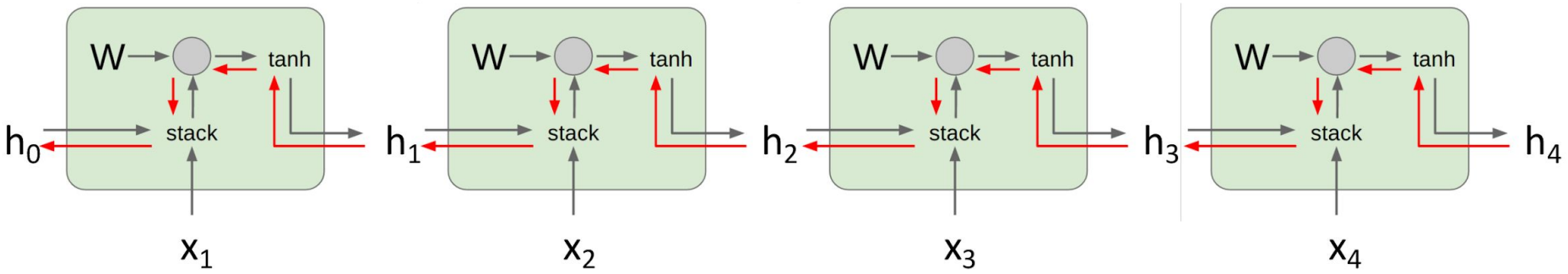
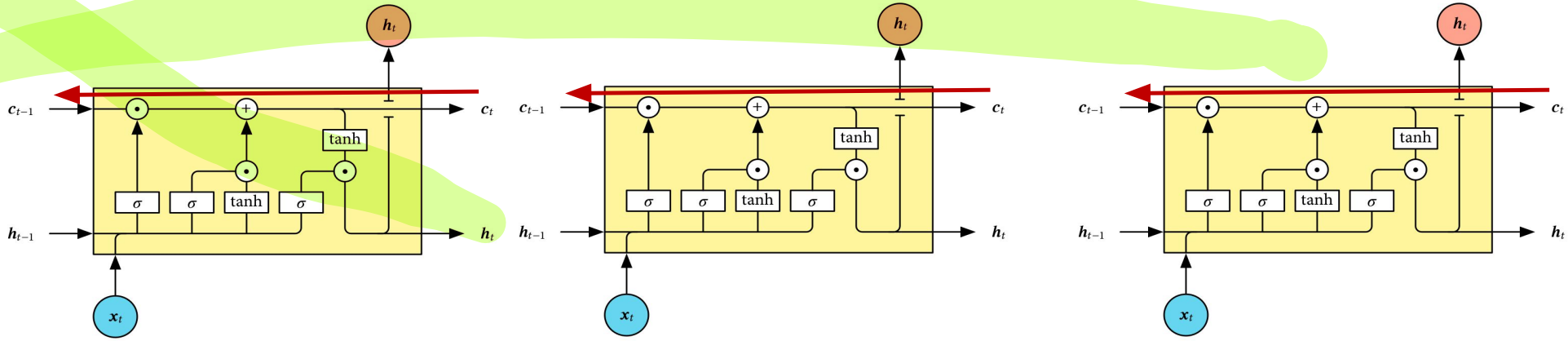
Deep Learning for Natural Language Processing (NLP)

Exploding/vanishing gradients issues in RNN



Long Short Term Memory (LSTM): Gradient Flow



reset and update gate.

Gated Recurrent Units (GRU) → cell state ni kaste.

LLMs

Main solution for better RNNs: Units

- Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks, introduced in 2014 by Kyunghyun Cho et al.

Main solution for better RNNs: Units

→ Lacks an output Gate.

- Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks, introduced in 2014 by Kyunghyun Cho et al.
- GRU is like a long short-term memory (LSTM) but has fewer parameters than LSTM, as it lacks an output gate.

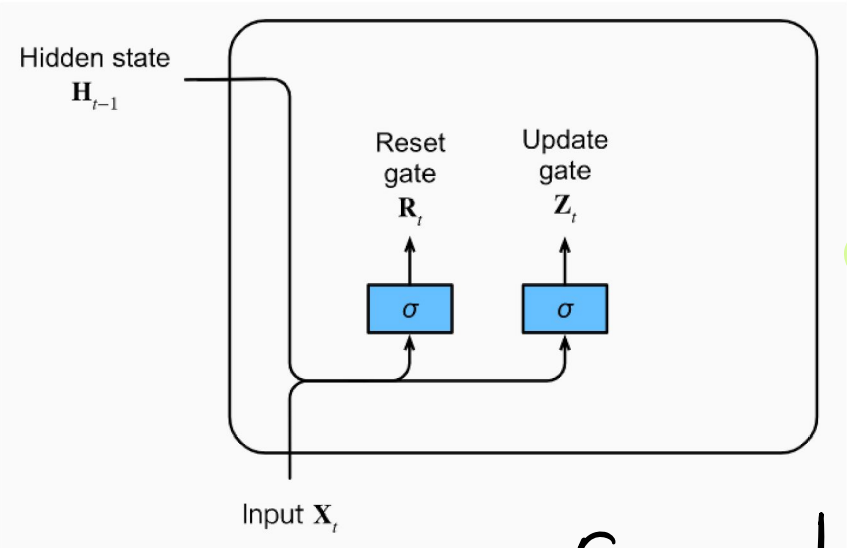
Main solution for better RNNs: Units

- Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks, introduced in 2014 by Kyunghyun Cho et al.
- GRU is like a long short-term memory (LSTM) but has fewer parameters than LSTM, as it lacks an output gate.
- GRU's performance on certain tasks of polyphonic music modeling, speech signal modeling and natural language processing was found to be similar to that of LSTM.

Main solution for better RNNs: Units

- In GRU, the LSTM's three gates are replaced by two
- **Reset gate** controls how much of the previous state we might still want to remember.
- **Update gate** would allow us to control how much of the new state is just a copy of the old state.

Gated Recurrent Units (GRU)



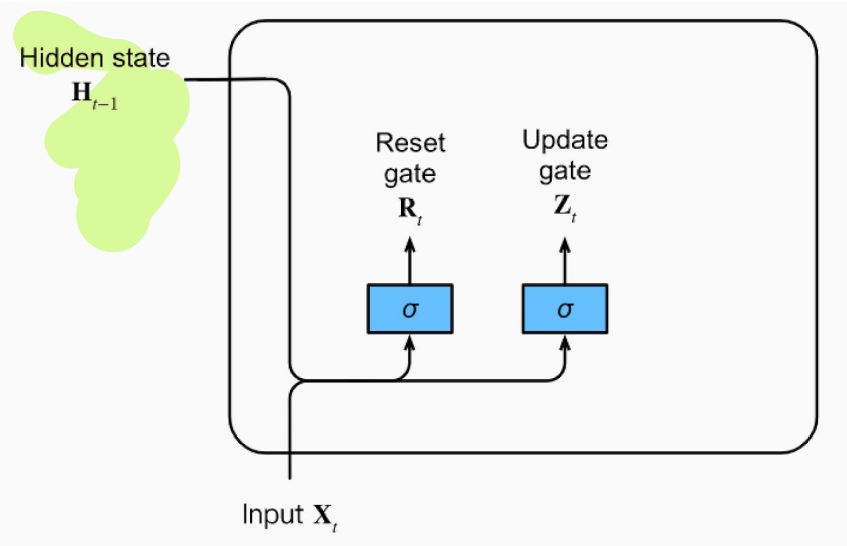
$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1})$$

