Summer semester 2023

344.063 KV Special Topic:

Natural Language Processing with Deep Learning

Recurrent Neural Networks

Navid Rekab-saz

navid.rekabsaz@jku.at

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computational Perception

Agenda

- Recurrent Neural Networks
- Backpropagation Through Time
- RNNs with Gates: LSTM, GRU

Element-wise Multiplication

- $\boldsymbol{a} \odot \boldsymbol{b} = \boldsymbol{c}$
  - dimensions: $1 \times d \odot 1 \times d = 1 \times d$

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \odot \begin{bmatrix} 3 & 0 & -2 \end{bmatrix} = \begin{bmatrix} 3 & 0 & -6 \end{bmatrix}$$

- $\boldsymbol{A} \odot \boldsymbol{B} = \boldsymbol{C}$
  - dimensions: $l \times m \odot l \times m = l \times m$

$$\begin{bmatrix} 2 & 3 \\ 0 & 1 \\ 1 & -1 \end{bmatrix} \odot \begin{bmatrix} -1 & 0 \\ 0 & 2 \\ 0.5 & -1 \end{bmatrix} = \begin{bmatrix} -2 & 0 \\ 0 & 2 \\ 0.5 & 1 \end{bmatrix}$$

Agenda

- **Recurrent Neural Networks**
- Backpropagation Through Time
- RNNs with Gates: LSTM, GRU

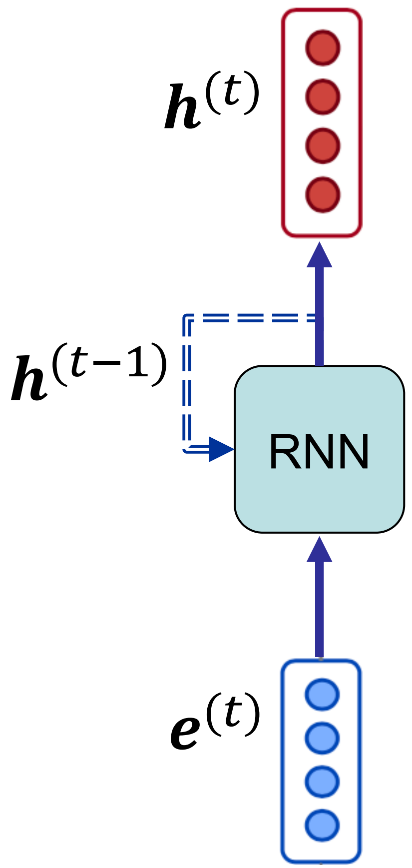
Recurrent Neural Network

- Recurrent Neural Network (RNN) encodes/embeds a sequential input of any size into compositional embeddings

- A sequence can be …
  - a stream of word/subword/character vectors
  - time series
  - etc.

- RNN models …
  - capture dependencies through the sequence
  - apply the same parameters repeatedly
  - output a final embedding but also intermediary embeddings on each time step

Recurrent Neural Networks



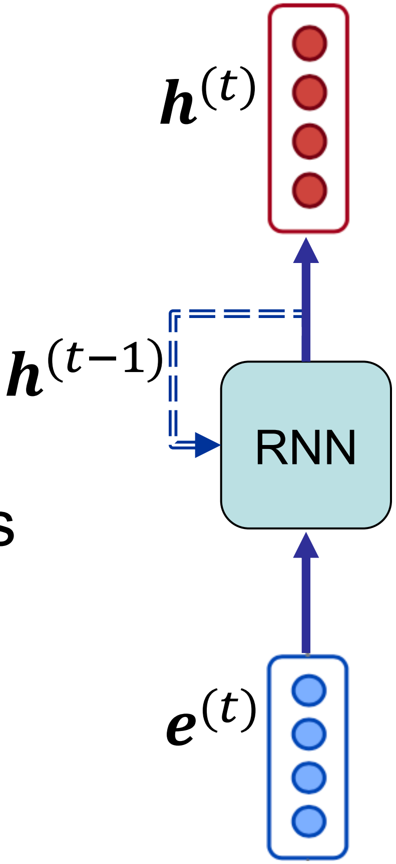
Recurrent Neural Networks

- Output $\boldsymbol{h}^{(t)}$ is a function of input $\boldsymbol{e}^{(t)}$ and the output of the previous time step $\boldsymbol{h}^{(t-1)}$

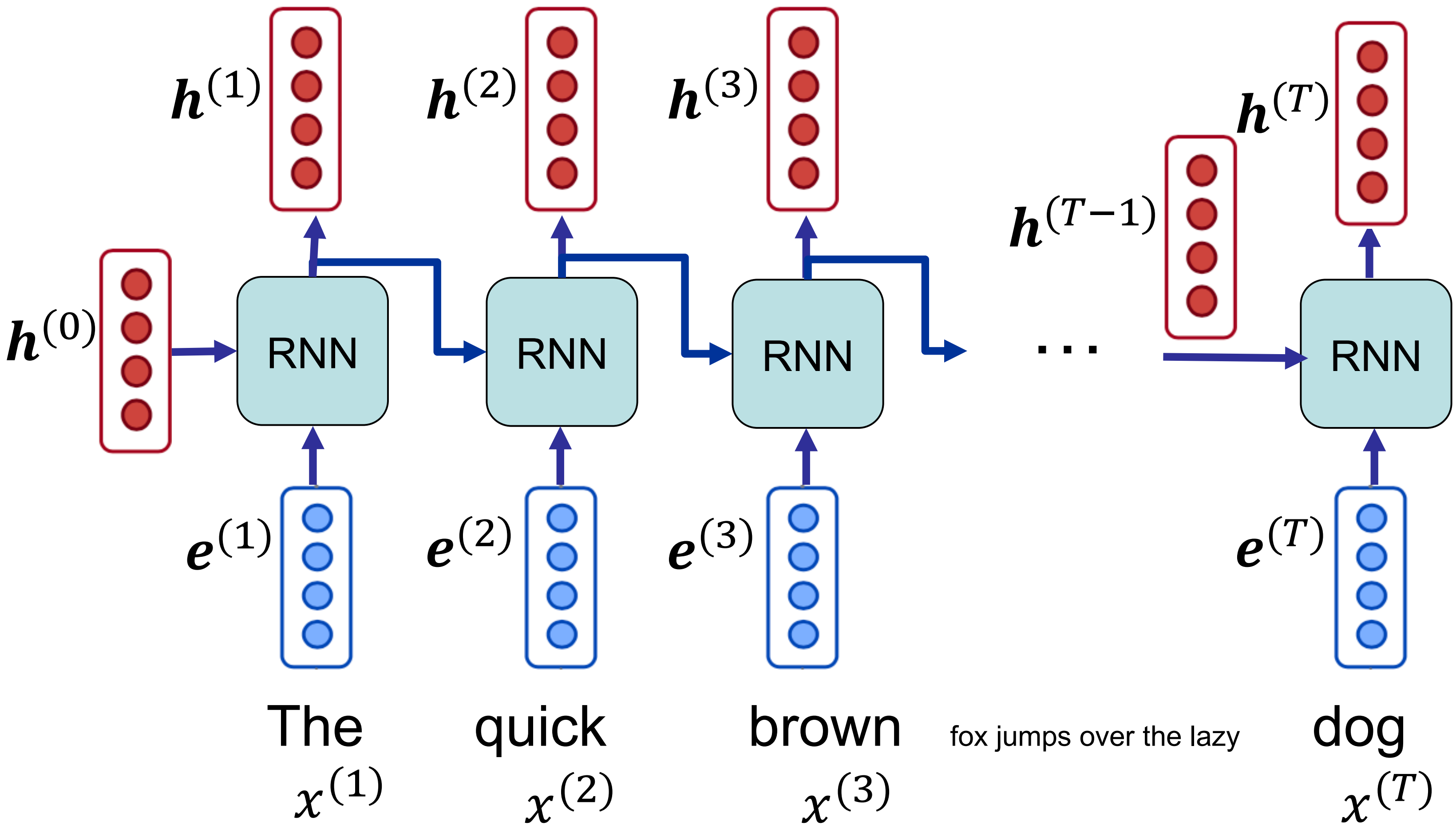
$$\boldsymbol{h}^{(t)} = \mathrm{RNN}(\boldsymbol{h}^{(t-1)}, \boldsymbol{e}^{(t)})$$

- $\boldsymbol{h}^{(t)}$ is called hidden state

- With hidden state $\boldsymbol{h}^{(t-1)}$, the model accesses to a sort of memory from all previous entities



