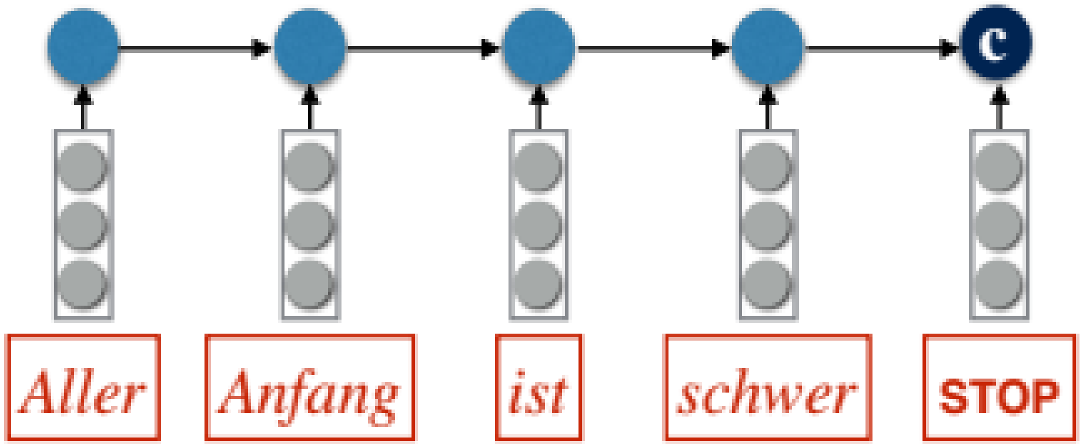
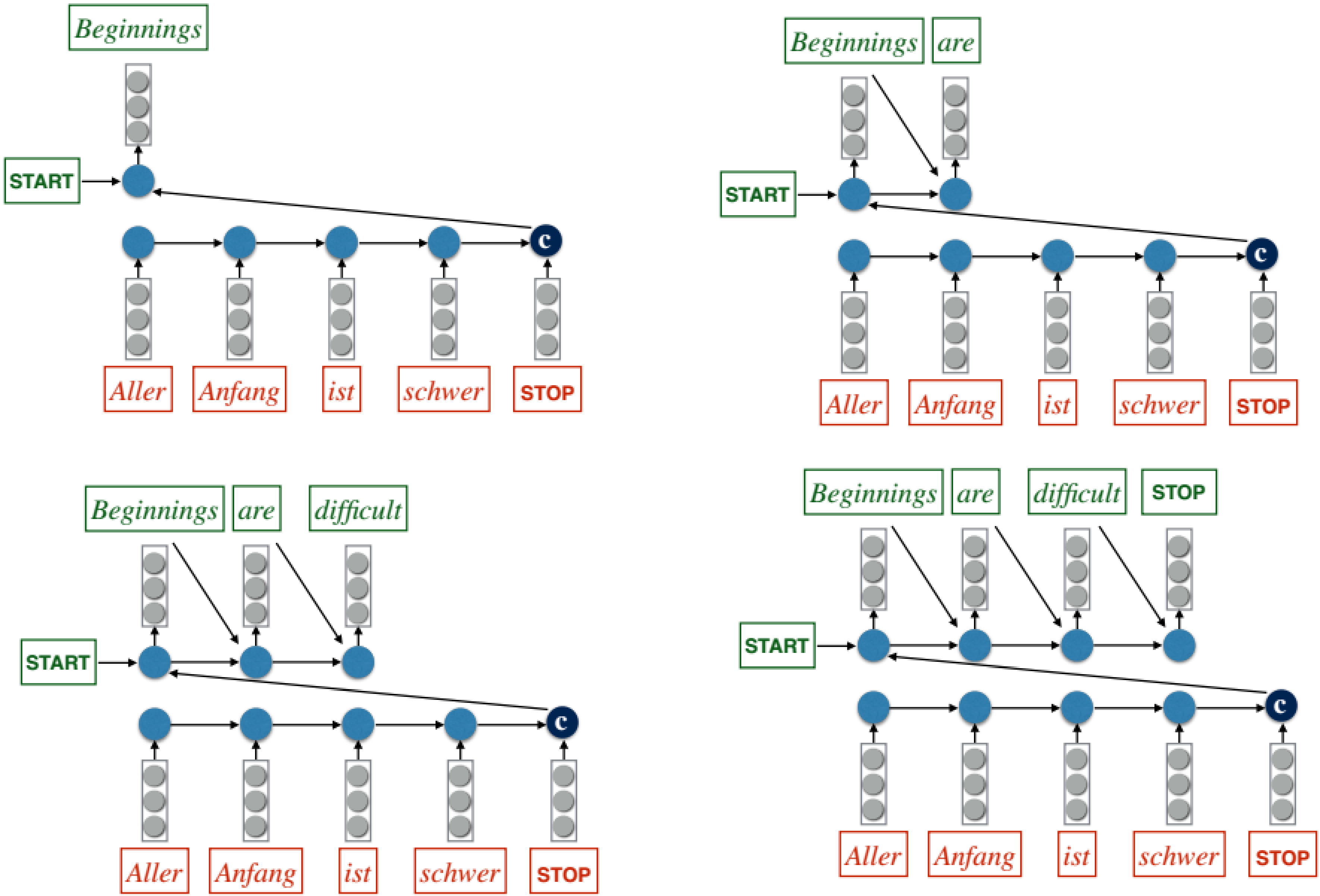
Machine translation models

Seq2Seq model1: Embed, Encode

Sentence encoder: fixed size vector



Decode from encoded vector



Pros & Cons

- **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!

Seq2Seq model2: Embed, Encode & Attention

Sentence encoding as fixed size vector

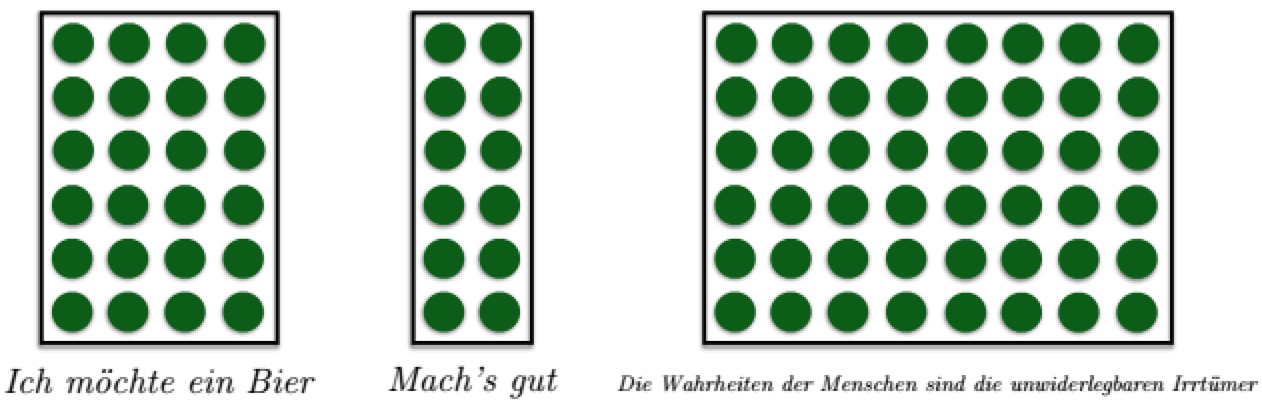
We are compressing a lot of information in a finite-sized vector.

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

What is to be done?

Sentence encoding as matrix

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



*Each matrix has fixed number of rows, but number of columns depends on the number of words

Sentence encoding as matrix

I. Using word vectors alone

II. Using CNN

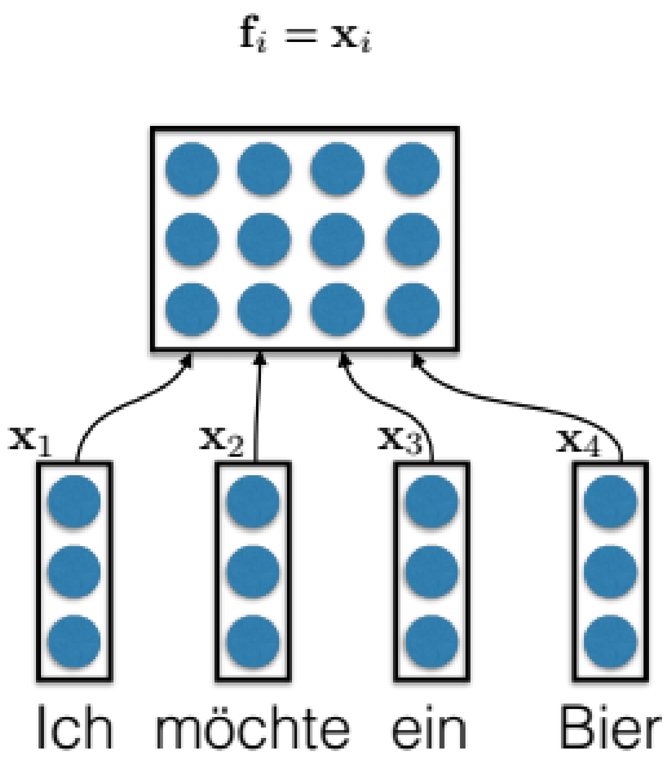
III. Using RNN/Bi-directional RNN

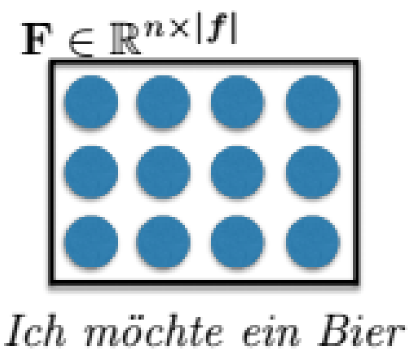
I. Sentence matrix with word vectors alone