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1})$$

$$\tilde{h} = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}))$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

Gated Recurrent Units (GRU)



$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1})$$

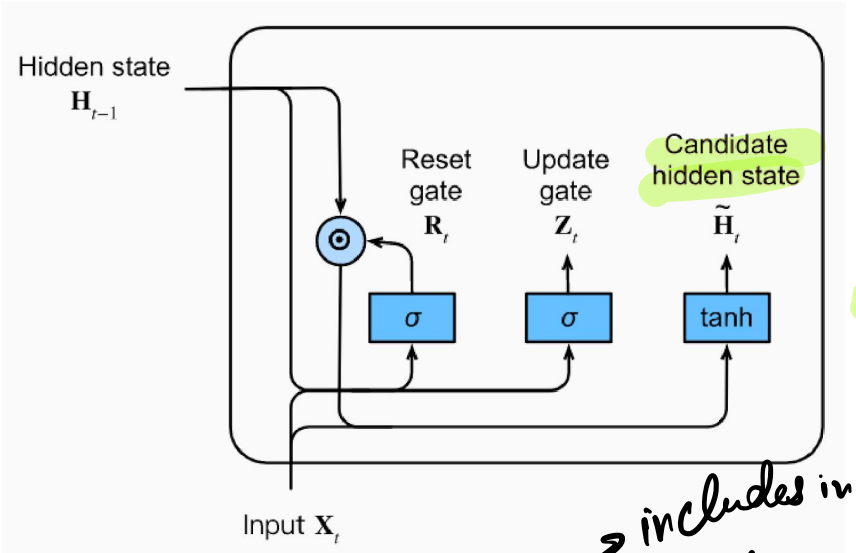
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1})$$

- Units with short-term dependencies will have active reset gates.
- Units with long term dependencies have active update gates.

Gated Recurrent Units (GRU)

LSTM

$$\begin{pmatrix} i_t \\ o_t \\ f_t \\ \tilde{c}_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b \right)$$



$$r_t = \sigma(W_{xr} x_t + W_{hr} h_{t-1})$$

$$z_t = \sigma(W_{xz} x_t + W_{hz} h_{t-1})$$

$$\hat{h}_t = \tanh\left(W_{xh} x_t + W_{hh}(r_t \odot h_{t-1})\right)$$

$$C_t = C_{t-1} \odot f_t + i_t \odot \tilde{C}_t$$

$$h_t = O_t \odot \tanh(C_t)$$

Memory Content

→ includes input and adamar product of reset and old history.

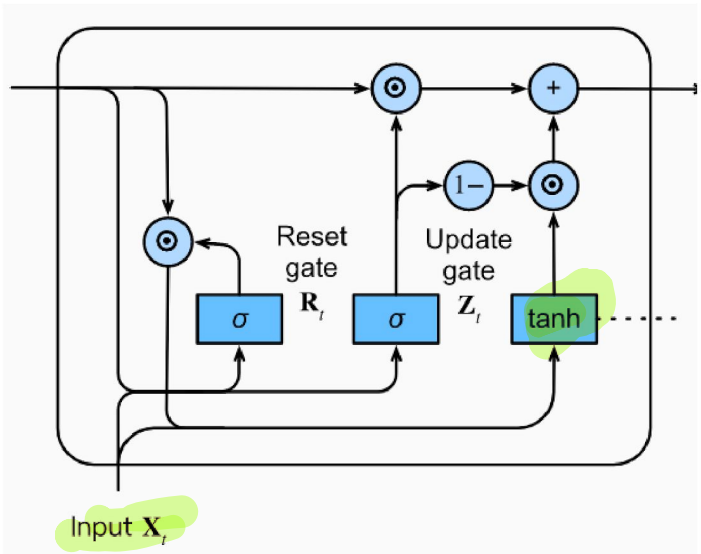
$$\hat{h}_t = \tanh\left(W_{xh} x_t + W_{hh}(r_t \odot h_{t-1})\right)$$

↑ Candidate hidden state.

$r_t = \sigma$

$z_t = \sigma$

Gated Recurrent Units (GRU)



$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1})$$

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1})$$

$$\hat{h_t} = \tanh\big(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1})\big)$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t)\hat{h_t}$$

Final Memory at current time step

r_t

u_t

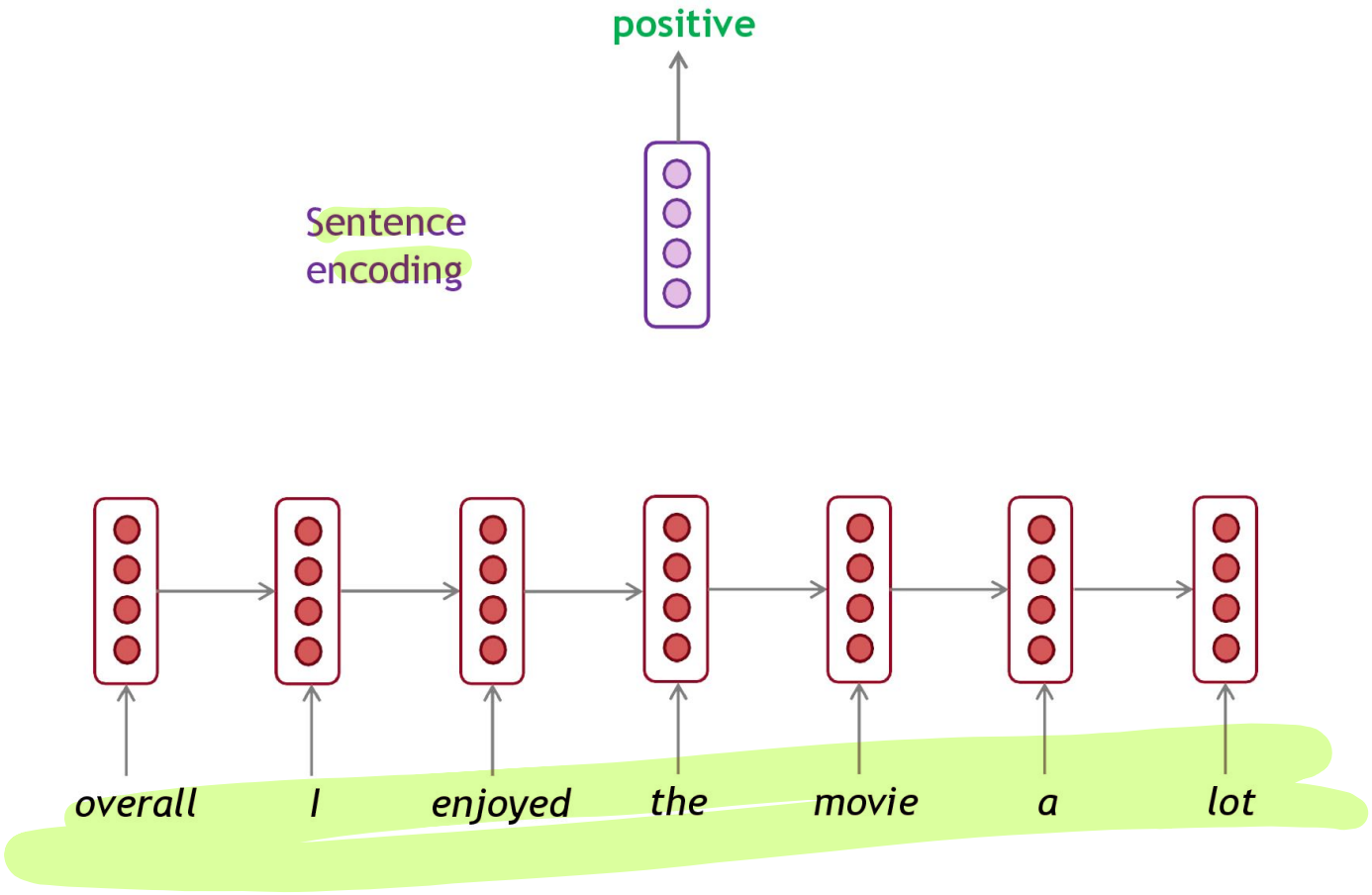
$\hat{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}))$