RNN – Unrolling



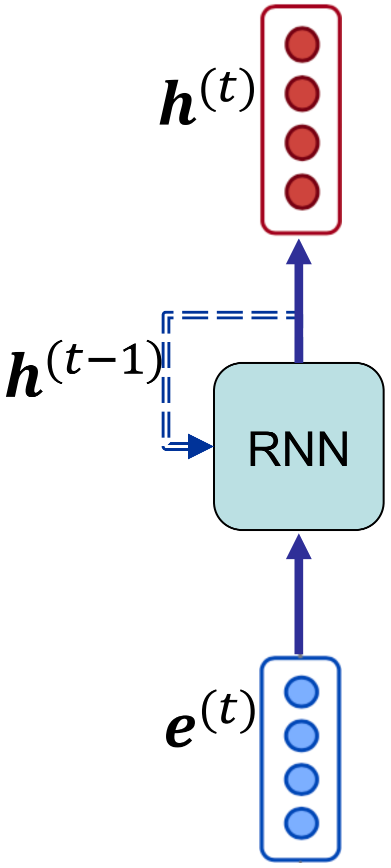
Vanilla (Elman) RNN

- General form of an RNN function

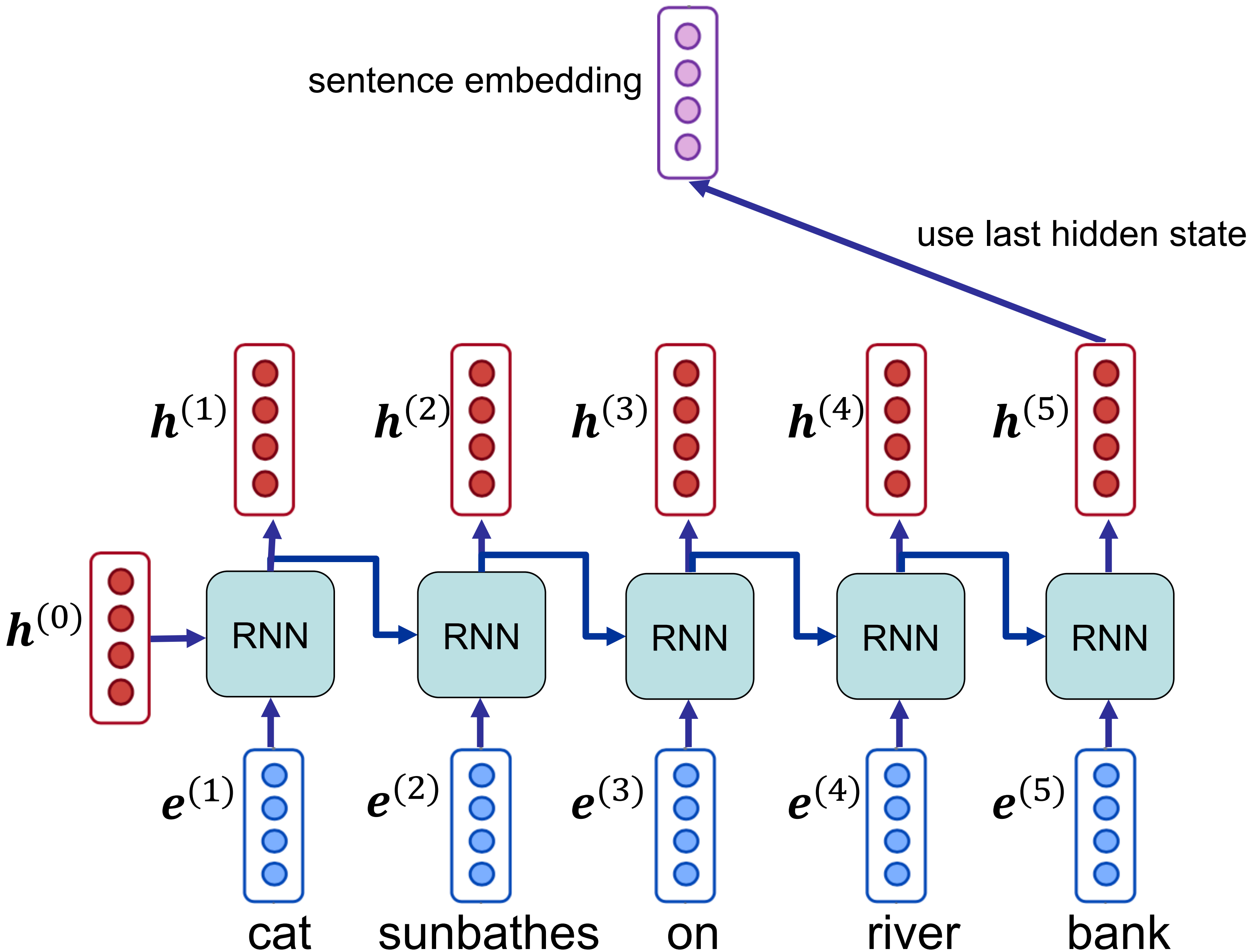
$$\boldsymbol{h}^{(t)} = \mathrm{RNN}(\boldsymbol{h}^{(t-1)}, \boldsymbol{e}^{(t)})$$

- Vanilla RNN:
  - linear projection of the previous hidden state $\boldsymbol{h}^{(t-1)}$
  - linear projection of input $\boldsymbol{e}^{(t)}$
  - summing the projections and applying a non-linearity

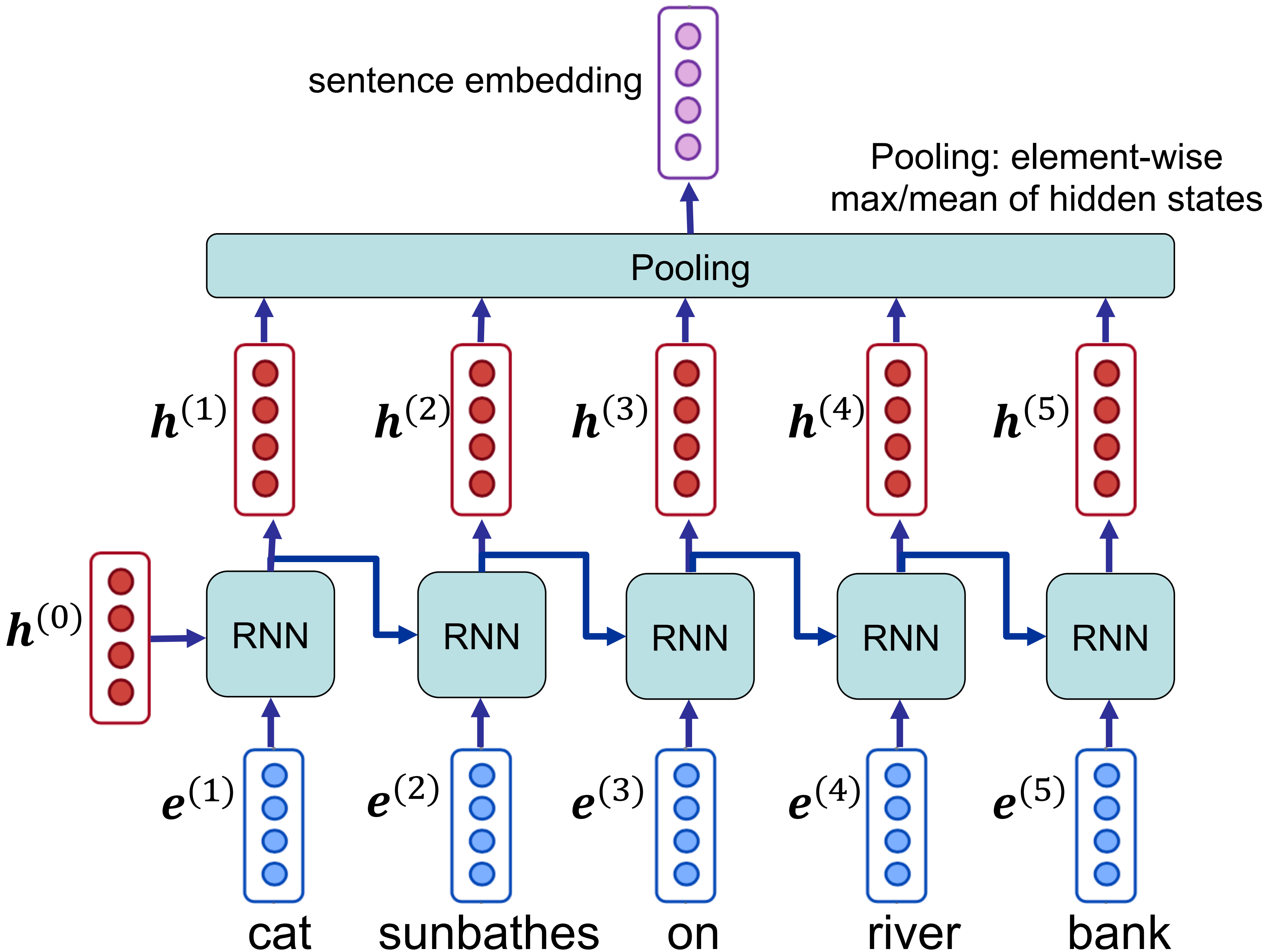
$$\boldsymbol{h}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_h + \boldsymbol{e}^{(t)}\boldsymbol{W}_e + \boldsymbol{b})$$



