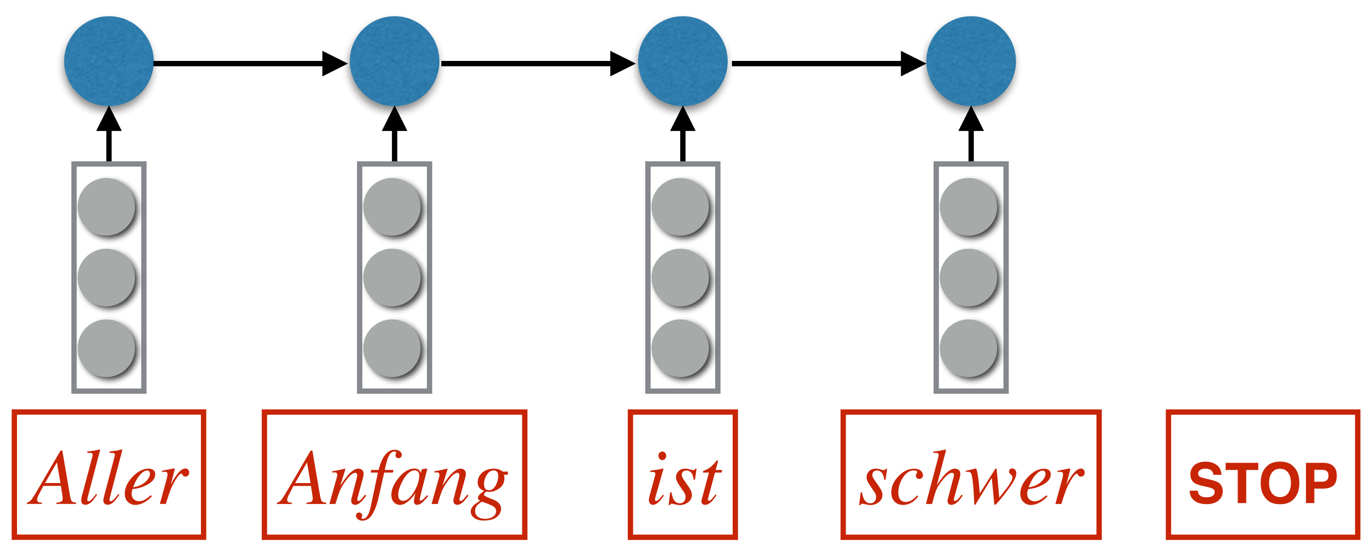


What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$?

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

RNN Encoder-Decoders

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

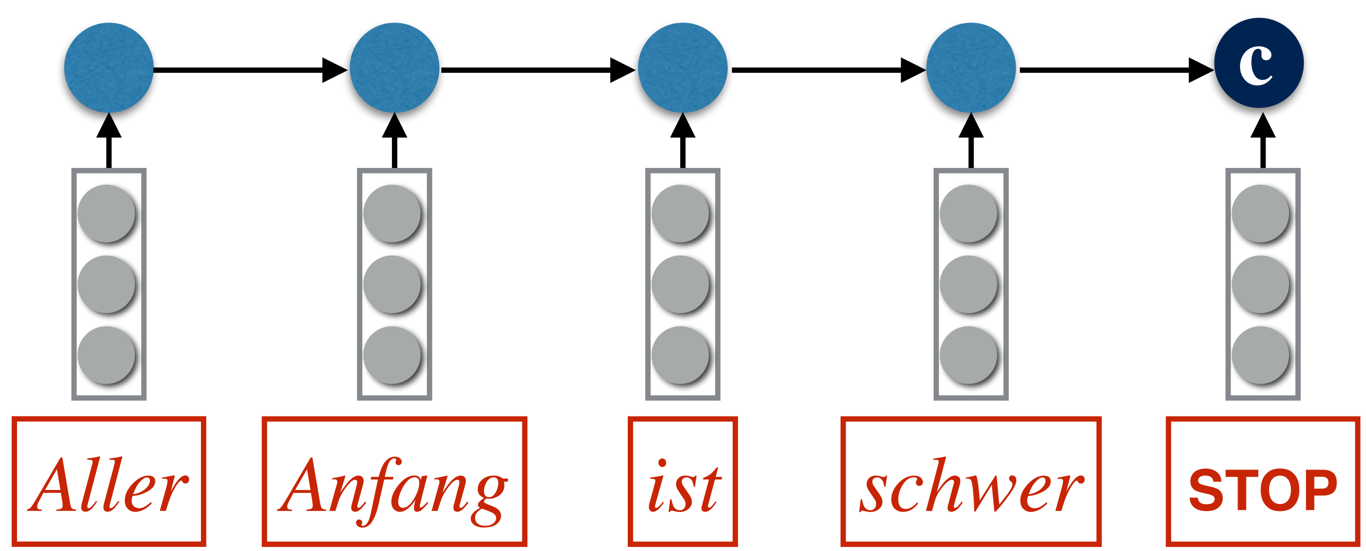


What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$**?**

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

RNN Encoder-Decoders

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

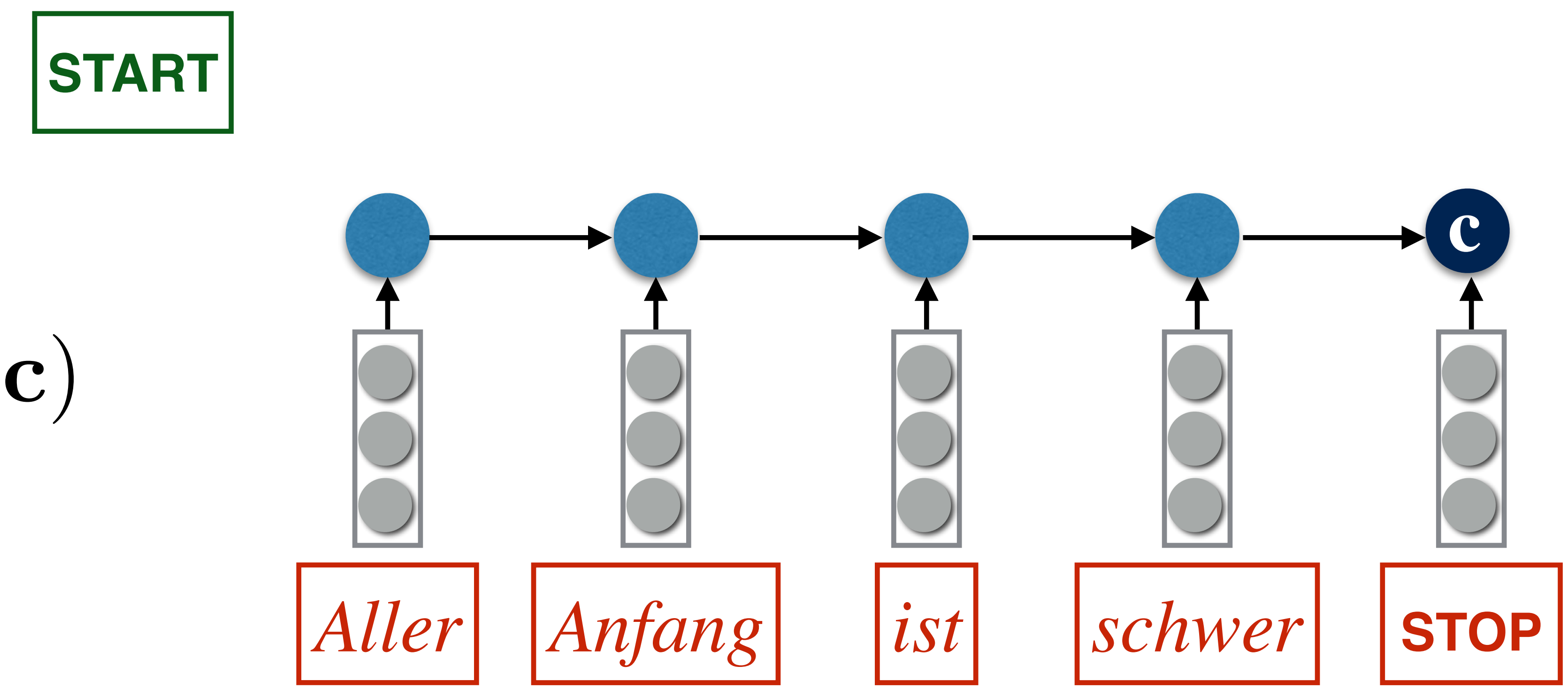


What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$?

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

RNN Encoder-Decoders

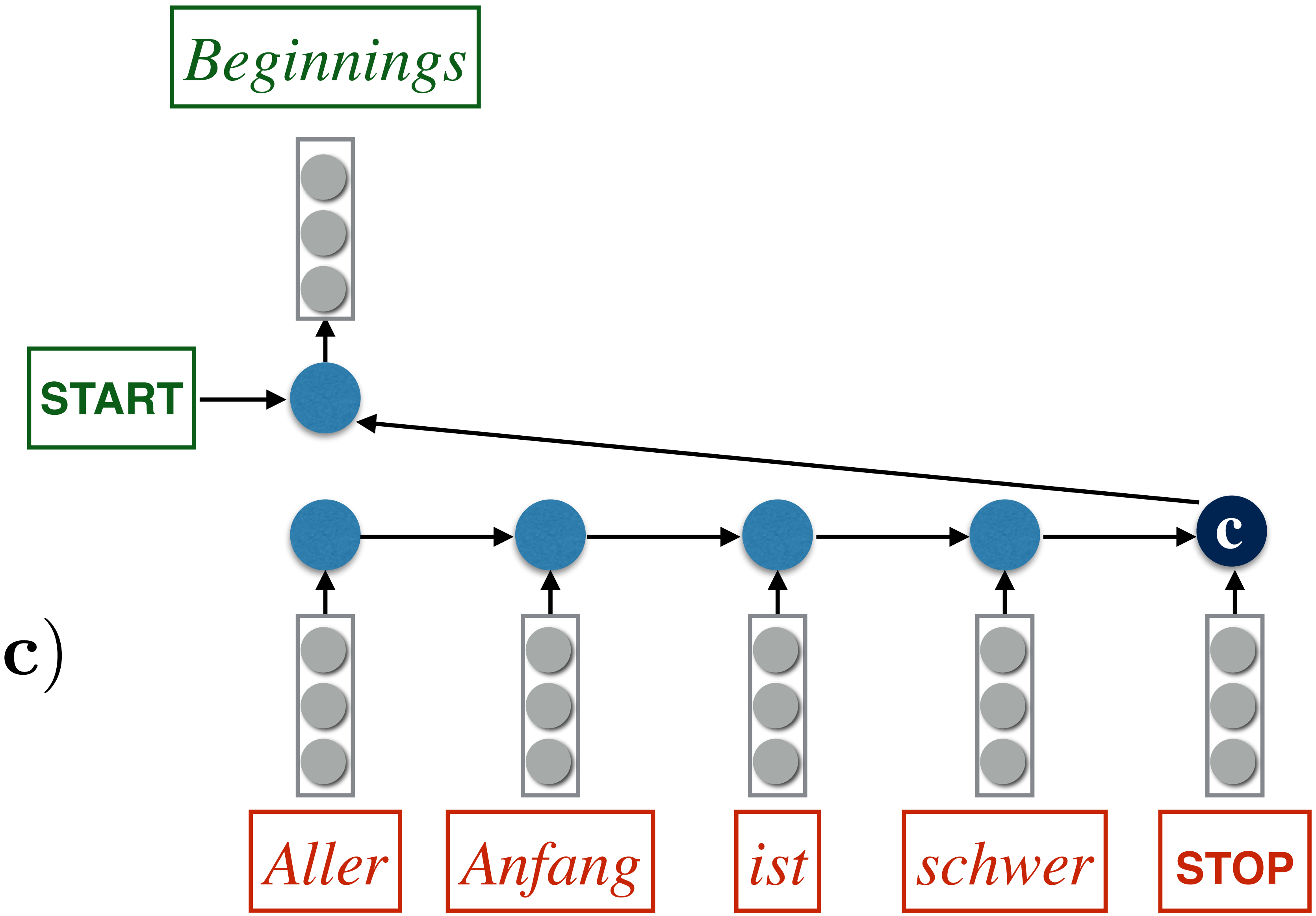
$$\mathbf{c} = \text{RNN}(\boldsymbol{x})$$
$$\boldsymbol{y} \mid \mathbf{c} \sim \text{RNNLM}(\mathbf{c})$$



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

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

RNN Encoder-Decoders



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

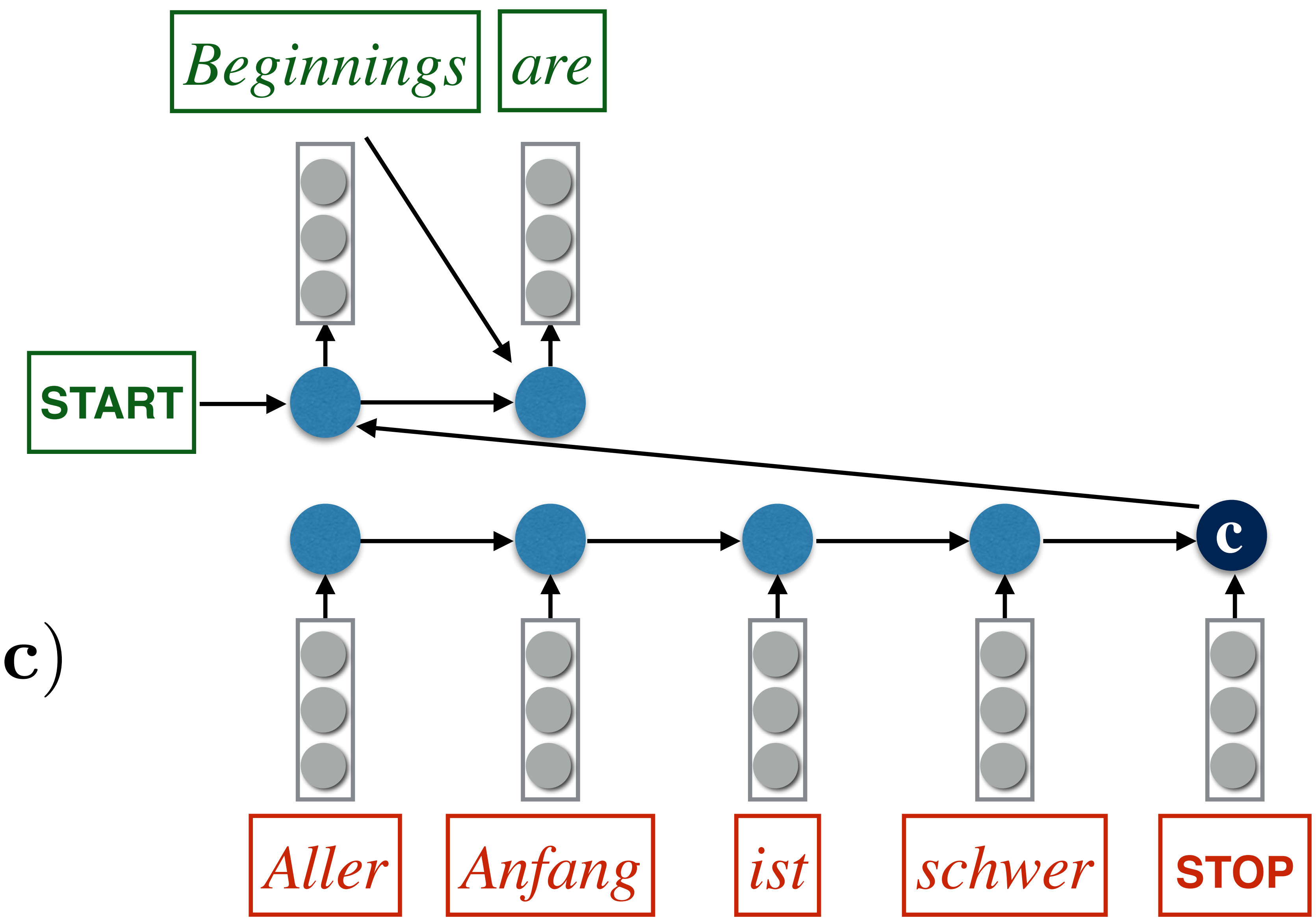
$$\boldsymbol{y} \mid \mathbf{c} \sim \text{RNNLM}(\mathbf{c})$$

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

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

RNN Encoder-Decoders

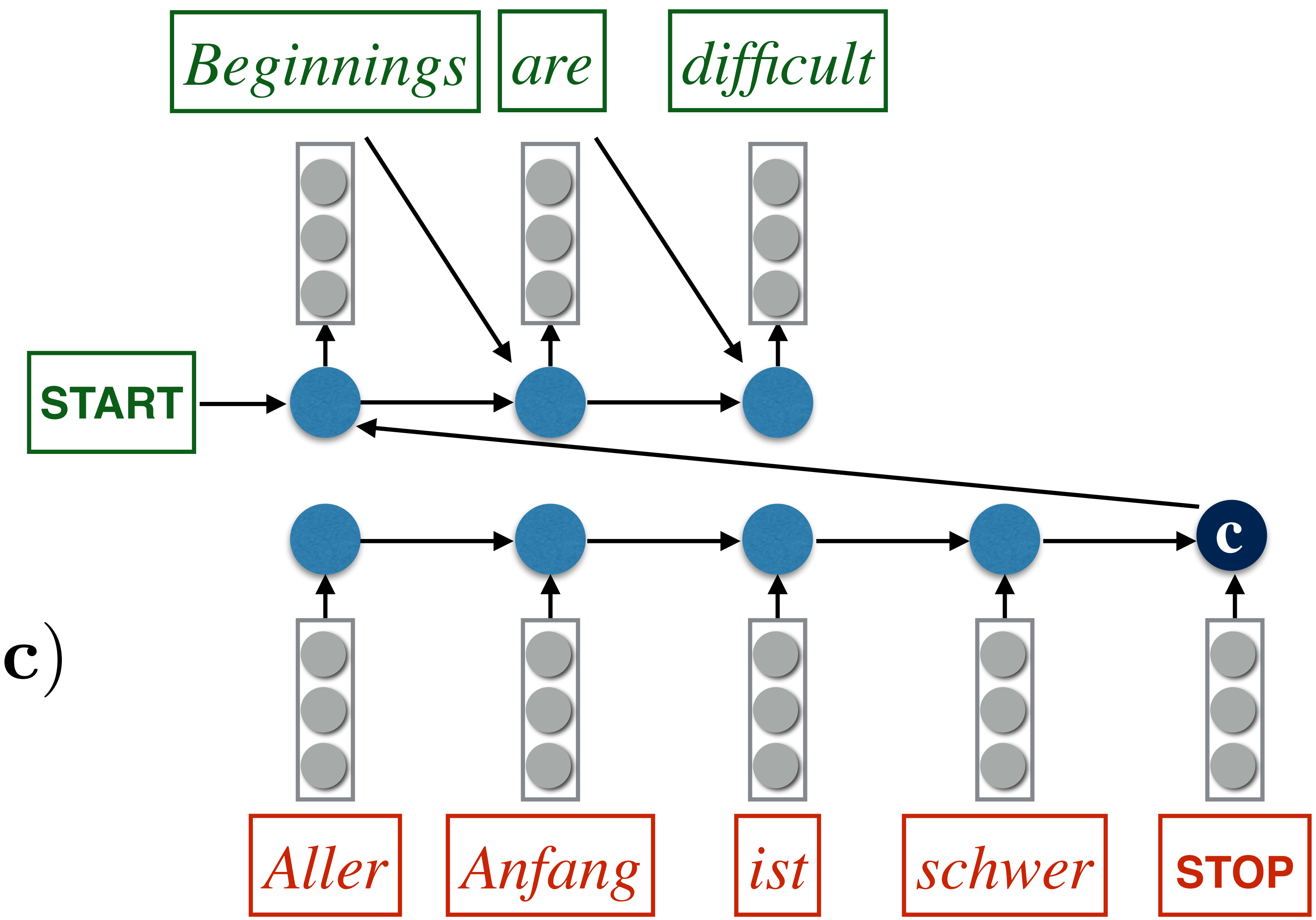
$$\mathbf{c} = \text{RNN}(\boldsymbol{x})$$
$$\boldsymbol{y} \mid \mathbf{c} \sim \text{RNNLM}(\mathbf{c})$$



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

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

RNN Encoder-Decoders



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

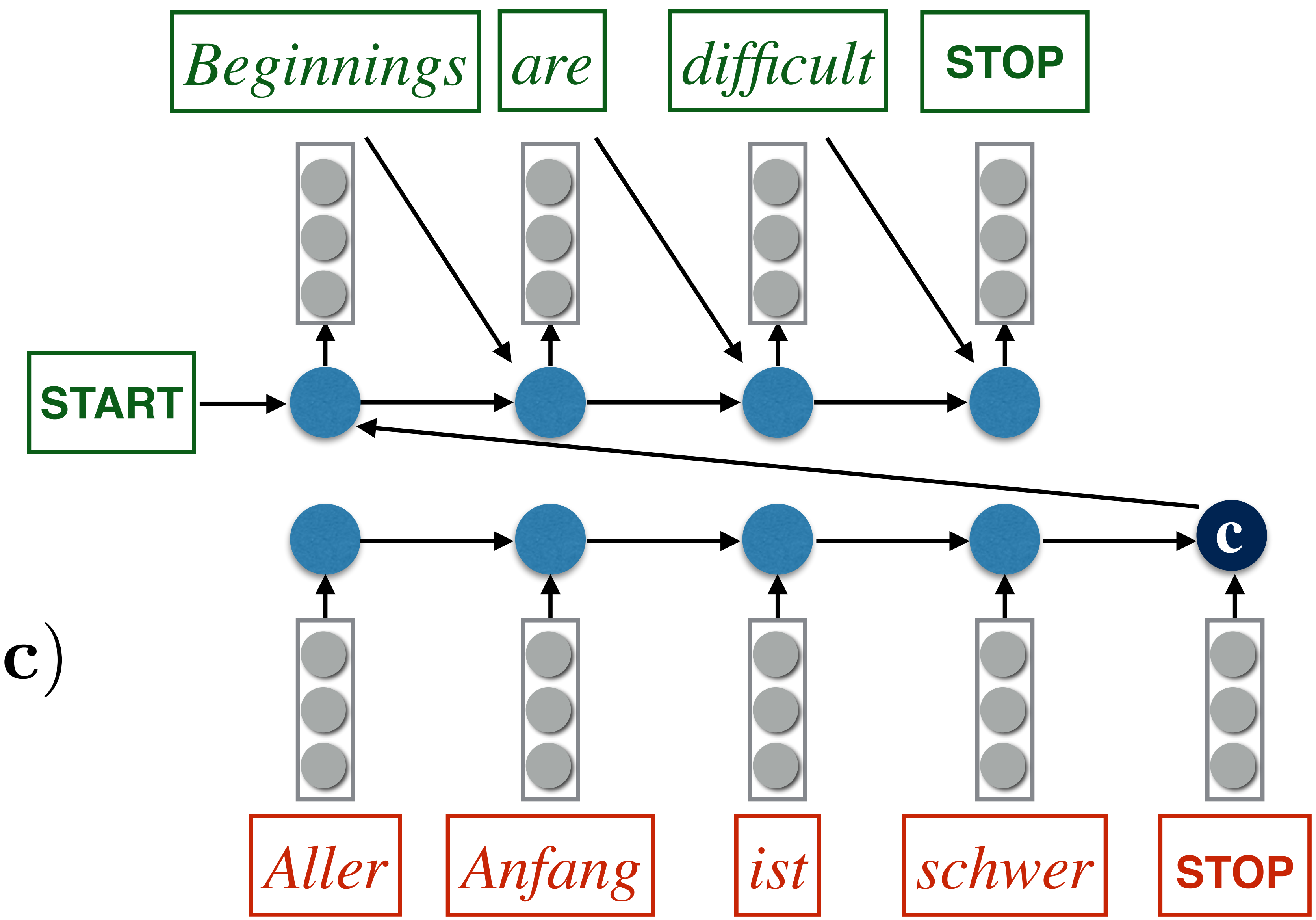
$$\boldsymbol{y} \mid \mathbf{c} \sim \mathrm{RNNLM}(\mathbf{c})$$

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

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

RNN Encoder-Decoders

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



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

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

Conditional LMs

Algorithms for Decoding

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

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

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

Unlike with Markov models, this is a hard problem. But we can approximate it with a **greedy search**:

