Better Conditional Language Modeling

Chris Dyer

DeepMind

Carnegie Mellon University

Conditional LMs

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

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

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

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

Kalchbrenner and Blunsom 2013

Encoder

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

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

Recurrent decoder

Recurrent connection

Embedding of w_{t-1}

Source sentence

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

Learnt bias

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

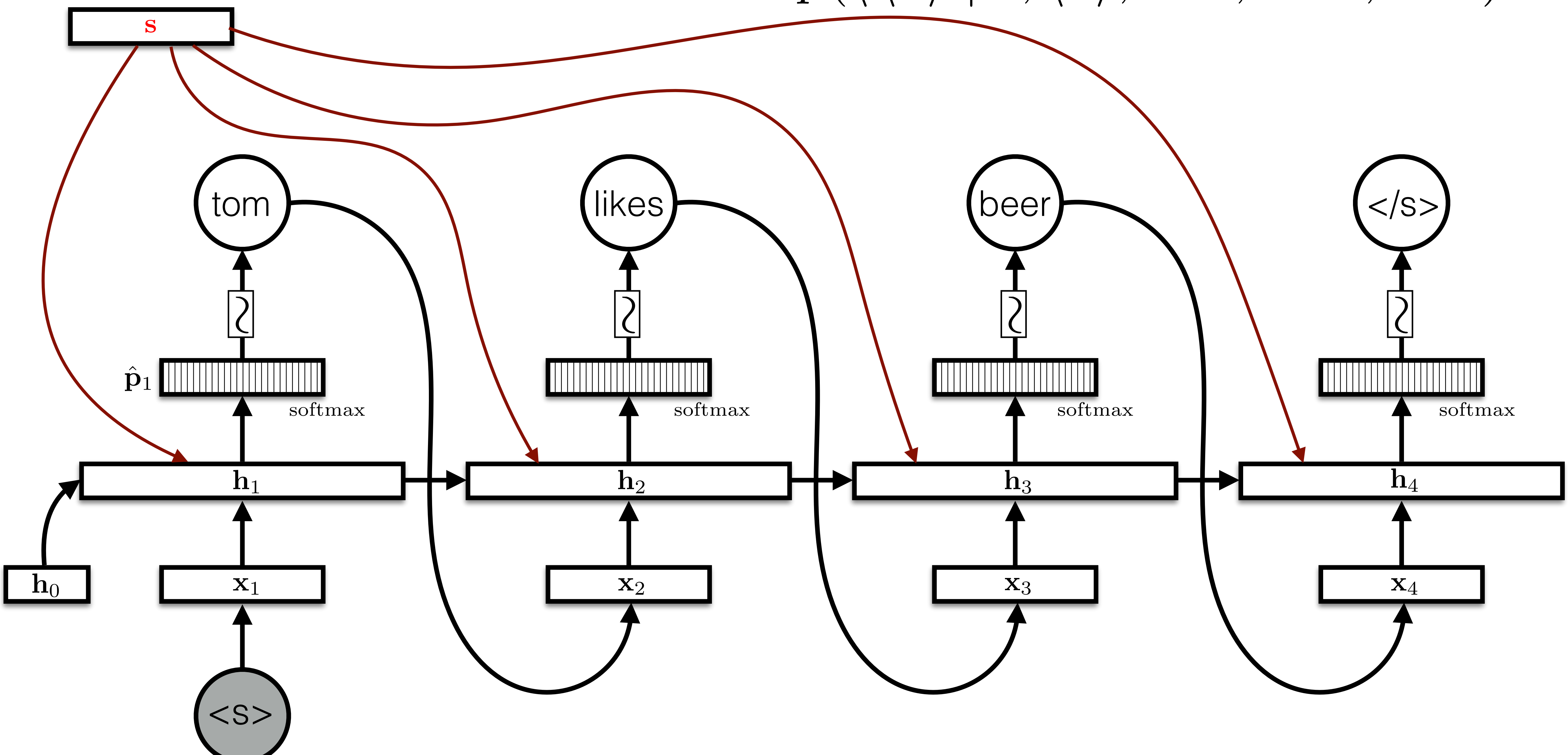
$$p(W_t \mid \boldsymbol{x}, \boldsymbol{w}_{<t}) = \mathrm{softmax}(\mathbf{u}_t)$$

Recall unconditional RNN

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

K&B 2013: RNN Decoder

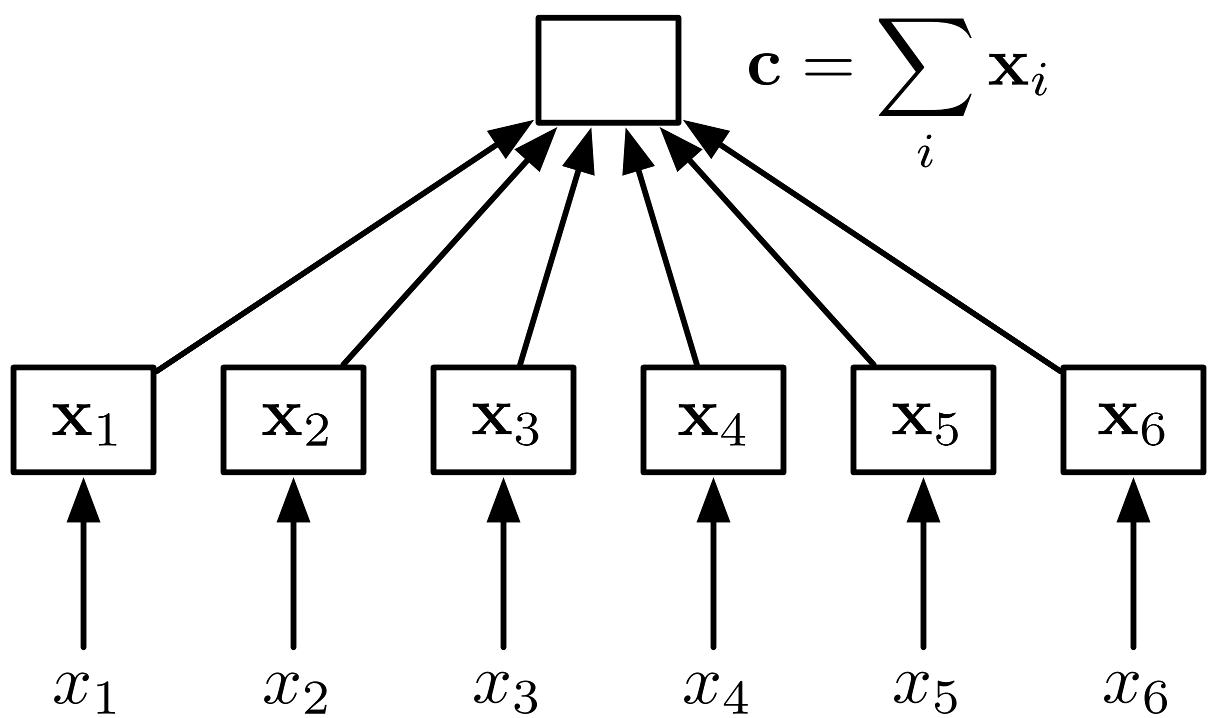
$$p(tom \mid \mathbf{s}, \langle \mathtt{s} \rangle) \times p(likes \mid \mathbf{s}, \langle \mathtt{s} \rangle, tom)$$
$$\times p(beer \mid \mathbf{s}, \langle \mathtt{s} \rangle, tom, likes)$$
$$\times p(\langle \backslash \mathtt{s} \rangle \mid \mathbf{s}, \langle \mathtt{s} \rangle, tom, likes,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



K&B 2013: Encoder

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

The simplest model possible:



K&B 2013: Problems

- The bag of words assumption is really bad (part 1)

  Alice saw Bob.
  Bob saw Alice.

  I would like some fresh bread with aged cheese.
  I would like some aged bread with fresh cheese.

- We are putting a lot of information inside a single vector (part 2)

Sutskever et al. (2014)

LSTM encoder

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

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

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

LSTM decoder

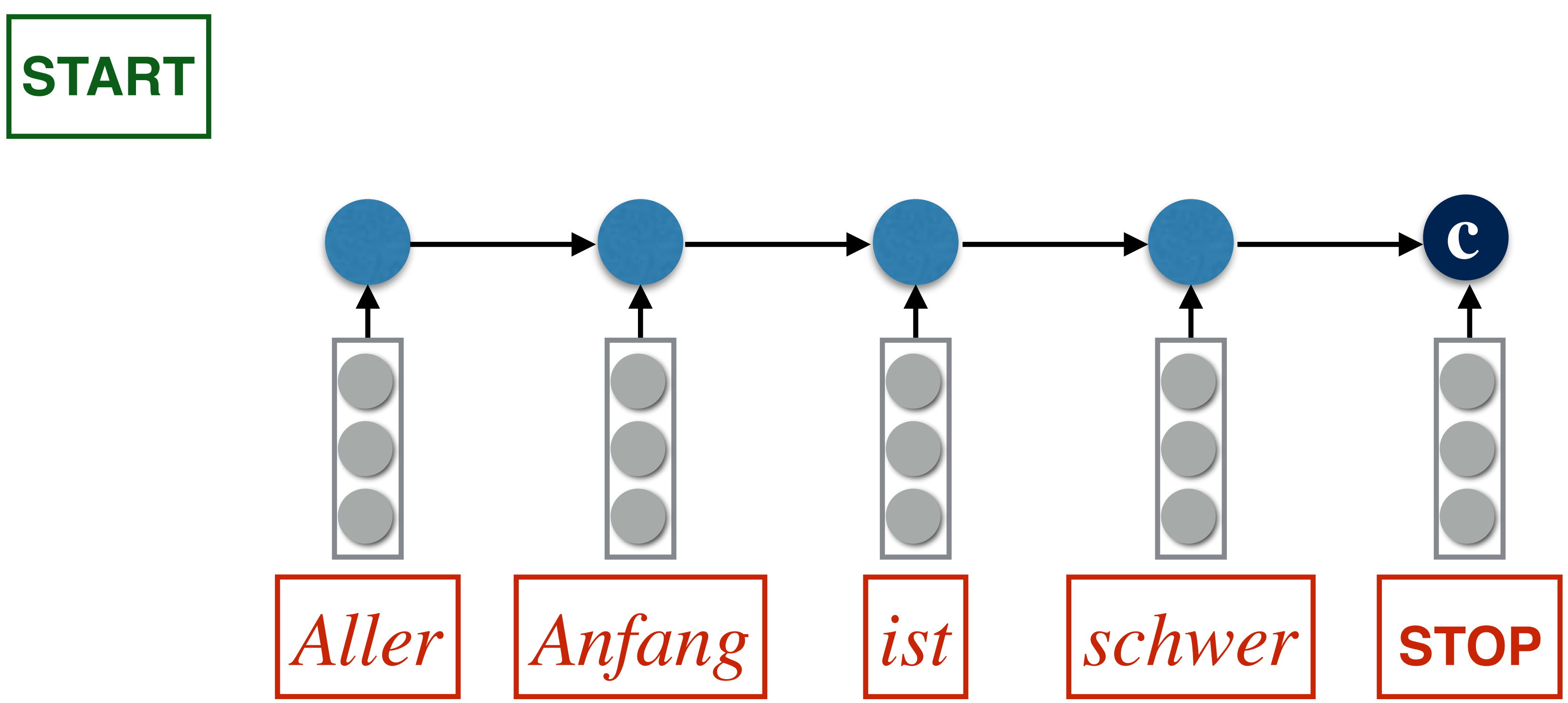
$$w_0 = \langle \mathbf{s} \rangle$$

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

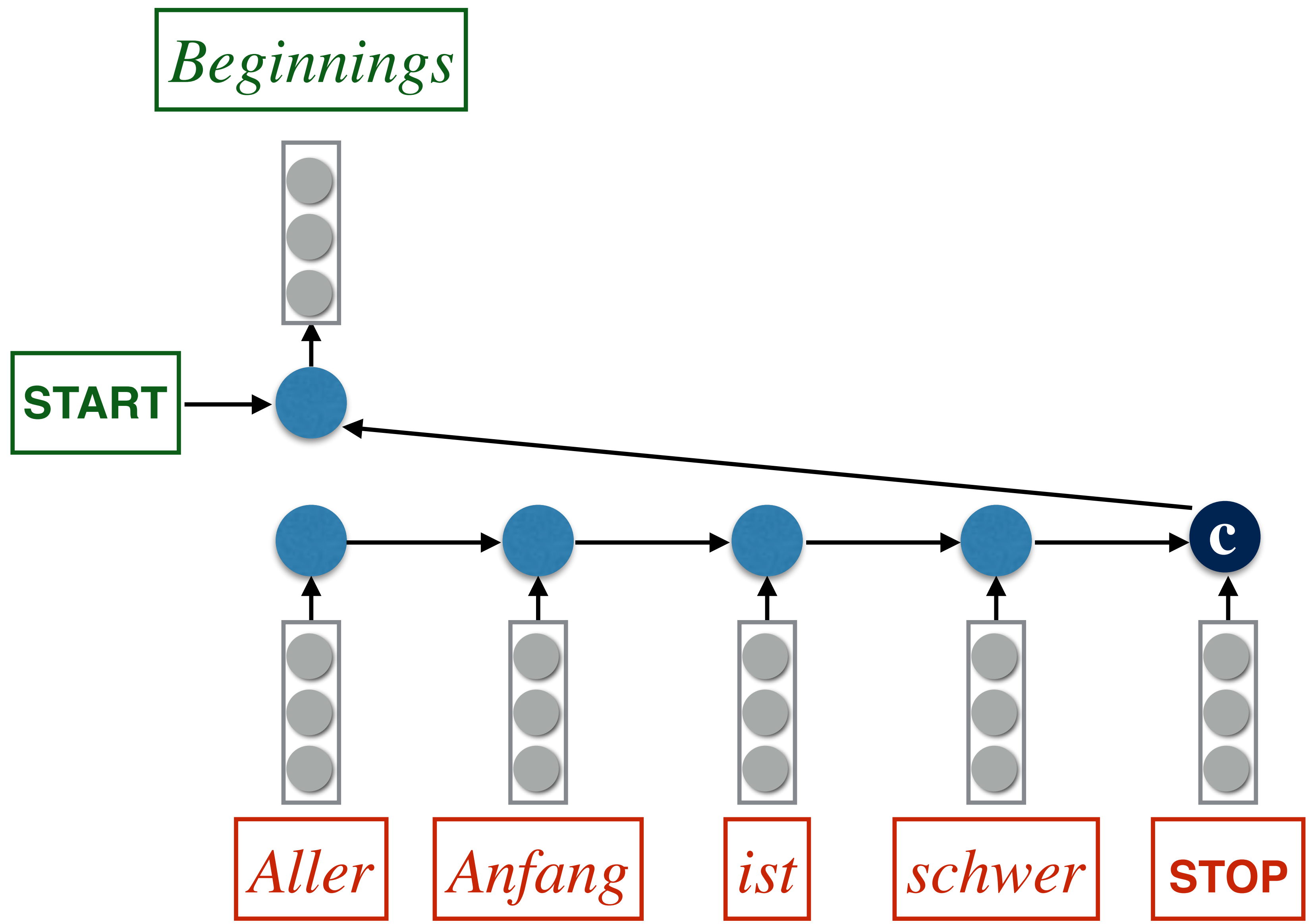
$$\mathbf{u}_t = \mathbf{P}\mathbf{h}_{t+\ell} + \mathbf{b}$$

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

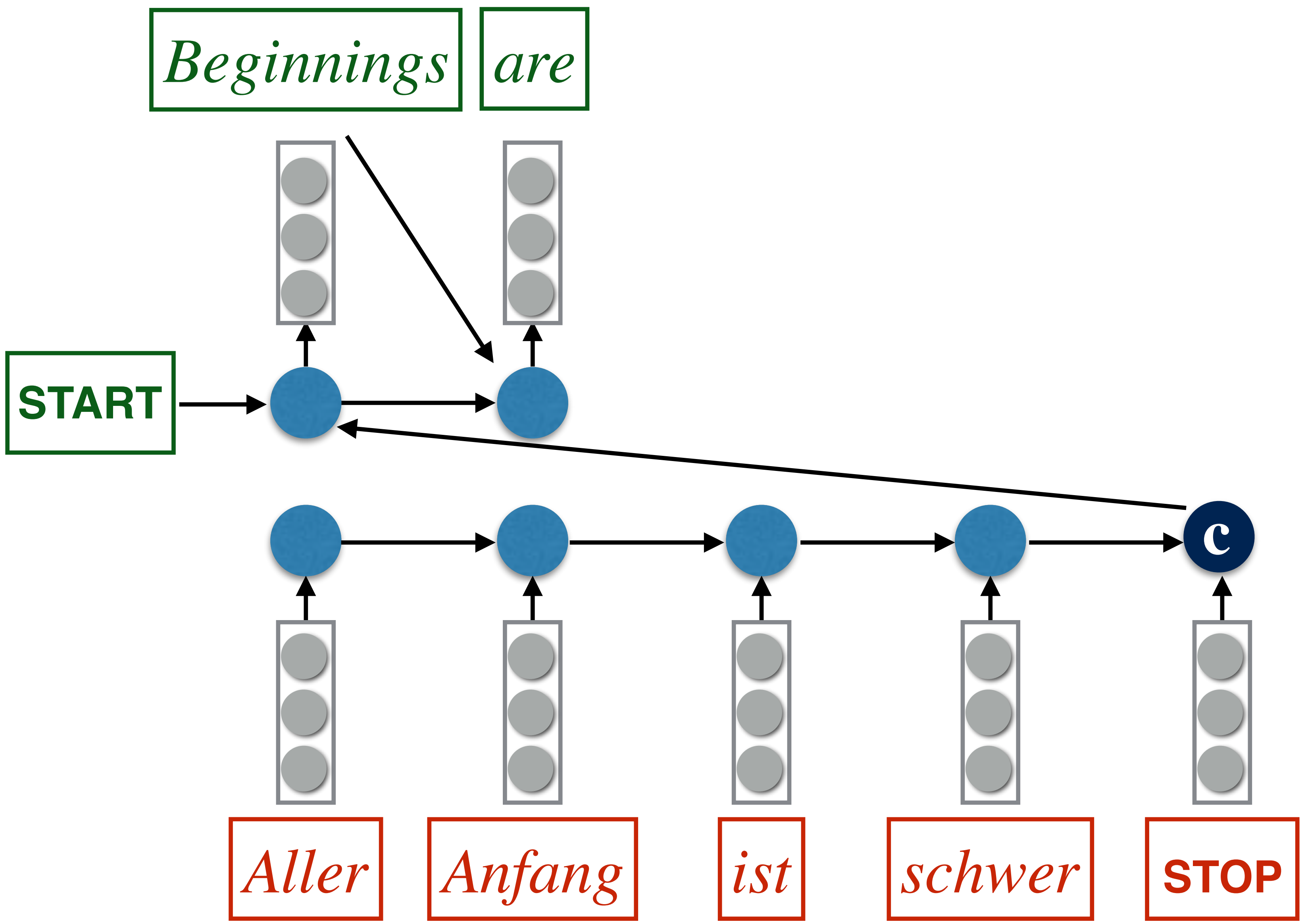
Sutskever et al. (2014)



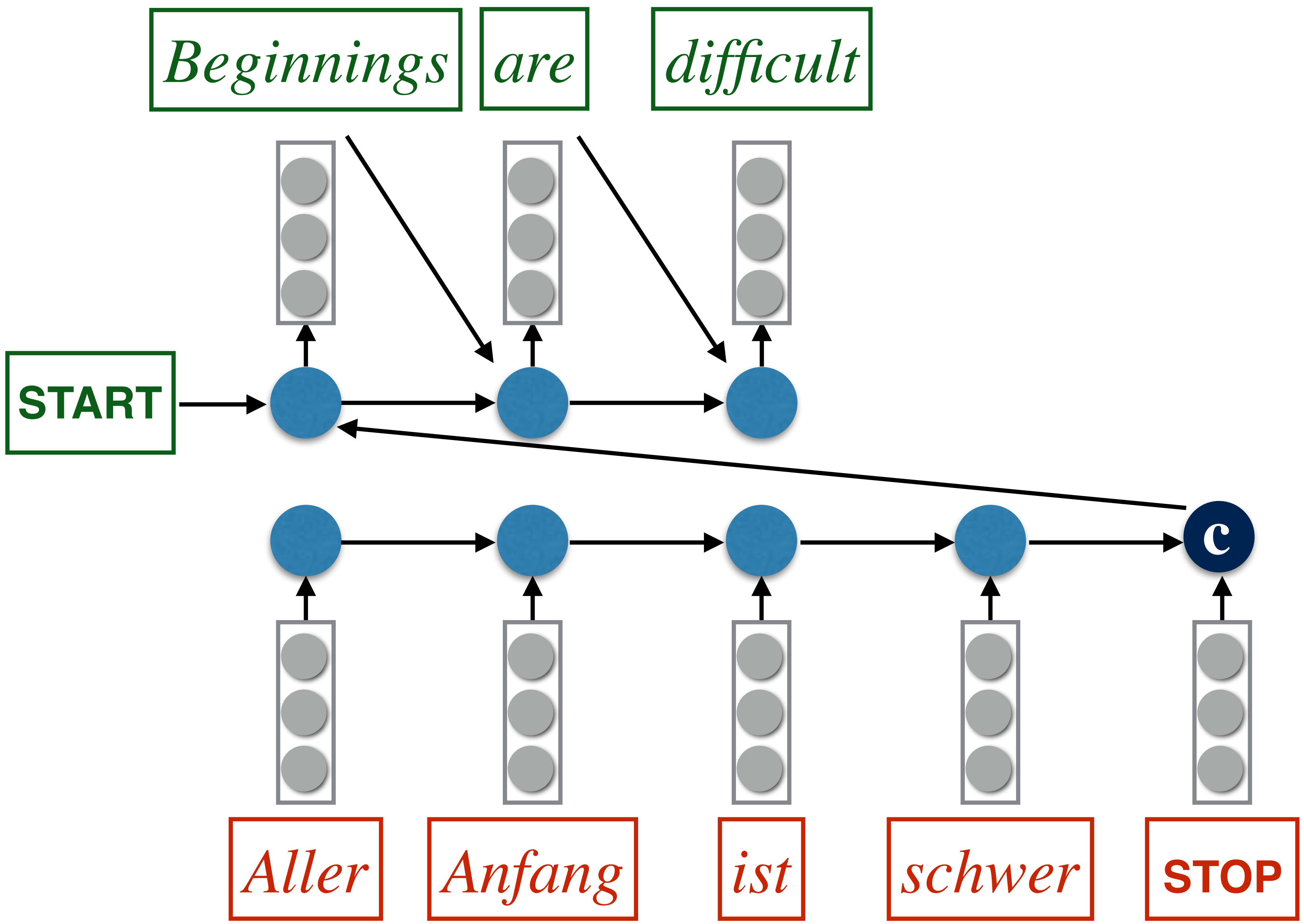
Sutskever et al. (2014)