RNN – Compositional embedding



RNN – Compositional embedding



Bidirectional RNNs

- Bidirectional RNN consists of two RNNs, one reads from the beginning to the end of sequence (forward), and the other reads from the end to the beginning (backward)

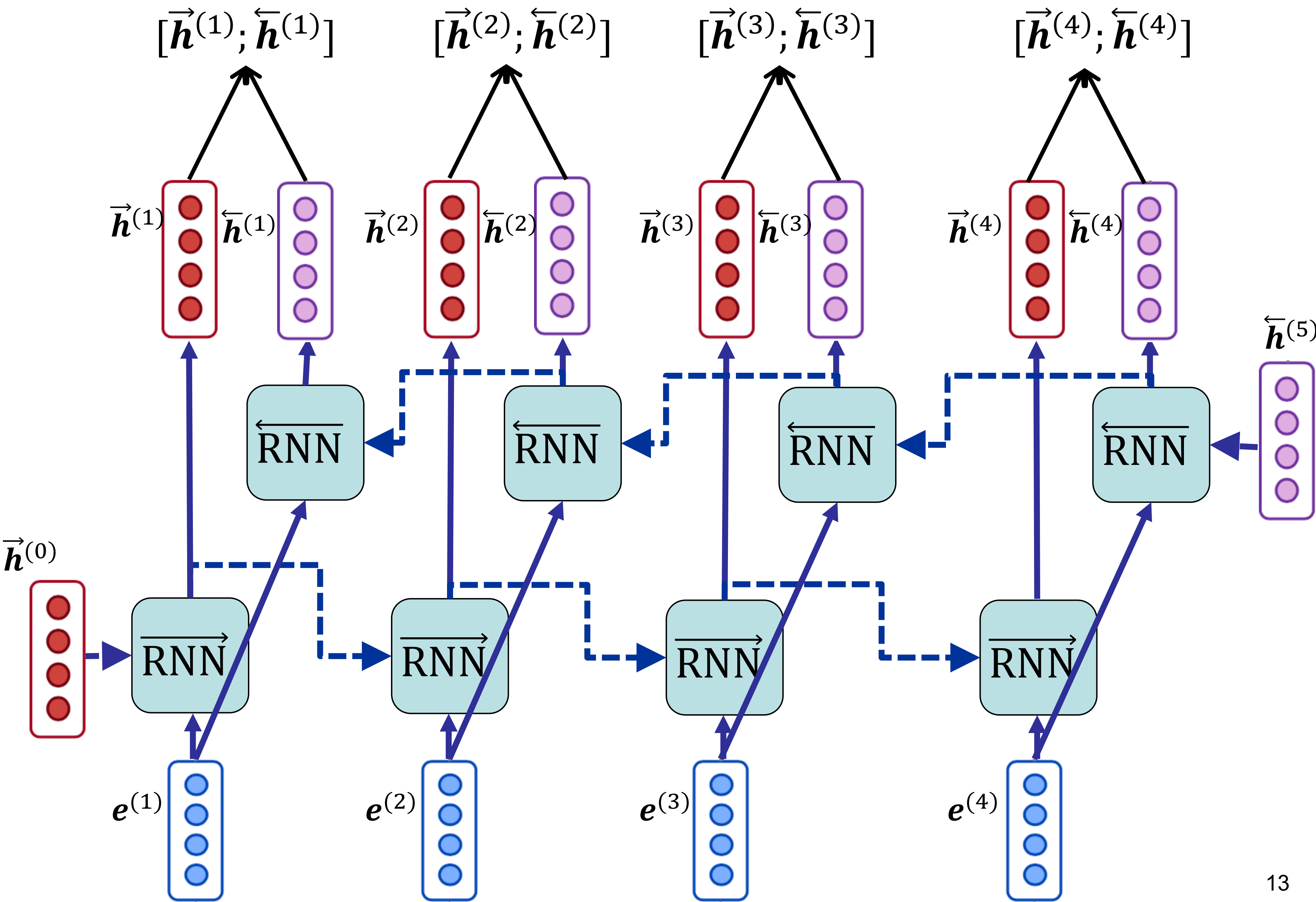
$$\overrightarrow{\boldsymbol{h}}^{(t)} = \overrightarrow{\mathrm{RNN}}\,(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{e}^{(t)})$$

$$\overleftarrow{\boldsymbol{h}}^{(t)} = \overleftarrow{\mathrm{RNN}}\,(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{e}^{(t)})$$

- Output at each time step is the concatenation of the outputs of both RNNs at that time step:

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

- *To remember:* Using bidirectional RNN is only possible when the entire sequence is available

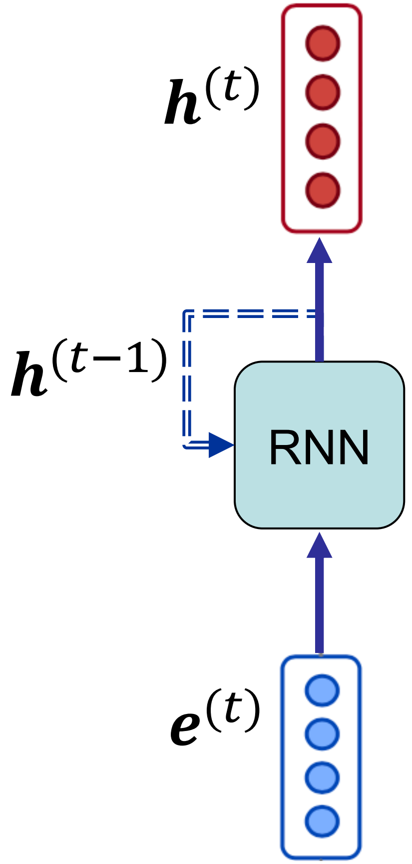


$[\overrightarrow{h}^{(1)}; \overleftarrow{h}^{(1)}]$
$[\overrightarrow{h}^{(2)}; \overleftarrow{h}^{(2)}]$
$[\overrightarrow{h}^{(3)}; \overleftarrow{h}^{(3)}]$
$[\overrightarrow{h}^{(4)}; \overleftarrow{h}^{(4)}]$
$\overrightarrow{h}^{(1)}$
$\overleftarrow{h}^{(1)}$
$\overrightarrow{h}^{(2)}$
$\overleftarrow{h}^{(2)}$
$\overrightarrow{h}^{(3)}$
$\overleftarrow{h}^{(3)}$
$\overrightarrow{h}^{(4)}$
$\overleftarrow{h}^{(4)}$
$\overleftarrow{h}^{(5)}$
$\overrightarrow{h}^{(0)}$
$\overleftarrow{\text{RNN}}$
$\overrightarrow{\text{RNN}}$
$e^{(1)}$
$e^{(2)}$
$e^{(3)}$
$e^{(4)}$

Agenda

- Recurrent Neural Networks
- **Backpropagation Through Time**
- RNNs with Gates: LSTM, GRU

