# Recurrent Neural Networks

Yoav Goldberg

# Dealing with Sequences

- For an input sequence **x1**,...,**xn**, we can:
  - If *n* is **fixed**: *concatenate* and feed into an MLP.
  - *sum* the vectors (*CBOW*) and feed into an MLP.
  - Break the sequence into *windows.* Find n-gram embedding, sum into an MLP.
  - Find good ngrams using ConvNet, using *pooling* (either sum/avg or max) to combine to a single vector.

# Dealing with Sequences

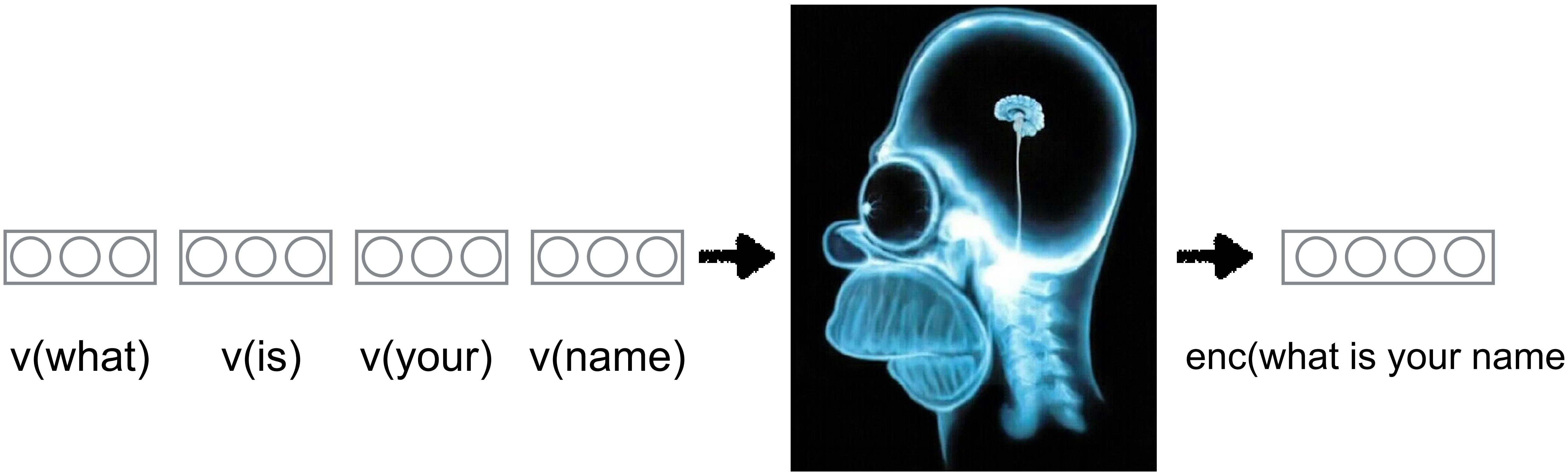
- For an input sequence **x1**,...,**xn**, we can:

Some of these approaches consider **local** word order (which ones?).

How can we consider **global** word order?

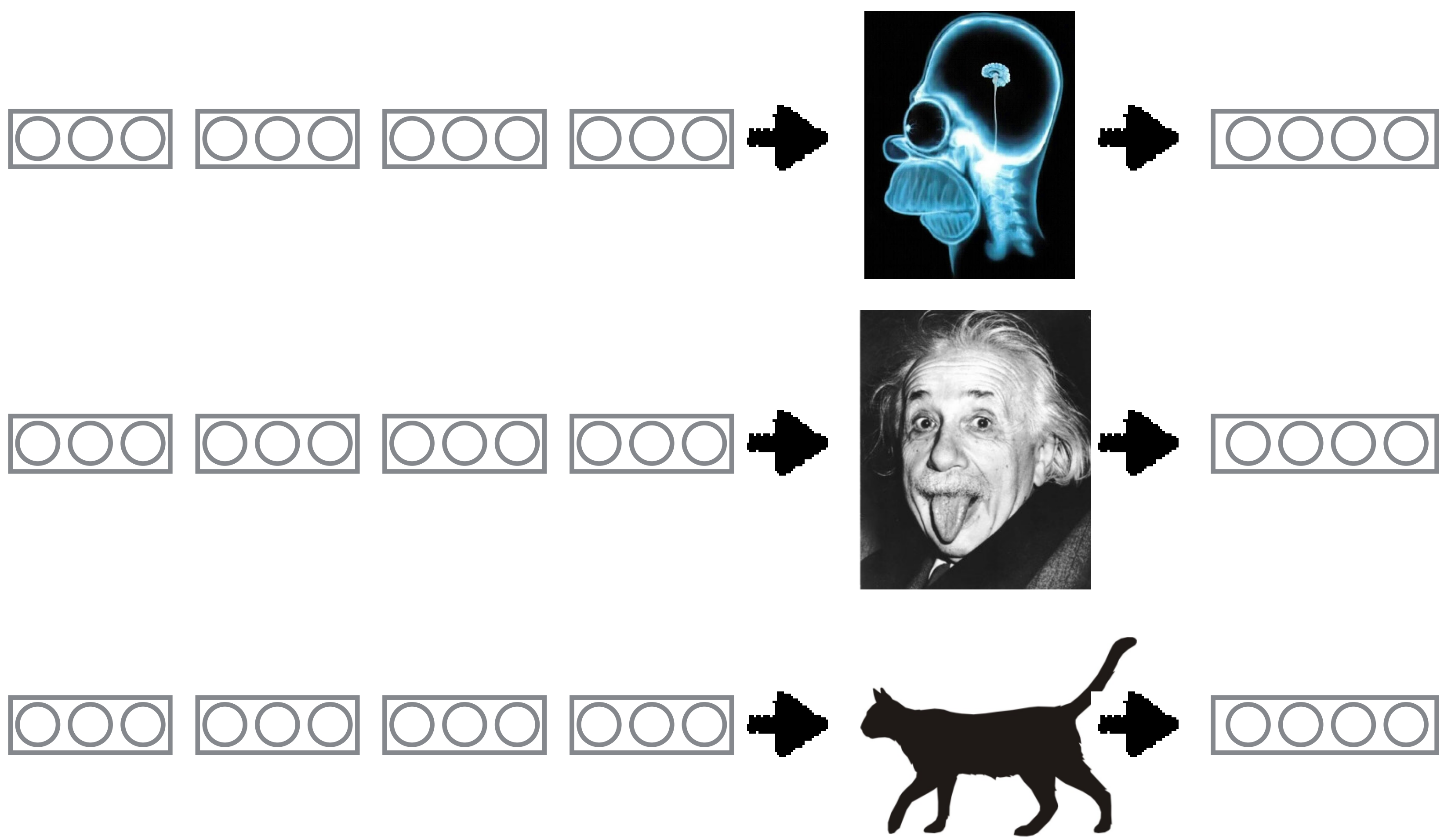
  - Find good ngrams using ConvNet, using *pooling* (either sum/avg or max) to combine to a single vector.

# Recurrent Neural Networks




- Very strong models of sequential data.
- **Trainable** function from *n* vectors to a single vector.

# Recurrent Neural Networks



- There are different variants (implementations).
- So far, we focused on the interface level.

# Recurrent Neural Networks

$$RNN(\mathbf{s_0}, \mathbf{x_{1:n}}) = \mathbf{s_n}, \mathbf{y_n}$$

$$\mathbf{x_i} \in \mathbb{R}^{d_{in}},\ \mathbf{y_i} \in \mathbb{R}^{d_{out}},\ \mathbf{s_i} \in \mathbb{R}^{f(d_{out})}$$

- Very strong models of sequential data.
- **Trainable** function from *n* vectors to a single* vector.

# Recurrent Neural Networks

$$RNN(\mathbf{s_0}, \mathbf{x_{1:n}}) = \mathbf{s_n}, \mathbf{y_n}$$

*this one is internal. we only care about the **y**

$$\mathbf{x_i} \in \mathbb{R}^{d_{in}},\ \mathbf{y_i} \in \mathbb{R}^{d_{out}},\ \mathbf{s_i} \in \mathbb{R}^{f(d_{out})}$$

- Very strong models of sequential data.
- **Trainable** function from *n* vectors to a single* vector.