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

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

$$\vdots$$

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

Beam search for decoding

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

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

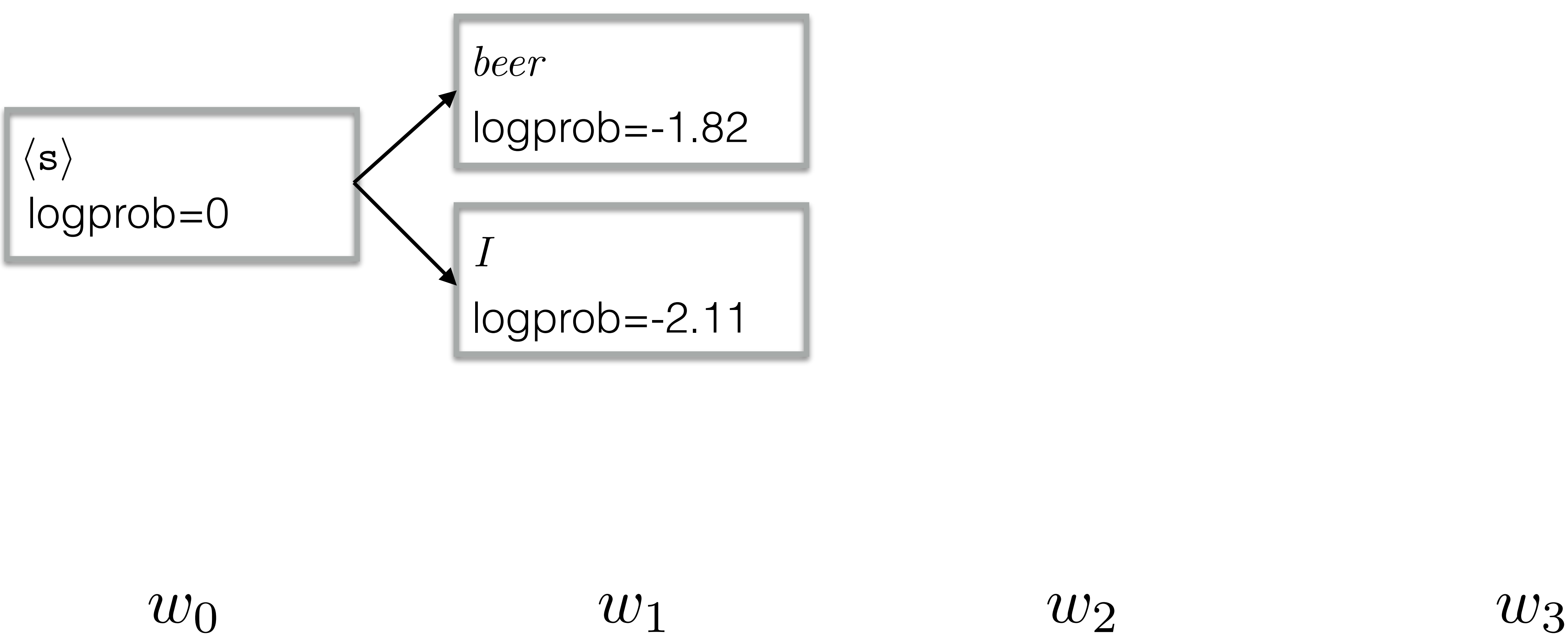
Beam search for decoding

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

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

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

beer **drink** **I**



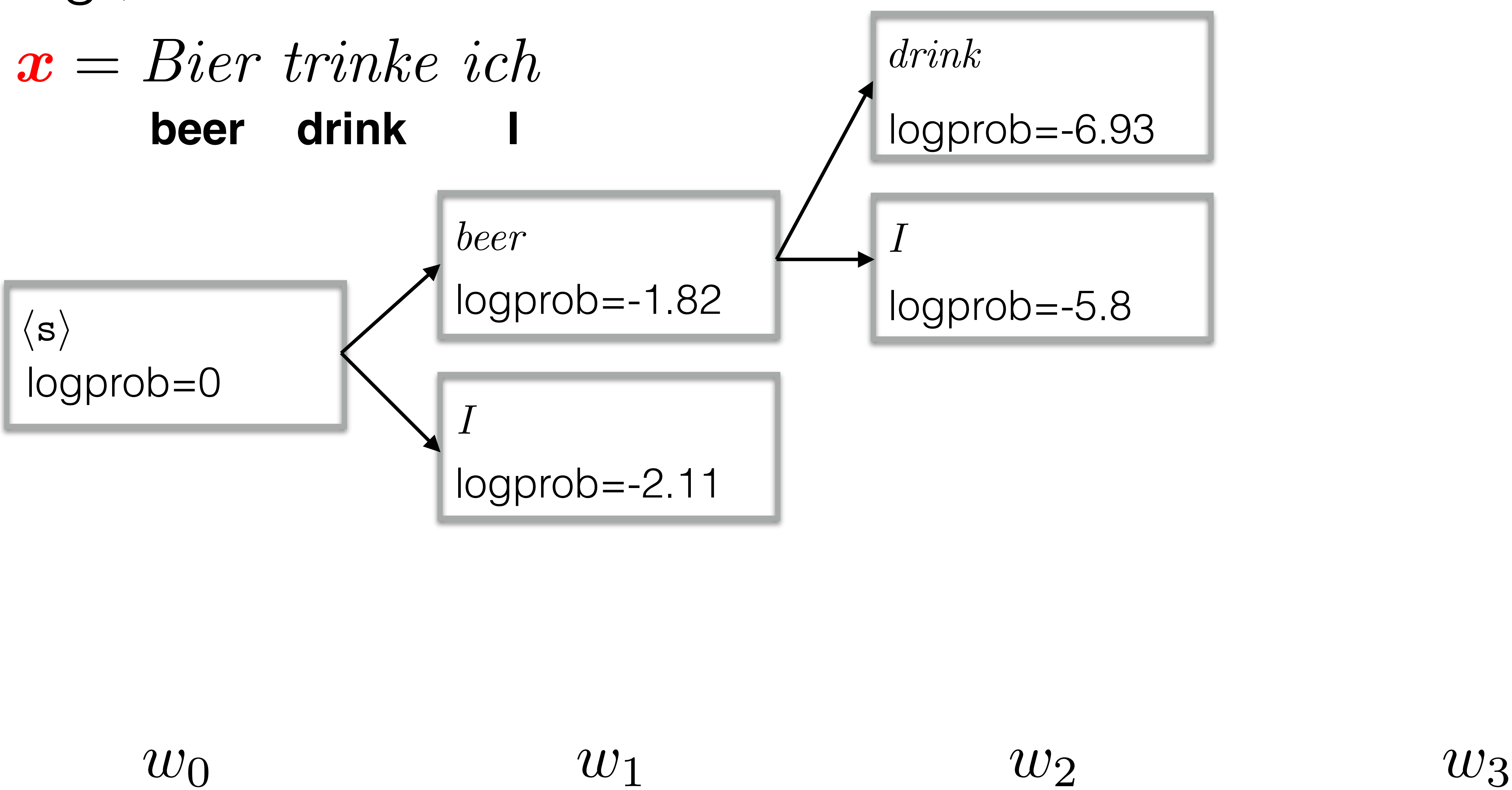
Beam search for decoding

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

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

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

beer **drink** **I**



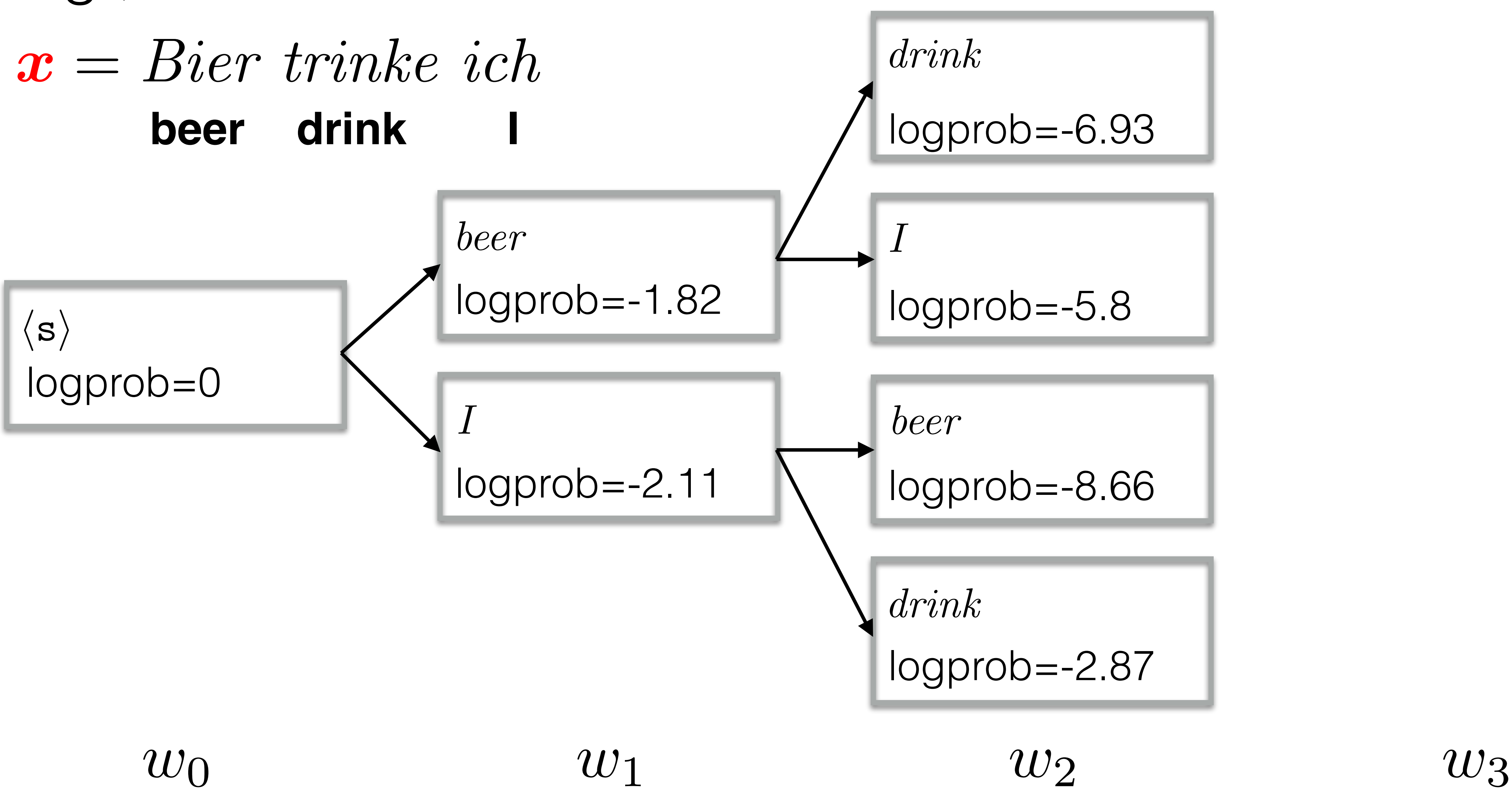
Beam search for decoding

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

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

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

beer **drink** **I**



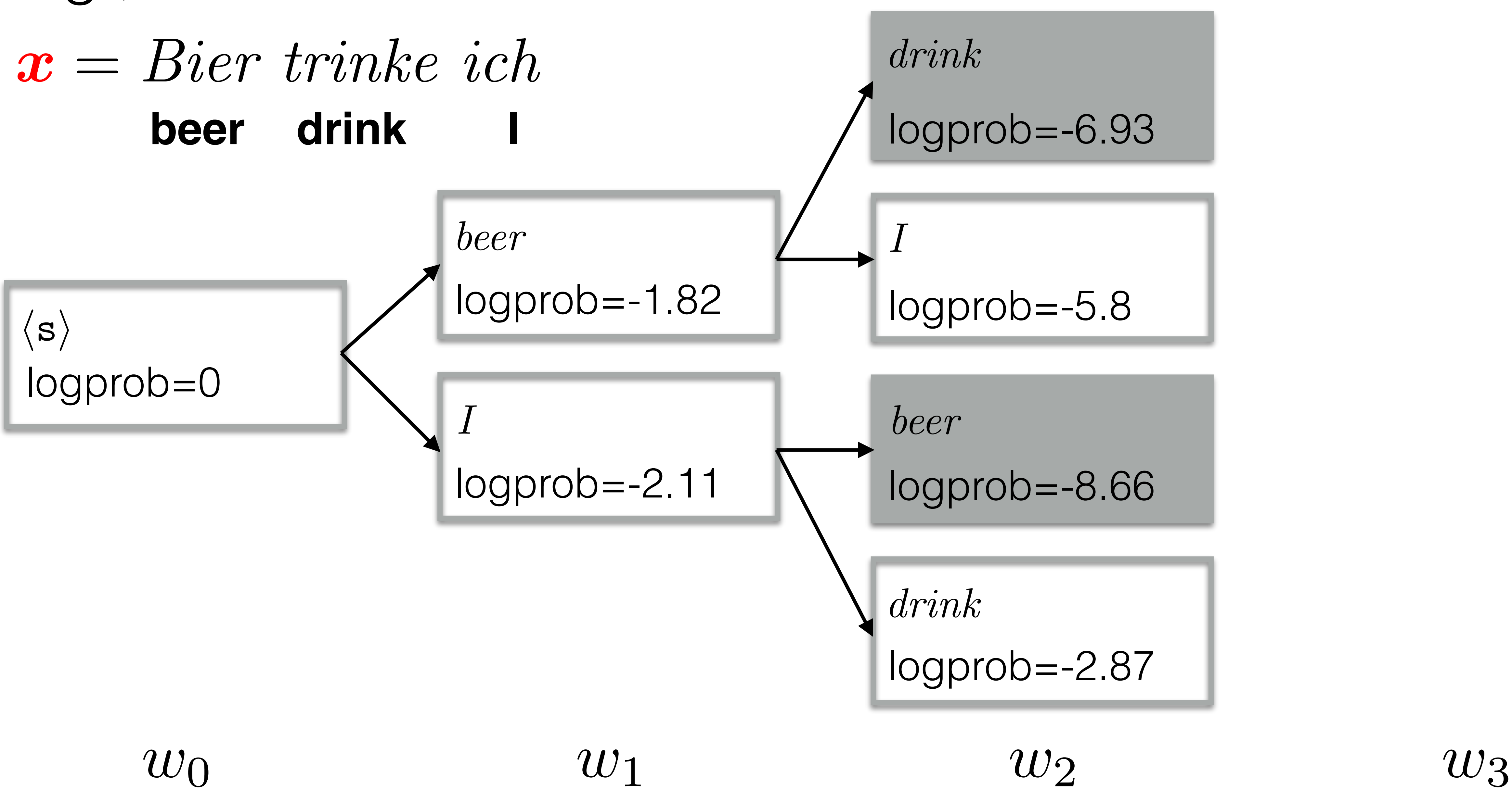
Beam search for decoding

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

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

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

beer drink I



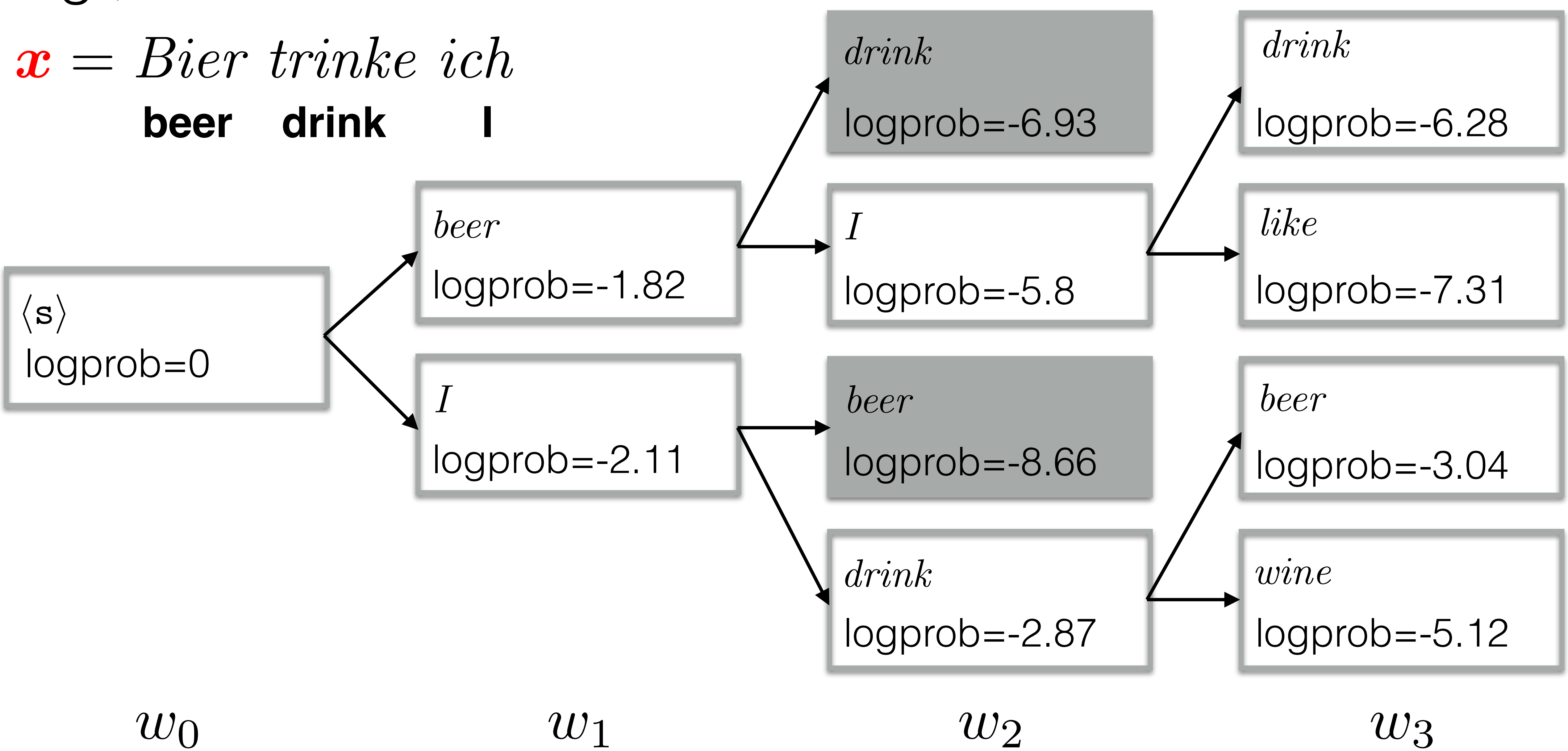
Beam search for decoding

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

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

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

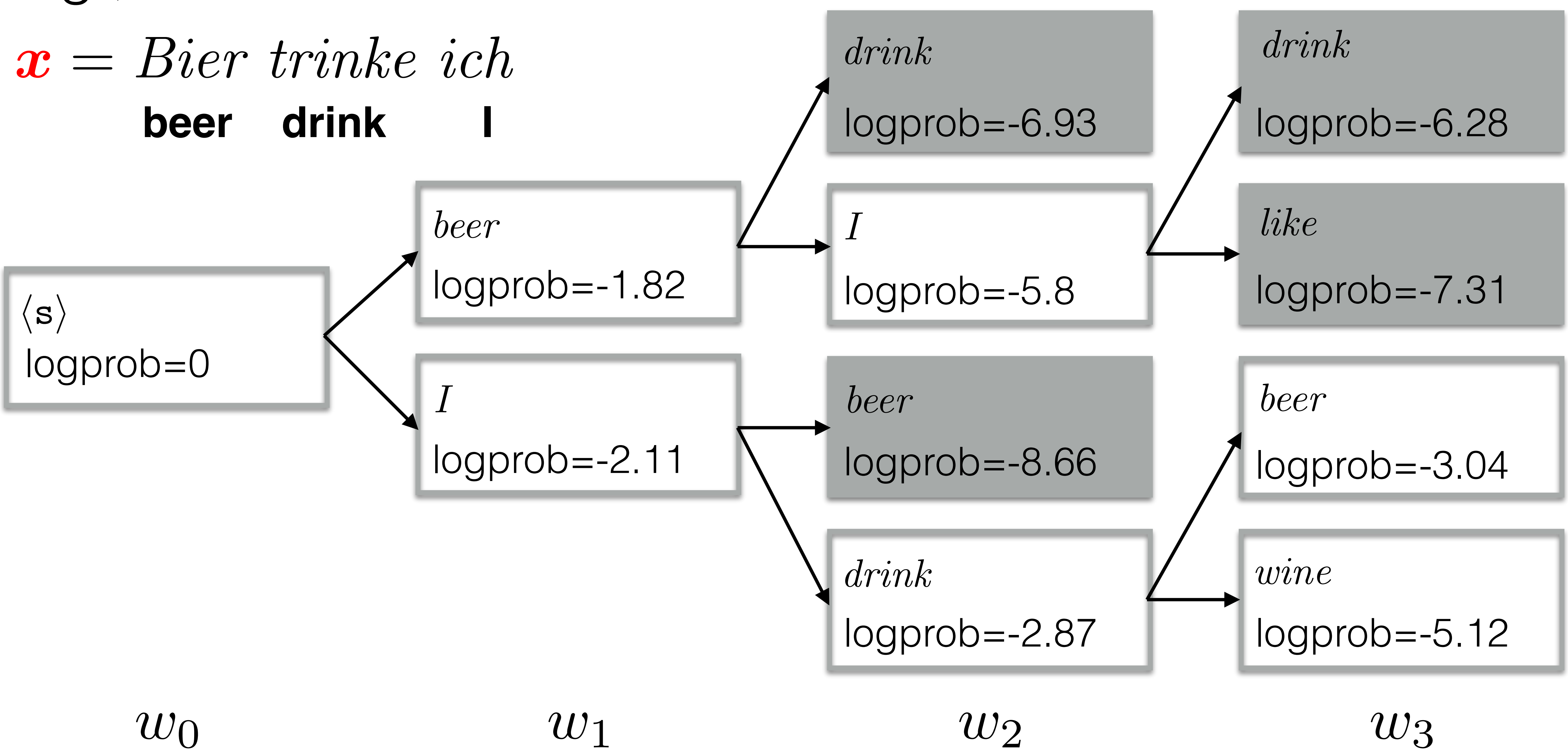
beer drink I



Beam search for decoding

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

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



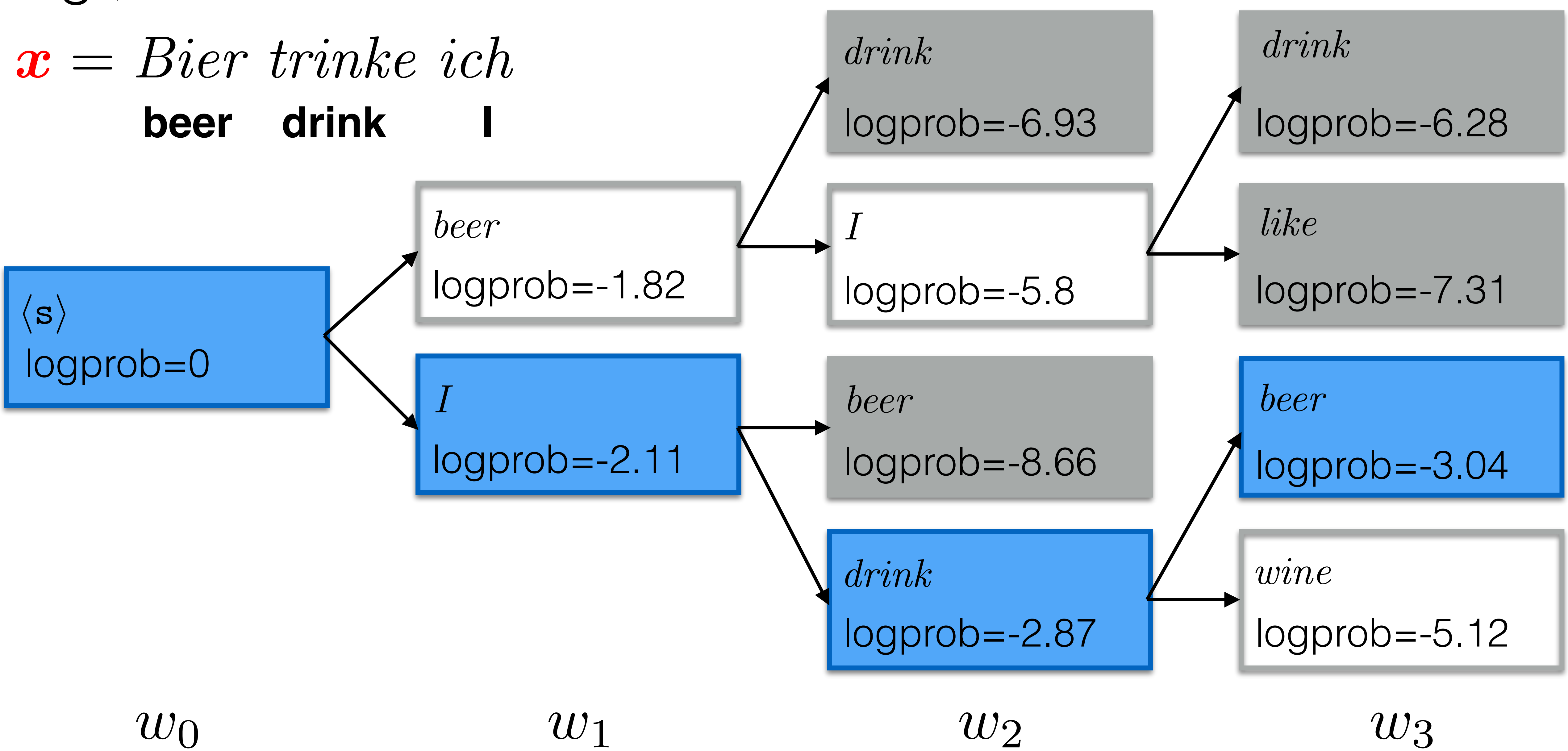
Beam search for decoding

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

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

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

beer drink I



Questions?

Conditioning with vectors

Encoder-decoder models like this compress a lot of information in a vector.

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

What is to be done?

Translation with Attention

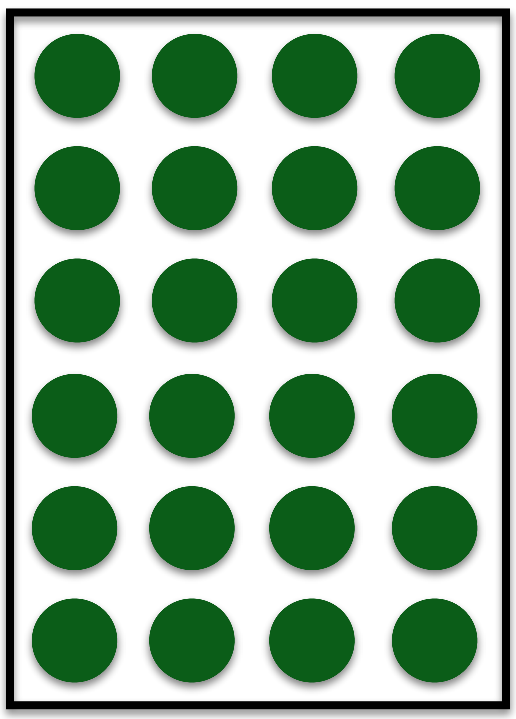
- Represent a source sentence as a matrix
- Generate a target sentence from a matrix

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

Sentences as Matrices

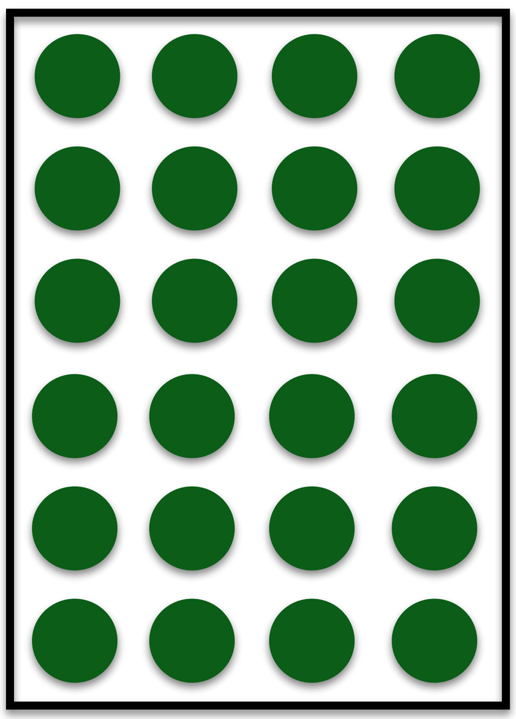
- Problem with the fixed-size vector model in translation (maybe in images?)
  - Sentences are of different sizes but vectors are of the same size
- Solution: use matrices instead
  - Fixed number of rows, but number of columns depends on the number of words
  - Usually $|\boldsymbol{f}|$ = #cols

