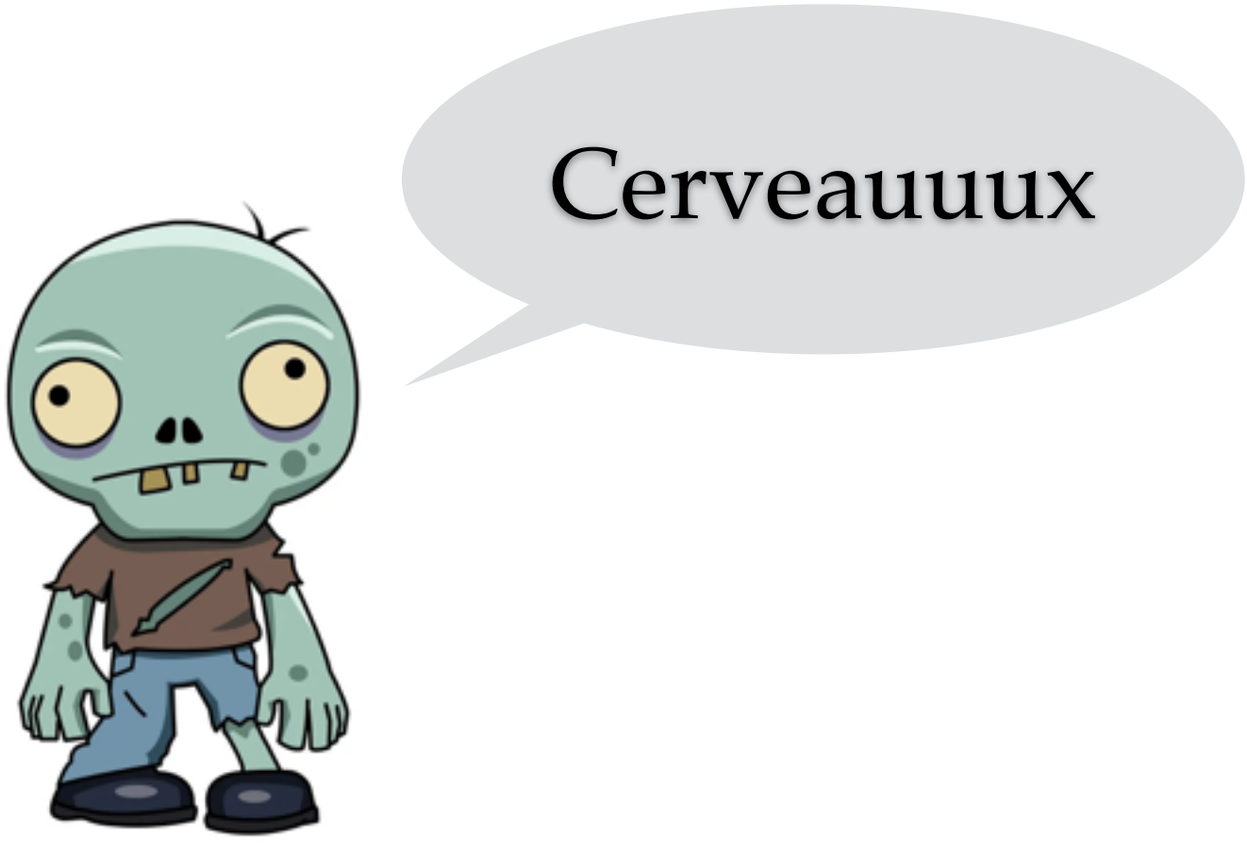
Neural machine translation

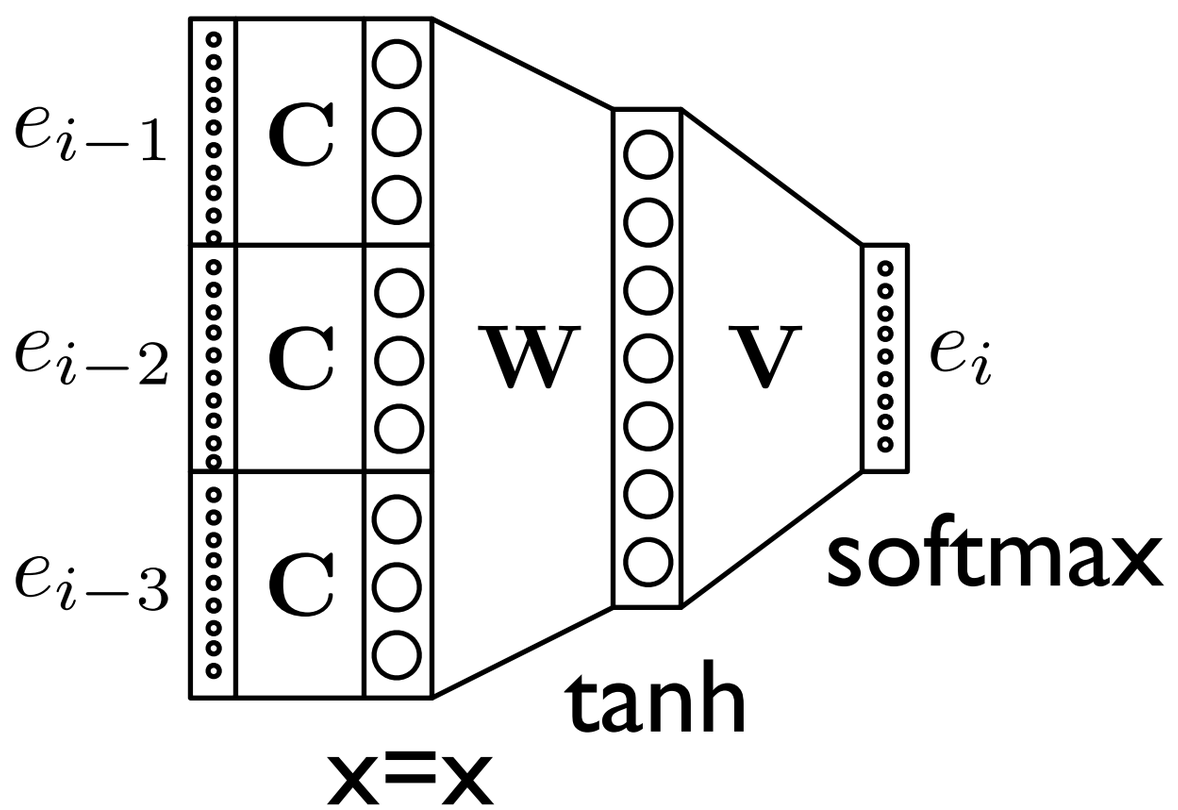


Bengio et al. (2003)

Everything is a vector

$$p(\mathbf{e}) = \prod_{i=1}^{|\mathbf{e}|} p(e_i \mid e_{i-n+1}, \ldots, e_{i-1})$$

$$p(e_i \mid e_{i-n+1}, \ldots, e_{i-1}) =$$

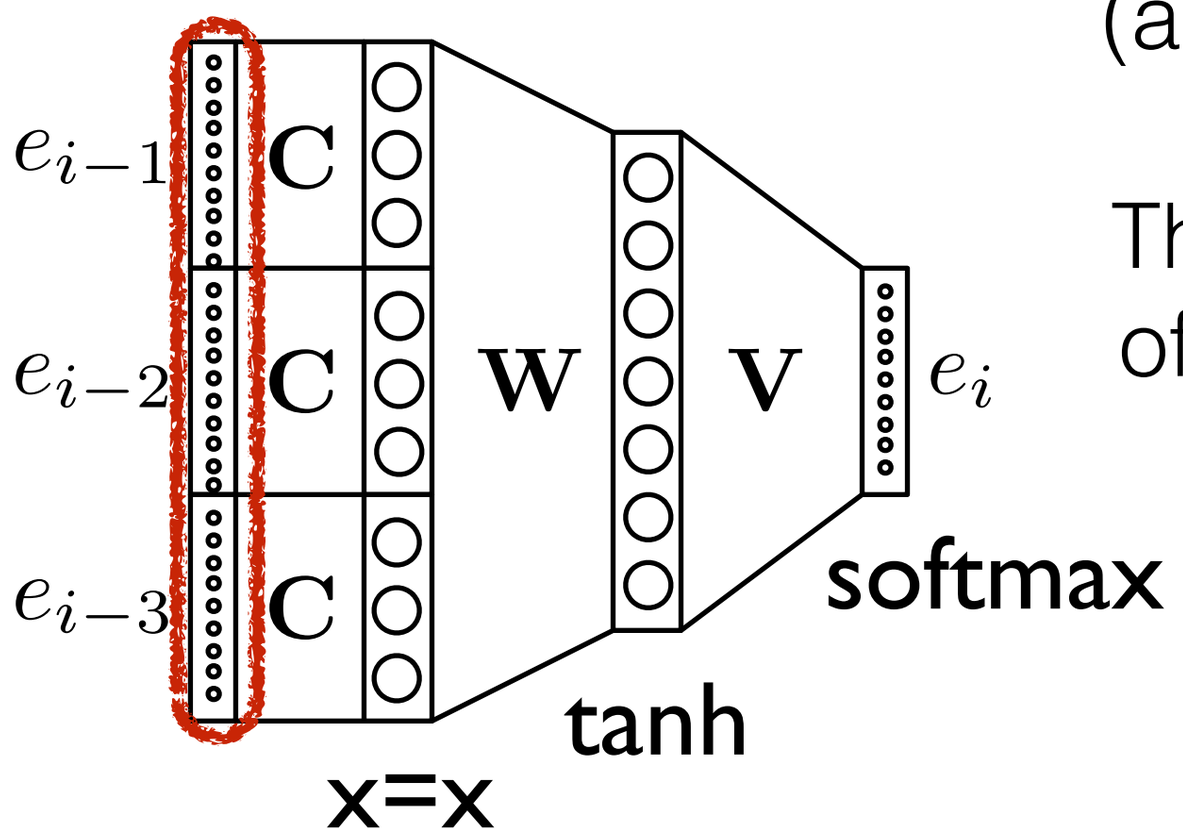


Bengio et al. (2003)

Everything is a vector

$$p(\mathbf{e}) = \prod_{i=1}^{|\mathbf{e}|} p(e_i \mid e_{i-n+1}, \ldots, e_{i-1})$$

$$p(e_i \mid e_{i-n+1}, \ldots, e_{i-1}) =$$



Every word is a vector (a one-hot vector)

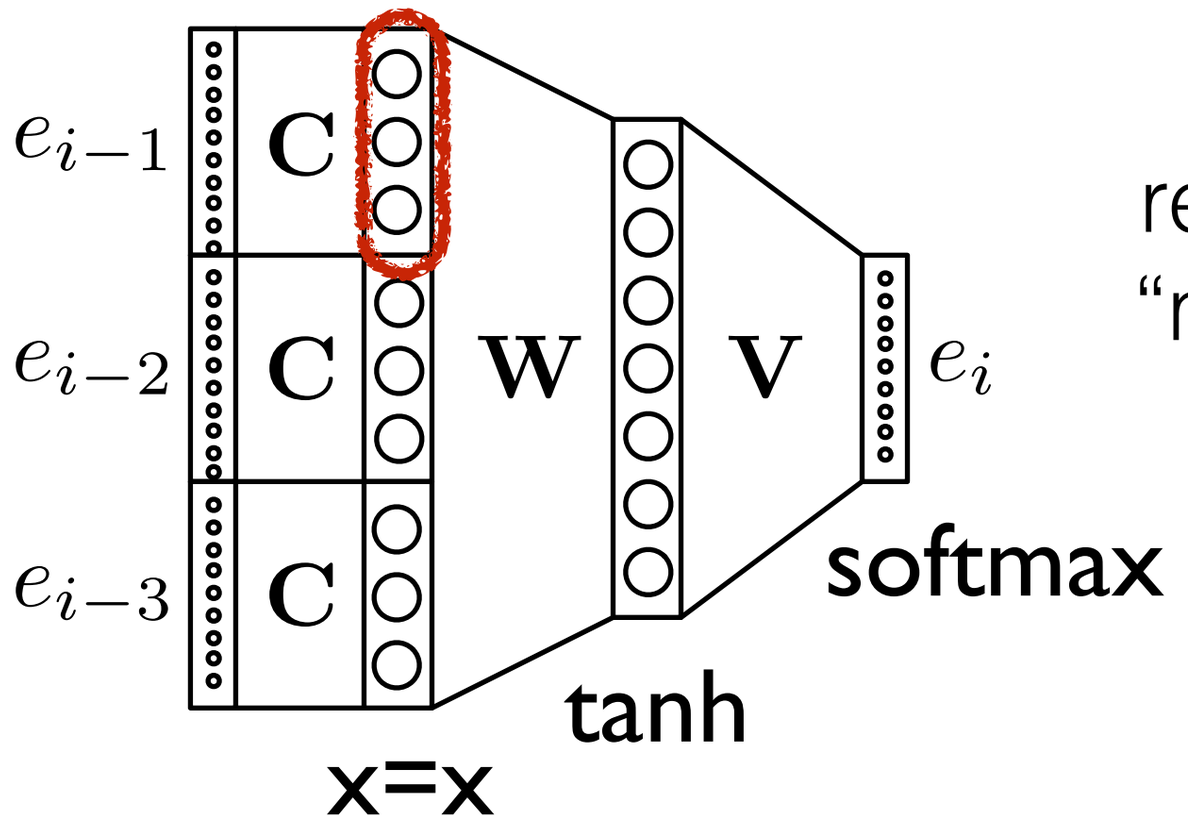
The concatenation of these vectors is an *n*-gram

Bengio et al. (2003)

Everything is a vector

$$p(\mathbf{e}) = \prod_{i=1}^{|\mathbf{e}|} p(e_i \mid e_{i-n+1}, \ldots, e_{i-1})$$

$$p(e_i \mid e_{i-n+1}, \ldots, e_{i-1}) =$$



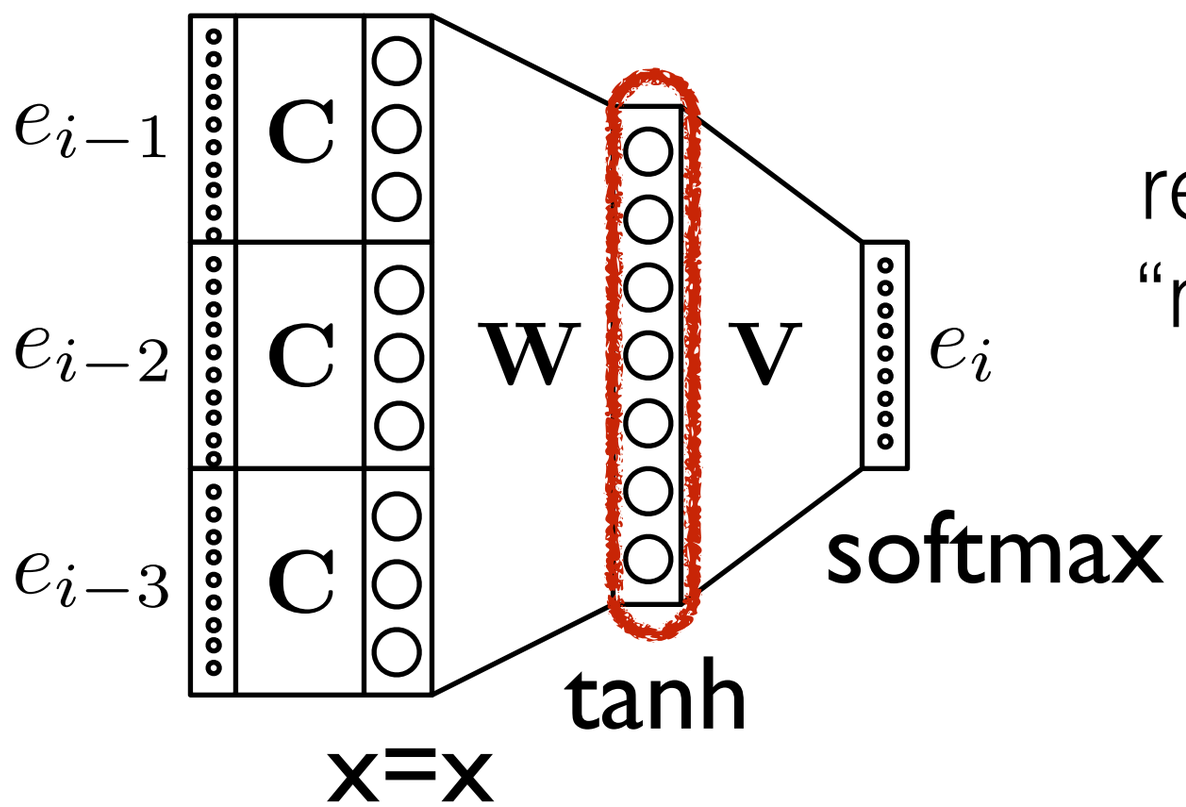
Word embeddings are vectors: *k* real values representing the "meaning" of the word.