Recurrent Neural Networks – recap



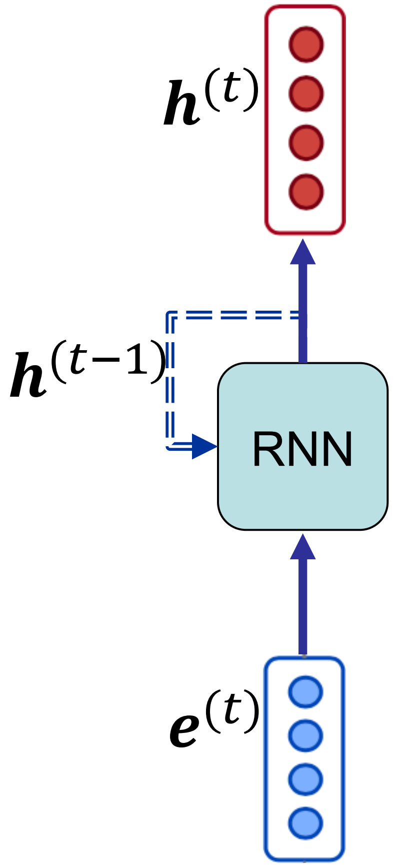
Vanilla (Elman) RNN – recap

- General form of an RNN function

$$\boldsymbol{h}^{(t)} = \mathrm{RNN}(\boldsymbol{h}^{(t-1)}, \boldsymbol{e}^{(t)})$$

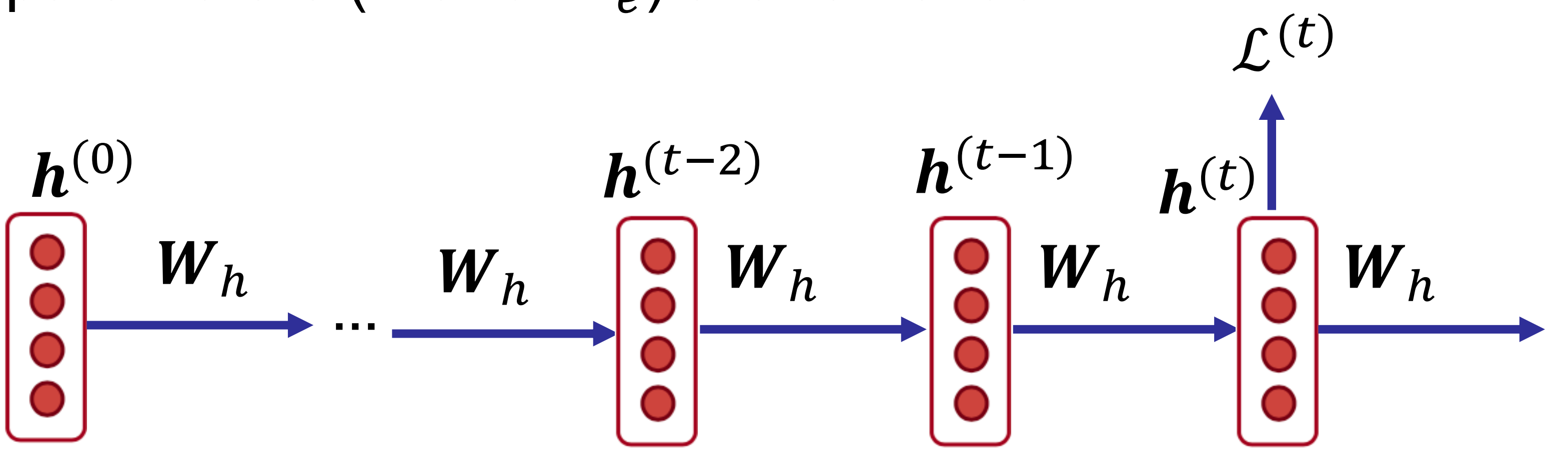
- Vanilla RNN:

$$\boldsymbol{h}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_h + \boldsymbol{e}^{(t)}\boldsymbol{W}_e + \boldsymbol{b})$$



Backpropagation for RNNs

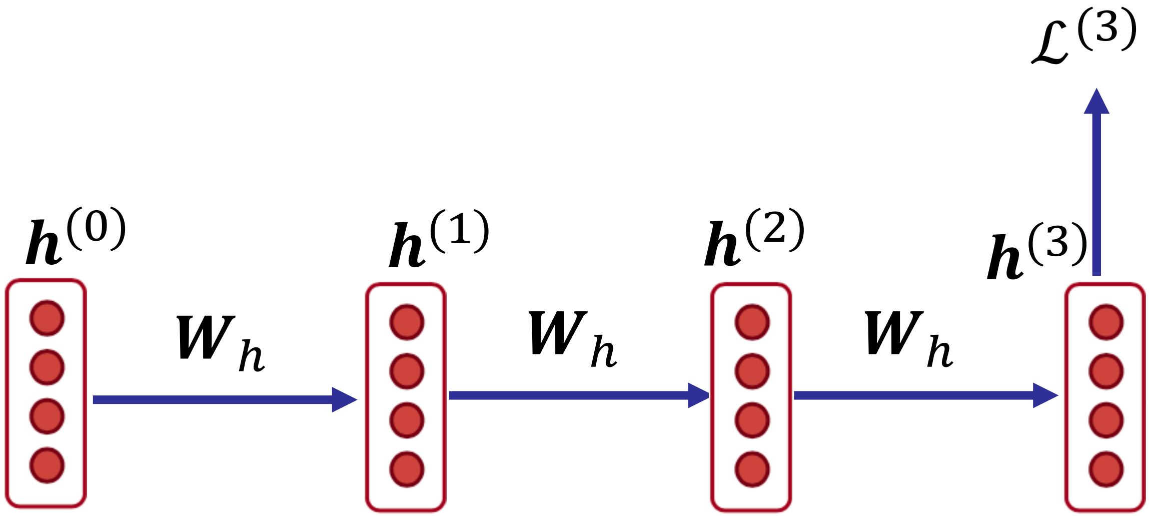
- Unrolling the computation graph of RNN
- Simplified: the interactions with $\boldsymbol{U}$ and also input parameters ($\boldsymbol{E}$ and $\boldsymbol{W}_e$) are removed



- What is …

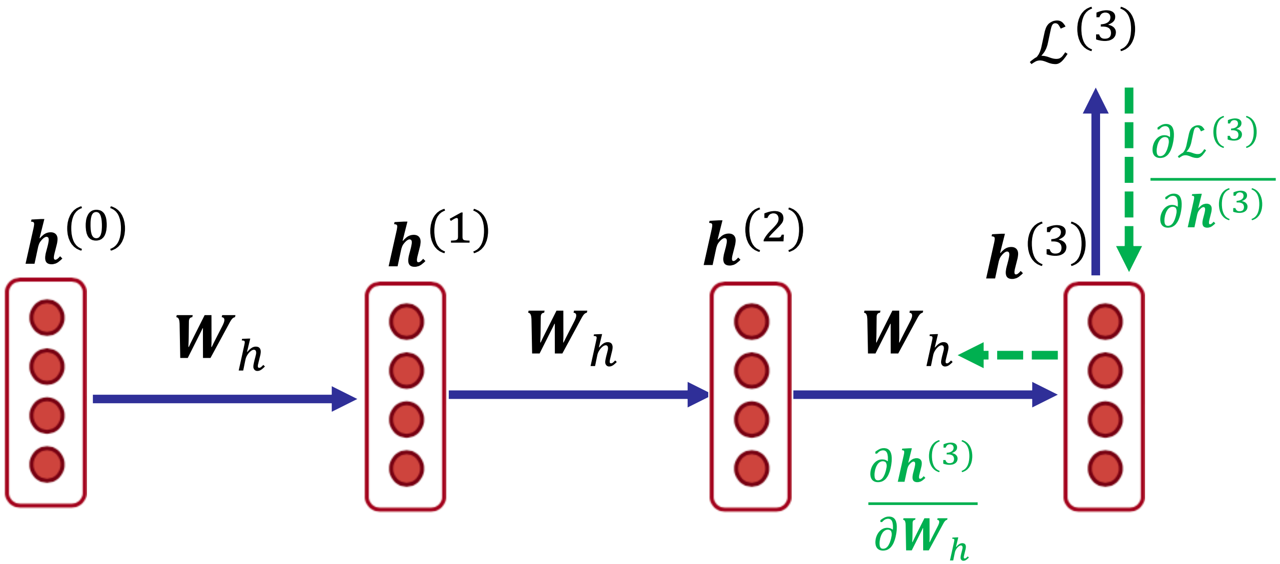
$$\frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h} = ?$$