# Recurrent Neural Networks

$$RNN(\mathbf{s_0}, \mathbf{x_{1:n}}) = \mathbf{s_n}, \mathbf{y_n}$$

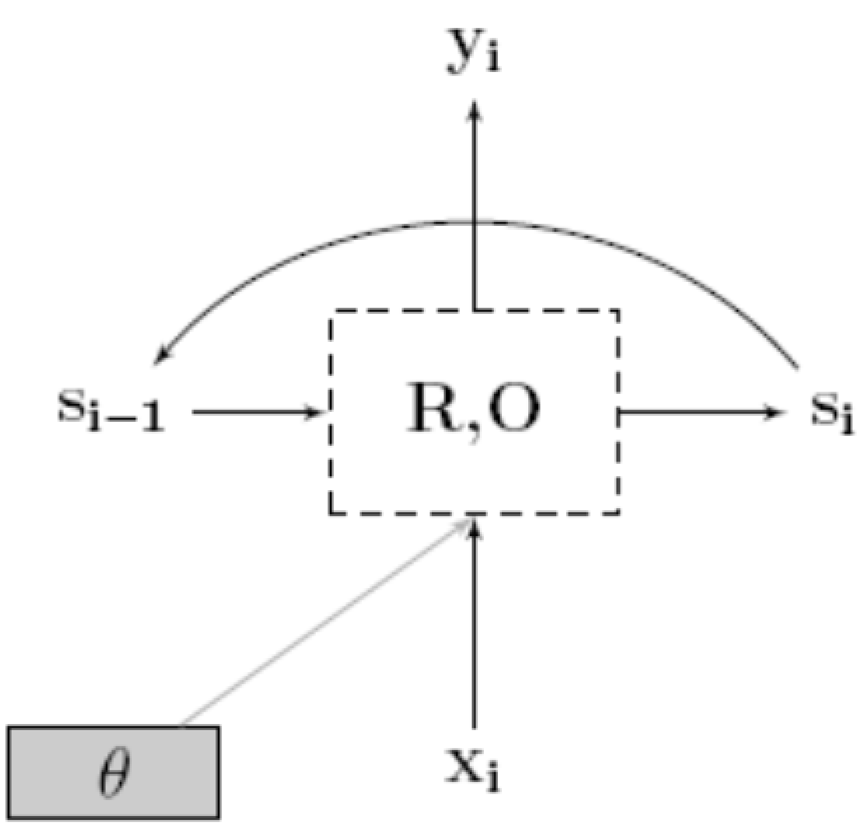
$$\mathbf{s_i} = R(\mathbf{s_{i-1}}, \mathbf{x_i})$$

$$\mathbf{y_i} = O(\mathbf{s_i})$$

$$\mathbf{x_i} \in \mathbb{R}^{d_{in}},\ \mathbf{y_i} \in \mathbb{R}^{d_{out}},\ \mathbf{s_i} \in \mathbb{R}^{f(d_{out})}$$

- **Recursively defined.**
- There's a vector $\mathbf{y_i}$ for every prefix $\mathbf{x_{1:i}}$

# Recurrent Neural Networks

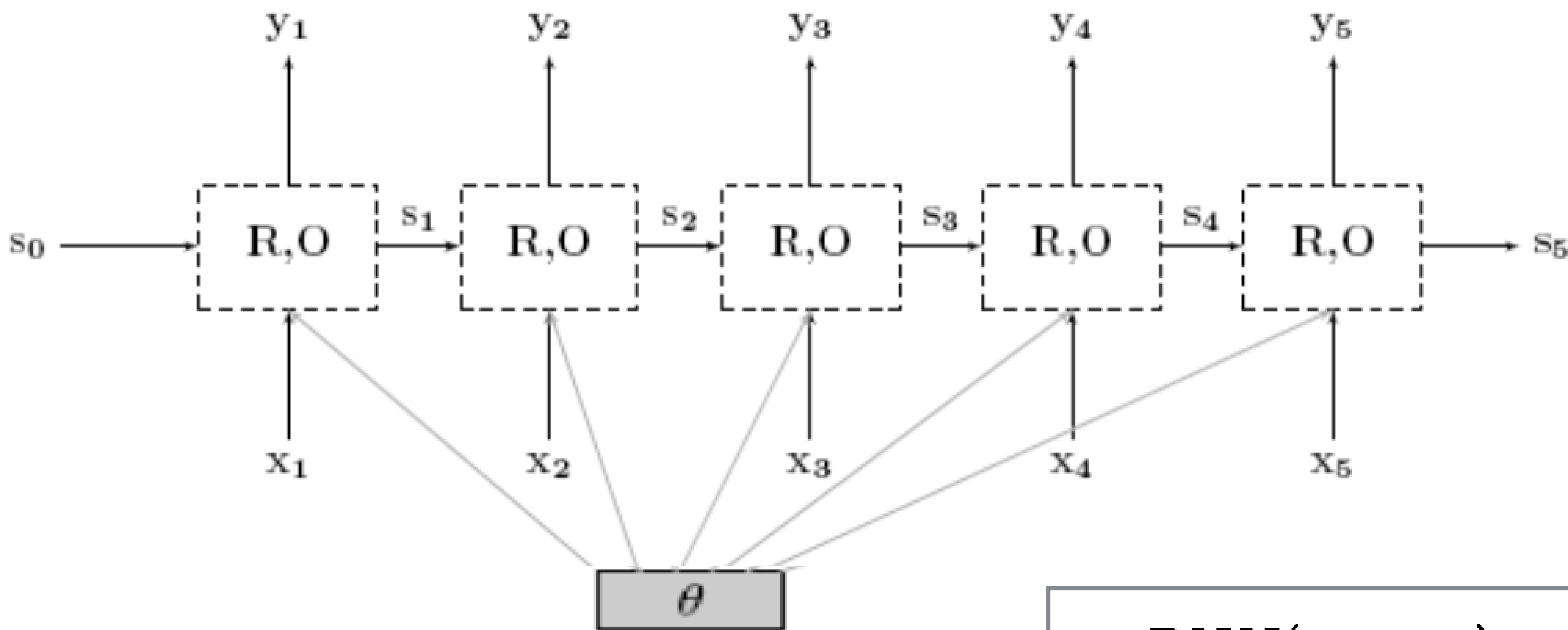



$$RNN(\mathbf{s_0}, \mathbf{x_{1:n}}) = \mathbf{s_n}, \mathbf{y_n}$$
$$\mathbf{s_i} = R(\mathbf{s_{i-1}}, \mathbf{x_i})$$
$$\mathbf{y_i} = O(\mathbf{s_i})$$

$$\mathbf{x_i} \in \mathbb{R}^{d_{in}},\ \mathbf{y_i} \in \mathbb{R}^{d_{out}},\ \mathbf{s_i} \in \mathbb{R}^{f(d_{out})}$$

- **Recursively defined**.
- There's a vector $\mathbf{y_i}$ for every prefix $\mathbf{x_{1:i}}$

# Recurrent Neural Networks




for every finite input sequence,
can unroll the recursion.

