Neural Networks for Natural Language Processing

Lecture 5 – RNN/LSTM/CNN Language Models

20.11.2023

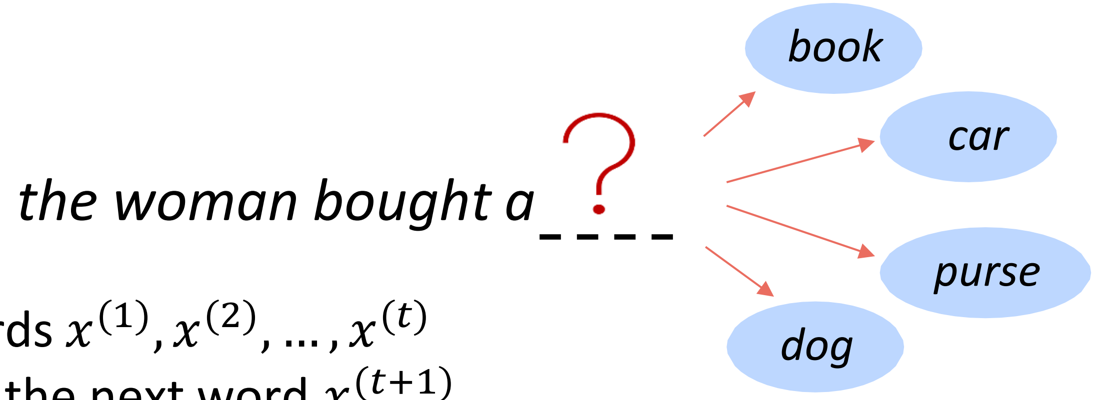
Jun.-Prof. Sophie Fellenz

Agenda

- Recap: N-gram language models
- RNN language models
- LSTM language models
- CNN language models

Language Modeling

- Language Modeling is the task of predicting what word comes next.

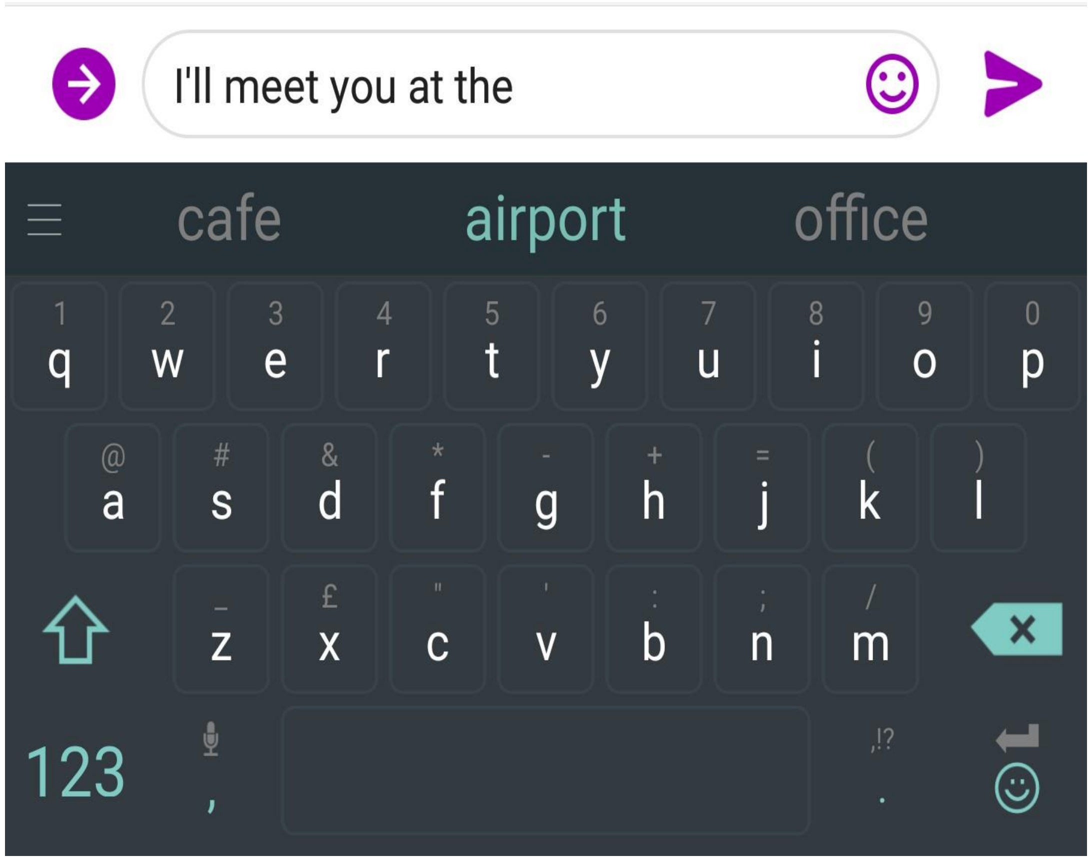


- More formally: given a sequence of words $x^{(1)}, x^{(2)}, \dots, x^{(t)}$
- compute the probability distribution of the next word $x^{(t+1)}$
- where, w_j is a word in the vocabulary $V = \{w_1, \dots., w_{|V|}\}$

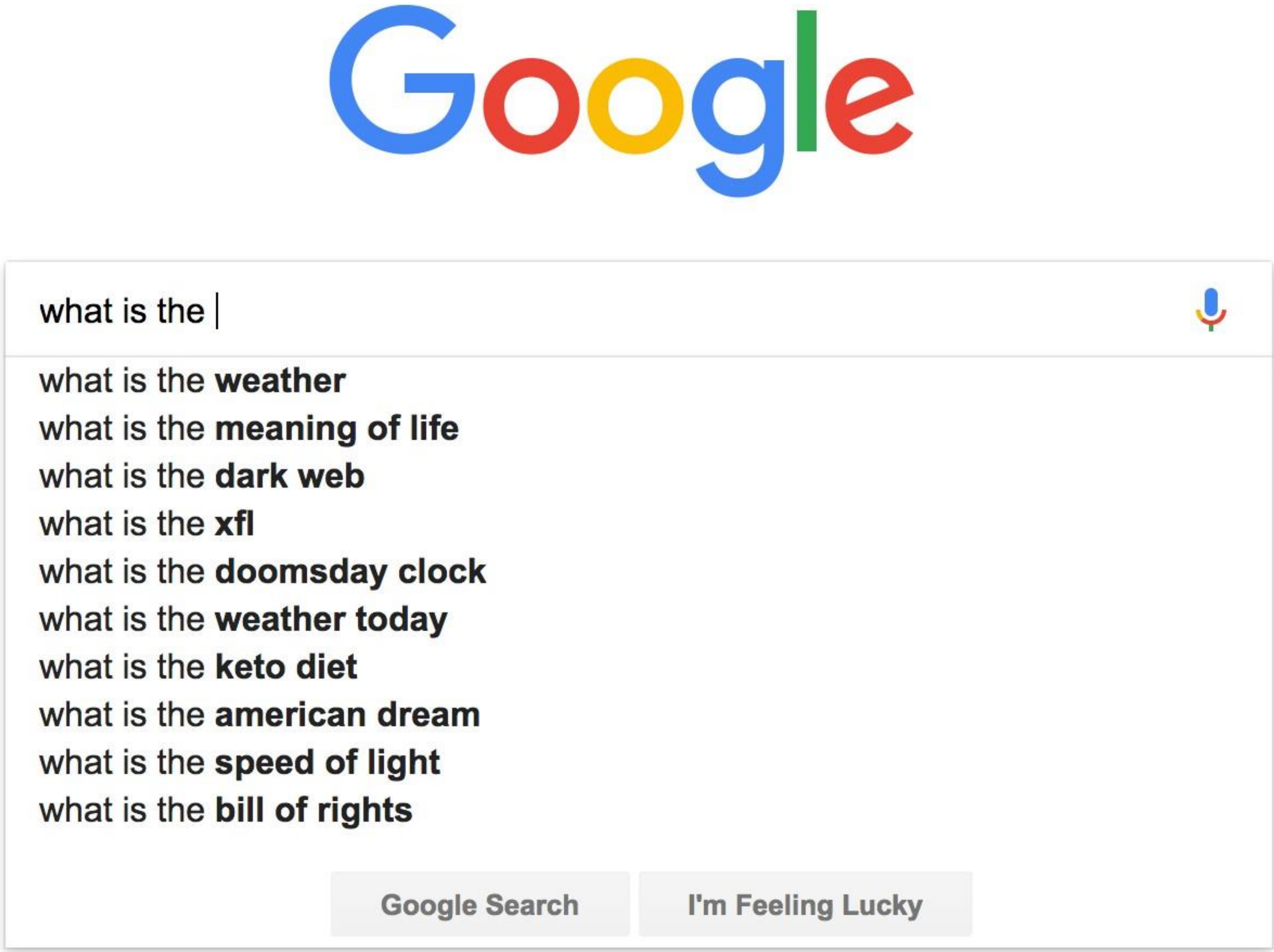
$$P(x^{(t+1)} = w_j | x^{(t)}, \dots, x^{(1)})$$

- A system that does this is called a **Language Model**.

We use language models everyday!



We use language models everyday!



n-gram Language Models

- *the woman bought a _ _ _ _*
- *Question:* How to learn a language model?
- *Answer* (prior to Deep Learning): learn an *n*-gram Language Model!
- *Definition:* An *n*-gram is a chunk of *n* consecutive words.
  - unigrams: “the”, “woman”, “bought”, “a”
  - bigrams: “the woman”, “woman bought”, “bought a”
  - trigrams: “the woman bought”, “woman bought a”
  - 4-grams: “the woman bought a”
- *Idea:* Frequency of the n-grams can be used to predict the next word in the sequence.

Generating text with an n-gram language model

You can also use a language model to generate text

today the price of ________

Condition on this

get probability distribution

the	0.072	
18	0.043	
oil	0.043	
its	0.036	
gold	0.018	sample
…		

Generating text with an n-gram language model

You can also use a language model to generate text.

today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks, sept 30 end primary 76 cts a share .