Sentences as Matrices

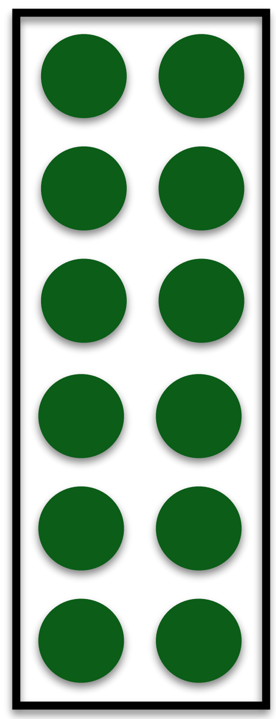


Ich möchte ein Bier

Sentences as Matrices

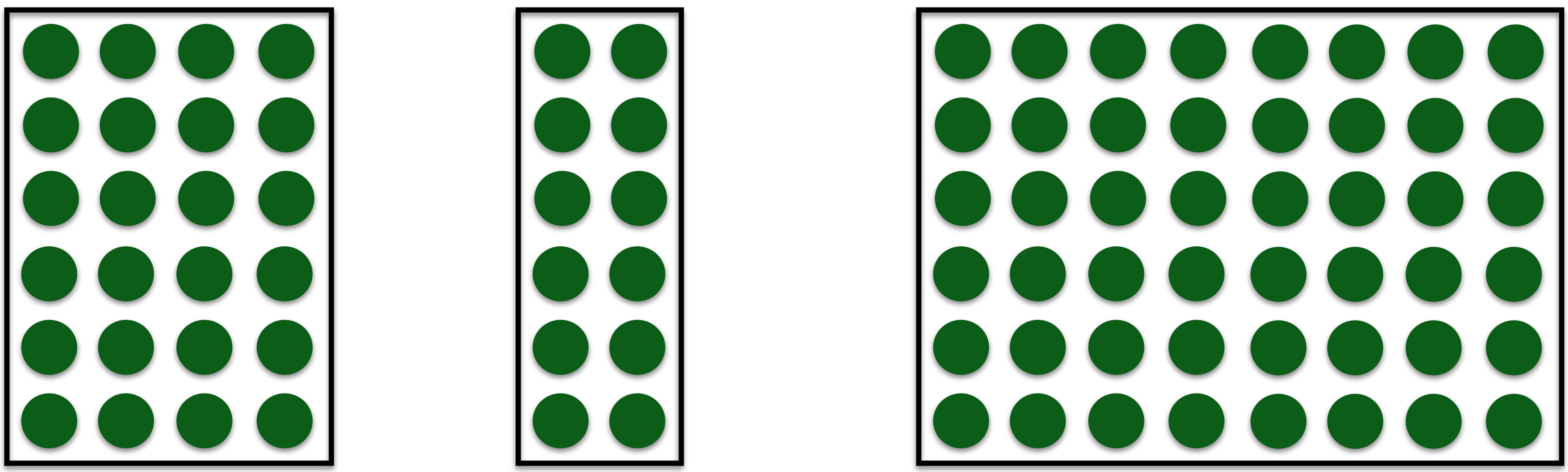


Ich möchte ein Bier



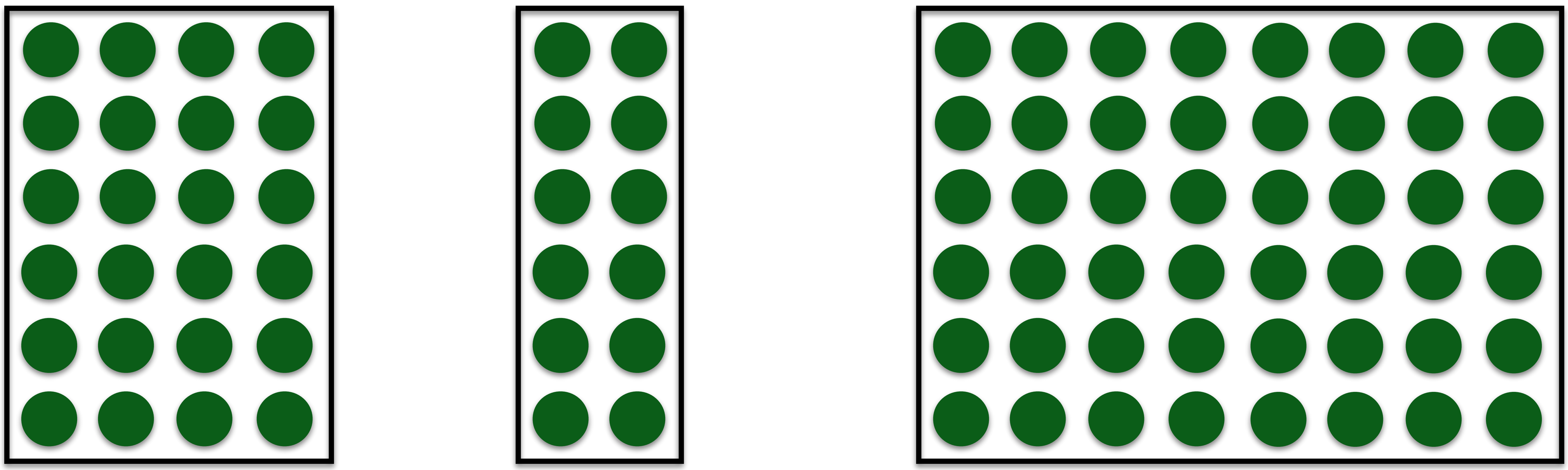
Mach's gut

Sentences as Matrices



Ich möchte ein Bier　*Mach's gut*　*Die Wahrheiten der Menschen sind die unwiderlegbaren Irrtümer*

Sentences as Matrices

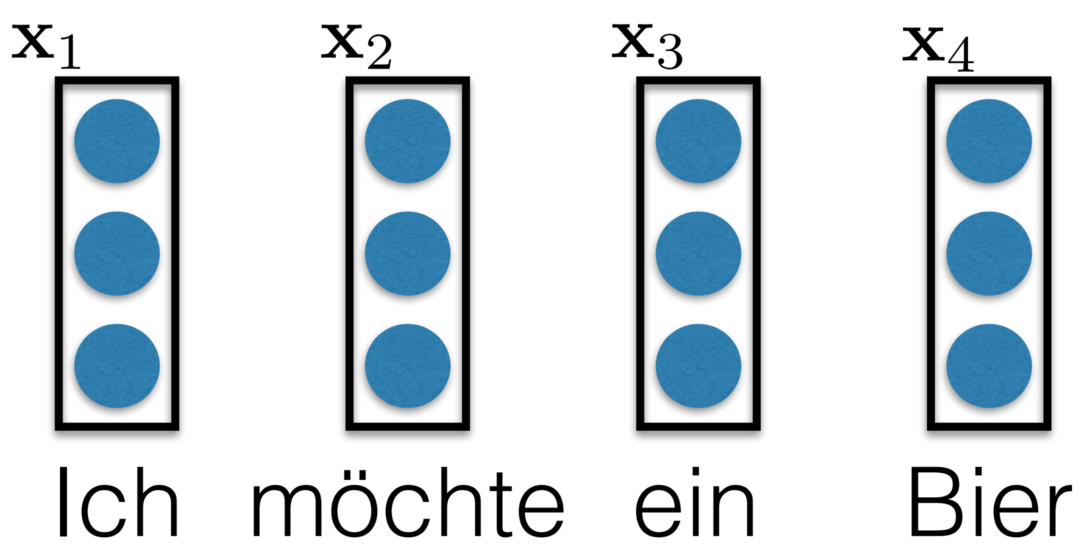


Ich möchte ein Bier　　*Mach's gut*　　*Die Wahrheiten der Menschen sind die unwiderlegbaren Irrtümer*

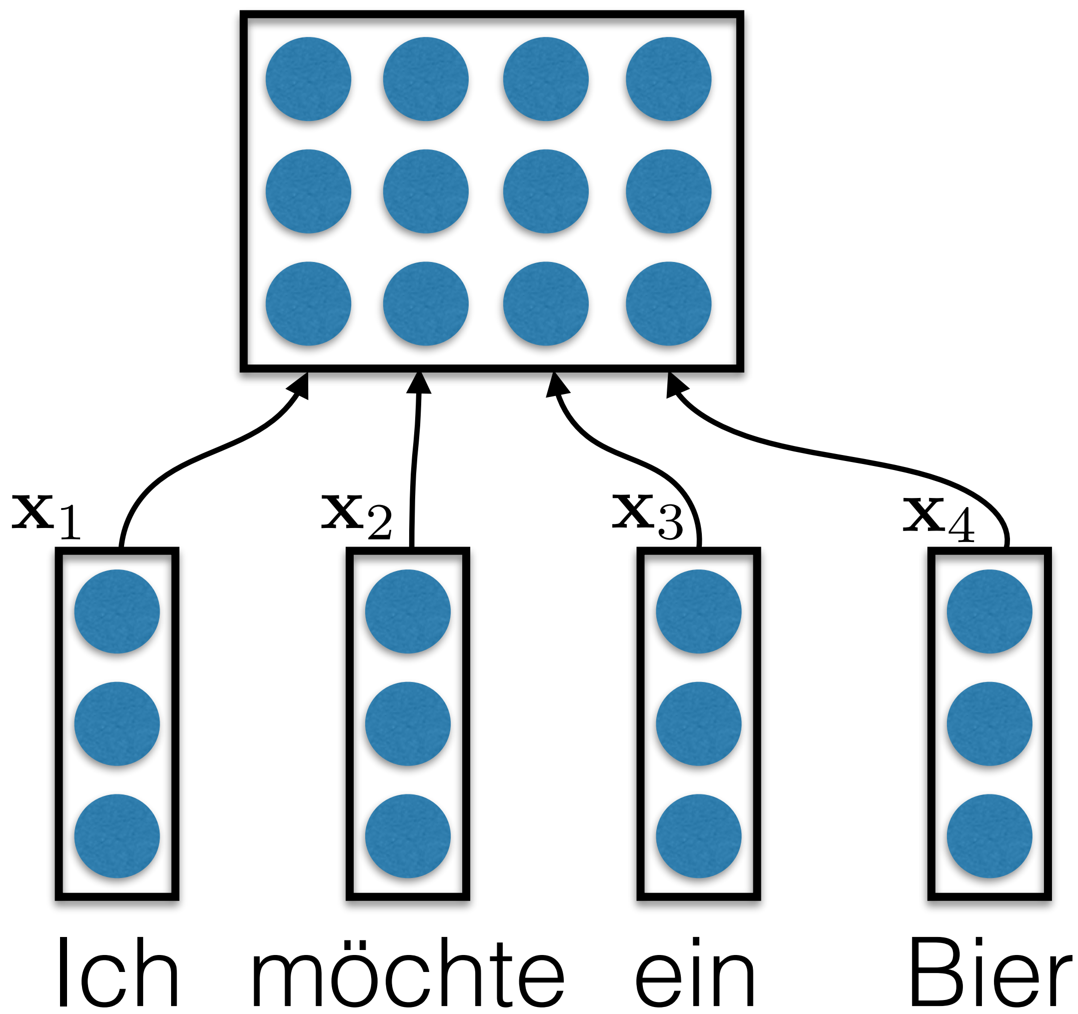
Question: How do we build these matrices?

With Concatenation

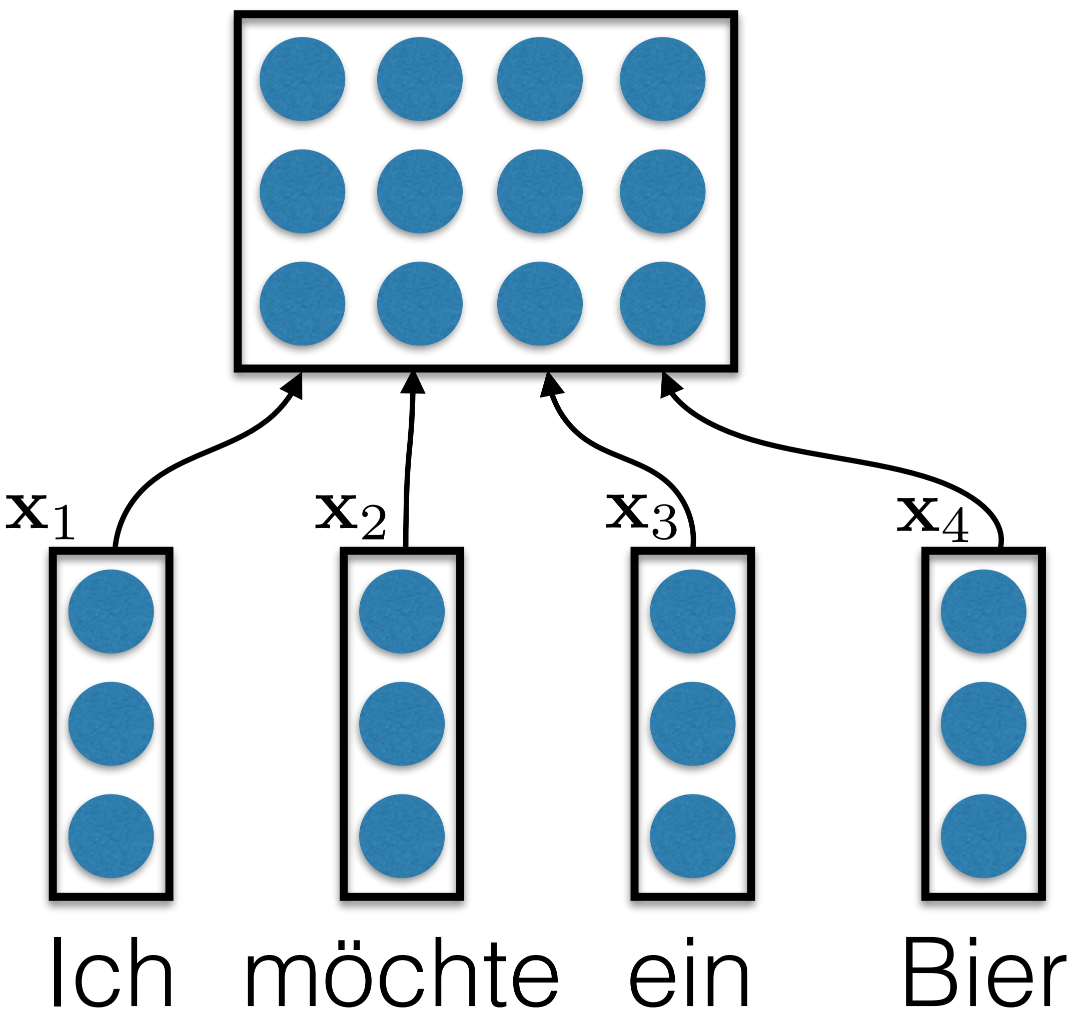
- We can represent a sentence by stacking word vectors into a matrix representing a sentence
- This is easy and fast, but it has the following limitations
  - There is no positional information about the words in the representation
  - Word meanings depend on the context they are used in


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

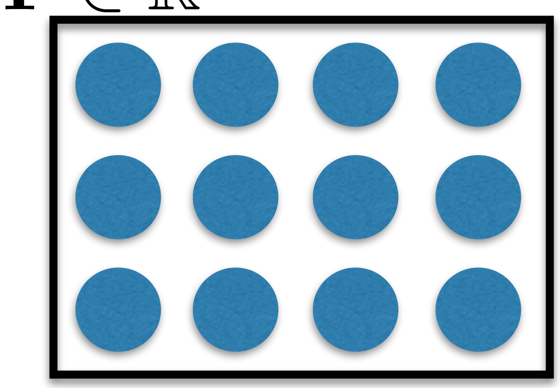
$$\mathbf{f}_i = \mathbf{x}_i$$



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



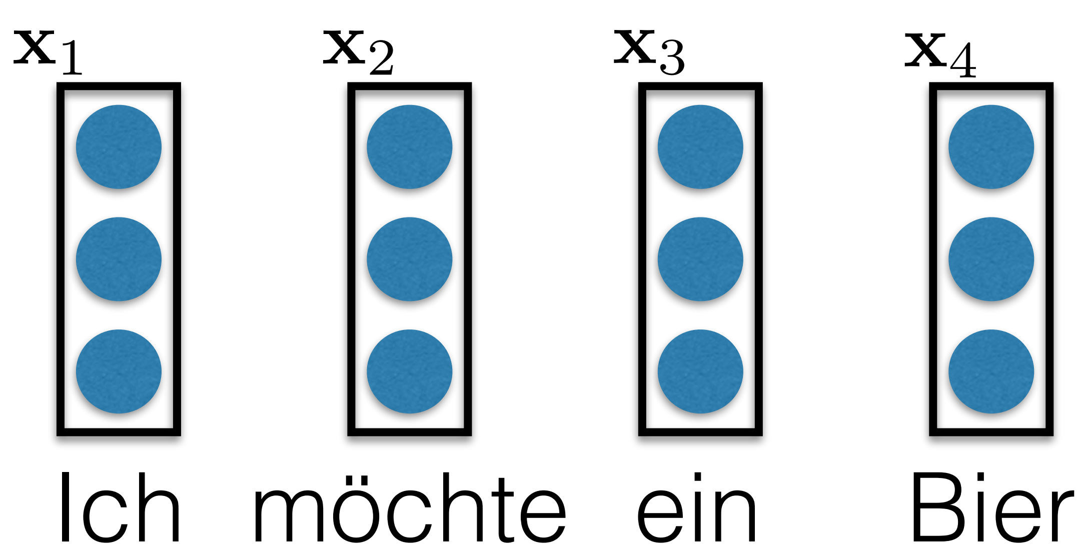
$\mathbf{F} \in \mathbb{R}^{n \times |\boldsymbol{f}|}$



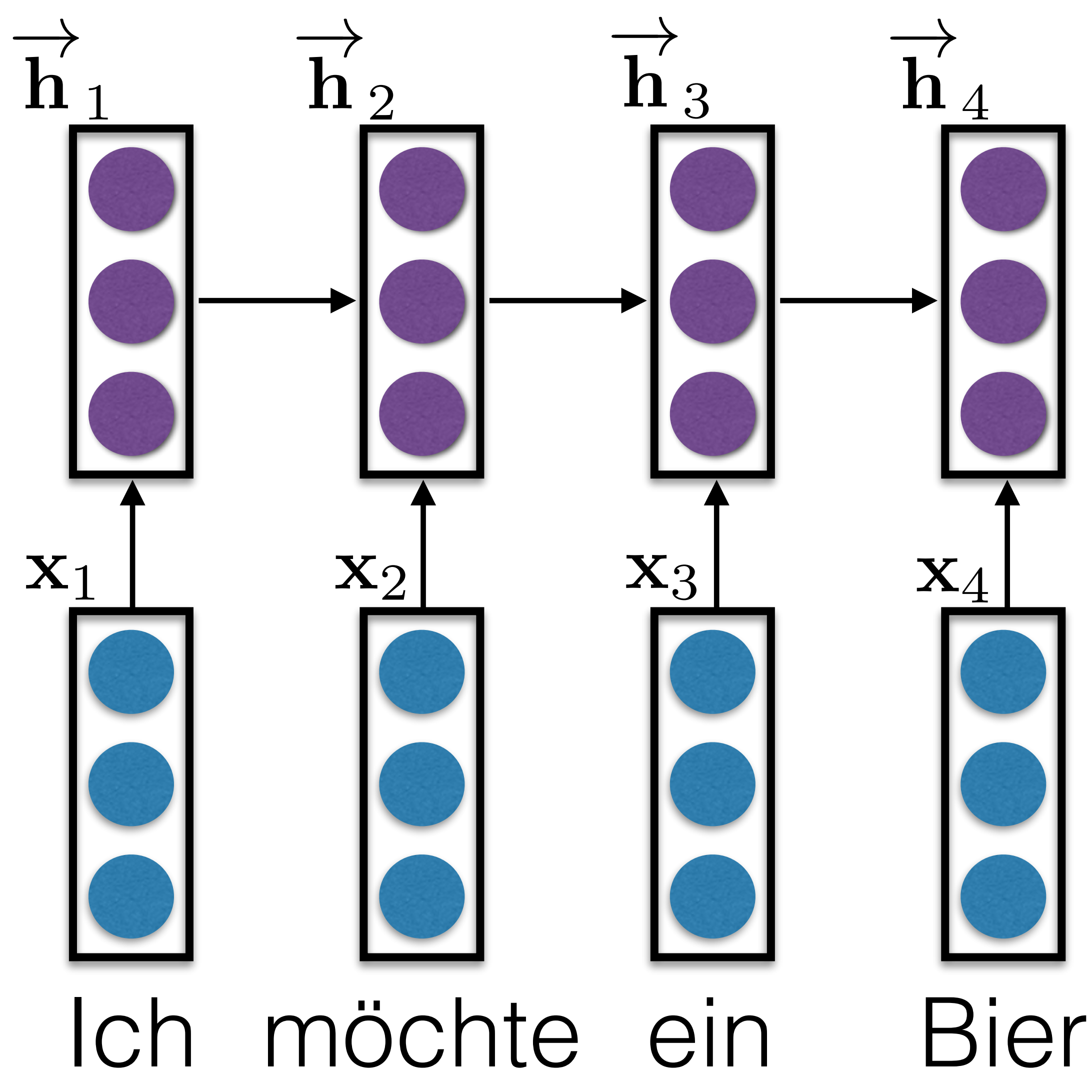
Ich möchte ein Bier

With Bidirectional RNNs

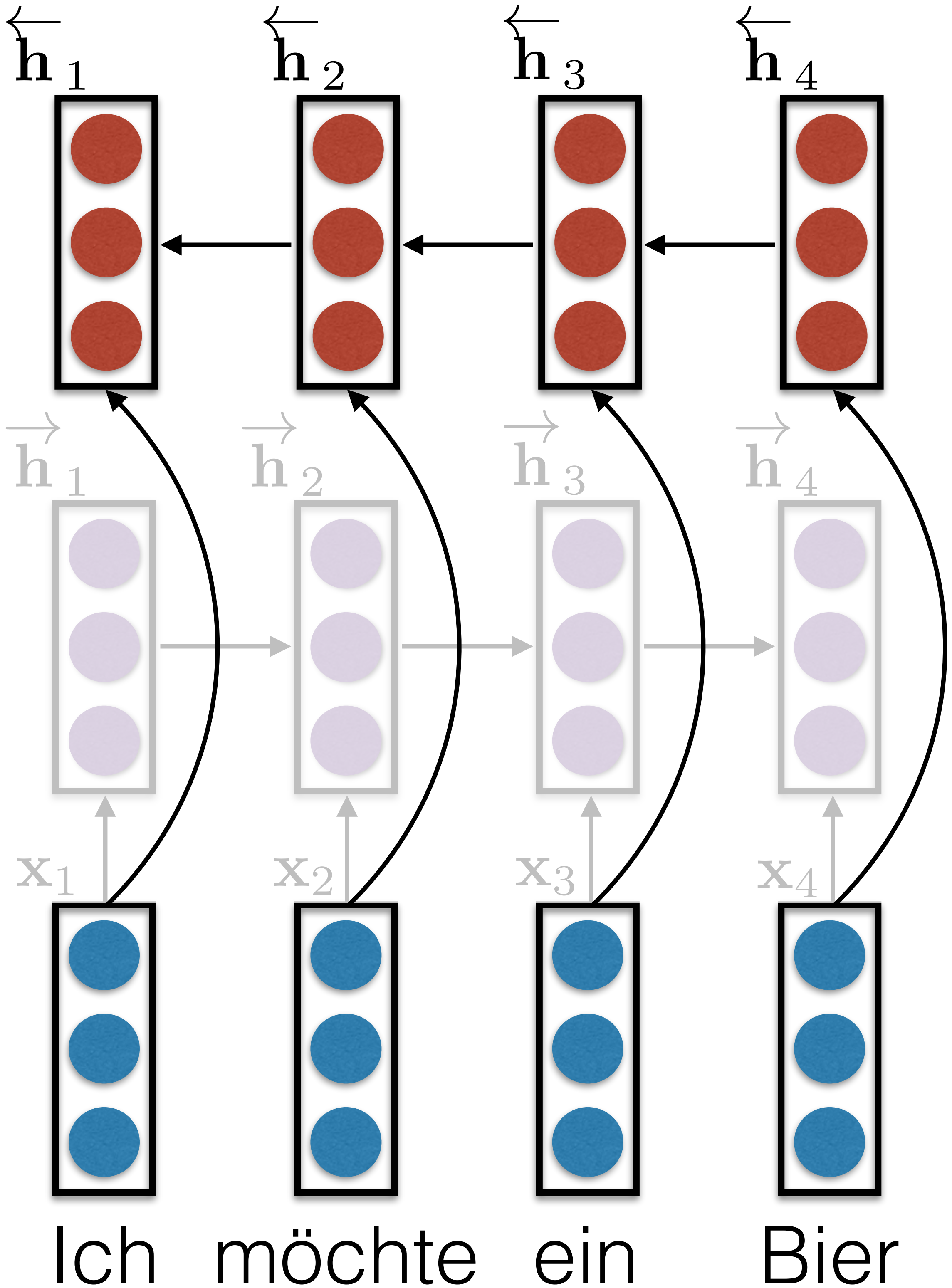
- A widely used matrix representation, due to Bahdanau et al (2015)
- One column per word
- Each column (word) has two halves concatenated together:
    - a “forward representation”, i.e., a word and its left context
    - a “reverse representation”, i.e., a word and its right context
- Implementation: 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

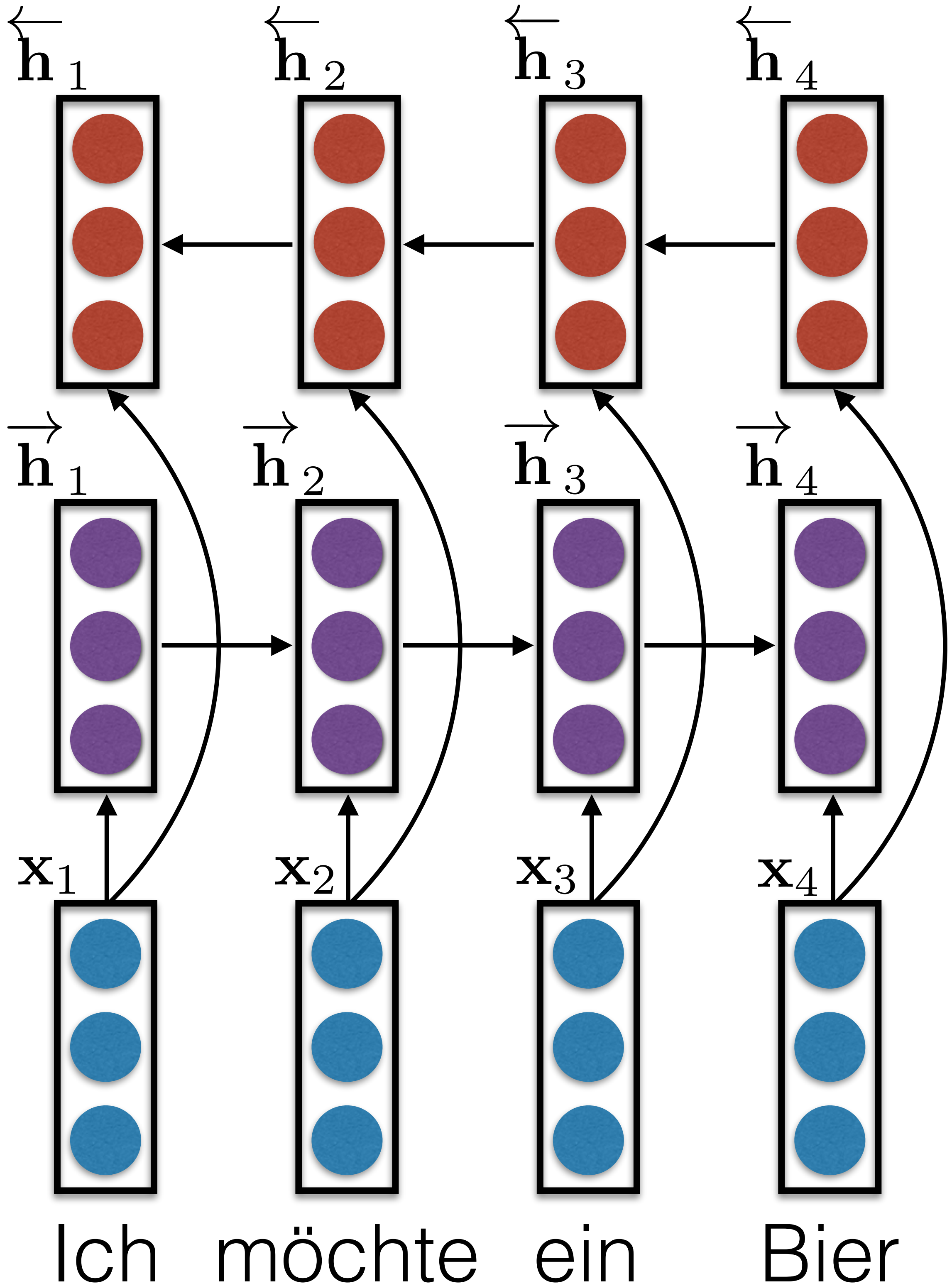


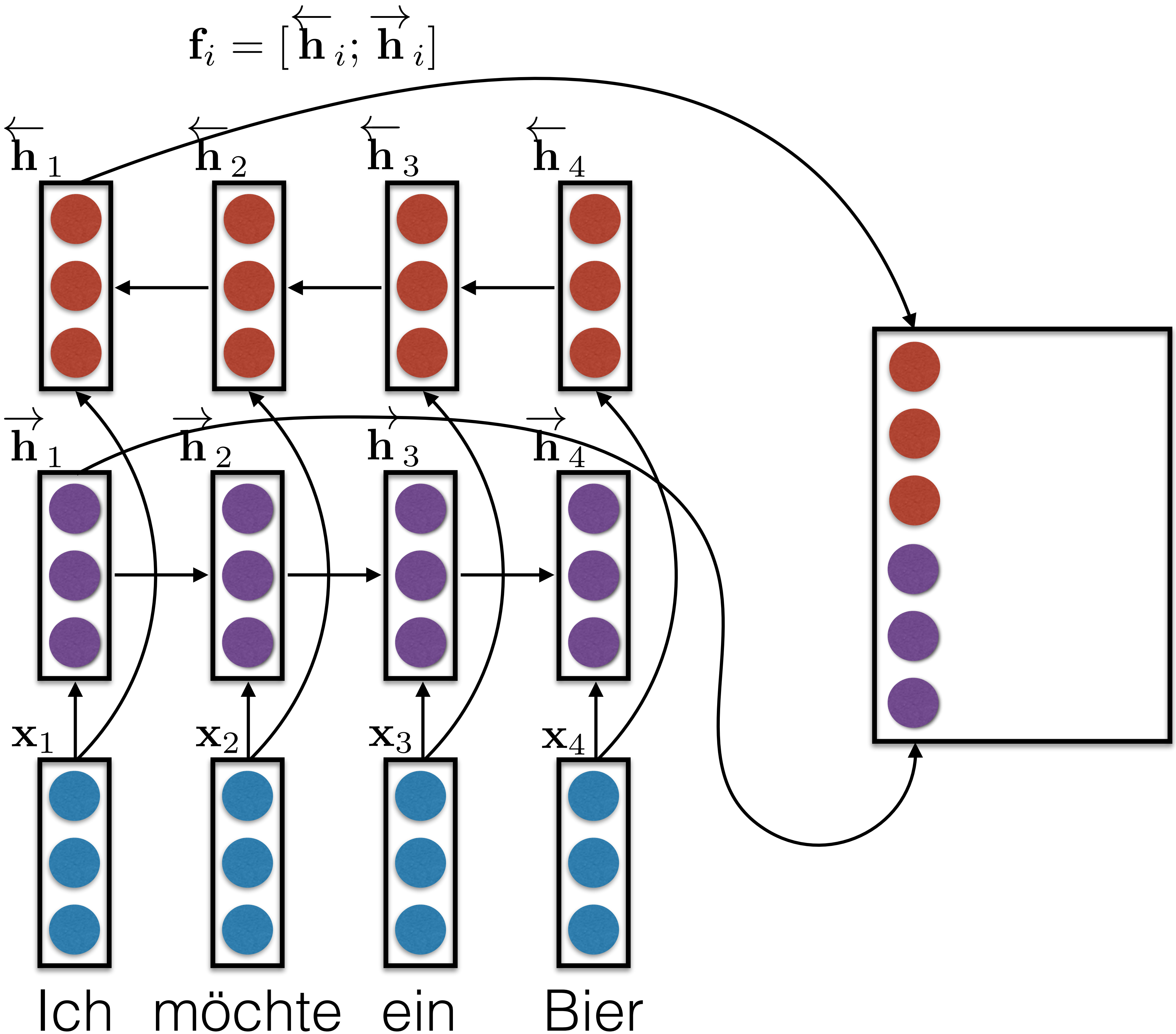
$\overrightarrow{\mathbf{h}}_1$
$\overrightarrow{\mathbf{h}}_2$
$\overrightarrow{\mathbf{h}}_3$
$\overrightarrow{\mathbf{h}}_4$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier



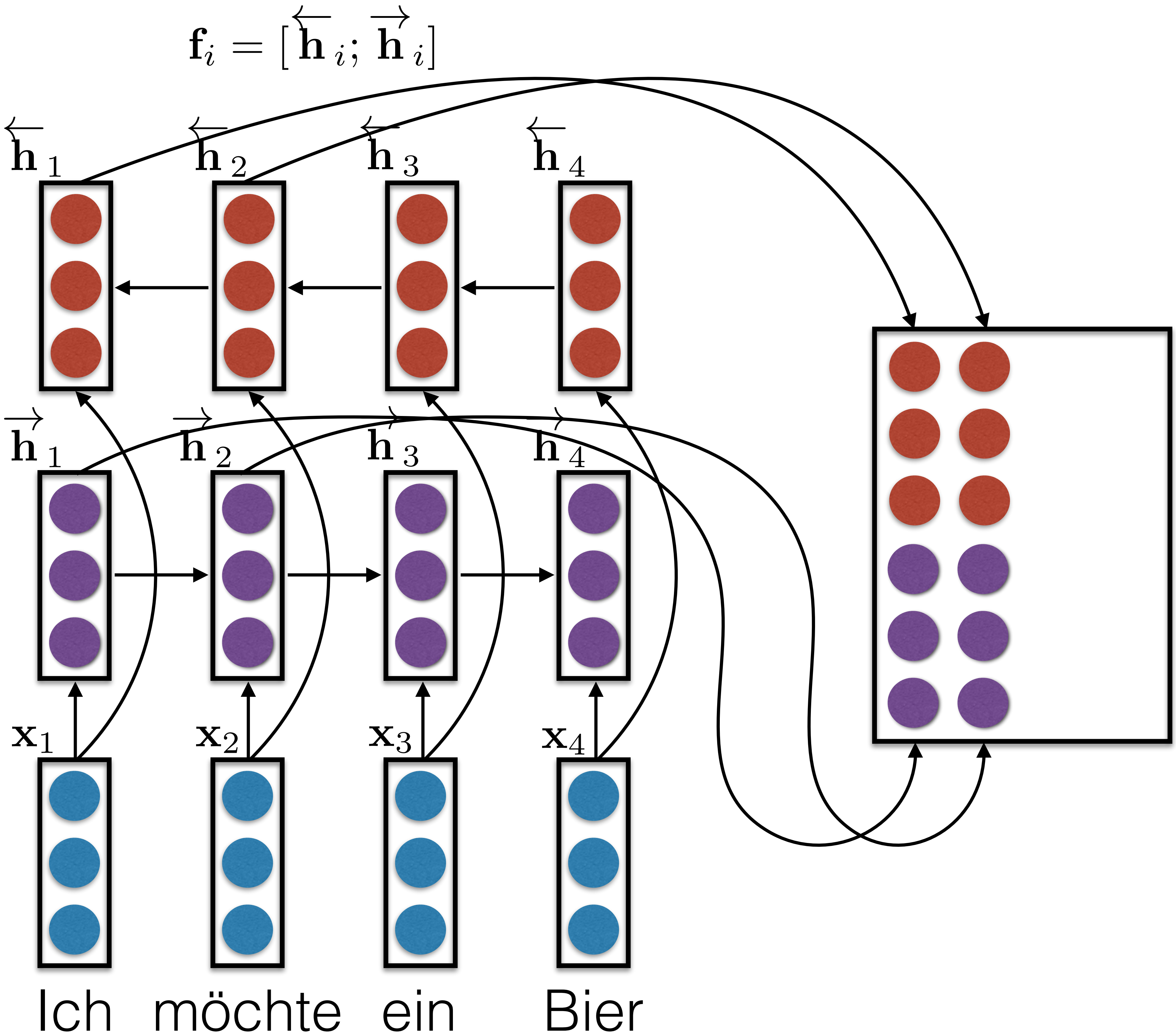
$\overleftarrow{\mathbf{h}}_1$ $\overleftarrow{\mathbf{h}}_2$ $\overleftarrow{\mathbf{h}}_3$ $\overleftarrow{\mathbf{h}}_4$
$\overrightarrow{\mathbf{h}}_1$ $\overrightarrow{\mathbf{h}}_2$ $\overrightarrow{\mathbf{h}}_3$ $\overrightarrow{\mathbf{h}}_4$
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$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

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

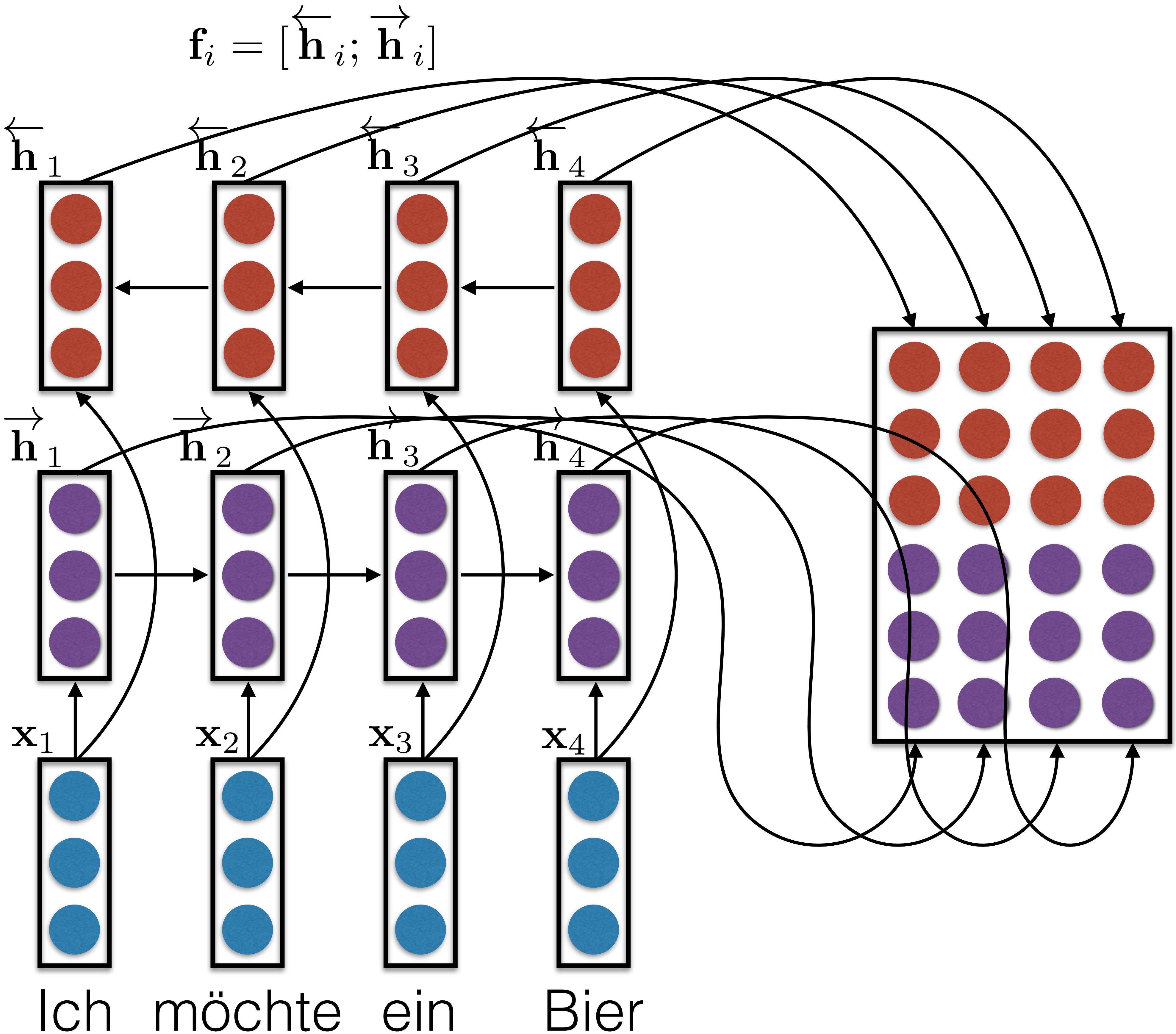




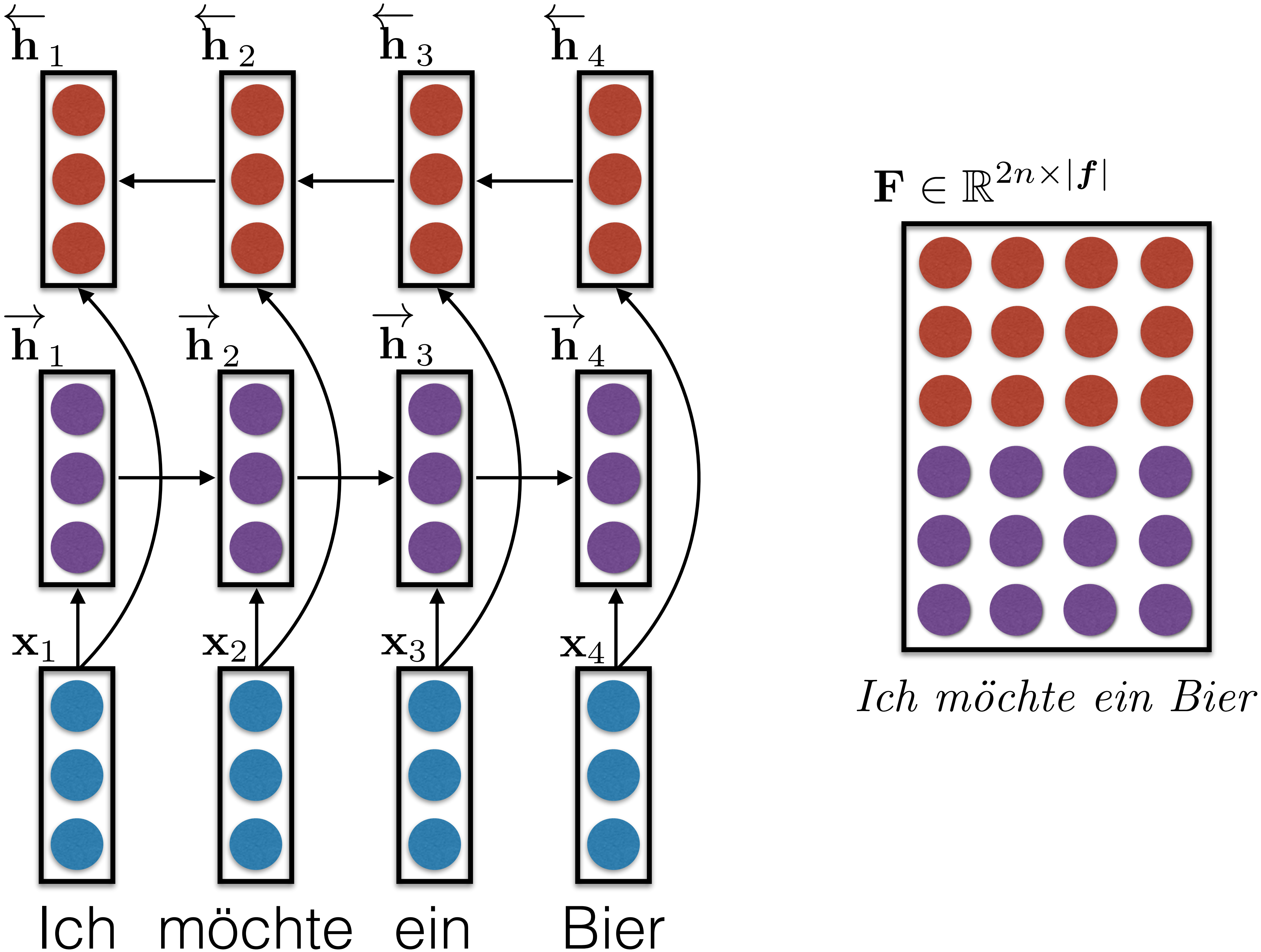
$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$
$\overleftarrow{\mathbf{h}}_1$
$\overleftarrow{\mathbf{h}}_2$
$\overleftarrow{\mathbf{h}}_3$
$\overleftarrow{\mathbf{h}}_4$
$\overrightarrow{\mathbf{h}}_1$
$\overrightarrow{\mathbf{h}}_2$
$\overrightarrow{\mathbf{h}}_3$
$\overrightarrow{\mathbf{h}}_4$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


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

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

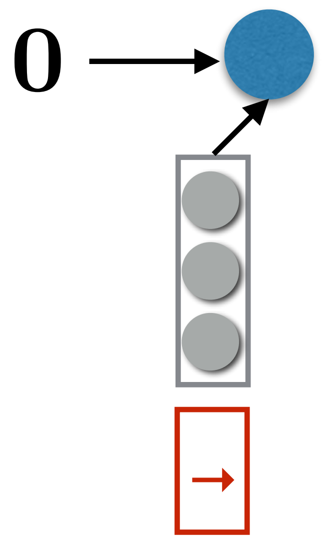


Generation from Matrices

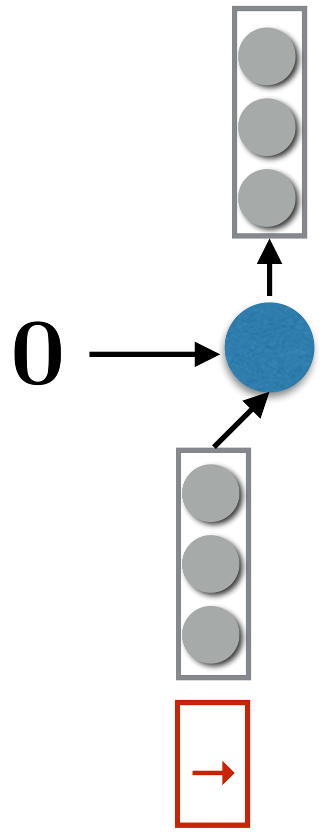
- We have a matrix **F** representing the input, now we need to generate from it
- Bahdanau et al. (2015) were the first to propose using ***attention*** for translating from matrix-encoded sentences
- High-level idea
  - Generate the output sentence word by word using an RNN
  - At each output position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