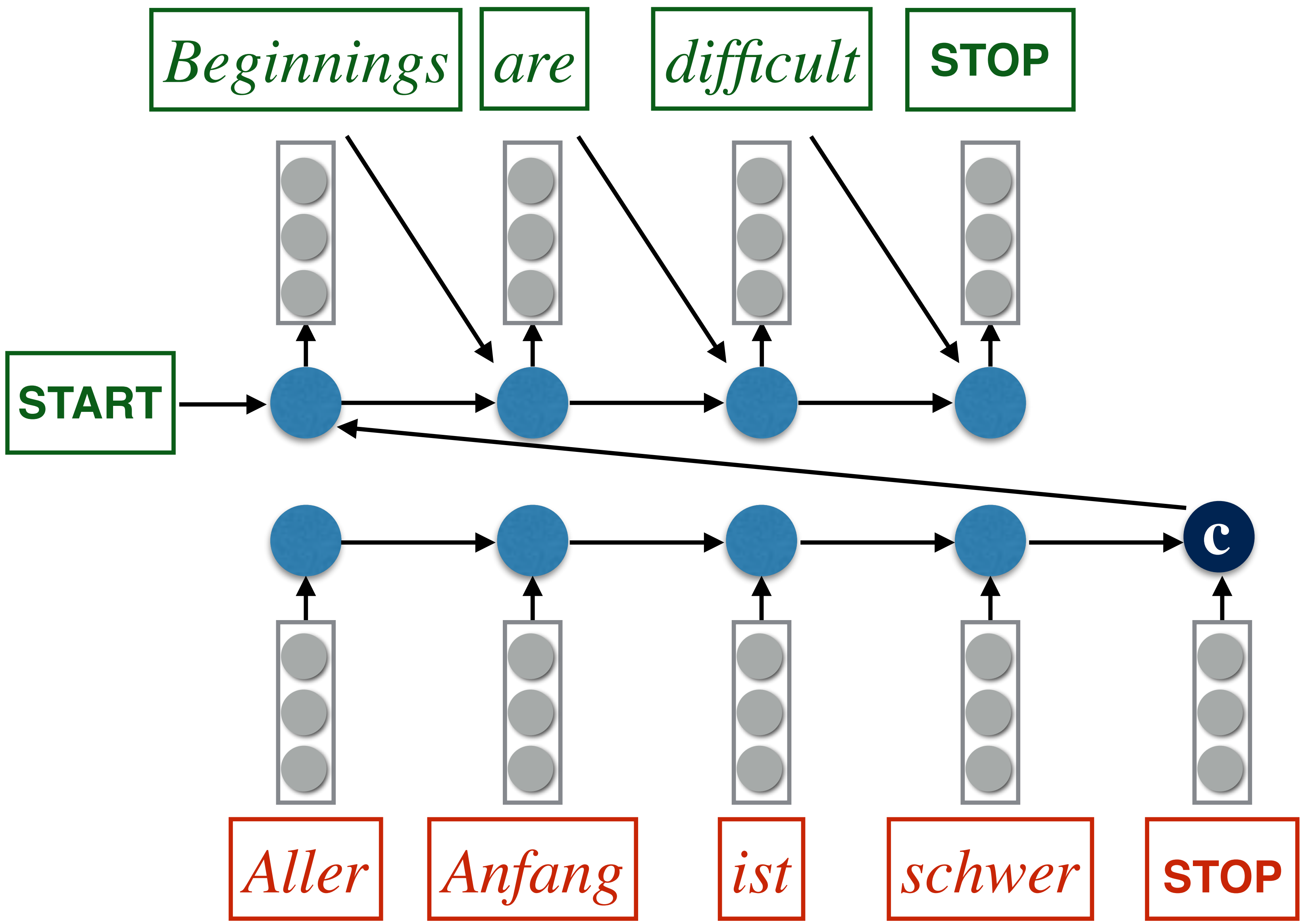
Sutskever et al. (2014)



Sutskever et al. (2014)



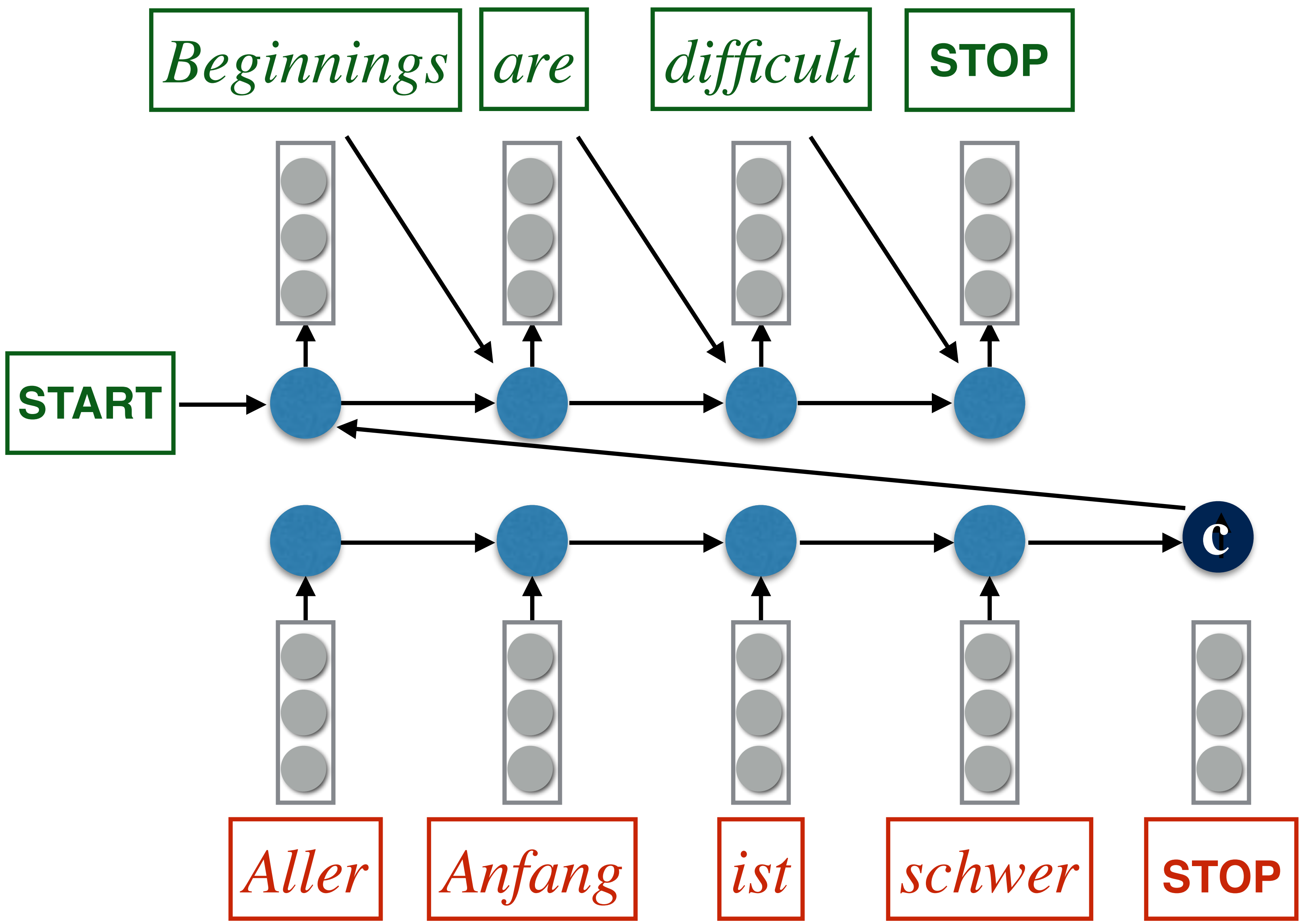
Sutskever et al. (2014)



Sutskever et al. (2014)

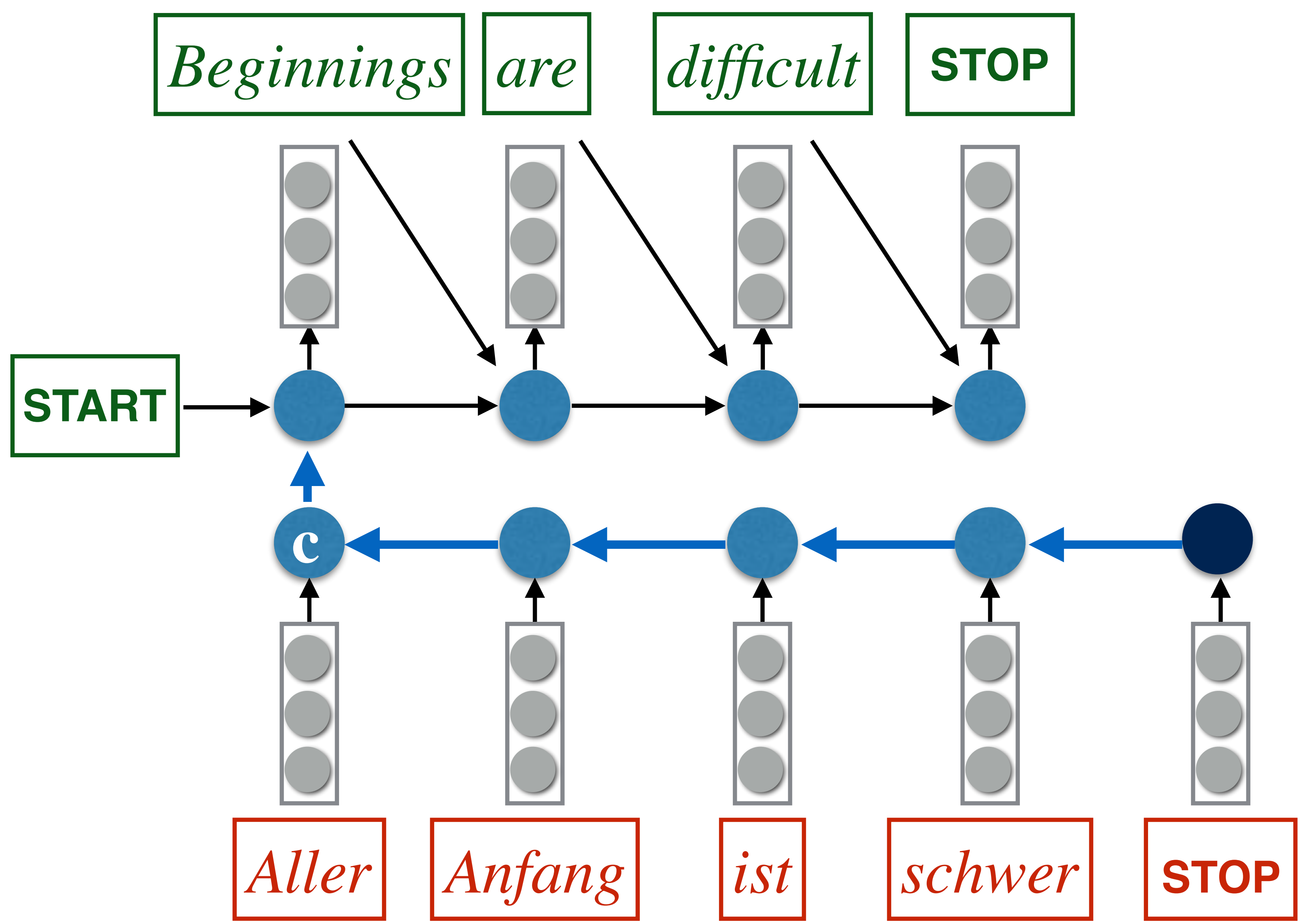
- **Good**
  - RNNs deal naturally with sequences of various lengths
  - LSTMs in principle can propagate gradients a long distance
  - Very simple architecture!
- **Bad**
  - The hidden state has to remember a lot of information!

Sutskever et al. (2014): Tricks



Sutskever et al. (2014): Tricks

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



Sutskever et al. (2014): Tricks

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

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

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

Decoder:

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

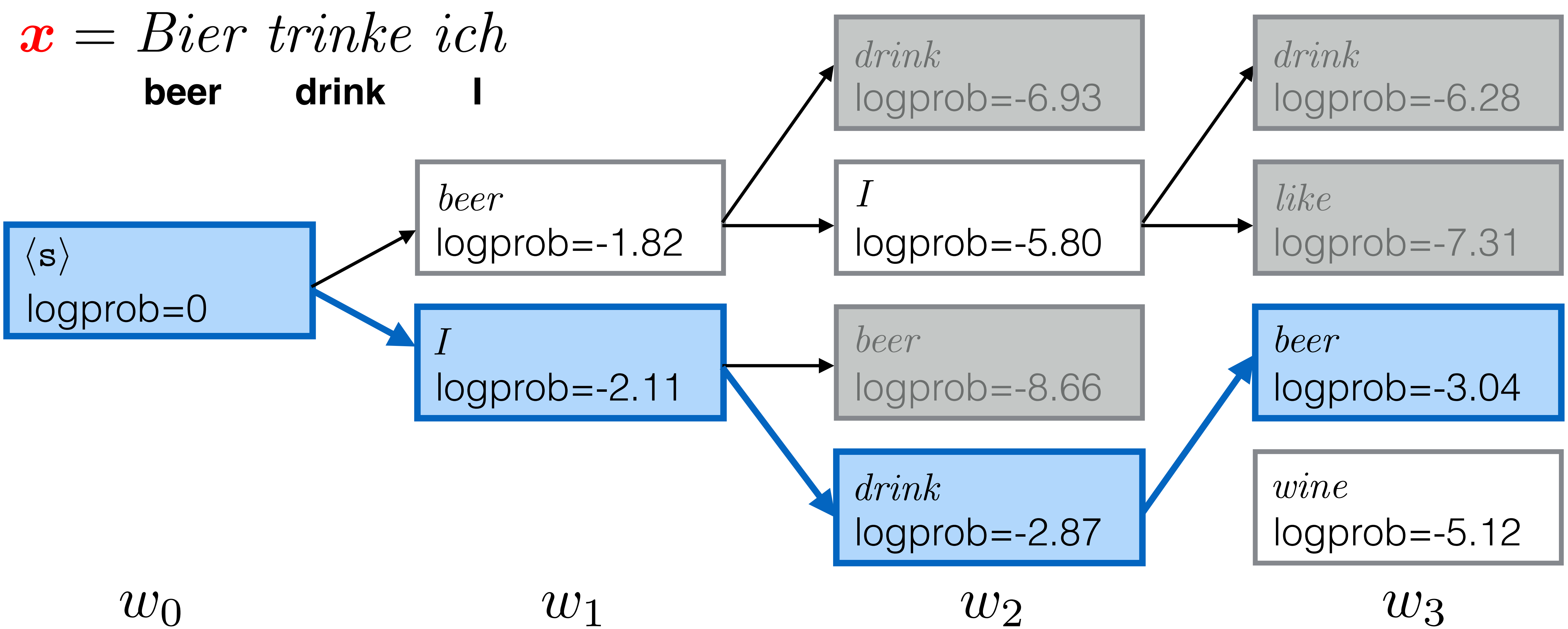
$$\mathbf{u}_t^{(j)} = \mathbf{P}\mathbf{h}_t^{(j)} + \mathbf{b}^{(j)}$$

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

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

Sutskever et al. (2014): Tricks

Use beam search: **+1 BLEU**

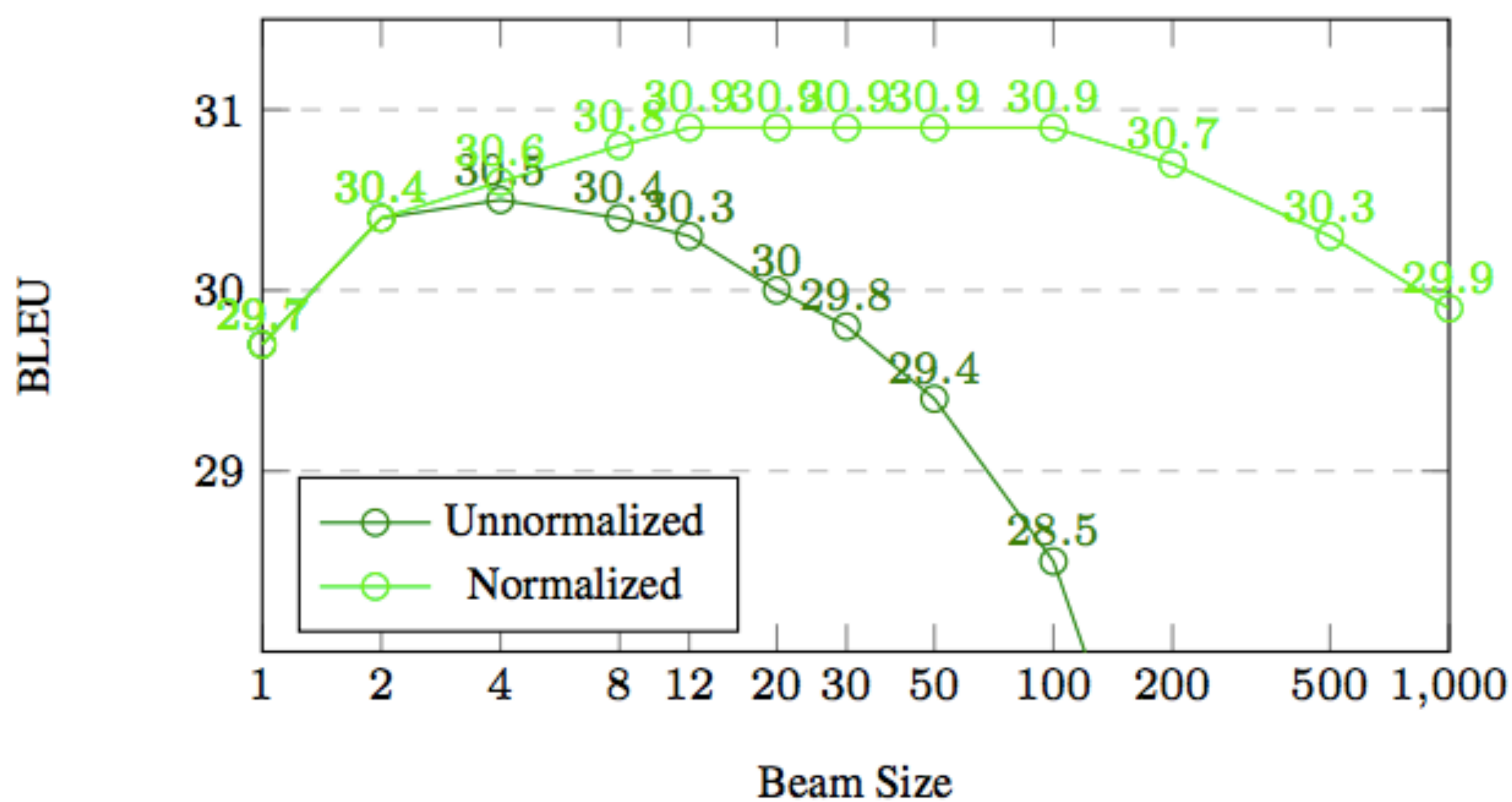


Sutskever et al. (2014): Tricks

Use beam search: **+1 BLEU**

Make the beam really big: **-1 BLEU**

(Koehn and Knowles, 2017)



Conditioning with vectors

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

Conditioning with vectors

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

Prof. Ray Mooney

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

Conditioning with vectors

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

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

Conditioning with vectors

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

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

What is to be done?