Incoherent! We need to consider more than 3 words at a time if we want to generate good text.

But increasing *n* worsens sparsity problem, and exponentially increases model size…

How to build a *neural* Language Model

- Recall the Language Modeling task:
  - Input: sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$
  - Output: prob dist of the next word $P\left(x^{(t+1)} = w_j \mid x^{(t)}, \ldots, x^{(1)}\right)$

- **How about a window-based neural model?**

~~after winning a big lottery,~~ the woman bought a

discard

fixed window

A fixed-window neural Language Model

output distribution

$$\hat{\boldsymbol{y}} = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

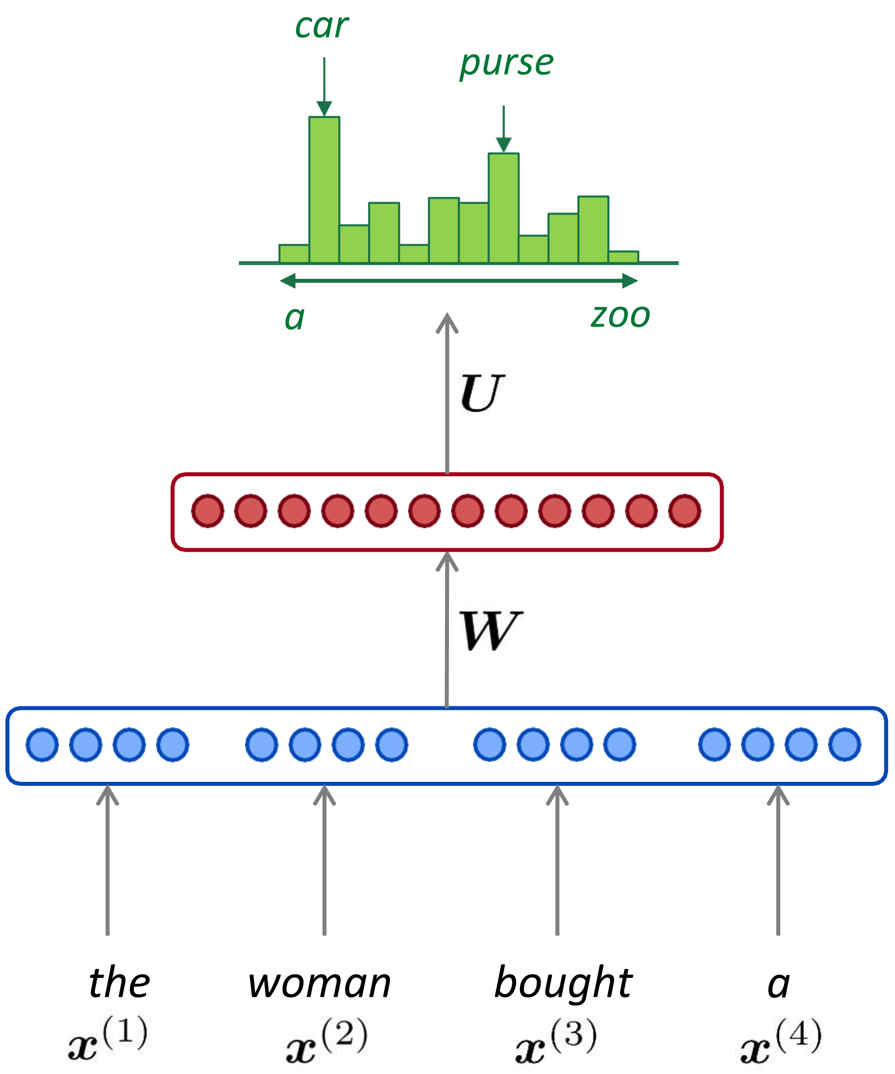
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [e^{(1)}; e^{(2)}; e^{(3)}; e^{(4)}]$$

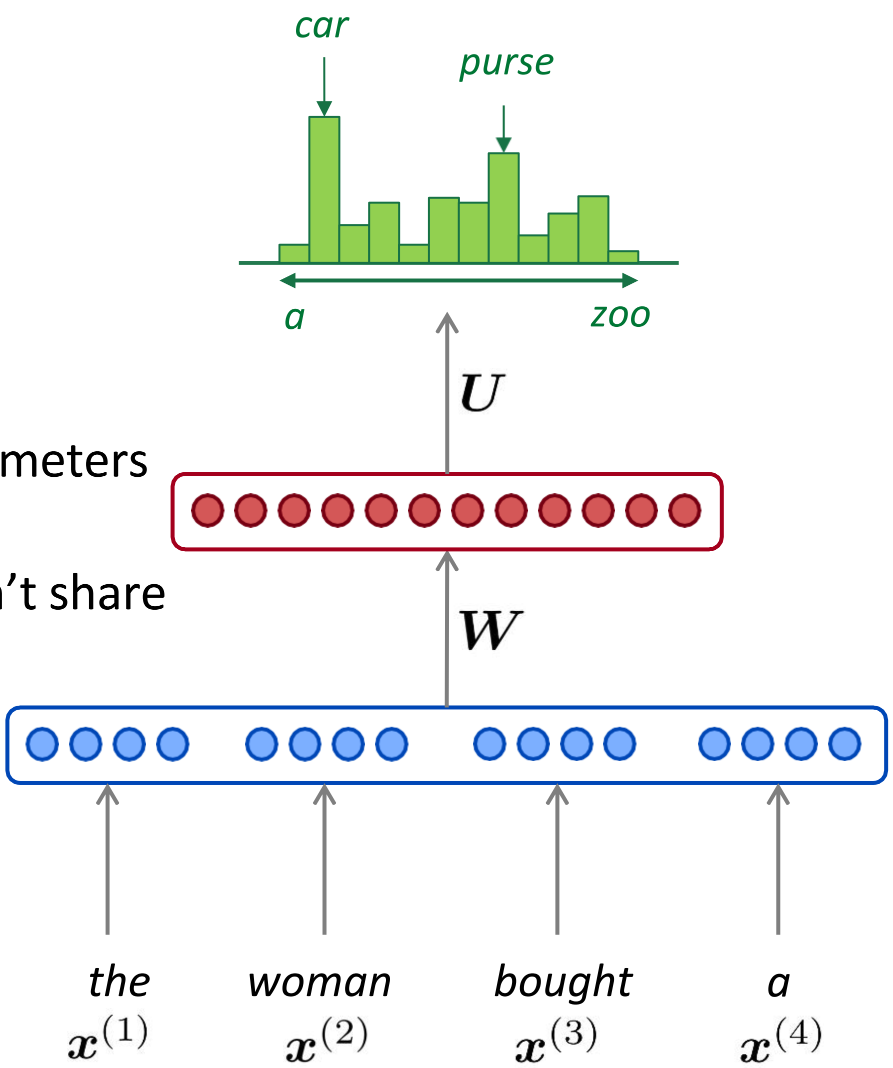
words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



A fixed-window neural Language Model

- Improvements over *n*-gram LM:
  - No sparsity problem
  - Model size is O(*n*) not O(exp(*n*))
- Remaining problems:
  - Fixed window is too small
  - Enlarging window enlarges number of parameters
  - Window can never be large enough!
  - Each $x^{(i)}$ uses different rows of w. We don't share weights across the window.

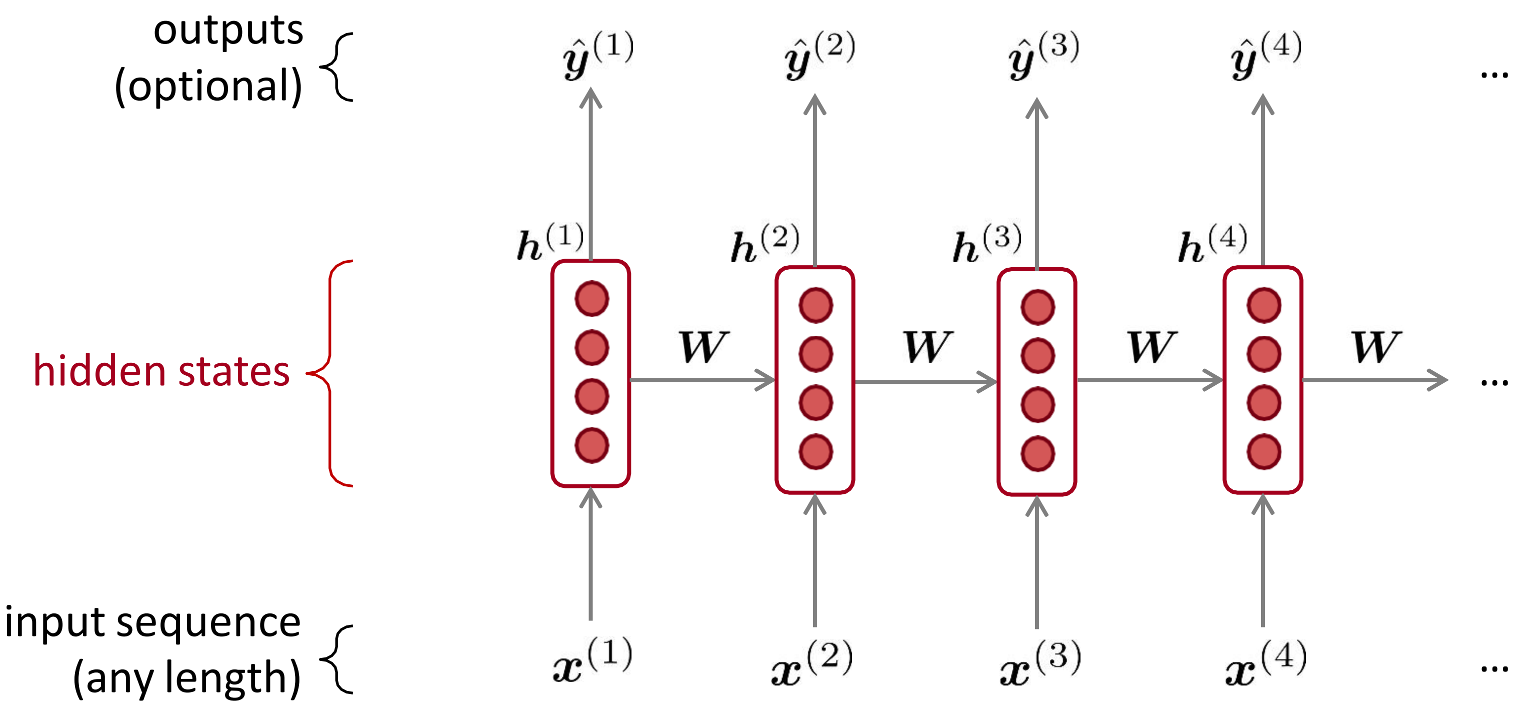


We need a neural architecture that can process *any length input.*

Recurrent Neural Network (RNN)