$h_t = z_t \odot h_{t-1} + (1 - z_t)\tilde{h}_t$

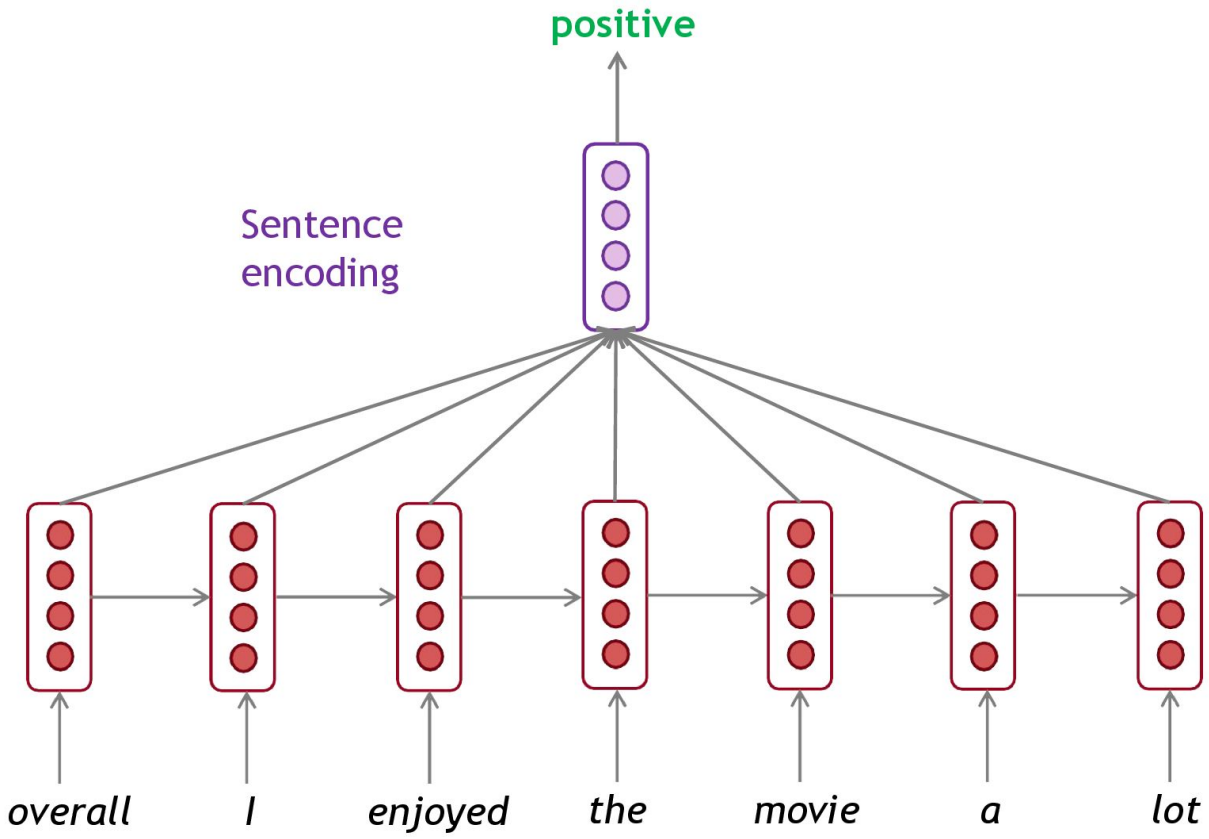
Bidirectional RNNs

Bidirectional and Multilayer RNN: Motivation

e.g. sentiment classification

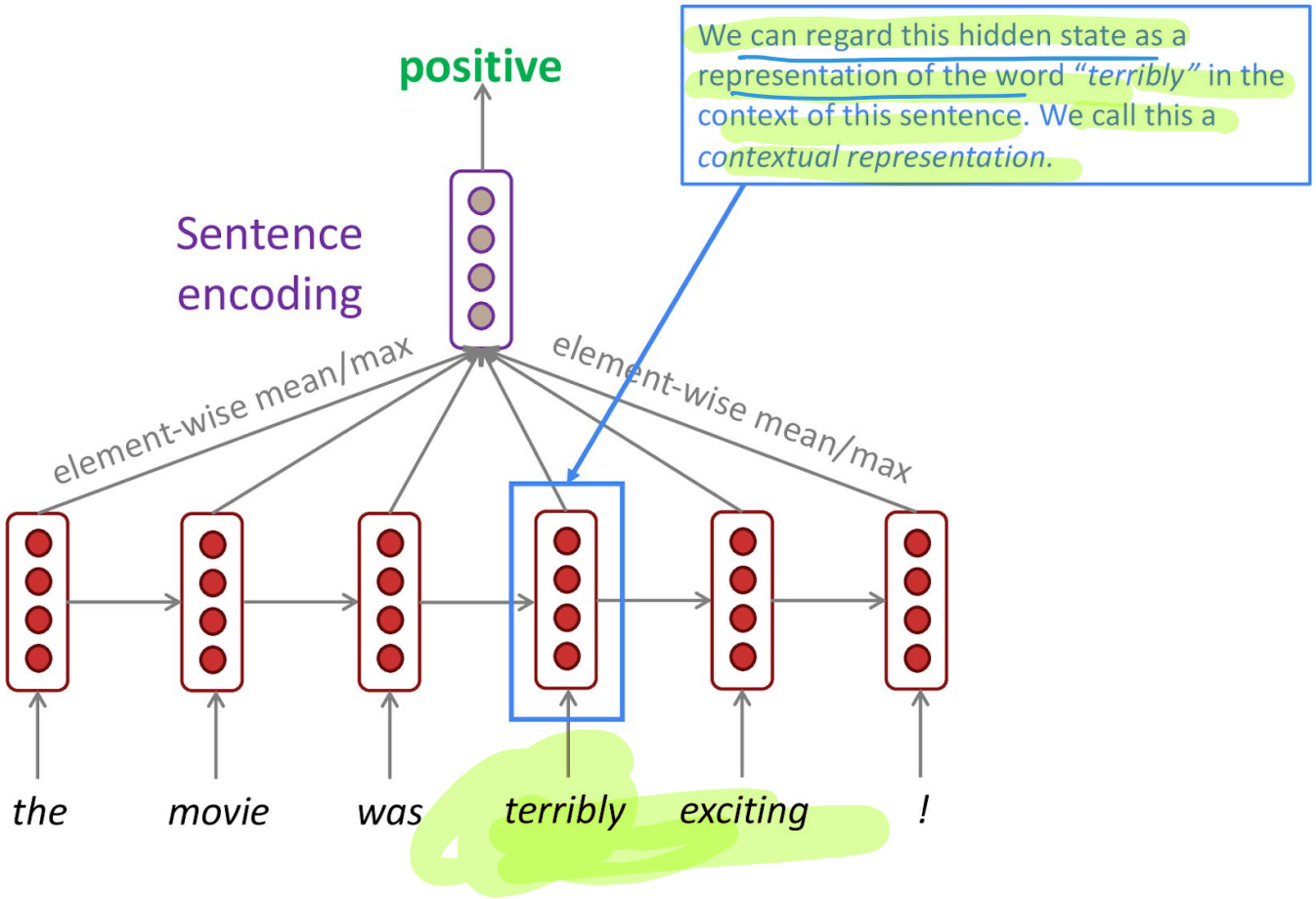


Bidirectional and Multilayer RNN: Motivation



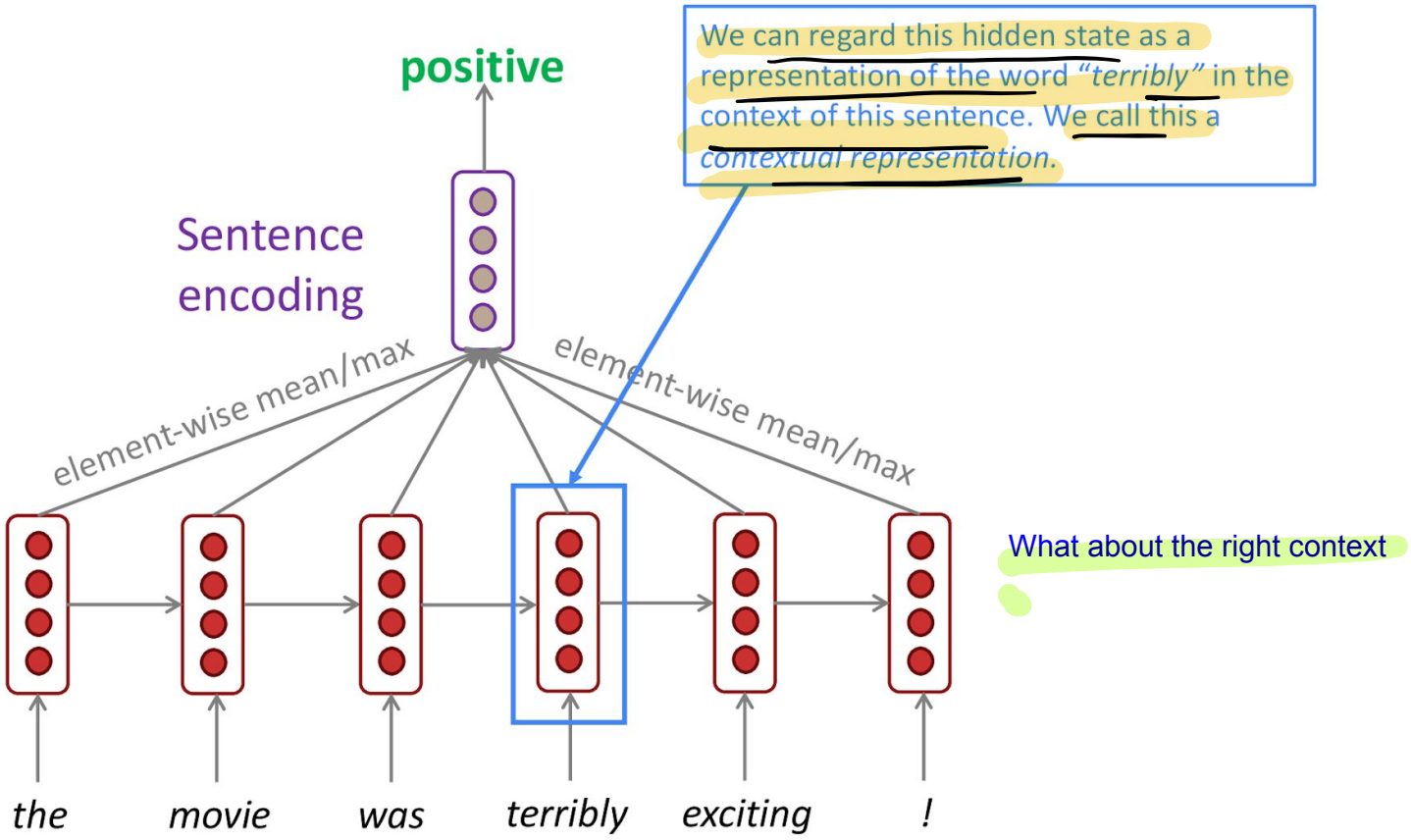
Bidirectional and Multilayer RNN: Motivation

Task: Sentiment Classification



Bidirectional and Multilayer RNN: Motivation

Task: Sentiment Classification



Bidirectional RNNs

→ specific for classification.

For classification you want to incorporate information from words both preceding and following.

Bidirectional RNNs

For classification you want to incorporate information from words both preceding and following.

Two type of connections:

1) One going forward in time, which helps us learn from previous representations

2) Another going backward in time, which helps us learn from future representations

$$\overrightarrow{h}_t = f_w(h_{t-1}, x_t)$$

Bidirectional RNNs

$\overleftarrow{h}_t = f_W(h_{t+1}, x_t)$

For classification you want to incorporate information from words both preceding and following.