Backpropagation for RNNs



$$\frac{\partial \mathcal{L}^{(3)}}{\partial \boldsymbol{W}_h} = ?$$

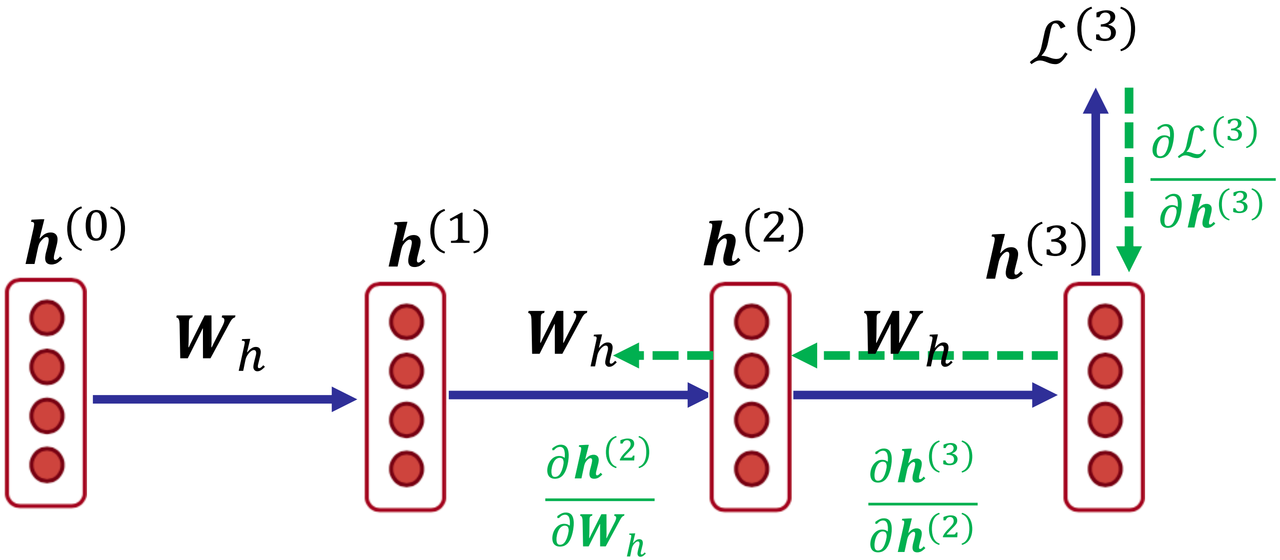
Backpropagation for RNNs



$$\left.\frac{\partial \mathcal{L}^{(3)}}{\partial \boldsymbol{W}_h}\right|_{(3)} = \frac{\partial \mathcal{L}^{(3)}}{\partial \boldsymbol{h}^{(3)}} \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{W}_h}$$

- Gradient regarding $\boldsymbol{W}_h$ at time step 3

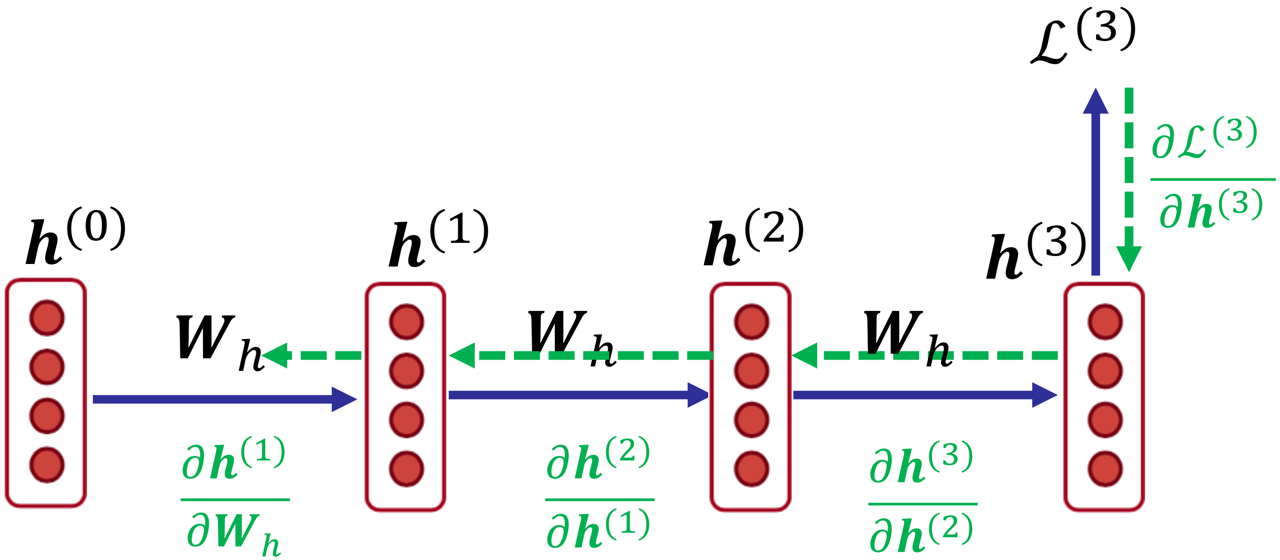
Backpropagation for RNNs



$$\left.\frac{\partial \mathcal{L}^{(3)}}{\partial \boldsymbol{W}_h}\right|_{(2)} = \frac{\partial \mathcal{L}^{(3)}}{\partial \boldsymbol{h}^{(3)}} \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{W}_h}$$

- Gradient regarding $\boldsymbol{W}_h$ at time step 2

Backpropagation for RNNs



$$\left.\frac{\partial \mathcal{L}^{(3)}}{\partial \boldsymbol{W}_h}\right|_{(1)} = \frac{\partial \mathcal{L}^{(3)}}{\partial \boldsymbol{h}^{(3)}} \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{W}_h}$$

- Gradient regarding $\boldsymbol{W}_h$ at time step 1

Backpropagation Through Time (BPTT)

- Final gradient is the sum of the gradients regarding the model parameters (such as $\boldsymbol{W}_h$) from the current time step back to the beginning of corpus (or batch)

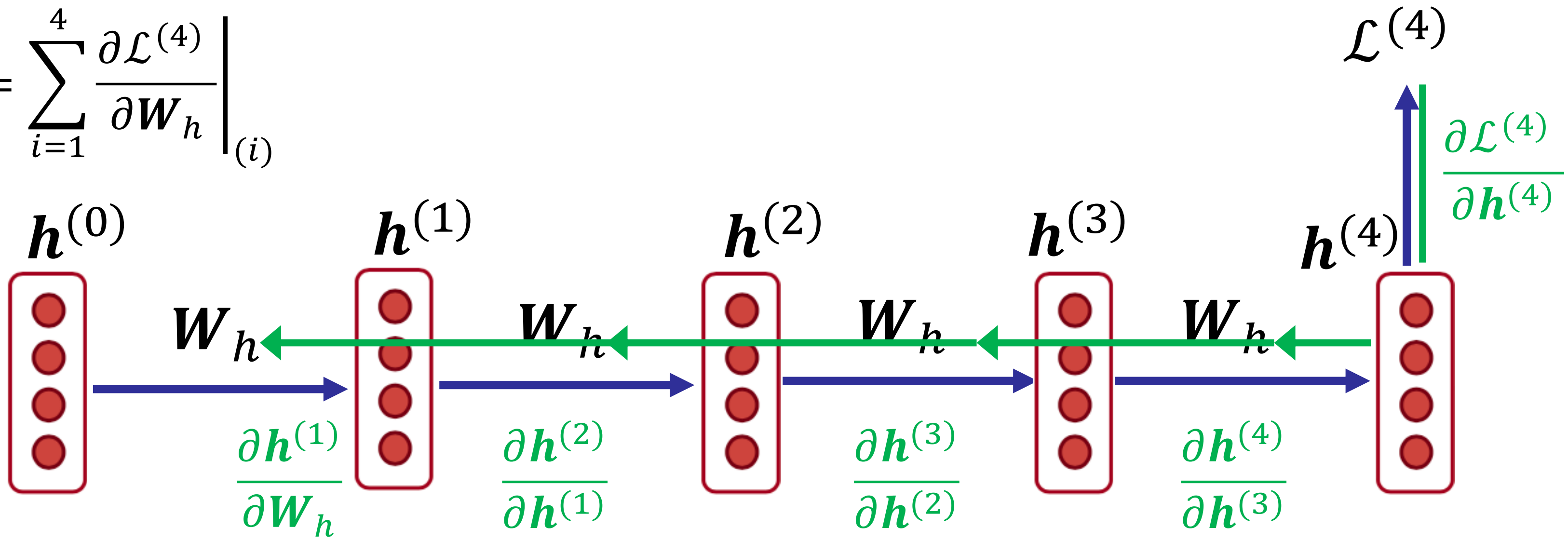
$$\frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{t} \left. \frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h} \right|_{(i)}$$

- In this simplified case, this can be written as:

$$\frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{t} \frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \cdots \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{W}_h}$$

Backpropagation Through Time (BPTT) – all in one!