$$RNN(\mathbf{s_0}, \mathbf{x_{1:n}}) = \mathbf{s_n}, \mathbf{y_n}$$

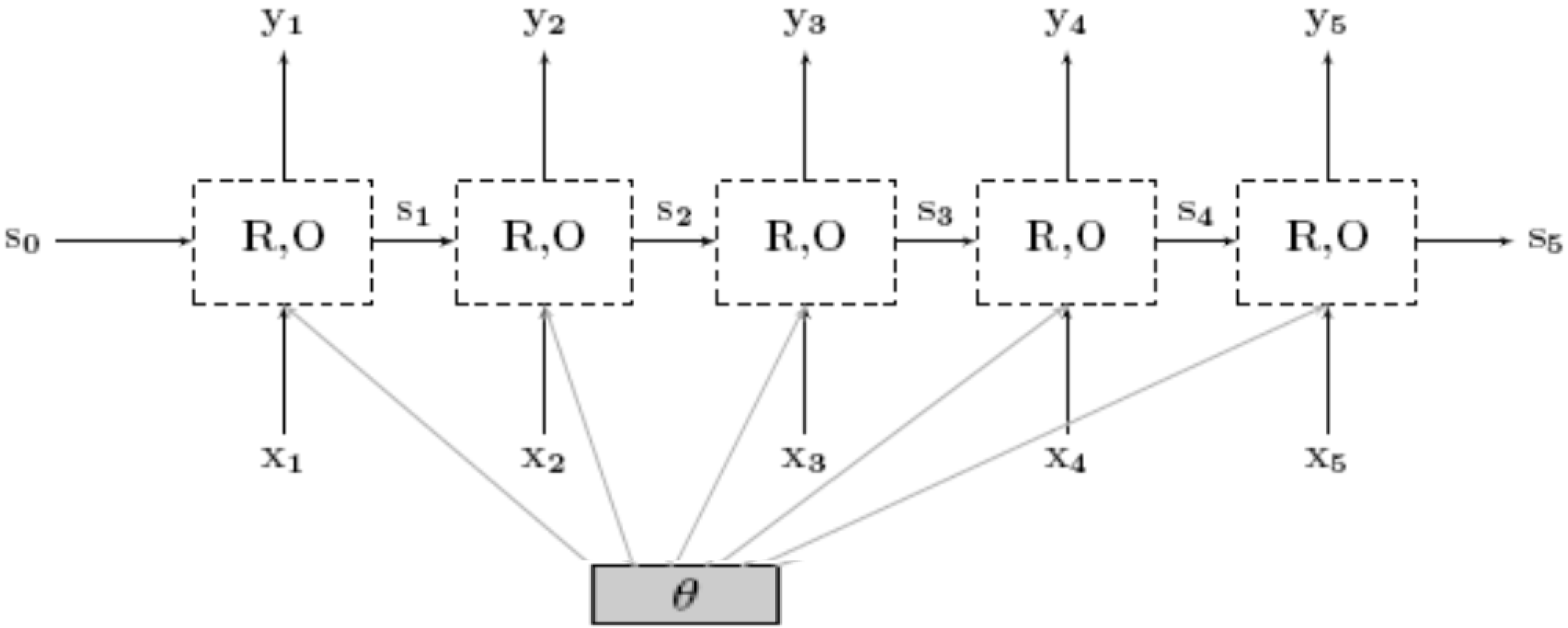
$$\mathbf{s_i} = R(\mathbf{s_{i-1}}, \mathbf{x_i})$$

$$\mathbf{y_i} = O(\mathbf{s_i})$$

$$\mathbf{x_i} \in \mathbb{R}^{d_{in}}, \ \mathbf{y_i} \in \mathbb{R}^{d_{out}}, \ \mathbf{s_i} \in \mathbb{R}^{f(d_{out})}$$

- Recursively defined.
- There's a vector $\mathbf{y_i}$ for every prefix $\mathbf{x_{1:i}}$

# Recurrent Neural Networks




for every finite input sequence,
can unroll the recursion.

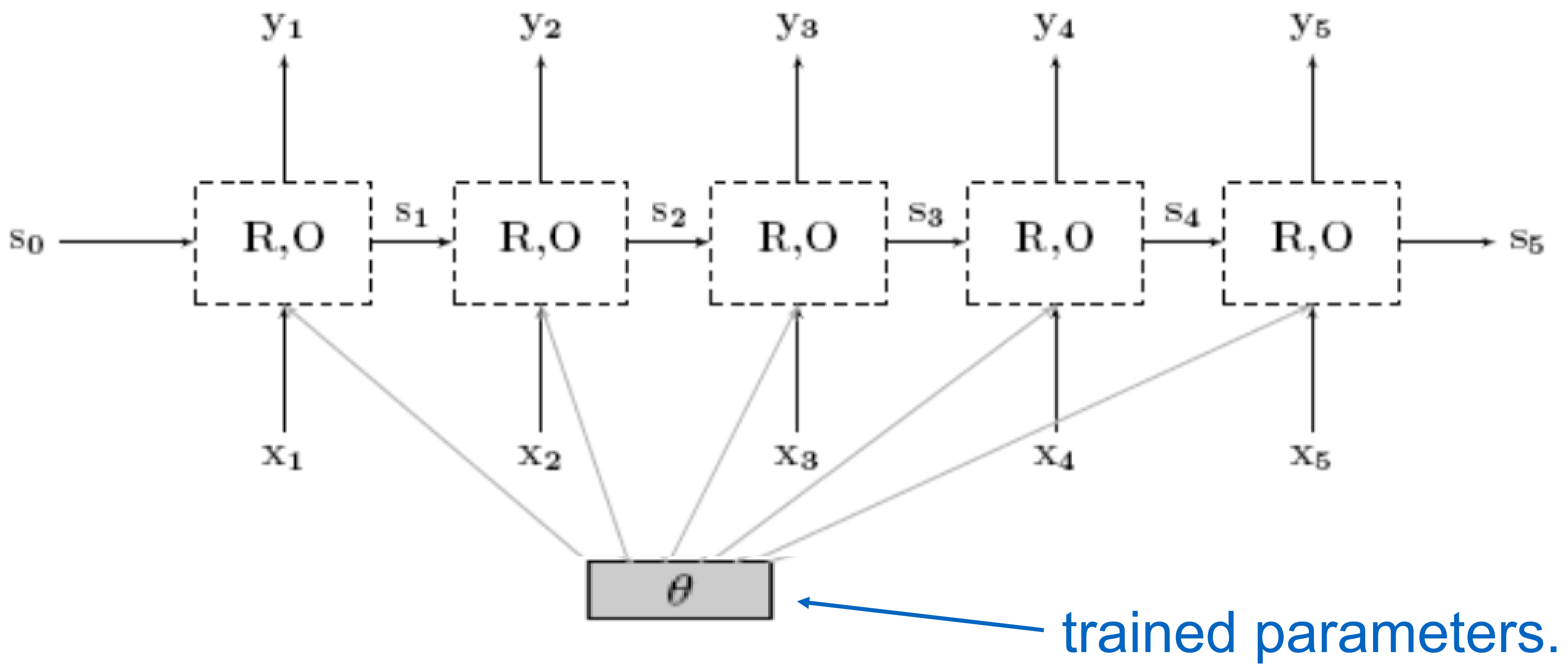
# Recurrent Neural Networks

$$\mathbf{y_4} = O(\mathbf{s_4})$$

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

- The output vector $\mathbf{y_i}$ depends on **all** inputs $\mathbf{x_{1:i}}$

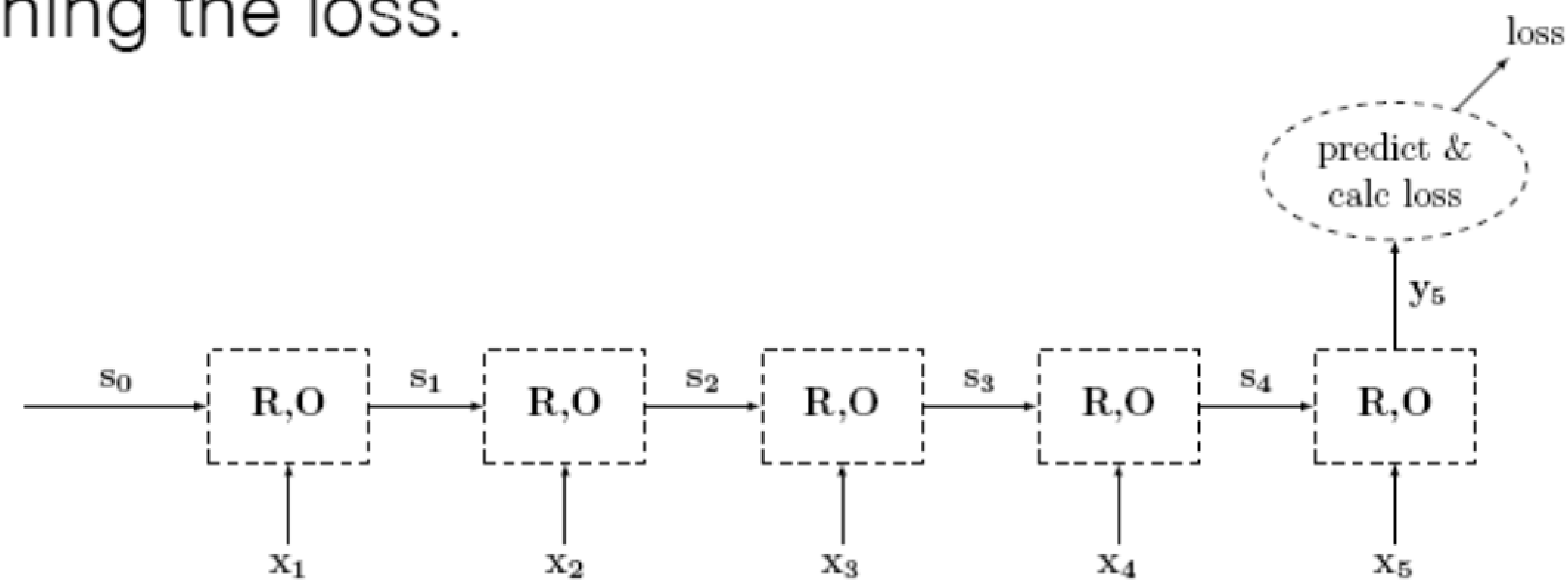
# Recurrent Neural Networks




- **But we can train them.** → **define function form** / define loss

# Recurrent Neural Networks for Text Classification

Defining the loss.