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

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

I. Sentence matrix with word vectors alone

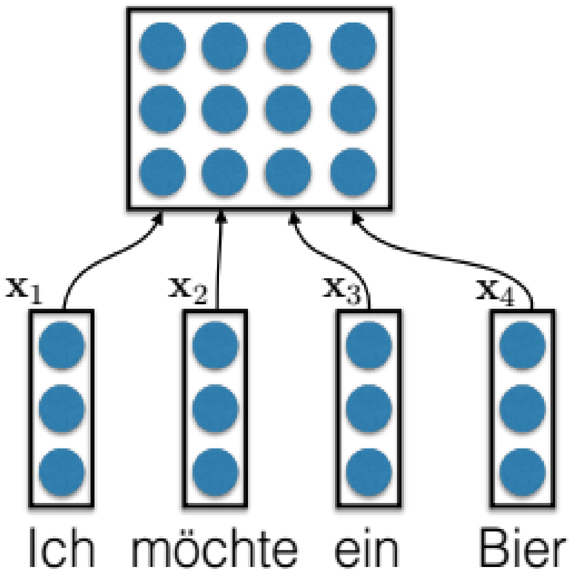


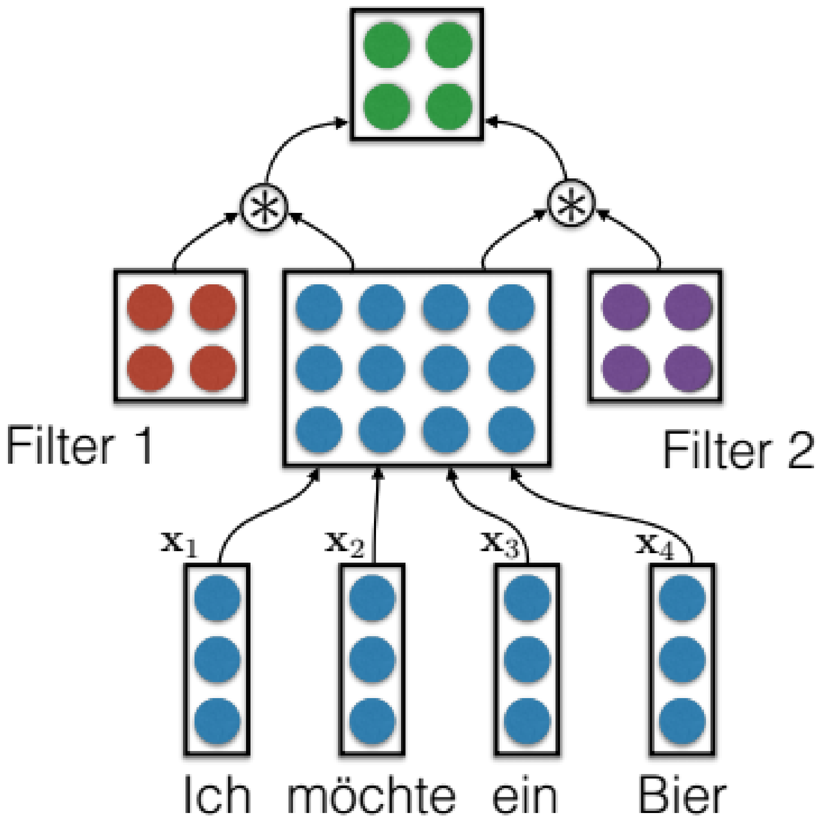


II. Sentence matrix with CNN

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

II. Sentence matrix with CNN





III. Sentence matrix with Bi-RNN

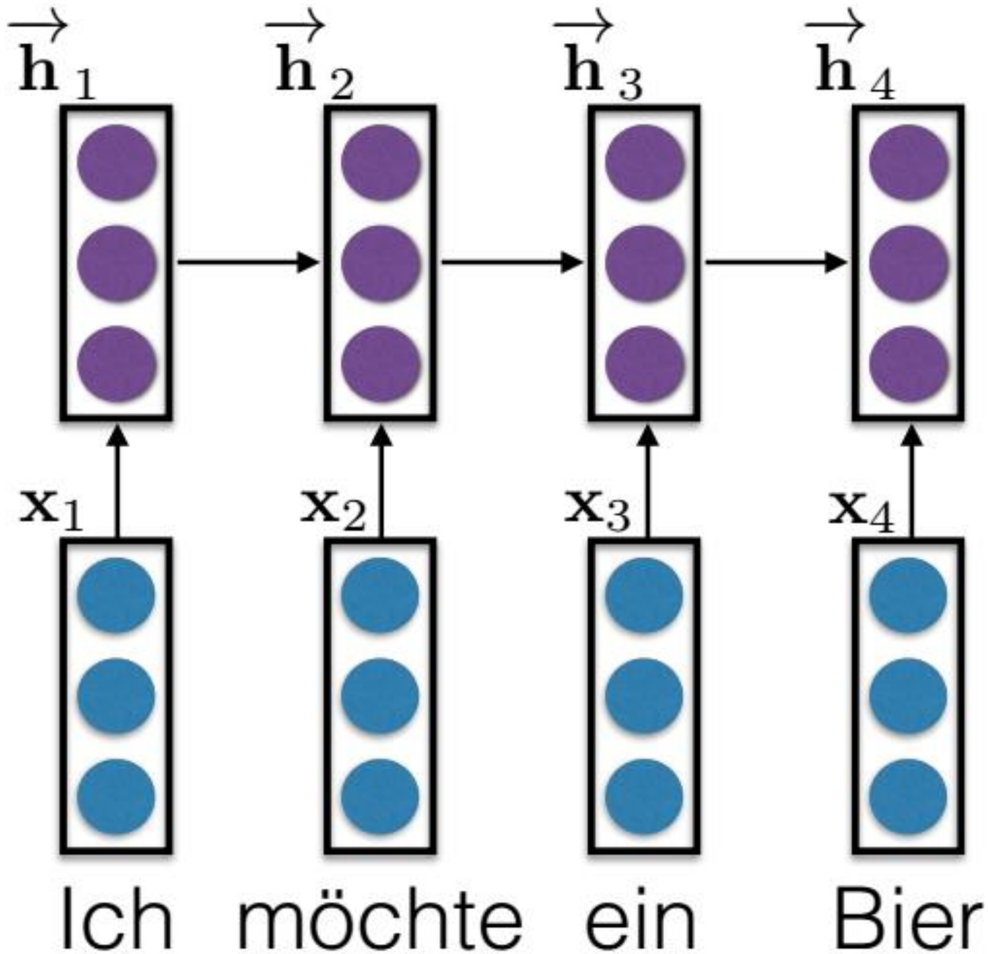
- By far the most widely used matrix representation, due to Bahdanau et al (2015)