$$\frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{4} \frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{W}_h}\bigg|_{(i)}$$



$$\frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{W}_h}\bigg|_{(4)} = \frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{h}^{(4)}} \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{W}_h}$$

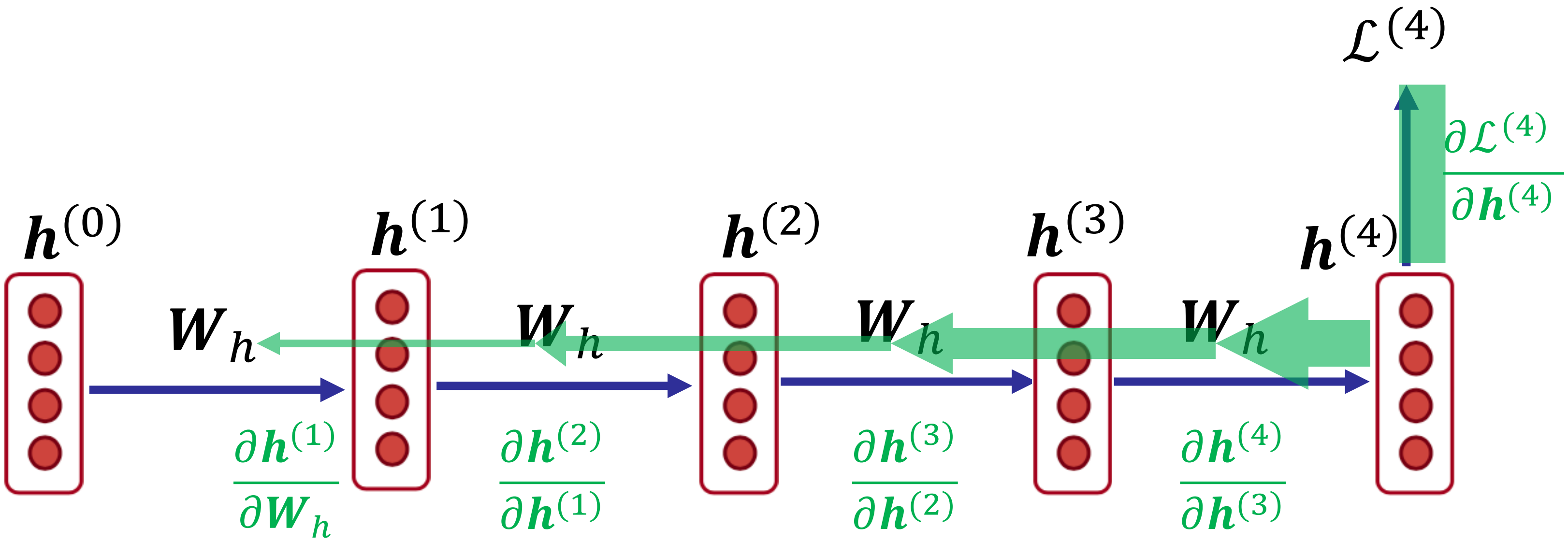
$$\frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{W}_h}\bigg|_{(3)} = \frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{h}^{(4)}} \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{W}_h}$$

$$\frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{W}_h}\bigg|_{(2)} = \frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{h}^{(4)}} \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{W}_h}$$

$$\frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{W}_h}\bigg|_{(1)} = \frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{h}^{(4)}} \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{W}_h}$$

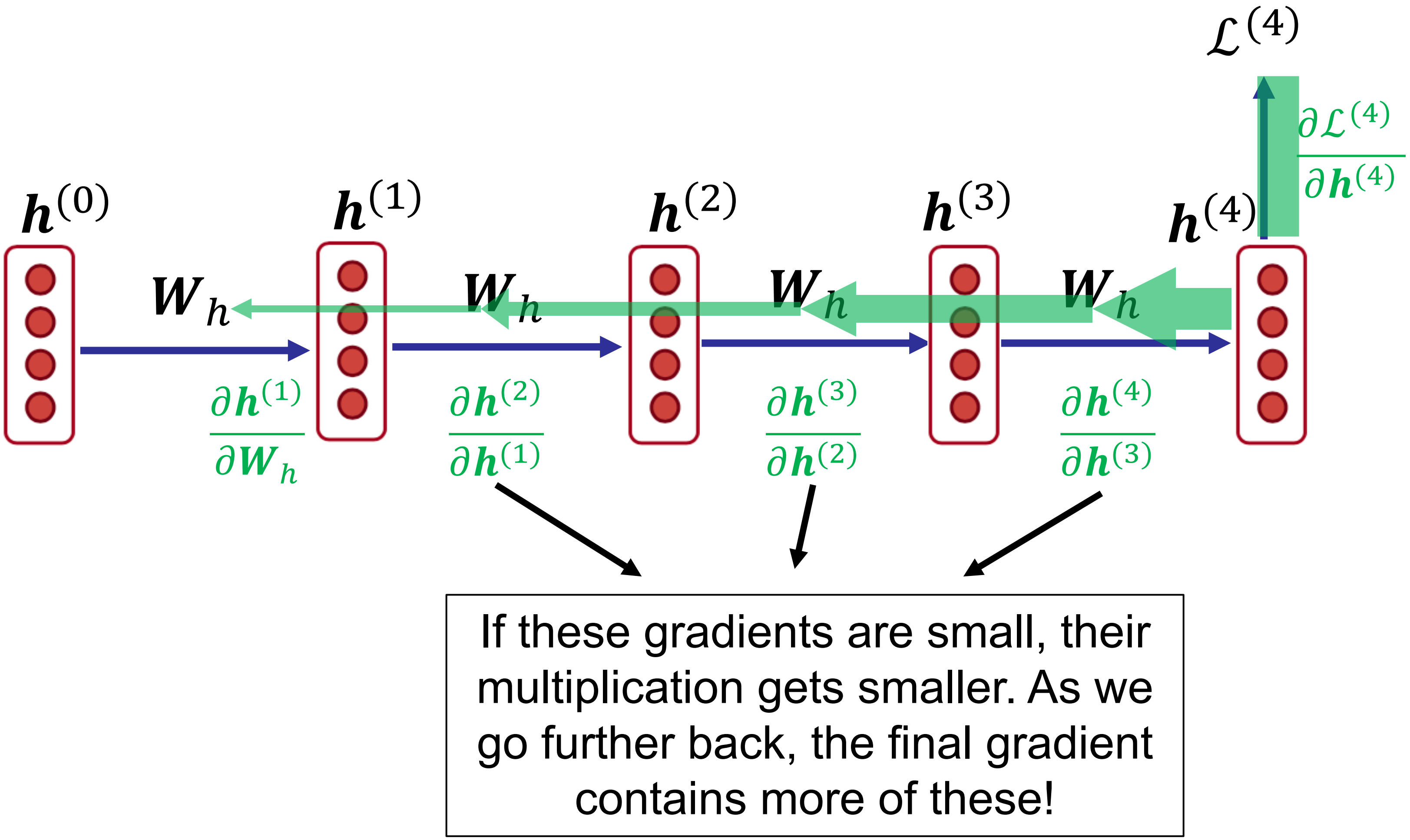
$$\frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{t} \frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h}\bigg|_{(i)} = \sum_{i=1}^{t} \frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \cdots \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{W}_h}$$

Vanishing/Exploding gradient



- In practice, the gradient regarding each time step becomes smaller and smaller as it goes back in time → Vanishing gradient
- While less often, this may also happen other way around: the gradient regarding further time steps becomes larger and larger→ Exploding gradient

Vanishing/Exploding gradient – why?



$$\left.\frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{W}_h}\right|_{(1)} = \frac{\partial \mathcal{L}^{(4)}}{\partial \boldsymbol{h}^{(4)}} \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{W}_h}$$

Vanishing/Exploding gradient – why?

- What is $\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$?!
- Recall the definition of RNN:

$$\boldsymbol{h}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_h + \boldsymbol{e}^{(t)}\boldsymbol{W}_e + \boldsymbol{b})$$

- Let's replace sigmoid (σ) with a simple linear activation ($y = x$) function.

$$\boldsymbol{h}^{(t)} = \boldsymbol{h}^{(t-1)}\boldsymbol{W}_h + \boldsymbol{e}^{(t)}\boldsymbol{W}_e + \boldsymbol{b}$$

- In this case:

$$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} = \boldsymbol{W}_h$$

Vanishing/Exploding gradient – why?