**Acceptor**: predict something from end state.
Backprop the error all the way back.
Train the network to capture meaningful information

# CBOW as an RNN

$$R_{CBOW}(\mathbf{s_{i-1}}, \mathbf{x_i}) = \mathbf{s_{i-1}} + \mathbf{x_i}$$

(what are the parameters?)

# CBOW as an RNN

$$R_{CBOW}(\mathbf{s_{i-1}}, \mathbf{x_i}) = \mathbf{s_{i-1}} + \mathbf{x_i}$$

(what are the parameters?)

$$R_{CBOW}(\mathbf{s_{i-1}}, x_i) = \mathbf{s_{i-1}} + \mathbf{E}_{[x_i]}$$

# CBOW as an RNN

$$R_{CBOW}(\mathbf{s_{i-1}}, x_i) = \mathbf{s_{i-1}} + \mathbf{E}_{[x_i]}$$

# CBOW as an RNN

$$R_{CBOW}(\mathbf{s_{i-1}}, x_i) = \underline{\tanh}(\mathbf{s_{i-1}} + \mathbf{E}_{[x_i]})$$

# Simple RNN (Elman RNN)

$$R_{SRNN}(\mathbf{s_{i-1}}, \mathbf{x_i}) = tanh(\mathbf{W^s} \cdot \mathbf{s_{i-1}} + \mathbf{W^x} \cdot \mathbf{x_i})$$

# Simple RNN (Elman RNN)

$$R_{SRNN}(\mathbf{s_{i-1}}, \mathbf{x_i}) = tanh(\mathbf{W^s} \cdot \mathbf{s_{i-1}} + \mathbf{W^x} \cdot \mathbf{x_i})$$

- Looks very simple.
- Theoretically very powerful.
- In practice not so much (hard to train).
- Why? Vanishing gradients.

# Simple RNN (Elman RNN)

$$R_{SRNN}(\mathbf{s_{i-1}}, \mathbf{x_i}) = tanh(\mathbf{W^s} \cdot \mathbf{s_{i-1}} + \mathbf{W^x} \cdot \mathbf{x_i})$$

Another view on behavior:

- RNN as a "computer":
  input **xi** arrives, memory **s** is updated.
- In the Elman RNN, **entire memory is written** at each time-step.
- entire memory = output!

# LSTM RNN

better controlled memory access

# continuous gates

# Differentiable "Gates"

- The main idea behind the LSTM is that you want to somehow control the "memory access".

- In a SimpleRNN:

$$R_{SRNN}(\mathbf{s_{i-1}}, \mathbf{x_i}) = tanh(\mathbf{W^s} \cdot \mathbf{s_{i-1}} + \mathbf{W^x} \cdot \mathbf{x_i})$$

read previous state memory ($\mathbf{W^s}$) — write new input ($\mathbf{W^x}$)

- All the memory gets overwritten

# Vector "Gates"

- We'd like to:
  * Selectively read from some memory "cells".
  * Selectively write to some memory "cells".

# Vector "Gates"

- We'd like to:
  * Selectively read from some memory "cells".
  * Selectively write to some memory "cells".

- A gate function:

$$\underset{\mathbf{g}}{\begin{bmatrix}0\\1\\0\\0\\0\\1\end{bmatrix}} \odot \underset{\mathbf{x}}{\begin{bmatrix}10\\11\\12\\13\\14\\15\end{bmatrix}}$$

(element-wise multiplication)

gate controls access

vector of values

# Vector "Gates"

- We'd like to:
  * Selectively read from some memory "cells".
  * Selectively write to some memory "cells".

- A gate function: $\mathbf{s_{i-1}} \odot \mathbf{g}$ $\quad\quad$ $\mathbf{g} \in \{0, 1\}^d$

vector of values

gate controls access

# Vector "Gates"

- Using the gate function to control access:

$$\mathbf{s_i} \leftarrow \mathbf{s_{i-1}} \odot \mathbf{g^r} + \mathbf{x_i} \odot \mathbf{g^w} \qquad \mathbf{g} \in \{0,1\}^d$$

which cells to read

which cells to write

# Vector "Gates"

- Using the gate function to control access:

$$\mathbf{s_i} \leftarrow \mathbf{s_{i-1}} \odot \mathbf{g^r} + \mathbf{x_i} \odot \mathbf{g^w} \qquad \mathbf{g} \in \{0,1\}^d$$

which cells to read

which cells to write

- (can also tie them: $\mathbf{g^r} = 1 - \mathbf{g^w}$ )

# Vector "Gates"

$$\underset{\mathbf{s}'}{\begin{bmatrix} 8 \\ 11 \\ 3 \\ 7 \\ 5 \\ 15 \end{bmatrix}} \leftarrow \underset{\mathbf{g}}{\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}} \odot \underset{\mathbf{x}}{\begin{bmatrix} 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{bmatrix}} + \underset{(\mathbf{1}-\mathbf{g})}{\begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}} \odot \underset{\mathbf{s}}{\begin{bmatrix} 8 \\ 9 \\ 3 \\ 7 \\ 5 \\ 8 \end{bmatrix}}$$

# Differentiable "Gates"

- **Problem with the gates**:
  * they are fixed.
  * they don't depend on the input or the output.

# Differentiable "Gates"

- **Problem with the gates**:
  * they are fixed.
  * they don't depend on the input or the output.

- Solution: make them smooth, input dependent, and trainable.

$$\mathbf{g^r} = \sigma(\mathbf{W} \cdot \mathbf{x_i} + \mathbf{U} \cdot \mathbf{s_{i-1}})$$

"almost 0"
or
"almost 1"

function of input and state

# LSTM
## (Long short-term Memory)

- The LSTM is a specific combination of gates.

$$R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) = [\mathbf{c_j}; \mathbf{h_j}]$$
$$\mathbf{c_j} = \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i}$$

$$\mathbf{i} = \sigma(\mathbf{W^{xi}} \cdot \mathbf{x_j} + \mathbf{W^{hi}} \cdot \mathbf{h_{j-1}})$$
$$\mathbf{f} = \sigma(\mathbf{W^{xf}} \cdot \mathbf{x_j} + \mathbf{W^{hf}} \cdot \mathbf{h_{j-1}})$$

$$\mathbf{g} = \tanh(\mathbf{W^{xg}} \cdot \mathbf{x_j} + \mathbf{W^{hg}} \cdot \mathbf{h_{j-1}})$$

# LSTM
(Long short-term Memory)

- The LSTM is a specific combination of gates.

$$R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) = [\mathbf{c_j}; \mathbf{h_j}]$$

$$\mathbf{c_j} = \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i}$$

$$\mathbf{h_j} = \tanh(\mathbf{c_j})$$

$$\mathbf{i} = \sigma(\mathbf{W^{xi}} \cdot \mathbf{x_j} + \mathbf{W^{hi}} \cdot \mathbf{h_{j-1}})$$

$$\mathbf{f} = \sigma(\mathbf{W^{xf}} \cdot \mathbf{x_j} + \mathbf{W^{hf}} \cdot \mathbf{h_{j-1}})$$

$$\mathbf{g} = \tanh(\mathbf{W^{xg}} \cdot \mathbf{x_j} + \mathbf{W^{hg}} \cdot \mathbf{h_{j-1}})$$