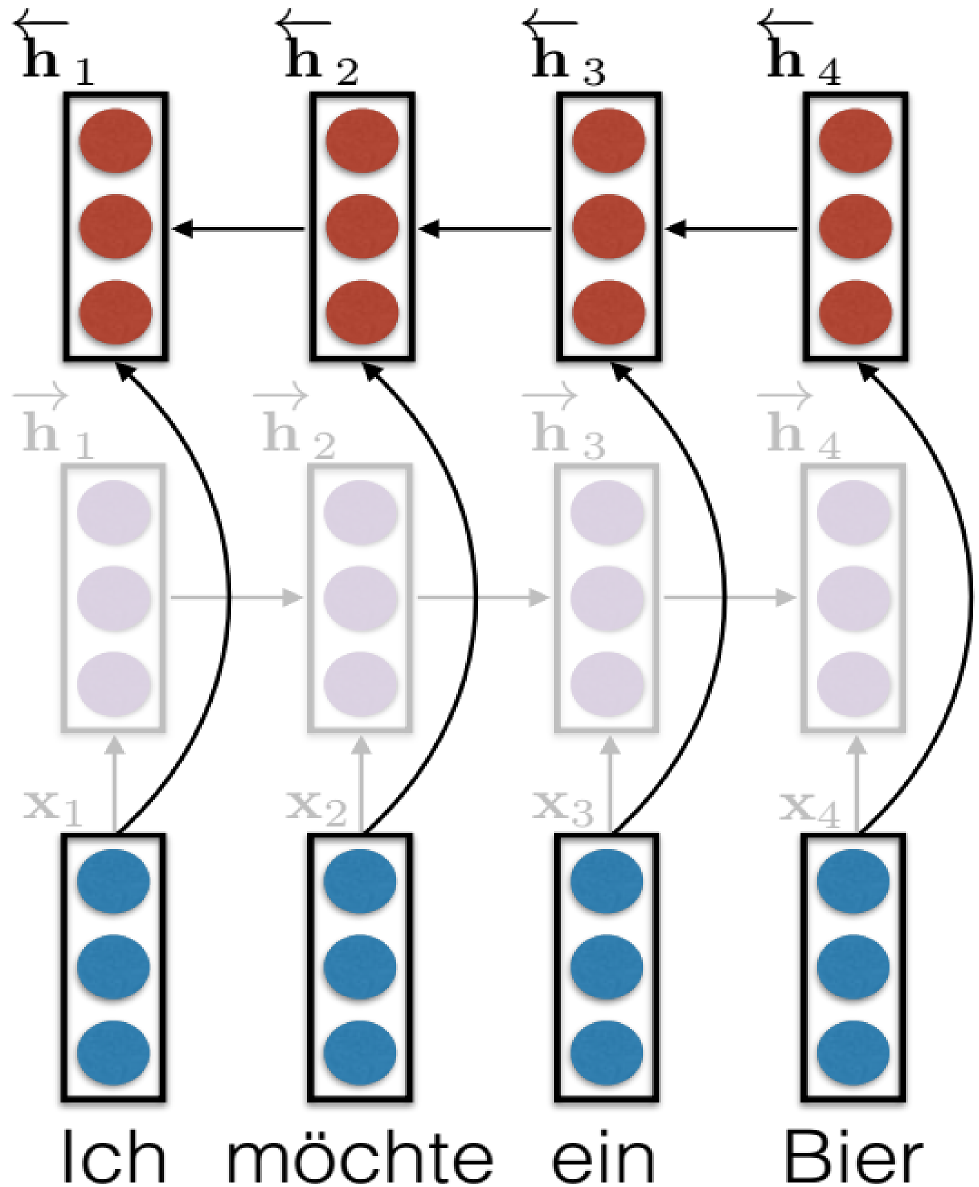
- One column per word

- 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

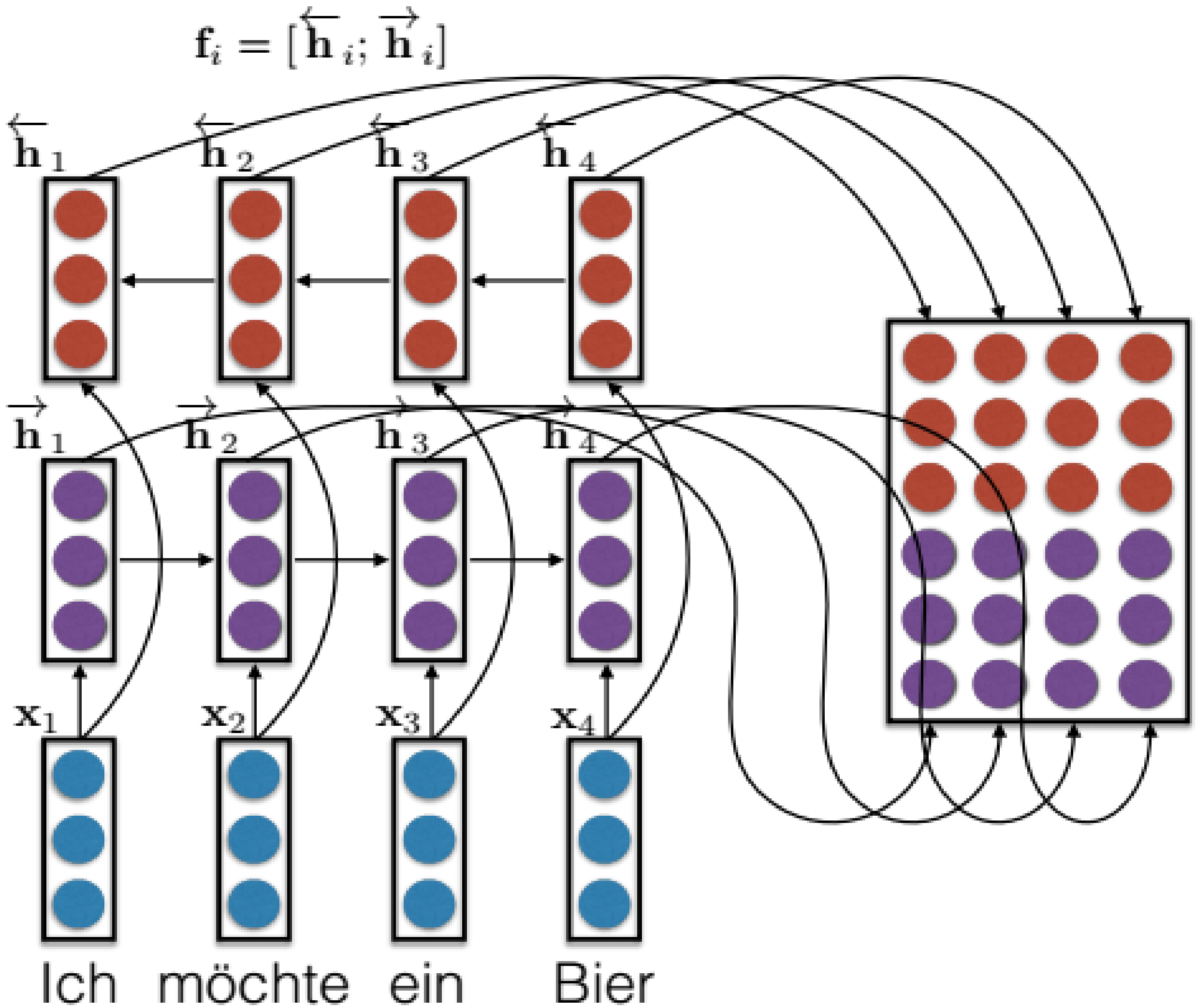
- Implementation: bidirectional RNNs (GRUs or LSTMs) to read f from left to right and right to left, concatenate representations

III. Sentence matrix with Bi-RNN

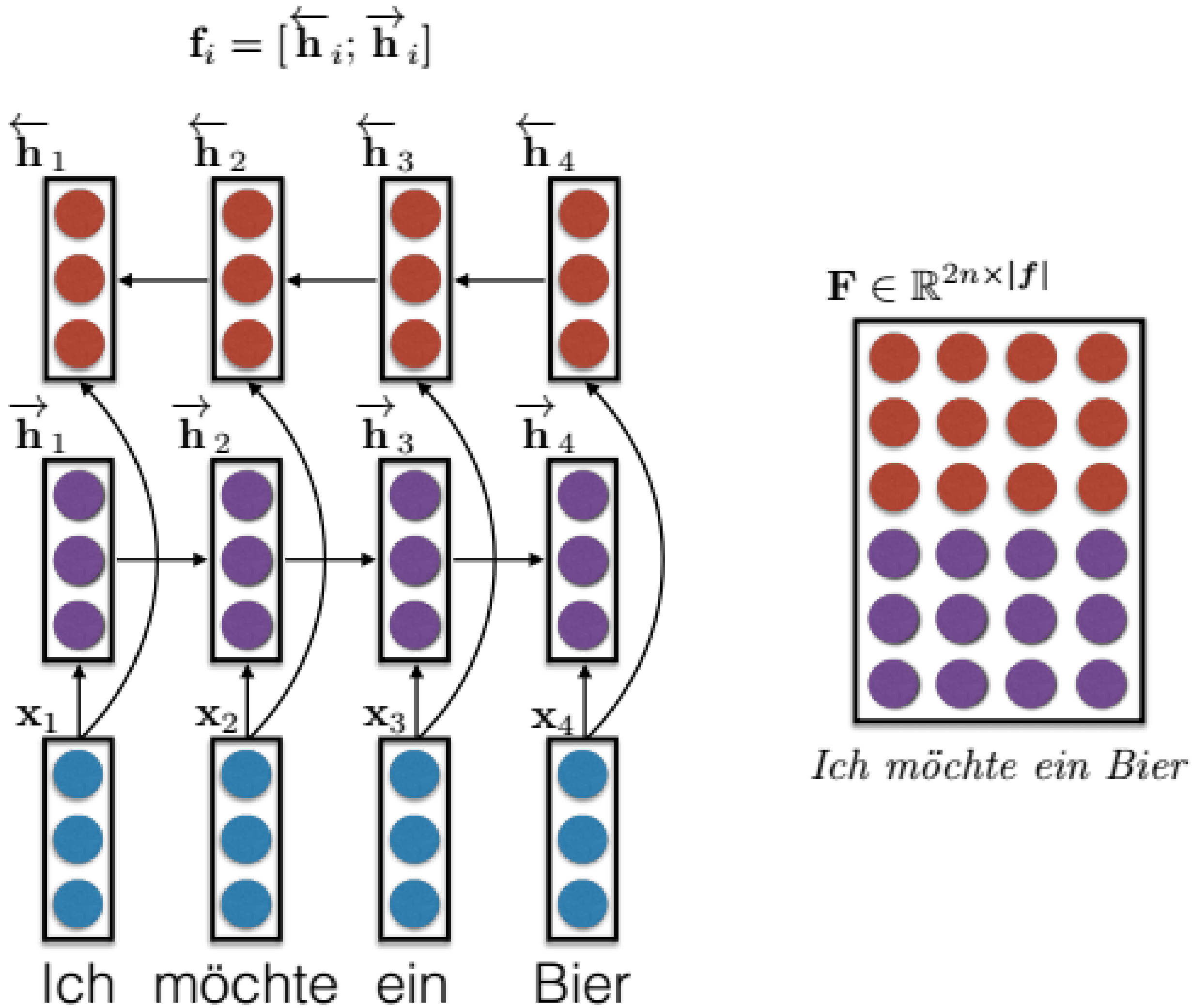




$\overleftarrow{\mathbf{h}}_1$ $\overleftarrow{\mathbf{h}}_2$ $\overleftarrow{\mathbf{h}}_3$ $\overleftarrow{\mathbf{h}}_4$
$\overrightarrow{\mathbf{h}}_1$ $\overrightarrow{\mathbf{h}}_2$ $\overrightarrow{\mathbf{h}}_3$ $\overrightarrow{\mathbf{h}}_4$
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$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