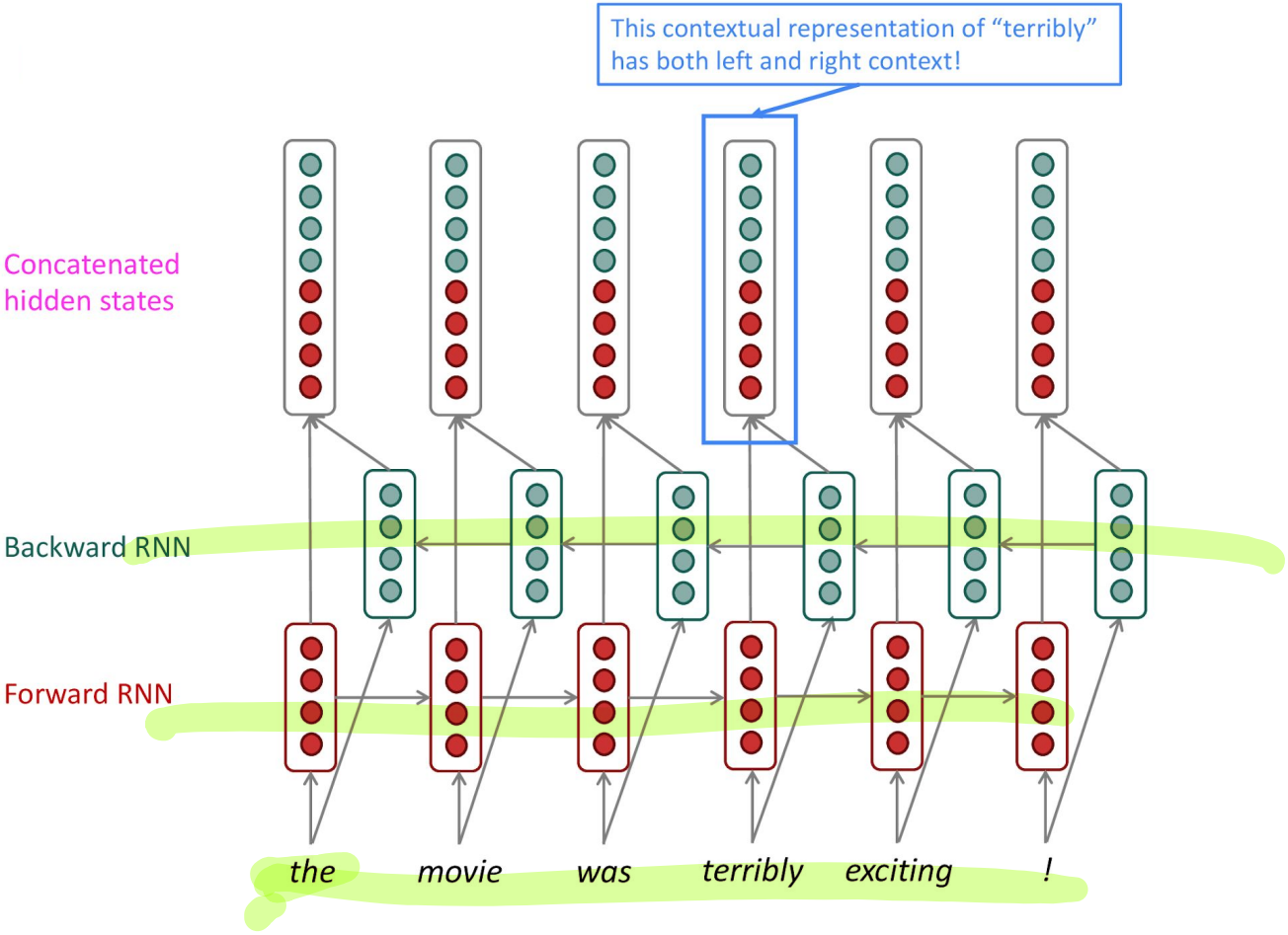
Two type of connections:

1) One going forward in time, which helps us learn from previous representations

2) Another going backward in time, which helps us learn from future representations

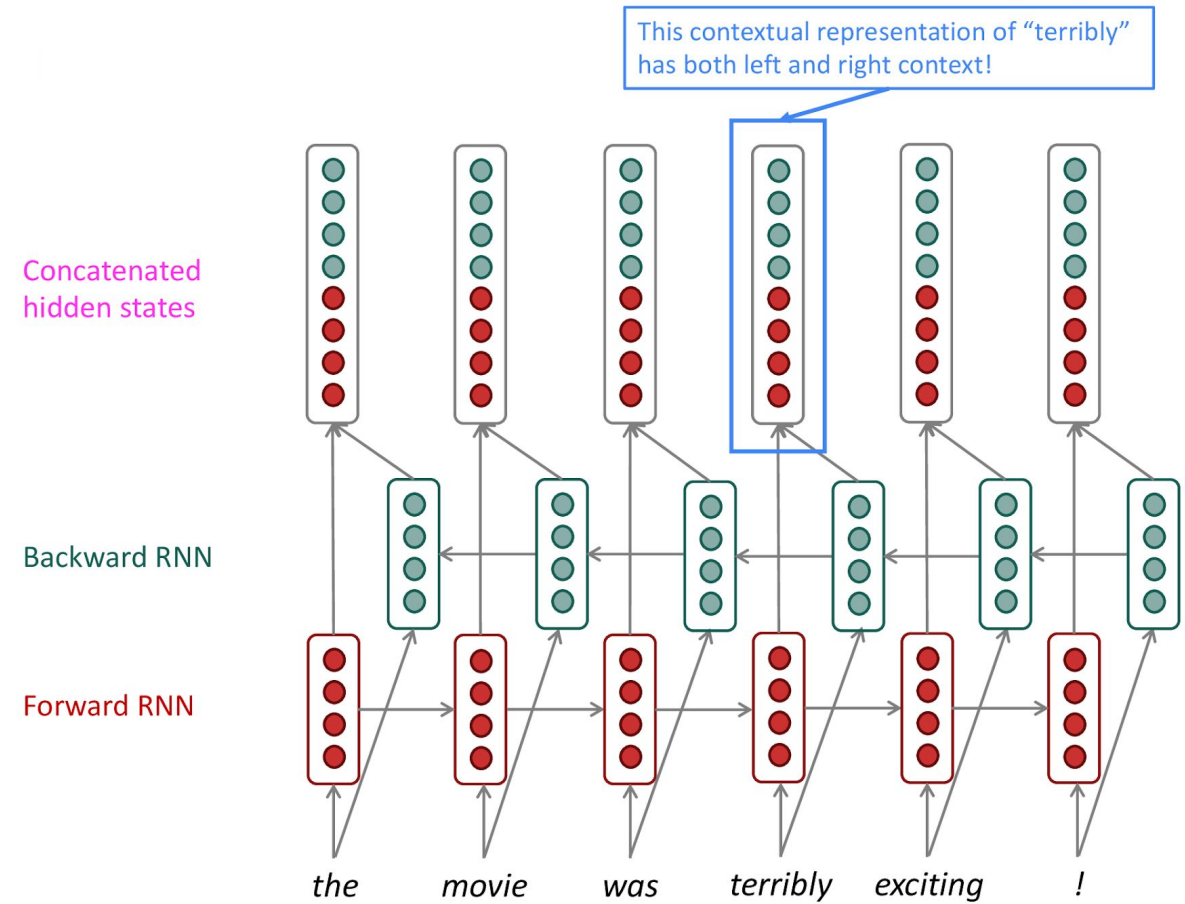
Bidirectional RNNs can better exploit context in both directions,

Bidirectional RNNs



Bidirectional RNNs

Data is processed in both directions with two separate hidden layers, which are then fed forward into the same output layer.