# LSTM
(Long short-term Memory)

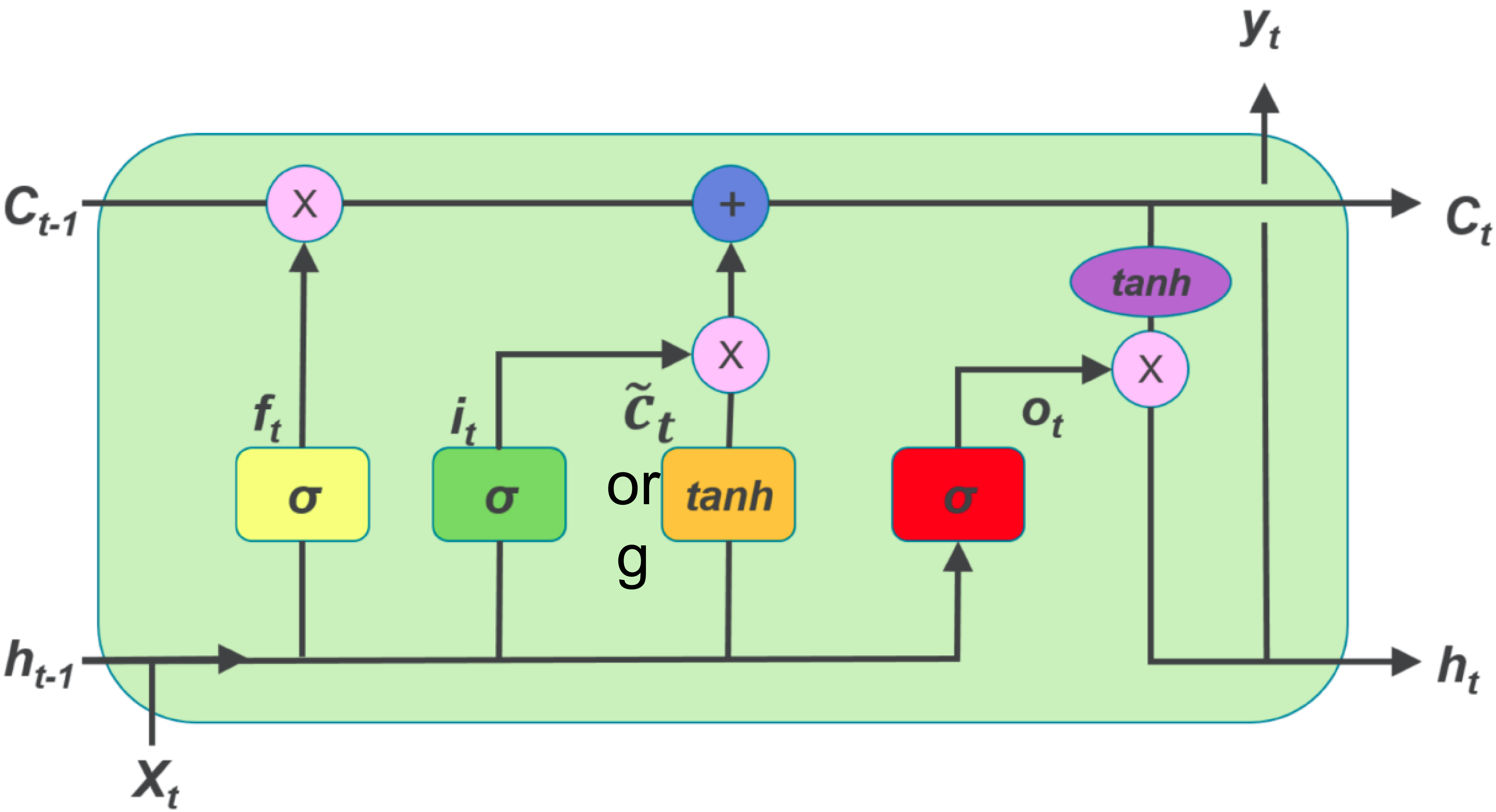
- The LSTM is a specific combination of gates.

$$
\begin{aligned}
R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) =& [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} =& \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} =& \tanh(\mathbf{c_j}) \\
\mathbf{i} =& \sigma(\mathbf{W^{xi}} \cdot \mathbf{x_j} + \mathbf{W^{hi}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{f} =& \sigma(\mathbf{W^{xf}} \cdot \mathbf{x_j} + \mathbf{W^{hf}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{o} =& \sigma(\mathbf{W^{xo}} \cdot \mathbf{x_j} + \mathbf{W^{ho}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{g} =& \tanh(\mathbf{W^{xg}} \cdot \mathbf{x_j} + \mathbf{W^{hg}} \cdot \mathbf{h_{j-1}})
\end{aligned}
$$

# LSTM
## (Long short-term Memory)

- The LSTM is a specific combination of gates.

$$
\begin{aligned}
R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) &= [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} &= \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} &= \tanh(\mathbf{c_j}) \odot \mathbf{o} \\
\mathbf{i} &= \sigma(\mathbf{W^{xi}} \cdot \mathbf{x_j} + \mathbf{W^{hi}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{f} &= \sigma(\mathbf{W^{xf}} \cdot \mathbf{x_j} + \mathbf{W^{hf}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{o} &= \sigma(\mathbf{W^{xo}} \cdot \mathbf{x_j} + \mathbf{W^{ho}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{g} &= \tanh(\mathbf{W^{xg}} \cdot \mathbf{x_j} + \mathbf{W^{hg}} \cdot \mathbf{h_{j-1}})
\end{aligned}
$$



$y_t$
$C_{t-1}$
X
+
$C_t$
tanh
X
X
$f_t$
$i_t$
$\tilde{c}_t$
$o_t$
σ
σ
or g
tanh
σ
$h_{t-1}$
$h_t$
$X_t$

# GRU
## (Gated Recurrent Unit)

- The GRU is a different combination of gates.

$$\mathbf{s_j} = R_{\mathrm{GRU}}(\mathbf{s_{j-1}}, \mathbf{x_j}) = (1 - \mathbf{z}) \odot \mathbf{s_{j-1}} + \mathbf{z} \odot \tilde{\mathbf{s_j}}$$

$$\mathbf{z} = \sigma(\mathbf{x_j}\mathbf{W^{xz}} + \mathbf{s_{j-1}}\mathbf{W^{sz}})$$

$$\mathbf{r} = \sigma(\mathbf{x_j}\mathbf{W^{xr}} + \mathbf{s_{j-1}}\mathbf{W^{sr}})$$

$$\tilde{\mathbf{s_j}} = \tanh(\mathbf{x_j}\mathbf{W^{xs}} + (\mathbf{r} \odot \mathbf{s_{j-1}})\mathbf{W^{sg}})$$

# GRU vs LSTM

- The GRU and the LSTM are very similar ideas.
- Invented independently of the LSTM, almost two decades later.

# GRU
## (Gated Recurrent Unit)

- The GRU formulation:

$$\mathbf{s_j} = R_{\mathrm{GRU}}(\mathbf{s_{j-1}}, \mathbf{x_j}) =$$

**Proposal state:** $\tilde{\mathbf{s_j}} = \tanh(\mathbf{x_j}\mathbf{W^{xs}} + (\mathbf{r} \odot \mathbf{s_{j-1}})\mathbf{W^{sg}})$

# GRU
## (Gated Recurrent Unit)

- The GRU formulation:

$$\mathbf{s_j} = R_{\mathrm{GRU}}(\mathbf{s_{j-1}}, \mathbf{x_j}) =$$

**gate controlling effect of prev on proposal:**

$$\mathbf{r} = \sigma(\mathbf{x_j}\mathbf{W^{xr}} + \mathbf{s_{j-1}}\mathbf{W^{sr}})$$

$$\tilde{\mathbf{s_j}} = \tanh(\mathbf{x_j}\mathbf{W^{xs}} + (\mathbf{r} \odot \mathbf{s_{j-1}})\mathbf{W^{sg}})$$