Bengio et al. (2003)

Everything is a vector

$$p(\mathbf{e}) = \prod_{i=1}^{|\mathbf{e}|} p(e_i \mid e_{i-n+1}, \ldots, e_{i-1})$$

$$p(e_i \mid e_{i-n+1}, \ldots, e_{i-1}) =$$



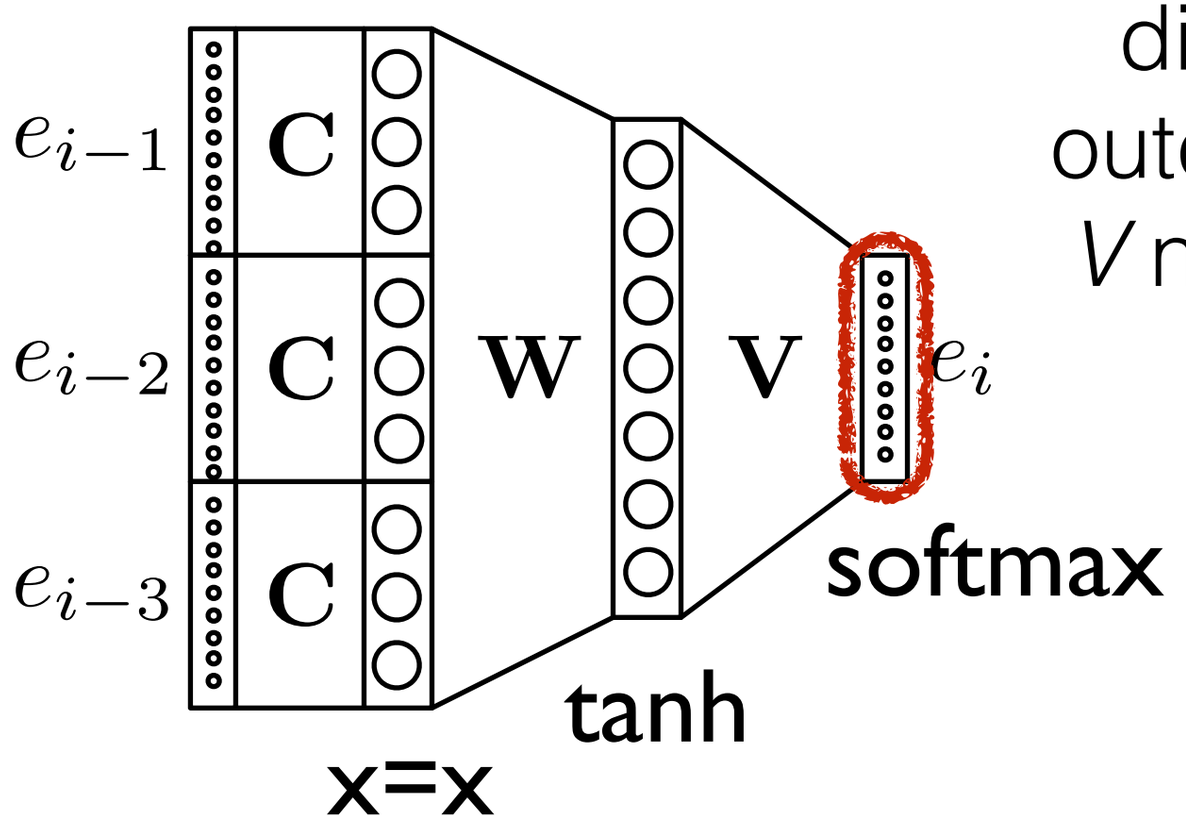
n-grams are vectors:
d real values representing the "meaning" of the *n*-gram.

Bengio et al. (2003)

Everything is a vector

$$p(\mathbf{e}) = \prod_{i=1}^{|\mathbf{e}|} p(e_i \mid e_{i-n+1}, \ldots, e_{i-1})$$

$$p(e_i \mid e_{i-n+1}, \ldots, e_{i-1}) =$$



a discrete probability distribution over *V* outcomes is a vector: *V* non-negative reals summing to 1.

LMs in MT

$$p(\mathbf{e}, \mathbf{f}) = p_{LM}(\mathbf{e}) \times p_{TM}(\mathbf{f} \mid \mathbf{e})$$

LMs in MT

$$p(\mathbf{e}, \mathbf{f}) = p_{LM}(\mathbf{e}) \times p_{TM}(\mathbf{f} \mid \mathbf{e})$$

$$= \prod_{i=1}^{|\mathbf{e}|} p(e_i \mid e_{i-1}, ..., e_1) \times \prod_{j=1}^{|\mathbf{f}|} p(f_j \mid f_{j-1}, ..., f_1, \mathbf{e})$$

LMs in MT

$$p(\mathbf{e}, \mathbf{f}) = p_{LM}(\mathbf{e}) \times p_{TM}(\mathbf{f} \mid \mathbf{e})$$

$$= \prod_{i=1}^{|\mathbf{e}|} p(e_i \mid e_{i-1}, ..., e_1) \times \prod_{j=1}^{|\mathbf{f}|} p(f_j \mid f_{j-1}, ..., f_1, \mathbf{e})$$

Language model

LMs in MT

$$p(\mathbf{e}, \mathbf{f}) = p_{LM}(\mathbf{e}) \times p_{TM}(\mathbf{f} \mid \mathbf{e})$$

$$= \prod_{i=1}^{|\mathbf{e}|} p(e_i \mid e_{i-1}, ..., e_1) \times \prod_{j=1}^{|\mathbf{f}|} p(f_j \mid f_{j-1}, ..., f_1, \mathbf{e})$$

Conditional language model

LMs in MT

$$p(\mathbf{e}, \mathbf{f}) = p_{LM}(\mathbf{e}) \times p_{TM}(\mathbf{f} \mid \mathbf{e})$$

$$= \underbrace{\prod_{i=1}^{|\mathbf{e}|} p(e_i \mid e_{i-1}, ..., e_1)}_{} \times \underbrace{\prod_{j=1}^{|\mathbf{f}|} p(f_j \mid f_{j-1}, ..., f_1, \mathbf{e})}_{}$$

Conditioned on entire sentence!

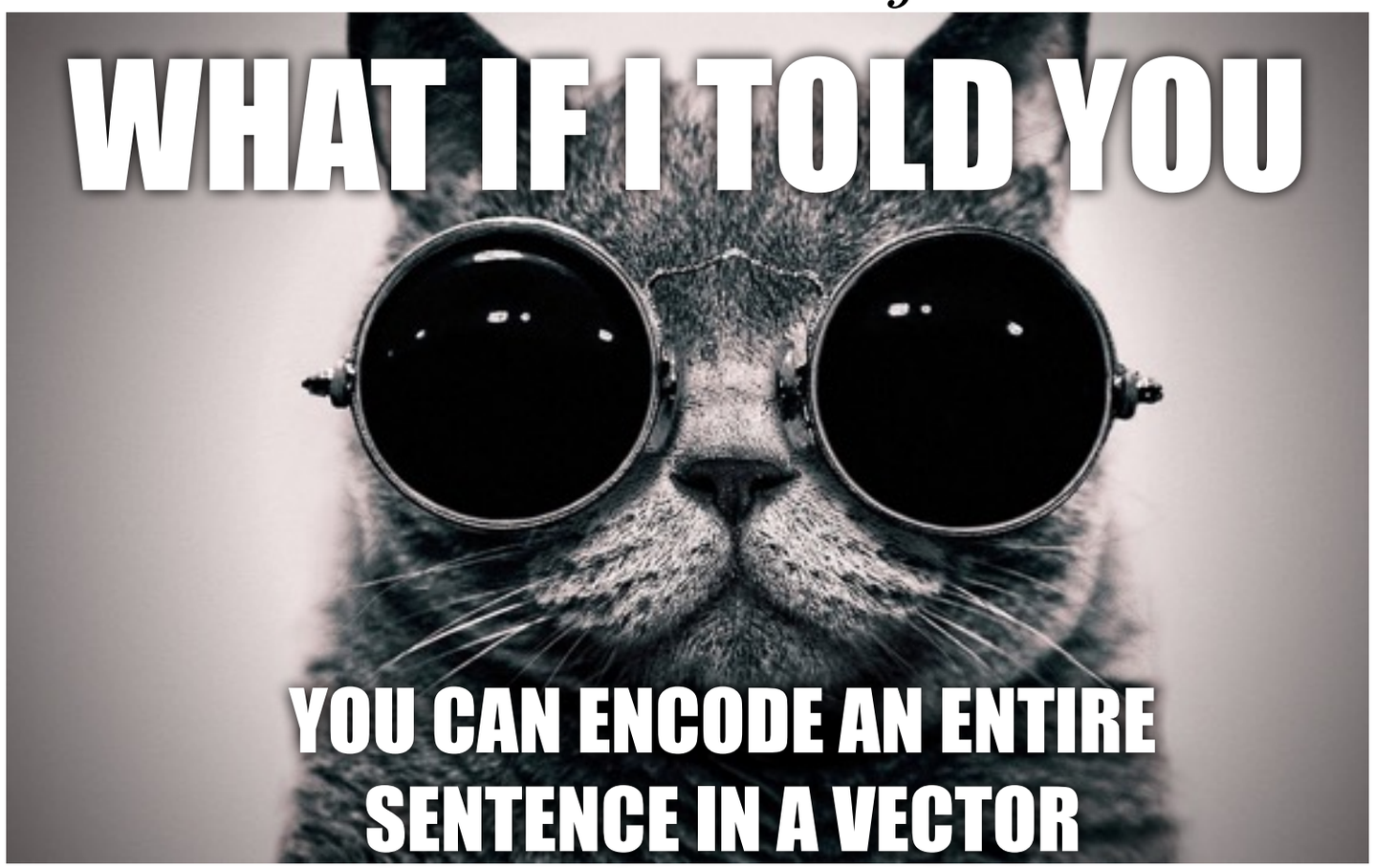
With direct multinomial parameterizations (n-gram models, IMB Model 1), we had to make independence assumptions for this to work.

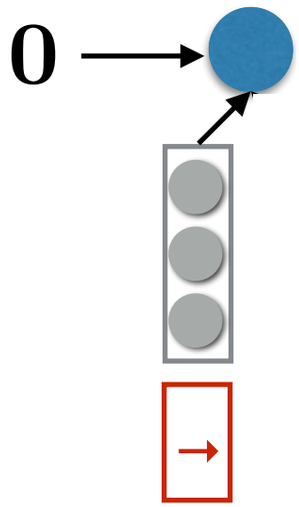
Bengio's NLM inherits these assumptions.

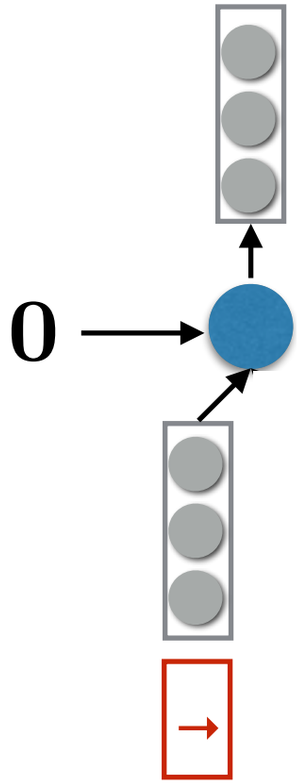
LMs in MT

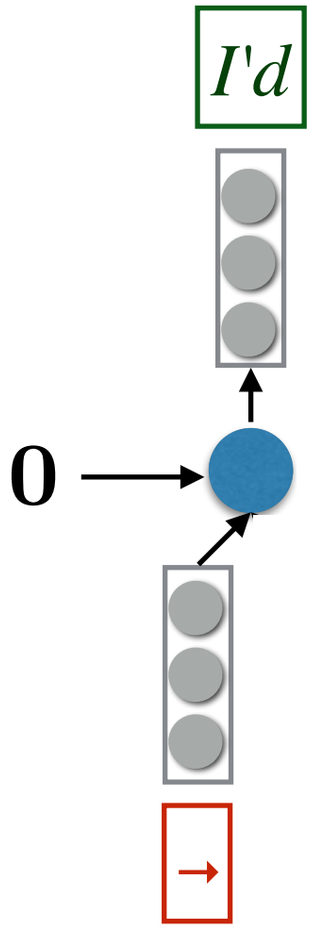
$$p(\mathbf{e}, \mathbf{f}) = p_{LM}(\mathbf{e}) \times p_{TM}(\mathbf{f} \mid \mathbf{e})$$

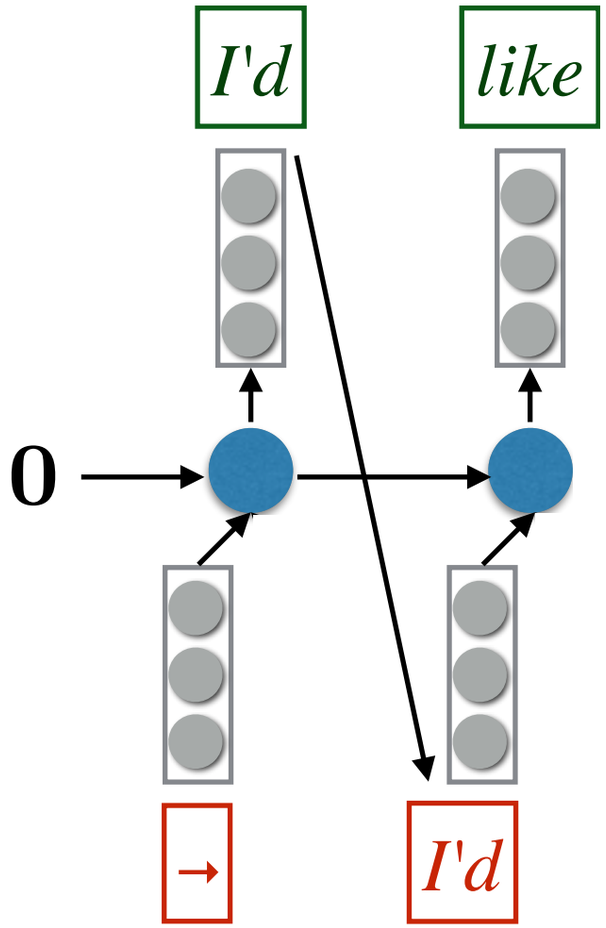
$$= \prod_{i=1}^{|\mathbf{e}|} p(e_i \mid e_{i-1}, ..., e_1) \times \prod_{j=1}^{|\mathbf{f}|} p(f_j \mid f_{j-1}, ..., f_1, \mathbf{e})$$