- **A family of neural architectures**

Core idea: Apply the same weights $\boldsymbol{W}$ *repeatedly*



A *(typical)* RNN Language Model

output distribution

$\hat{y}^{(t)} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$

hidden states

$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e \boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$

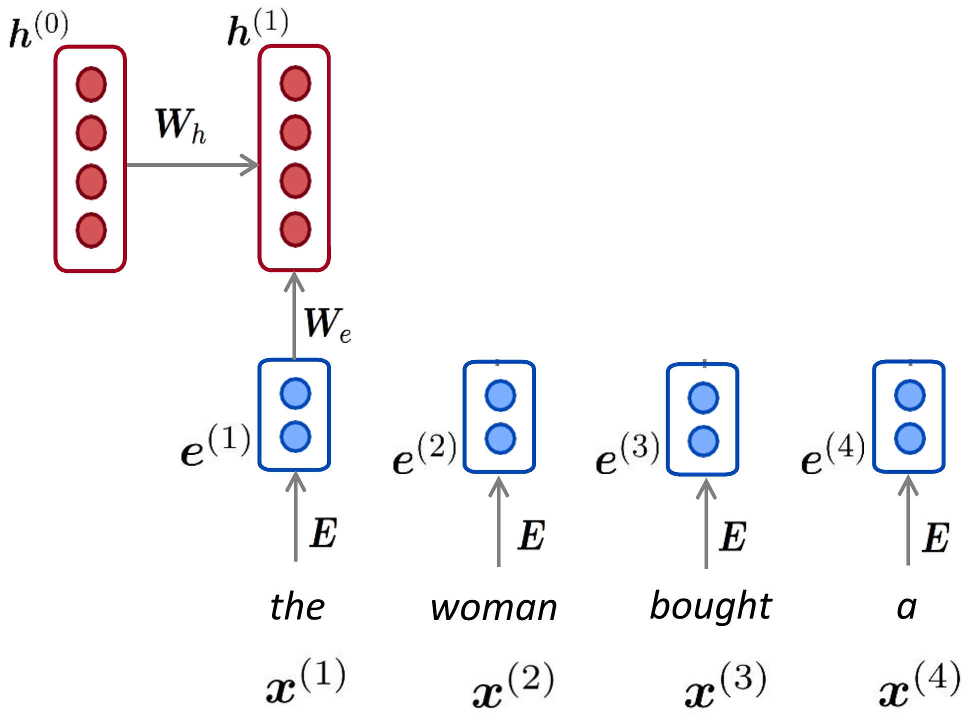
$\boldsymbol{h}_{(0)}$ is the initial hidden state

word embeddings

$\boldsymbol{e}^{(t)} = \boldsymbol{E}\,\boldsymbol{x}^{(t)}$

words / one-hot vector

$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$



***Note**: the input sequence could be much longer*

A *(typical)* RNN Language Model

output distribution

$\hat{y}^{(t)} = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b_2}) \in \mathbb{R}^{|V|}$

hidden states

$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e \boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$

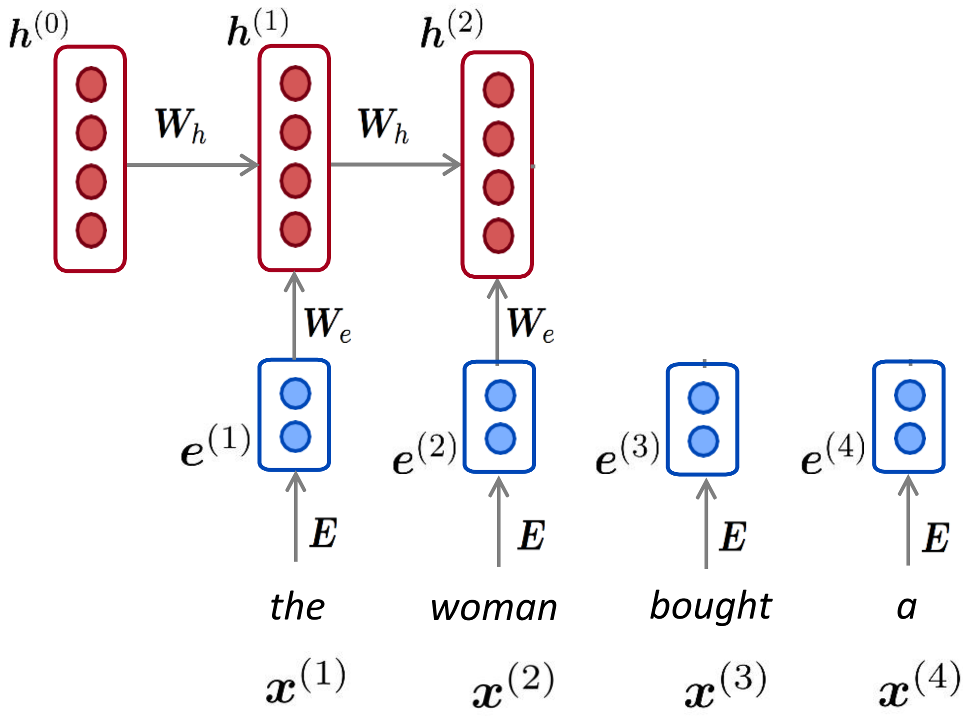
$\boldsymbol{h}_{(0)}$ is the initial hidden state

word embeddings

$\boldsymbol{e}^{(t)} = \boldsymbol{E}\,\boldsymbol{x}^{(t)}$

words / one-hot vector

$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$



***Note**: the input sequence could be much longer*

A *(typical)* RNN Language Model

output distribution

$\hat{y}^{(t)} = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$

hidden states

$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e \boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$

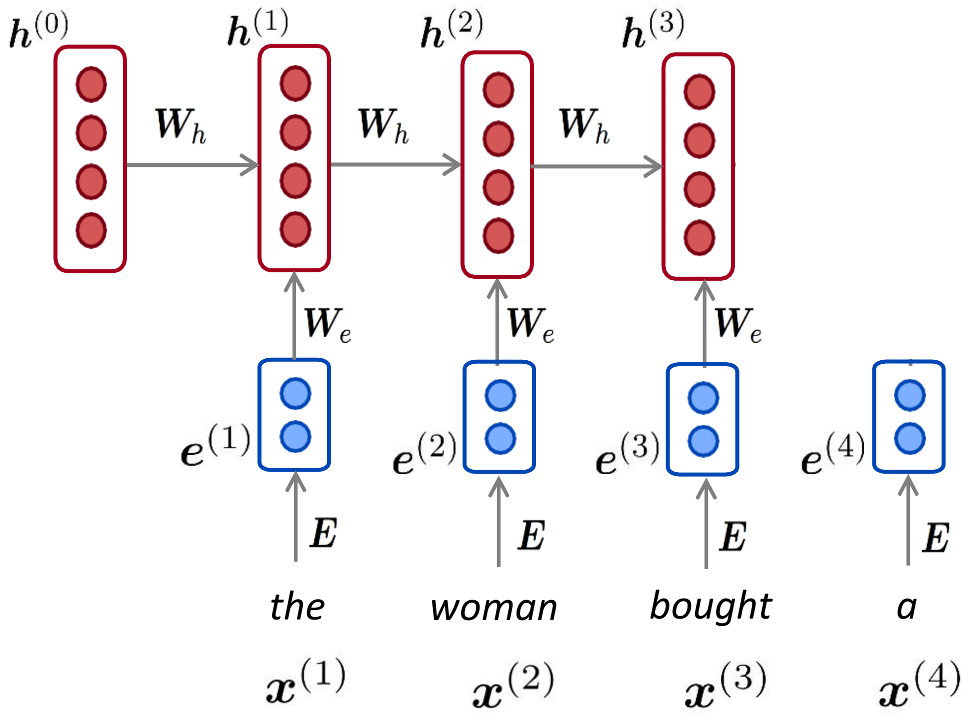
$\boldsymbol{h}_{(0)}$ is the initial hidden state

word embeddings

$\boldsymbol{e}^{(t)} = \boldsymbol{E}\,\boldsymbol{x}^{(t)}$

words / one-hot vector

$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$



Note: *the input sequence could be much longer*

A *(typical)* RNN Language Model

output distribution

$\hat{y}^{(t)} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$

hidden states

$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e \boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$