# GRU
## (Gated Recurrent Unit)

**blend of old state and proposal state**

$$\mathbf{s_j} = R_{\text{GRU}}(\mathbf{s_{j-1}}, \mathbf{x_j}) = (1 - \mathbf{z}) \odot \mathbf{s_{j-1}} + \mathbf{z} \odot \tilde{\mathbf{s_j}}$$

$$\mathbf{r} = \sigma(\mathbf{x_j} \mathbf{W^{xr}} + \mathbf{s_{j-1}} \mathbf{W^{sr}})$$

$$\tilde{\mathbf{s_j}} = \tanh(\mathbf{x_j} \mathbf{W^{xs}} + (\mathbf{r} \odot \mathbf{s_{j-1}}) \mathbf{W^{sg}})$$

# GRU
## (Gated Recurrent Unit)

$$\mathbf{s_j} = R_{\text{GRU}}(\mathbf{s_{j-1}}, \mathbf{x_j}) = (1 - \mathbf{z}) \odot \mathbf{s_{j-1}} + \mathbf{z} \odot \tilde{\mathbf{s_j}}$$

**gate for controlling the blend**

$$\mathbf{z} = \sigma(\mathbf{x_j}\mathbf{W^{xz}} + \mathbf{s_{j-1}}\mathbf{W^{sz}})$$

$$\mathbf{r} = \sigma(\mathbf{x_j}\mathbf{W^{xr}} + \mathbf{s_{j-1}}\mathbf{W^{sr}})$$

$$\tilde{\mathbf{s_j}} = \tanh(\mathbf{x_j}\mathbf{W^{xs}} + (\mathbf{r} \odot \mathbf{s_{j-1}})\mathbf{W^{sg}})$$

# GRU
## (Gated Recurrent Unit)

- The GRU formulation.

$$
\begin{aligned}
\mathbf{s_j} = R_{\mathrm{GRU}}(\mathbf{s_{j-1}}, \mathbf{x_j}) =& (\mathbf{1} - \mathbf{z}) \odot \mathbf{s_{j-1}} + \mathbf{z} \odot \tilde{\mathbf{s_j}} \\
\mathbf{z} =& \sigma(\mathbf{x_j}\mathbf{W^{xz}} + \mathbf{s_{j-1}}\mathbf{W^{sz}}) \\
\mathbf{r} =& \sigma(\mathbf{x_j}\mathbf{W^{xr}} + \mathbf{s_{j-1}}\mathbf{W^{sr}}) \\
\tilde{\mathbf{s_j}} =& \tanh(\mathbf{x_j}\mathbf{W^{xs}} + (\mathbf{r} \odot \mathbf{s_{j-1}})\mathbf{W^{sg}})
\end{aligned}
$$

# Other Variants

- Many other variants exist.
- Mostly perform similarly to each other.
  - Different tasks may work better with different variants.
- **The important idea is the differentiable gates.**

# LSTM

## (Long short-term Memory)

- The LSTM is formulation:

$$
\begin{aligned}
R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) =& [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} =& \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} =& \tanh(\mathbf{c_j}) \odot \mathbf{o} \\
\mathbf{i} =& \sigma(\mathbf{W^{xi}} \cdot \mathbf{x_j} + \mathbf{W^{hi}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{f} =& \sigma(\mathbf{W^{xf}} \cdot \mathbf{x_j} + \mathbf{W^{hf}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{o} =& \sigma(\mathbf{W^{xo}} \cdot \mathbf{x_j} + \mathbf{W^{ho}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{g} =& \tanh(\mathbf{W^{xg}} \cdot \mathbf{x_j} + \mathbf{W^{hg}} \cdot \mathbf{h_{j-1}})
\end{aligned}
$$

# LSTM

## (Long short-term Memory)

- The LSTM is formulation:

$$R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) = [\mathbf{c_j}; \mathbf{h_j}]$$

$$\mathbf{c_j} = \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i}$$

$\mathbf{h_j} = \tanh(\mathbf{c_j})$ ~~$\odot \mathbf{o}$~~

$$\mathbf{i} = \sigma(\mathbf{W^{xi}} \cdot \mathbf{x_j} + \mathbf{W^{hi}} \cdot \mathbf{h_{j-1}})$$

$$\mathbf{f} = \sigma(\mathbf{W^{xf}} \cdot \mathbf{x_j} + \mathbf{W^{hf}} \cdot \mathbf{h_{j-1}})$$

~~$\mathbf{o} = \sigma(\mathbf{W^{xo}} \cdot \mathbf{x_j} + \mathbf{W^{ho}} \cdot \mathbf{h_{j-1}})$~~

$$\mathbf{g} = \tanh(\mathbf{W^{xg}} \cdot \mathbf{x_j} + \mathbf{W^{hg}} \cdot \mathbf{h_{j-1}})$$

# LSTM
## (Long short-term Memory)

- The LSTM is formulation:

$$
\begin{aligned}
R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) &= [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} &= \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} &= \tanh(\mathbf{c_j}) \\
\mathbf{i} &= \sigma(\mathbf{W^{xi}} \cdot \mathbf{x_j} + \mathbf{W^{hi}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{f} &= \sigma(\mathbf{W^{xf}} \cdot \mathbf{x_j} + \mathbf{W^{hf}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{g} &= \tanh(\mathbf{W^{xg}} \cdot \mathbf{x_j} + \mathbf{W^{hg}} \cdot \mathbf{h_{j-1}})
\end{aligned}
$$

# Recurrent Additive Networks

- The LSTM is formulation:

$$
\begin{aligned}
R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) &= [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} &= \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} &= \tanh(\mathbf{c_j}) \\
\mathbf{i} &= \sigma(\mathbf{W^{xi}} \cdot \mathbf{x_j} + \mathbf{W^{hi}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{f} &= \sigma(\mathbf{W^{xf}} \cdot \mathbf{x_j} + \mathbf{W^{hf}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{g} &= \mathbf{W^{xg}} \cdot \mathbf{x_j}
\end{aligned}
$$

TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS

# LSTM: A Search Space Odyssey

Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, Jürgen Schmidhuber

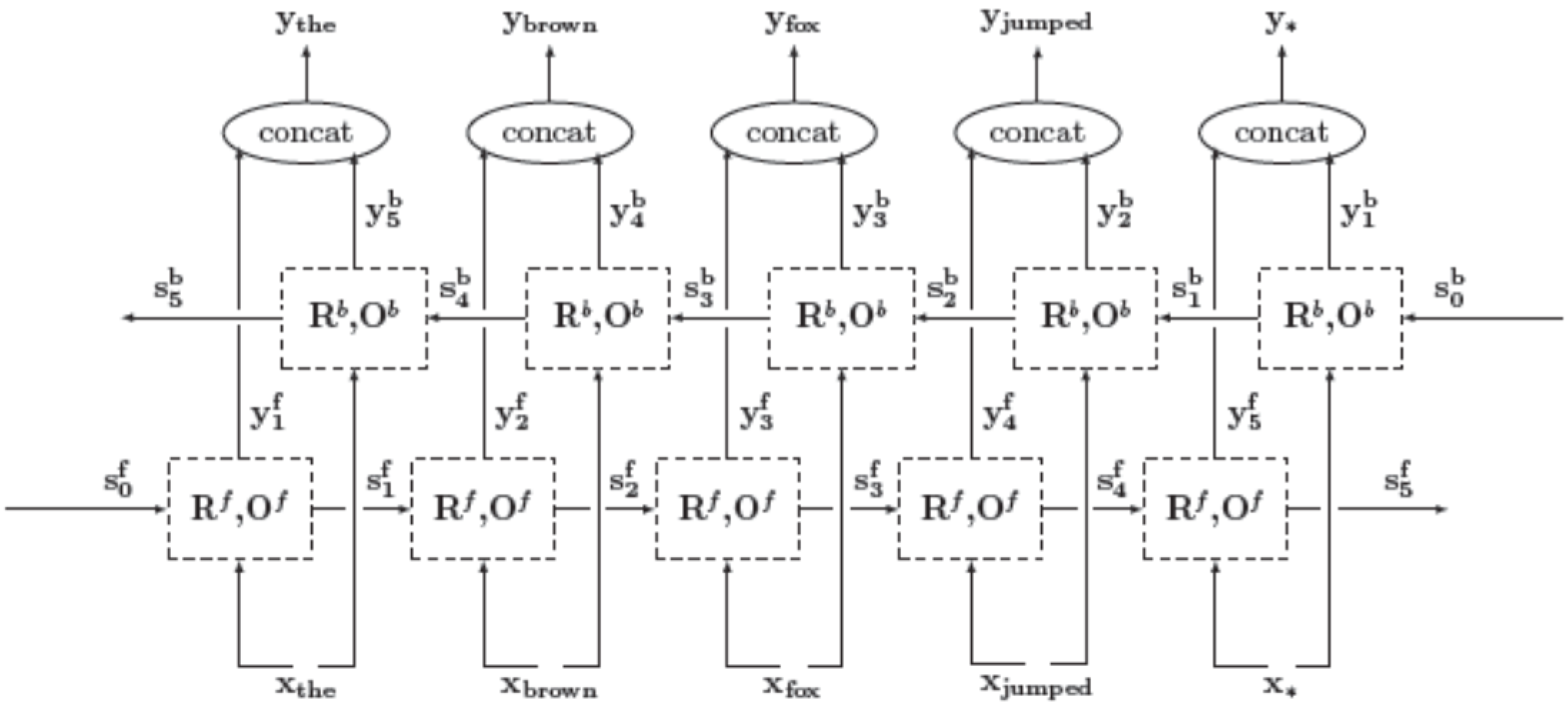
- Systematic search over LSTM choices
- Find that (1) forget gate is most important
- (2) non-linearity in output important since cell state can be unbounded
- GRU effective since it doesn't let cell state be unbounded