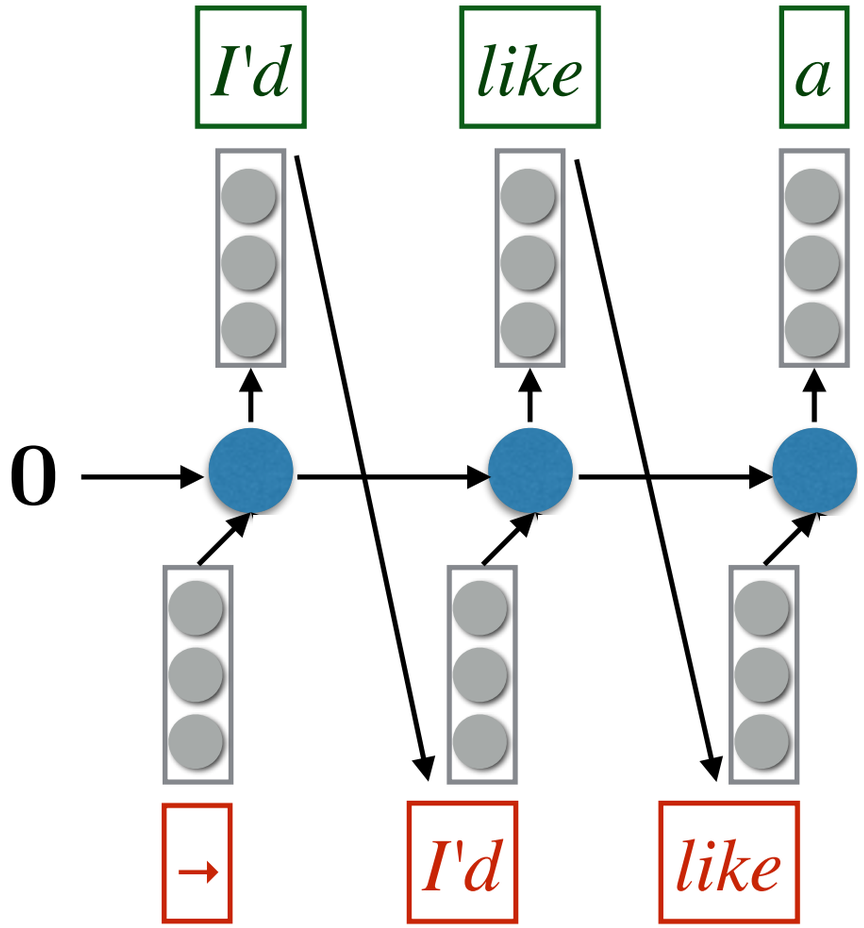


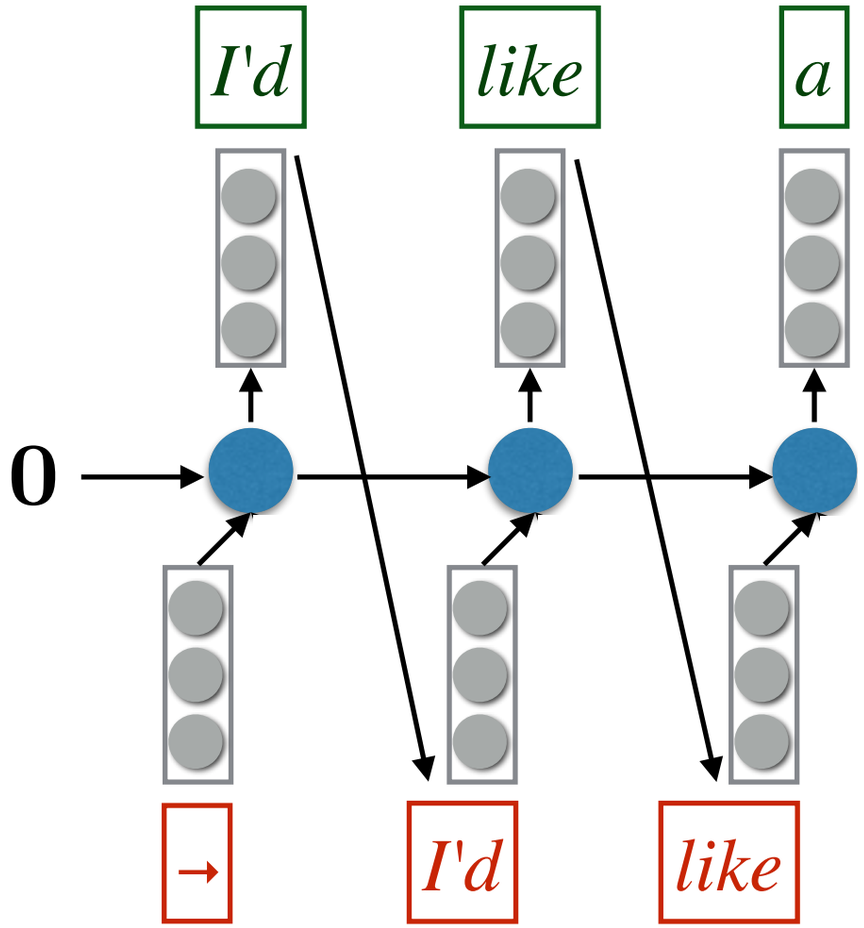

0

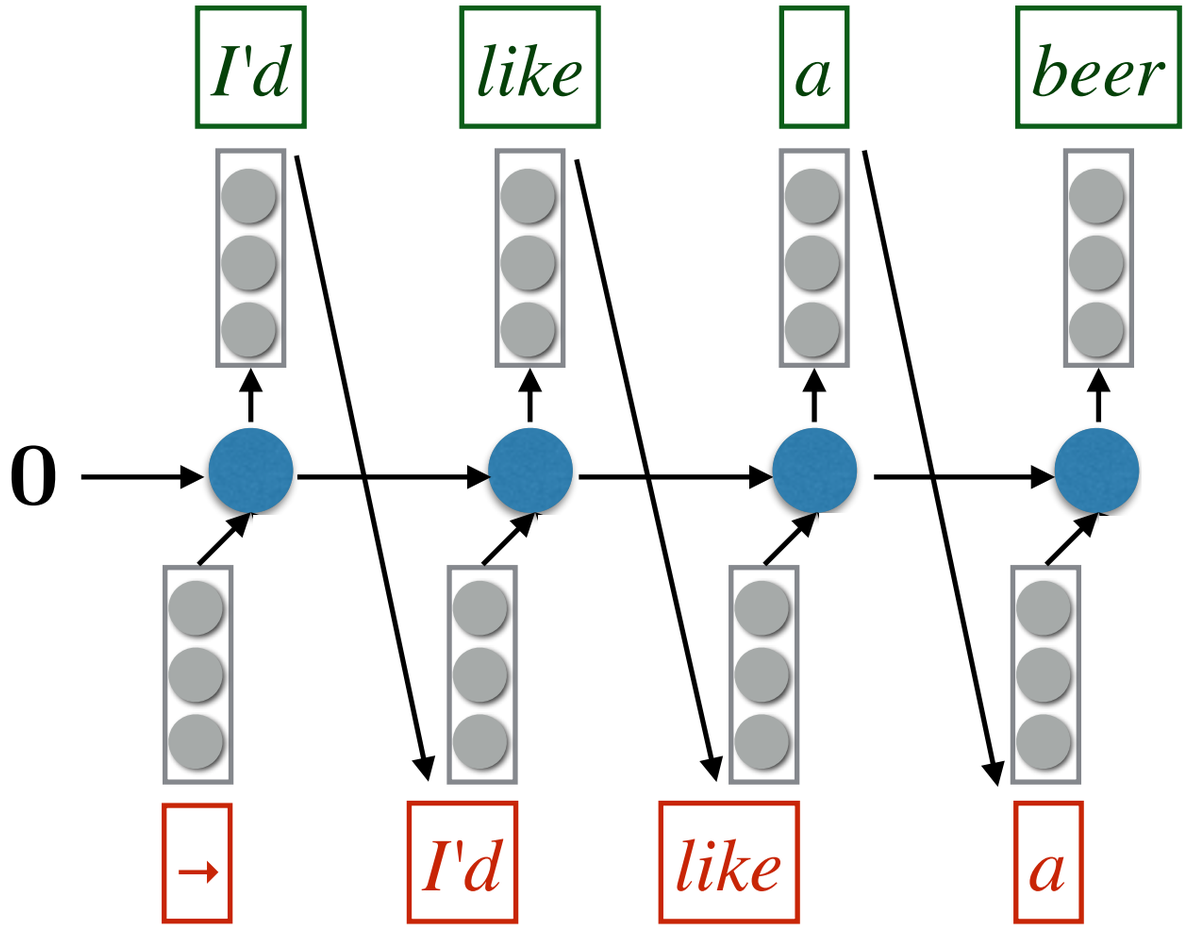

0

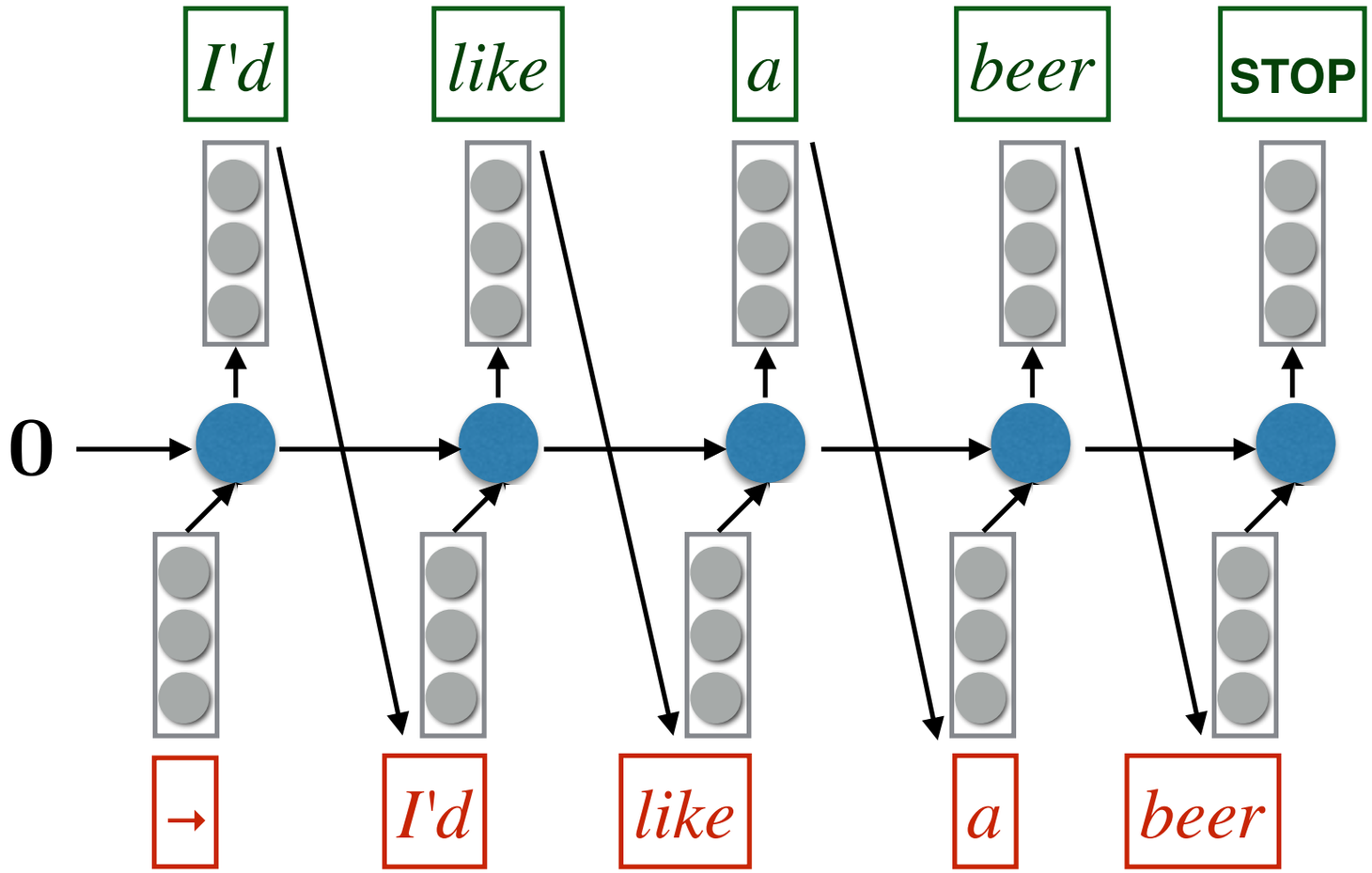

I'd
0
→

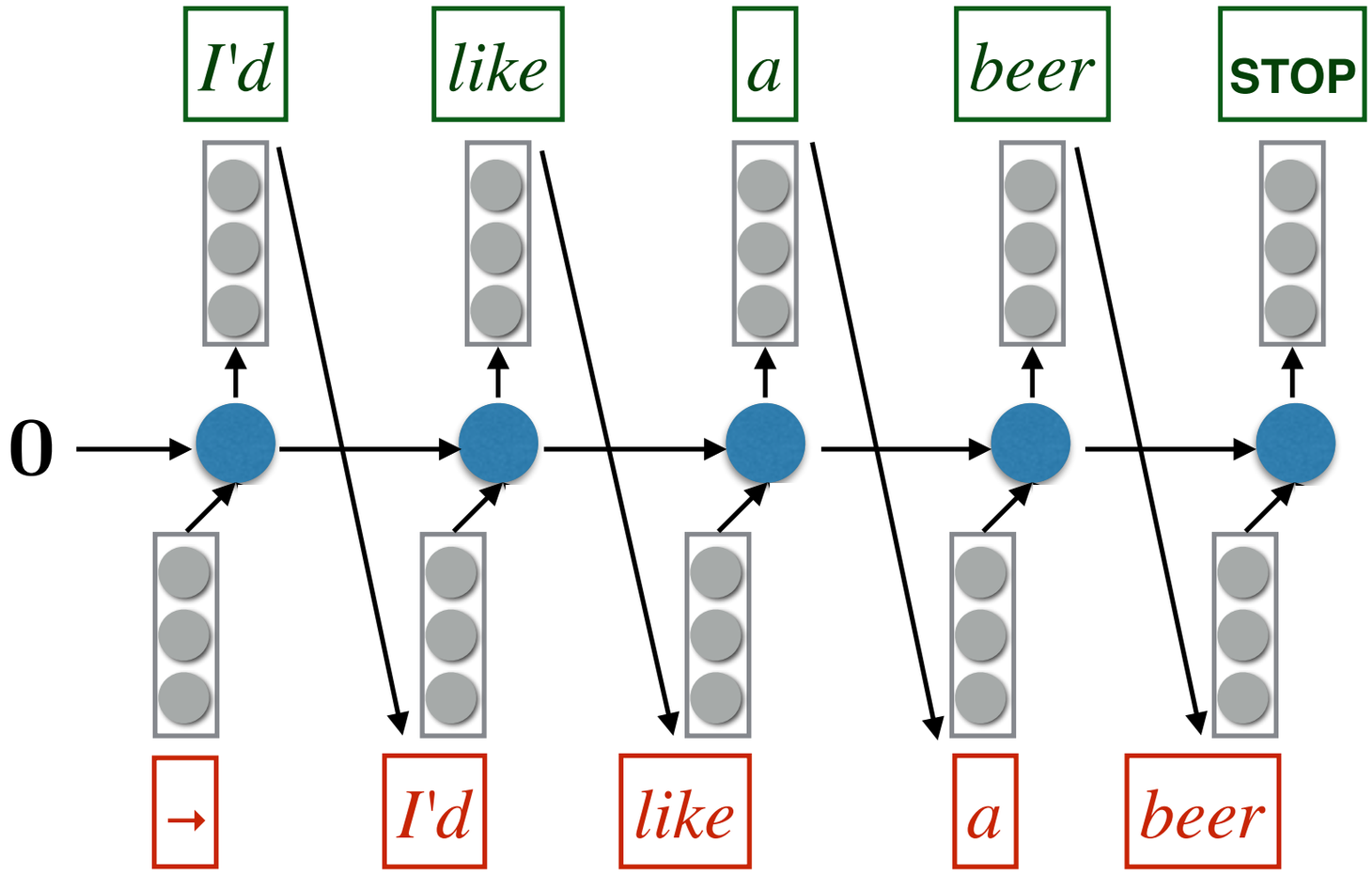

I'd
like
0
→
I'd


I'd
like
a
0
→
I'd
like

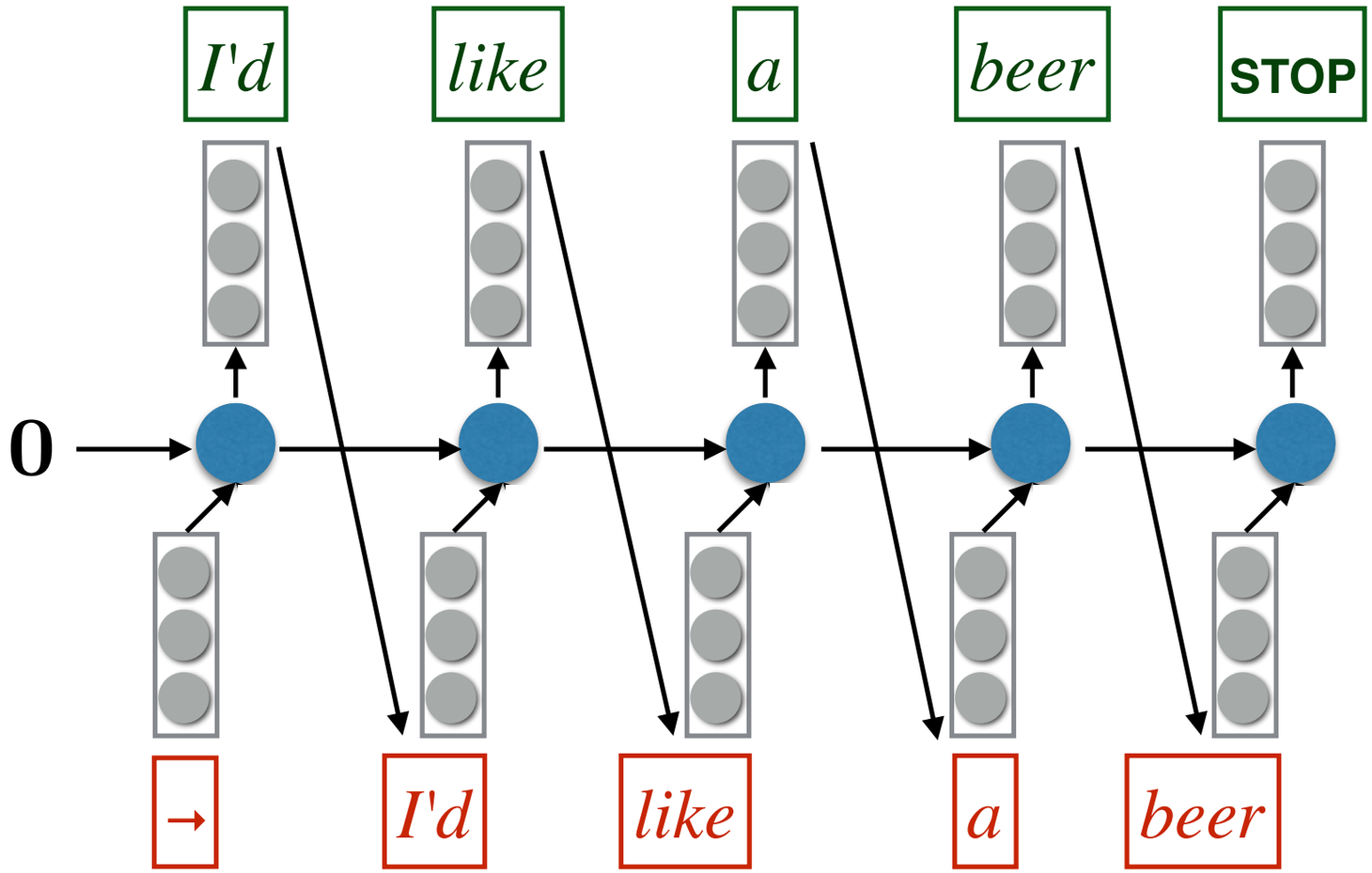

I'd
like
a
0
→
I'd
like


I'd
like
a
beer
0
→
I'd
like
a


I'd
like
a
beer
STOP
0
→
I'd
like
a
beer

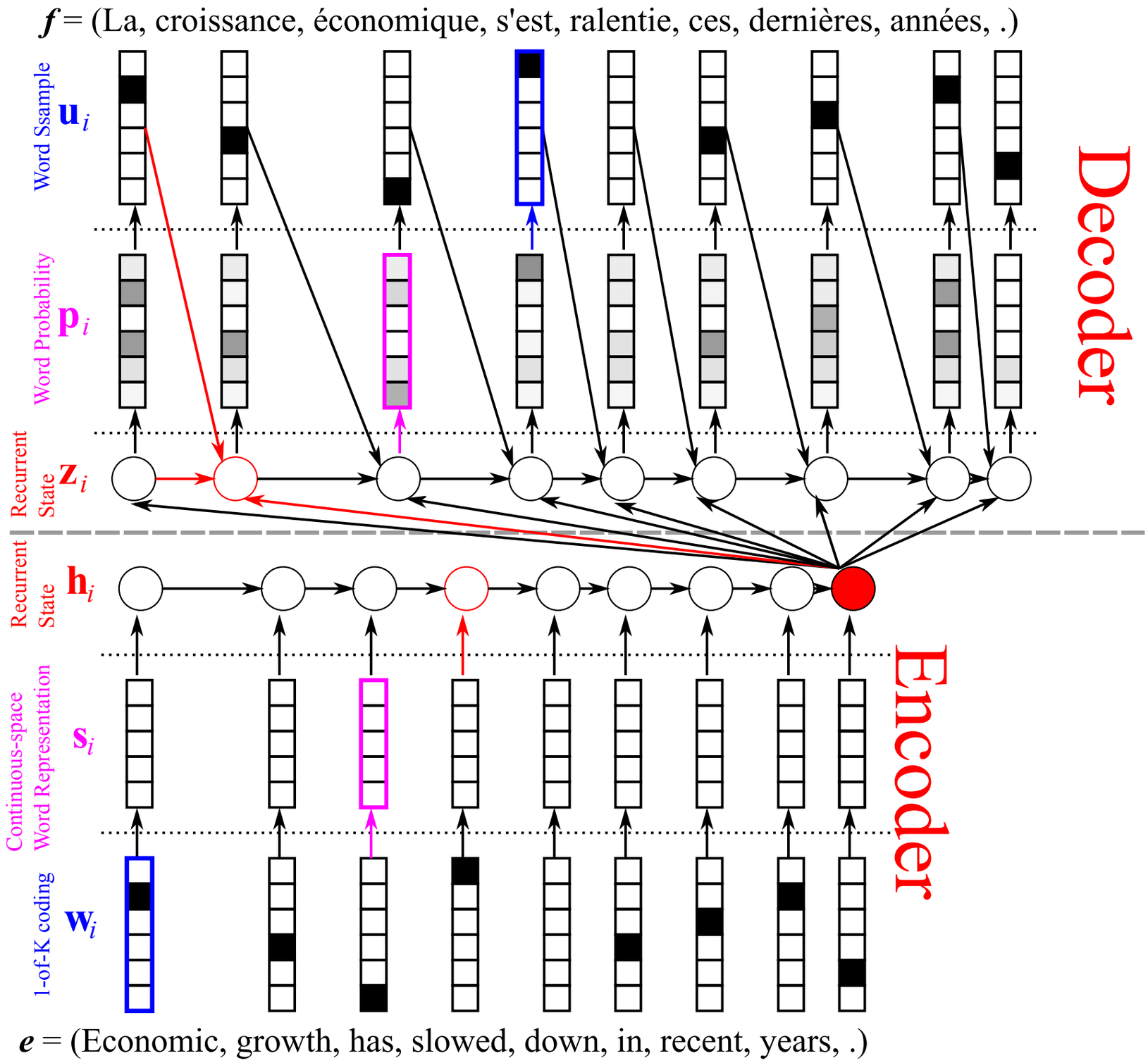


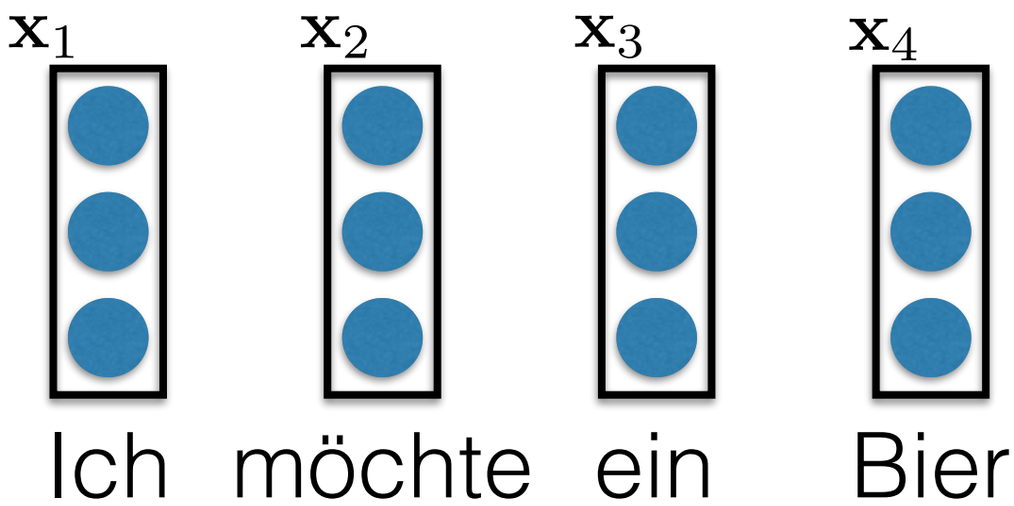
$$
\begin{aligned}
\mathbf{s_4} &= R(\mathbf{s_3}, \mathbf{x_4}) \\