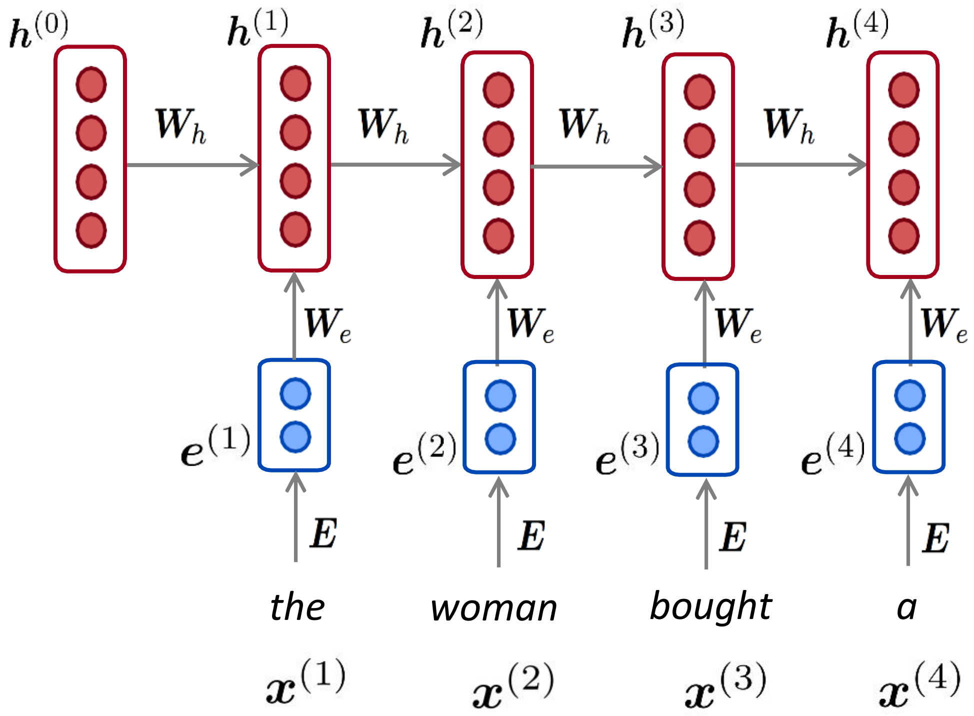
$\boldsymbol{h}_{(\boldsymbol{0})}$ is the initial hidden state

word embeddings

$\boldsymbol{e}^{(t)} = \boldsymbol{E}\,\boldsymbol{x}^{(t)}$

words / one-hot vector

$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$



Note: *the input sequence could be much longer*

A *(typical)* RNN Language Model

output distribution

$\hat{y}^{(t)} = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$

hidden states

$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e \boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$

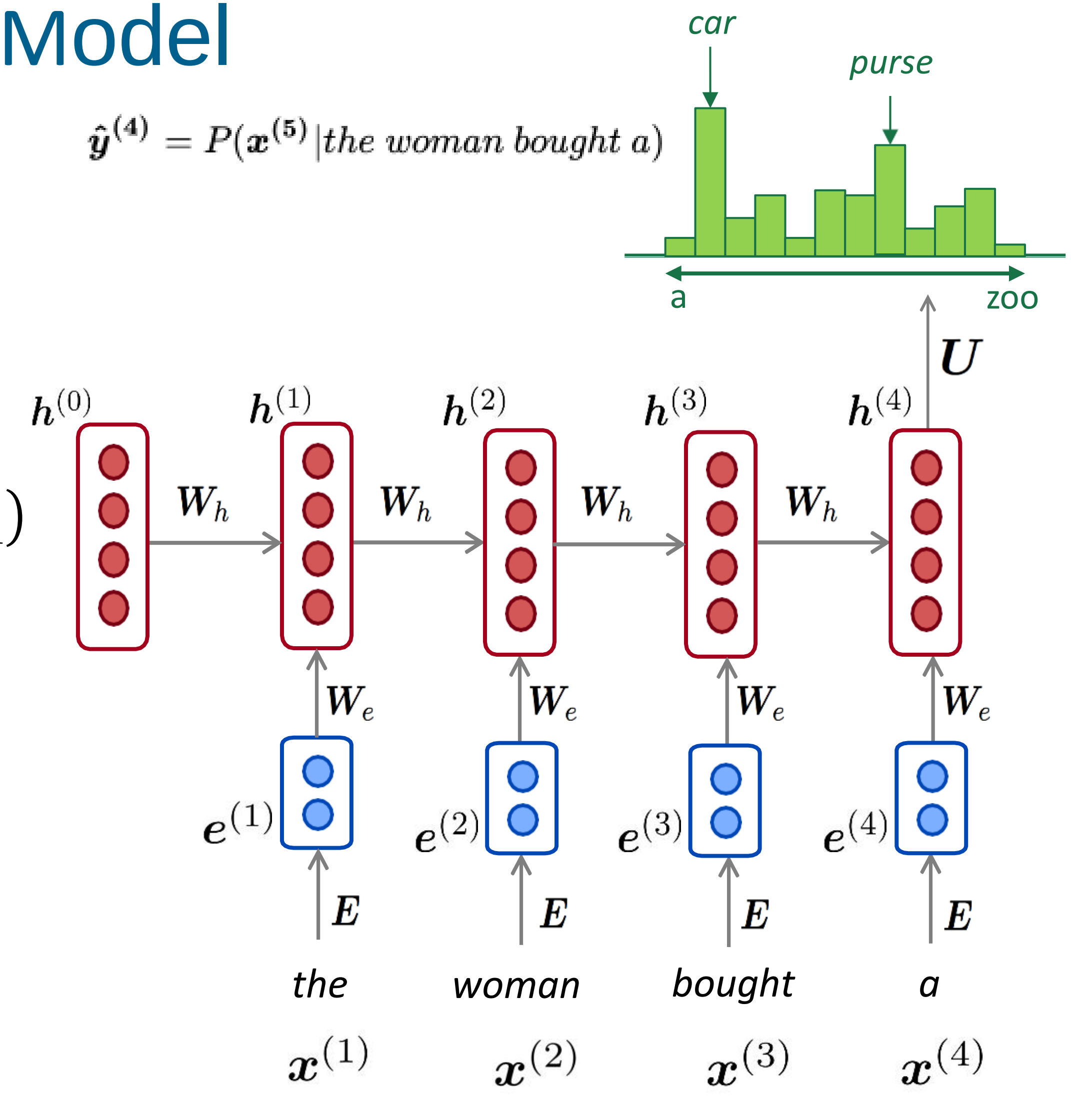
$\boldsymbol{h}_{(0)}$ is the initial hidden state

word embeddings

$\boldsymbol{e}^{(t)} = \boldsymbol{E}\,\boldsymbol{x}^{(t)}$

words / one-hot vector

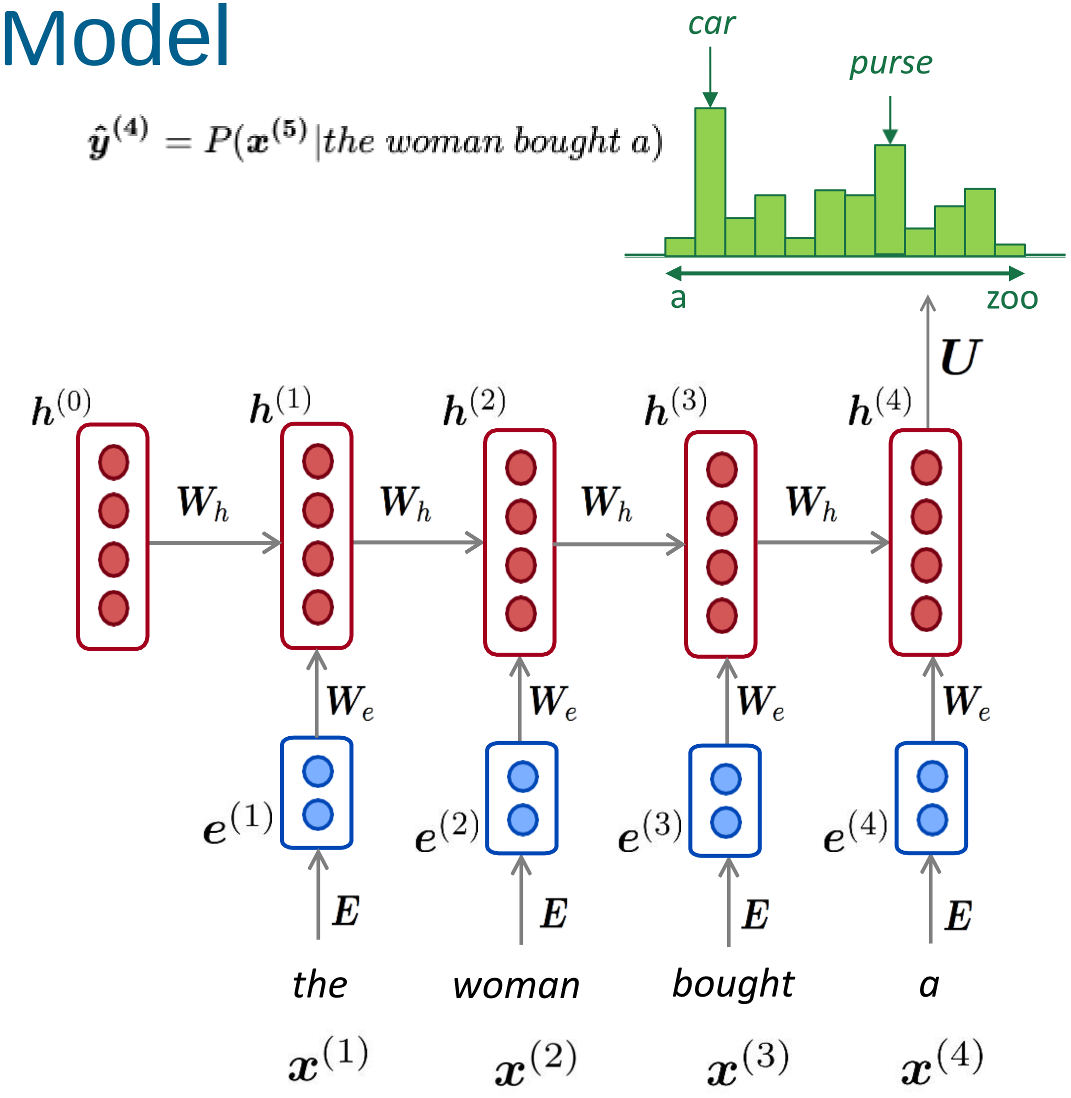
$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$