# LSTM
## (Long short-term Memory)

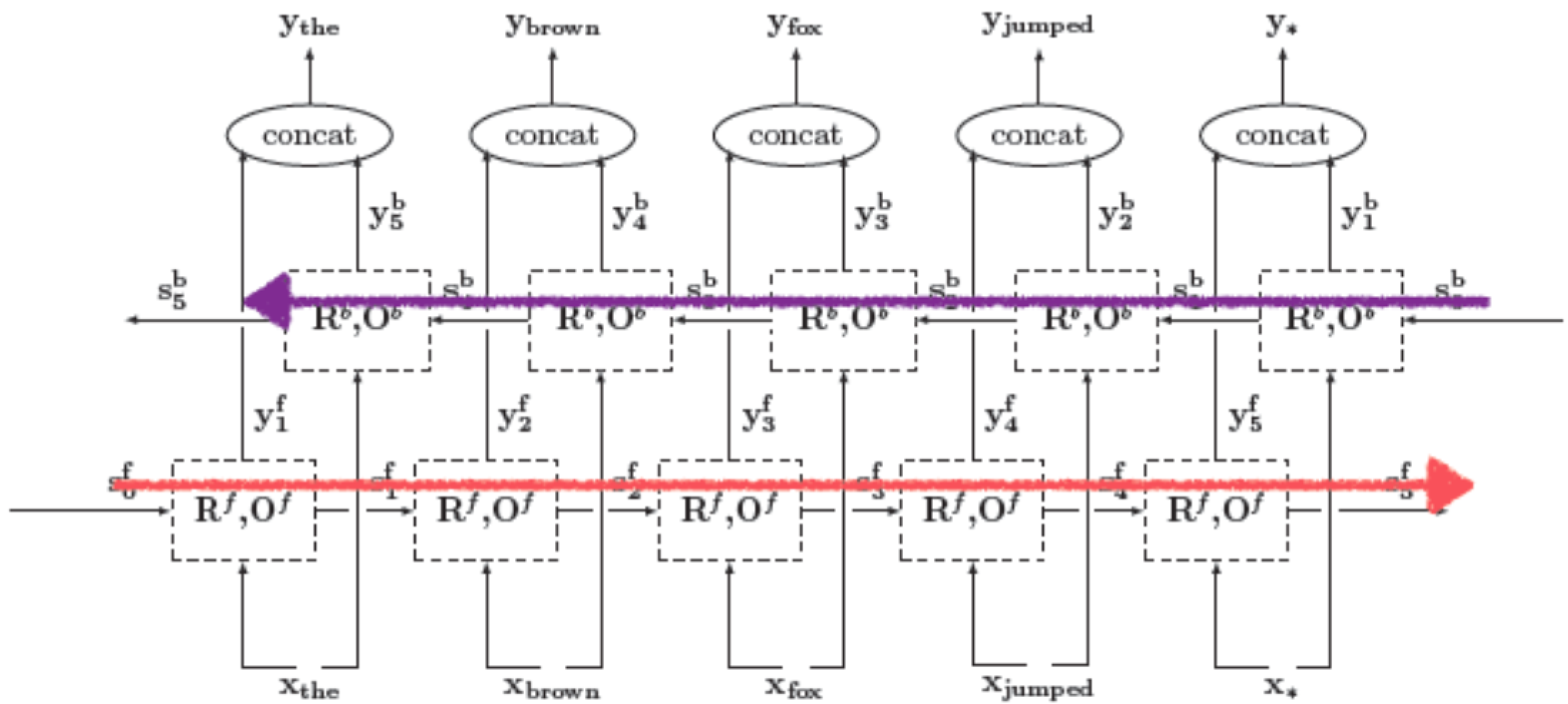
- The LSTM is formulation:

$$h_t = \mathcal{H}(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$
$$y_t = W_{hy}h_t + b_0$$

# Bidirectional LSTMs




One RNN runs left to right.
Another runs right to left.
Encode **both future and history** of a word.

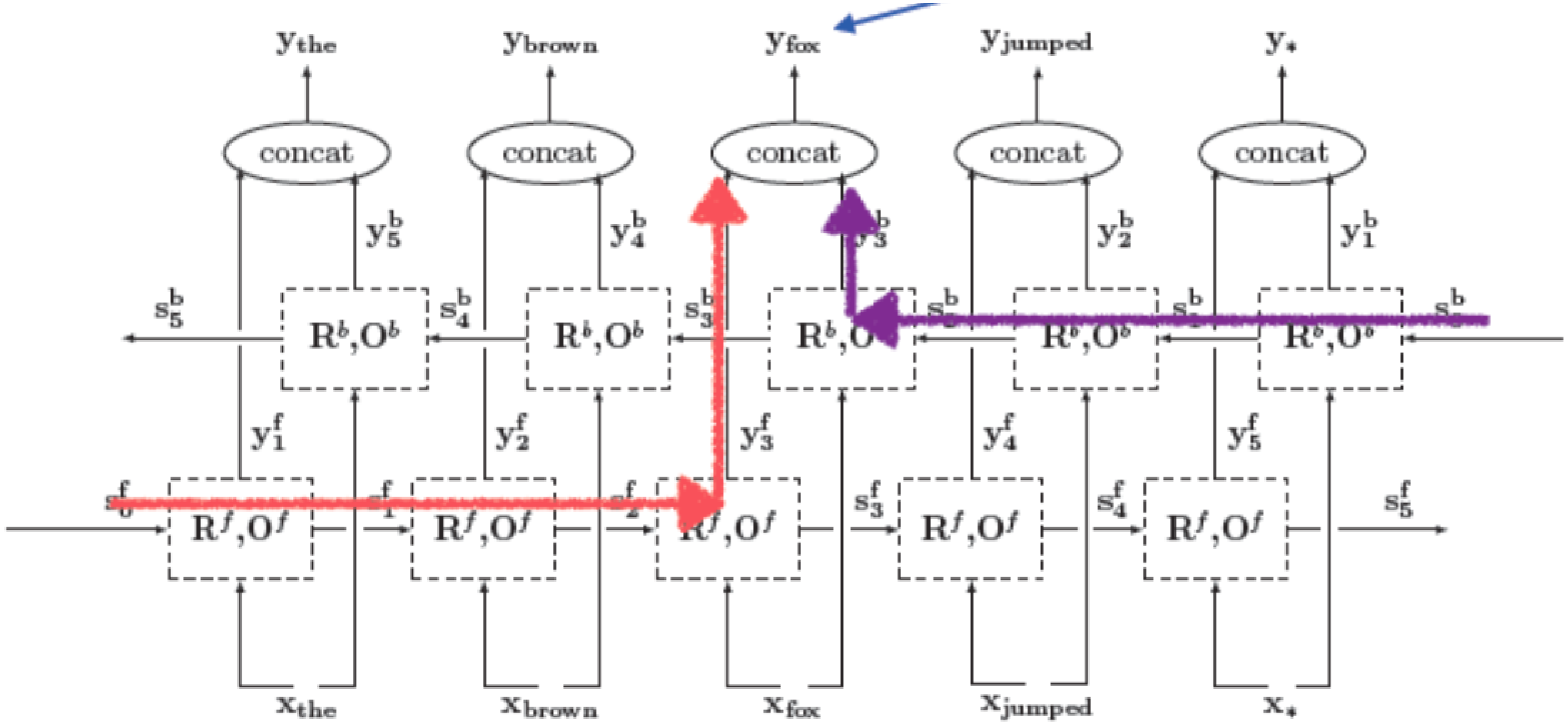



One RNN runs left to right.