&= R(\overbrace{R(\mathbf{s_2}, \mathbf{x_3})}^{\mathbf{s_3}}, \mathbf{x_4}) \\
&= R(R(\overbrace{R(\mathbf{s_1}, \mathbf{x_2})}^{\mathbf{s_2}}, \mathbf{x_3}), \mathbf{x_4}) \\
&= R(R(R(\overbrace{R(\mathbf{s_0}, \mathbf{x_1})}^{\mathbf{s_1}}, \mathbf{x_2}), \mathbf{x_3}), \mathbf{x_4})
\end{aligned}
$$

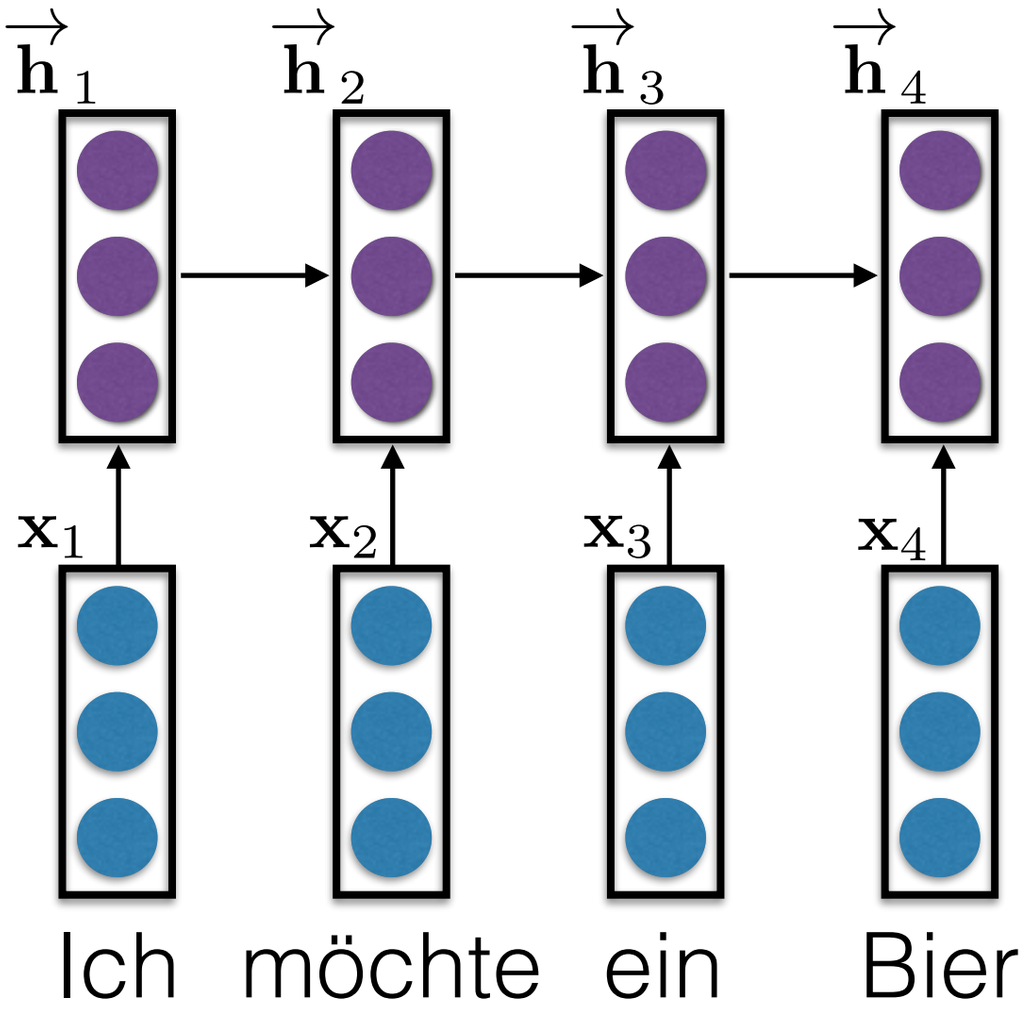


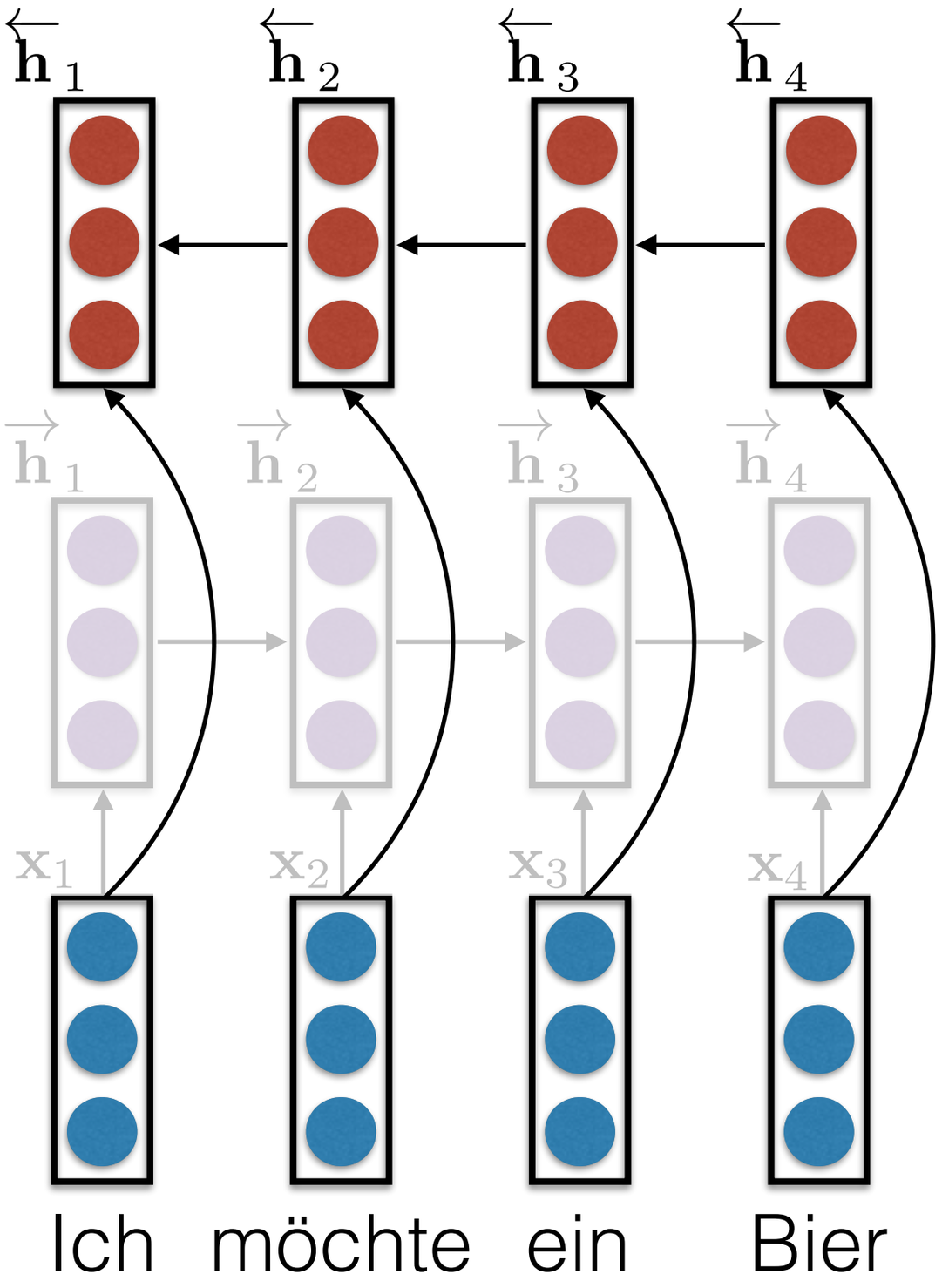
$$p(\mathbf{e}) = \prod_{i=1}^{|\mathbf{e}|} p(e_i|e_{i-1}, ..., e_1)$$

$$p(\mathbf{f}|\mathbf{e}) = \prod_{i=1}^{|\mathbf{f}|} p(f_i|f_{i-1}, ..., f_1, \mathbf{e})$$