Solving the vector bottleneck

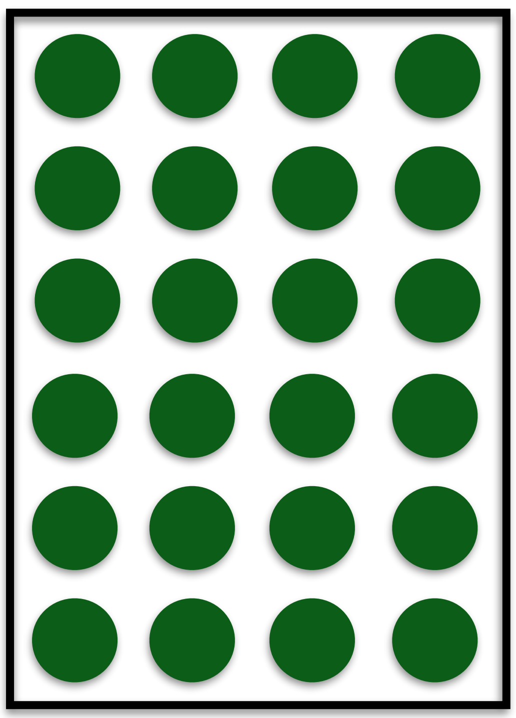
- Represent a source sentence as a matrix
- Generate a target sentence from a matrix

- This will
  - Solve the capacity problem
  - Solve the gradient flow problem

Sentences as ~~vectors~~ matrices

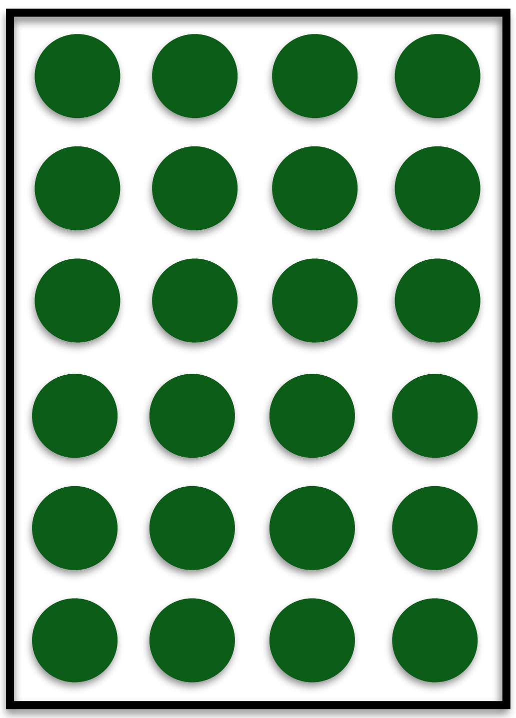
- Problem with the fixed-size vector model
  - Sentences are of different sizes but vectors are of the same size
- Solution: use matrices instead
  - Fixed number of rows, but number of columns depends on the number of words
  - Usually $|\boldsymbol{f}|$ = #cols

Sentences as matrices

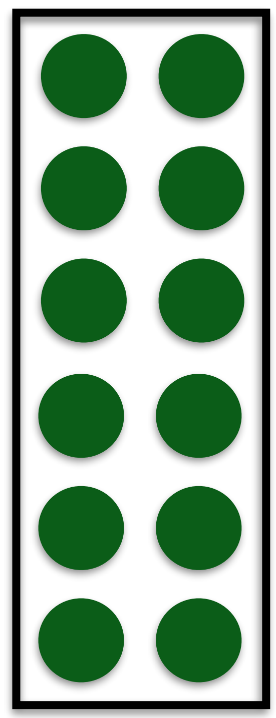


Ich möchte ein Bier

Sentences as matrices

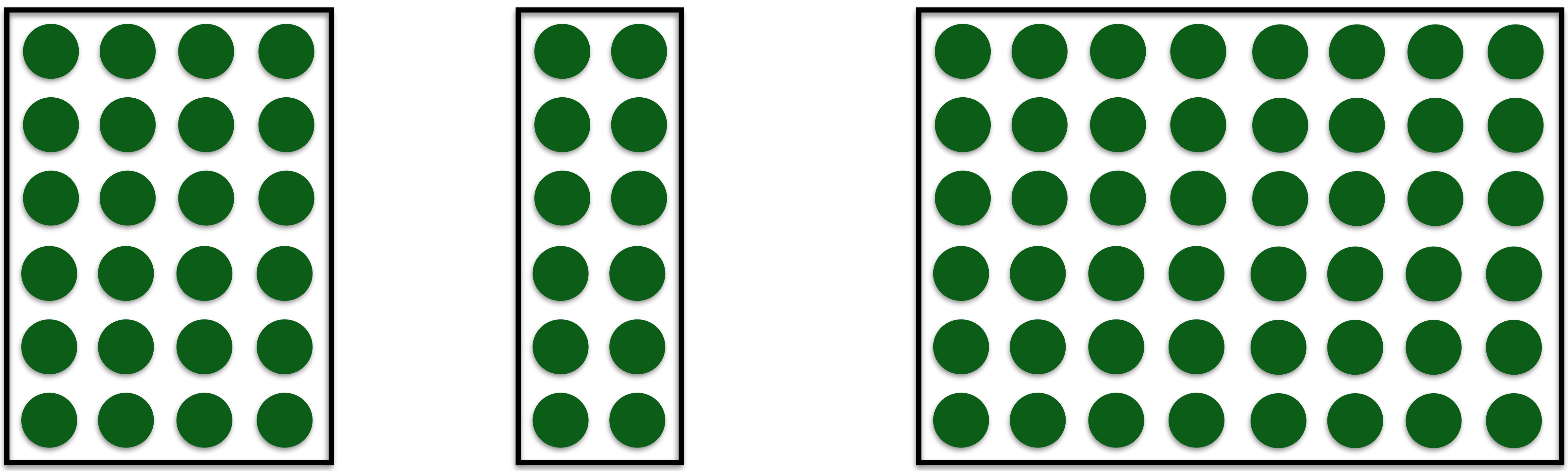


Ich möchte ein Bier



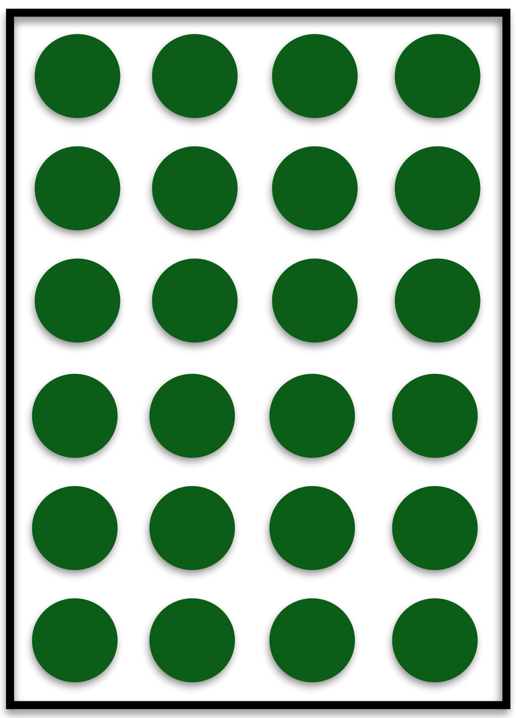
Mach's gut

Sentences as matrices

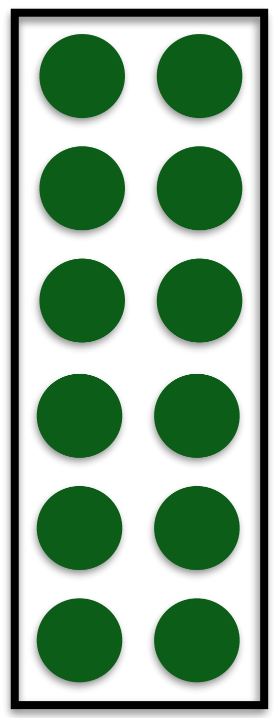


Ich möchte ein Bier *Mach's gut* *Die Wahrheiten der Menschen sind die unwiderlegbaren Irrtümer*

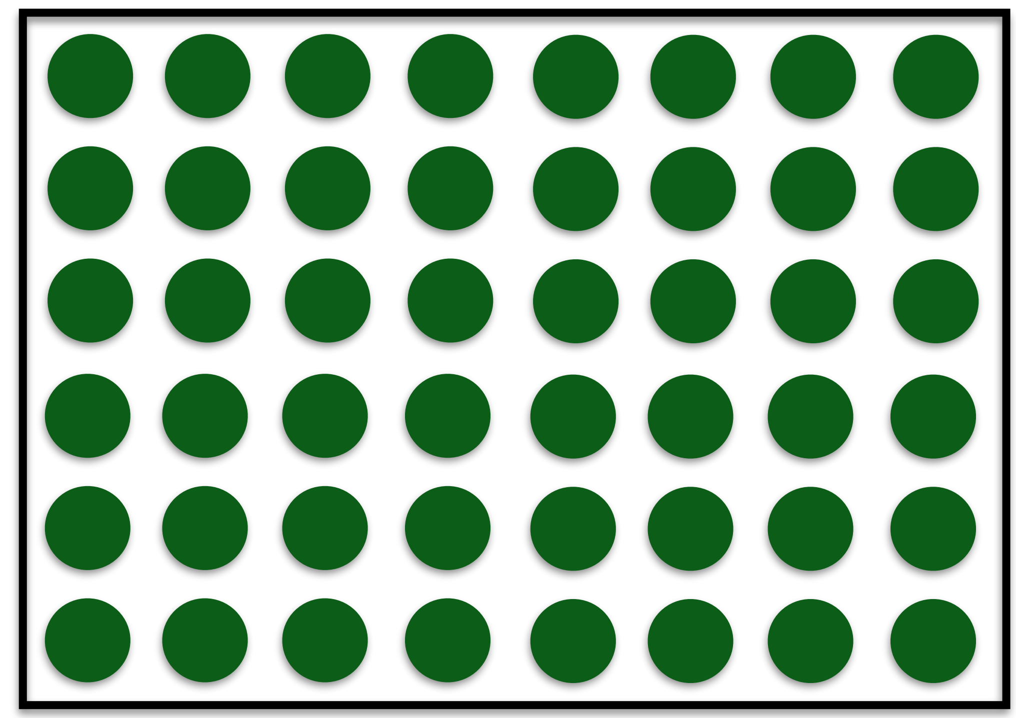
Sentences as matrices



Ich möchte ein Bier



Mach's gut



Die Wahrheiten der Menschen sind die unwiderlegbaren Irrtümer

Question: How do we build these matrices?

Sentences as matrices

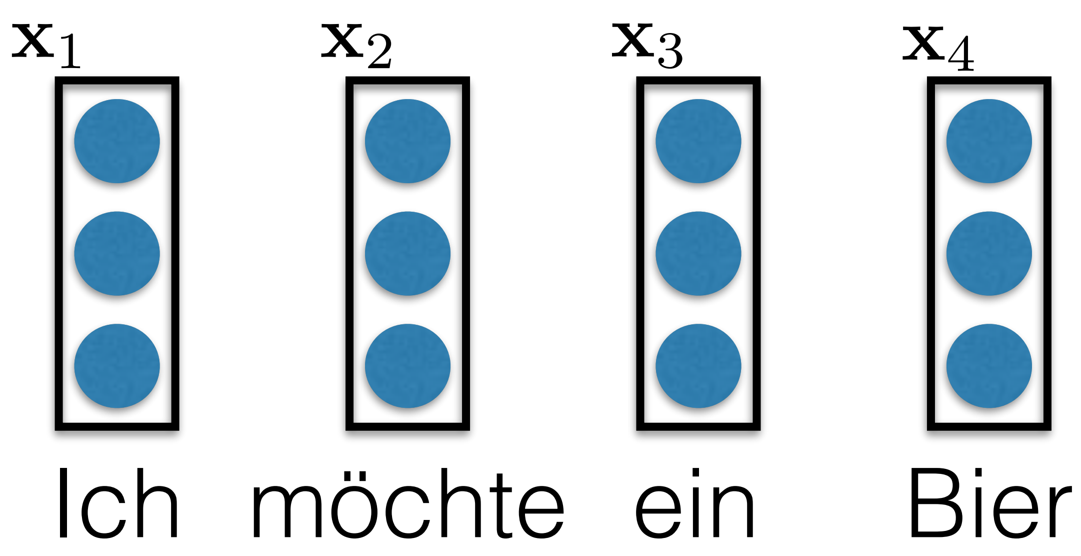
With concatenation

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

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

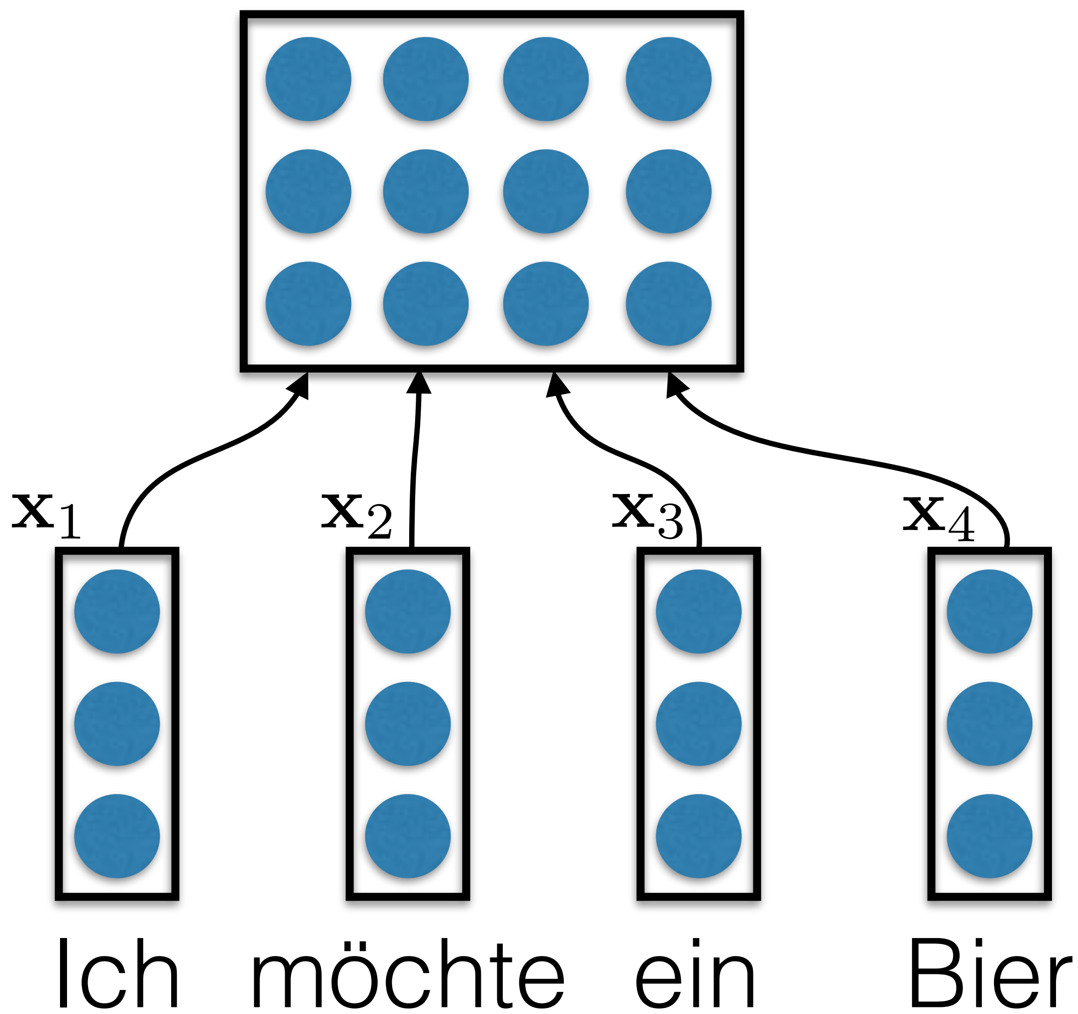
- Simplest possible model

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

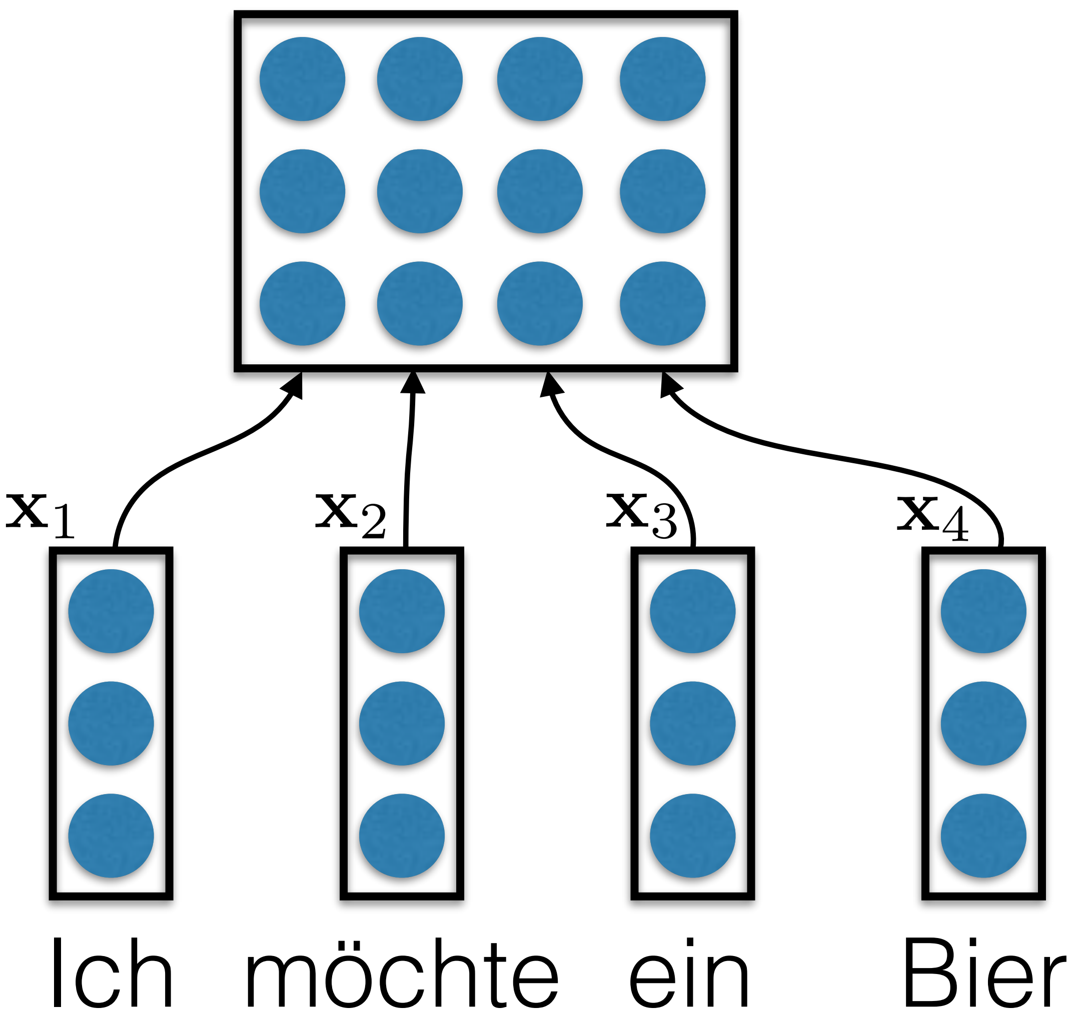


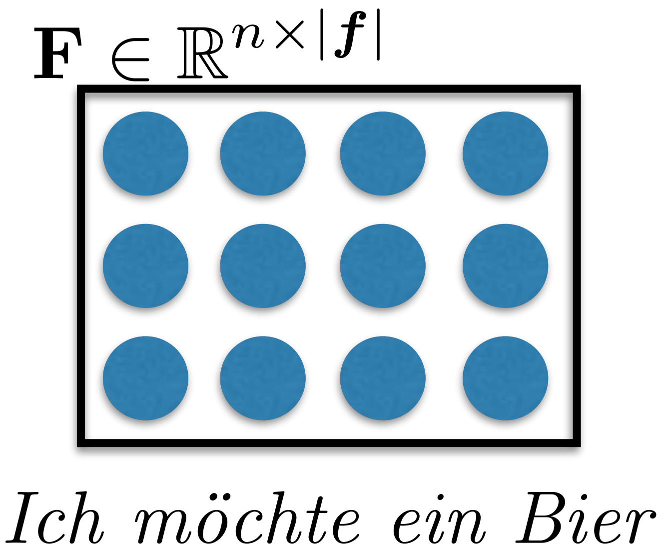
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier

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



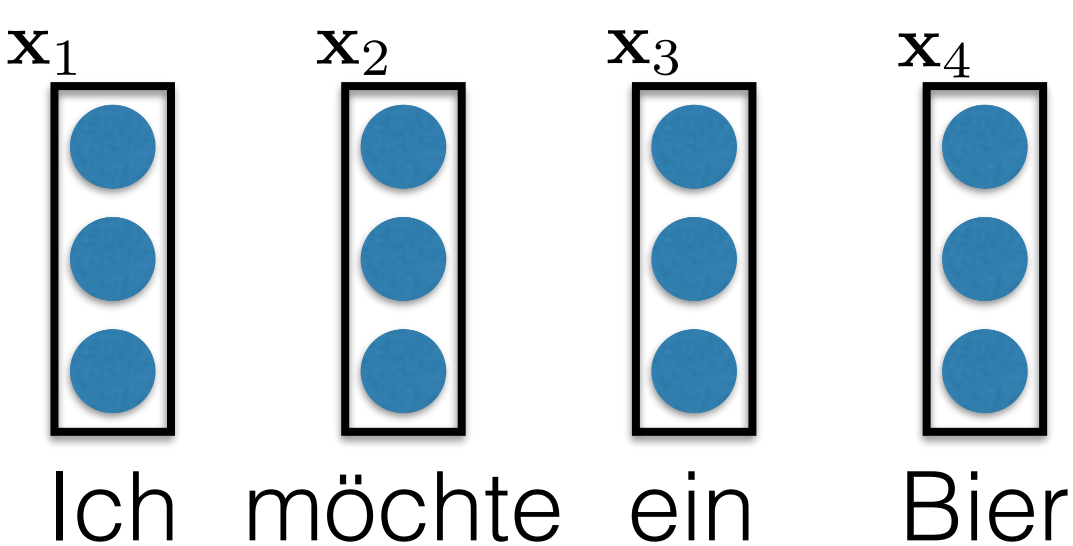
$$\mathbf{f}_i = \mathbf{x}_i$$



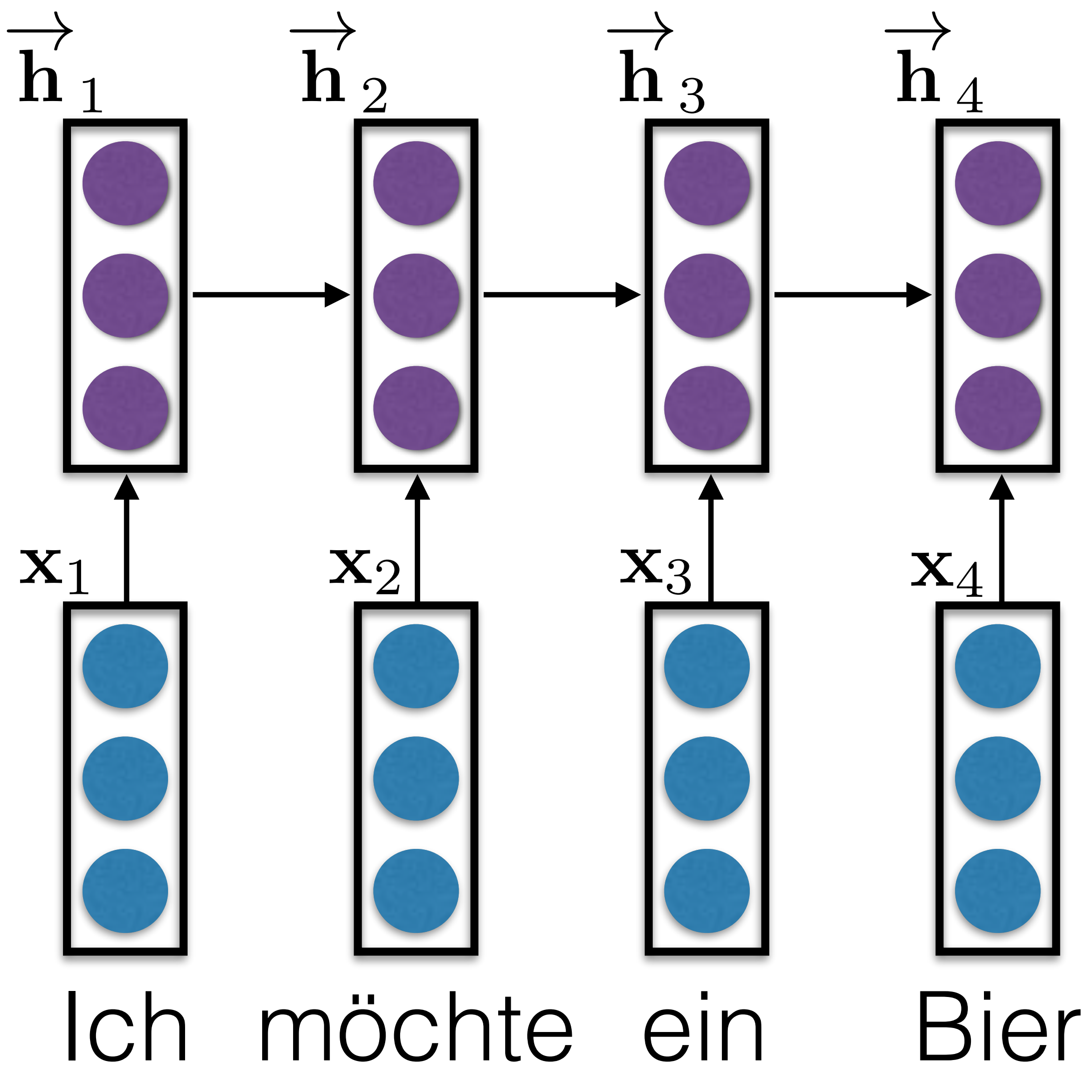


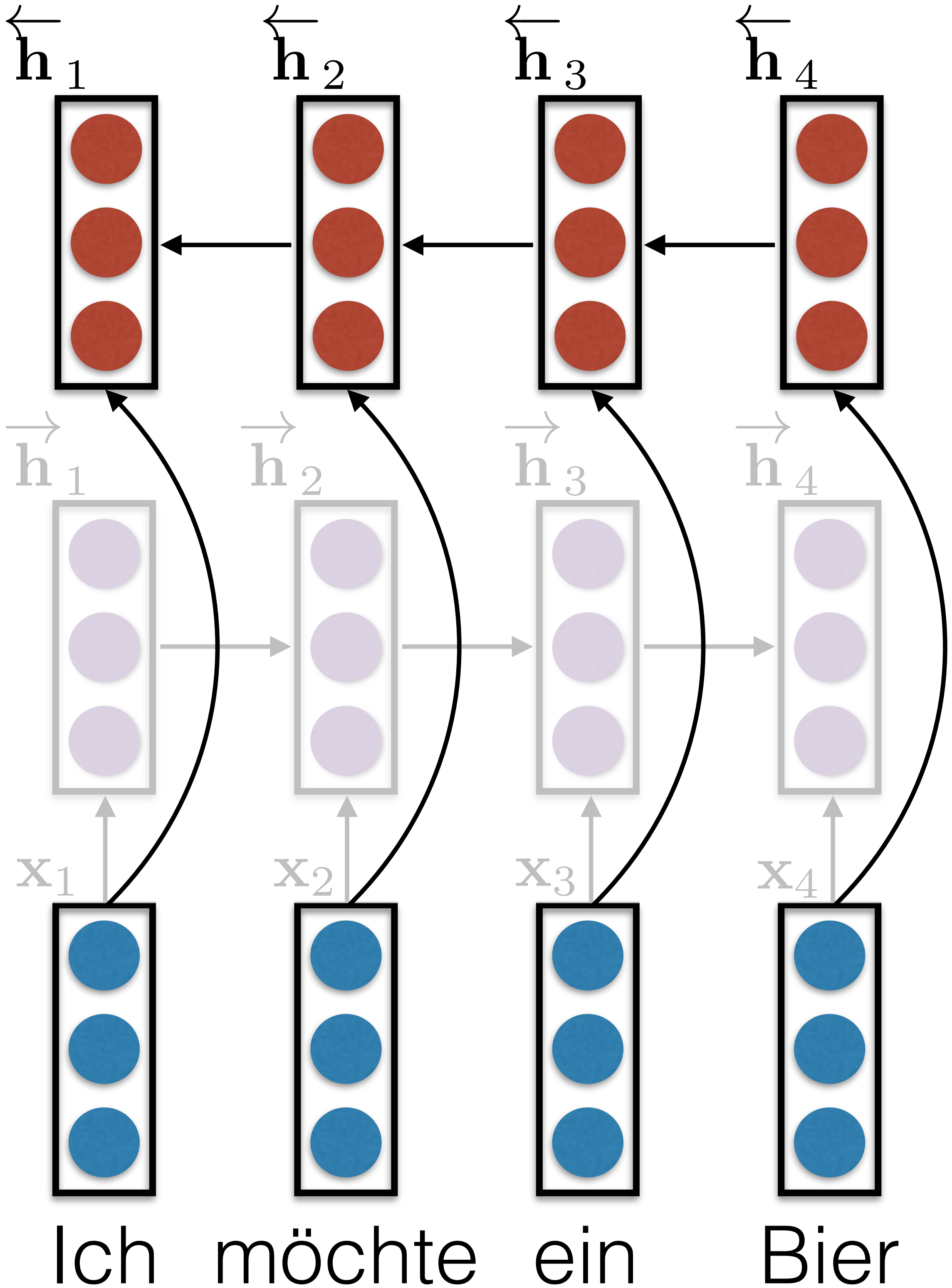
Sentences as matrices
With bidirectional RNNs

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



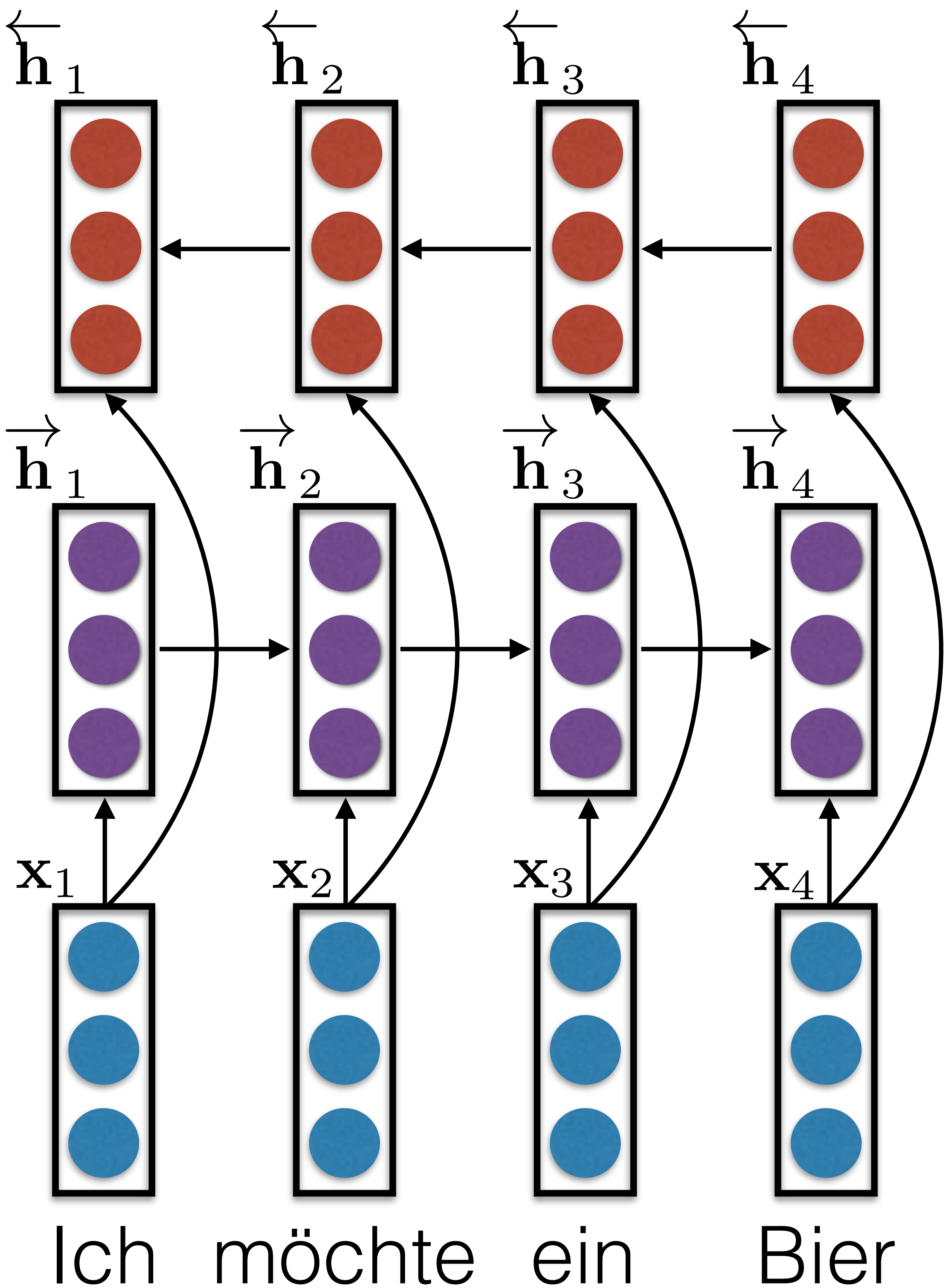
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier


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

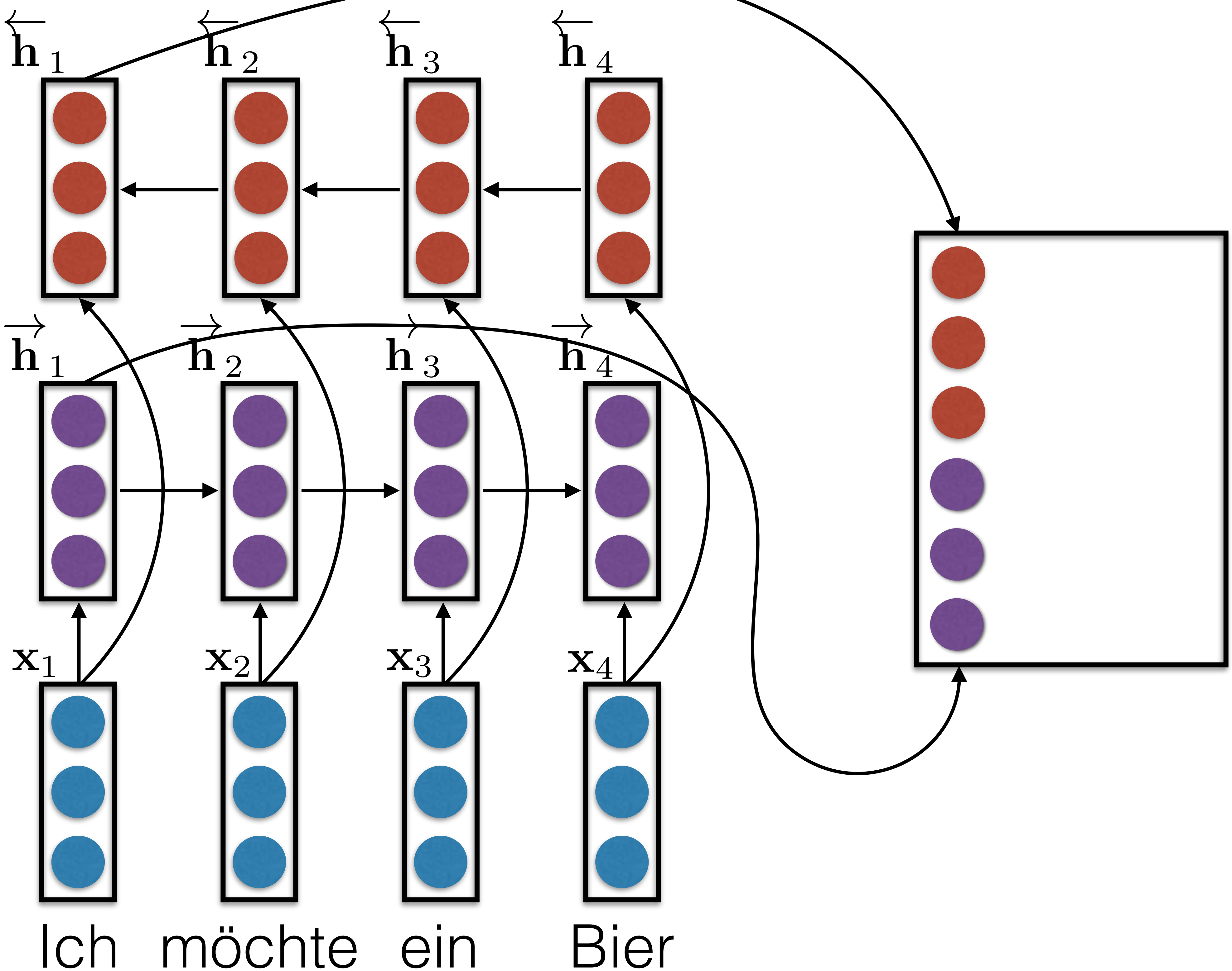


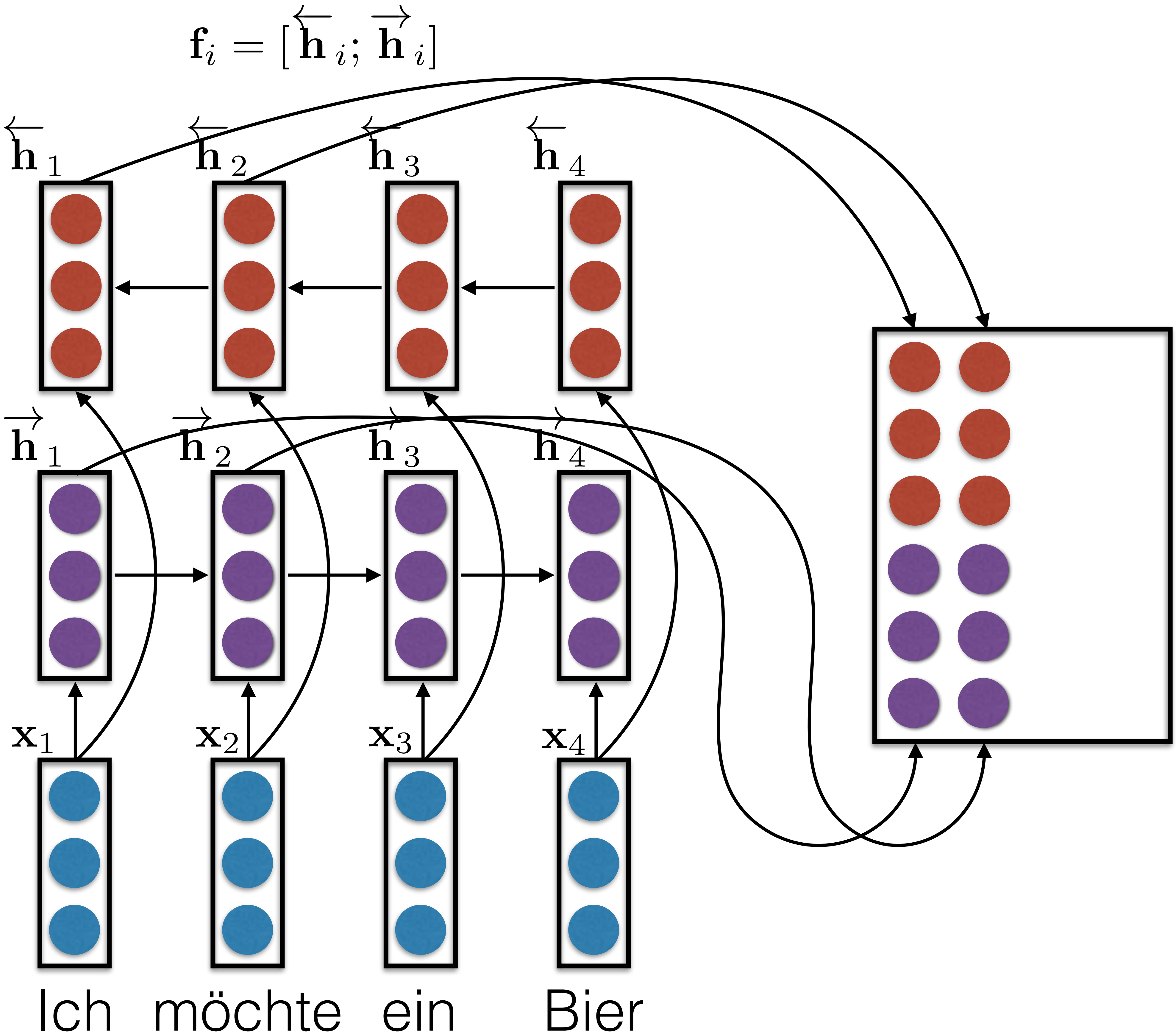
$\overleftarrow{\mathbf{h}}_1$
$\overleftarrow{\mathbf{h}}_2$
$\overleftarrow{\mathbf{h}}_3$
$\overleftarrow{\mathbf{h}}_4$
$\overrightarrow{\mathbf{h}}_1$
$\overrightarrow{\mathbf{h}}_2$
$\overrightarrow{\mathbf{h}}_3$
$\overrightarrow{\mathbf{h}}_4$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$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