- Recall the BPTT formula (for the simplified case):

$$\left.\frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h}\right|_{(i)} = \frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \underbrace{\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \cdots \frac{\partial \boldsymbol{h}^{(i+1)}}{\partial \boldsymbol{h}^{(i)}}} \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{W}_h}$$

- Given $l = t - i$, the BPTT formula can be rewritten as:

$$\left.\frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{W}_h}\right|_{(i)} = \frac{\partial \mathcal{L}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \boxed{(\boldsymbol{W}_h)^l} \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{W}_h}$$

If weights in $\boldsymbol{W}_h$ are small (i.e. eigenvalues of $\boldsymbol{W}_h$ are smaller than 1), these term gets *exponentially* smaller

Why is vanishing/exploding gradient a problem?

- Vanishing gradient
  - Gradient signal from faraway “fades away” and becomes insignificant in comparison with the gradient signal from close-by
  - Long-term dependencies are not captured, since model weights are updated only with respect to near effects

  → one approach to address it: RNNs with gates – LSTM, GRU

- Exploding gradient
  - Gradients become too big → SGD update steps become too large
  - This causes (large loss values and) large updates on parameters, and eventually unstable training

  → main approach to address it: Gradient clipping

Gradient clipping

- Gradient clipping: if the norm of the gradient is greater than some threshold, scale the gradient down

---

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

- Intuition: take the step in the same direction, but with a smaller step

Problem with vanilla RNN – summary

- It is too difficult for the hidden state of vanilla RNN to learn and preserve information of several time steps
  - In particular as new contents are constantly added to the hidden state in every step

$$\boldsymbol{h}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_h + \boxed{\boldsymbol{e}^{(t)}\boldsymbol{W}_e} + \boldsymbol{b})$$

In every step, input vector “adds” new content to hidden state

Agenda

- Recurrent Neural Networks
- Backpropagation Through Time
- **RNNs with Gates: LSTM, GRU**

Gate vector

- Gate vector:
  - A vector with values between 0 and 1
  - Gate vector acts as "gate-keeper", such that it controls the content flow of another vector
- Gate vectors are typically defined using sigmoid:

$$\boldsymbol{g} = \sigma(some\ vector)$$

… and are applied to a vector $\boldsymbol{v}$ with element-wise multiplication to control its contents:

$$\boldsymbol{g} \odot \boldsymbol{v}$$

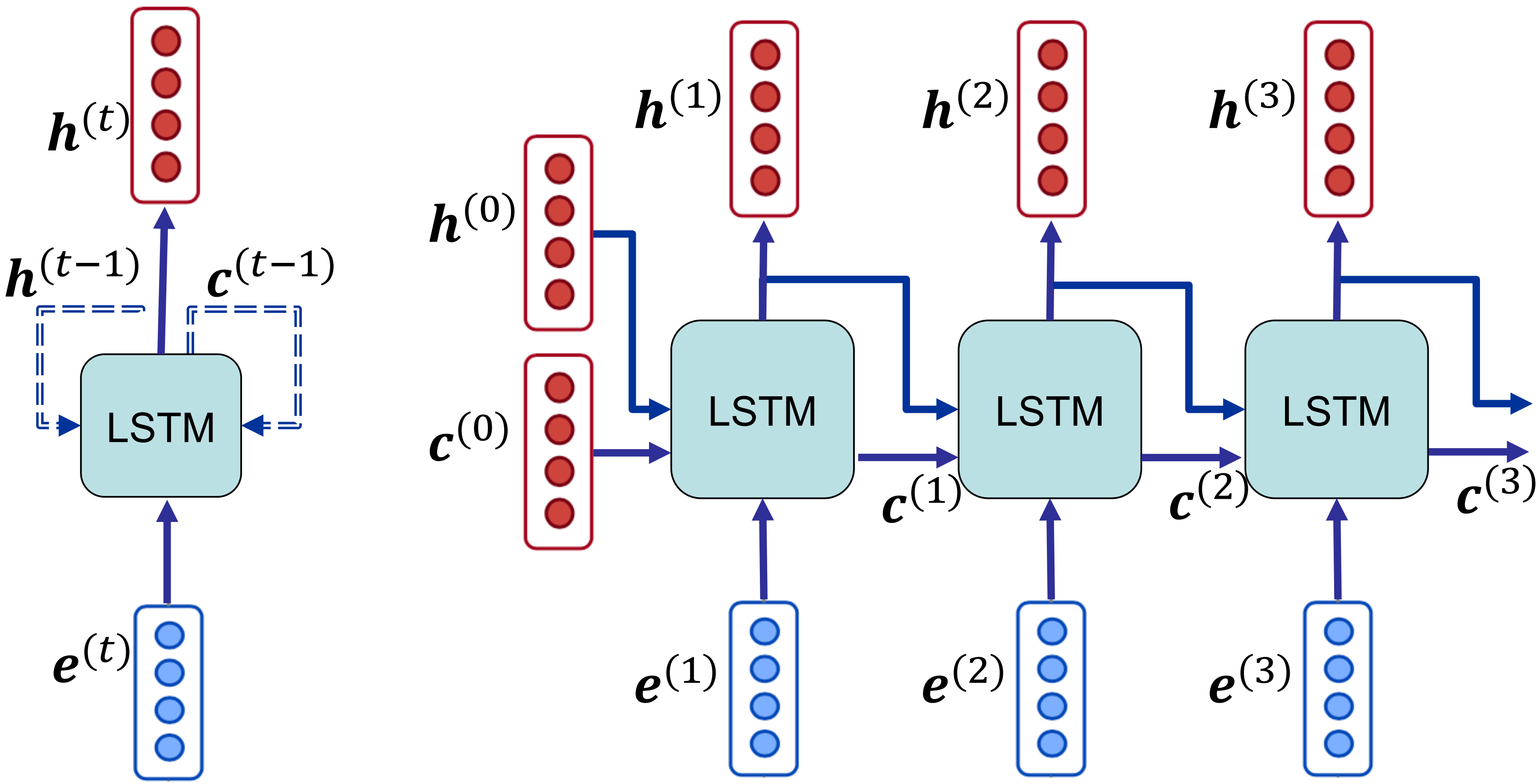
- For each element (feature) i of the vectors:
  - If g_i is 1 $\rightarrow v_i$ remains the same; everything passes; *open* gate!
  - If g_i is 0 $\rightarrow v_i$ becomes 0; nothing passes; *closed* gate!

Long Short-Term Memory (LSTM)

- Proposed by Hochreiter and Schmidhuber in 1997
- LSTM exploits a new vector cell state $\boldsymbol{c}^{(t)}$ to carry the memory of previous states
  - The cell state stores long-term information
  - As in vanilla RNN, hidden states $\boldsymbol{h}^{(t)}$ is used as output vector
- LSTM controls the process of reading, writing, and erasing information in/from memory states
  - These controls are done using gate vectors
  - Gates are dynamic and defined based on the input vector and hidden state

Hochreiter, Sepp, and Jürgen Schmidhuber. "**Long short-term memory**" *Neural computation* (1997)

LSTM – unrolled



LSTM definition – gates

- Gates are functions of input vector $\boldsymbol{e}^{(t)}$ and previous hidden state $\boldsymbol{h}^{(t-1)}$

$$\boldsymbol{i}^{(t)} = \text{function}(\boldsymbol{h}^{(t-1)}, \boldsymbol{e}^{(t)})$$
$$\boldsymbol{i}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{hi} + \boldsymbol{e}^{(t)}\boldsymbol{W}_{xi} + \boldsymbol{b}_i)$$

input gate: controls what parts of the new cell content are written to cell

$$\boldsymbol{f}^{(t)} = \text{function}(\boldsymbol{h}^{(t-1)}, \boldsymbol{e}^{(t)})$$
$$\boldsymbol{f}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{hf} + \boldsymbol{e}^{(t)}\boldsymbol{W}_{xf} + \boldsymbol{b}_f)$$

forget gate: controls what is kept vs forgotten, from previous cell state

$$\boldsymbol{o}^{(t)} = \text{function}(\boldsymbol{h}^{(t-1)}, \boldsymbol{e}^{(t)})$$
$$\boldsymbol{o}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{ho} + \boldsymbol{e}^{(t)}\boldsymbol{W}_{xo} + \boldsymbol{b}_o)$$

output gate: controls what parts of cell are output to hidden state

Parameters are shown in red

Inspired from: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture06-fancy-rnn.pdf