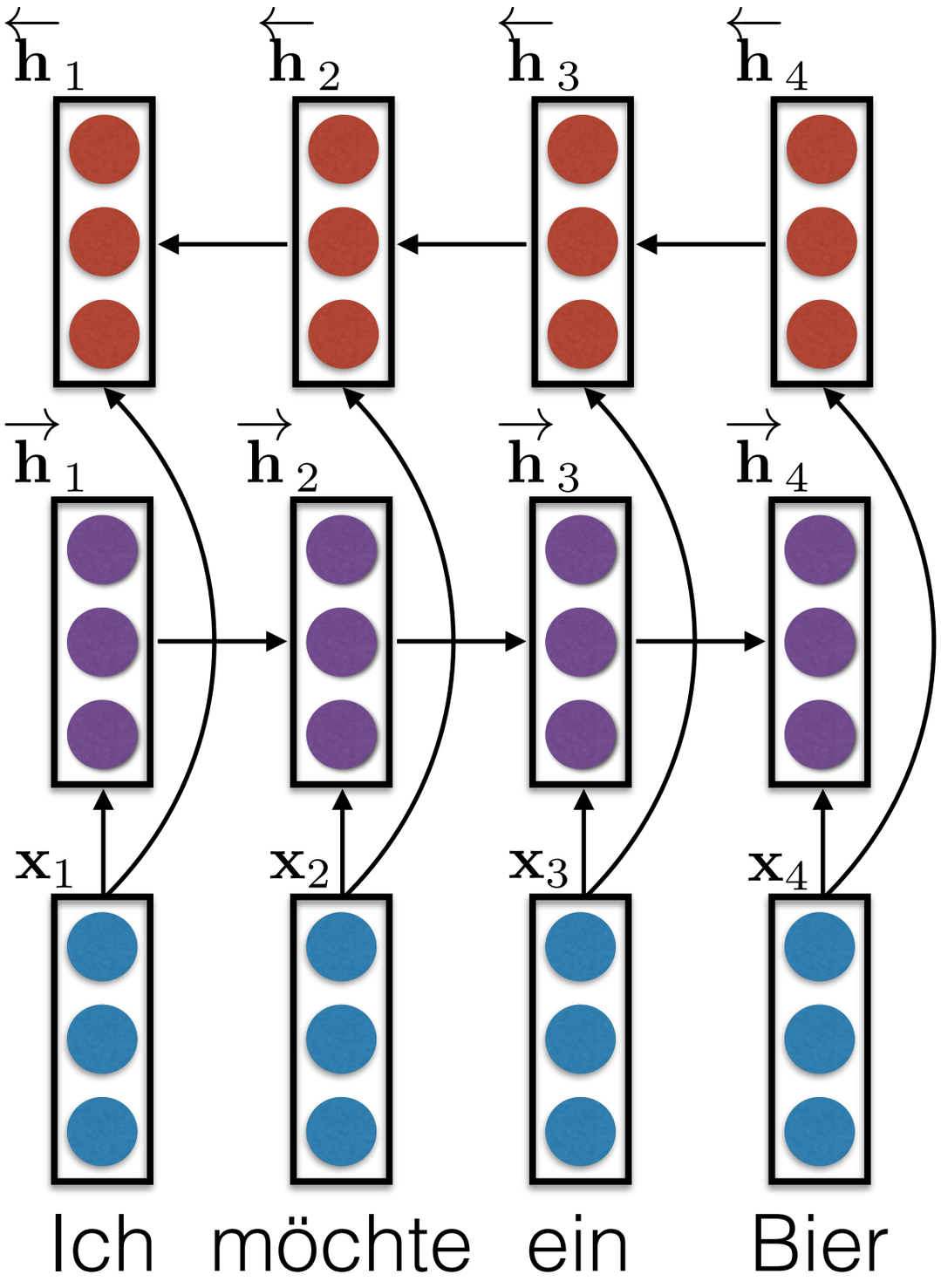


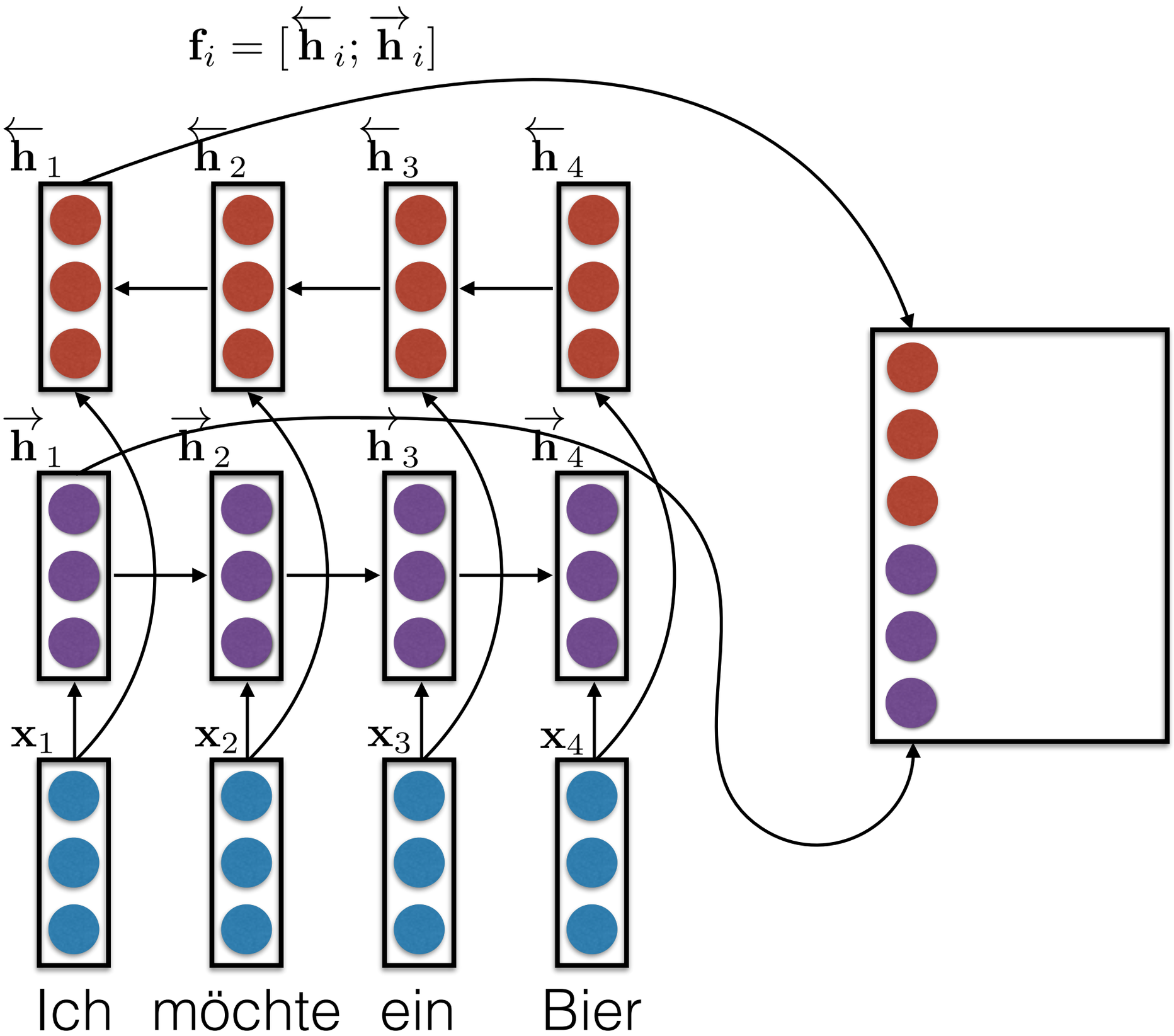

$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
Ich
möchte
ein
Bier


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

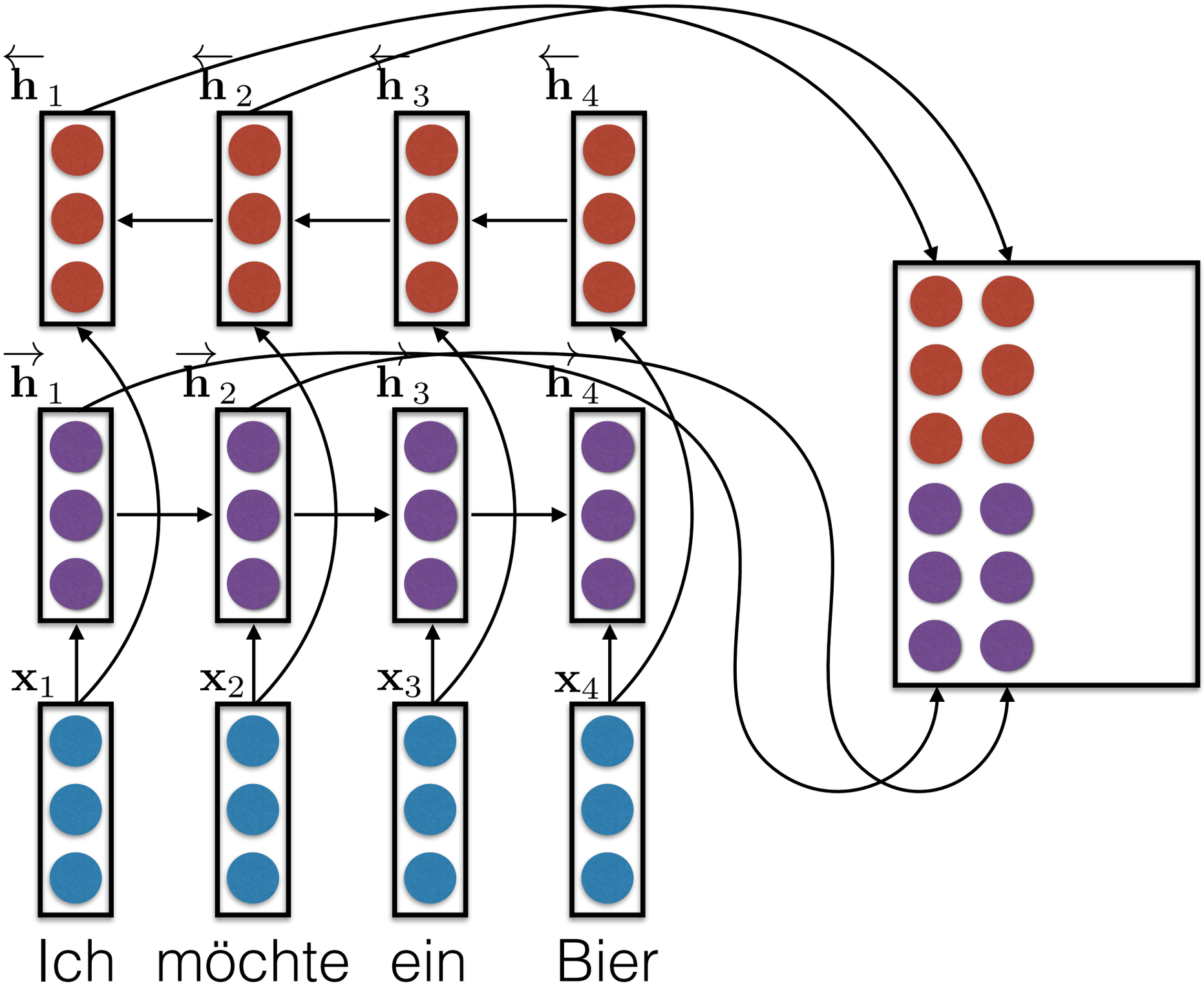

$\overleftarrow{\mathbf{h}}_1$
$\overleftarrow{\mathbf{h}}_2$
$\overleftarrow{\mathbf{h}}_3$
$\overleftarrow{\mathbf{h}}_4$
$\overrightarrow{\mathbf{h}}_1$
$\overrightarrow{\mathbf{h}}_2$
$\overrightarrow{\mathbf{h}}_3$
$\overrightarrow{\mathbf{h}}_4$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$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

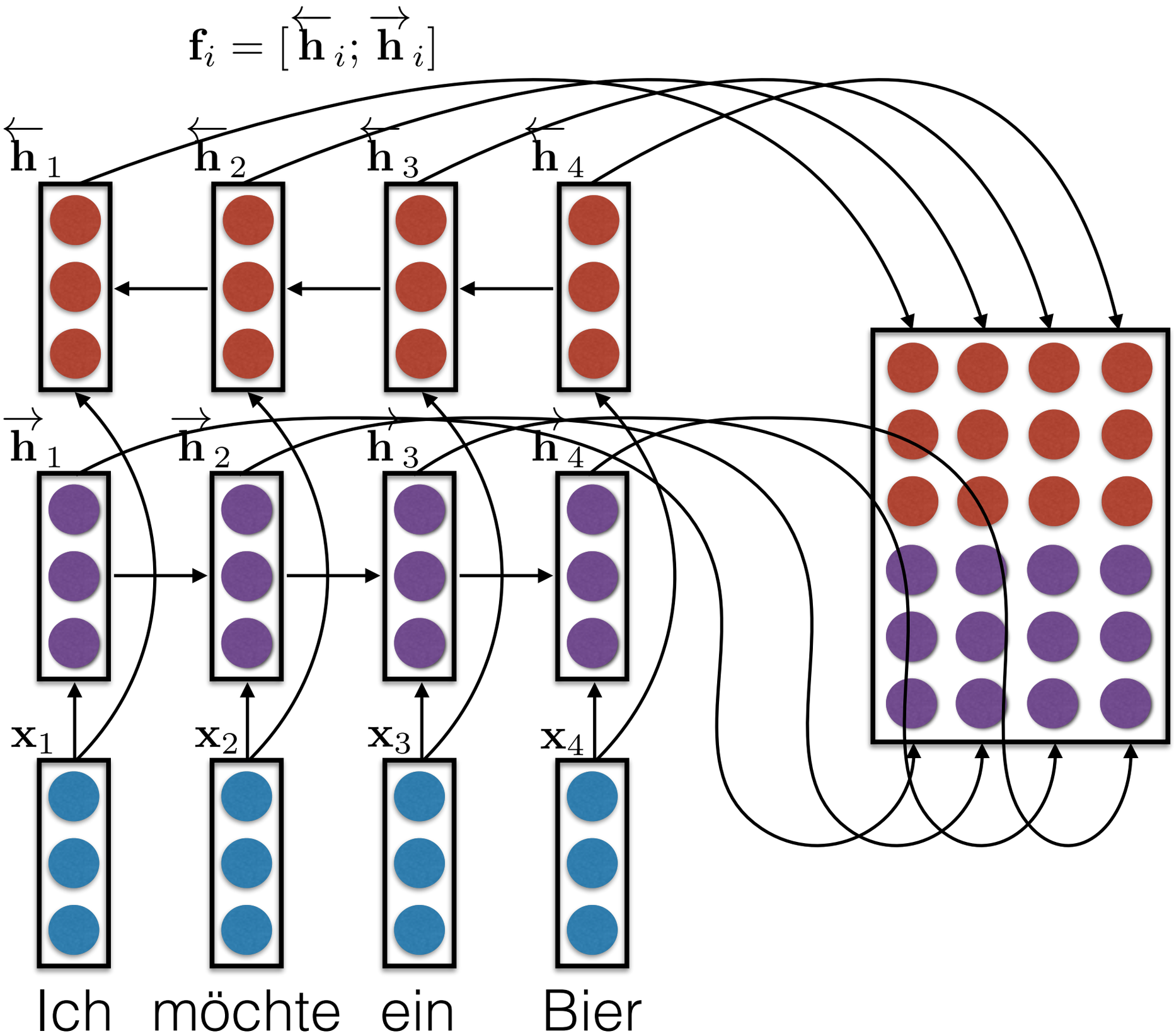
$$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$$