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

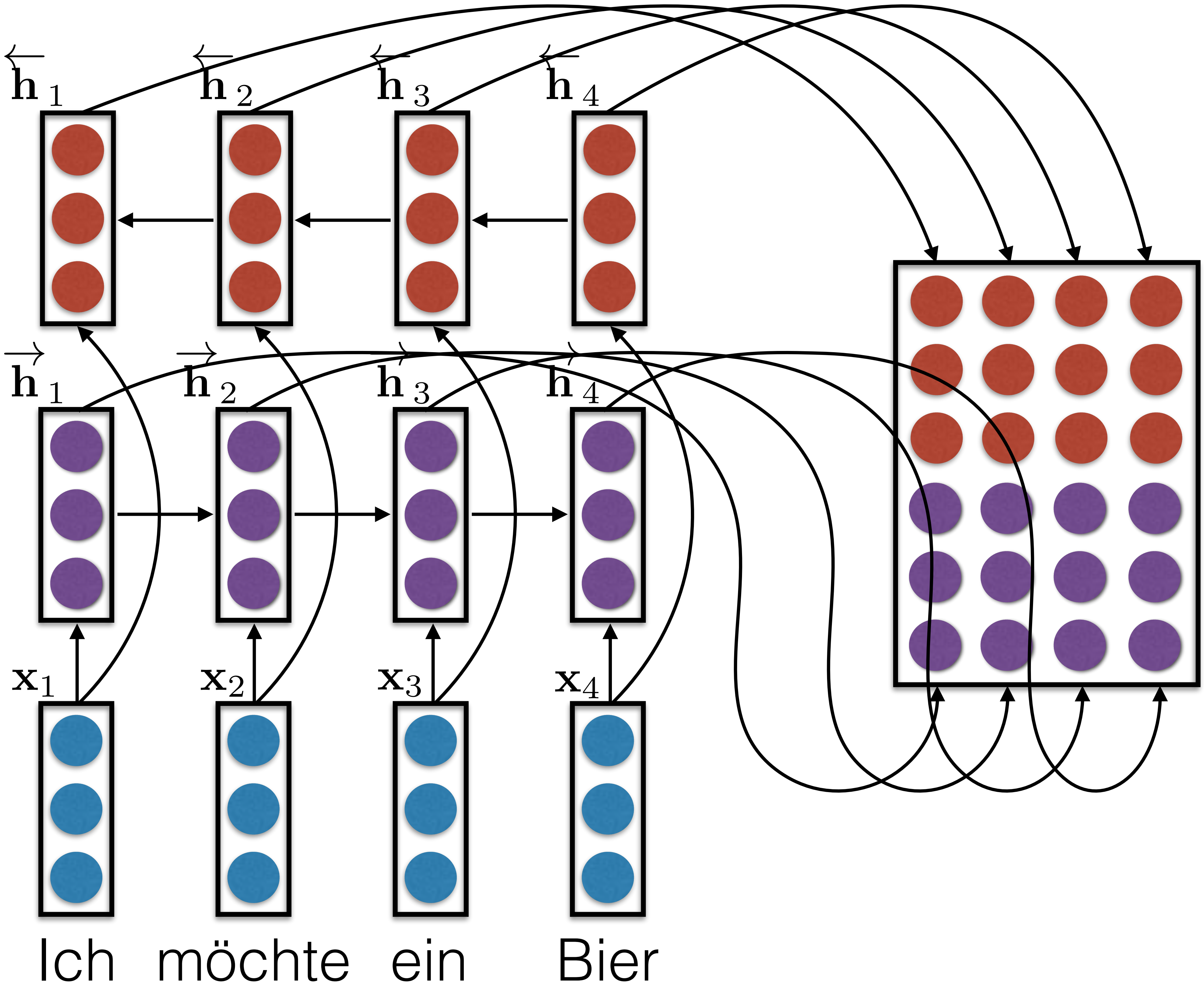


$$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$$

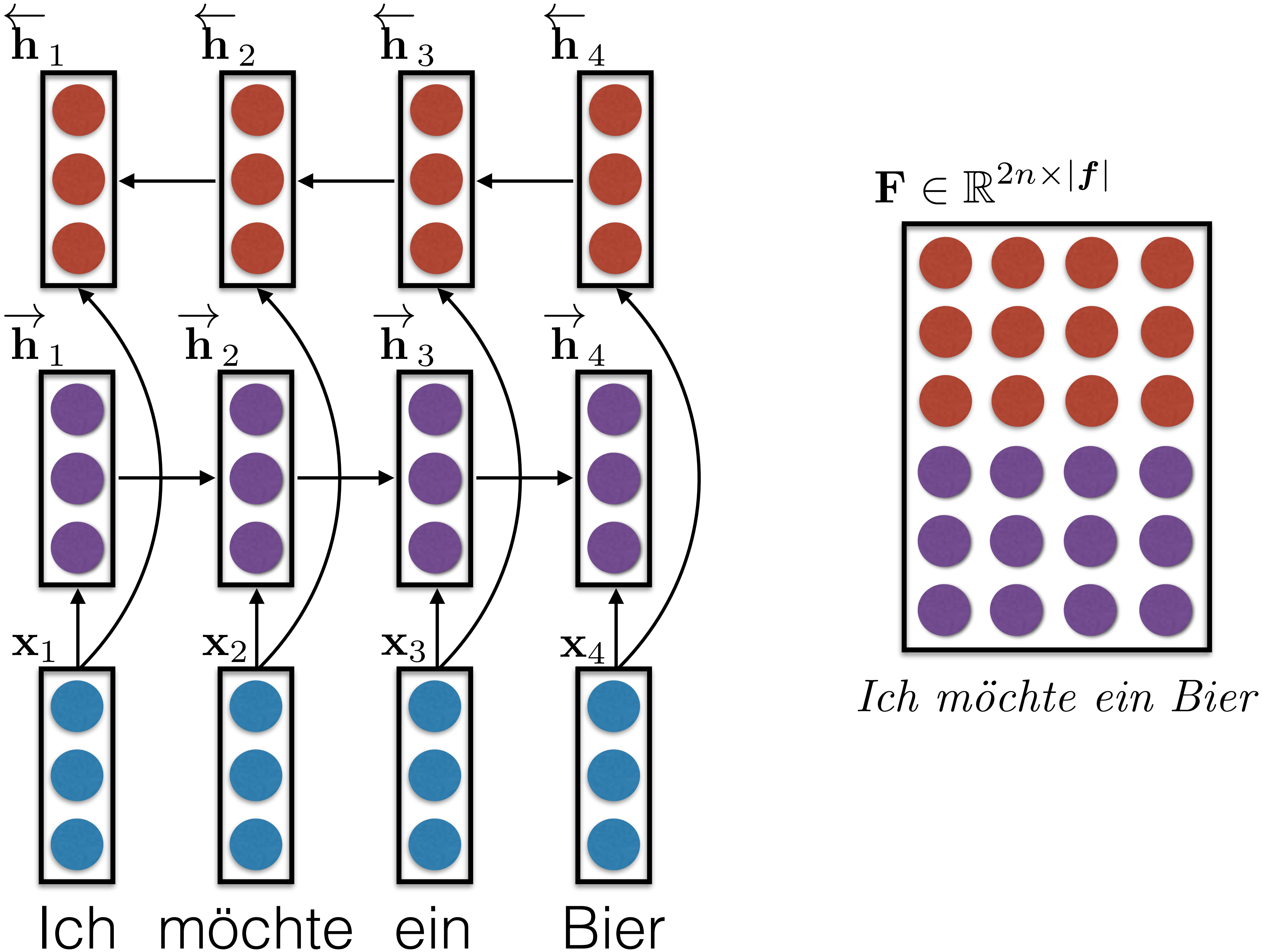



$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$
$\overleftarrow{\mathbf{h}}_1$
$\overleftarrow{\mathbf{h}}_2$
$\overleftarrow{\mathbf{h}}_3$
$\overleftarrow{\mathbf{h}}_4$
$\overrightarrow{\mathbf{h}}_1$
$\overrightarrow{\mathbf{h}}_2$
$\overrightarrow{\mathbf{h}}_3$
$\overrightarrow{\mathbf{h}}_4$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

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



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



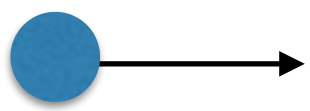
Sentences as matrices
Where are we in 2018?

- There are lots of ways to construct **F**
  - More exotic architectures coming out daily
  - Increasingly common goal: get rid of O(|***f***|) sequential processing steps, i.e., RNNs during training
  - syntactic information can help (Sennrich & Haddow, 2016; Nadejde et al., 2017), but many more integration strategies are possible
  - try something with phrase types instead of word types?

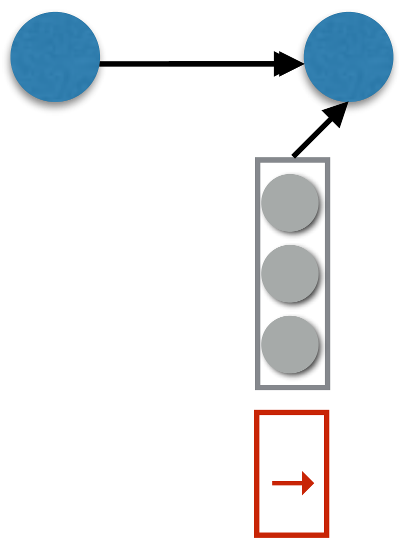
<u>Multi-word</u> <u>expressions</u> <u>are</u> <u>a</u> <u>pain in the neck</u> .

Conditioning on matrices

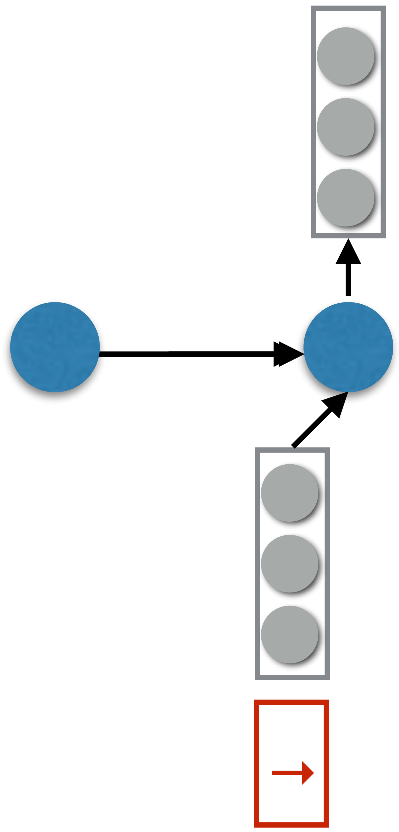
- We have a matrix **F** representing the input, now we need to generate from it
- Bahdanau et al. (2015) and Luong et al. (2015) concurrently proposed using ***attention*** for translating from matrix-encoded sentences
- High-level idea
    - Generate the output sentence word by word using an RNN
    - 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***



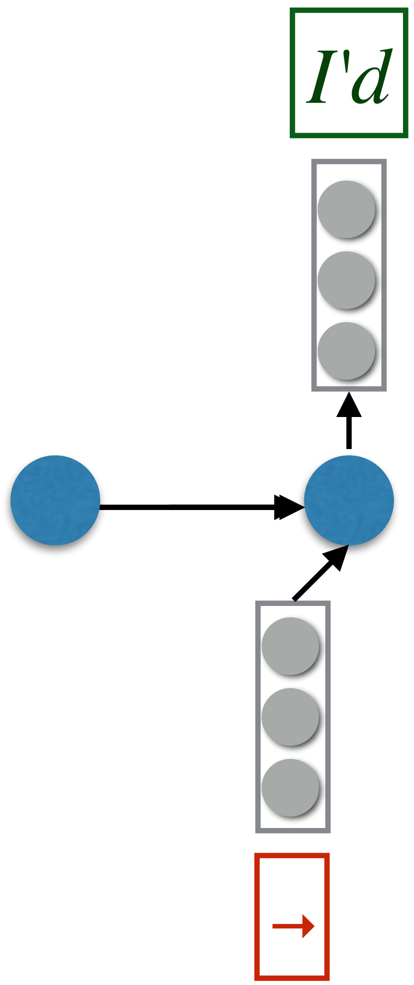
Recall RNNs…



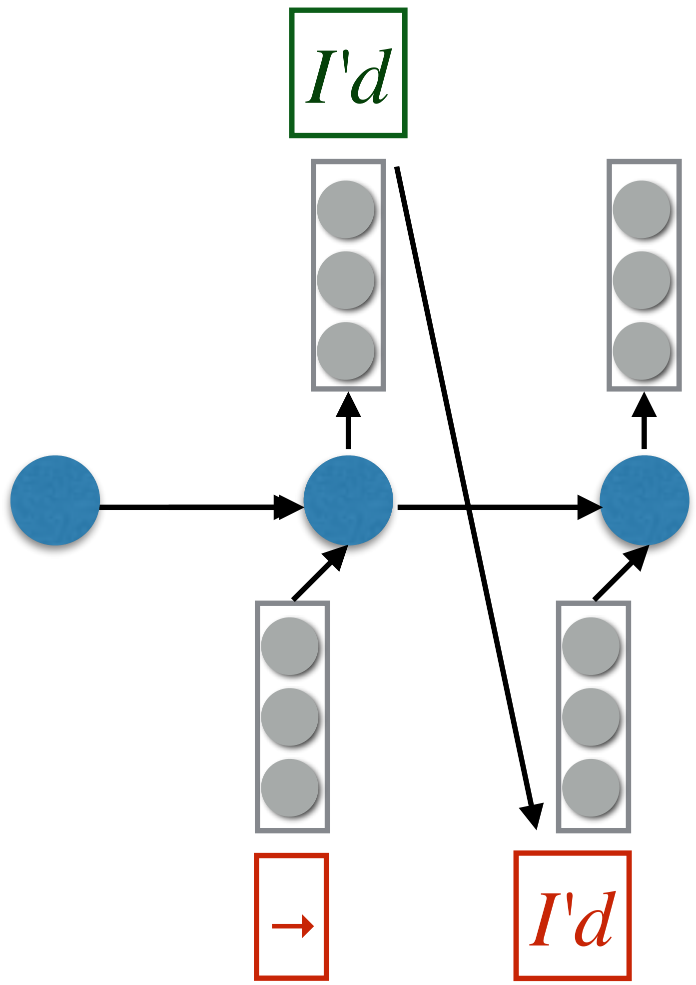
Recall RNNs…



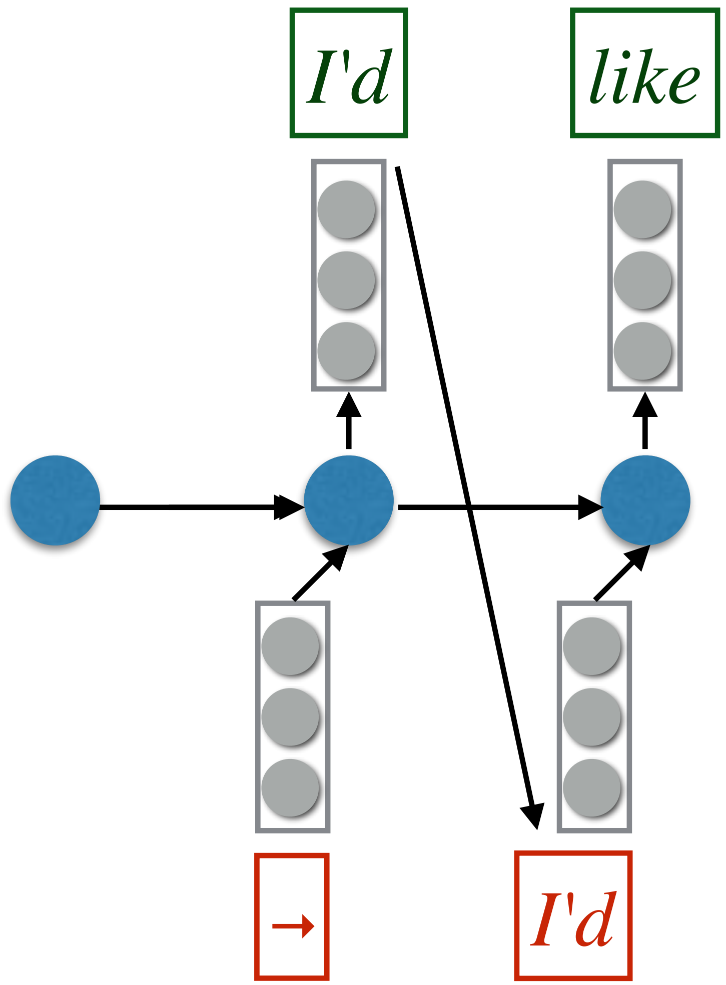
Recall RNNs…



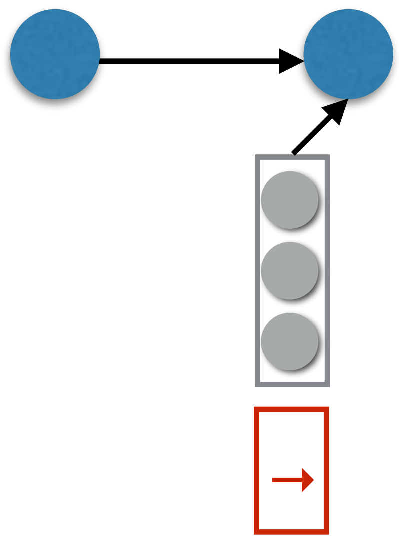
Recall RNNs…

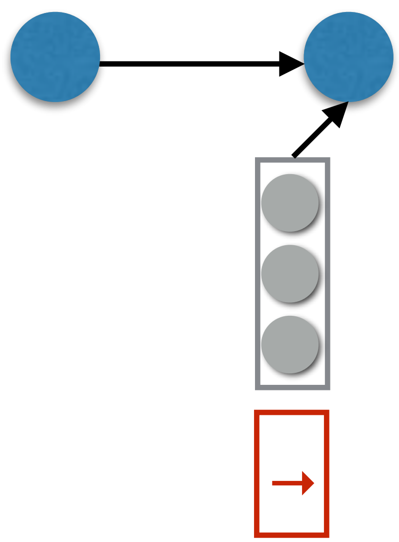


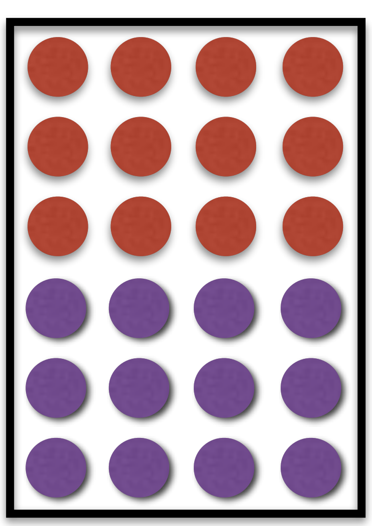
Recall RNNs…



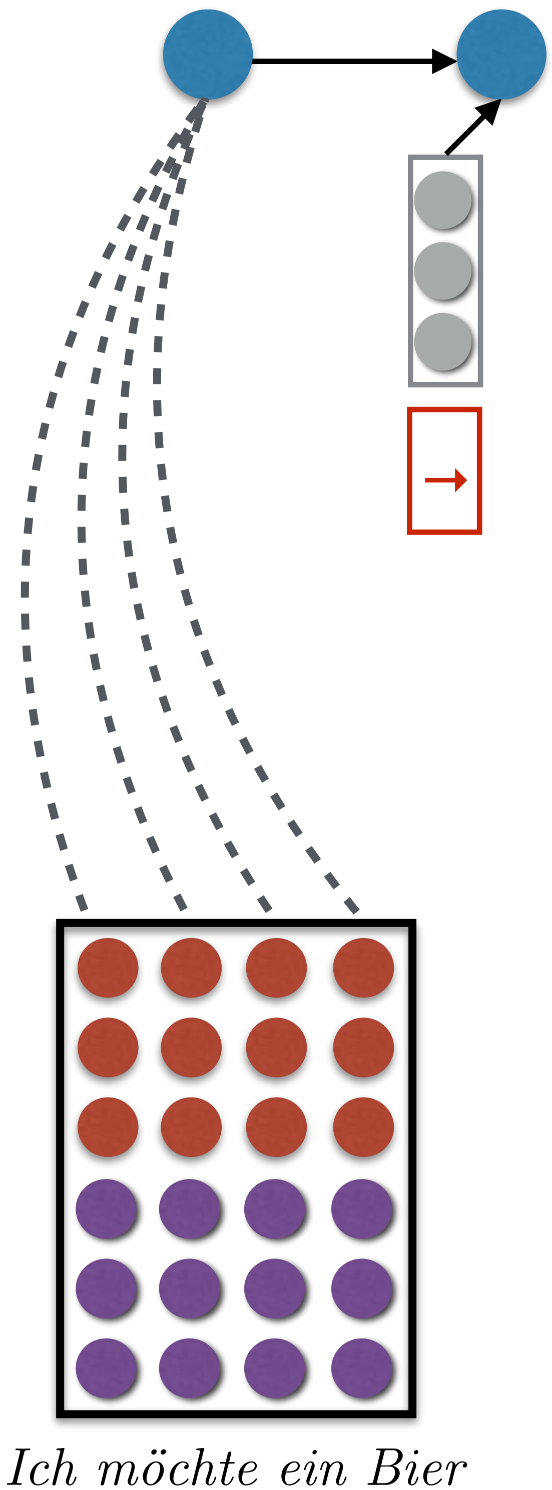
Recall RNNs…

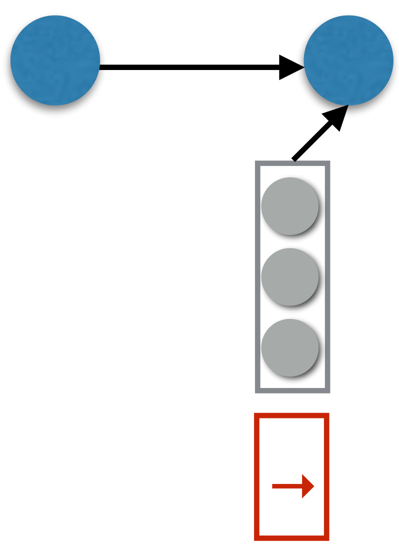


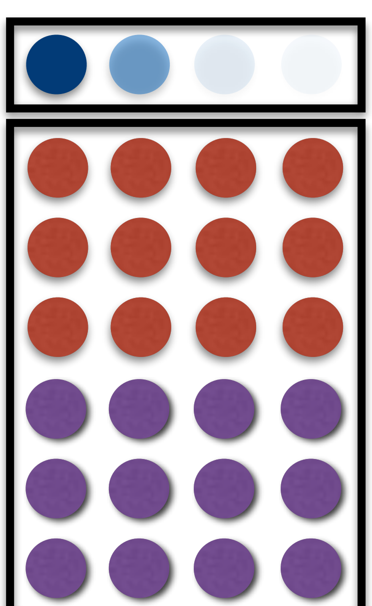




Ich möchte ein Bier


→
Ich möchte ein Bier

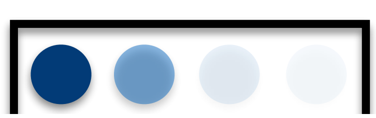


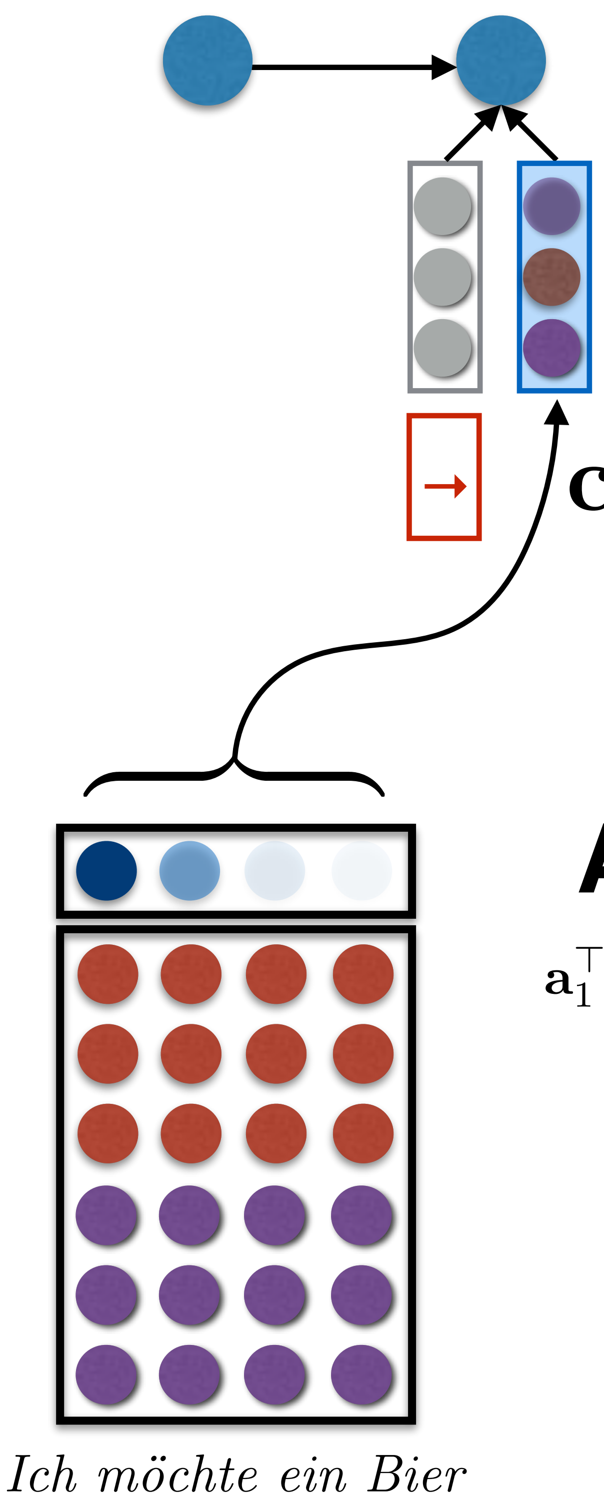


Ich möchte ein Bier

Attention history:

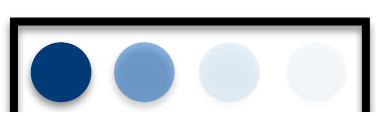
$\mathbf{a}_1^\top$

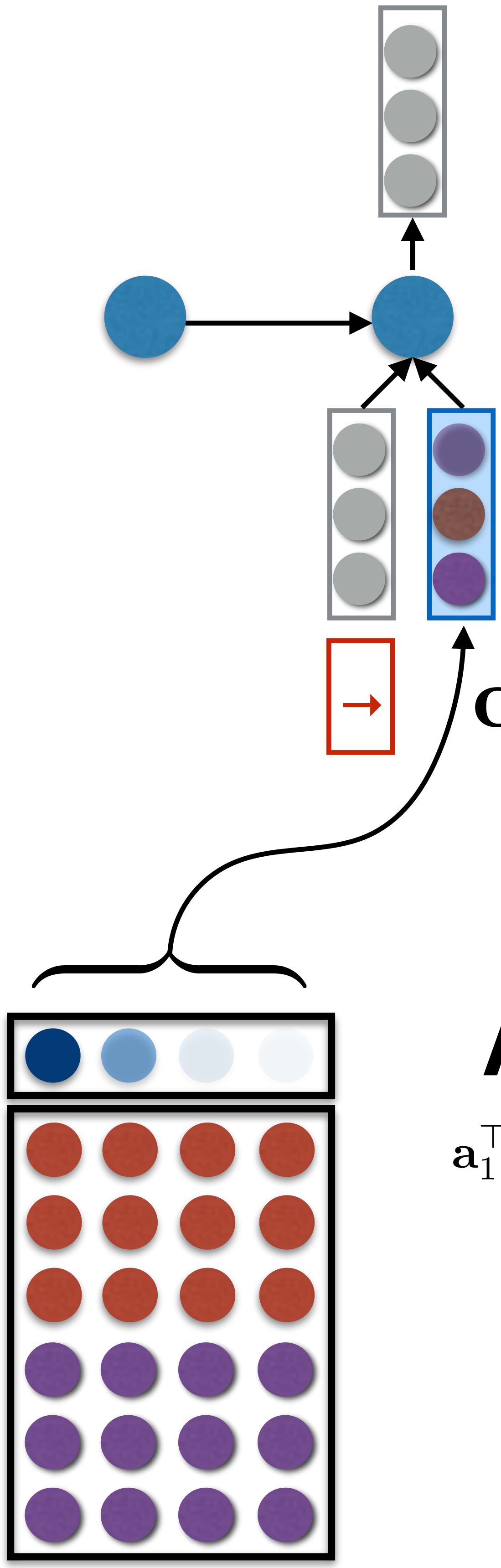




$$\mathbf{c}_1 = \mathbf{F}\mathbf{a}_1$$

Attention history:

$\mathbf{a}_1^\top$ 

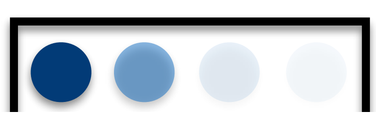


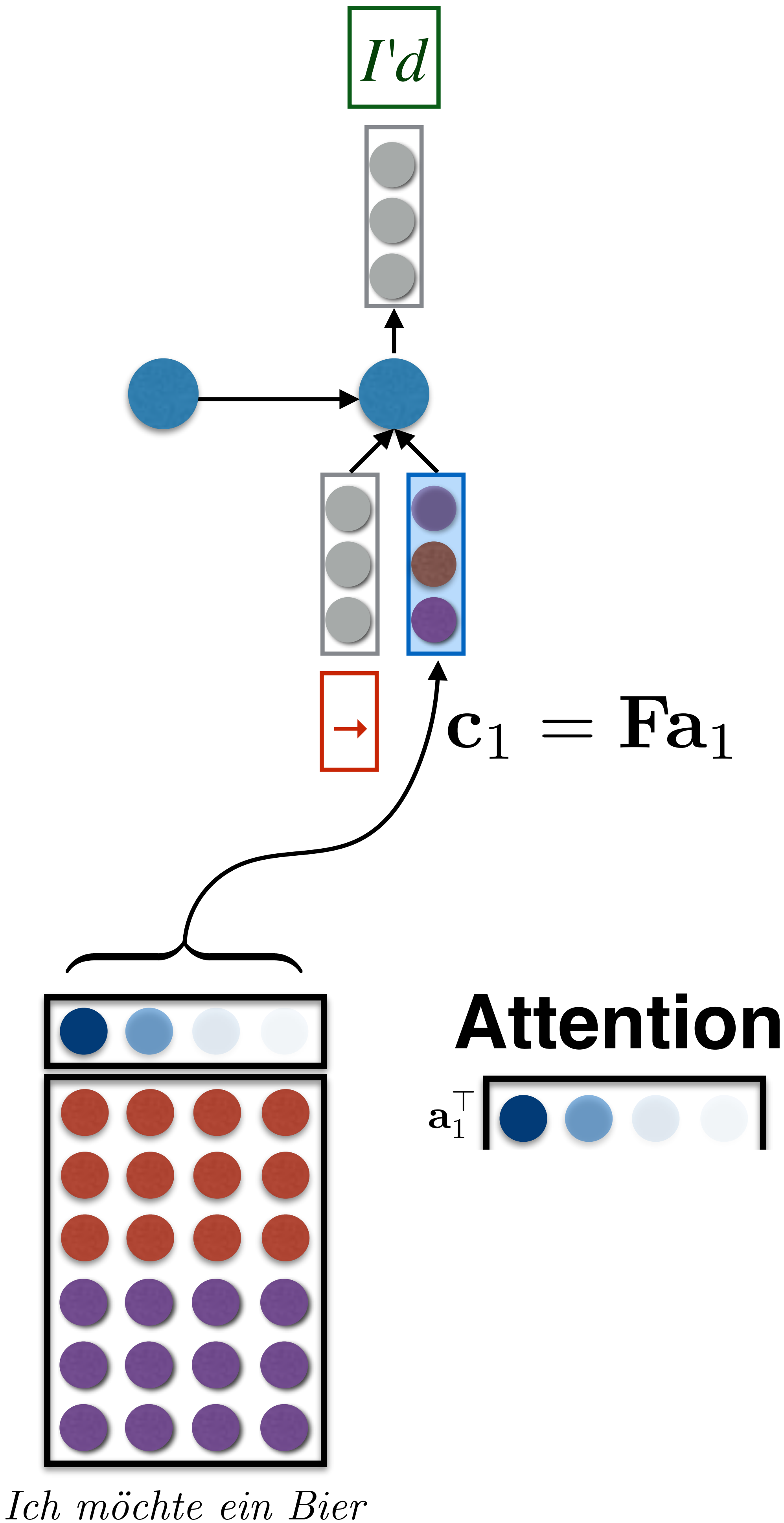
$$\mathbf{c}_1 = \mathbf{F}\mathbf{a}_1$$

Ich möchte ein Bier

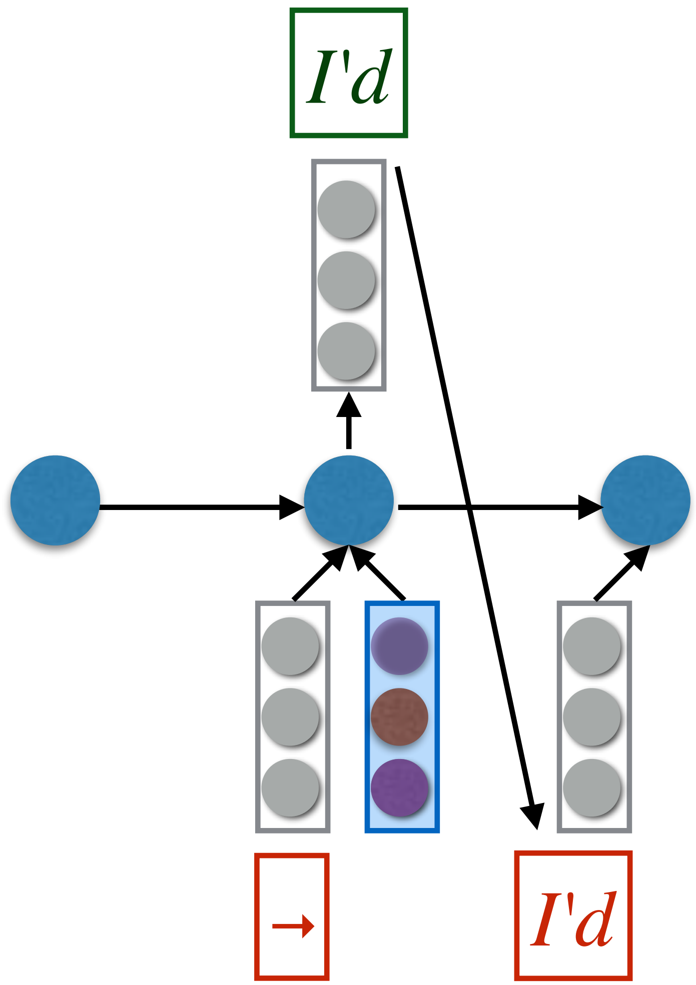
Attention history:

$\mathbf{a}_1^\top$

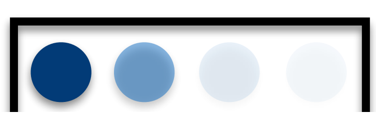


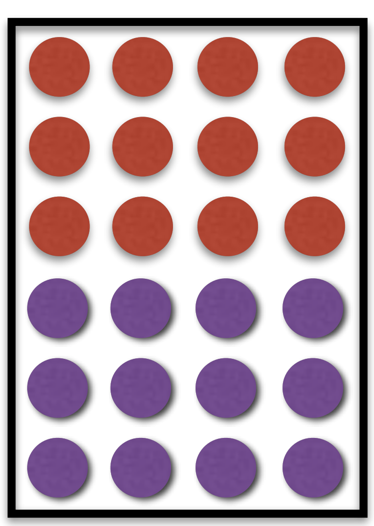


Attention history:

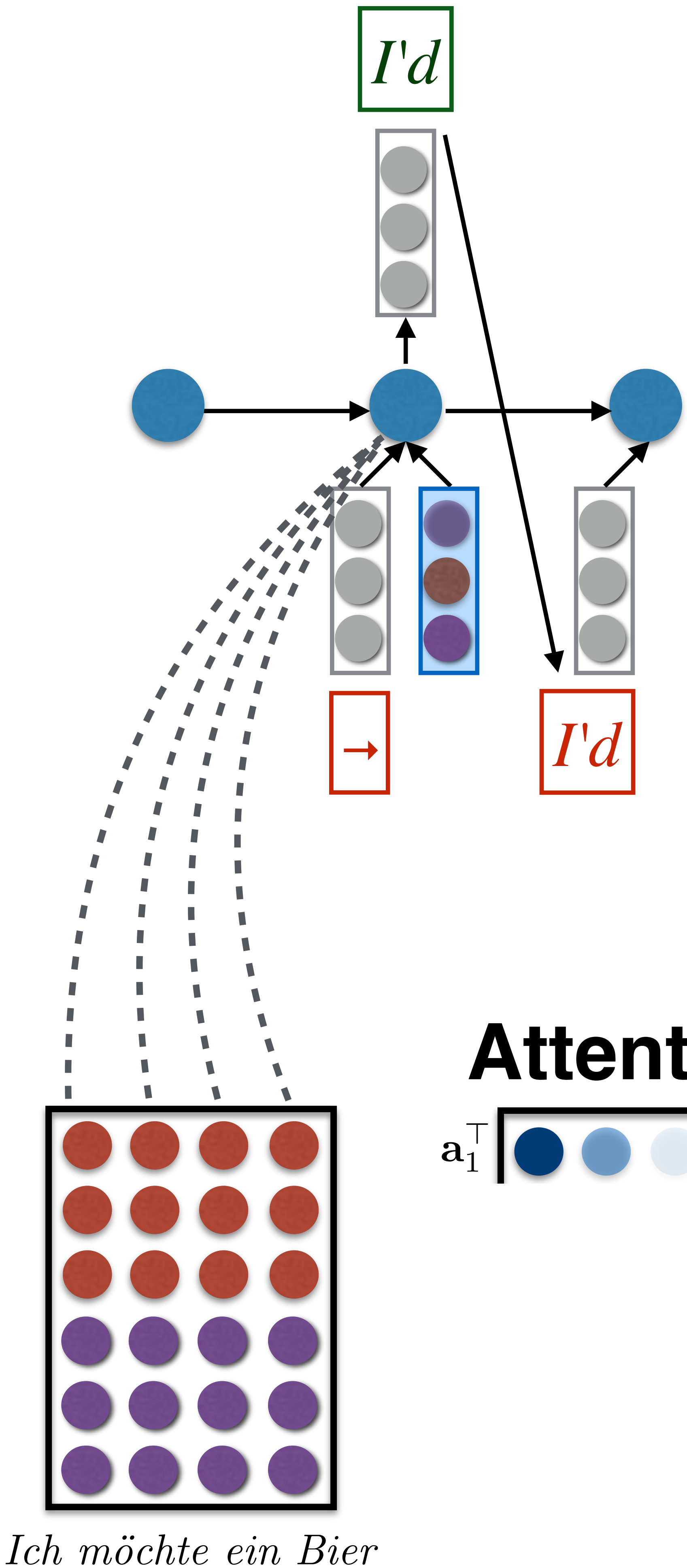


Attention history:

$\mathbf{a}_1^\top$ 

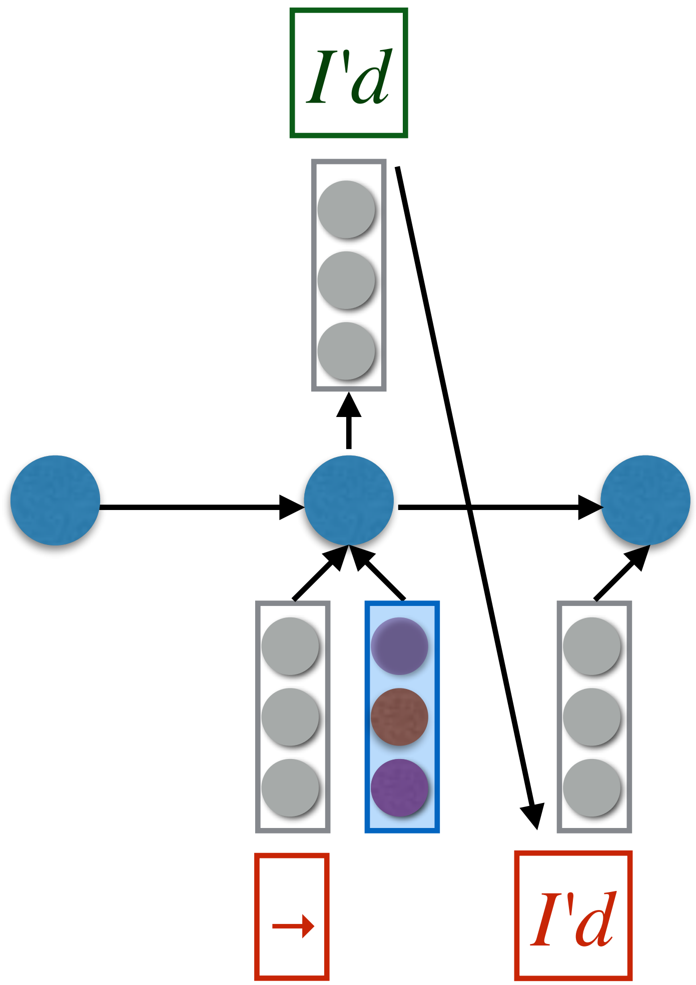


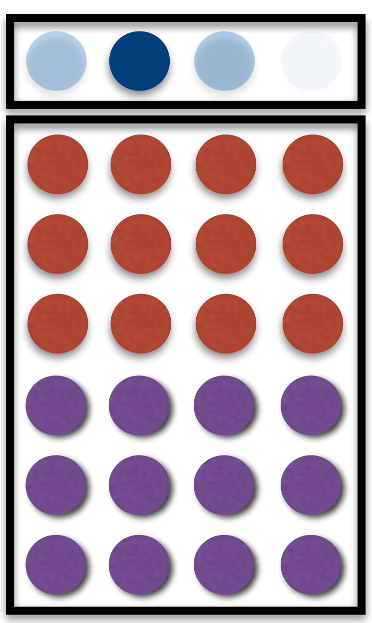
Ich möchte ein Bier



Attention history:

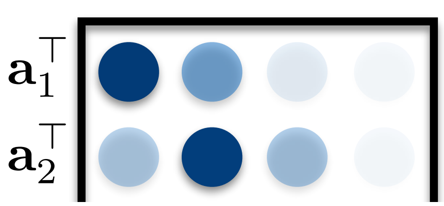
$\mathbf{a}_1^\top$

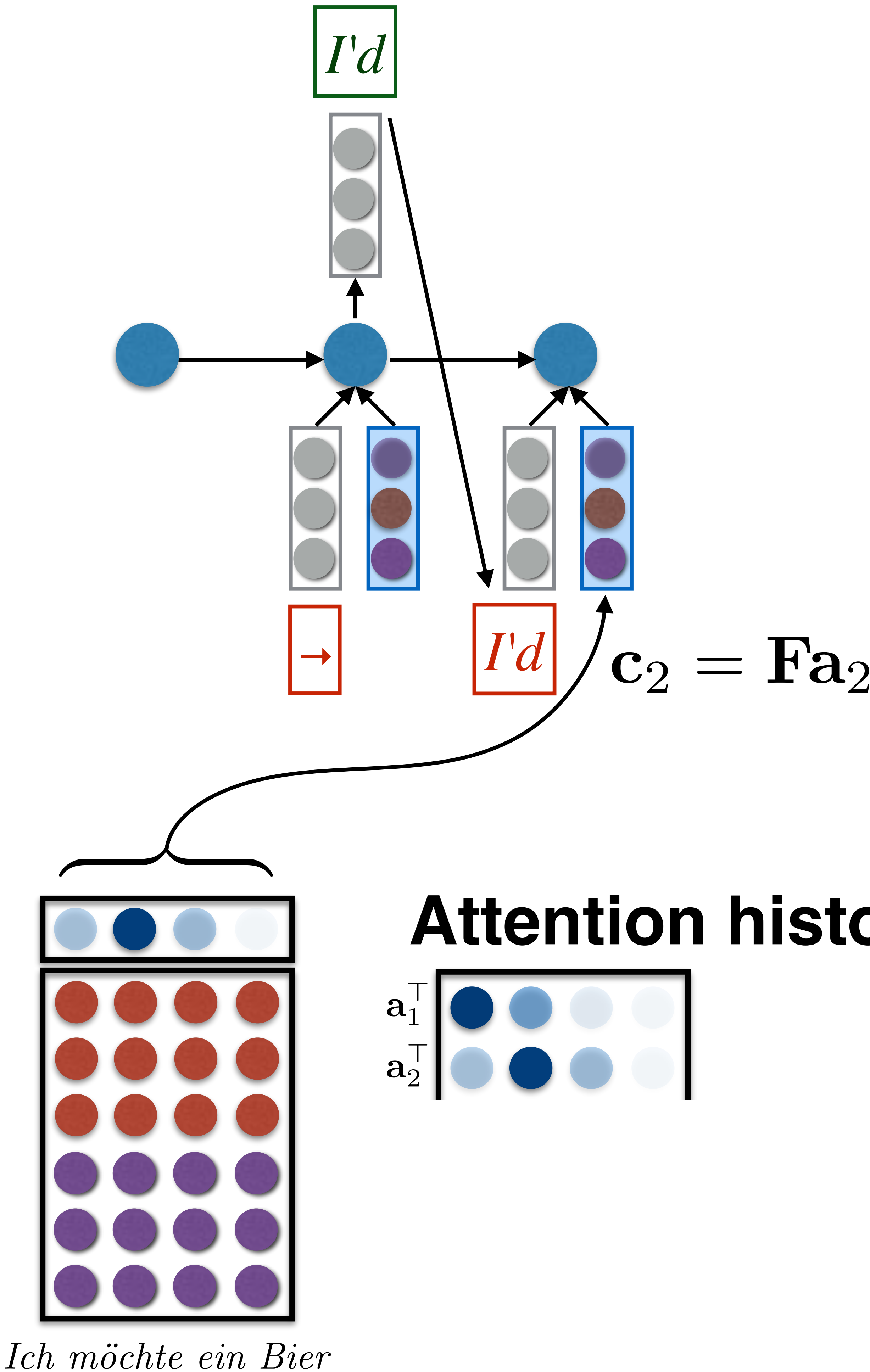




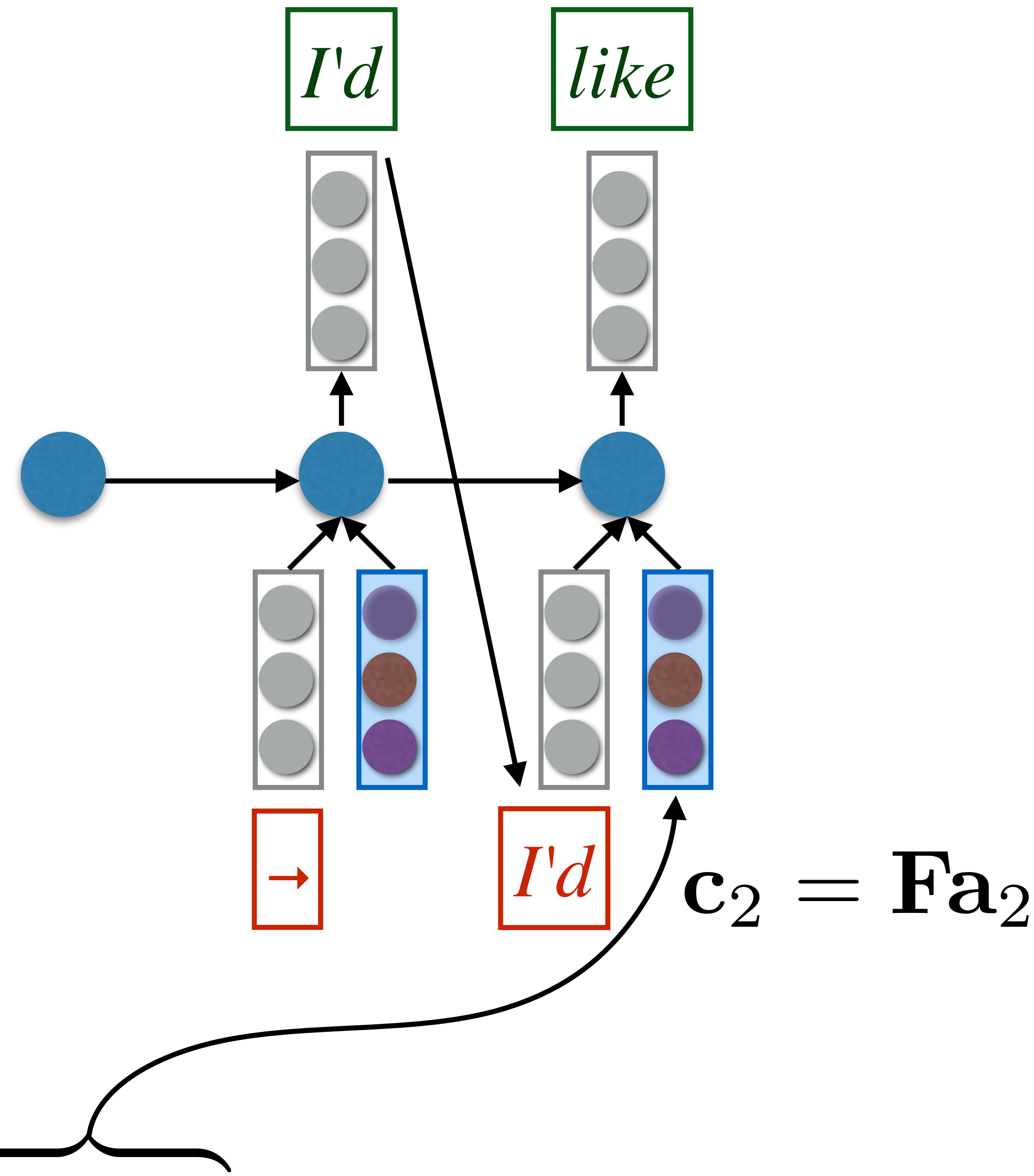
Ich möchte ein Bier

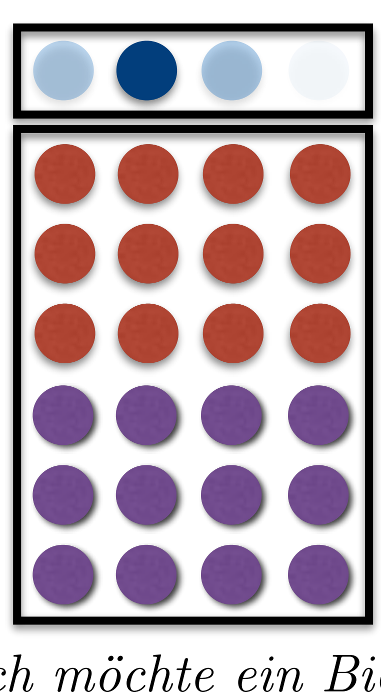
Attention history:



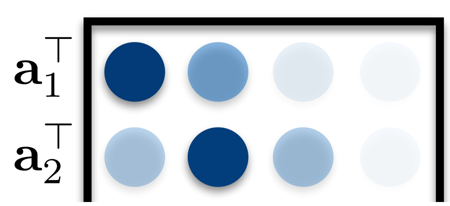


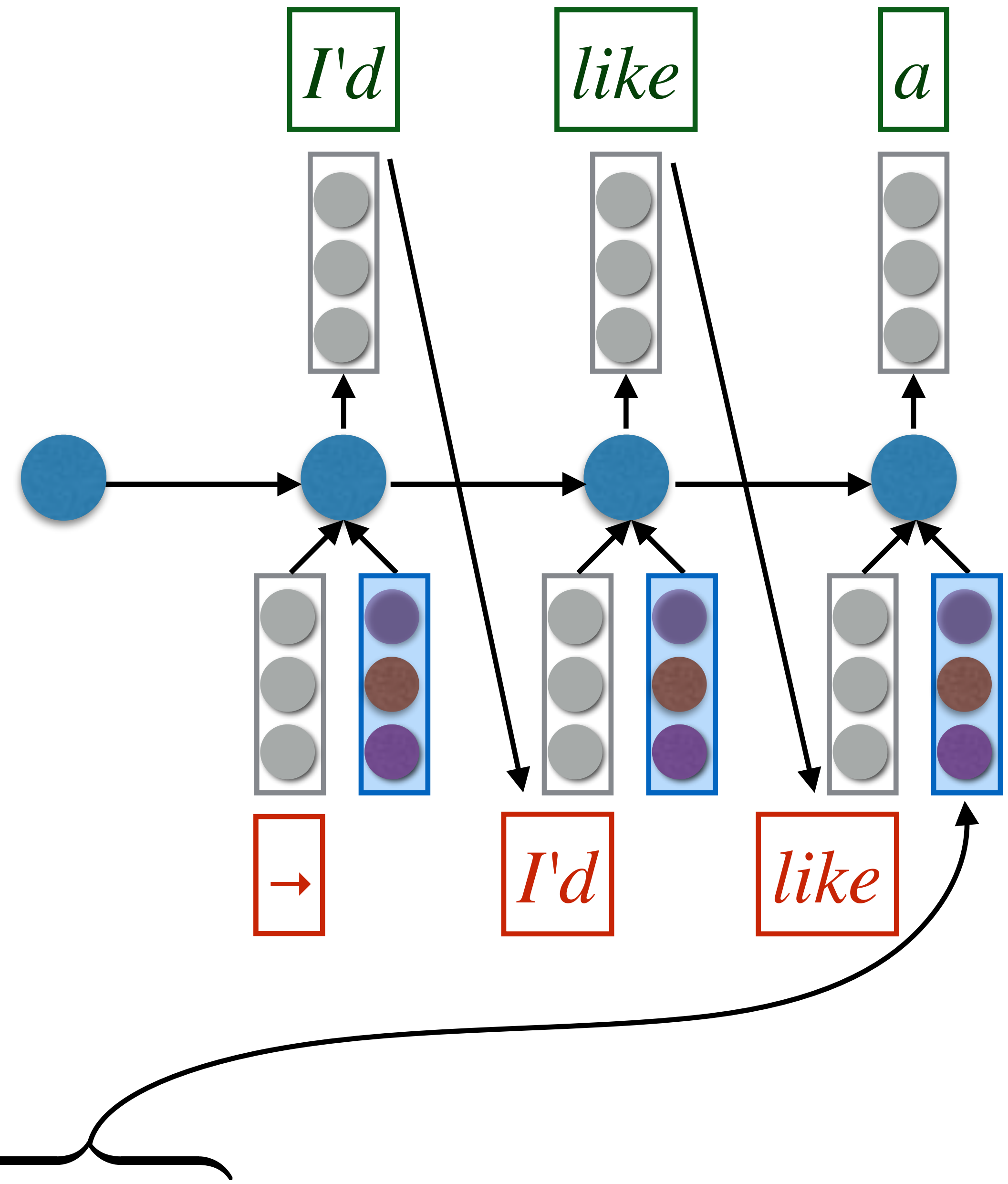
Attention history:

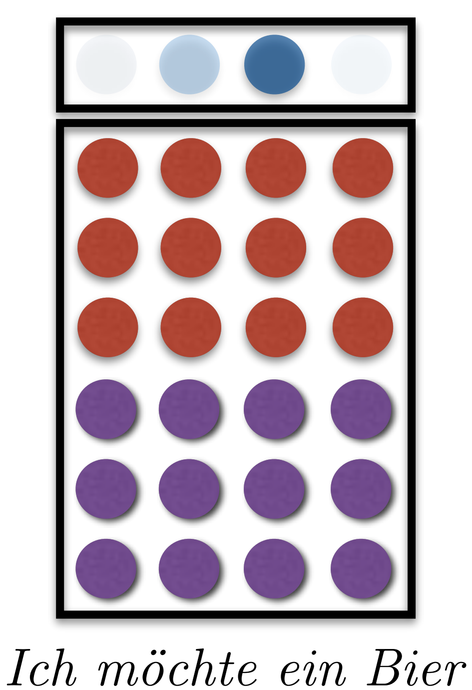




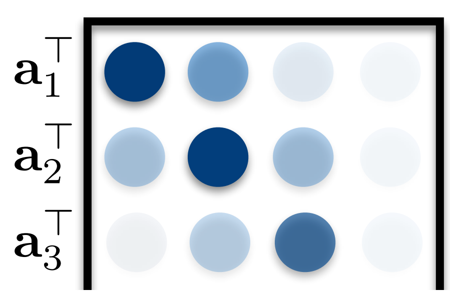
Attention history:

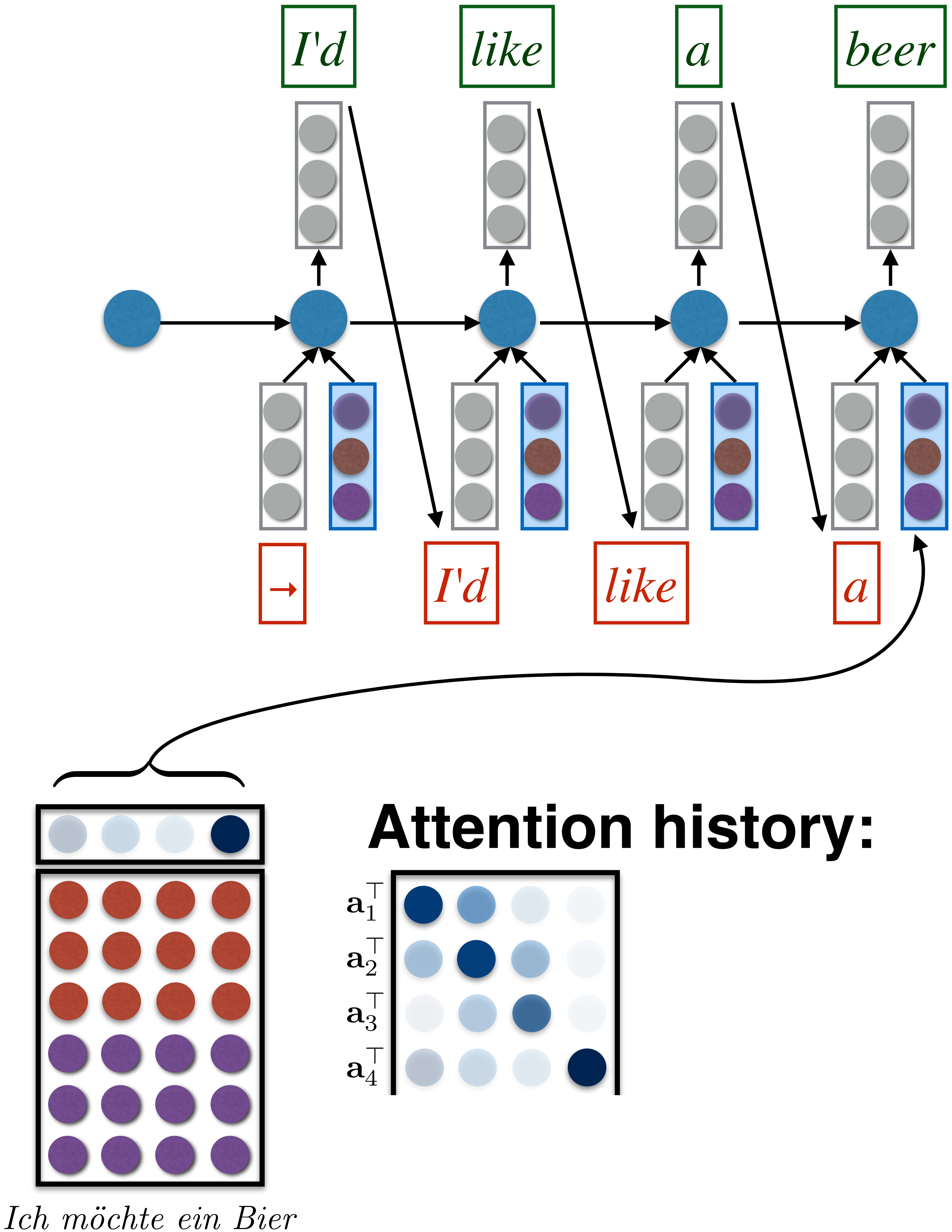




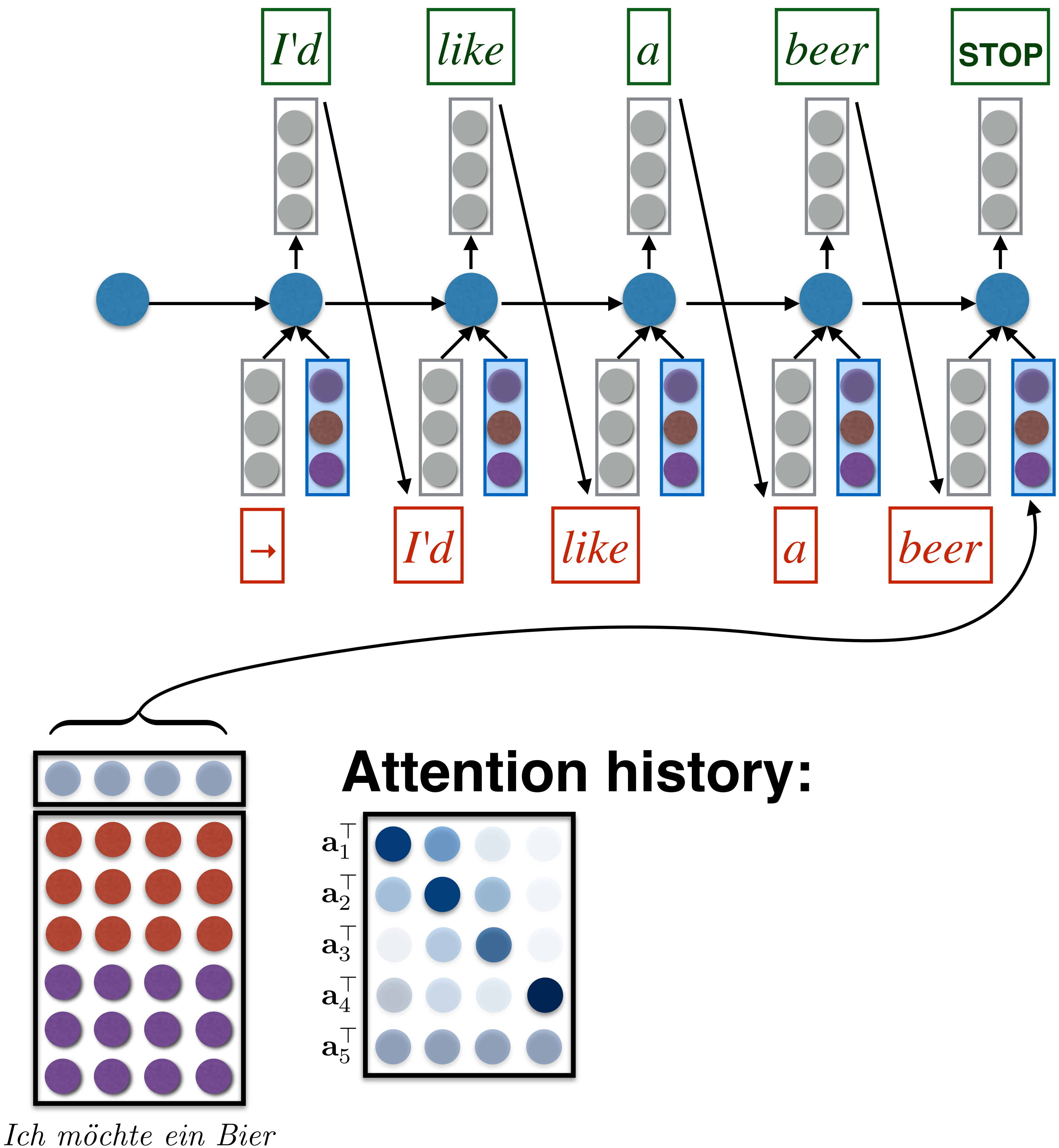


Attention history:





Attention history:


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

Attention

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

Computing attention

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

Computing attention

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

Computing attention

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

Computing attention

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

Computing attention

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

Computing attention

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

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

Computing attention

Nonlinear additive attention model

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

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

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

Computing attention
Nonlinear additive attention model

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

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

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

- Here, **W** and **v** are learned parameters of appropriate dimension and + “broadcasts” over the |***f***| columns in **WF**

- This can learn more complex interactions
  - It is unclear if the added complexity is necessary for good performance

Putting it all together

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

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

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

$t = 0$

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

$$
\begin{aligned}
& t = t + 1 \\
& \left.
\begin{aligned}
\mathbf{r}_t &= \mathbf{V}\mathbf{s}_{t-1} \\
\mathbf{u}_t &= \mathbf{v}^\top \tanh(\mathbf{W}\mathbf{F} + \mathbf{r}_t) \\
\mathbf{a}_t &= \mathrm{softmax}(\mathbf{u}_t)
\end{aligned}
\right\} \text{(Compute attention; part 2 of lecture)} \\
& \mathbf{c}_t = \mathbf{F}\mathbf{a}_t \\
& \mathbf{s}_t = \mathrm{RNN}(\mathbf{s}_{t-1}, [\mathbf{e}_{t-1}; \mathbf{c}_t]) \quad (\mathbf{e}_{t-1} \text{ is a learned embedding of } e_t) \\
& \mathbf{y}_t = \mathrm{softmax}(\mathbf{P}\mathbf{s}_t + \mathbf{b}) \quad (\mathbf{P} \text{ and } \mathbf{b} \text{ are learned parameters}) \\
& e_t \mid \boldsymbol{e}_{<t} \sim \mathrm{Categorical}(\mathbf{y}_t)
\end{aligned}
$$

Putting it all together

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

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

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

$t = 0$

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

$$t = t + 1$$

doesn't depend on output decisions

$$\left.\begin{aligned} \mathbf{r}_t &= \mathbf{V}\mathbf{s}_{t-1} \\ \mathbf{u}_t &= \mathbf{v}^\top \tanh(\mathbf{W}\mathbf{F} + \mathbf{r}_t) \\ \mathbf{a}_t &= \mathrm{softmax}(\mathbf{u}_t) \end{aligned}\right\} \text{(Compute attention; part 2 of lecture)}$$

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

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

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

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

Putting it all together

$$
\begin{aligned}
&\mathbf{F} = \mathrm{EncodeAsMatrix}(\boldsymbol{f}) && \text{(Part 1 of lecture)}\\
&e_0 = \langle \mathbf{s} \rangle\\
&\mathbf{s}_0 = \mathbf{w} \quad \text{(Learned initial state; Bahdanau uses } \mathbf{U}\overleftarrow{\mathbf{h}}_1 \text{)}\\
&t = 0\\
&\mathbf{X} = \mathbf{WF}\\
&\mathbf{while}\ e_t \neq \langle /\mathbf{s} \rangle :\\
&\qquad t = t + 1\\
&\qquad \left.\begin{aligned}
&\mathbf{r}_t = \mathbf{V}\mathbf{s}_{t-1}\\
&\mathbf{u}_t = \mathbf{v}^\top \tanh(\overset{\mathbf{X}}{\cancel{\mathbf{WF}}} + \mathbf{r}_t)\\
&\mathbf{a}_t = \mathrm{softmax}(\mathbf{u}_t)
\end{aligned}\right\} && \text{(Compute attention; part 2 of lecture)}\\
&\qquad \mathbf{c}_t = \mathbf{F}\mathbf{a}_t\\
&\qquad \mathbf{s}_t = \mathrm{RNN}(\mathbf{s}_{t-1}, [\mathbf{e}_{t-1}; \mathbf{c}_t]) && \text{(}\mathbf{e}_{t-1} \text{ is a learned embedding of } e_t\text{)}\\
&\qquad \mathbf{y}_t = \mathrm{softmax}(\mathbf{P}\mathbf{s}_t + \mathbf{b}) && \text{(}\mathbf{P} \text{ and } \mathbf{b} \text{ are learned parameters)}\\
&\qquad e_t \mid \boldsymbol{e}_{<t} \sim \mathrm{Categorical}(\mathbf{y}_t)
\end{aligned}
$$