0

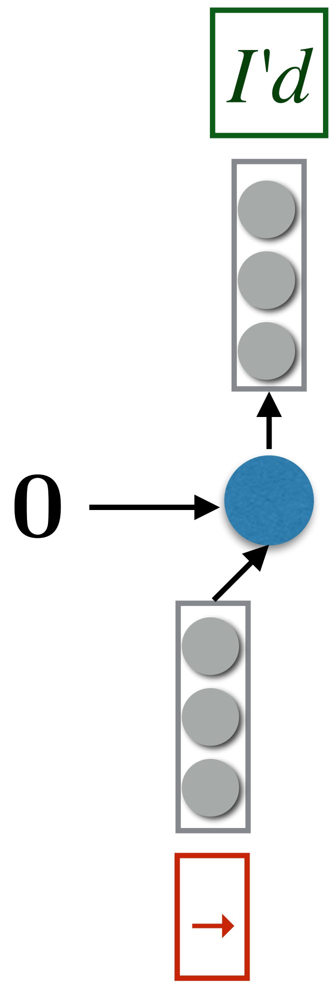
Recall RNNs…



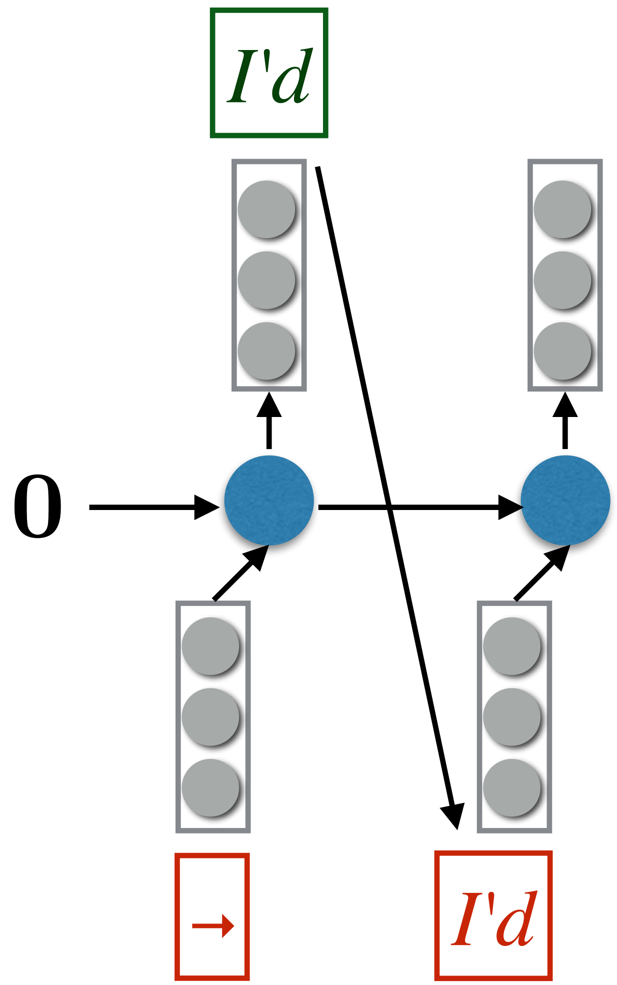
Recall RNNs…



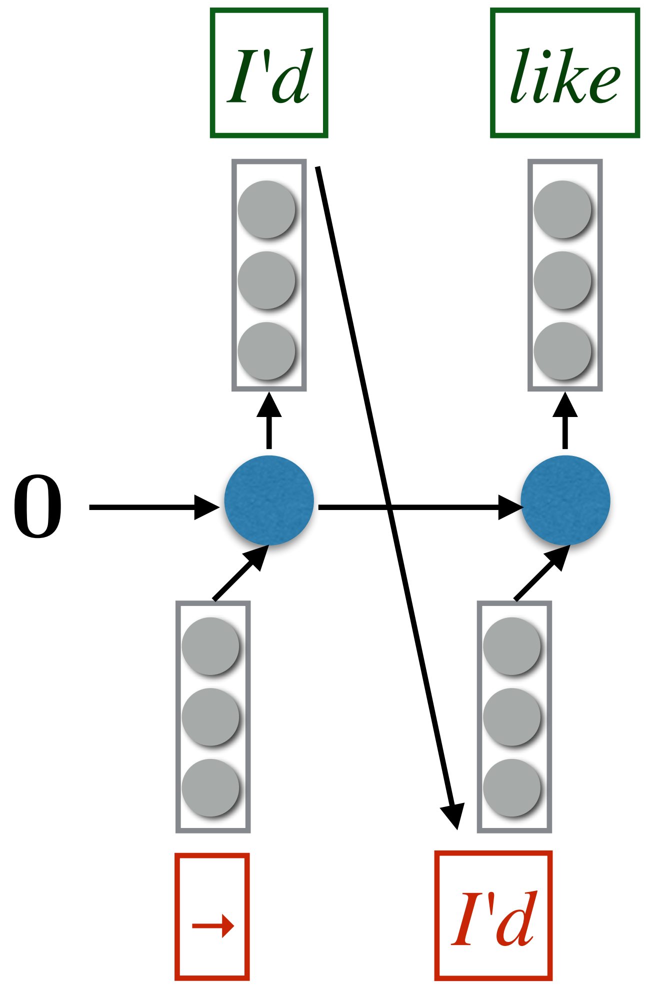
Recall RNNs…



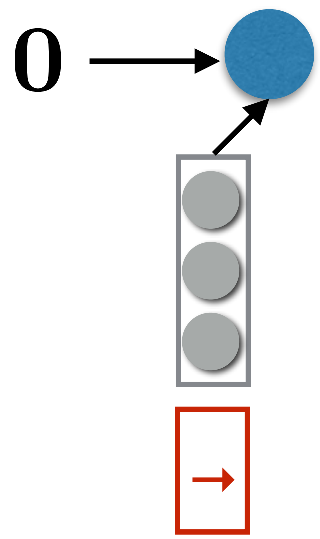
Recall RNNs…

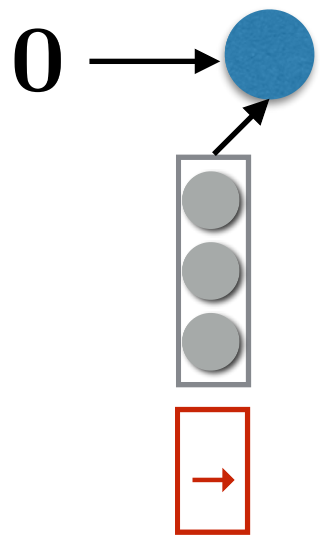


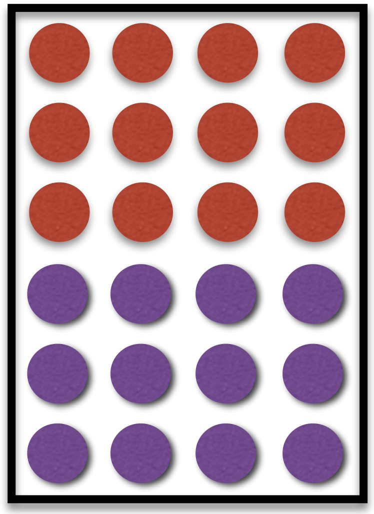
Recall RNNs…



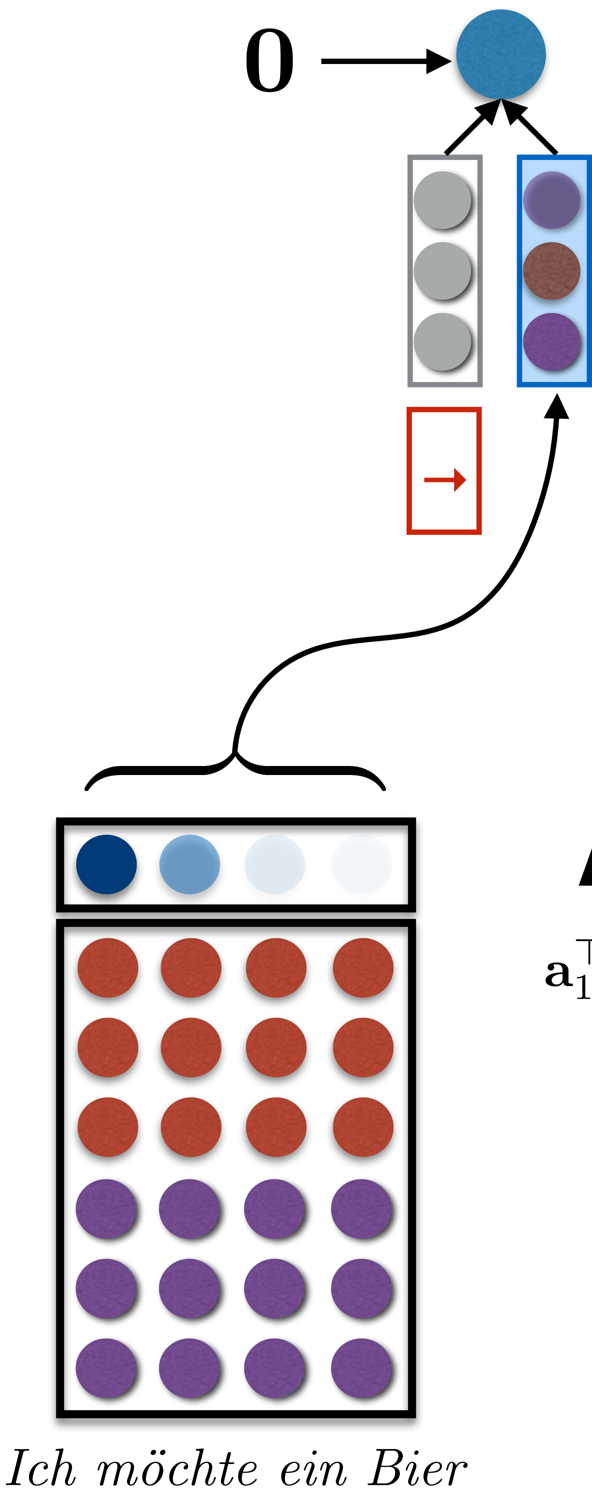
Recall RNNs…


0

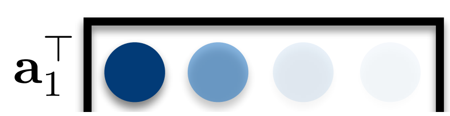


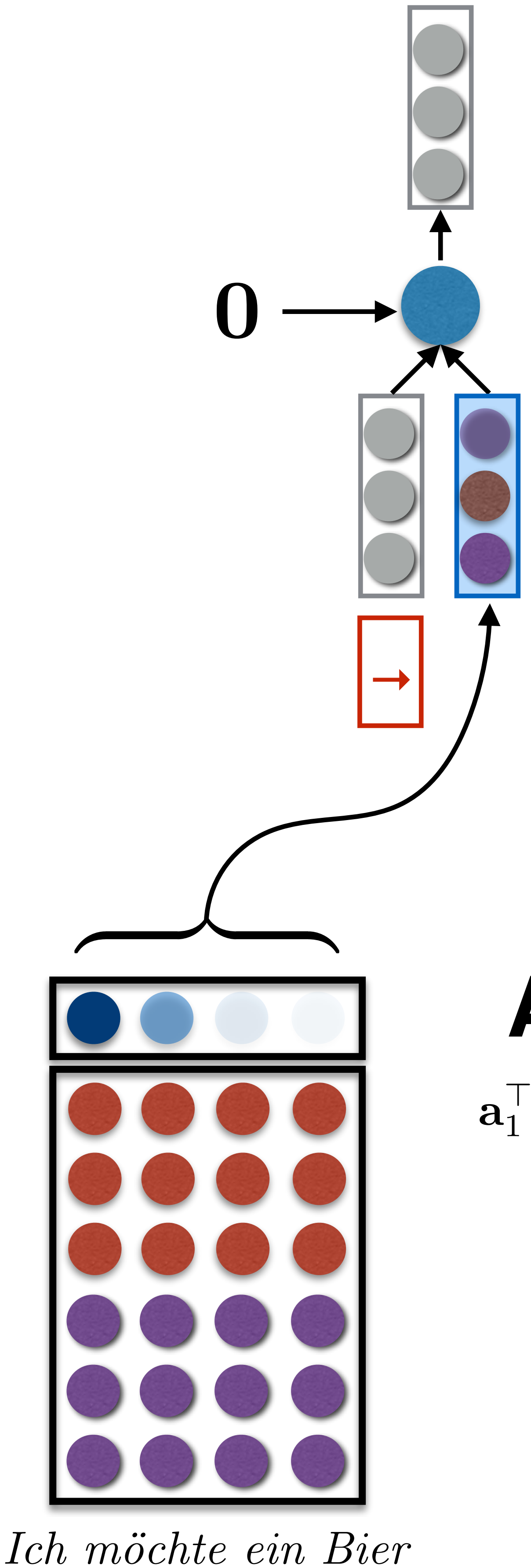


Ich möchte ein Bier

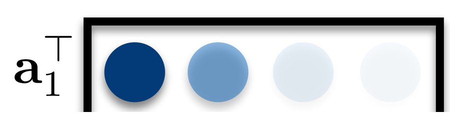


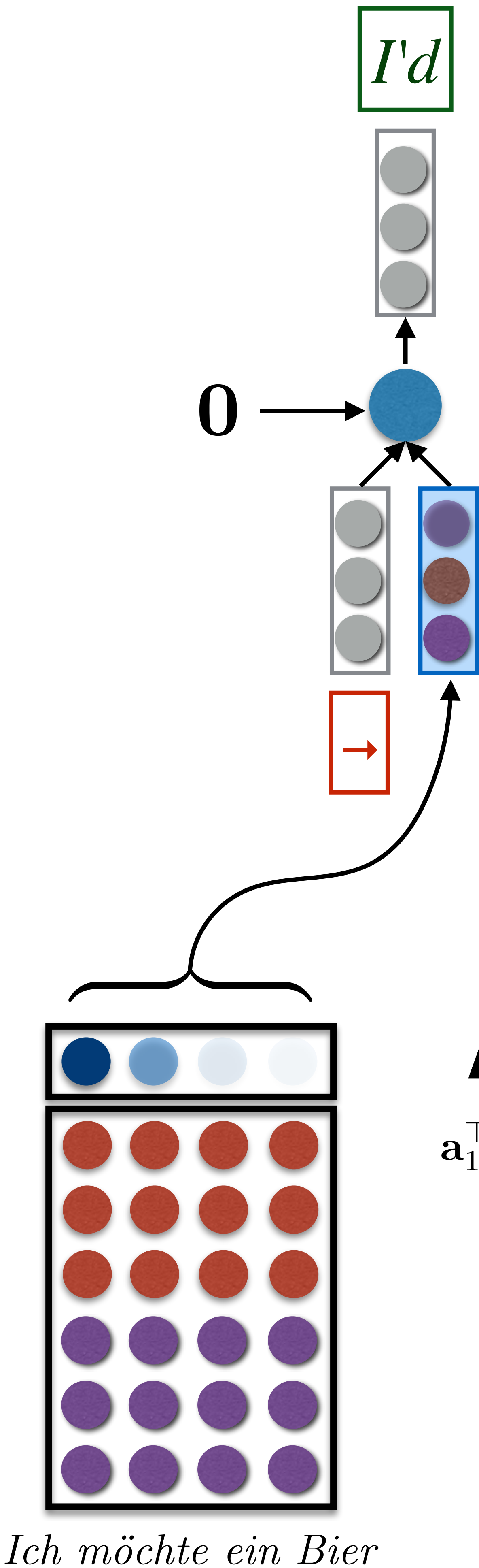
Attention history:



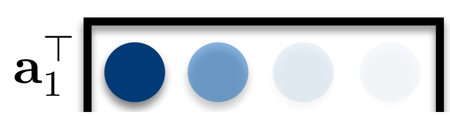


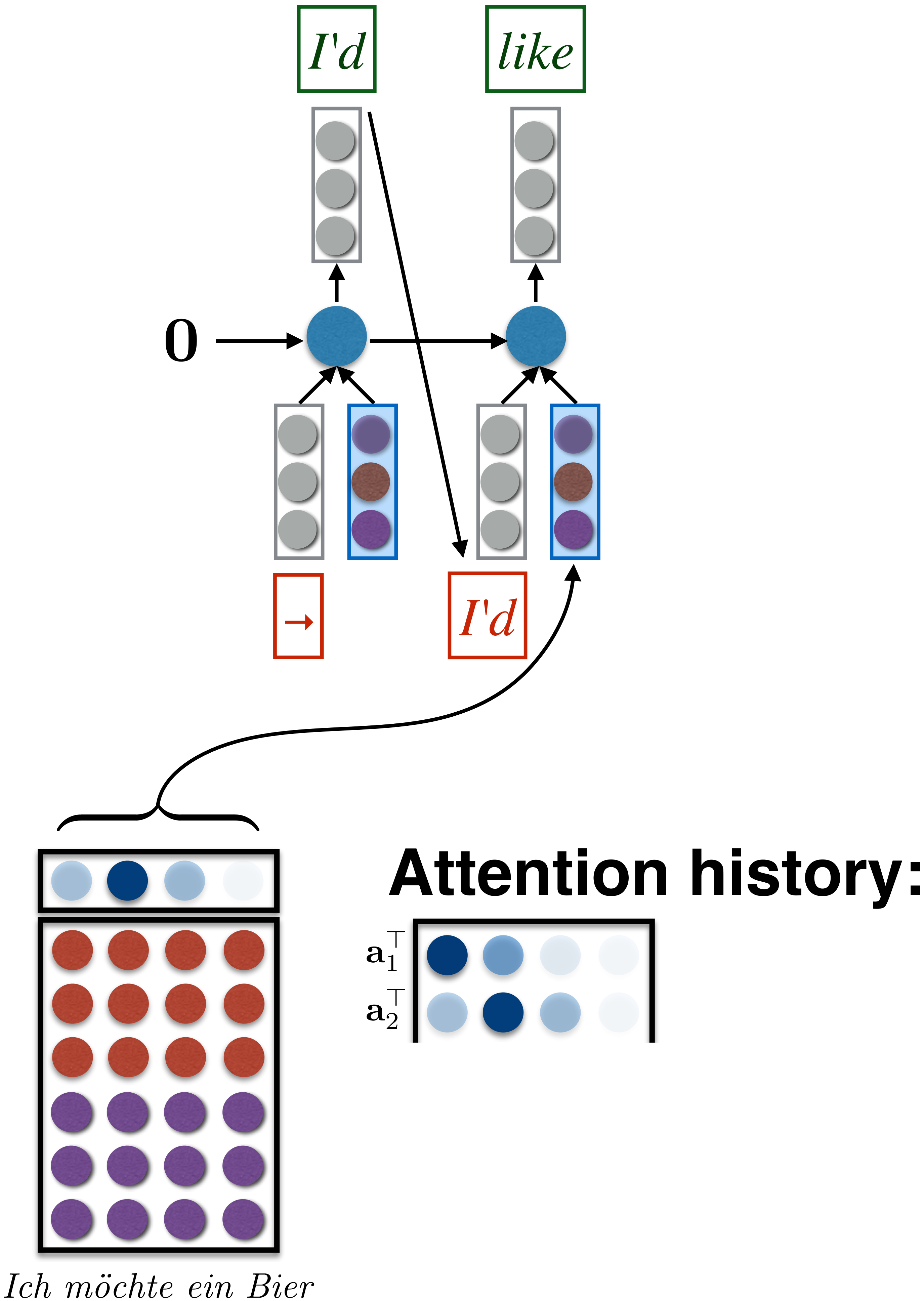
Attention history:

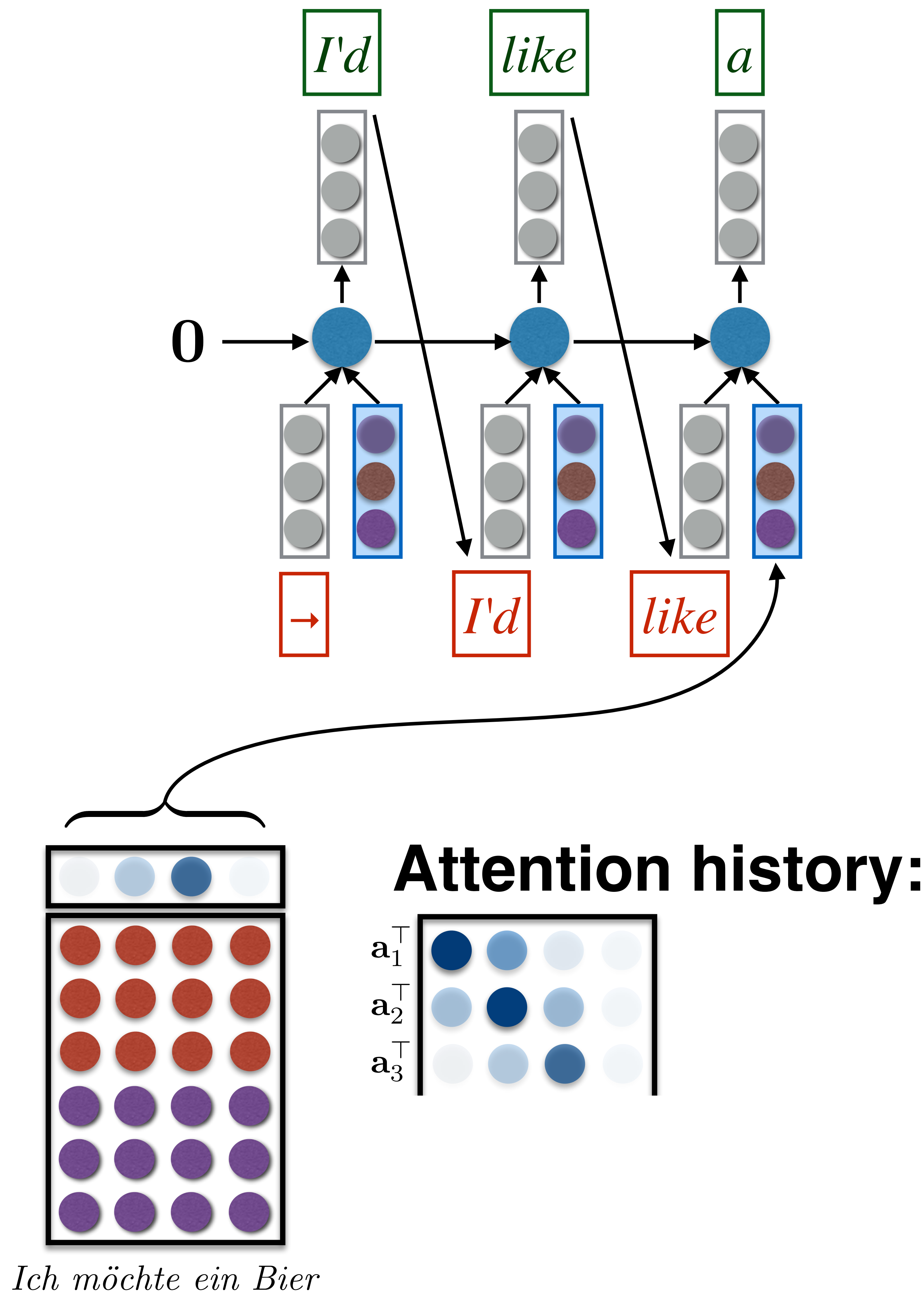


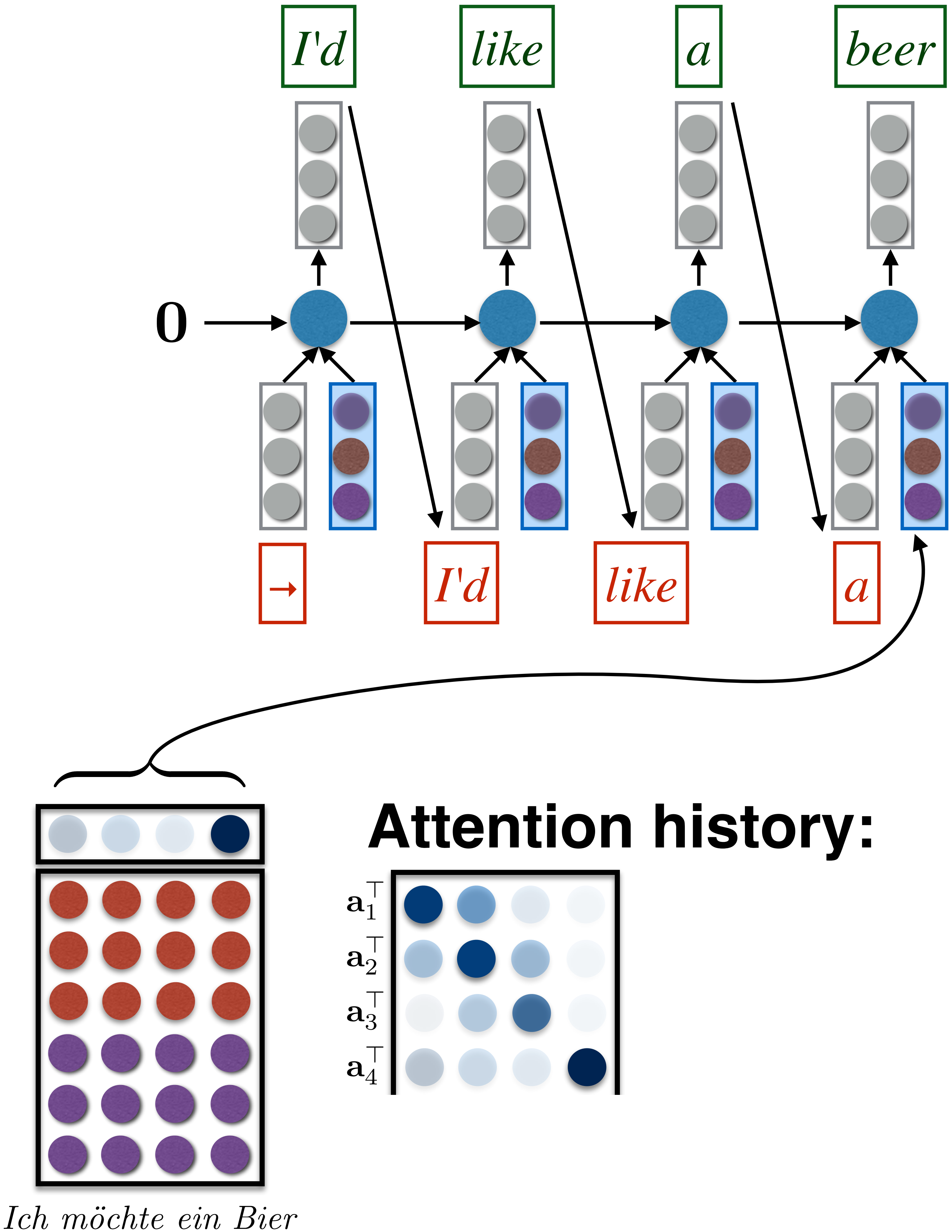


Attention history:

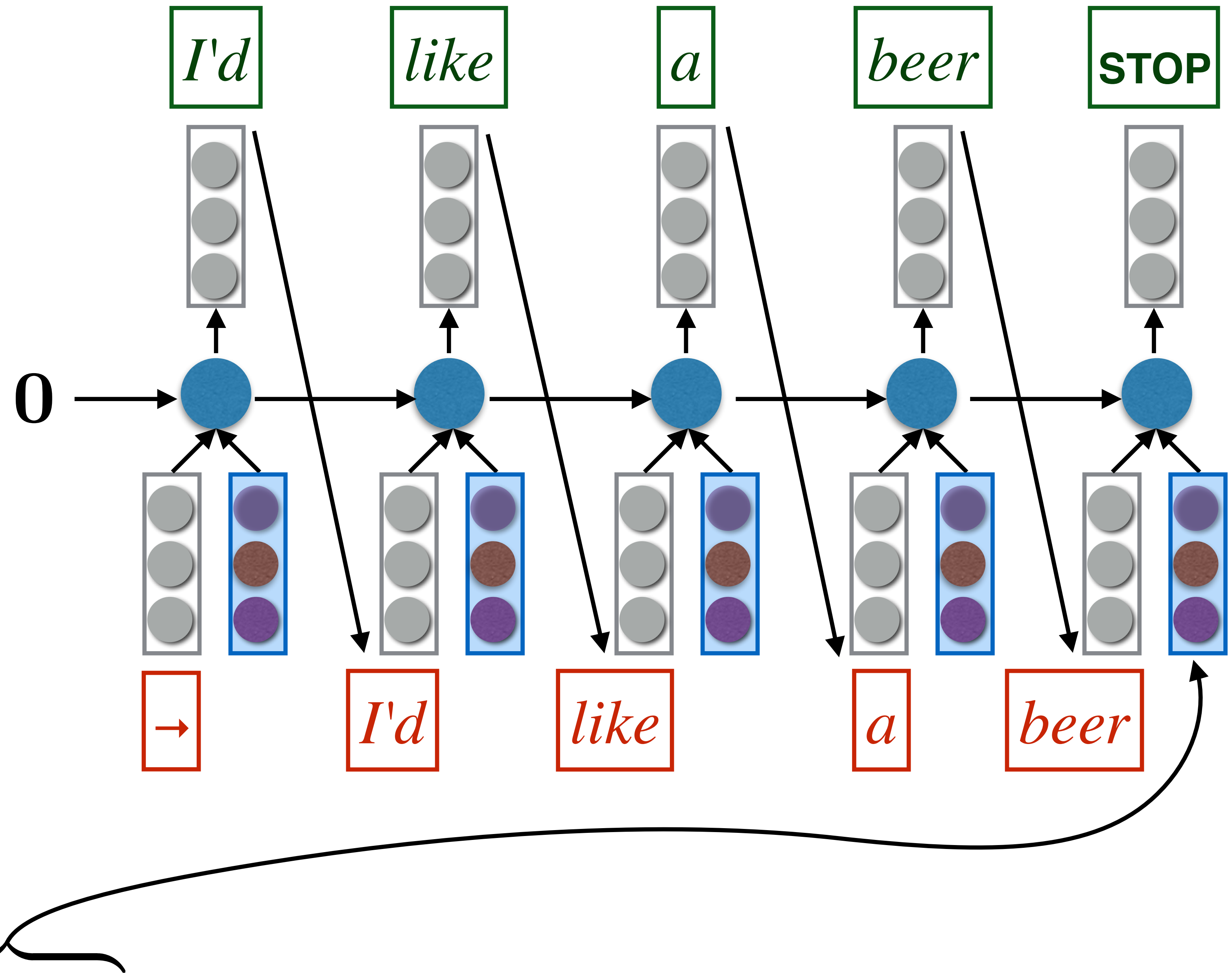


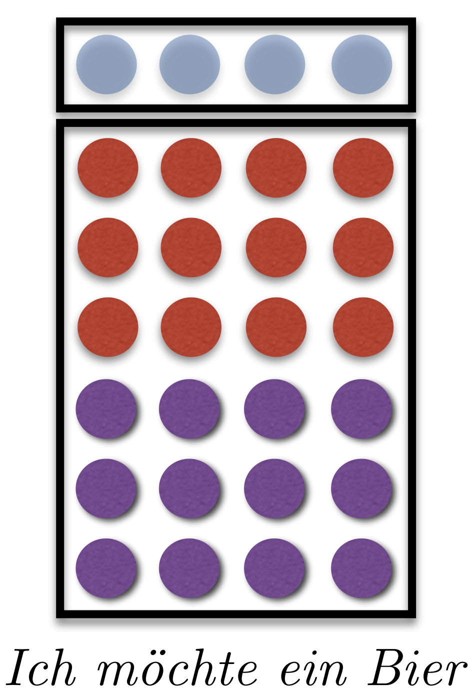

I'd
like
0
→
I'd
Attention history:
$\mathbf{a}_1^\top$
$\mathbf{a}_2^\top$
Ich möchte ein Bier


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

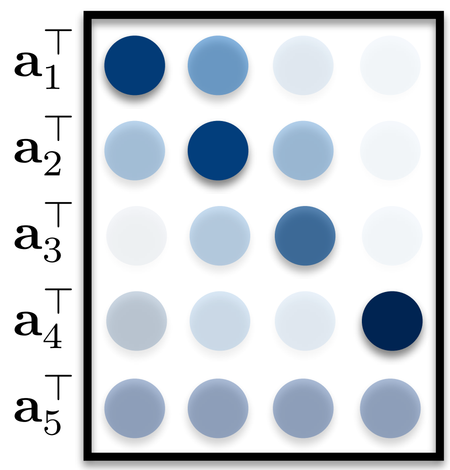


I'd
like
a
beer
0
→
I'd
like
a
Attention history:
$\mathbf{a}_1^\top$
$\mathbf{a}_2^\top$
$\mathbf{a}_3^\top$
$\mathbf{a}_4^\top$
Ich möchte ein Bier





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 word: this is an $|\boldsymbol{f}|$-length vector $\mathbf{a}_t$
  - Here is a simplified version of Bahdanau et al.'s solution
    - Use an RNN to predict model output, call the hidden states $\mathbf{s}_t$
      ($\mathbf{s}_t$ has a fixed dimensionality, call it m)

Computing Attention

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

Computing Attention

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

Computing Attention

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

Computing Attention

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

Summary

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

Representing Words in Context with Self-Attention

- RNNs are computationally inconvenient: to compute $\mathbf{h}_t$, we need to first compute $\mathbf{h}_{t-1}$, for which we need to compute $\mathbf{h}_{t-2}\ldots$

- LSTMs have to use their memories to remember everything in the past

- We will solve both of these problems with **self-attention**.

    - Each $\mathbf{h}_t$ will be computed in parallel (take advantage of GPUs which can do a lot of things in parallel)

    - Each $\mathbf{h}_t$ will be able to create a direct “connection” to anything else in the sequence without resorting to a single vector “memory”

- This architecture is called a “transformer”

我	**昨天**	**看了**	**三部**	**电影**
wǒ	zuótiān	kànle	sān bù	diànyǐng

我	昨天	看了	三部	电影
wǒ	zuótiān	kànle	sān bù	diànyǐng
I me	yesterday	watch look watched looked read	three	movie movies

I watched three movies yesterday.

我	昨天	看了	三部	电影
wǒ	zuótiān	kànle	sān bù	diànyǐng
I me	yesterday	watch look watched looked read	three	movie movies

Chinese pronouns don't indicate whether they are subjects or objects!

But in English, we need to know this.

我	昨天	看了	三部	电影
wǒ	zuótiān	kànle	sān bù	diànyǐng
I me	yesterday	watch look watched looked read	three	movie movies

Chinese pronouns don't indicate whether they are subjects or objects!

But in English, we need to know this.

我	昨天	看了	三部	电影
wǒ	zuótiān	kànle	sān bù	diànyǐng
I me	yesterday	watch look watched looked read	three	movie movies

Chinese nouns don’t indicate whether they are singular or plural!

But in English, we need to know this.

我	昨天	看了	三部	电影
wǒ	zuótiān	kànle	sān bù	diànyǐng
I me	yesterday	watch look watched looked read	three	movie movies

Chinese nouns don’t indicate whether they are singular or plural!

But in English, we need to know this.

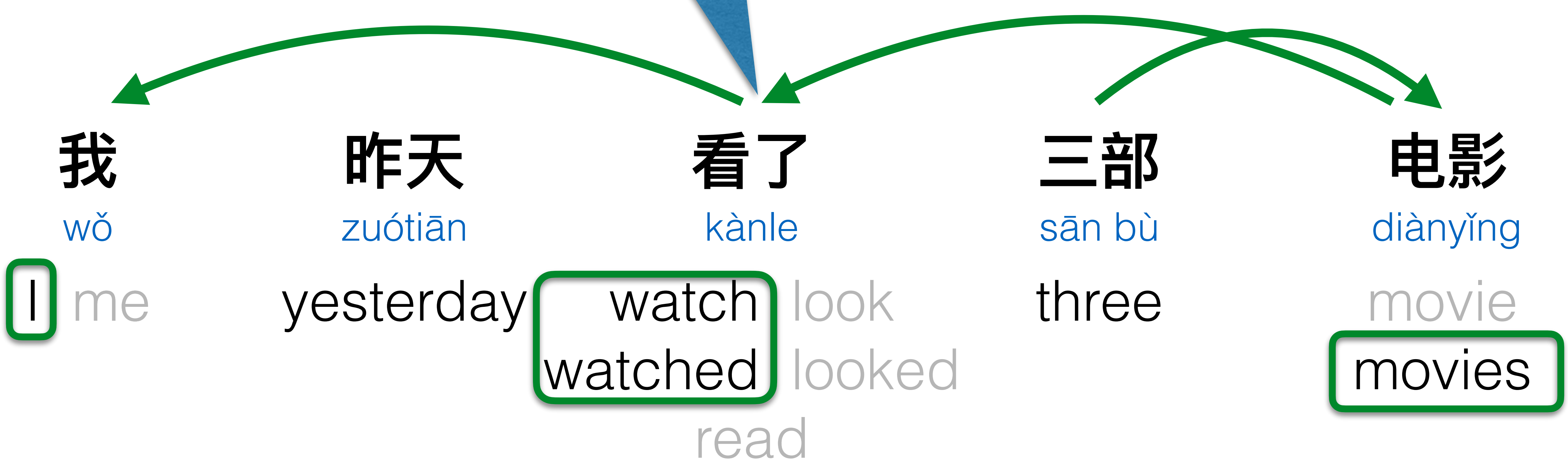
我	昨天	看了	三部	电影
wǒ	zuótiān	kànle	sān bù	diànyǐng
I me	yesterday	watch look	three	movie
		watched looked		movies
		read		


Chinese verbs don’t map directly on to English verbs! The right verb depends on the object.
我
wǒ
I
me
昨天
zuótiān
yesterday
看了
kànle
watch
look
watched
looked
read
三部
sān bù
three
电影
diànyǐng
movie
movies

Chinese verbs don’t map directly on to English verbs! The right verb depends on the object.
我
wǒ
I
me
昨天
zuótiān
yesterday
看了
kànle
watch
watched
look
looked
read
三部
sān bù
three
电影
diànyǐng
movie
movies

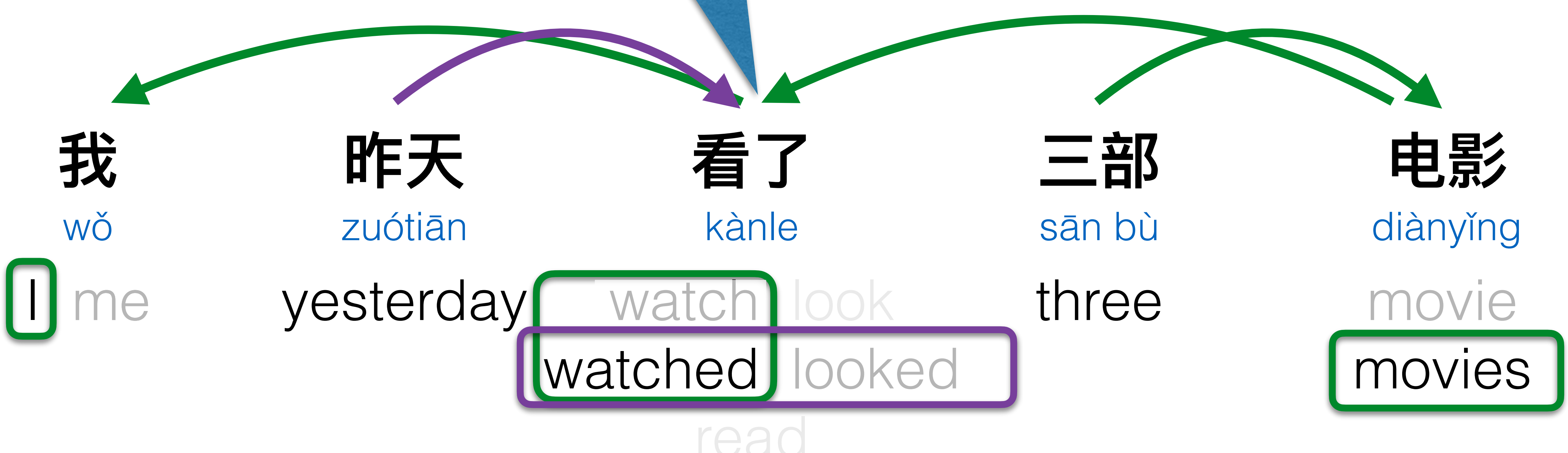
Chinese verbs don’t inflect for tense!