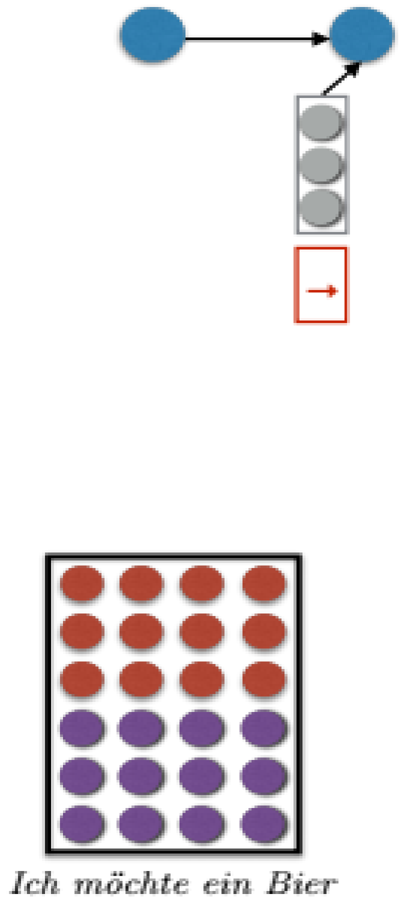


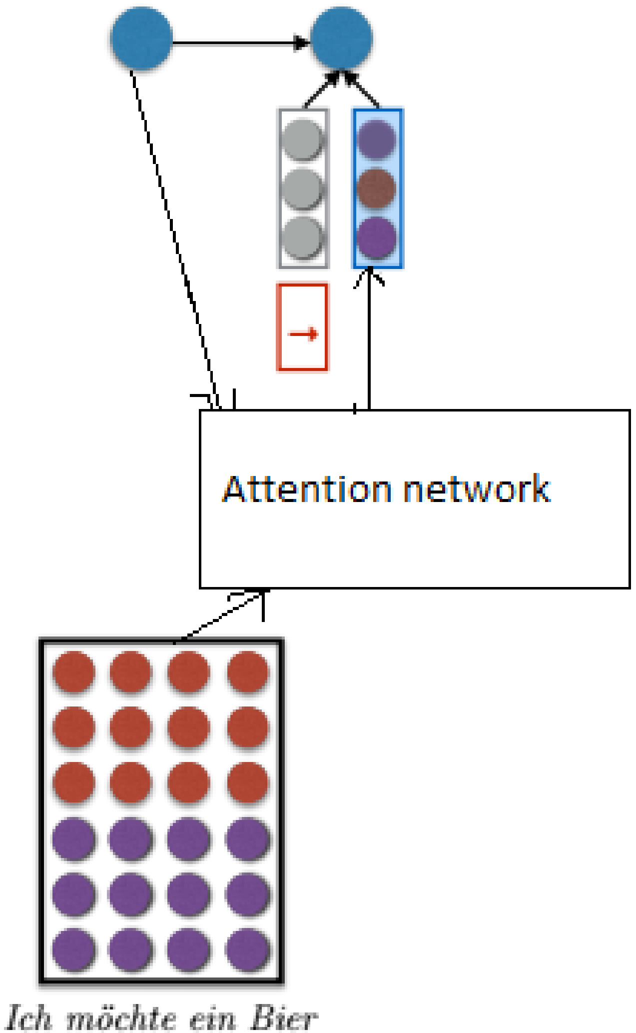
$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$
$\overleftarrow{\mathbf{h}}_1$
$\overleftarrow{\mathbf{h}}_2$
$\overleftarrow{\mathbf{h}}_3$
$\overleftarrow{\mathbf{h}}_4$
$\overrightarrow{\mathbf{h}}_1$
$\overrightarrow{\mathbf{h}}_2$
$\overrightarrow{\mathbf{h}}_3$
$\overrightarrow{\mathbf{h}}_4$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

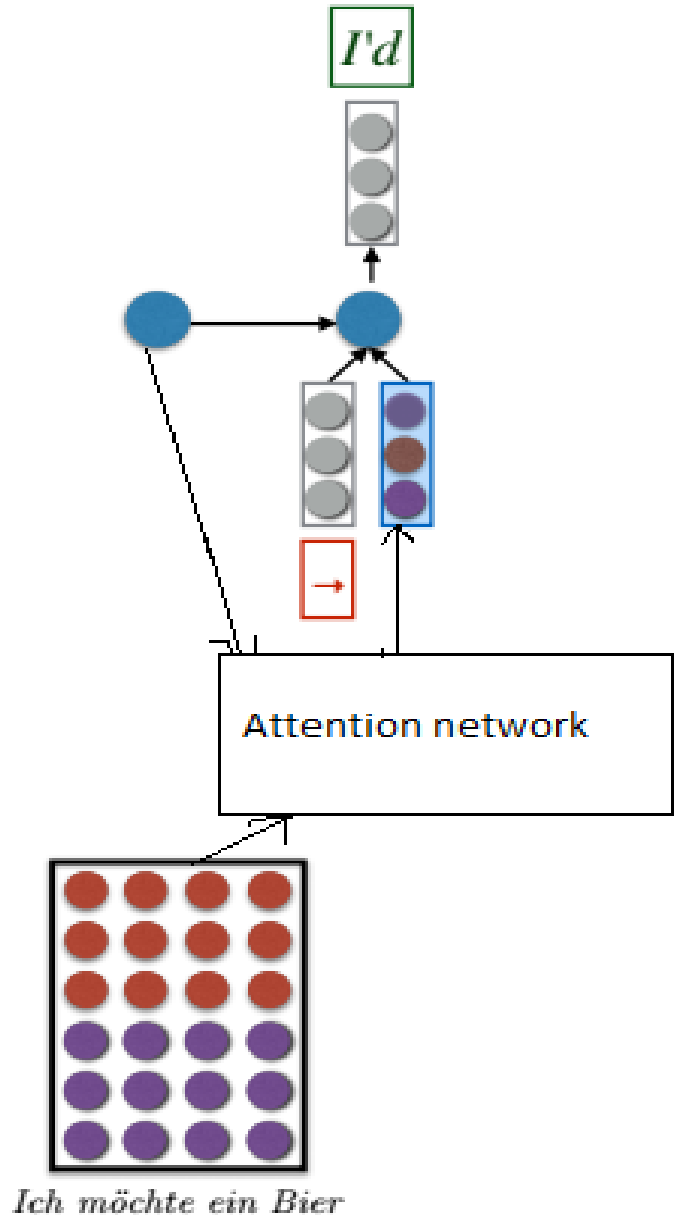


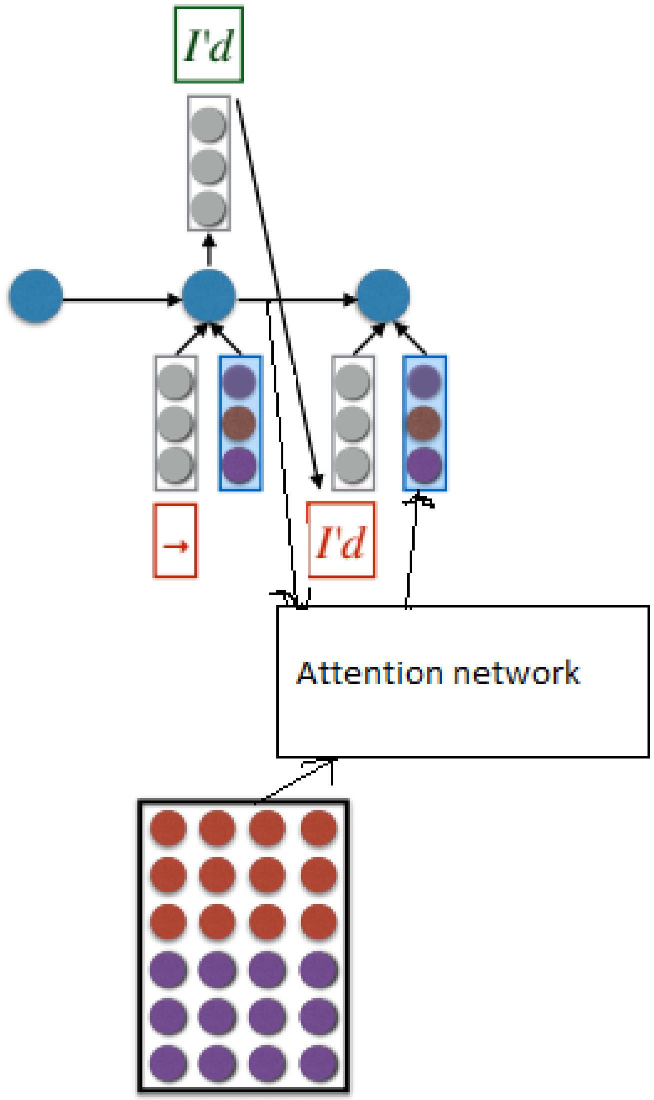
$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$
$\overleftarrow{\mathbf{h}}_1$ $\overleftarrow{\mathbf{h}}_2$ $\overleftarrow{\mathbf{h}}_3$ $\overleftarrow{\mathbf{h}}_4$
$\overrightarrow{\mathbf{h}}_1$ $\overrightarrow{\mathbf{h}}_2$ $\overrightarrow{\mathbf{h}}_3$ $\overrightarrow{\mathbf{h}}_4$
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$
Ich möchte ein Bier
$\mathbf{F} \in \mathbb{R}^{2n \times |f|}$
Ich möchte ein Bier