Putting it all together

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

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

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

$t = 0$

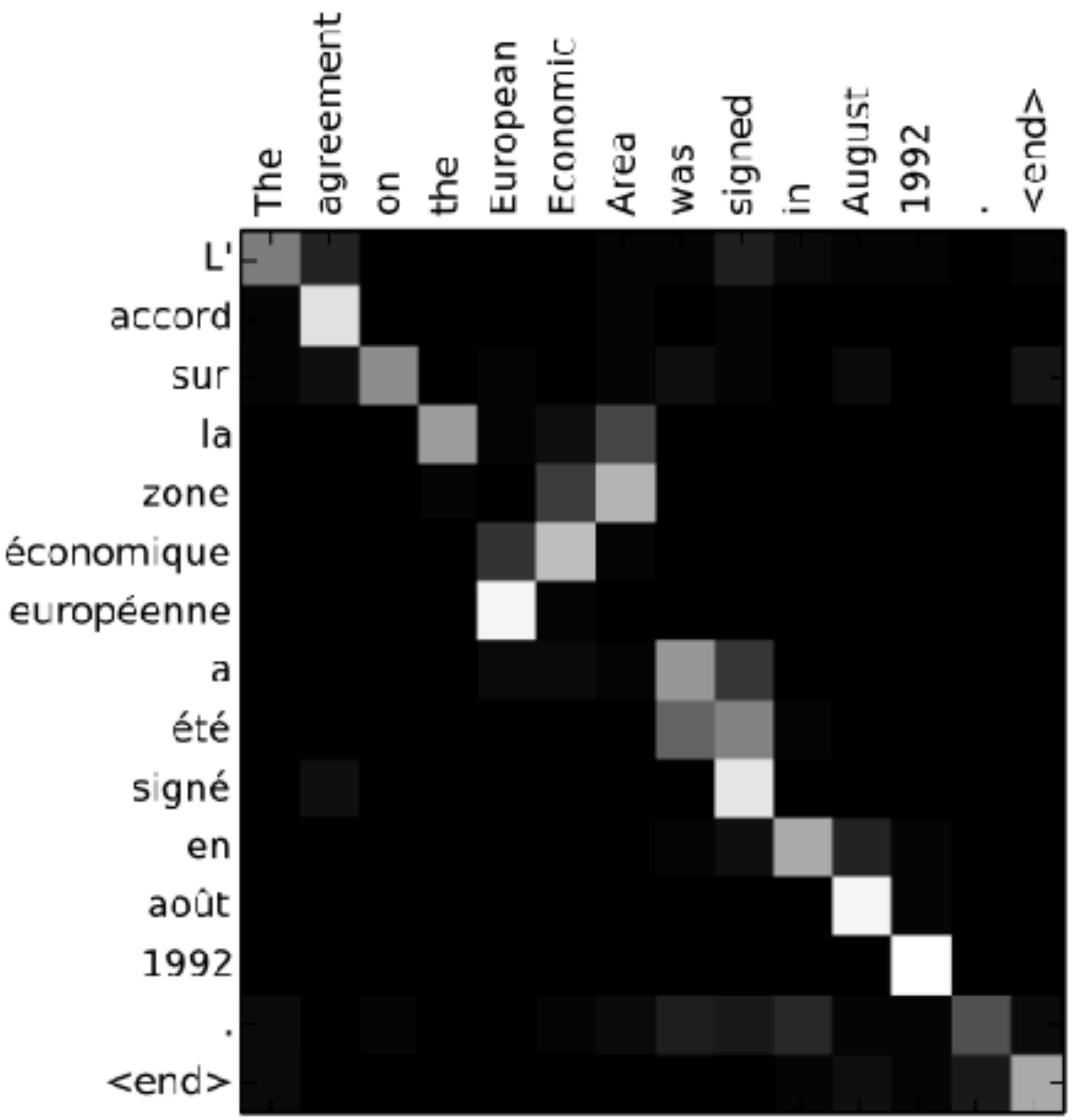
$\mathbf{X} = \mathbf{W}\mathbf{F}$

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

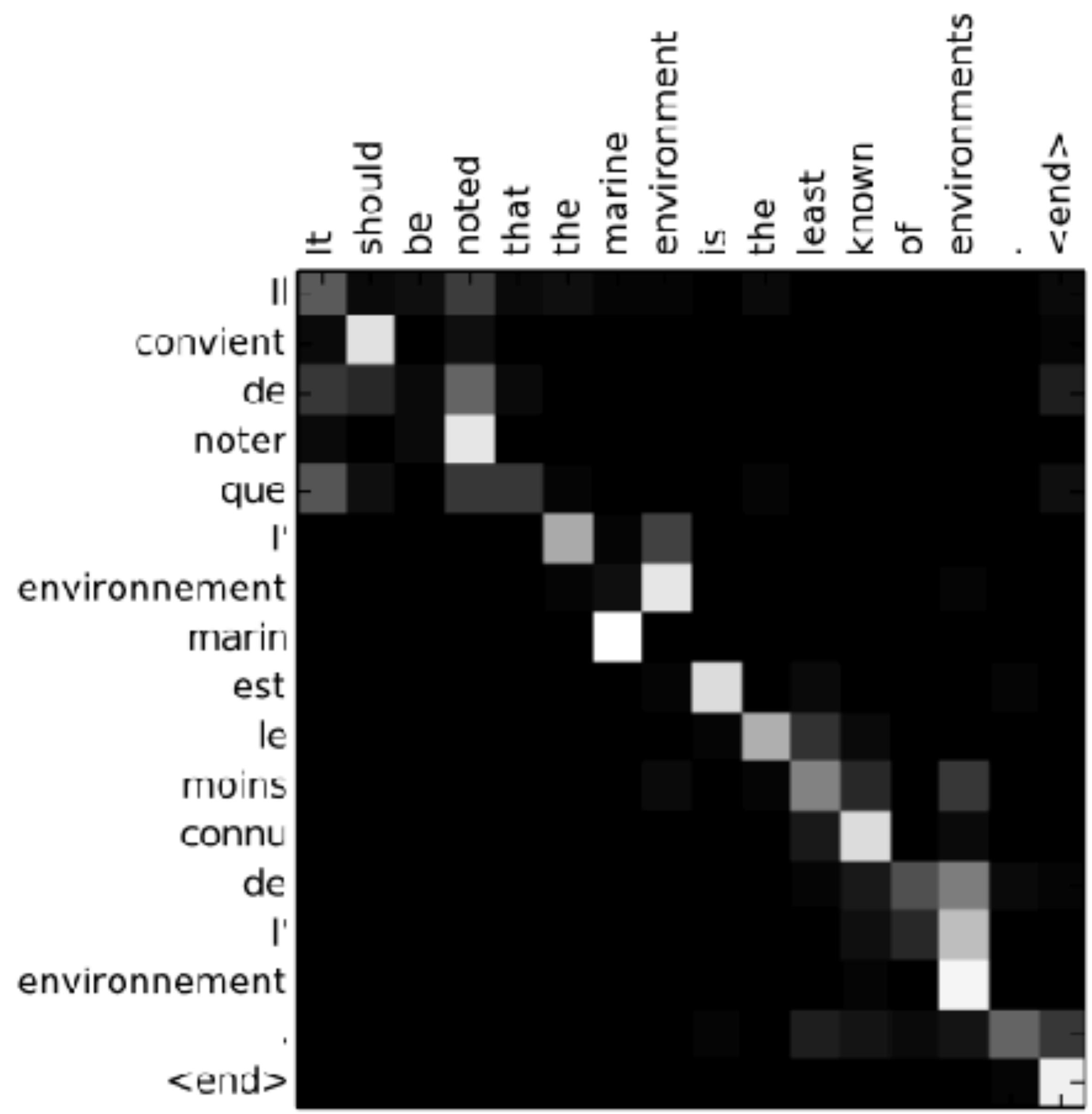
$$
\begin{aligned}
& t = t + 1 \\
& \left.\begin{aligned}
\mathbf{r}_t &= \mathbf{V}\mathbf{s}_{t-1} \\
\mathbf{u}_t &= \mathbf{v}^\top \tanh(\mathbf{X} + \mathbf{r}_t) \\
\mathbf{a}_t &= \text{softmax}(\mathbf{u}_t)
\end{aligned}\right\} \text{(Compute attention; part 2 of lecture)} \\
& \mathbf{c}_t = \mathbf{F}\mathbf{a}_t \\
& \mathbf{s}_t = \text{RNN}(\mathbf{s}_{t-1}, [\mathbf{e}_{t-1}; \mathbf{c}_t]) \qquad (\mathbf{e}_{t-1} \text{ is a learned embedding of } e_t) \\
& \mathbf{y}_t = \text{softmax}(\mathbf{P}\mathbf{s}_t + \mathbf{b}) \qquad (\mathbf{P} \text{ and } \mathbf{b} \text{ are learned parameters}) \\
& e_t \mid \boldsymbol{e}_{<t} \sim \text{Categorical}(\mathbf{y}_t)
\end{aligned}
$$

Attention in MT: Results

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

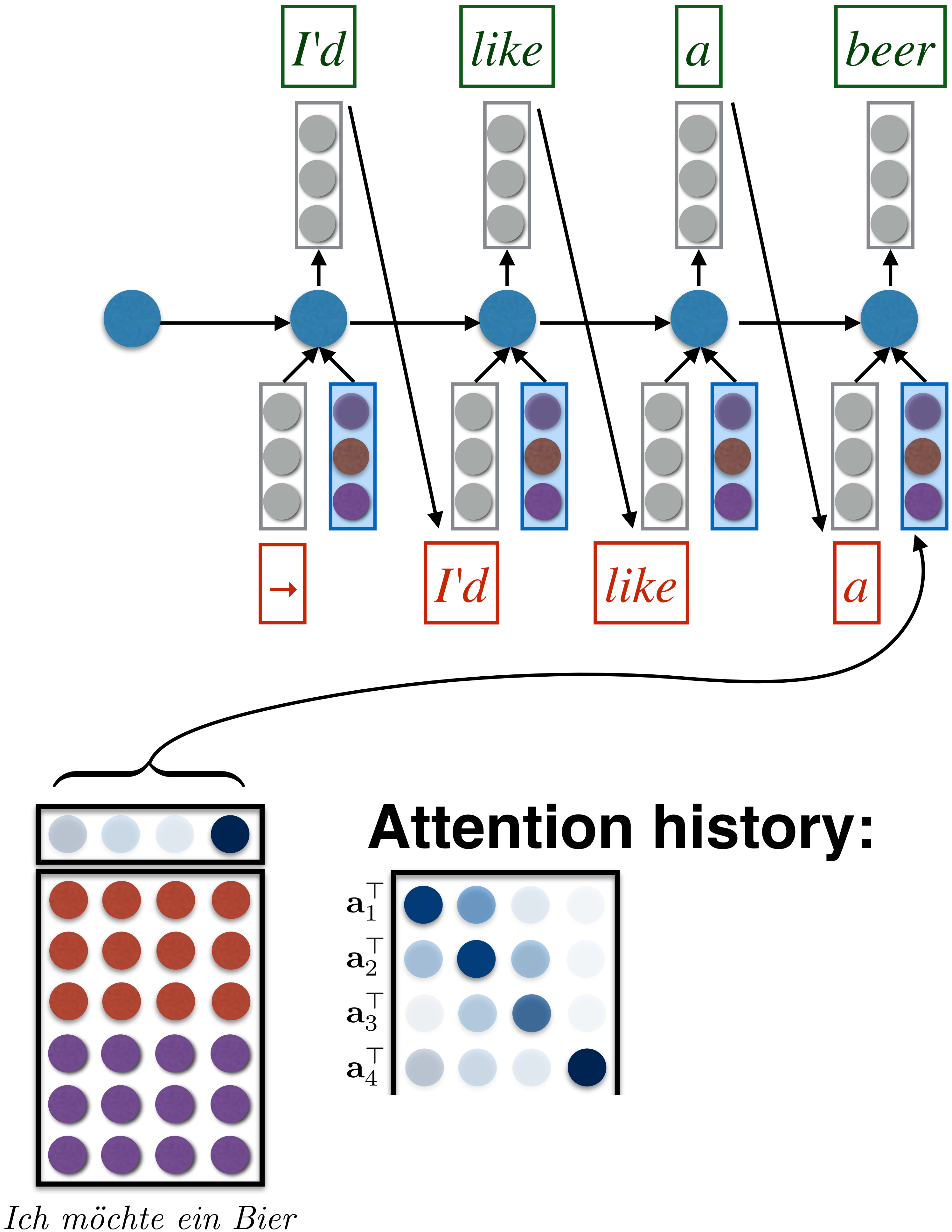

The
agreement
on
the
European
Economic
Area
was
signed
in
August
1992
.
<end>
L'
accord
sur
la
zone
économique
européenne
a
été
signé
en
août
1992
.
<end>

(a)

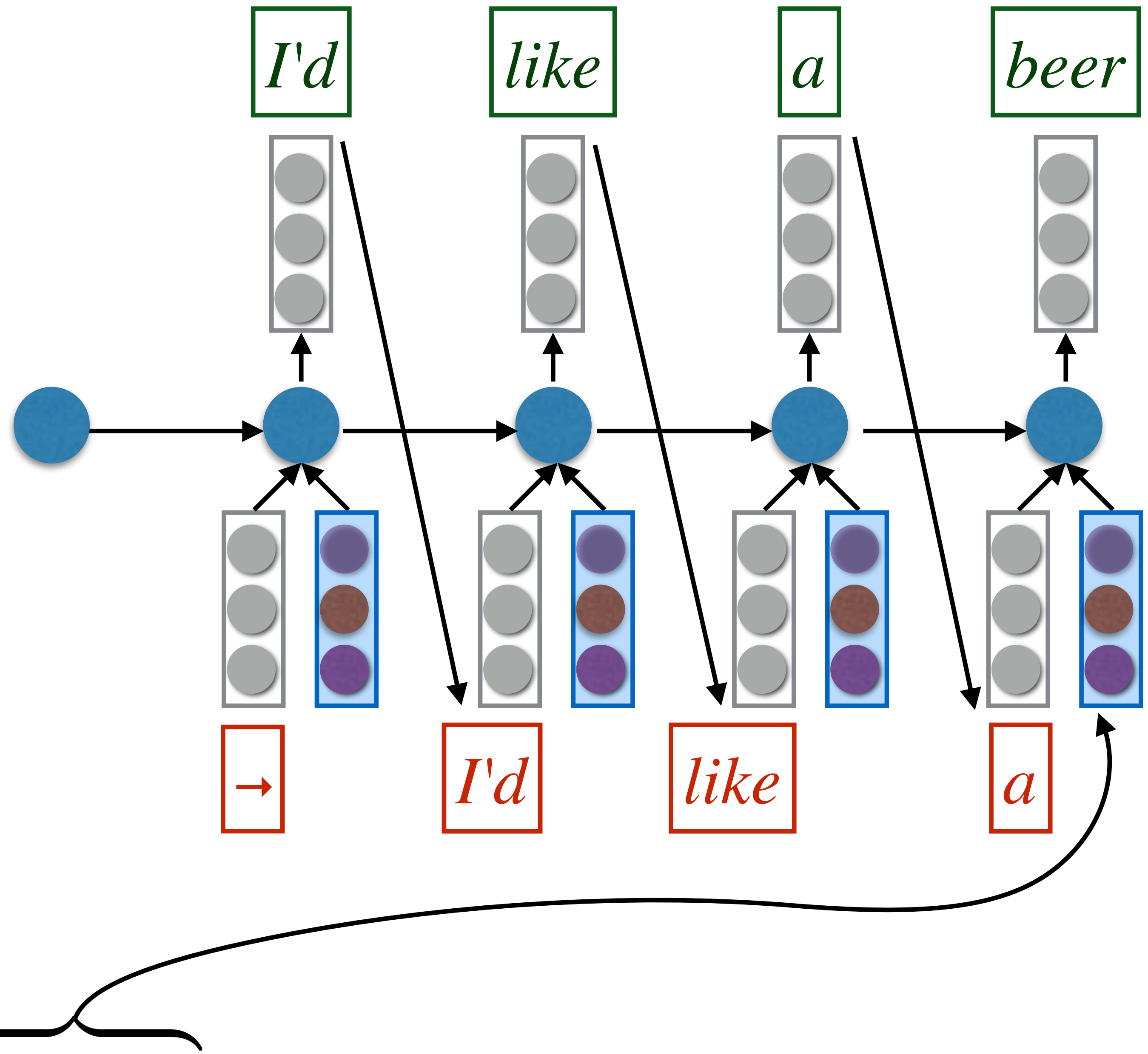

It
should
be
noted
that
the
marine
environment
is
the
least
known
of
environments
.
<end>
Il
convient
de
noter
que
l'
environnement
marin
est
le
moins
connu
de
l'
environnement
.
<end>

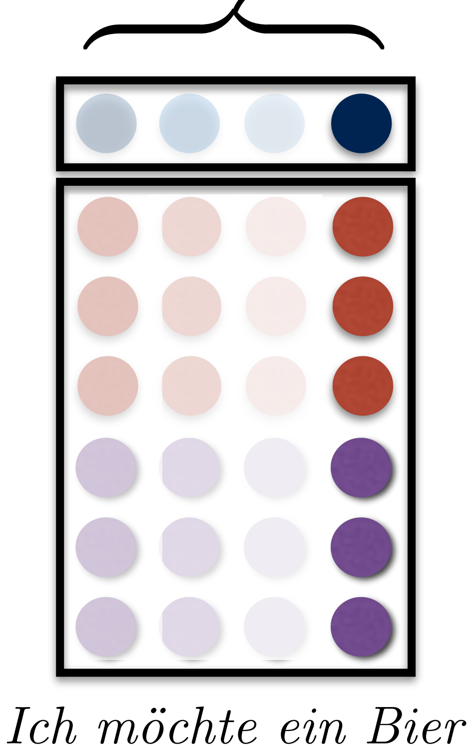
(b)

A word about gradients

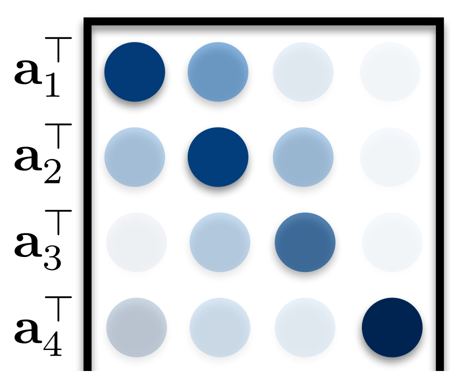


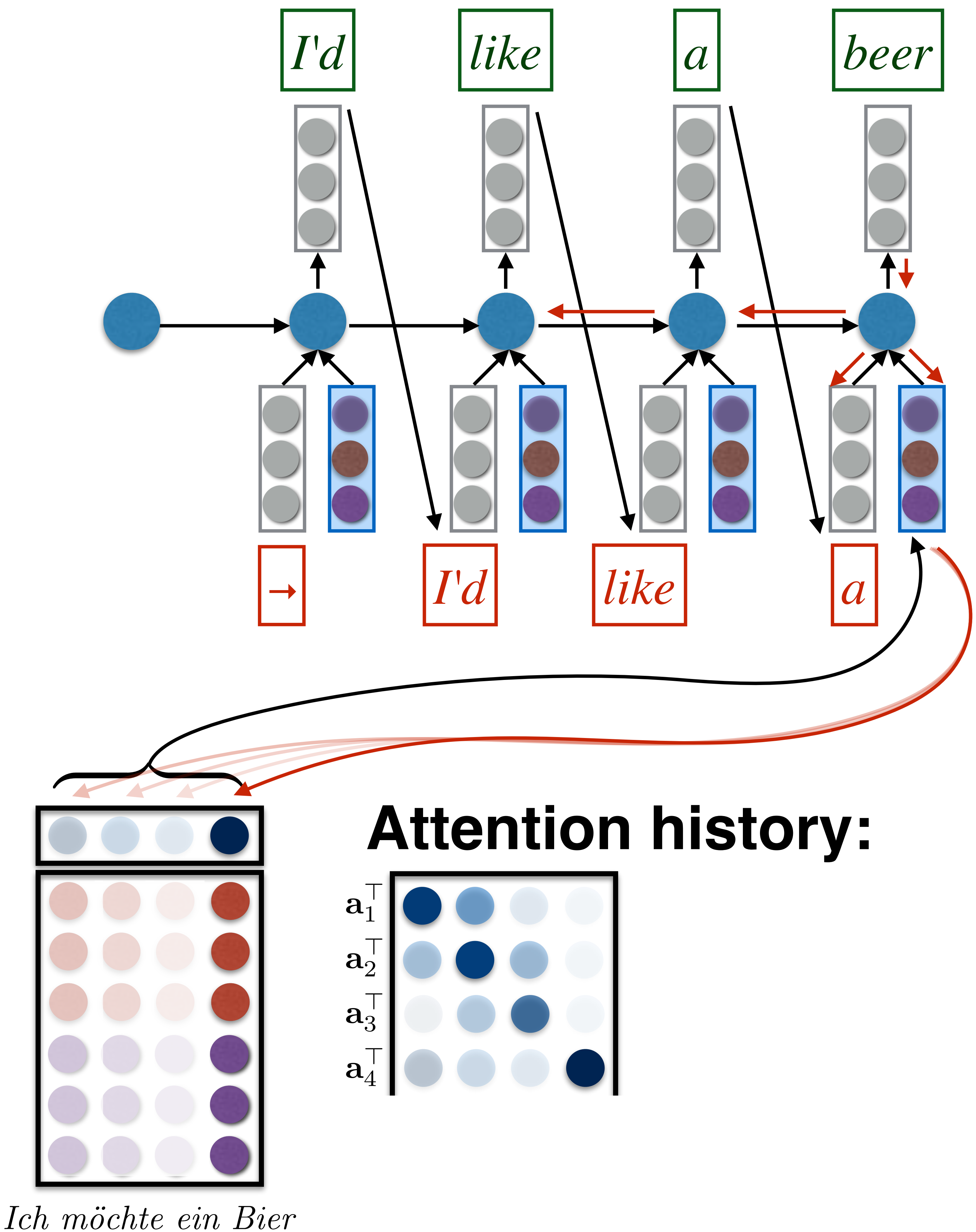
Attention history:





Attention history:





I'd
like
a
beer
→
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 and translation

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

Summary

- Attention
  - provides the ability to establish information flow directly from distant
  - closely related to “pooling” operations in convnets (and other architectures)
- Traditional attention model seems to only cares about “content”
  - No obvious bias in favor of diagonals, short jumps, fertility, etc.
  - Some work has begun to add other “structural” biases (Luong et al., 2015; Cohn et al., 2016), but there are lots more opportunities
  - Factorization into keys and values (Miller et al., 2016; Ba et al., 2016, Gulcehre et al., 2016)
- Attention weights provide interpretation you can look at

Questions?