Bidirectional RNNs

On timestep *t*:

Forward RNN $\overrightarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{FW}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$

Backward RNN $\overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

Bidirectional RNNs

On timestep *t*:

Forward RNN $\overrightarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{FW}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$

Backward RNN $\overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

Generally, these two RNNs have separate weights

Bidirectional RNNs

On timestep *t*:

Forward RNN $\overrightarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{FW}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$

Backward RNN $\overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

Generally, these two RNNs have separate weights

Concatenated hidden states $\boldsymbol{h}^{(t)} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$

Bidirectional RNNs

On timestep *t*:

This is a general notation to mean "compute one forward step of the RNN" – it could be a simple RNN or LSTM computation.

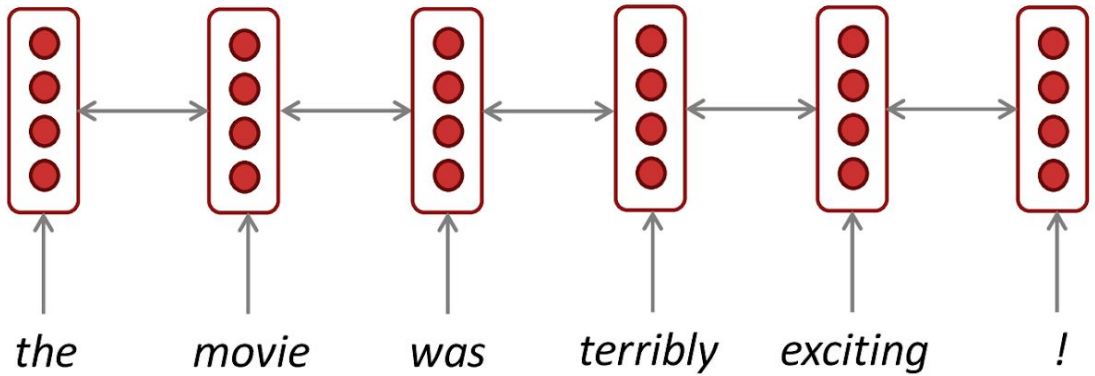
Forward RNN $\overrightarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{FW}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$

Backward RNN $\overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

Generally, these two RNNs have separate weights

Concatenated hidden states $\boldsymbol{h}^{(t)} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$

Bidirectional RNNs: simplified diagram



The two-way arrows indicate bidirectionality and the depicted hidden states are assumed to be the concatenated forwards+backwards states

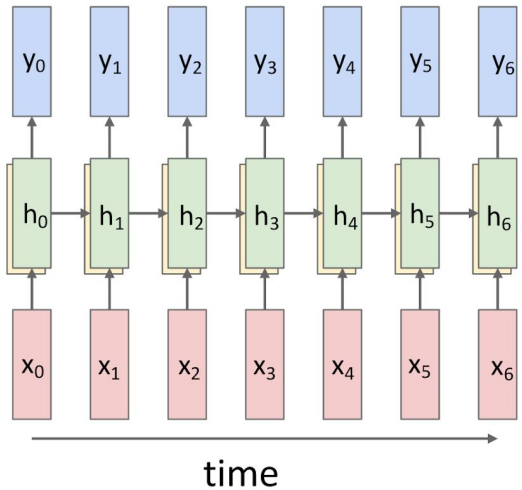
Bidirectional RNNs

- Bidirectional RNNs are only applicable if you have access to the entire input sequence.
  - Bidirectional RNNs are not applicable to Language Modeling, because in LM you only have left context available.
- Bidirectional LSTMs perform better than unidirectional ones in speech recognition.

Deep RNNs

Single-Layer RNNs

$$h_t = \tanh\left(W\begin{pmatrix}h_{t-1}\\ x_t\end{pmatrix} + b_h\right)$$



Single-Layer RNNs

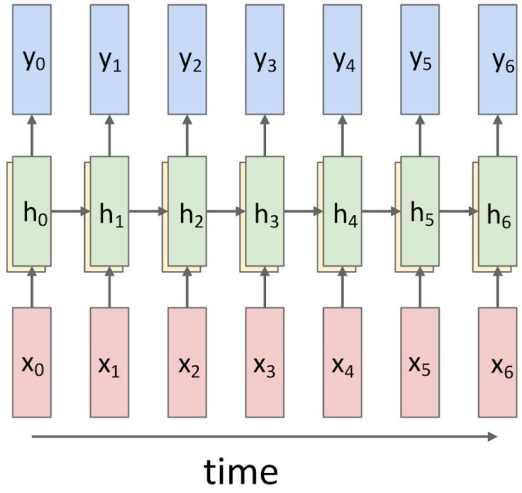
$$h_t = \tanh\left(W\begin{pmatrix}h_{t-1}\\ x_t\end{pmatrix} + b_h\right)$$

LSTM:

$$\begin{pmatrix}i_t\\ f_t\\ o_t\\ g_t\end{pmatrix} = \begin{pmatrix}\sigma\\ \sigma\\ \sigma\\ \tanh\end{pmatrix}\left(W\begin{pmatrix}h_{t-1}\\ x_t\end{pmatrix} + b_h\right)$$

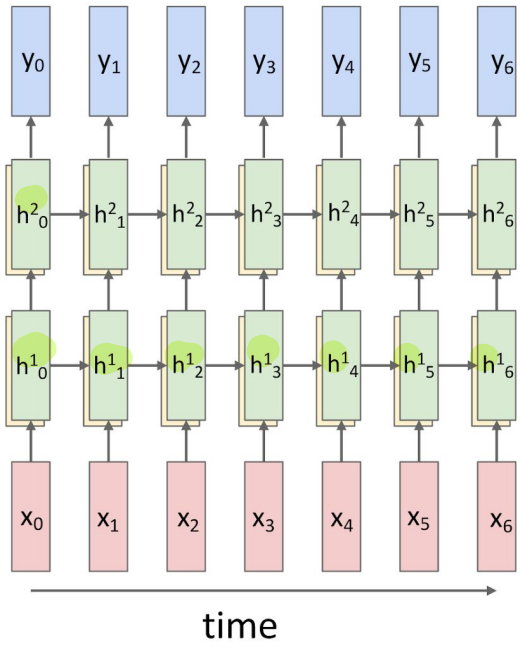
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot \tanh(c_t)$$