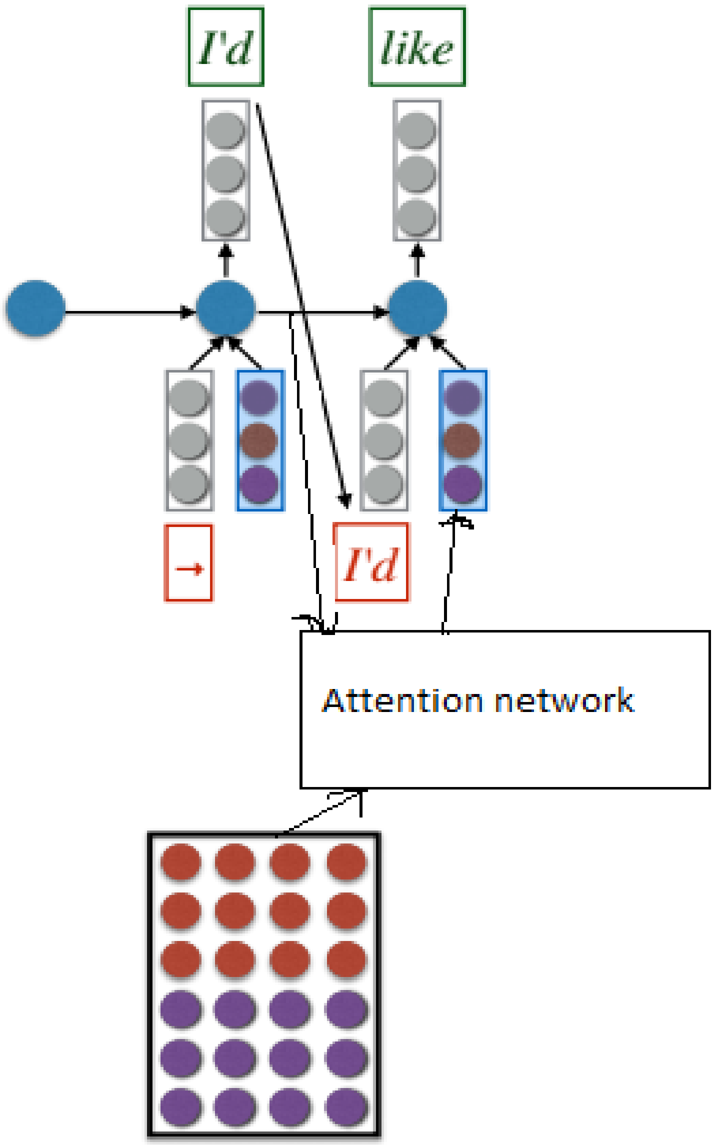
Decoder with Attention

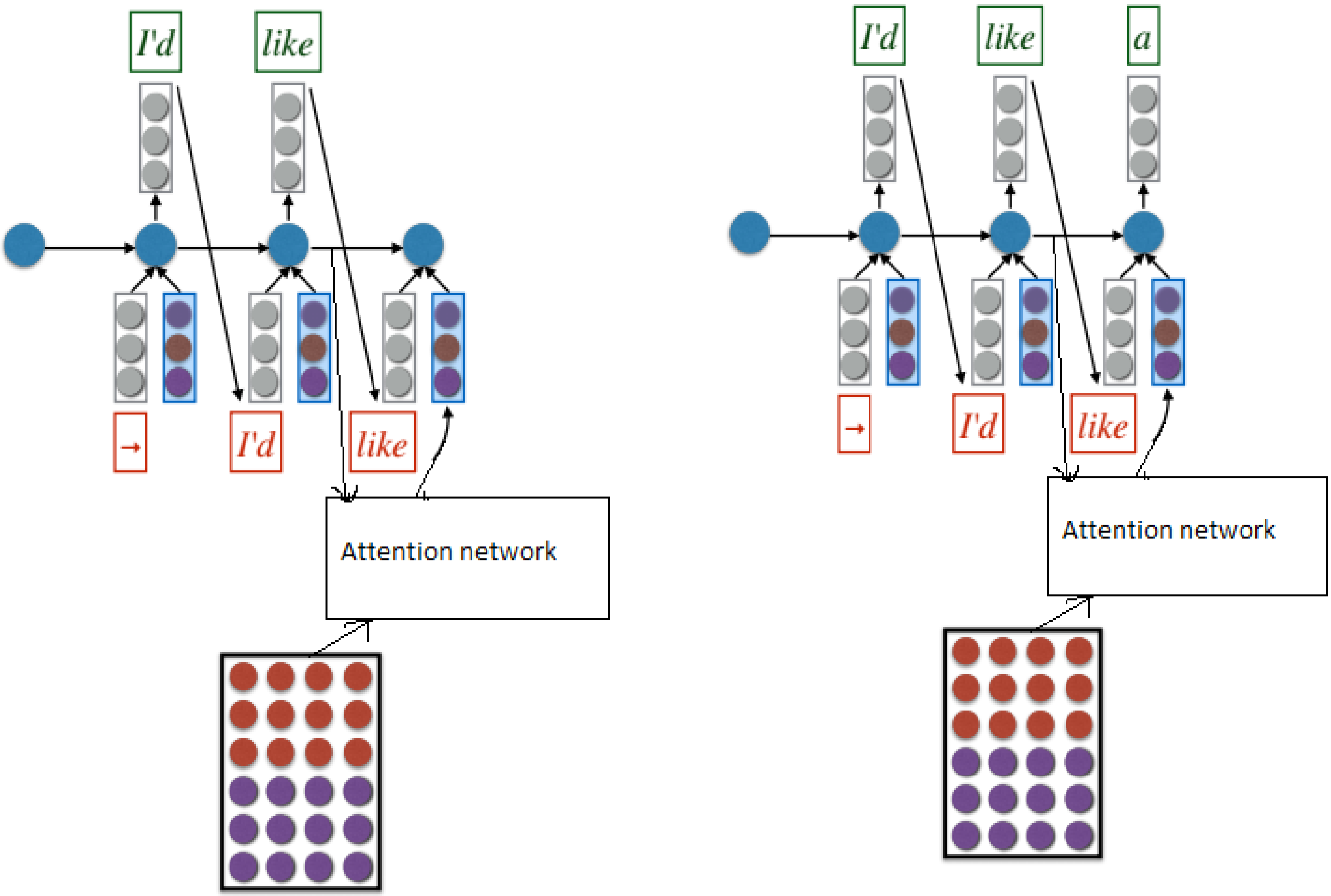



Attention network
Ich möchte ein Bier

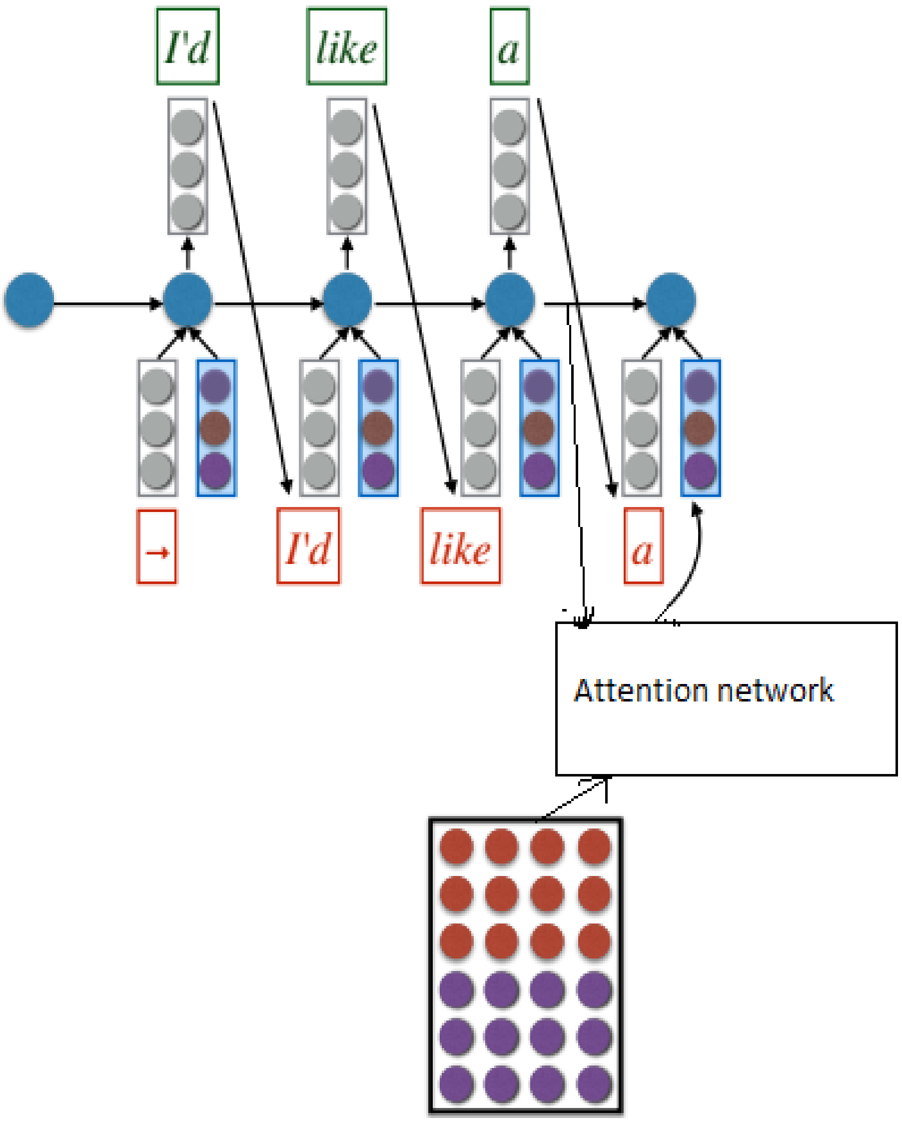
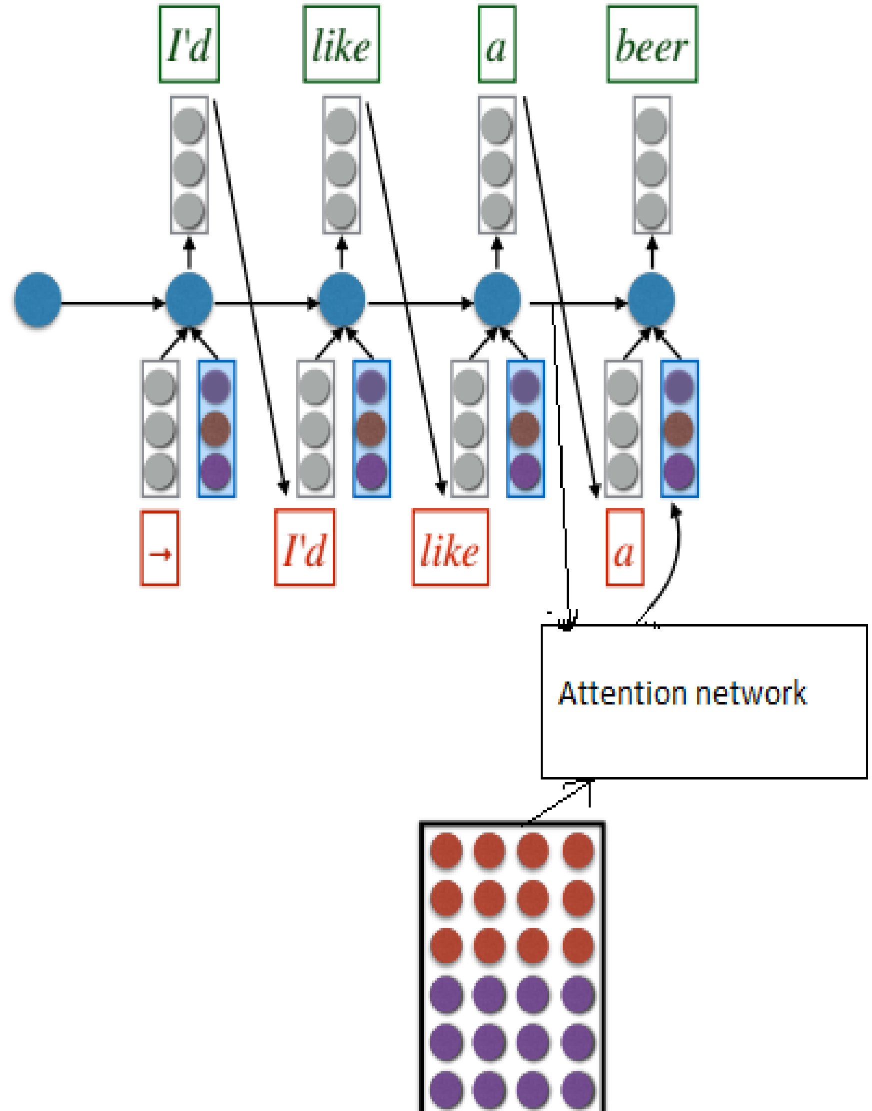