Note: *the input sequence could be much longer*

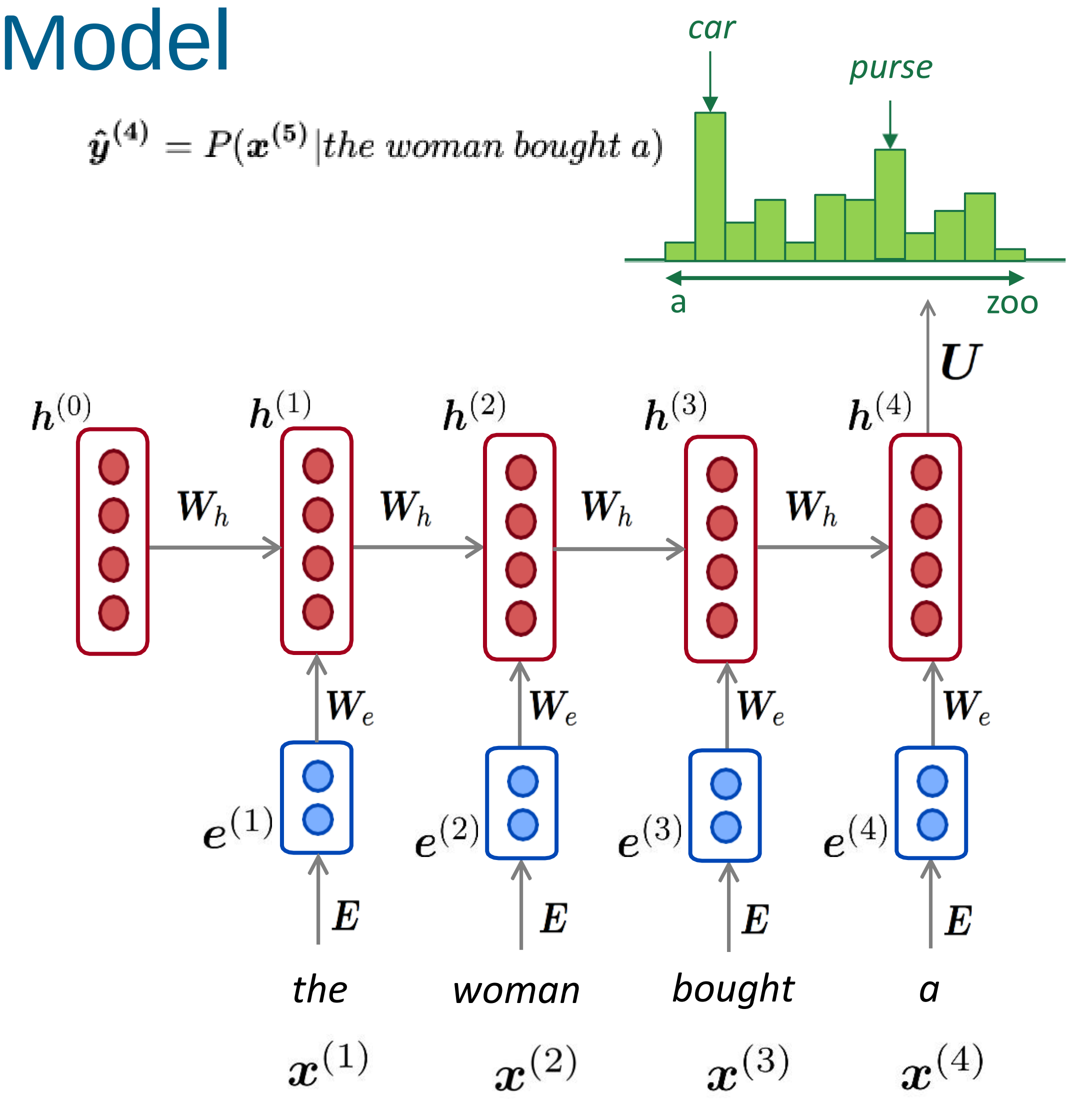
A *(typical)* RNN Language Model

- RNN Advantages:
  - Can process any length input
  - Model size doesn't increase for longer input.
  - Computation for step t can (in theory) use information from many steps back.
  - Weights are shared across timesteps.



A *(typical)* RNN Language Model

- RNN Disadvantages:
  - Recurrent computation is slow.
  - In practice, difficult to access information from many steps back.



Training a RNN Language Model

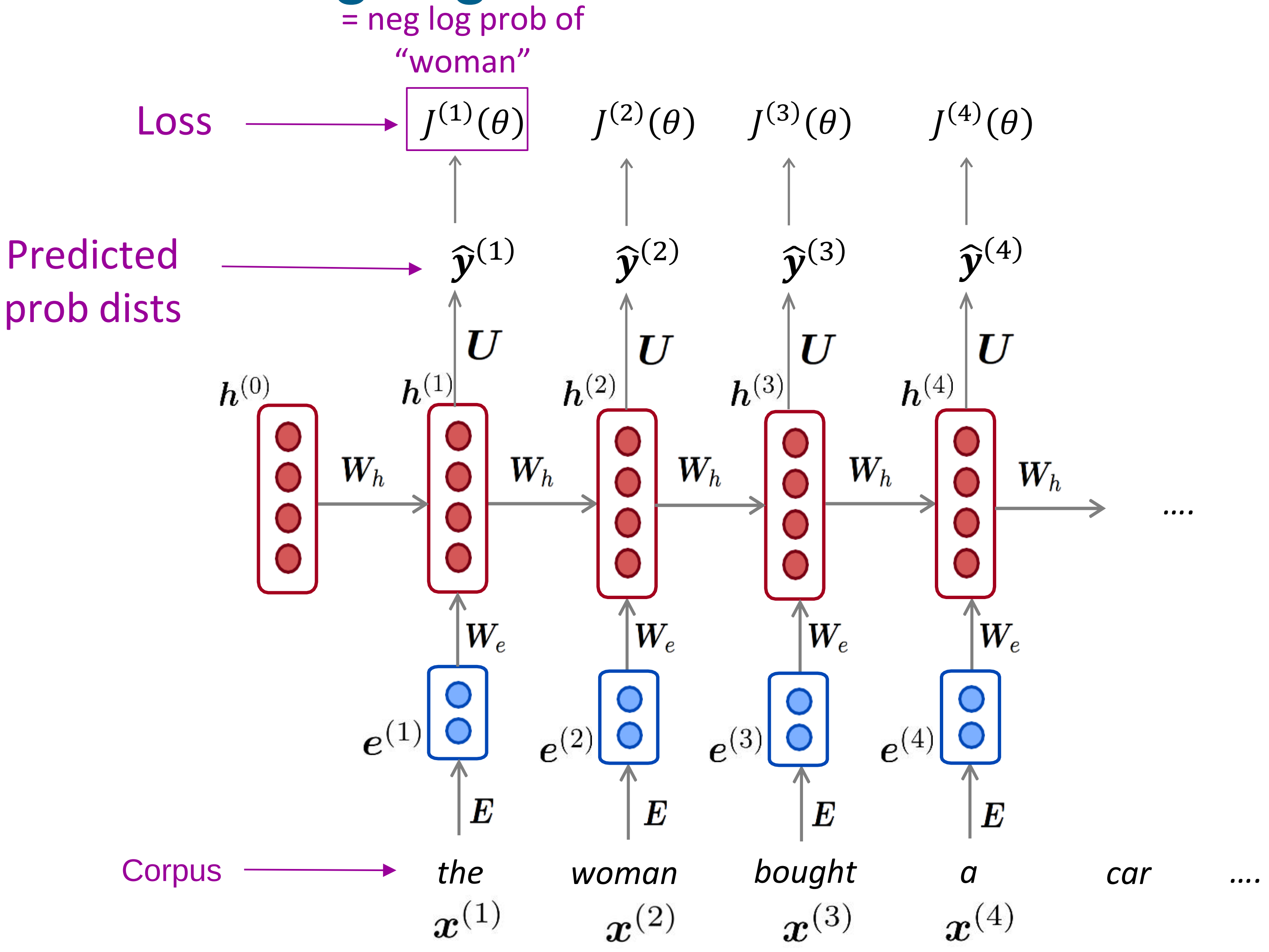
- Get a big corpus of text which is a sequence of words $x^{(1)}, \ldots, x^{(T)}$
- Feed the sequence into RNN; compute output distribution $\hat{y}^{(t)}$ for every step t.
- i.e. predict probability distribution of every word, given words so far
- Loss function on step t is usual cross-entropy between our predicted probability distribution $\hat{y}^{(t)}$,and the true next word $y^{(t)} = x^{(t+1)}$:

- Average this to get overall loss for entire training set:

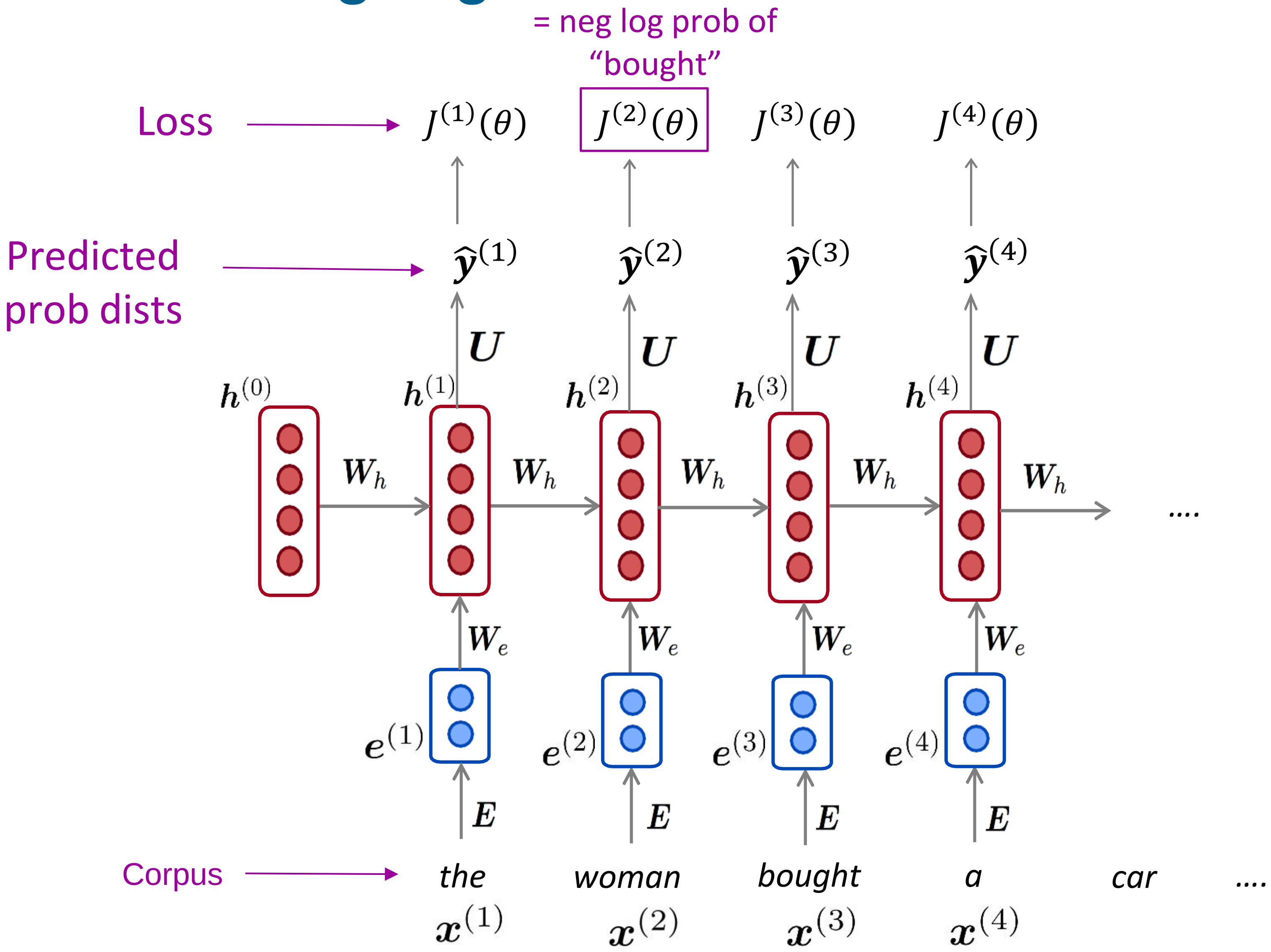
$$J^{(t)}(\theta) = CE\left(y^{(t)}, \hat{y}^{(t)}\right) = -\sum_{j=1}^{|V|} y_j^{(t)} \, log \, \hat{y}_j^{(t)} = -\log \hat{y}_{x_{t+1}}^{(t)}$$

$$\boxed{\begin{aligned} J(\theta) &= \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) \\ &= \frac{1}{T}\sum_{t=1}^{T} -\log \hat{y}_{x_{t+1}}^{(t)} \end{aligned}}$$