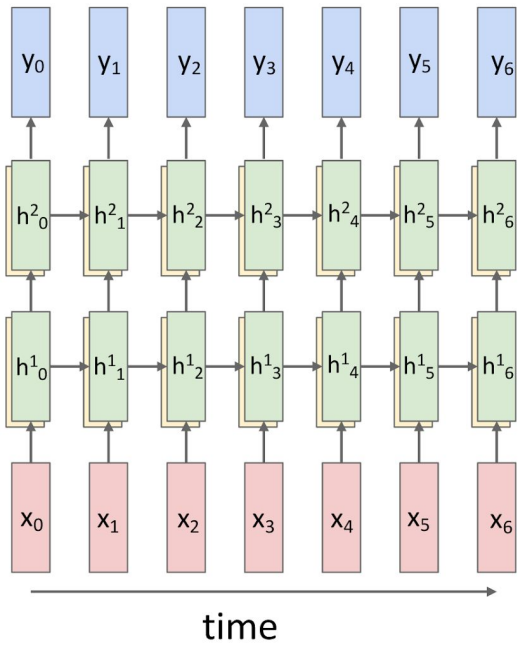
Multi-layer RNNs

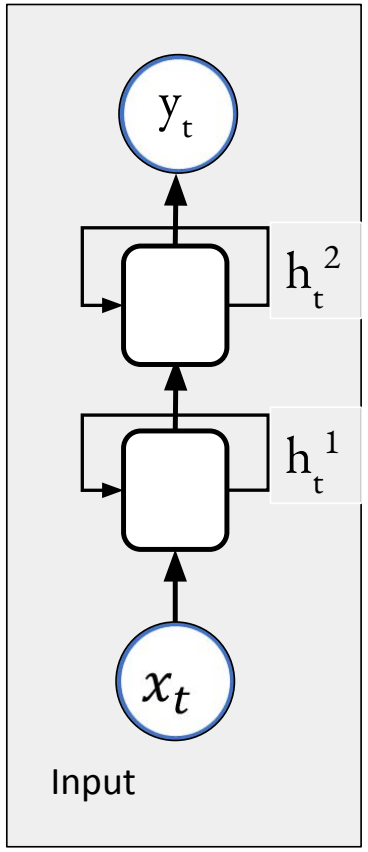
Two-layer RNN: Pass hidden states from one RNN as inputs to another RNN



Multi-layer RNNs

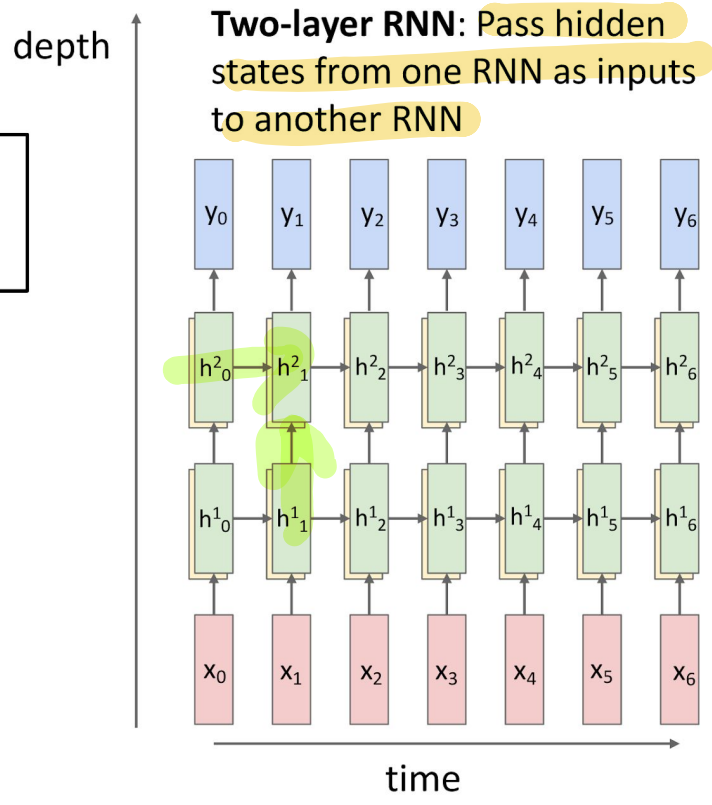
Two-layer RNN: Pass hidden states from one RNN as inputs to another RNN

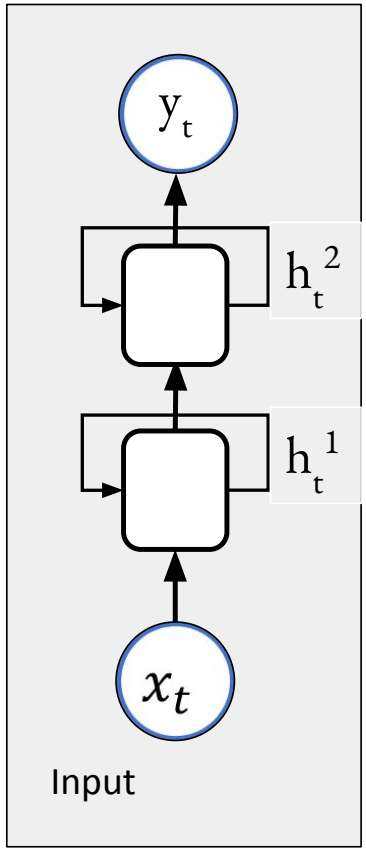




Multi-layer RNNs

$$h_t^{\ell} = \tanh\left(W \begin{pmatrix} h_{t-1}^{\ell} \\ h_t^{\ell-1} \end{pmatrix} + b_h^{\ell}\right)$$





Multi-layer RNNs

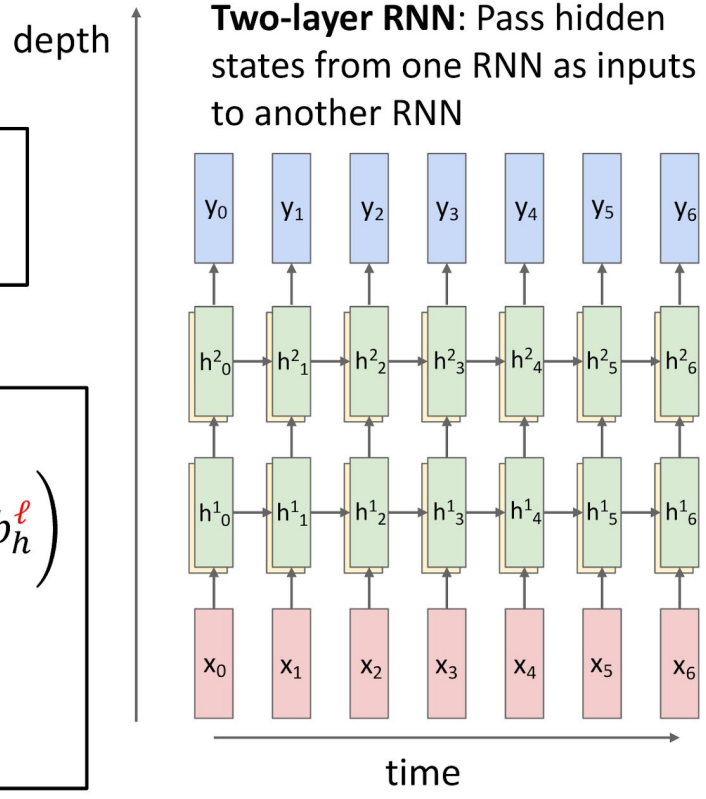
$$h_t^\ell = \tanh\left(W\begin{pmatrix}h_{t-1}^\ell \\ h_t^{\ell-1}\end{pmatrix} + b_h^\ell\right)$$