I'd
Attention network
Ich möchte ein Bier

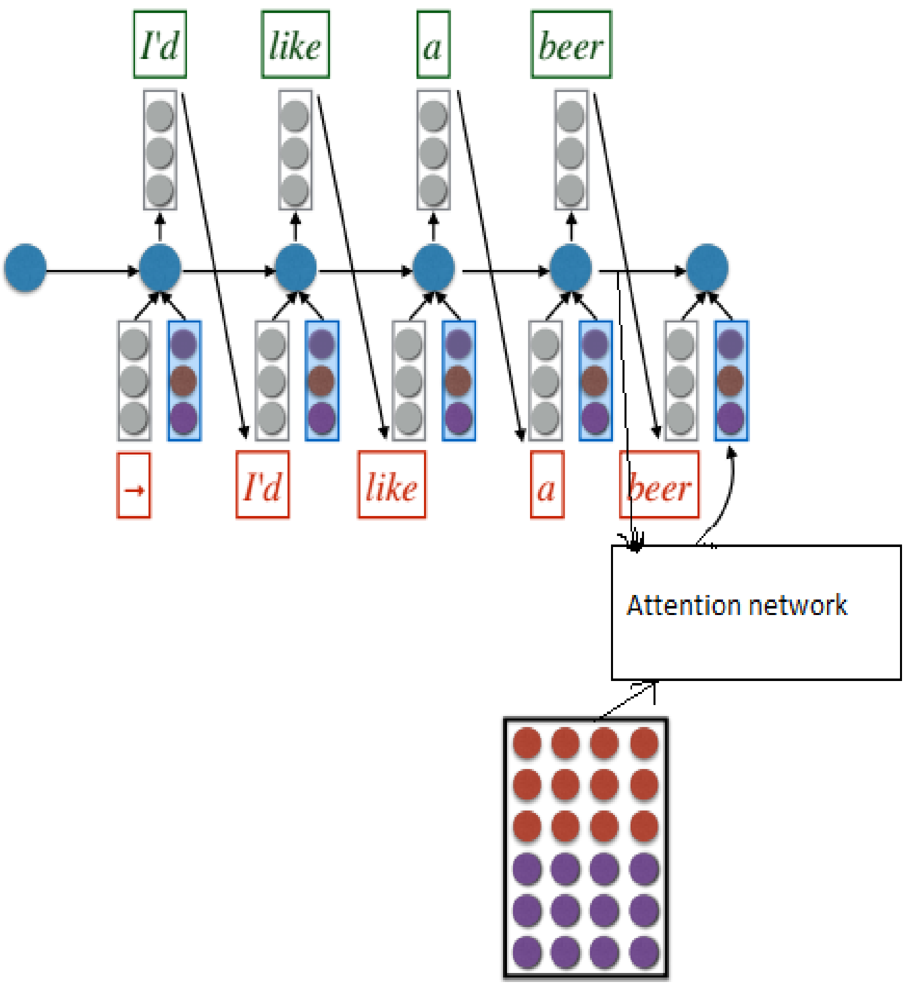

I'd
I'd
Attention network

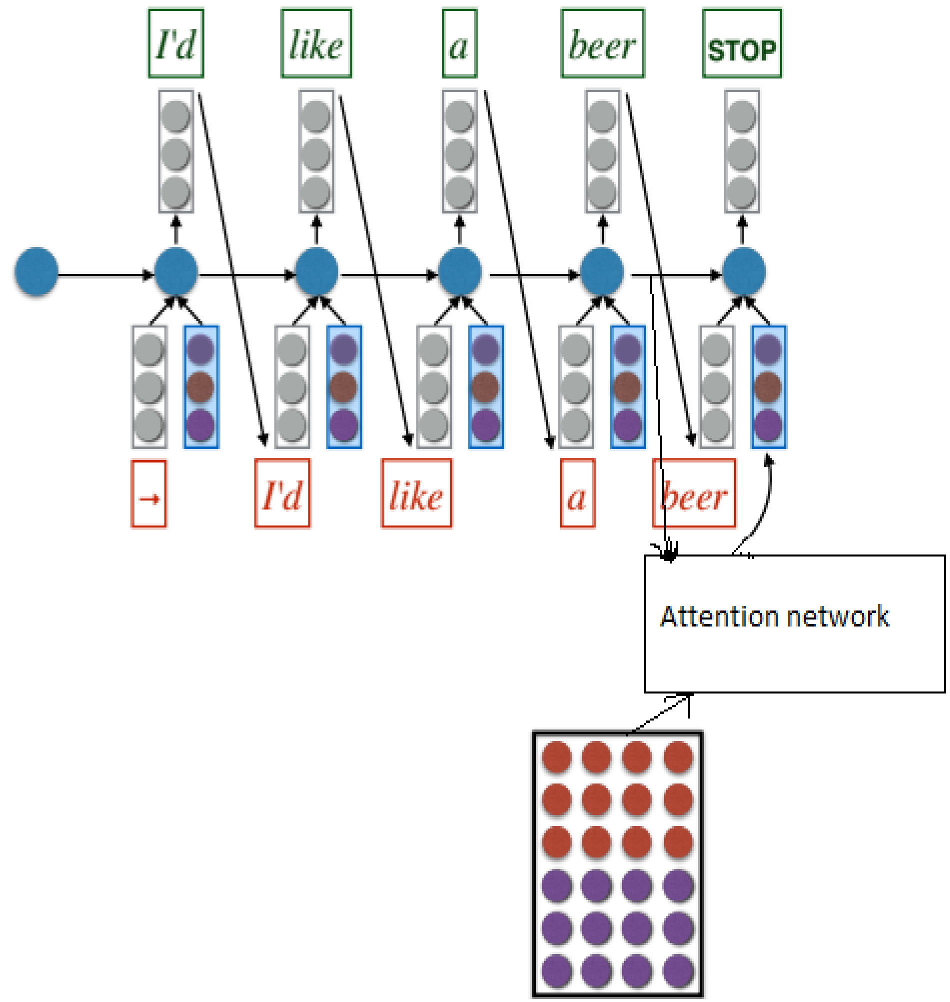

I'd
like
I'd
Attention network



I'd
like
→
I'd
like
Attention network
I'd
like
a
→
I'd
like
Attention network


I'd
like
a
→
I'd
like
a
Attention network

I'd
like
a
beer
→
I'd
like
a
Attention network


I'd
like
a
beer
→
I'd
like
a
beer
Attention network


I'd
like
a
beer
STOP
→
I'd
like
a
beer
Attention network

Decoding with seq2seq model

Decoding: Greedy search

Simply take the best prediction of the previous timestep as the input to the current timestep.

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

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

$$\vdots$$

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

Decoding: Greedy search

- The problem with greedy search, as its name suggested, is that it may not be optimal. More concretely, the joint probability of the output sequence may not be highest possible.

- To do better than greedy search, one can do a full search. We use the recurrent networks to compute the joint probability of every possible output sequence. This can guarantee to find the best possible sequence with a large computation cost.

Decoding: Full search

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

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

This is, for general RNNs, a hard problem.

Decoding: Beam search

- Keep a list of b possible sequences sorted by the joint probability (which is computed by multiplying the output probability of each prediction in the sequence produced so far).

- During the decode procedure, any sequence that does not belong to the top-b highest joint probability will be removed from our list of candidates.

- Even though it's not guaranteed for the beam search to achieve the optimal sequence, in practice, the generated sequences in the top-b list are generally very good.

Decoding: Beam search

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

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

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

beer **drink** **I**

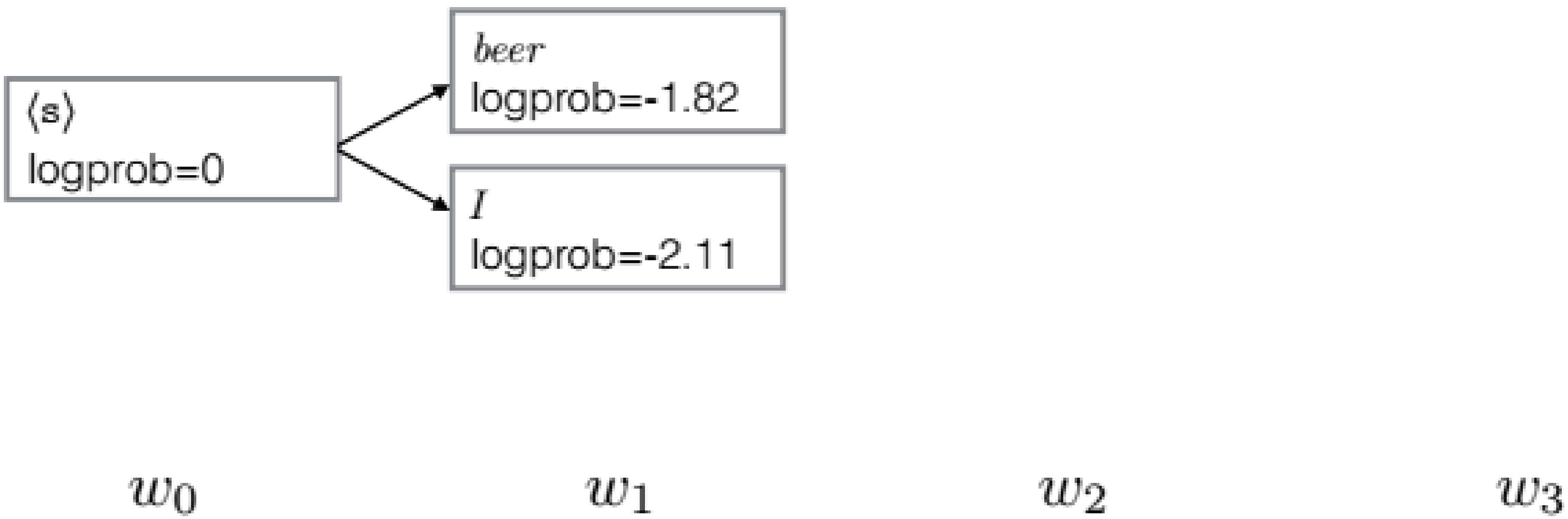
⟨s⟩ logprob=0

w_0 w_1 w_2 w_3

E.g., for b=2:

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

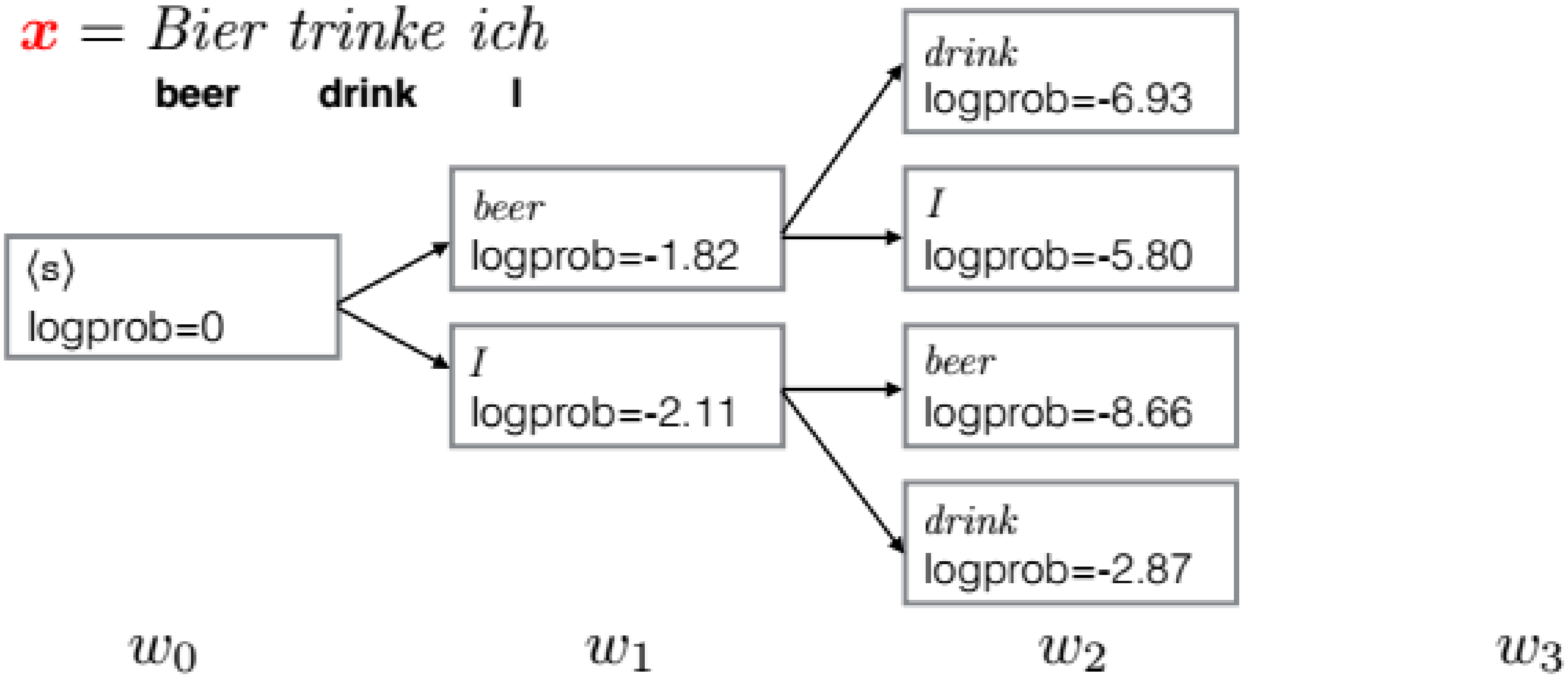
beer drink I



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

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

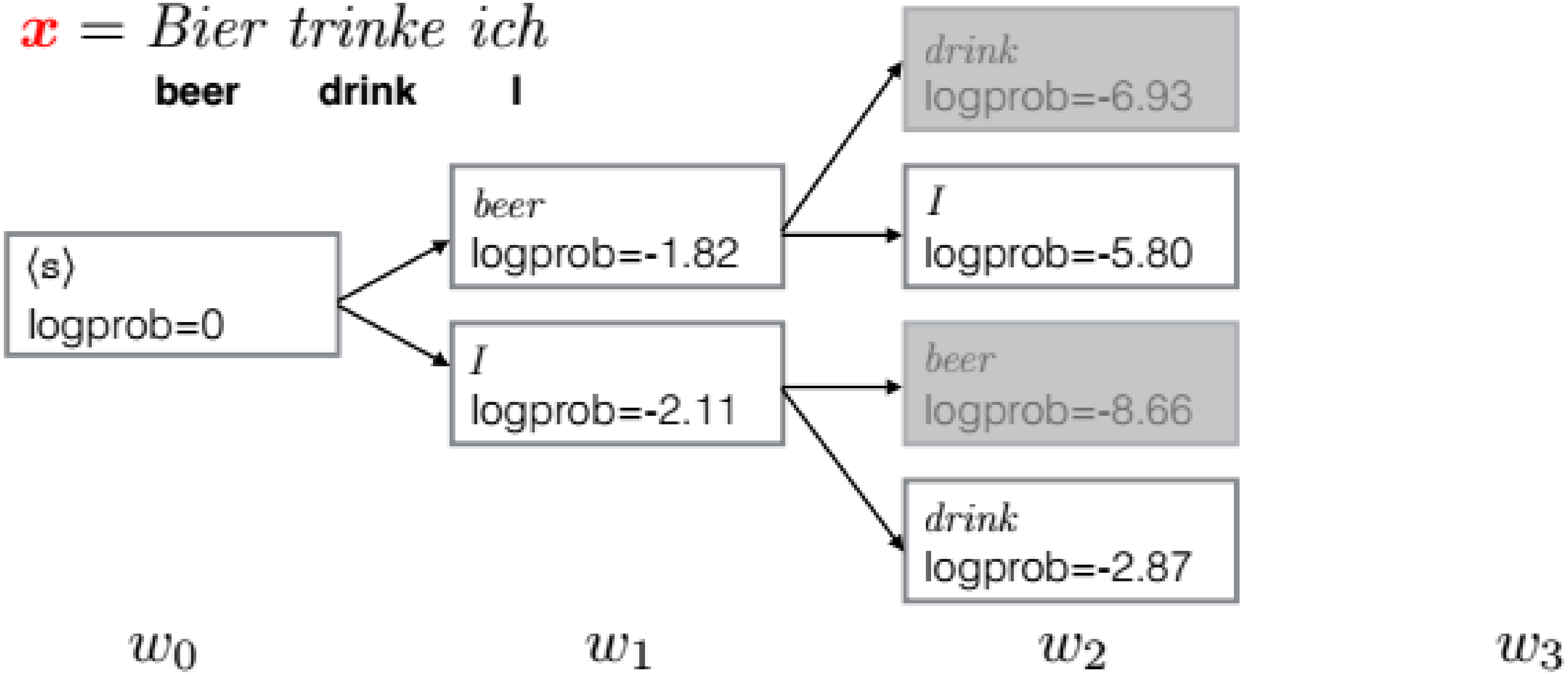
beer **drink** **I**



E.g., for b=2:

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