But in English, we need to know this.

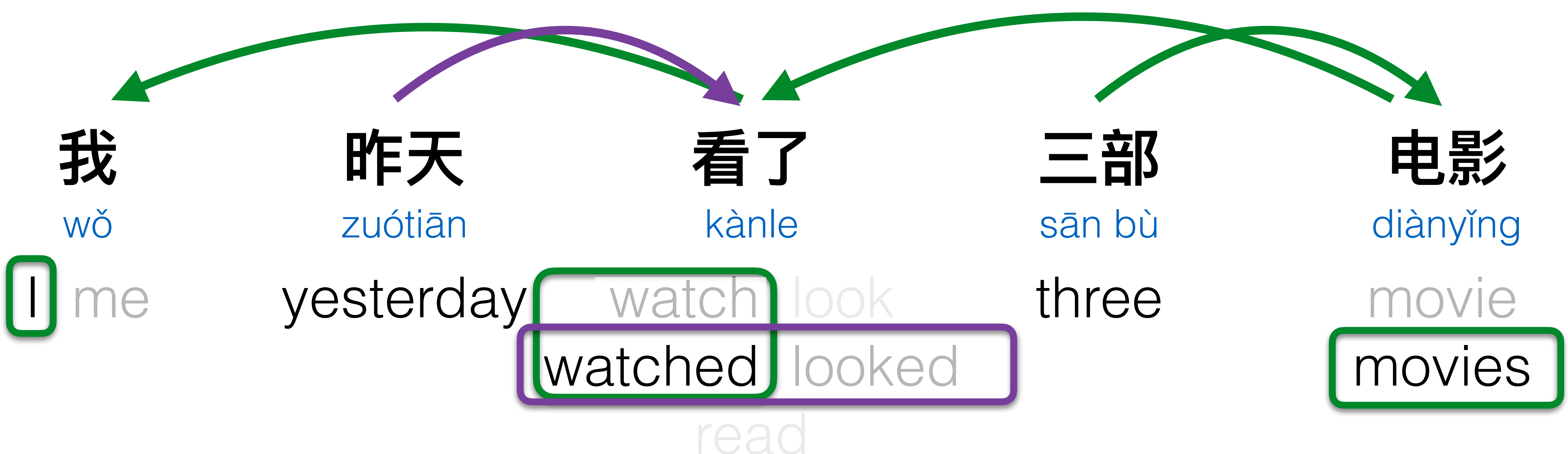


Chinese verbs don’t inflect for tense!

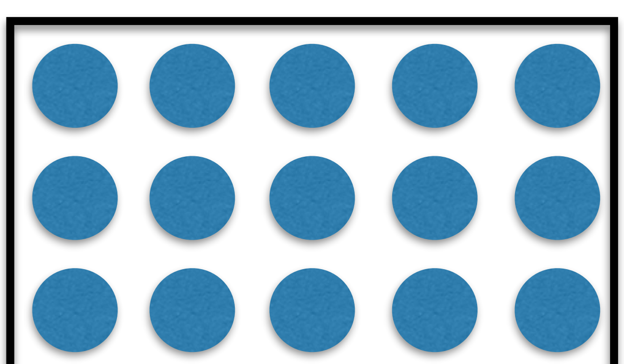
But in English, we need to know this.



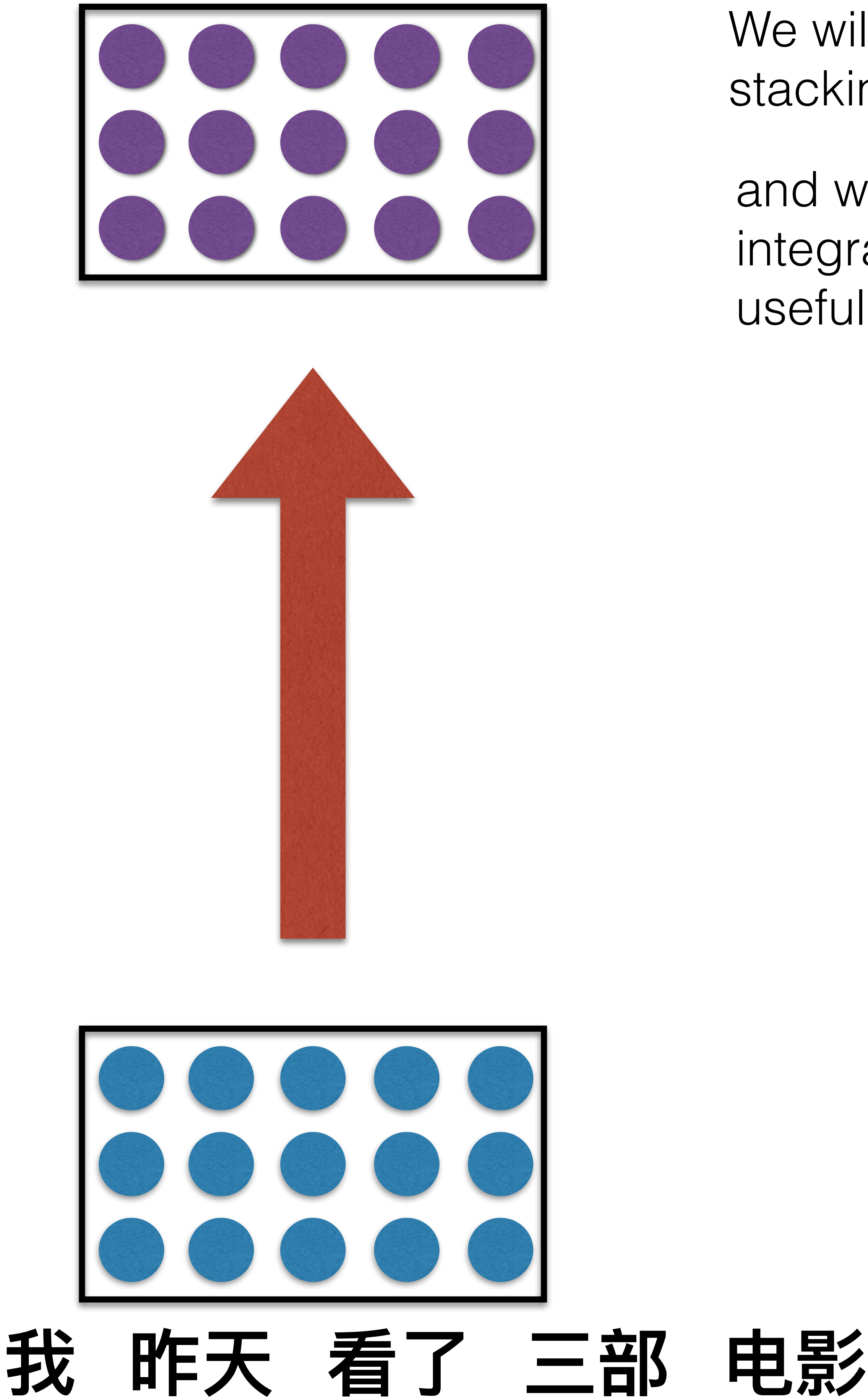
- Different words need to obtain **different kinds of information** from different places.
- Words need to integrate **multiple kinds of information**.
- Although we didn't consider an example, words may need to pass information along **multiple hops**.
- Let's design a model that supports this.



We will start with $\mathbf{X} \in \mathbb{R}^{n \times d}$ which is obtained to by stacking word vectors …



我 昨天 看了 三部 电影



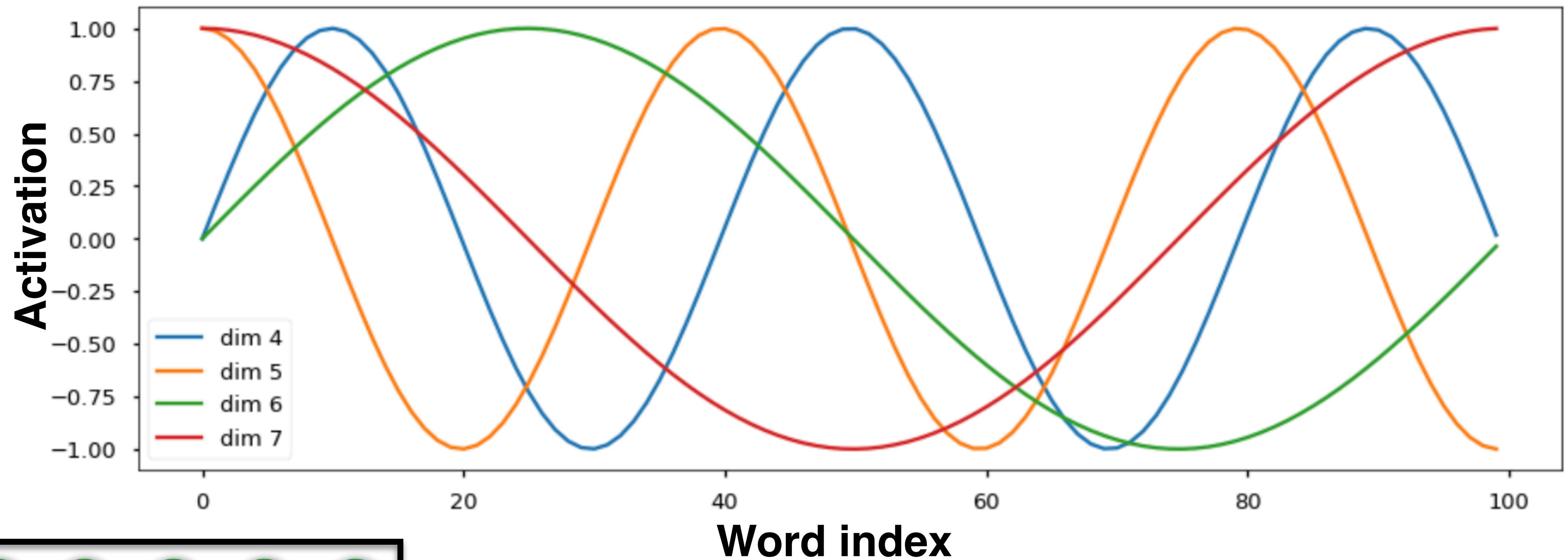
We will start with $\mathbf{X} \in \mathbb{R}^{n \times d}$ which is obtained to by stacking word vectors …

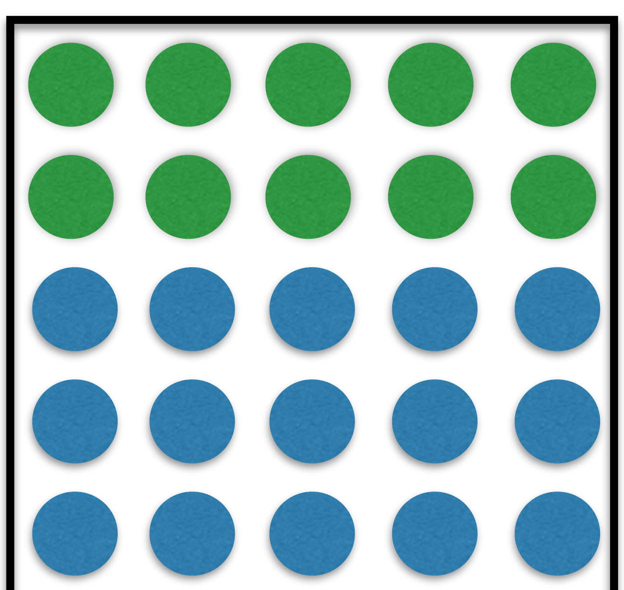
and we will transform it into a representation that integrates all the necessary contextual information useful for the task.

We will start with $\mathbf{X} \in \mathbb{R}^{n \times d}$ which is obtained to by stacking word vectors.

Since we need information about positions, we need to augment $\mathbf{X}$ with positional information.

Plot by Sasha Rush

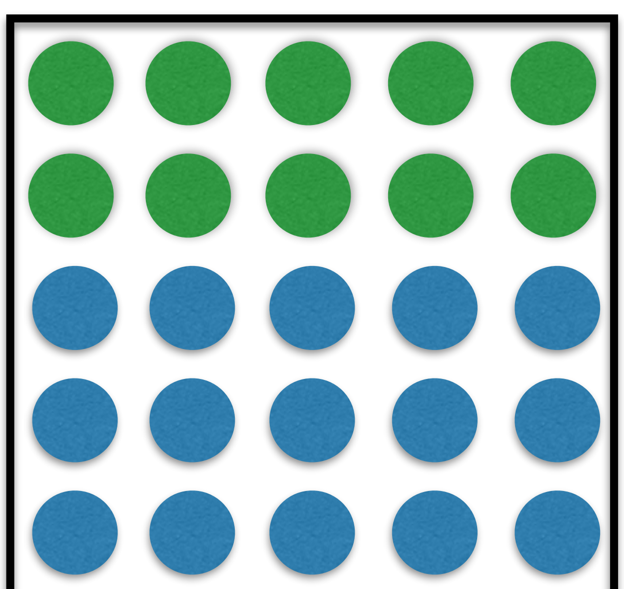




我 昨天 看了 三部 电影

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

$\mathbf{X} \in \mathbb{R}^{n \times d}$ is obtained by stacking word vectors and concatenating positional information.

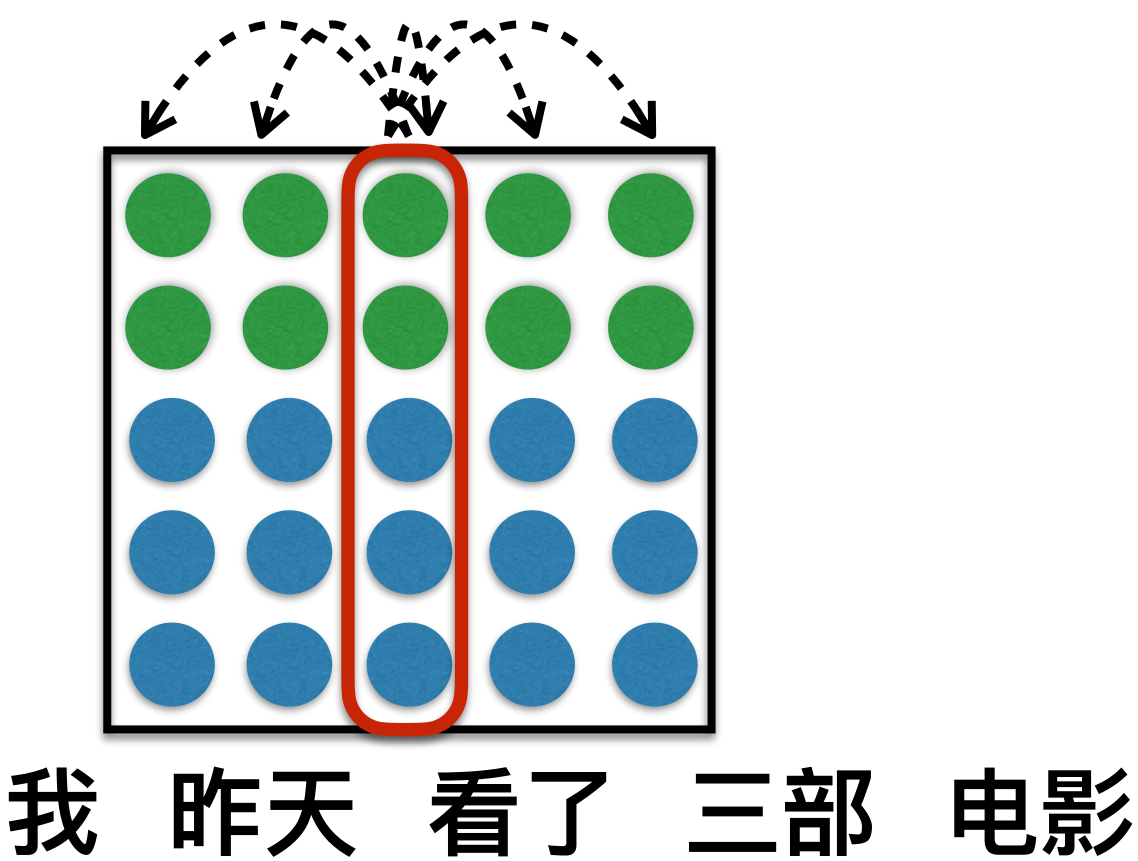


我 昨天 看了 三部 电影

$\mathbf{X} \in \mathbb{R}^{n \times d}$ is obtained by stacking word vectors and concatenating positional information.

Consider just one position. It must decide where else in the sentence to attend (at we permit it to attend to itself, since sometimes there may be no relevant external information.

If we compute the inner product $\mathbf{X}\mathbf{x}_i \in \mathbb{R}^n$, we will get a score for every position, which we can normalize into an attention weighting $\text{softmax}(\mathbf{X}\mathbf{x}_i)$.

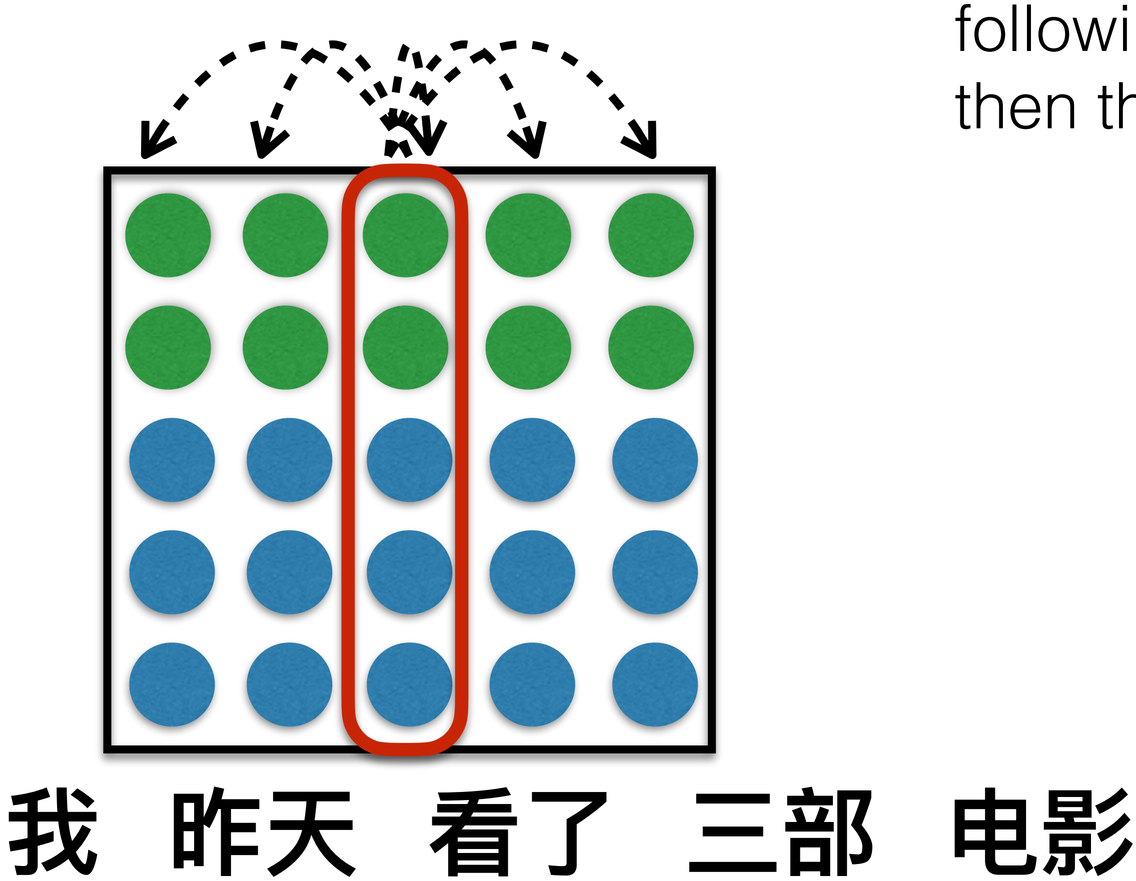


$\mathbf{X} \in \mathbb{R}^{n \times d}$ is obtained by stacking word vectors and concatenating positional information.

Consider just one position. It must decide where else in the sentence to attend (at we permit it to attend to itself, since sometimes there may be no relevant external information.

If we compute the inner product $\mathbf{X}\mathbf{x}_i \in \mathbb{R}^n$, we will get a score for every position, which we can normalize into an attention weighting $\mathrm{softmax}(\mathbf{X}\mathbf{x}_i)$.

We can do this "in parallel" for all positions by doing the following $\mathbf{A} = \mathrm{softmax}(\mathbf{X}\mathbf{X}^\top)$ which is in $[0,1]^{n \times n}$. And then the "output" is $\mathbf{Y} = \mathbf{A}\mathbf{X}$.



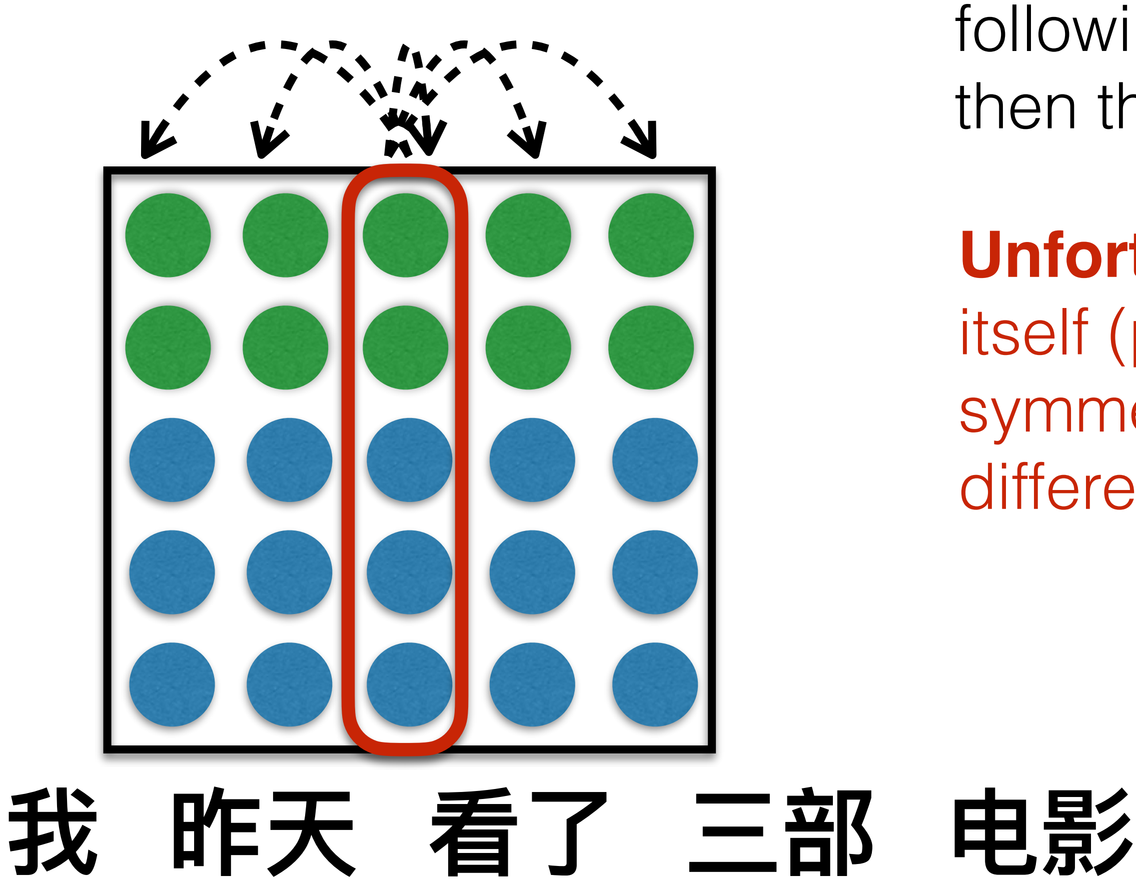
$\mathbf{X} \in \mathbb{R}^{n \times d}$ is obtained by stacking word vectors and concatenating positional information.

Consider just one position. It must decide where else in the sentence to attend (at we permit it to attend to itself, since sometimes there may be no relevant external information.

If we compute the inner product $\mathbf{X}\mathbf{x}_i \in \mathbb{R}^n$, we will get a score for every position, which we can normalize into an attention weighting $\mathrm{softmax}(\mathbf{X}\mathbf{x}_i)$.

We can do this "in parallel" for all positions by doing the following $\mathbf{A} = \mathrm{softmax}(\mathbf{X}\mathbf{X}^\top)$ which is in $[0,1]^{n \times n}$. And then the "output" is $\mathbf{Y} = \mathbf{A}\mathbf{X}$.

Unfortunately: each word will always want to attend to itself (property of inner products), attention will be symmetric (we don't want this), and we can't attend to different kinds of information.

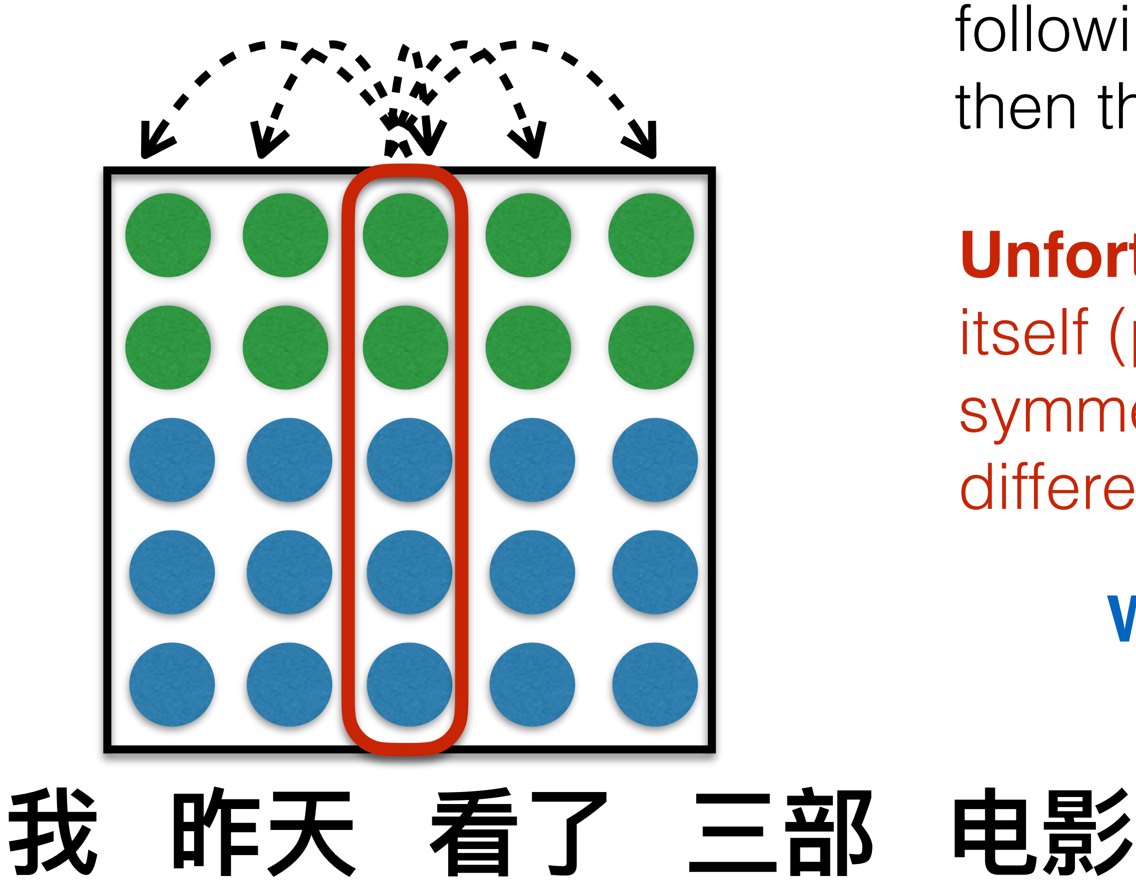


$\mathbf{X} \in \mathbb{R}^{n \times d}$ is obtained by stacking word vectors and concatenating positional information.