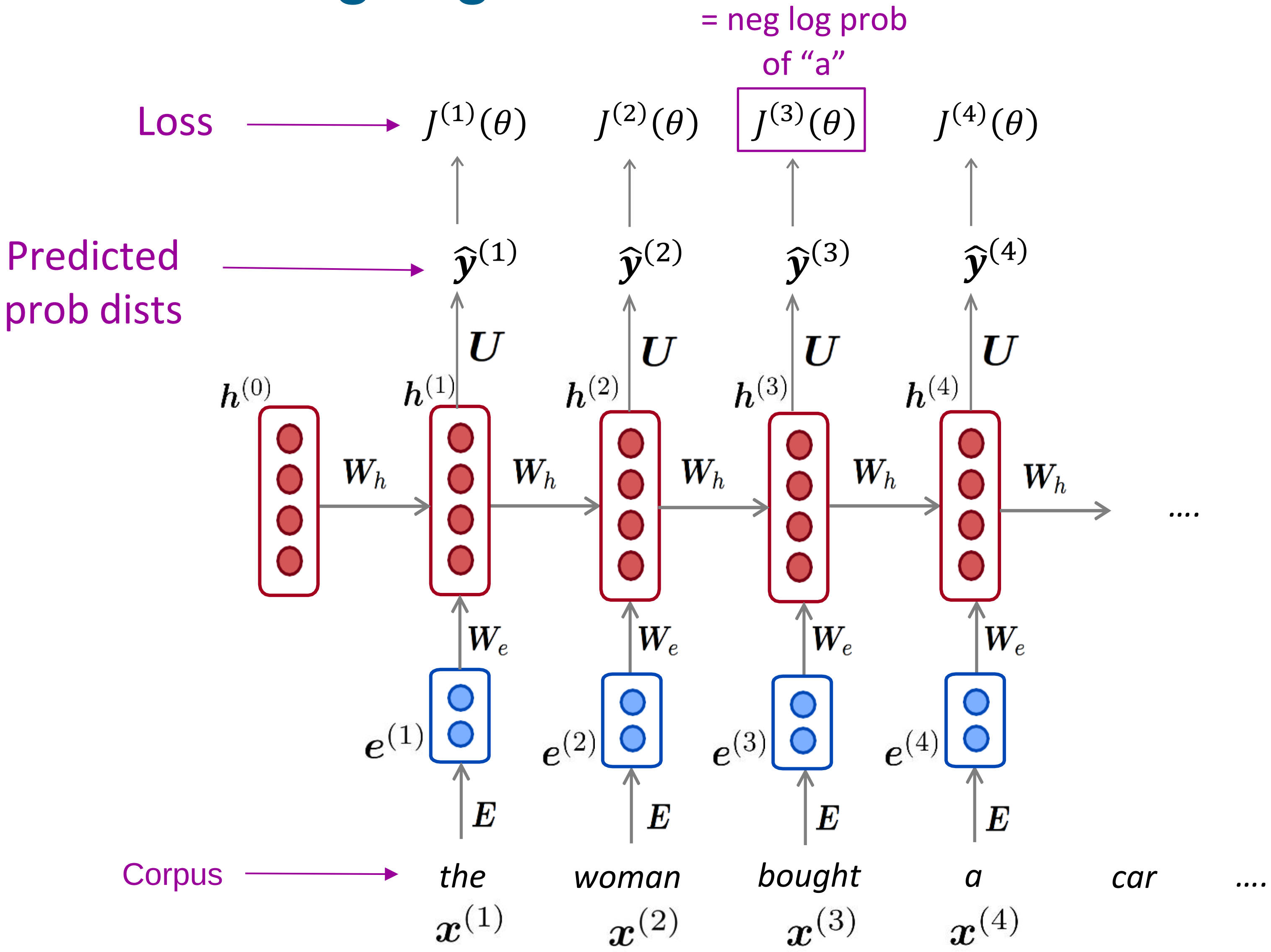
Training a RNN Language Model



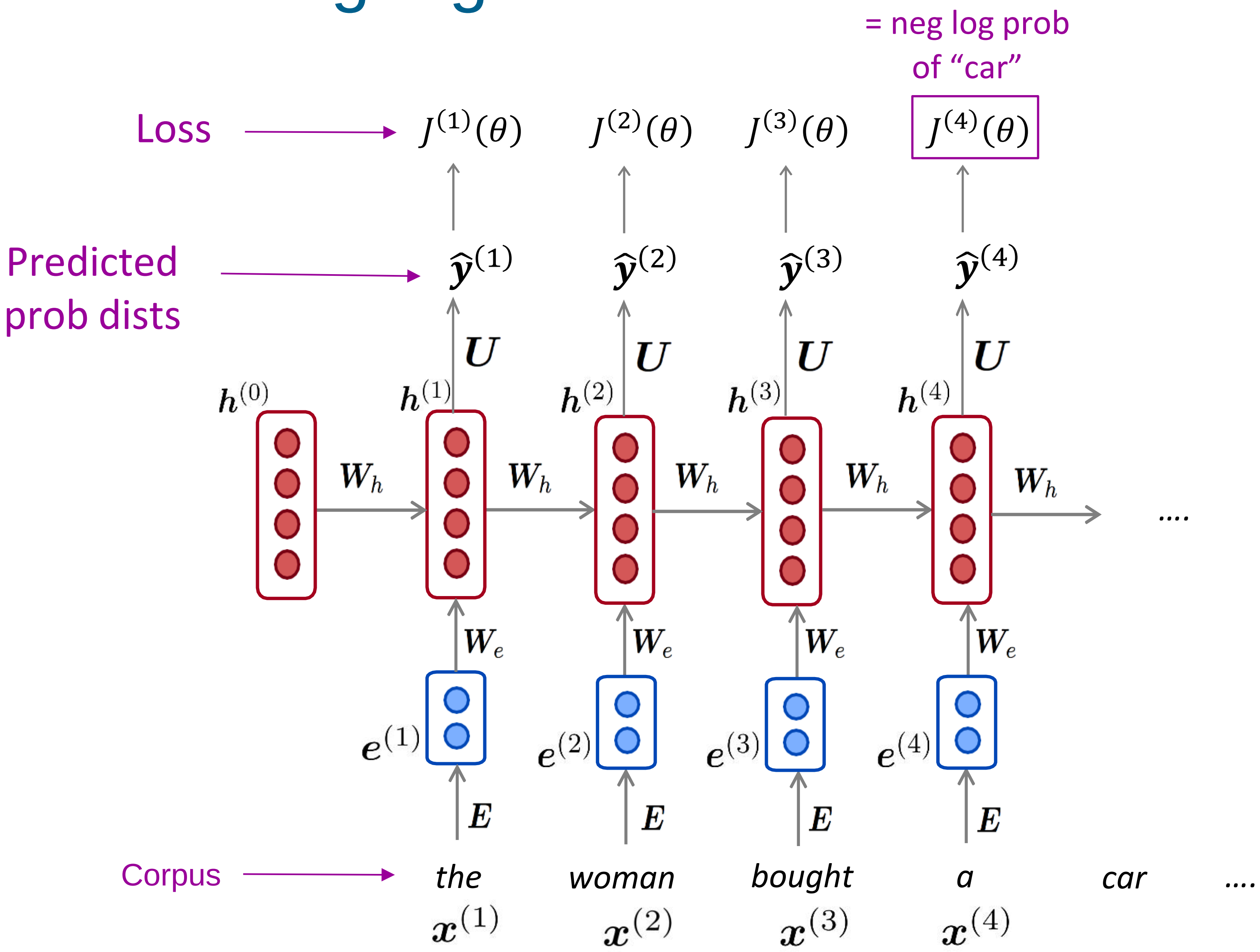
Training a RNN Language Model



Training a RNN Language Model

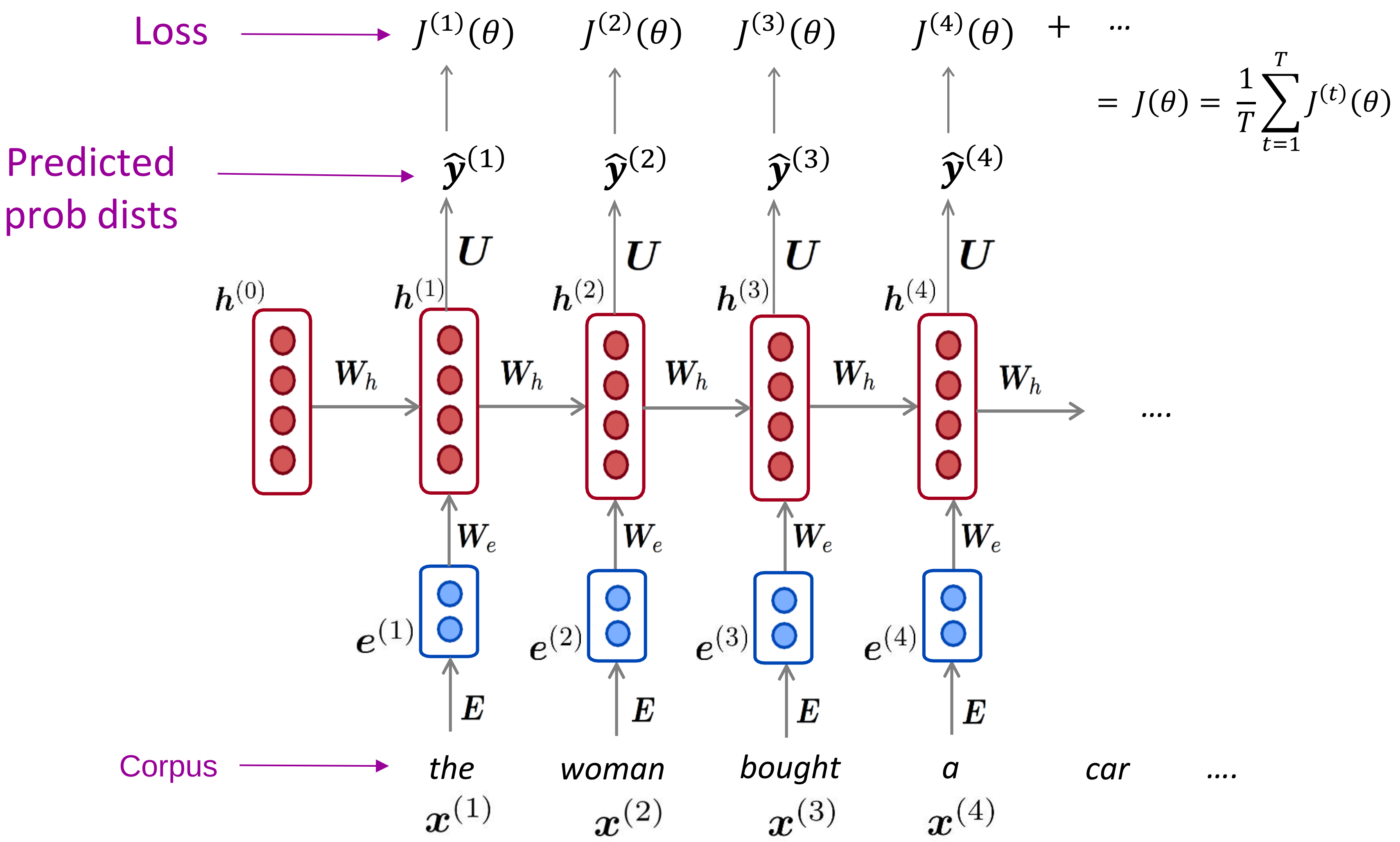


Training a RNN Language Model



RPTU

Training a RNN Language Model



Training a RNN Language Model

- However: Computing loss and gradients across entire corpus is too expensive!

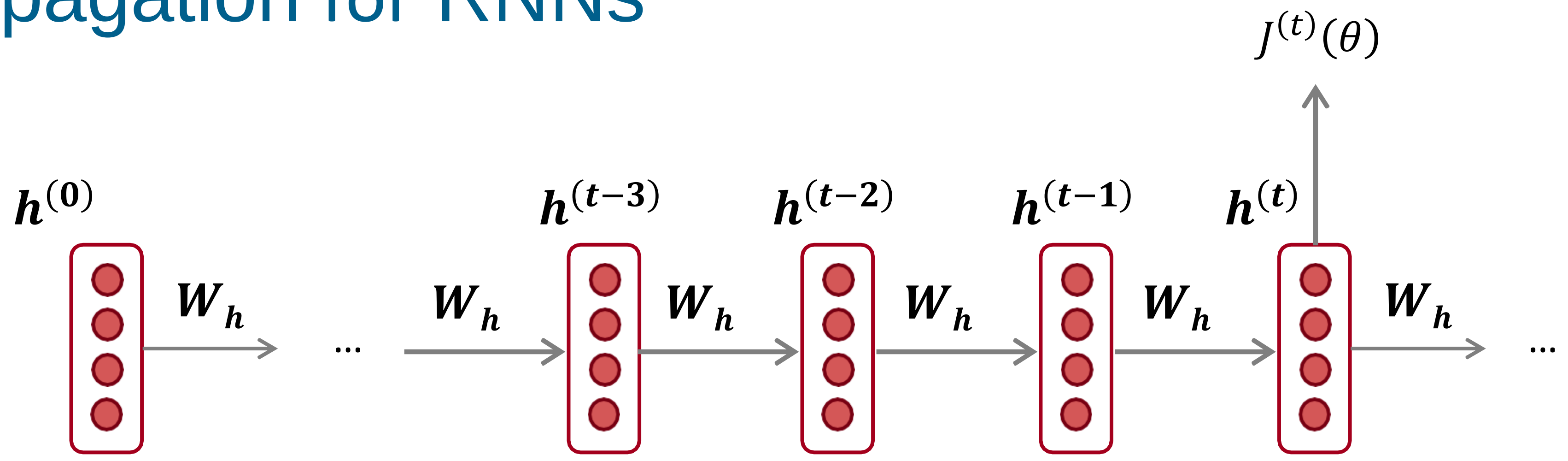
$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta)$$

- Stochastic Gradient Descent allows us to compute loss and gradients for small chunk of data, and update.

- In practice, consider $x^{(1)}, \ldots, x^{(T)}$ as a sentence (or a document)

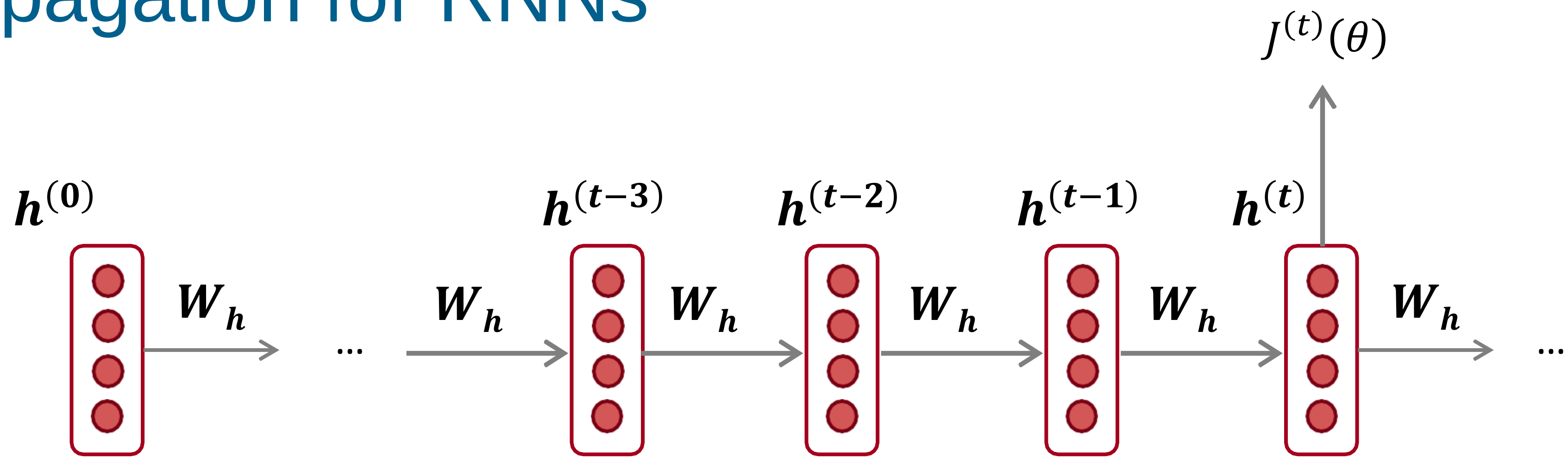
- Compute loss $J(\theta)$ for a sentence (actually usually a batch of sentences), compute gradients and update weights. Repeat. (Stochastic Gradient Descent)

Backpropagation for RNNs



- <u>**Question:**</u> What's the derivative of $J^{(t)}(\theta)$ w.r.t. the repeated weight matrix $\boldsymbol{W_h}$?