LSTM definition – states

$$\tilde{\boldsymbol{c}}^{(t)} = \text{function}(\boldsymbol{h}^{(t-1)}, \boldsymbol{e}^{(t)})$$

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{hc} + \boldsymbol{e}^{(t)}\boldsymbol{W}_{xc} + \boldsymbol{b}_c)$$

new cell content: the new content to be used for cell and hidden (output) state

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \odot \tilde{\boldsymbol{c}}^{(t)}$$

cell state: erases ("forgets") some content from last cell state, and writes ("inputs") some new cell content

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \odot \tanh(\boldsymbol{c}^{(t)})$$

hidden state: reads ("outputs") some content from the current cell state

Parameters are shown in red

Inspired from: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture06-fancy-rnn.pdf

LSTM definition – all together

$$\boldsymbol{i}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{hi} + \boldsymbol{e}^{(t)}\boldsymbol{W}_{xi} + \boldsymbol{b}_i)$$

input gate: controls what parts of the new cell content are written to cell

$$\boldsymbol{f}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{hf} + \boldsymbol{e}^{(t)}\boldsymbol{W}_{xf} + \boldsymbol{b}_f)$$

forget gate: controls what is kept vs forgotten, from previous cell state

$$\boldsymbol{o}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{ho} + \boldsymbol{e}^{(t)}\boldsymbol{W}_{xo} + \boldsymbol{b}_o)$$

output gate: controls what parts of cell are output to hidden state

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{hc} + \boldsymbol{e}^{(t)}\boldsymbol{W}_{xc} + \boldsymbol{b}_c)$$

new cell content: the new content to be used for cell and hidden (output) state

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \odot \tilde{\boldsymbol{c}}^{(t)}$$

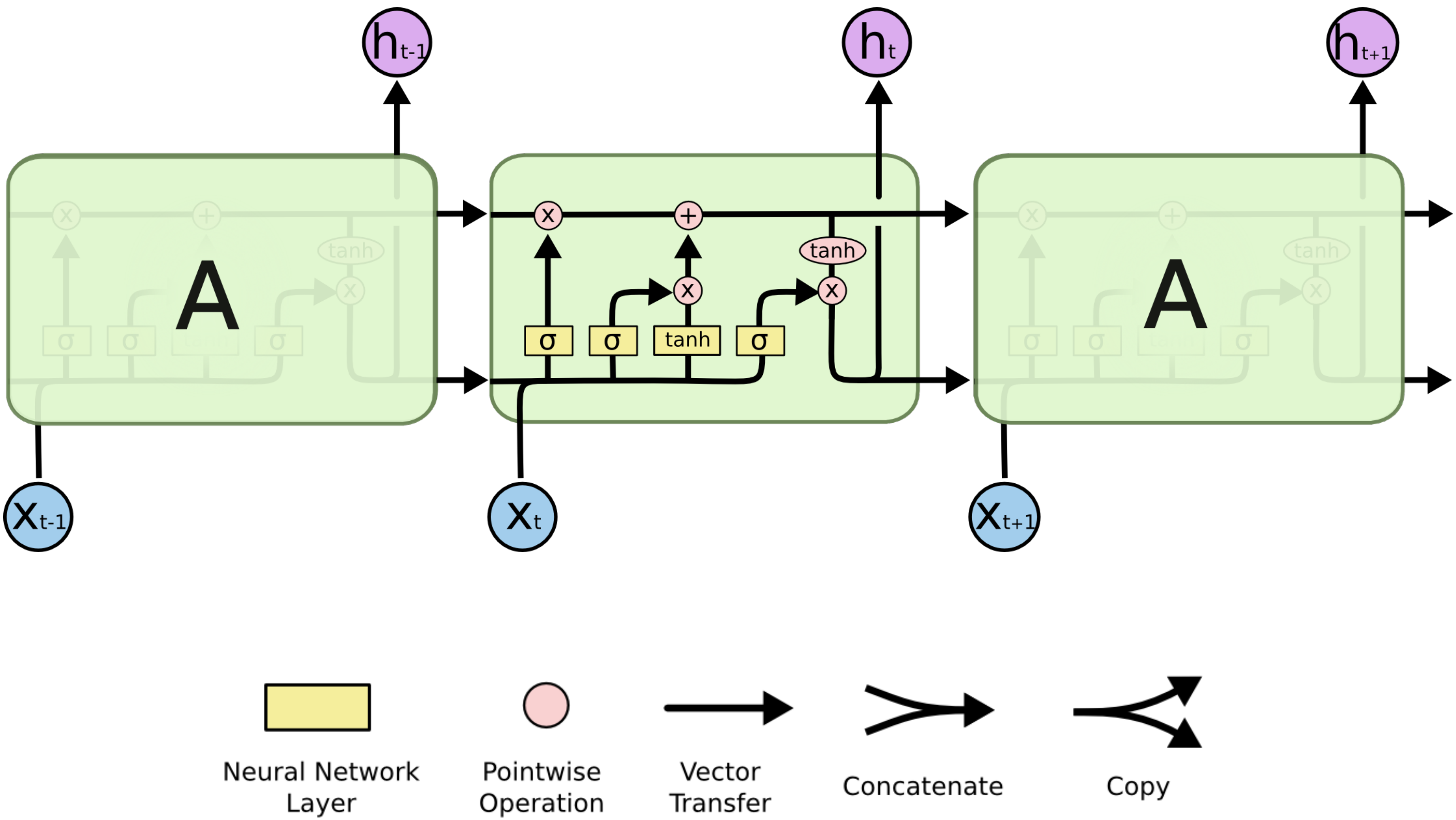
cell state: erases ("forgets") some content from last cell state, and writes ("inputs") some new cell content

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \odot \tanh(\boldsymbol{c}^{(t)})$$

hidden state: reads ("outputs") some content from the current cell state

Parameters are shown in red

LSTM definition – visually!



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gated Recurrent Unit (GRU)

$$\boldsymbol{u}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{hu} + \boldsymbol{e}^{(t)}\boldsymbol{W}_{xu} + \boldsymbol{b}_u)$$

$$\boldsymbol{r}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_{hr} + \boldsymbol{e}^{(t)}\boldsymbol{W}_{xr} + \boldsymbol{b}_r)$$

update gate: controls what parts of hidden state are updated vs preserved

reset gate: controls what parts of previous hidden state are used to compute new content

new hidden state content: (1) reset gate selects useful parts of previous hidden state. (2) Use this and current input to compute new hidden content.

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh((\boldsymbol{r}^{(t)} \odot \boldsymbol{h}^{(t-1)})\,\boldsymbol{W}_{hh} + \boldsymbol{e}^{(t)}\boldsymbol{W}_{xh} + \boldsymbol{b}_h)$$

$$\boldsymbol{h}^{(t)} = \left(1 - \boldsymbol{u}^{(t)}\right) \odot \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \odot \tilde{\boldsymbol{h}}^{(t)}$$

hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

Parameters are shown in red

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). **Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation**. In *Proc. Of EMNLP)*

RNNs with gates – counting parameters

- Parameters in LSTM (bias terms discarded)
  - $\boldsymbol{W}_{hi}, \boldsymbol{W}_{hf}, \boldsymbol{W}_{ho}, \boldsymbol{W}_{hc} \rightarrow h \times h \; * 4$
  - $\boldsymbol{W}_{xi}, \boldsymbol{W}_{xf}, \boldsymbol{W}_{xo}, \boldsymbol{W}_{xc} \rightarrow d \times h \; * 4$

- Parameters in GRU (bias terms discarded)
  - $\boldsymbol{W}_{hu}, \boldsymbol{W}_{hr}, \boldsymbol{W}_{hh} \rightarrow h \times h \; * 3$
  - $\boldsymbol{W}_{xu}, \boldsymbol{W}_{xr}, \boldsymbol{W}_{xh} \rightarrow d \times h \; * 3$

- If also considering encoder and decoder embeddings (e.g., in a Language Modeling network)
  - $\boldsymbol{E} \rightarrow N \times d$
  - $\boldsymbol{U} \rightarrow h \times N$

d: dimension of input embedding
h: dimension of hidden vectors and output embedding

RNNs with gates – summary

- LSTM (and GRU) with dynamic gate mechanisms makes it easier to preserve necessary information over many timesteps

- LSTM does not *guarantee* that there is no vanishing/exploding gradient, but its large success in practice has shown that it can learn long-distance dependencies

- LSTM vs. GRU: LSTM is usually the default choice. Especially, when enough training data is available and capturing longer distances is important. GRU is faster and more suited for settings with low computation resources