Another runs right to left.

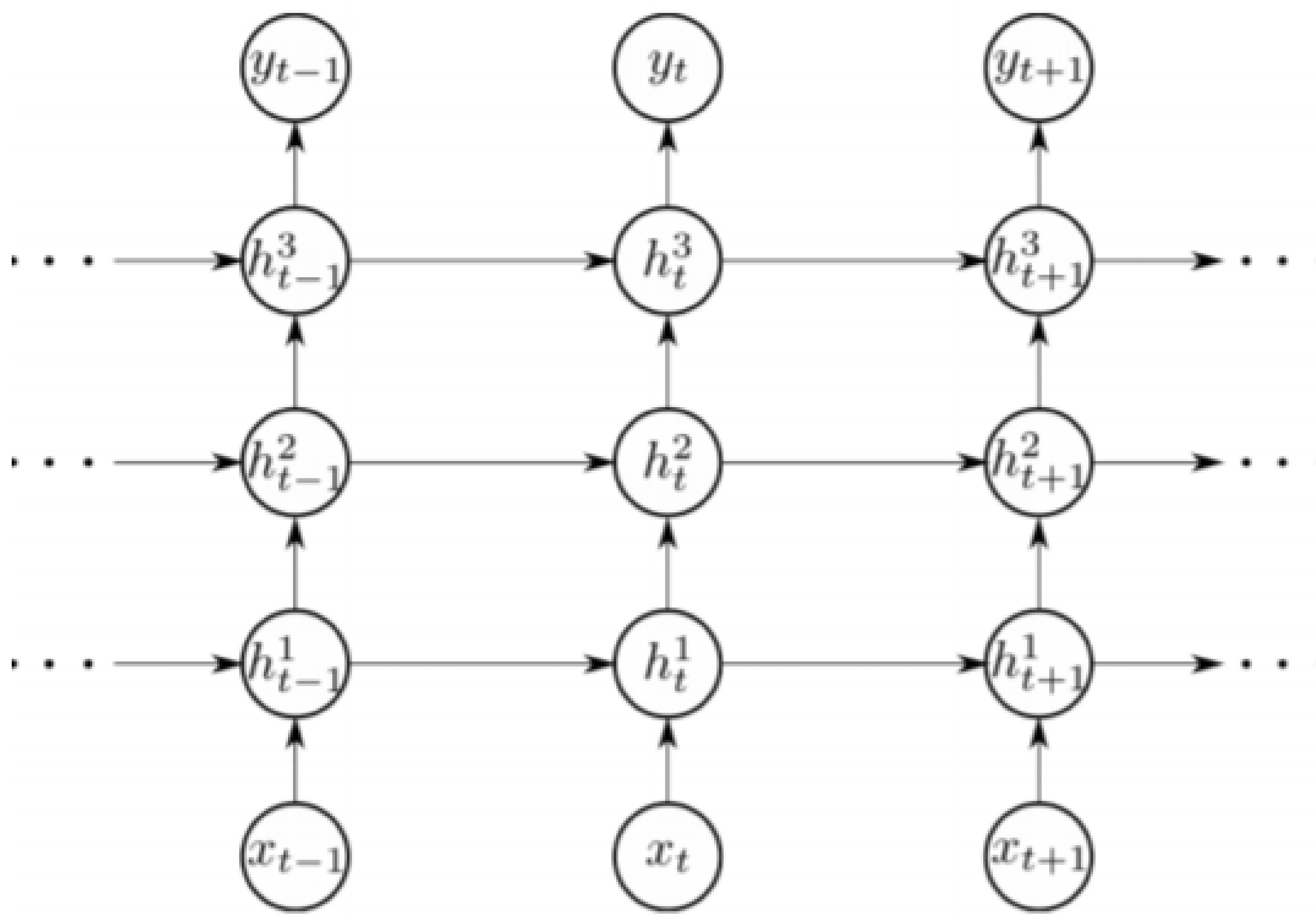
Encode **both future and history** of a word.

Infinite window around the word



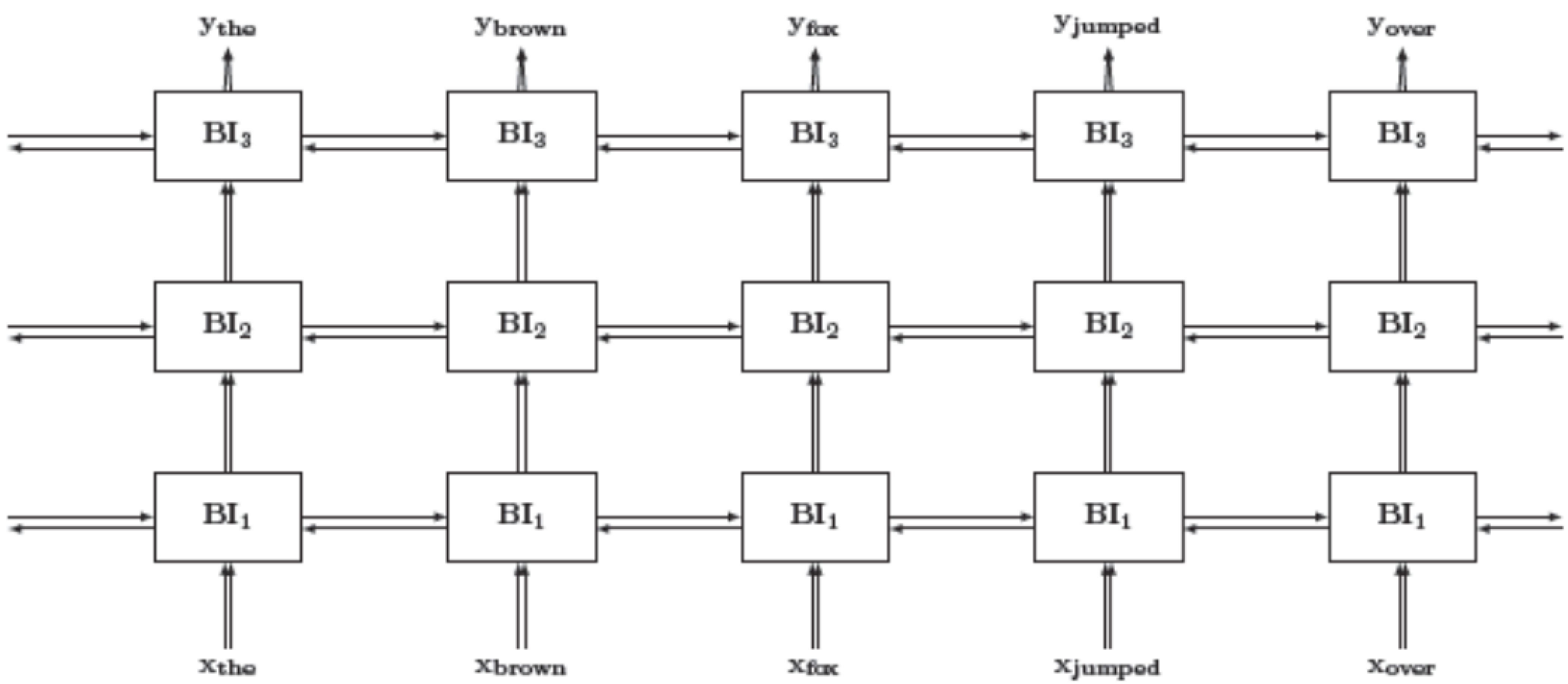

One RNN runs left to right.
Another runs right to left.
Encode **both future and history** of a word.

# Deep LSTMs




(a) Conventional stacked RNN

# Deep Bi-LSTMs



# Read More

- The gated architecture also helps the vanishing gradients problems.
- For a good explanation, see Kyunghyun Cho's notes: http://arxiv.org/abs/1511.07916 sections 4.2, 4.3
- Chris Olah's blog post