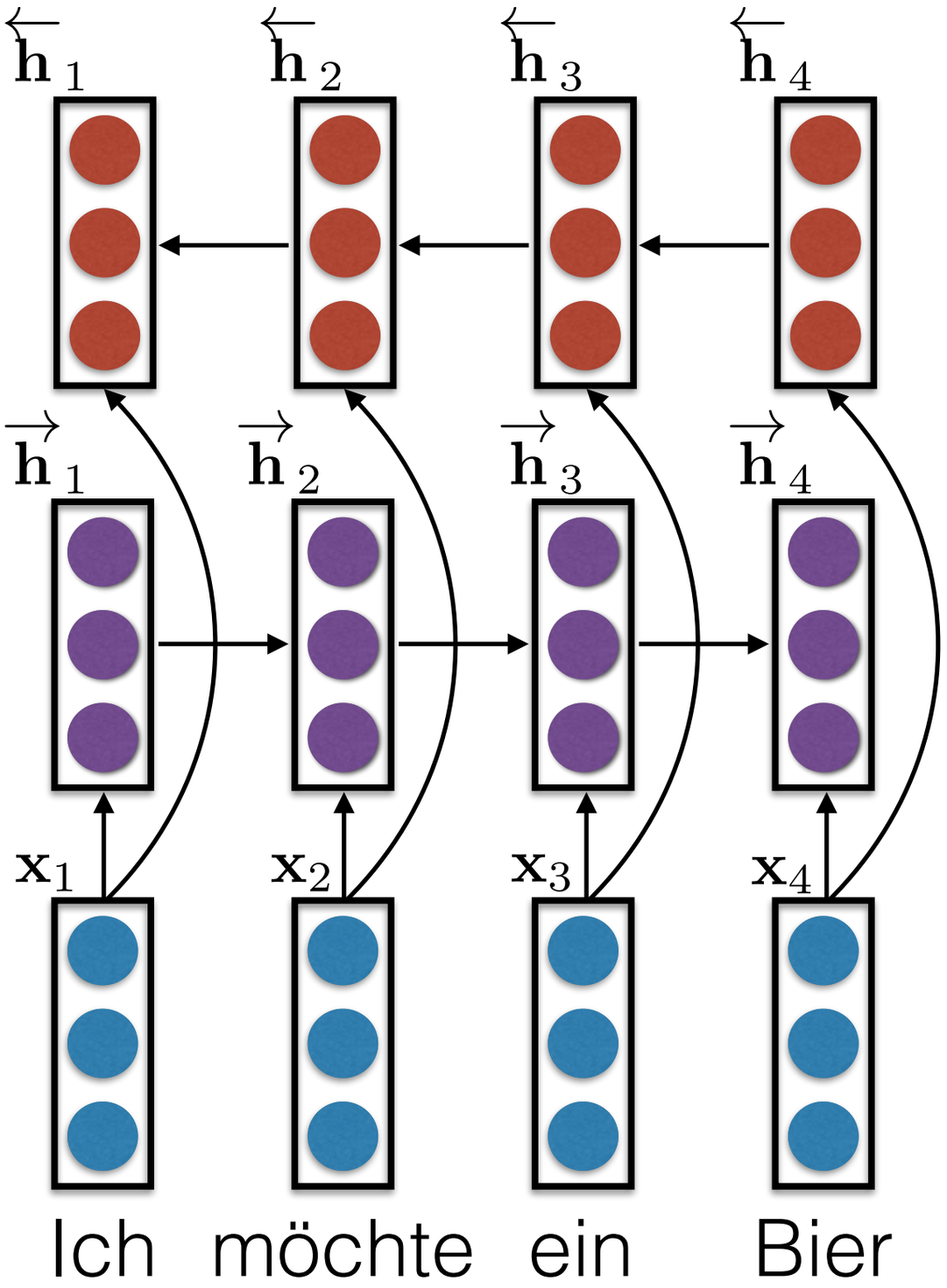

$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$
$\overleftarrow{\mathbf{h}}_1$
$\overleftarrow{\mathbf{h}}_2$
$\overleftarrow{\mathbf{h}}_3$
$\overleftarrow{\mathbf{h}}_4$
$\overrightarrow{\mathbf{h}}_1$
$\overrightarrow{\mathbf{h}}_2$
$\overrightarrow{\mathbf{h}}_3$
$\overrightarrow{\mathbf{h}}_4$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

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

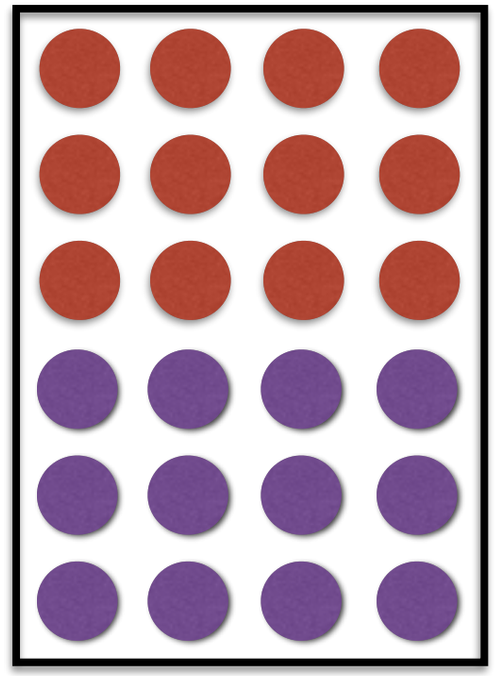



$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$
$\overleftarrow{\mathbf{h}}_1$
$\overleftarrow{\mathbf{h}}_2$
$\overleftarrow{\mathbf{h}}_3$
$\overleftarrow{\mathbf{h}}_4$
$\overrightarrow{\mathbf{h}}_1$
$\overrightarrow{\mathbf{h}}_2$
$\overrightarrow{\mathbf{h}}_3$
$\overrightarrow{\mathbf{h}}_4$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

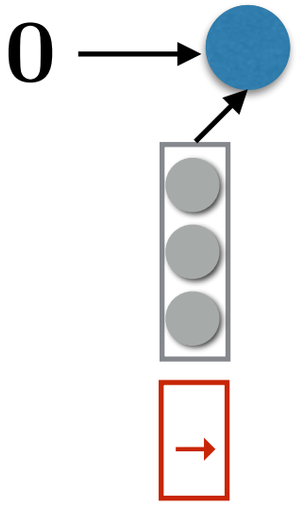
$$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$$

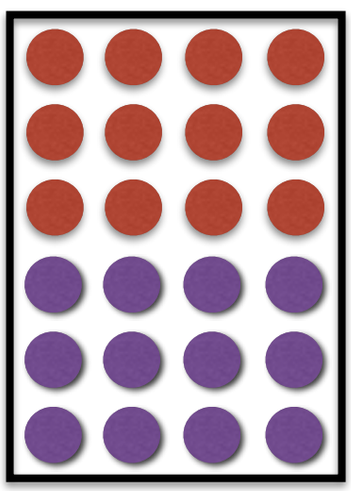


$$\mathbf{F} \in \mathbb{R}^{2n \times |\boldsymbol{f}|}$$

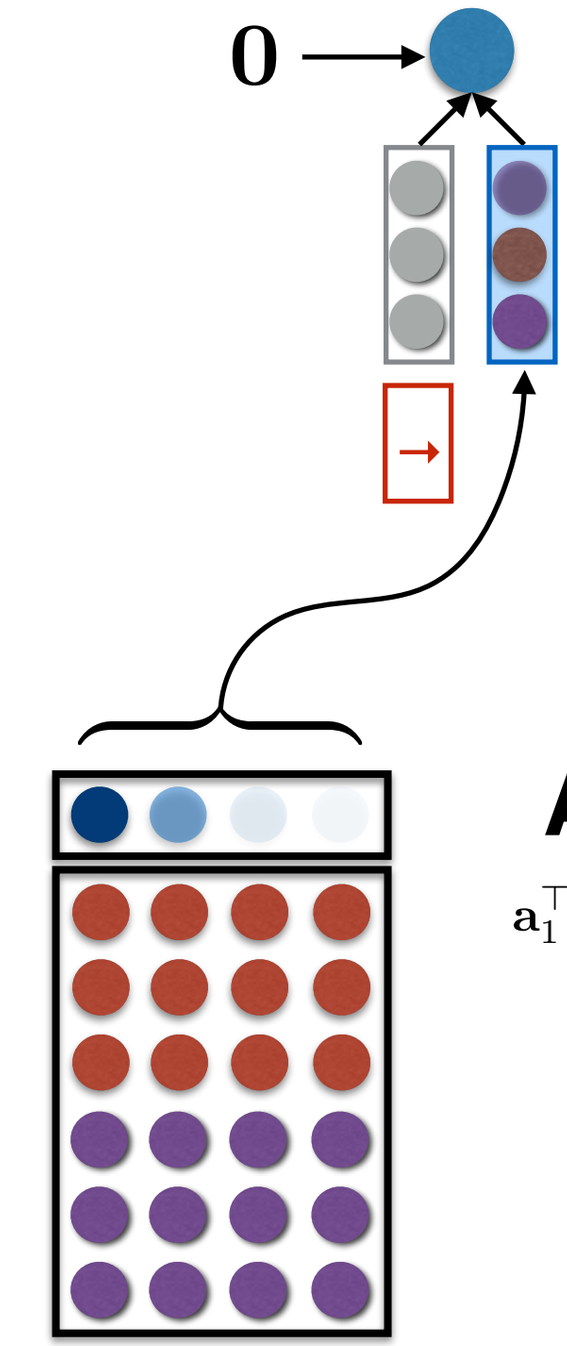


Ich möchte ein Bier

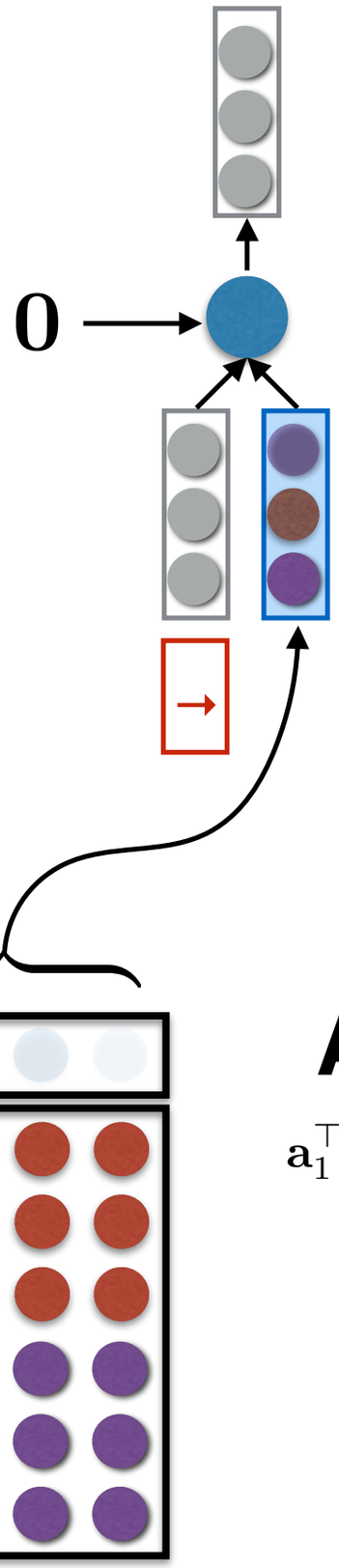




Ich möchte ein Bier

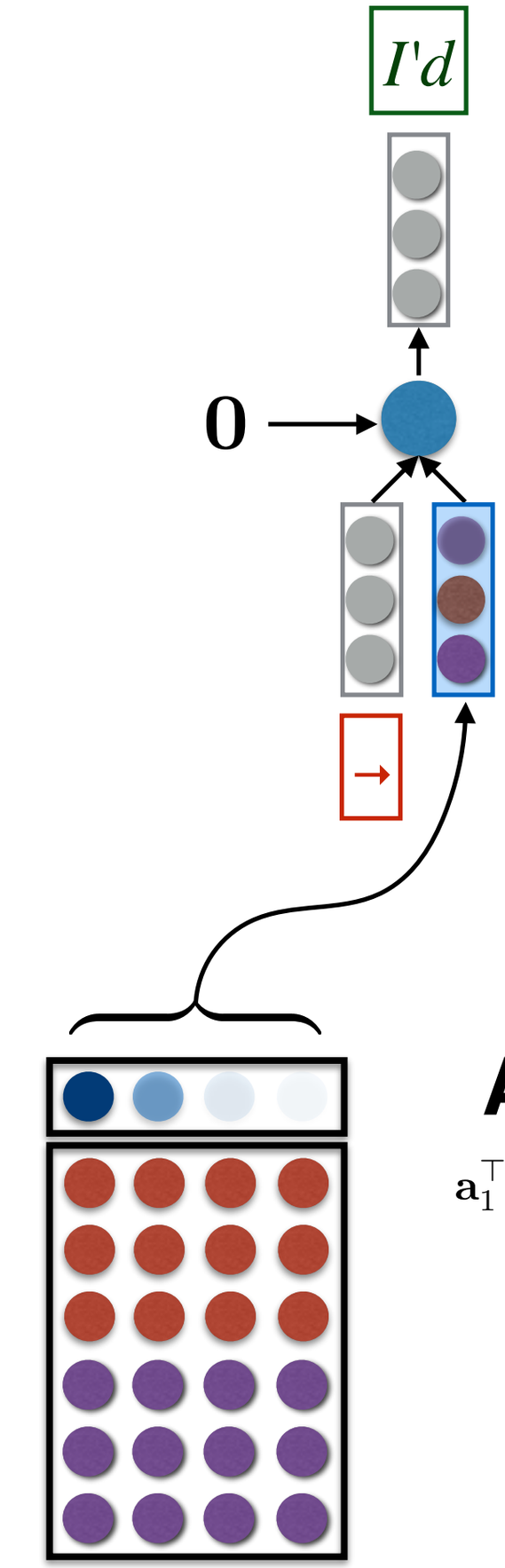


Attention history:



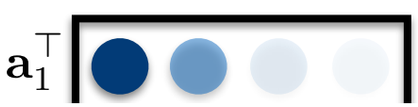
Attention history:

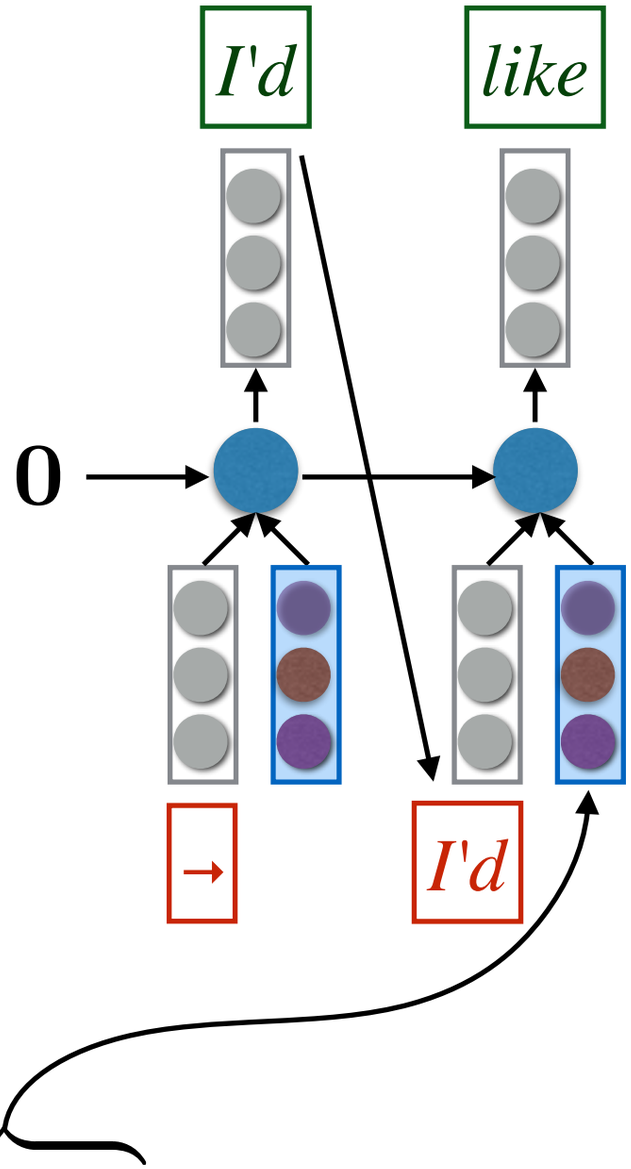
$\mathbf{a}_1^\top$

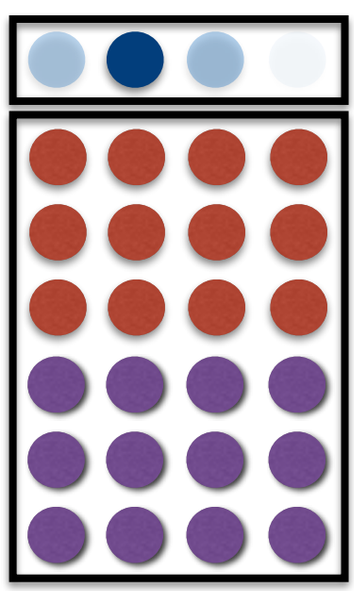


Attention history:

$\mathbf{a}_1^\top$

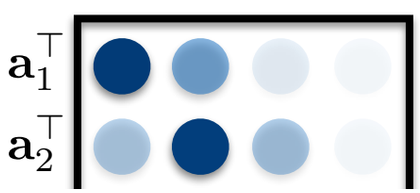


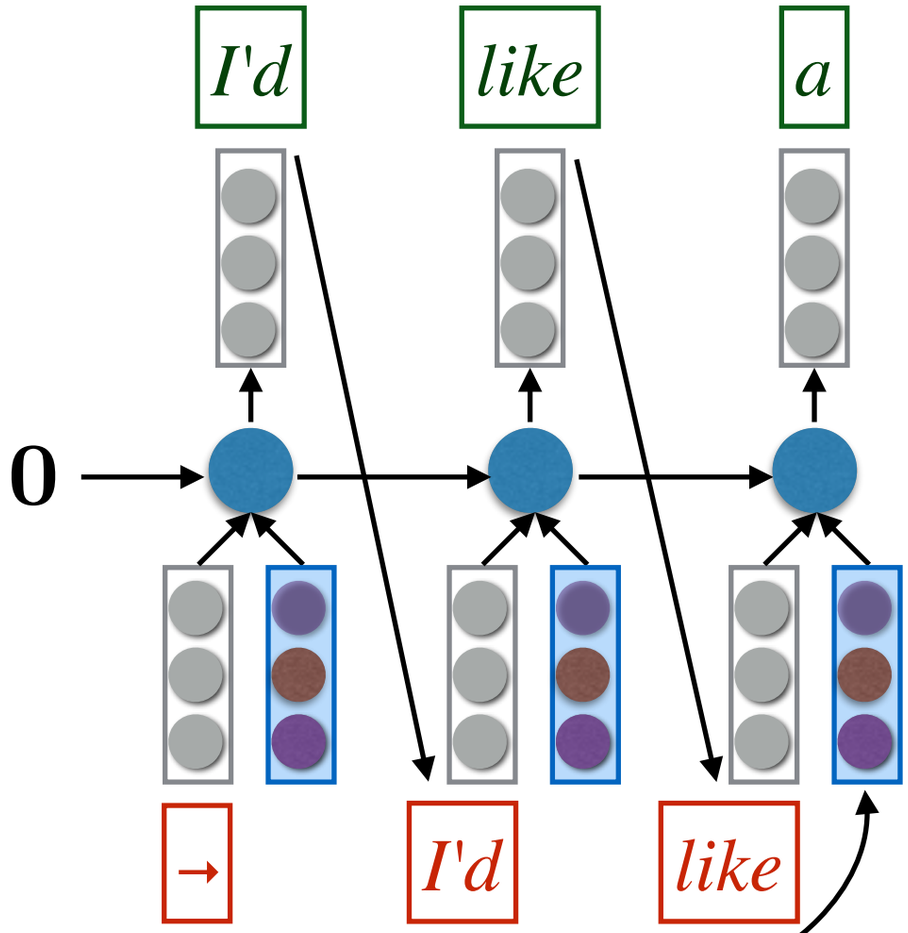


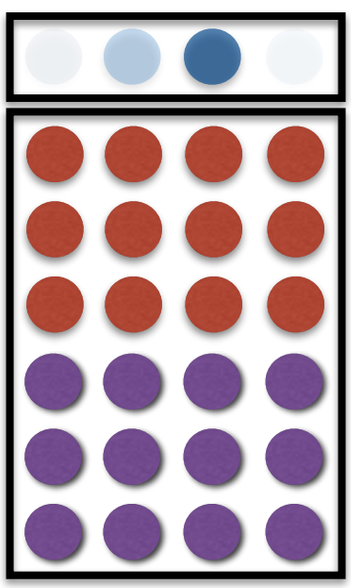


Ich möchte ein Bier

Attention history:

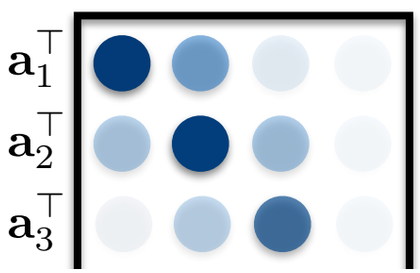


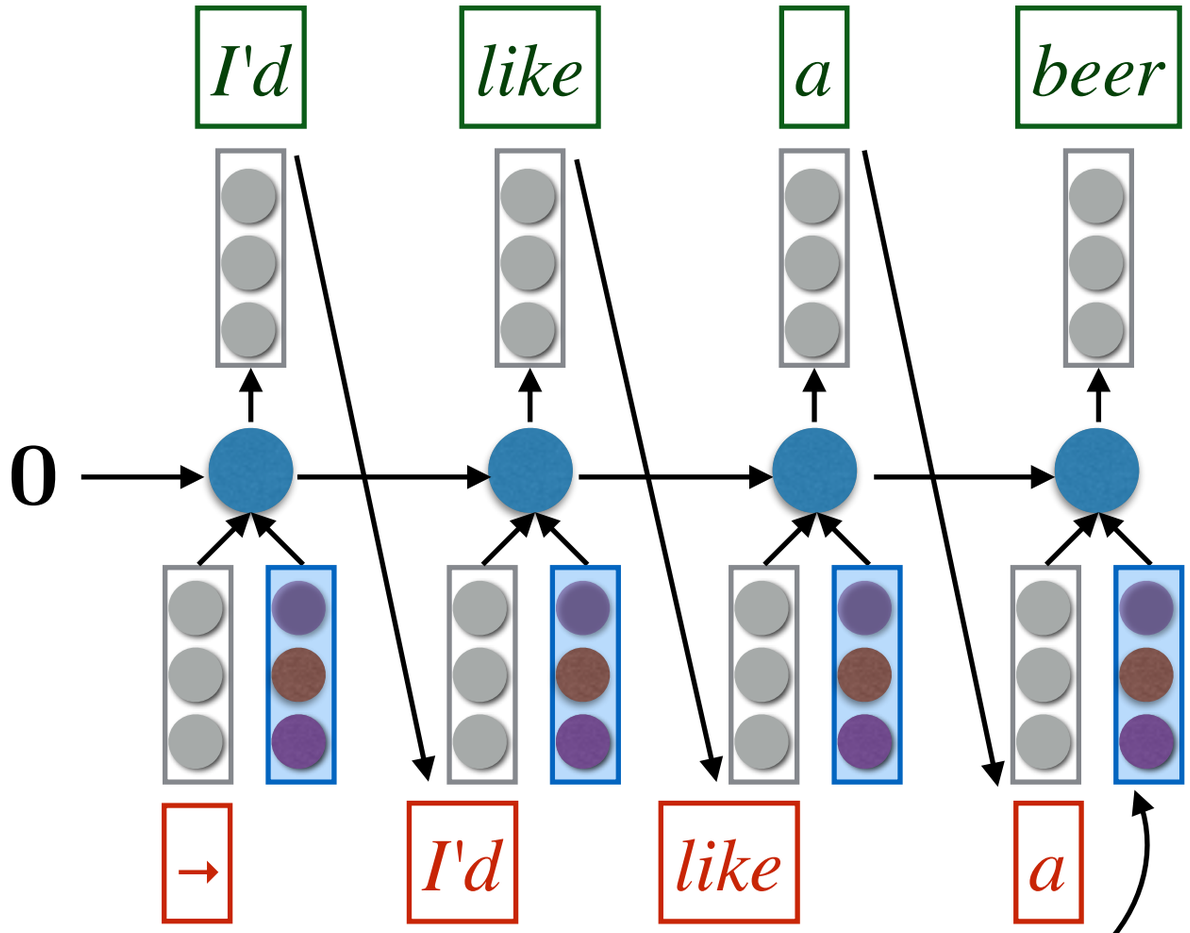


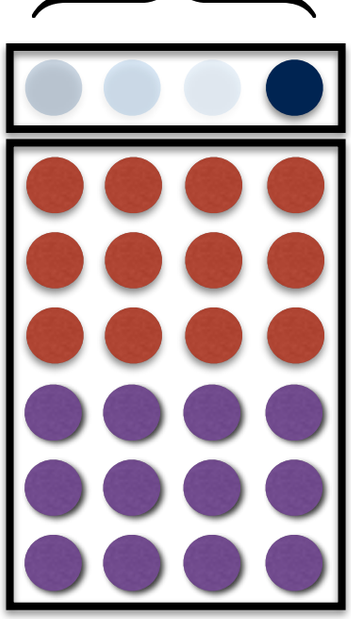


Ich möchte ein Bier

Attention history:

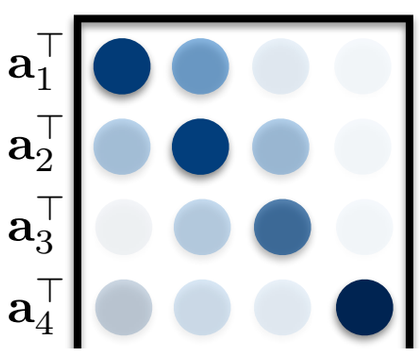


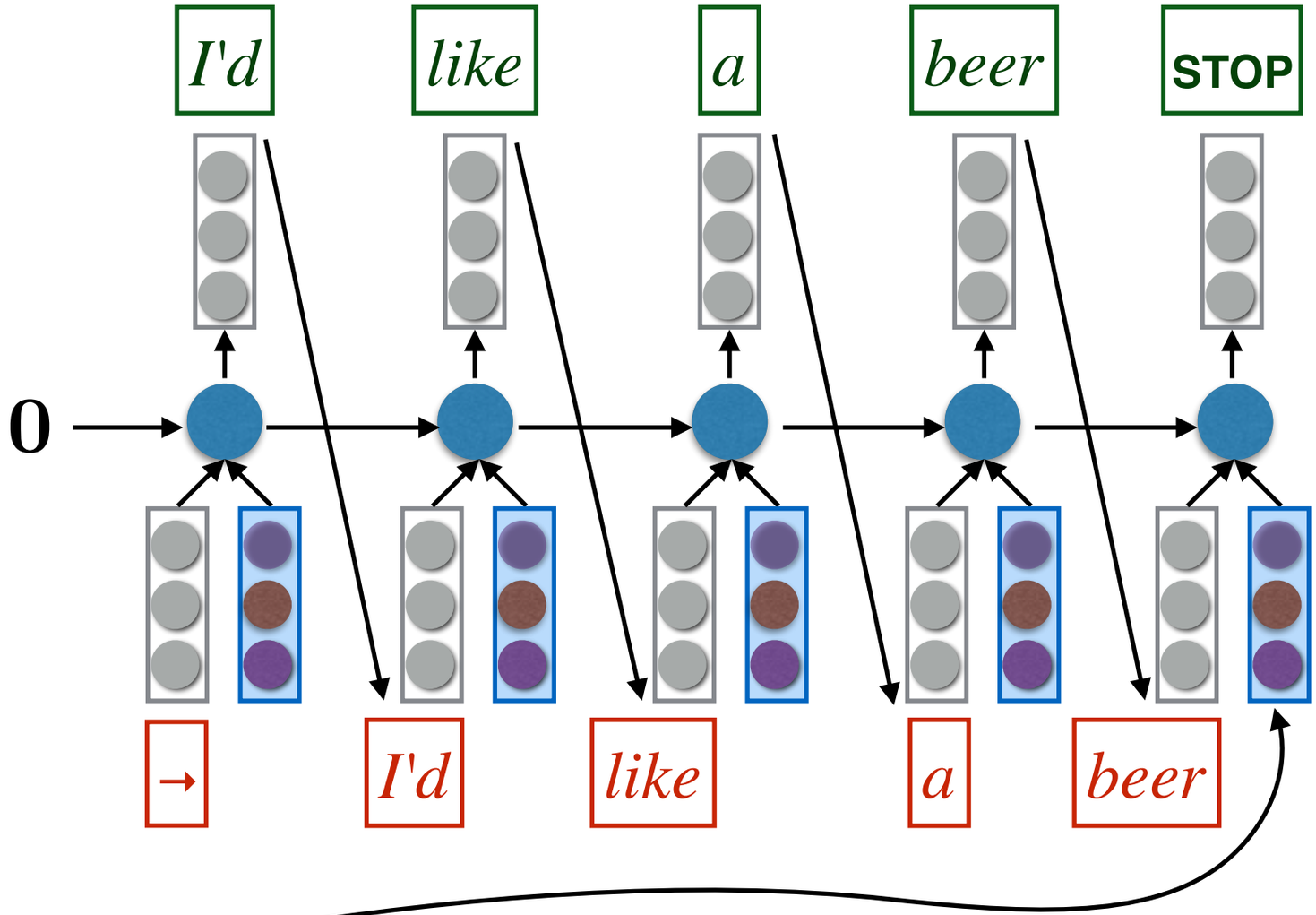


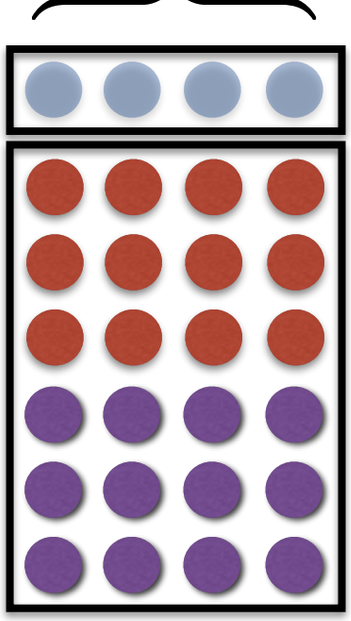


Ich möchte ein Bier

Attention history:

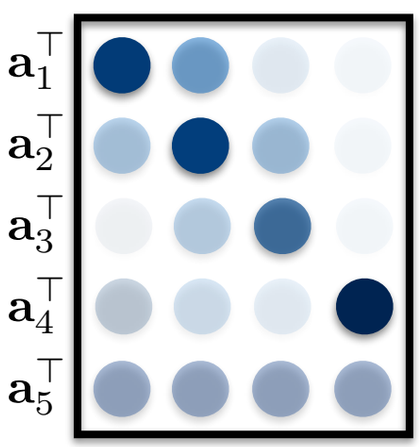




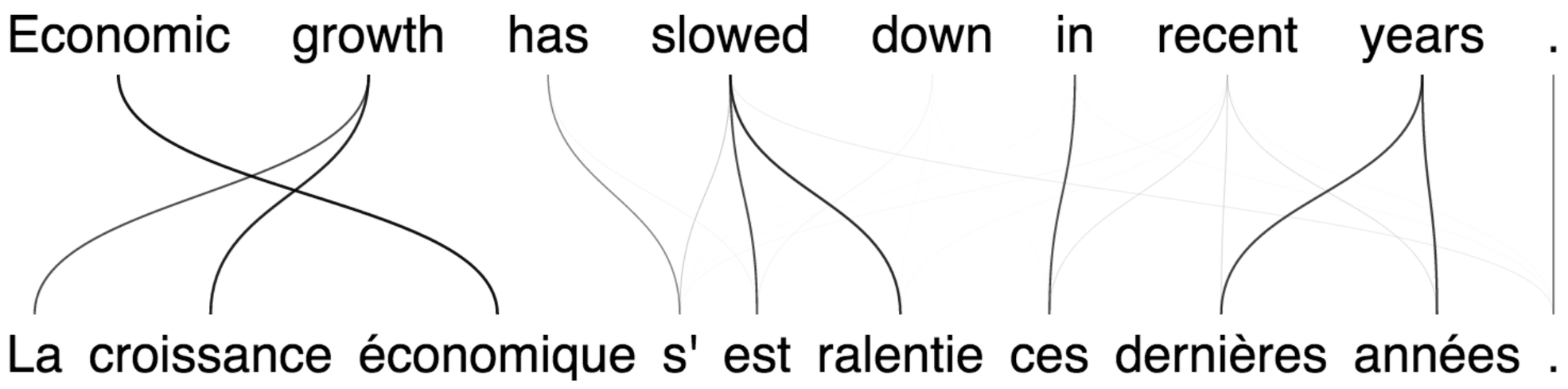


Ich möchte ein Bier

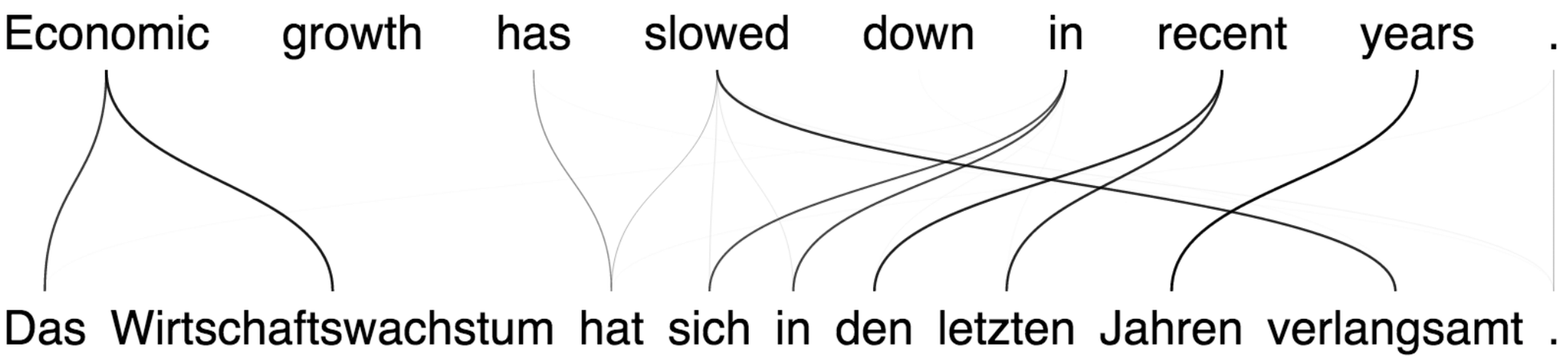
Attention history:



English-French



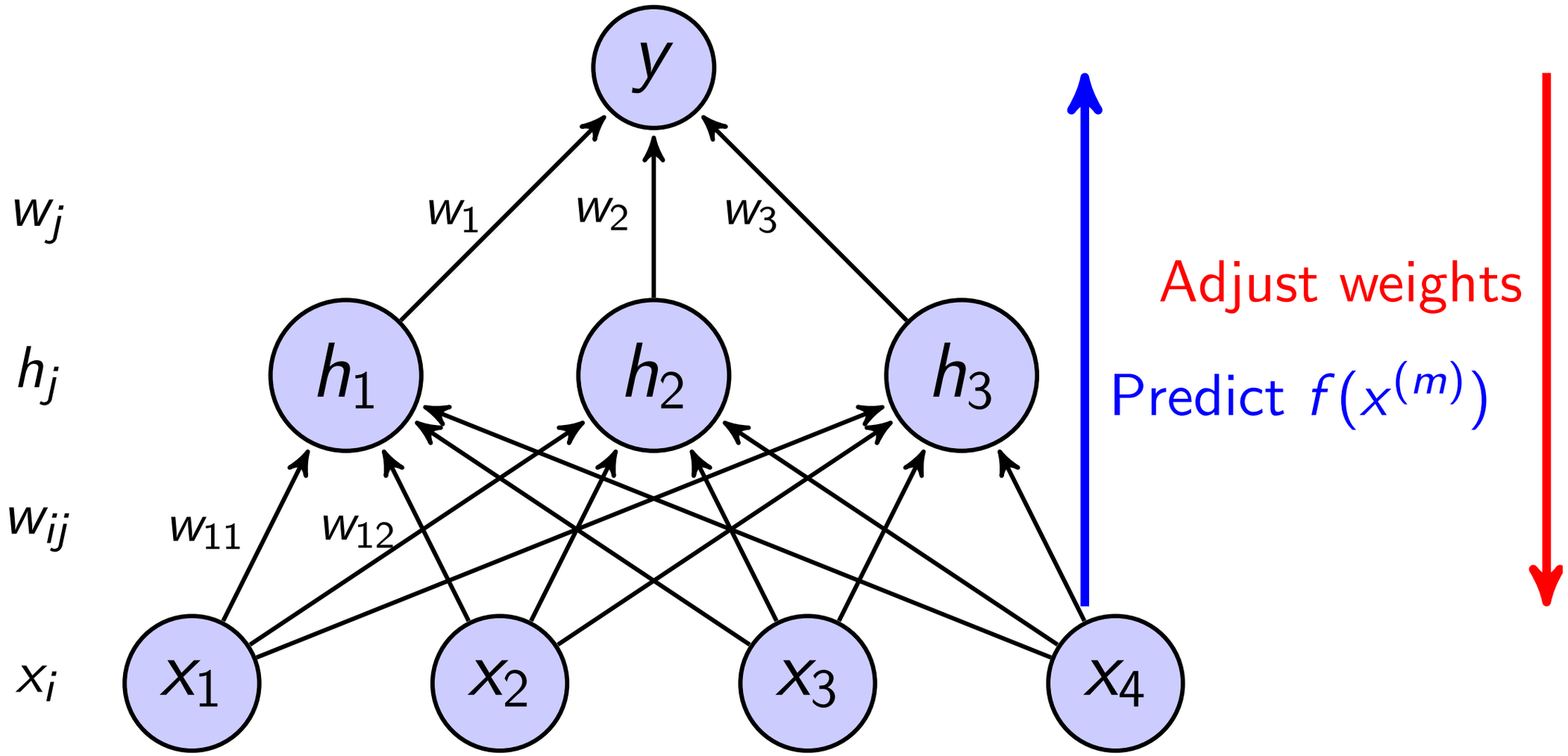
English-German



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 **F** has $|\boldsymbol{f}|$ columns, $\mathbf{u}_t$ has $|\boldsymbol{f}|$ rows)
    - Exponentiate and normalize to 1: $\mathbf{a}_t = \mathrm{softmax}(\mathbf{u}_t)$
      (called $\boldsymbol{\alpha}_t$ in the paper)
    - Finally, the ***input source vector*** for time *t* is $\mathbf{c}_t = \mathbf{F}\mathbf{a}_t$

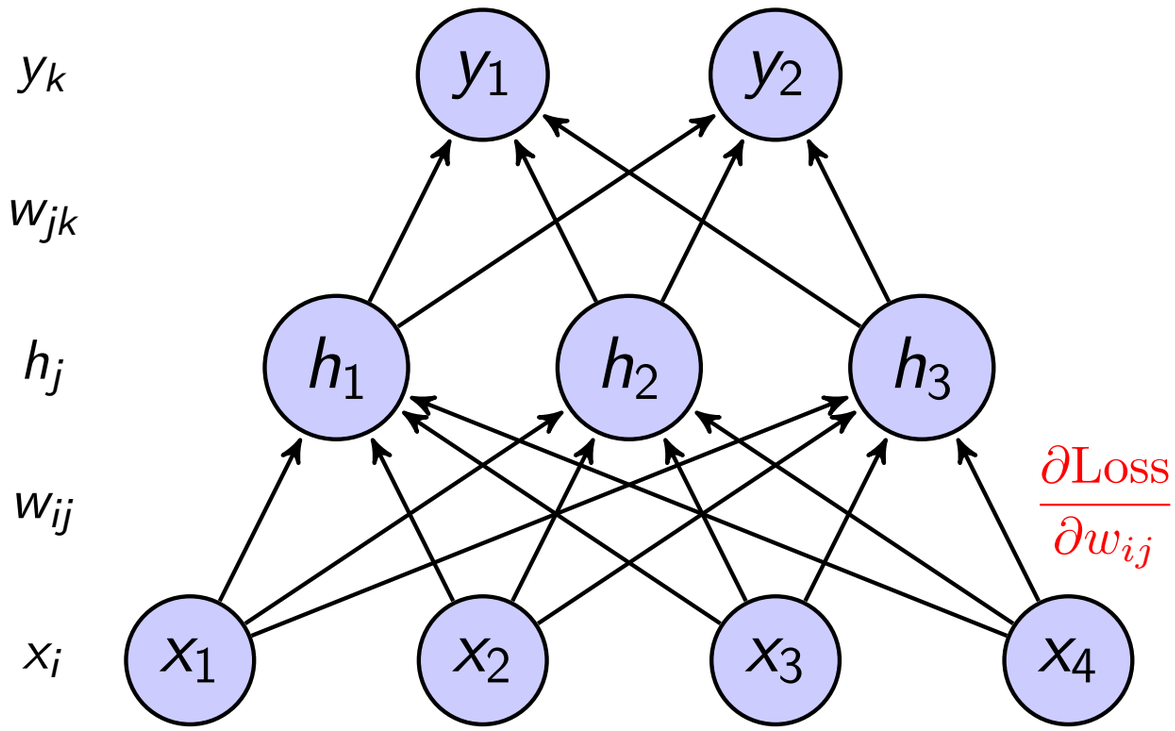
Training Neural Nets: Back-propagation



1. For each sample, compute
$f(x^{(m)}) = \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} x_i^{(m)}))$
2. If $f(x^{(m)}) \neq y^{(m)}$, back-propagate error and adjust weights $\{w_{ij}, w_j\}$.

Derivatives of the weights

Assume two outputs (y_1, y_2) per input x,
and loss per sample: $Loss = \sum_k \frac{1}{2} [\sigma(in_k) - y_k]^2$



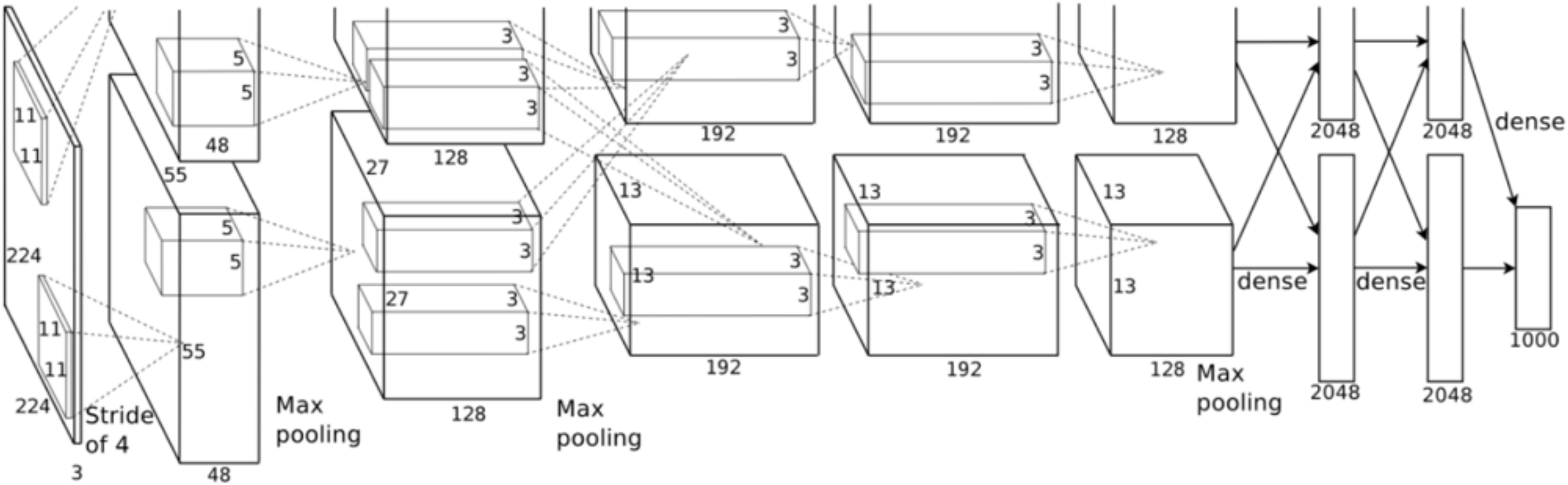
$$\frac{\partial Loss}{\partial w_{jk}} = \frac{\partial Loss}{\partial in_k} \frac{\partial in_k}{\partial w_{jk}} = \delta_k \frac{\partial(\sum_j w_{jk} h_j)}{\partial w_{jk}} = \delta_k h_j$$

$$\frac{\partial Loss}{\partial w_{ij}} = \frac{\partial Loss}{\partial in_j} \frac{\partial in_j}{\partial w_{ij}} = \delta_j \frac{\partial(\sum_j w_{ij} x_i)}{\partial w_{ij}} = \delta_j x_i$$

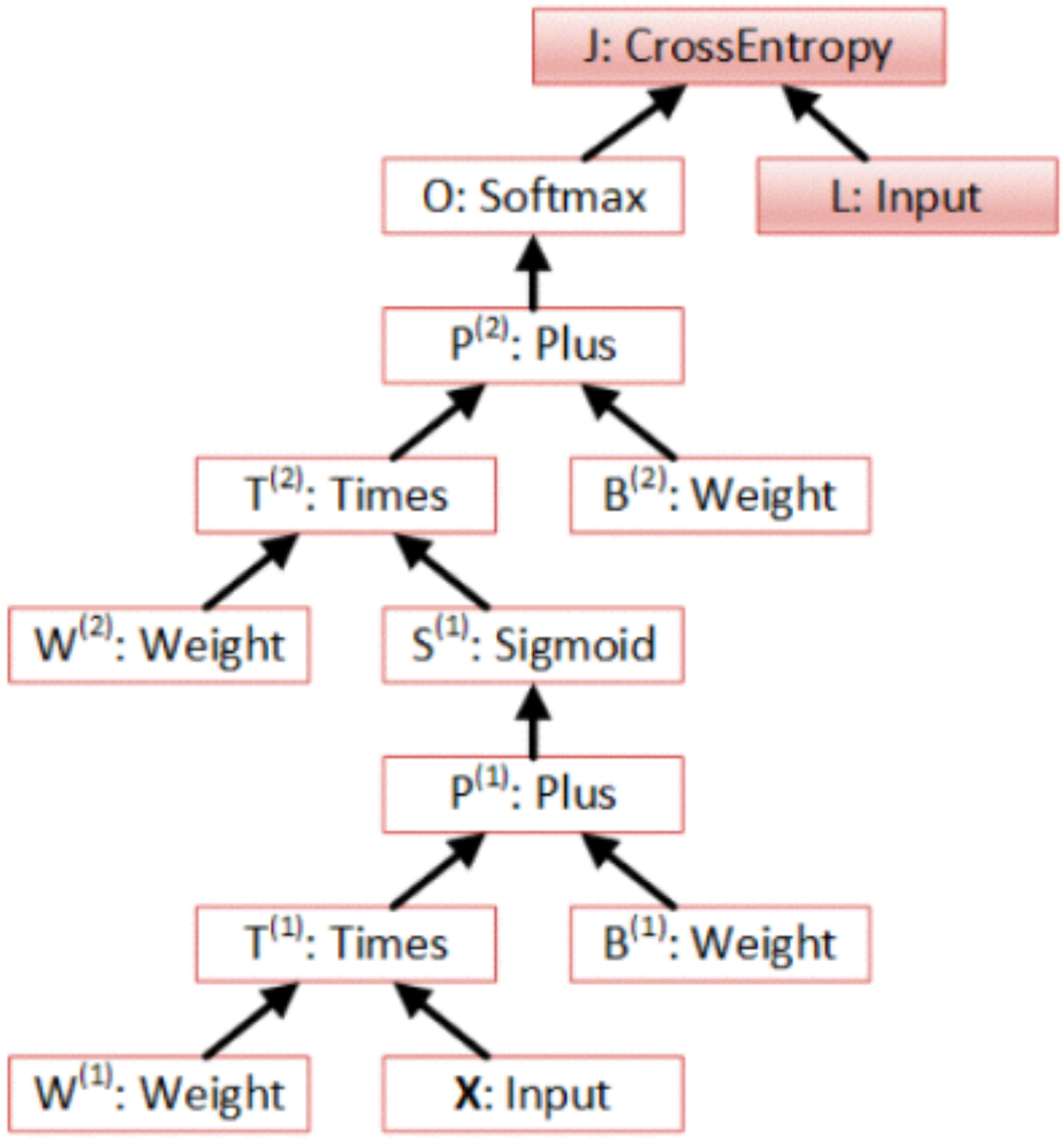
$$\delta_k = \frac{\partial}{\partial in_k} \left(\sum_k \frac{1}{2} [\sigma(in_k) - y_k]^2 \right) = [\sigma(in_k) - y_k] \sigma'(in_k)$$

$$\delta_j = \sum_k \frac{\partial Loss}{\partial in_k} \frac{\partial in_k}{\partial in_j} = \sum_k \delta_k \cdot \frac{\partial}{\partial in_j} \left(\sum_j w_{jk} \sigma(in_j) \right) = [\sum_k \delta_k w_{jk}] \sigma'(in_j)$$

Do you want to take the derivative of this?



- All computations can be viewed on a graph:
- e.g. Computing the loss term of the MLP



Since the derivatives and partials of each function are known, this is fully automatable (Torch, Theano, Tensorflow, etc.)

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

Summary II

- Nonlinear classification, automatic feature induction, and automatic differentiation are good.
- But just how good are they?

[–] **Programmering** 10 points 1 month ago*

What do you believe that AI capabilities could be in the close future?

permalink

[–] **wojzaremba** OpenAI 16 points 1 month ago

Speech recognition and machine translation between any languages should be fully solvable. We should see many more uses of computer vision applications, like for instance: - app that recognizes number of calories in food - app that tracks all products in a supermarket at all times - burglary detection - robotics

Coursework 2

- Implement this. (Follows from lab 2).
- Posted shortly.
- Due in three weeks.
- Start as soon as you get it… if you wait, you WILL NOT FINISH. Neural models take a long time to train and decode. (How many flops in a matrix multiplication?)