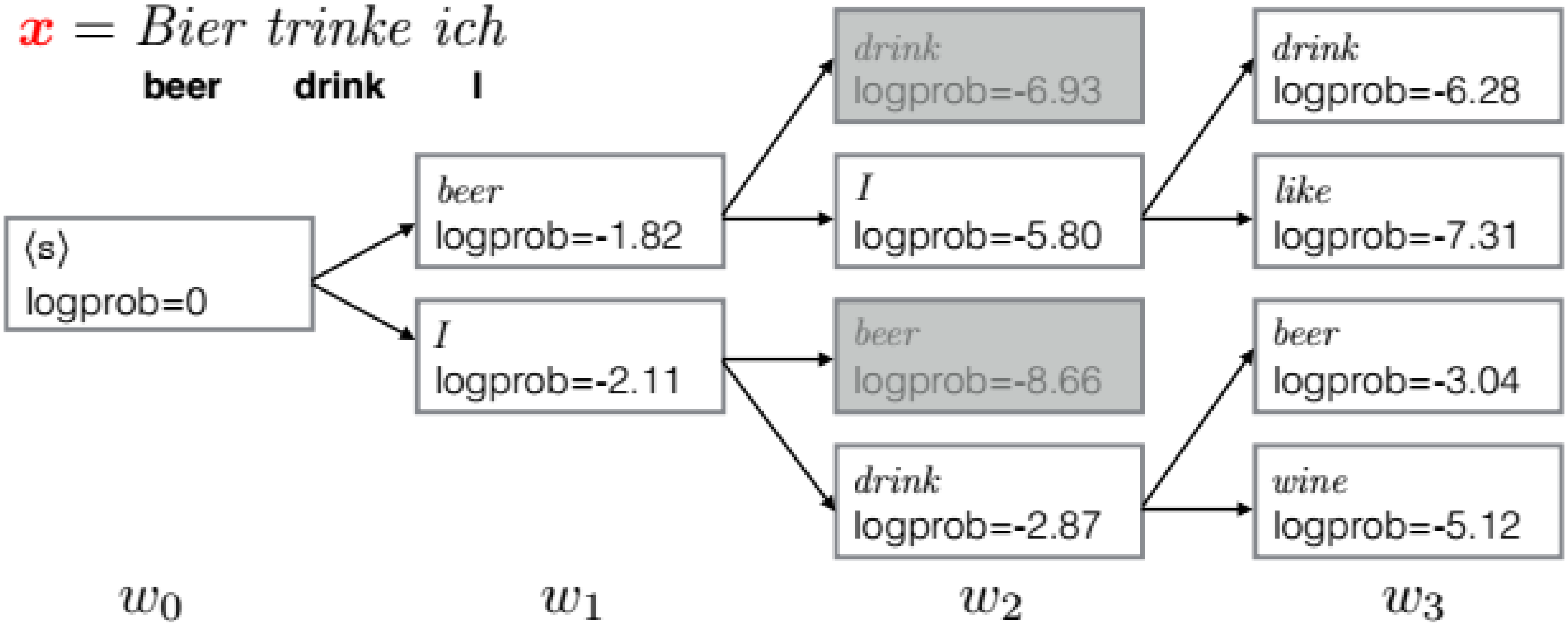
beer **drink** **I**



E.g., for b=2:

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

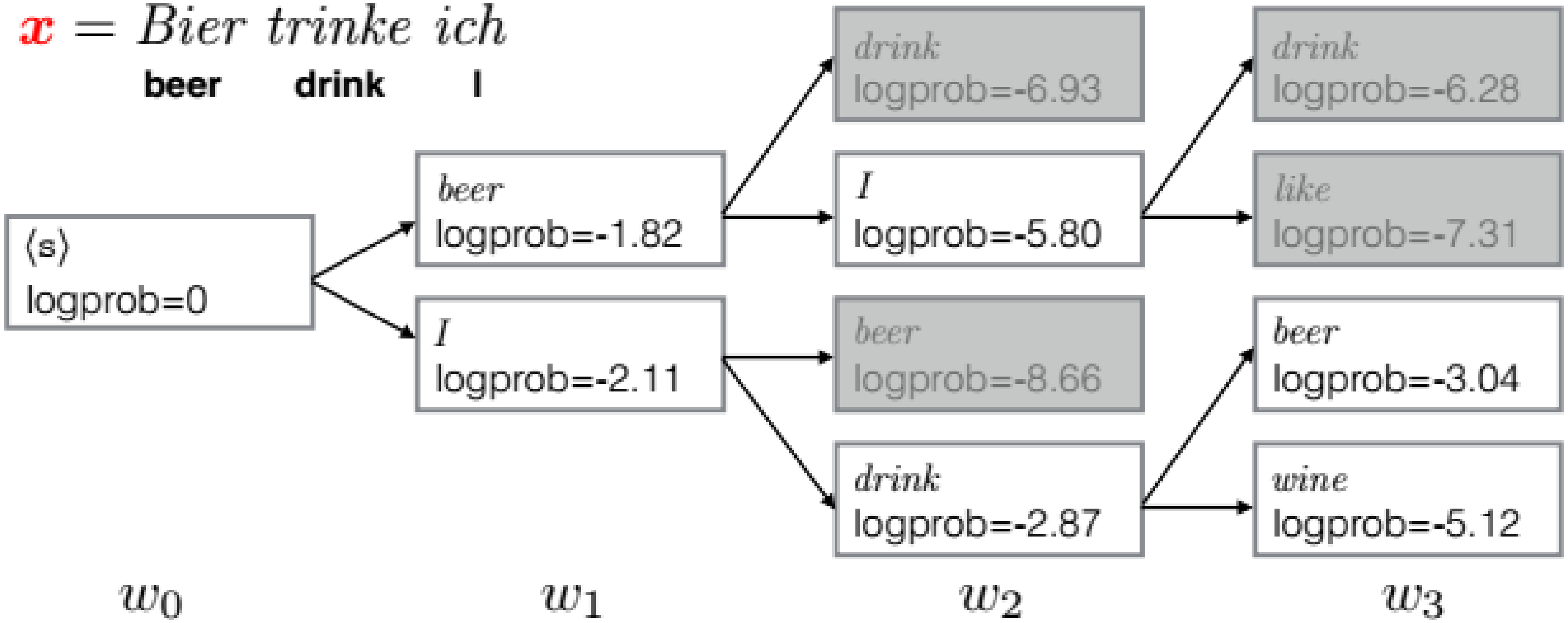
beer drink I



E.g., for b=2:

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

beer **drink** **I**



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

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