LSTM:

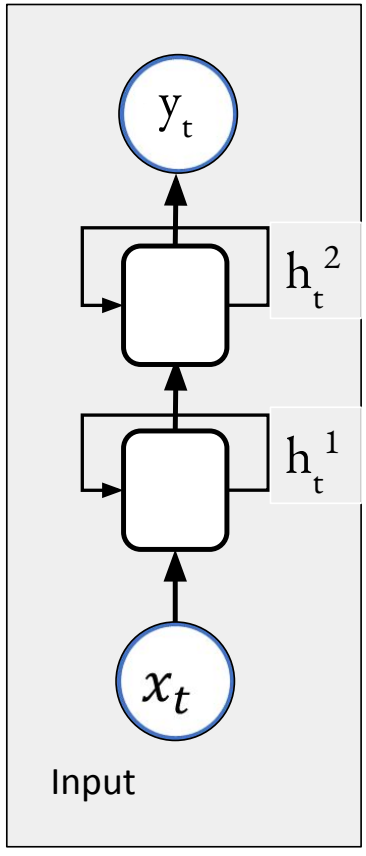
$$\begin{pmatrix}i_t^\ell \\ f_t^\ell \\ o_t^\ell \\ g_t^\ell\end{pmatrix} = \begin{pmatrix}\sigma \\ \sigma \\ \sigma \\ \tanh\end{pmatrix}\left(W\begin{pmatrix}h_{t-1}^\ell \\ h_t^{\ell-1}\end{pmatrix} + b_h^\ell\right)$$

$$c_t^\ell = f_t^\ell \odot c_{t-1}^\ell + i_t^\ell \odot g_t^\ell$$

$$h_t^\ell = o_t^\ell \odot \tanh(c_t^\ell)$$



Two-layer RNN: Pass hidden states from one RNN as inputs to another RNN



Multi-layer RNNs

Hidden state is given as

$$\mathbf{h}_t^{(l)} = \phi_l \left(\mathbf{h}_t^{(l-1)} \mathbf{W}_{\text{xh}}^{(l)} + \mathbf{h}_{t-1}^{(l)} \mathbf{W}_{\text{hh}}^{(l)} + \mathbf{b}_{\text{h}}^{(l)} \right)$$

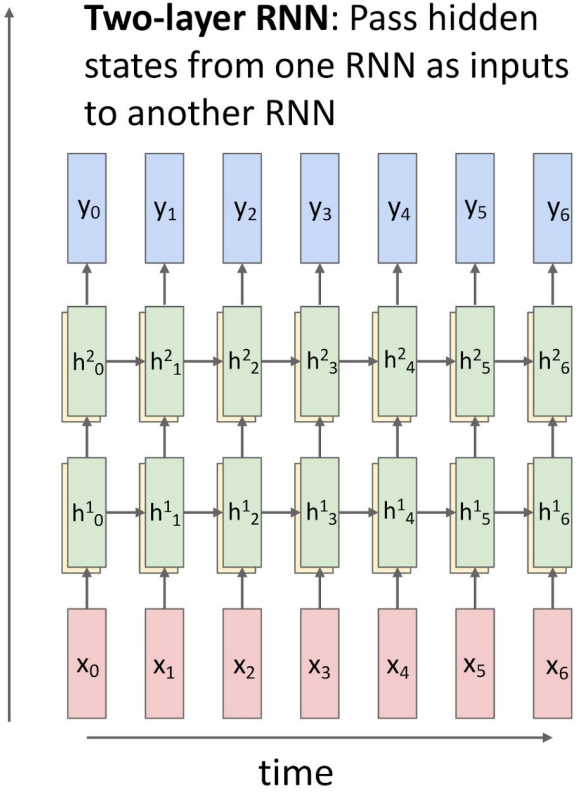
where the weights $\mathbf{W}_{\text{xh}}^{(l)} \in \mathbb{R}^{h \times h}$ and $\mathbf{W}_{\text{hh}}^{(l)} \in \mathbb{R}^{h \times h}$, together with the bias $\mathbf{b}_{\text{h}}^{(l)} \in \mathbb{R}^{1 \times h}$, are the model parameters of the l^{th} hidden layer.

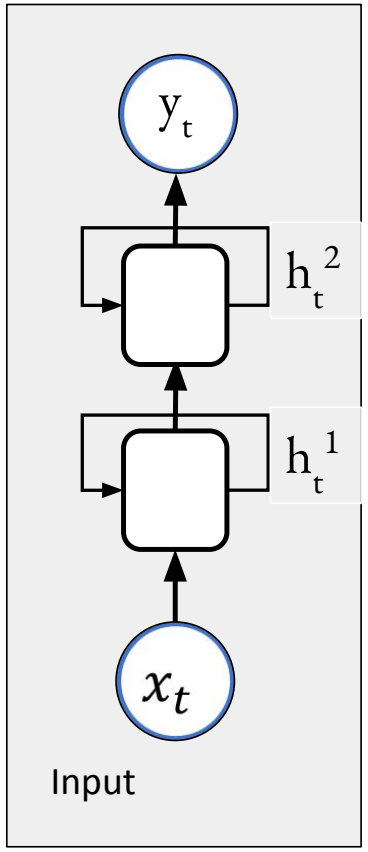
Output layer is given as

$$\mathbf{O}_t = \mathbf{h}_t^{(L)} \mathbf{W}_{\text{hq}} + \mathbf{b}_{\text{q}},$$

where the weight $\mathbf{W}_{\text{hq}} \in \mathbb{R}^{h \times q}$ and the bias $\mathbf{b}_{\text{q}} \in \mathbb{R}^{1 \times q}$ are the model parameters of the output layer.

Two-layer RNN: Pass hidden states from one RNN as inputs to another RNN





Multi-layer RNNs

- RNNs are already "deep" on one dimension (they unroll over many timesteps).

Multi-layer RNNs

- RNNs are already "deep" on one dimension (they unroll over many timesteps).
- We can also make them "deep" in another dimension by applying multiple RNNs — this is a multi-layer RNN.

Multi-layer RNNs

- RNNs are already "deep" on one dimension (they unroll over many timesteps).
- We can also make them "deep" in another dimension by applying multiple RNNs — this is a multi-layer RNN.
- This allows the network to compute more complex representations

Multi-layer RNNs

→ Let it learn low level features at lower level.

- RNNs are already "deep" on one dimension (they unroll over many timesteps).
- We can also make them "deep" in another dimension by applying multiple RNNs — this is a multi-layer RNN.
- This allows the network to compute more complex representations
- The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.

Multi-layer RNNs

- RNNs are already "deep" on one dimension (they unroll over many timesteps).
- We can also make them "deep" in another dimension by applying multiple RNNs — this is a multi-layer RNN.
- This allows the network to compute more complex representations
- The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.
- Multi-layer RNNs are also called stacked RNNs