Consider just one position. It must decide where else in the sentence to attend (at we permit it to attend to itself, since sometimes there may be no relevant external information.

If we compute the inner product $\mathbf{X}\mathbf{x}_i \in \mathbb{R}^n$, we will get a score for every position, which we can normalize into an attention weighting $\mathrm{softmax}(\mathbf{X}\mathbf{x}_i)$.

We can do this "in parallel" for all positions by doing the following $\mathbf{A} = \mathrm{softmax}(\mathbf{X}\mathbf{X}^\top)$ which is in $[0, 1]^{n \times n}$. And then the "output" is $\mathbf{Y} = \mathbf{A}\mathbf{X}$.



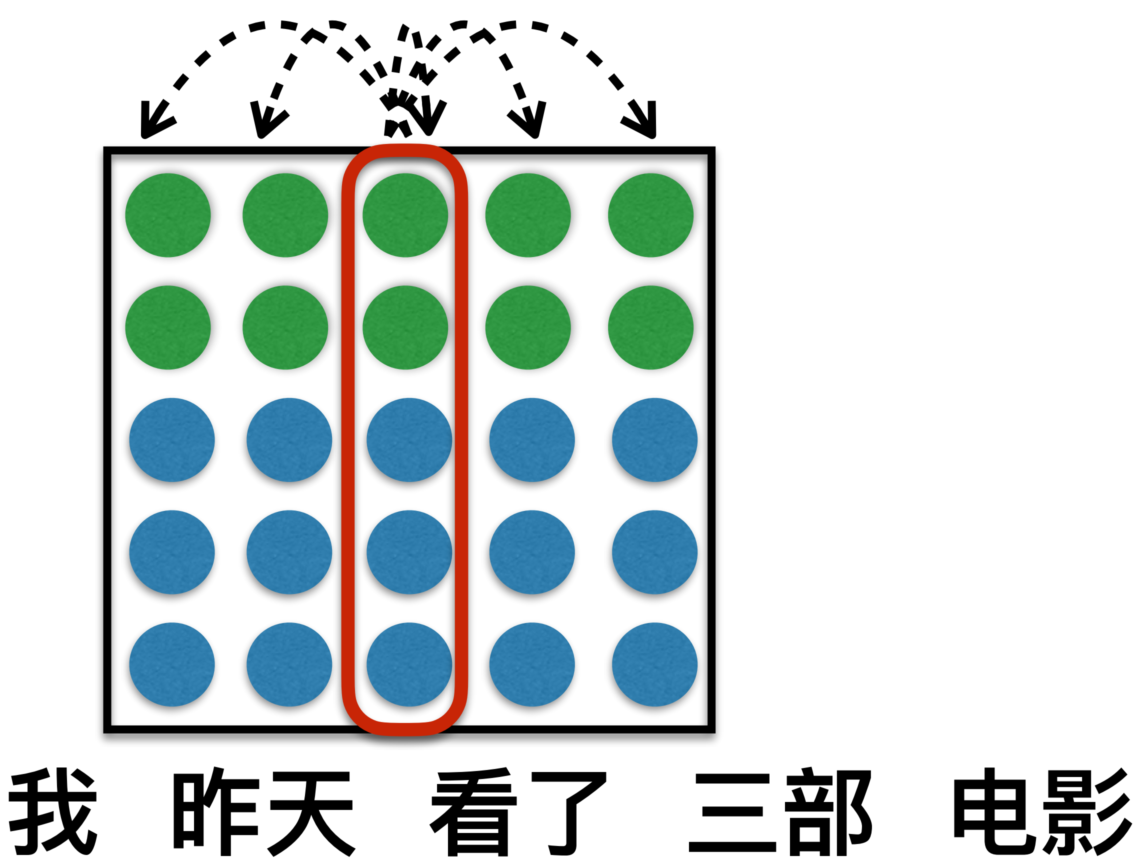
Unfortunately: each word will always want to attend to itself (property of inner products), attention will be symmetric (we don't want this), and we can't attend to different kinds of information.

We need some parameters!

$\mathbf{X} \in \mathbb{R}^{n \times d}$ is obtained by stacking word vectors and concatenating positional information.

Another attempt: Let's add a parameter $\mathbf{W} \in \mathbb{R}^{d \times d}$, now we can compute $\mathbf{X}\mathbf{W}\mathbf{x}_i \in \mathbb{R}^n$. This lets us control where we look, and attention is not necessarily symmetric.

Moreover, we can still do things very efficiently with by computing $\mathrm{softmax}(\mathbf{X}\mathbf{W}\mathbf{X}^\top)$.

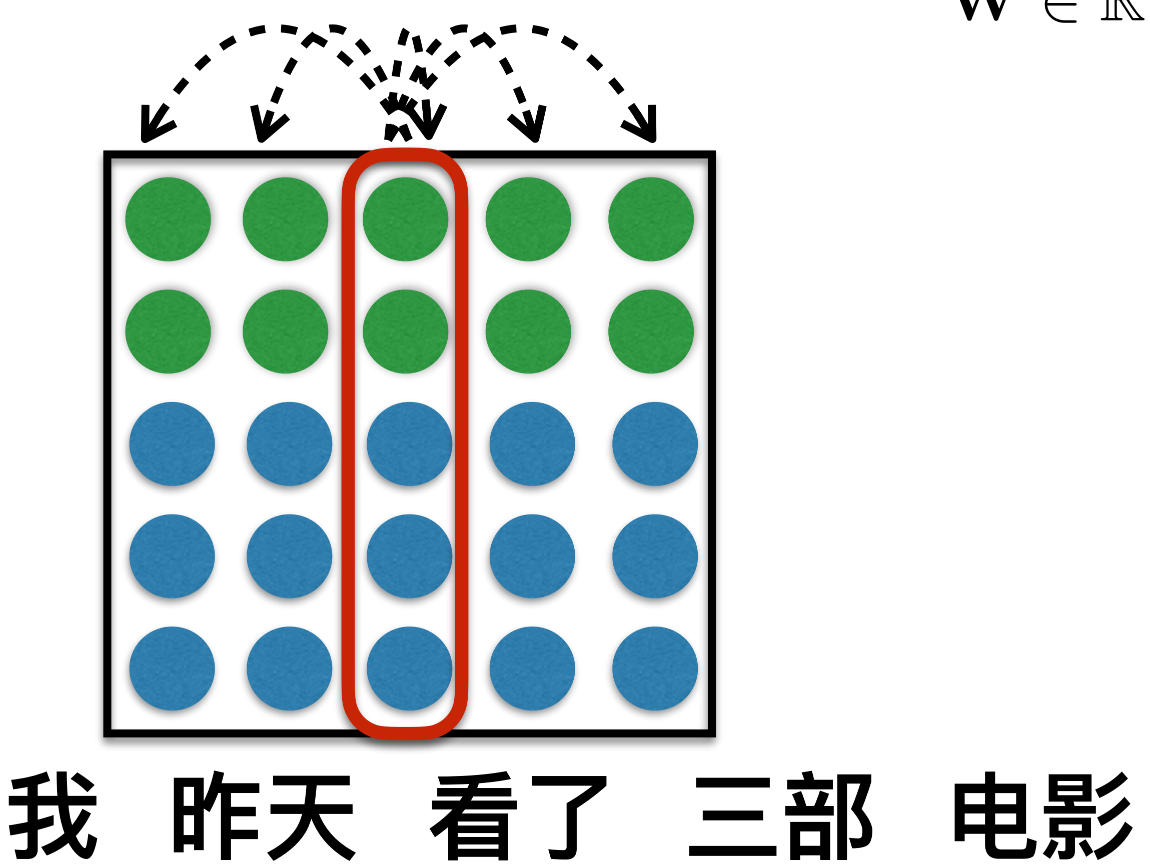


$\mathbf{X} \in \mathbb{R}^{n \times d}$ is obtained by stacking word vectors and concatenating positional information.

Another attempt: Let's add a parameter $\mathbf{W} \in \mathbb{R}^{d \times d}$, now we can compute $\mathbf{X}\mathbf{W}\mathbf{x}_i \in \mathbb{R}^n$. This lets us control where we look, and attention is not necessarily symmetric.

Moreover, we can still do things very efficiently with by computing $\mathrm{softmax}(\mathbf{X}\mathbf{W}\mathbf{X}^\top)$.

To attend to different kinds of attention, we can just add multiple **W**'s, or equivalently, redefine $\mathbf{W} \in \mathbb{R}^{h \times d \times d}$ and use batched matrix multiplies.



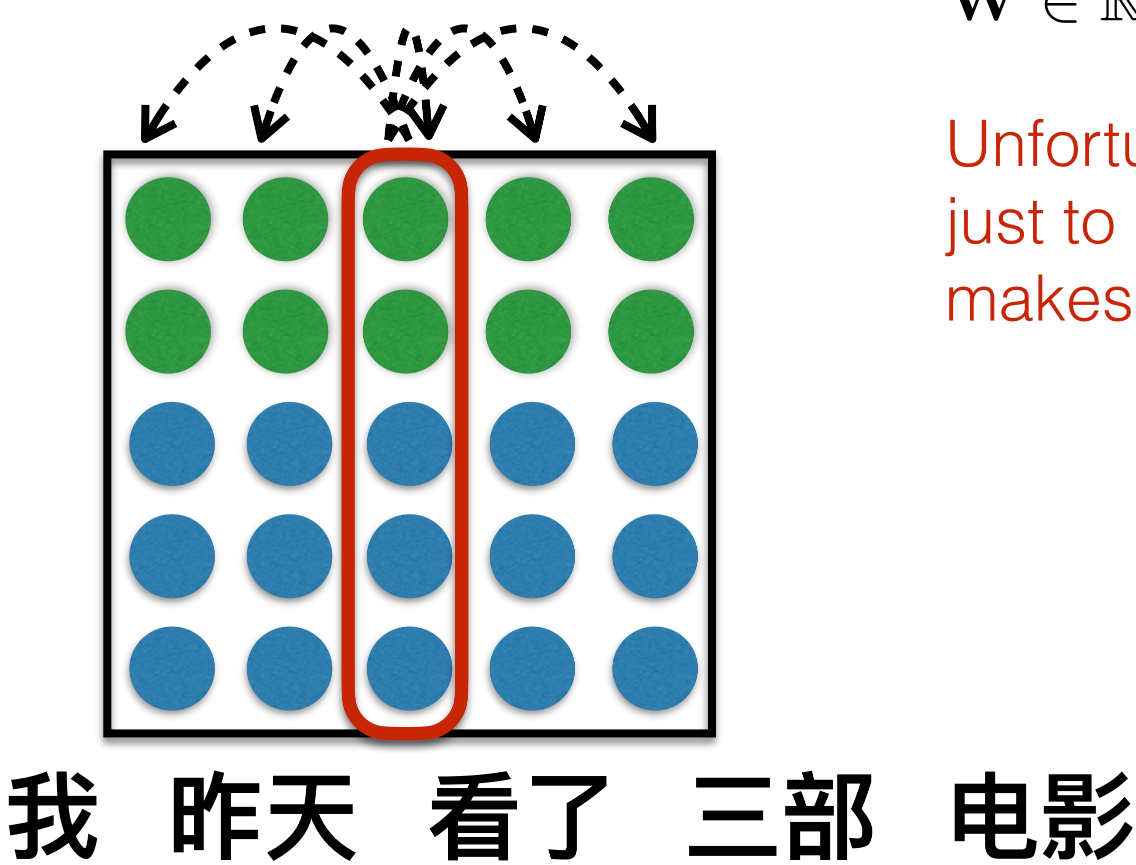
$\mathbf{X} \in \mathbb{R}^{n \times d}$ is obtained by stacking word vectors and concatenating positional information.

Another attempt: Let's add a parameter $\mathbf{W} \in \mathbb{R}^{d \times d}$, now we can compute $\mathbf{X}\mathbf{W}\mathbf{x}_i \in \mathbb{R}^n$. This lets us control where we look, and attention is not necessarily symmetric.

Moreover, we can still do things very efficiently with by computing $\mathrm{softmax}(\mathbf{X}\mathbf{W}\mathbf{X}^\top)$.

To attend to different kinds of attention, we can just add multiple **W**'s, or equivalently, redefine $\mathbf{W} \in \mathbb{R}^{h \times d \times d}$ and use batched matrix multiplies.

Unfortunately: **W** has massive number of parameters, just to decide where to attend to. This is slow and makes learning hard.

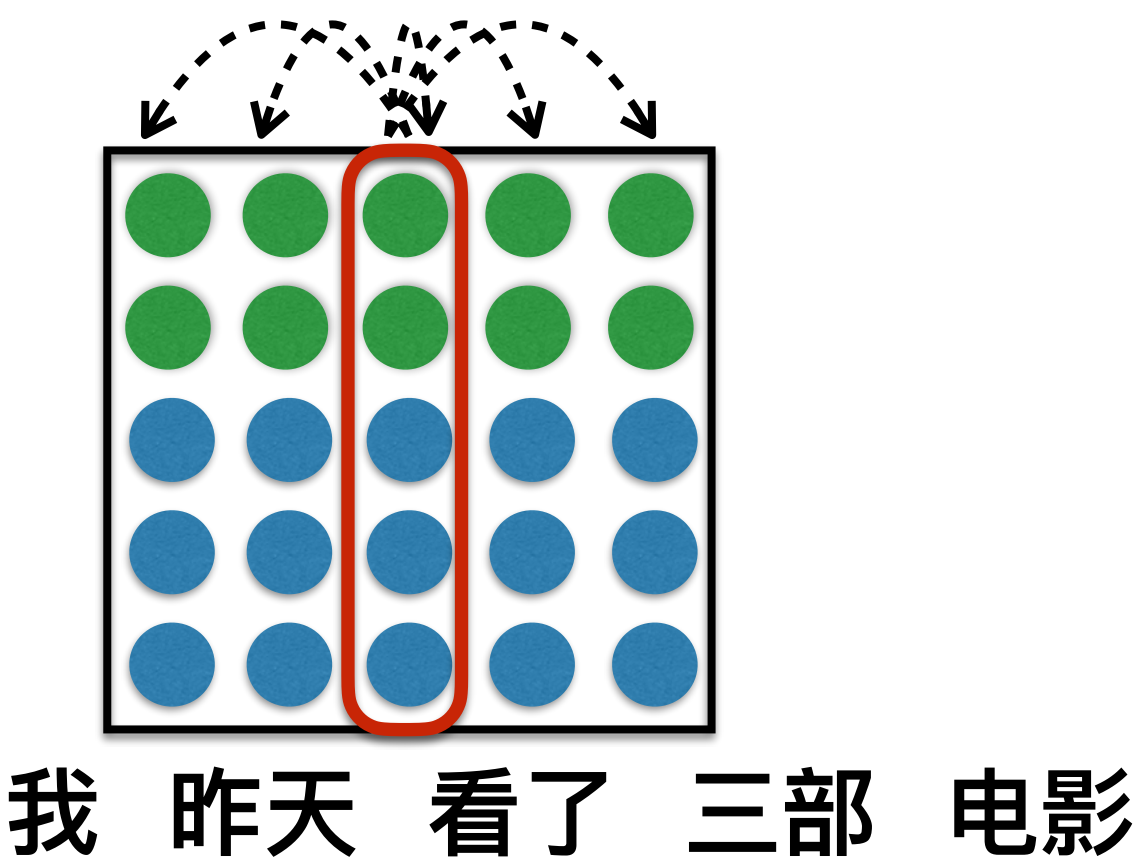


$\mathbf{X} \in \mathbb{R}^{n \times d}$ is obtained by stacking word vectors and concatenating positional information.

Another attempt: Let's use a **low rank** approximation of **W**. We define two matrices $\mathbf{L} \in \mathbb{R}^{d \times \ell}$ and $\mathbf{R} \in \mathbb{R}^{\ell \times d}$ and then do $\mathbf{A} = \mathrm{softmax}(\mathbf{XLRX}^\top)$.

Now we can control the number of parameters in the model by setting ℓ to be as small as we like! In practice, it's common to use $\ell = d/h$.

So we can write $\mathbf{Y} = \mathrm{softmax}(\mathbf{XLRX}^\top)\mathbf{X}$.



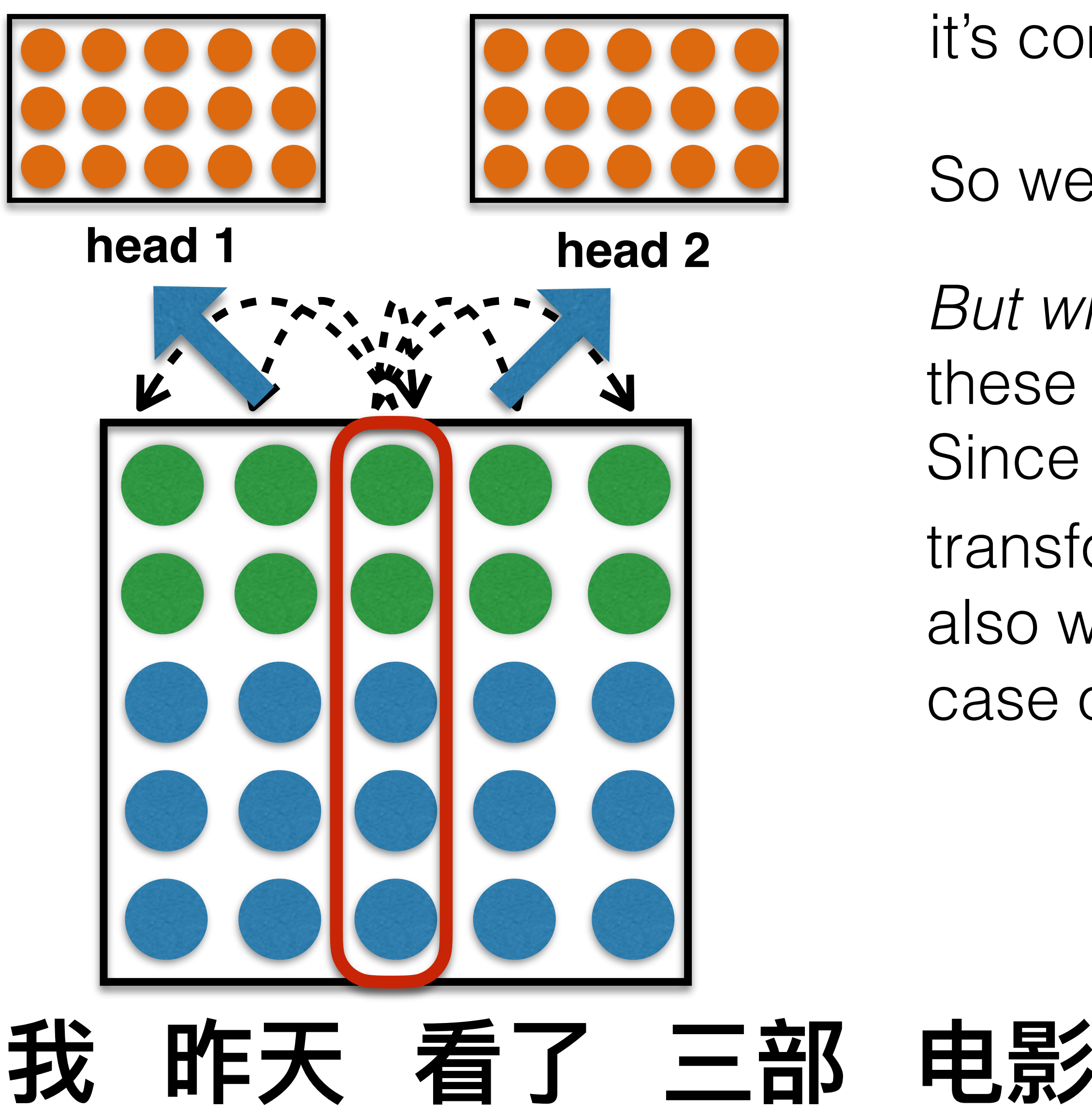
$\mathbf{X} \in \mathbb{R}^{n \times d}$ is obtained by stacking word vectors and concatenating positional information.

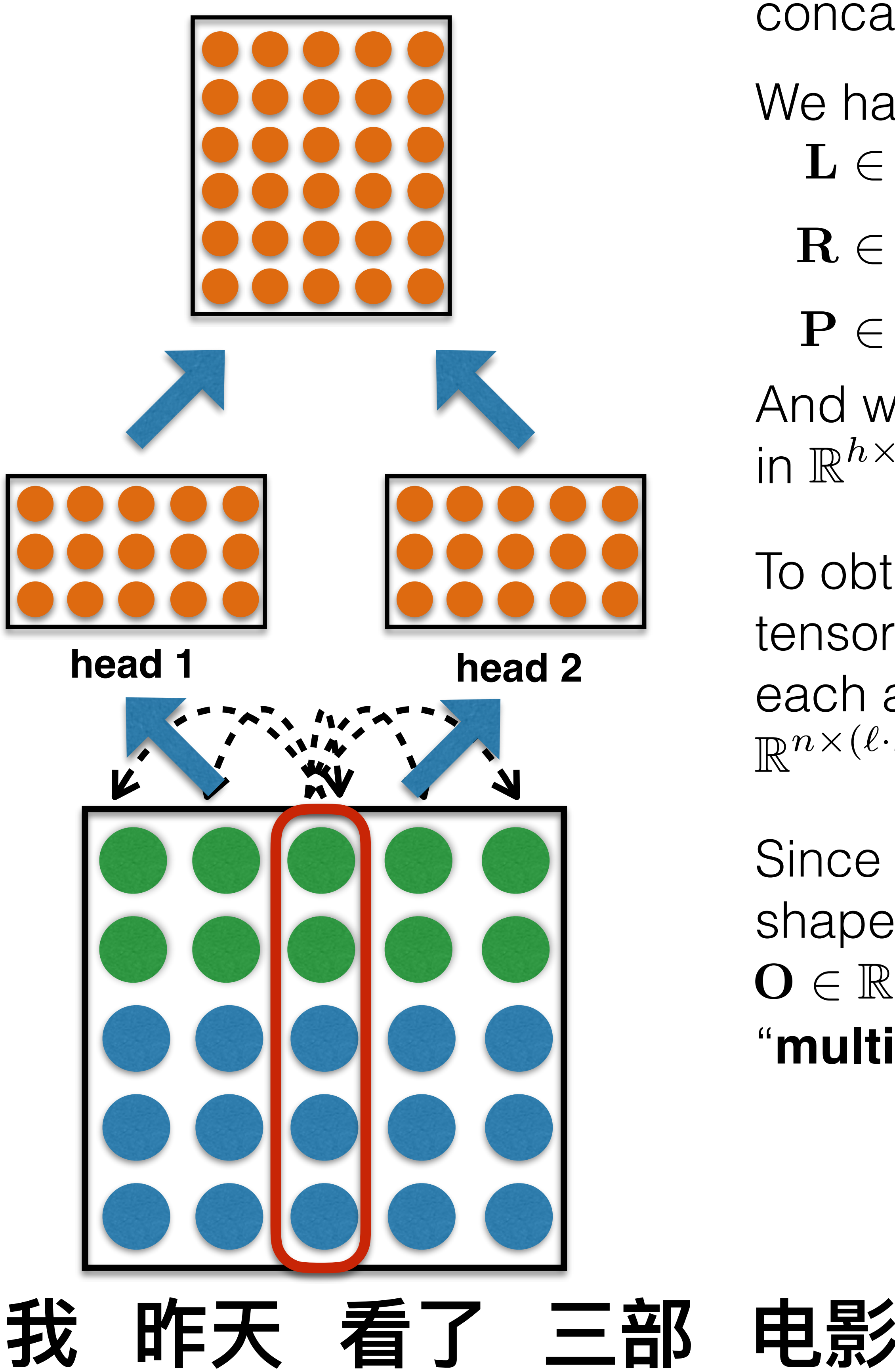
Another attempt: Let's use a **low rank** approximation of **W**. We define two matrices $\mathbf{L} \in \mathbb{R}^{d \times \ell}$ and $\mathbf{R} \in \mathbb{R}^{\ell \times d}$ and then do $\mathbf{A} = \mathrm{softmax}(\mathbf{XLRX}^\top)$.

Now we can control the number of parameters in the model by setting ℓ to be as small as we like! In practice, it's common to use $\ell = d/h$.

So we can write $\mathbf{Y} = \mathrm{softmax}(\mathbf{XLRX}^\top)\mathbf{X}$.

But what about multiple heads? We would like each of these to extract different information from different places. Since we want to extract different information, we need to transform **X**: $\mathbf{Y} = \mathrm{softmax}(\mathbf{XLRX}^\top)\mathbf{X}\mathbf{P}$ where we also want **P** to be low rank: $\mathbf{P} \in \mathbb{R}^{d \times \ell}$, or rather, in the case of multiple heads, $\mathbf{P} \in \mathbb{R}^{h \times d \times \ell}$.





$\mathbf{X} \in \mathbb{R}^{n \times d}$ is obtained by stacking word vectors and concatenating positional information.

We have auxiliary parameters:

$\mathbf{L} \in \mathbb{R}^{h \times d \times \ell}$

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

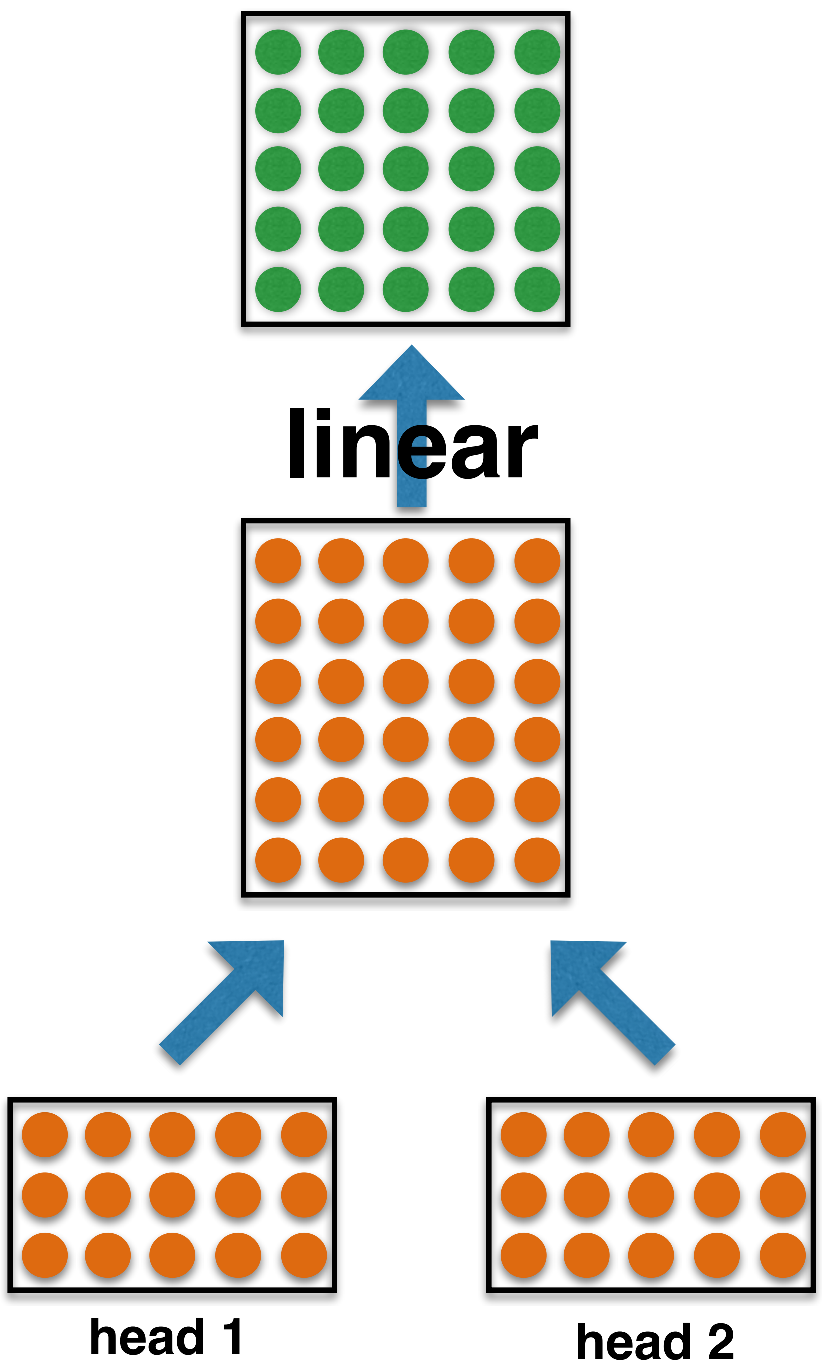
$\mathbf{P} \in \mathbb{R}^{h \times d \times \ell}$

And we compute $\mathbf{Z} = \text{softmax}(\mathbf{XLRX}^\top)\mathbf{XP}$ which is in $\mathbb{R}^{h \times n \times \ell}$.

To obtain one vector per position, we rearrange this tensor so that all ℓ-length representations for each each are adjacent; i.e., the reshaped matrix is in $\mathbb{R}^{n \times (\ell \cdot h)}$.

Since we would like the final output to have the same shape as the input, we use a final linear projection, $\mathbf{O} \in \mathbb{R}^{(\ell \cdot h) \times d}$, the conclude what the authors call "**multiheaded attention**":

$$\begin{aligned} \mathbf{Y} &= \text{reshape}(\mathbf{Z})\mathbf{O} \\ &= \text{reshape}(\text{softmax}(\mathbf{XLRX}^\top)\mathbf{XP})\mathbf{O} \end{aligned}$$



$\mathbf{X} \in \mathbb{R}^{n \times d}$ is obtained by stacking word vectors and concatenating positional information.

We have auxiliary parameters:

$\mathbf{L} \in \mathbb{R}^{h \times d \times \ell}$

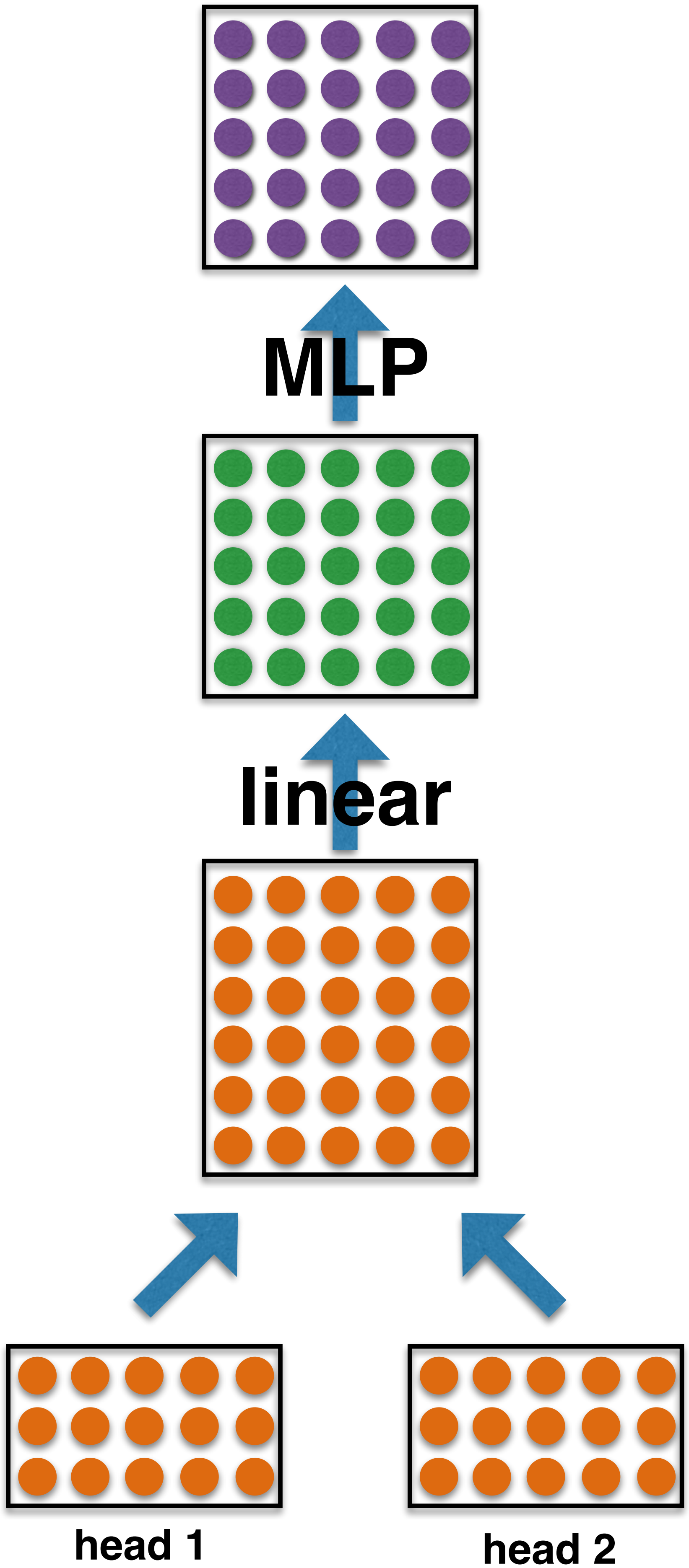
$\mathbf{R} \in \mathbb{R}^{h \times \ell \times d}$

$\mathbf{P} \in \mathbb{R}^{h \times d \times \ell}$

$\mathbf{O} \in \mathbb{R}^{(\ell \cdot h) \times d}$

And we compute $\mathbf{Y} = \text{reshape}(\text{softmax}(\mathbf{XLRX}^\top)\mathbf{XP})\mathbf{O}$, which is in $\mathbb{R}^{n \times d}$.

Happily, these operations exploit the very efficient batched matrix multiply operations (fast on your GPUs).



$\mathbf{X} \in \mathbb{R}^{n \times d}$ is obtained by stacking word vectors and concatenating positional information.

We have auxiliary parameters:

$$\mathbf{L} \in \mathbb{R}^{h \times d \times \ell}$$

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

$$\mathbf{P} \in \mathbb{R}^{h \times d \times \ell}$$

$$\mathbf{O} \in \mathbb{R}^{(\ell \cdot h) \times d}$$

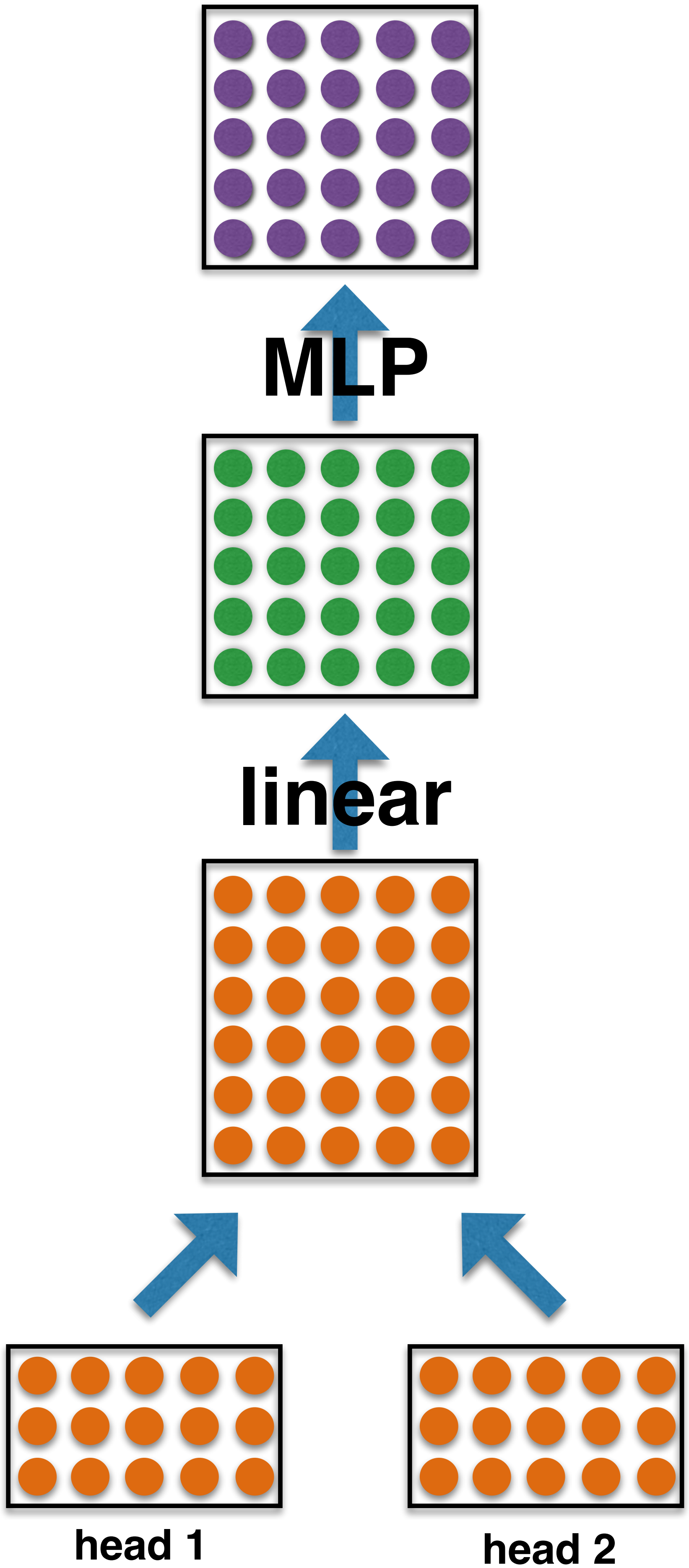
And we compute $\mathbf{Y} = \mathrm{reshape}(\mathrm{softmax}(\mathbf{XLRX}^\top)\mathbf{XP})\mathbf{O}$, which is in $\mathbb{R}^{n \times d}$.

Happily, these operations exploit the very efficient batched matrix multiply operations (fast on your GPUs).

But we're not done yet. After multi-headed attention, $\mathbf{Y}$ is further transformed by passing each position through an MLP in parallel. Intuitively this let's the model extract conjunctions of features that were integrated via attention.

$$\mathbf{F} = \mathrm{relu}(\mathbf{YW} + \mathbf{b})\mathbf{V} + \mathbf{c}$$

where $\mathbf{W} \in \mathbb{R}^{d \times k}$ and *k* is “large” (eg 4 x *d*).



$\mathbf{X} \in \mathbb{R}^{n \times d}$ is obtained by stacking word vectors and concatenating positional information.

We have auxiliary parameters:

$\mathbf{L} \in \mathbb{R}^{h \times d \times \ell}$ $\quad \mathbf{W} \in \mathbb{R}^{d \times k}$

$\mathbf{R} \in \mathbb{R}^{h \times \ell \times d}$ $\quad \mathbf{V} \in \mathbb{R}^{k \times d}$

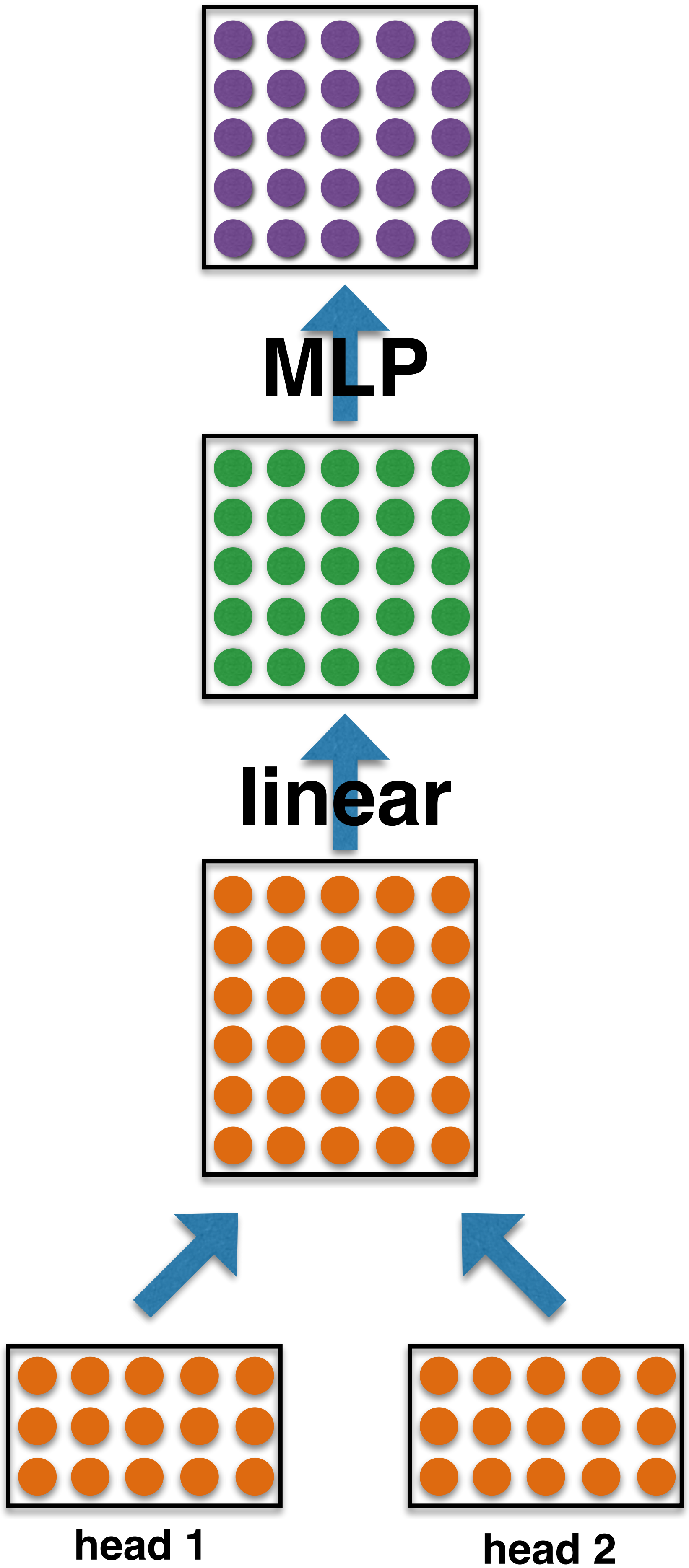
$\mathbf{P} \in \mathbb{R}^{h \times d \times \ell}$

$\mathbf{O} \in \mathbb{R}^{(\ell \cdot h) \times d}$

And we compute

$$\mathbf{Y} = \text{reshape}(\text{softmax}(\mathbf{XLRX}^\top)\mathbf{XP})\mathbf{O} + \mathbf{X}$$

$$\mathbf{F} = \text{relu}(\mathbf{YW} + \mathbf{b})\mathbf{V} + \mathbf{c} + \mathbf{Y}$$



$\mathbf{X} \in \mathbb{R}^{n \times d}$ is obtained by stacking word vectors and concatenating positional information.

We have auxiliary parameters:

$\mathbf{L} \in \mathbb{R}^{h \times d \times \ell}$ $\quad$ $\mathbf{W} \in \mathbb{R}^{d \times k}$

$\mathbf{R} \in \mathbb{R}^{h \times \ell \times d}$ $\quad$ $\mathbf{V} \in \mathbb{R}^{k \times d}$

$\mathbf{P} \in \mathbb{R}^{h \times d \times \ell}$

$\mathbf{O} \in \mathbb{R}^{(\ell \cdot h) \times d}$

And we compute

$$\mathbf{Y} = \mathrm{reshape}(\mathrm{softmax}(\mathbf{XLRX}^\top)\mathbf{XP})\mathbf{O} + \mathbf{X}$$

$$\mathbf{F} = \mathrm{relu}(\mathbf{YW} + \mathbf{b})\mathbf{V} + \mathbf{c} + \mathbf{Y}$$

Some final details:

- residual connections make deeper deeper networks easier to learn
- layer normalization is used, which rescales and shifts **Y** and **F**.
- to enable propagation of information over multiple hops, and to learn more complex interactions, we stack many of these layers on top of each other

Transformer encoders

- We have now built an encoder that uses attention to compute representations of words-in-context
- We could replace the bidirectional encoder used in the previous section with this
- *We now turn to how to build a “decoder” out of transformer components*

Transformer **decoders**

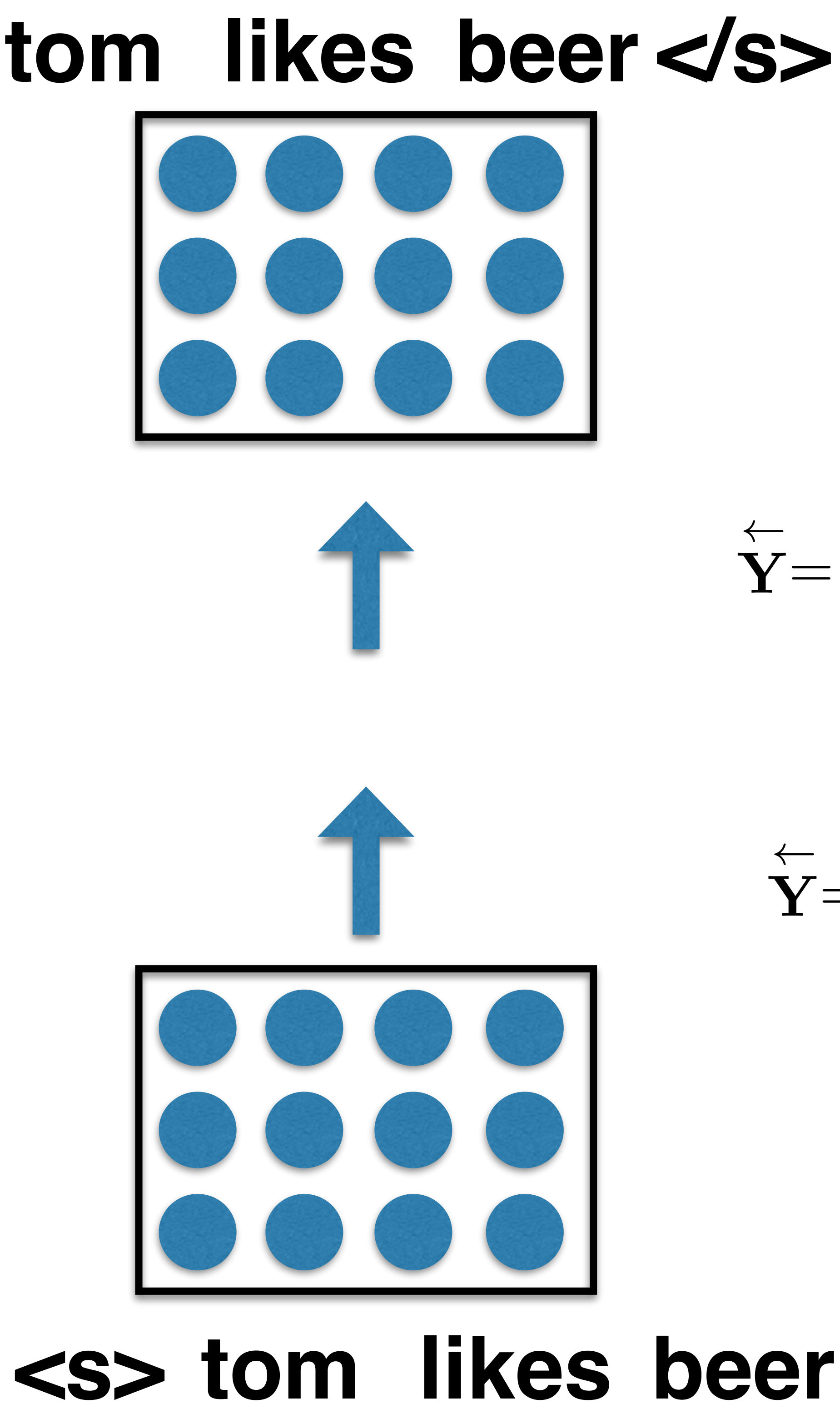
- Transformers can attend forwards and backward
  - This is what makes them powerful, but a language model can't look into the future for words that haven't been generated (at training time it could, but it wouldn't help you at test time)
  - Trick: we will manipulate the attention so that a word can only look to its left. This is a very simple tweak to the model:

$$\mathbf{Y} = \mathrm{reshape}(\mathrm{softmax}(\mathbf{XLRX}^\top)\mathbf{XP})\mathbf{O} + \mathbf{X}$$

$$\overleftarrow{\mathbf{Y}} = \mathrm{reshape}(\mathrm{softmax}(\mathbf{XLRX}^\top + \mathbf{M})\mathbf{XP})\mathbf{O} + \mathbf{X}$$

Here, $\mathbf{M} \in \{-\infty, 0\}^{n \times n}$, such that the pre-softmax attention "scores" are set to -∞ for all attention from position *i* to position *j* where *j* > *i*.

Unconditional LMs

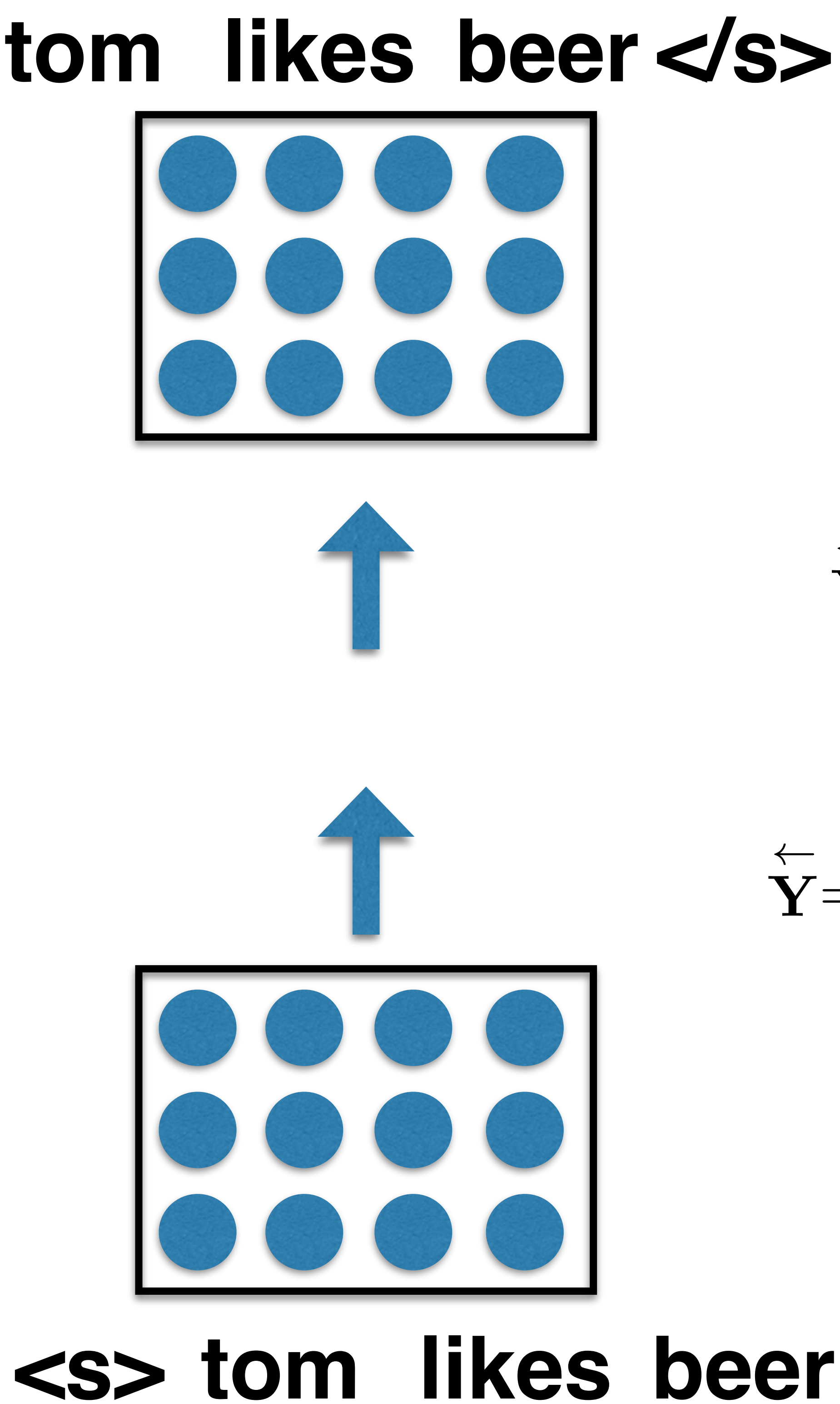


$$\mathbf{Q} = \mathrm{softmax}(\overleftarrow{\mathbf{Y}}\,\mathbf{R})$$

$$\overleftarrow{\mathbf{Y}} = \mathrm{reshape}(\mathrm{softmax}(\overleftarrow{\mathbf{Y}}\,\mathbf{LR}\,\overleftarrow{\mathbf{Y}}^\top + \mathbf{M})\,\overleftarrow{\mathbf{Y}}\,\mathbf{P})\mathbf{O} + \overleftarrow{\mathbf{Y}}$$

$$\overleftarrow{\mathbf{Y}} = \mathrm{reshape}(\mathrm{softmax}(\mathbf{XLRX}^\top + \mathbf{M})\mathbf{XP})\mathbf{O} + \mathbf{X}$$

Conditional LMs



$$\mathbf{Q} = \mathrm{softmax}(\overleftarrow{\mathbf{Y}}\,\mathbf{R})$$

$$\overleftarrow{\mathbf{Y}} = \mathrm{reshape}(\mathrm{softmax}(\overleftarrow{\mathbf{Y}}\,\mathbf{LR}\mathbf{C}^\top)\mathbf{C}\mathbf{P})\mathbf{O} + \overleftarrow{\mathbf{Y}}$$

$$\overleftarrow{\mathbf{Y}} = \mathrm{reshape}(\mathrm{softmax}(\mathbf{XLRX}^\top + \mathbf{M})\mathbf{XP})\mathbf{O} + \mathbf{X}$$

1. Build a representation of the target history
2. Incorporate conditioning context by “attending to” the source context **C**.

Transformer Summary

- Current state of the art
  - Good mix of computationally efficient and a reasonably effective model
- Still many opportunities to improve things!
  - Low-rank approximations are one way to reduce parameters— there are many others.
  - Does every attention head have to sum to 1? Maybe sometimes certain heads should be turned off
  - Should attention be dense? Maybe it should be sparse. Maybe it should correlate with linguistic structure
  - **Your ideas here…**

Questions?

Thanks!

Obrigado!