Backpropagation for RNNs



- **Question:** What's the derivative of $J^{(t)}(\theta)$ w.r.t. the repeated weight matrix $\boldsymbol{W}_{\boldsymbol{h}}$?

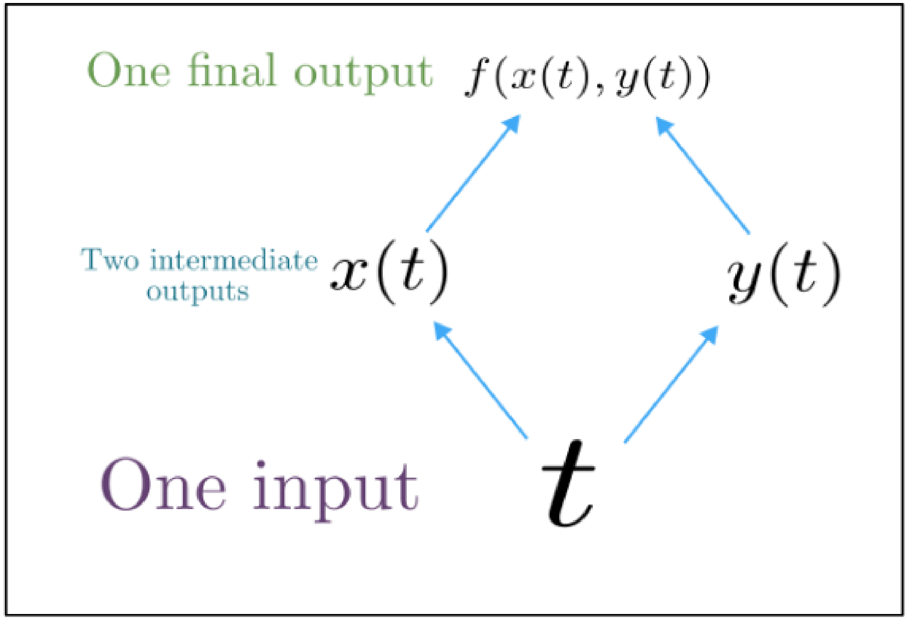
- **Answer:** $\frac{\partial J^{(t)}}{\partial \boldsymbol{W}_{\boldsymbol{h}}} = \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{t}} \frac{\partial J^{(t)}}{\partial \boldsymbol{W}_{\boldsymbol{h}}}\Big|_{(i)}$

"The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears"

Multivariable Chain Rule

- Given a multivariable function $f(x, y)$, and two single variable functions $x(t)$ and $y(t)$, here's what the multivariable chain rule says:

$$\underbrace{\frac{d}{dt} f(x(t), y(t))}_{\text{Derivative of composition function}} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$



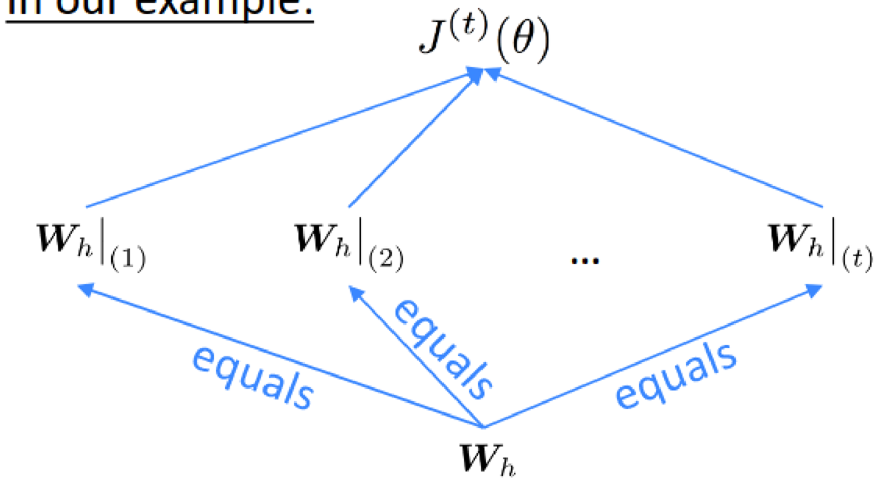
Source:
https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/differentiating-vector-valued-functions/a/multivariable-chain-rule-simple-version

Backpropagation for RNNs: Proof sketch

- Given a multivariable function $f(x, y)$, and two single variable functions $x(t)$ and $y(t)$, here's what the multivariable chain rule says:

$$\underbrace{\frac{d}{dt} f(x(t), y(t))}_{\text{Derivative of composition function}} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

In our example:



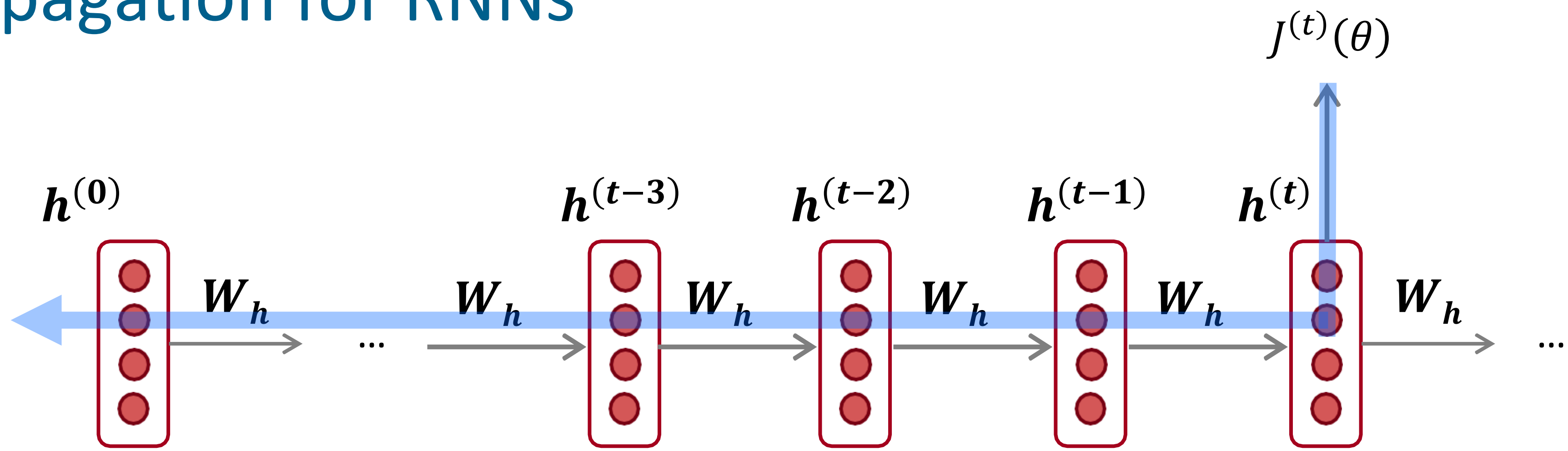
Apply the multivariable chain rule:

$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h}\bigg|_{(i)} \underbrace{\boxed{\frac{\partial \boldsymbol{W}_h\big|_{(i)}}{\partial \boldsymbol{W}_h}}}_{= 1}$$

$$= \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h}\bigg|_{(i)}$$

Source:
https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/differentiating-vector-valued-functions/a/multivariable-chain-rule-simple-version

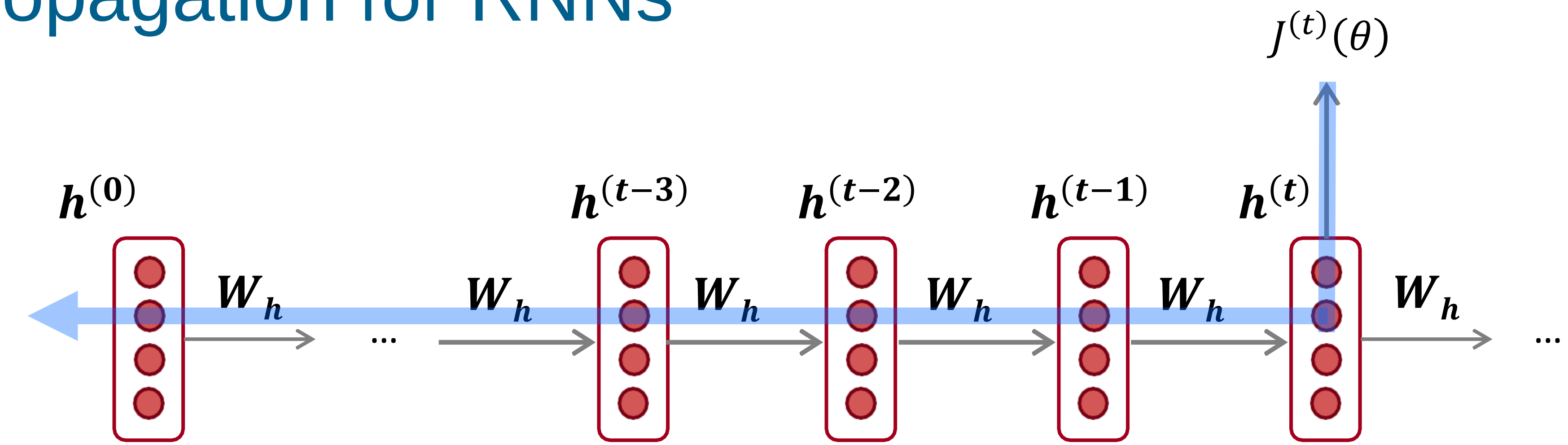
Backpropagation for RNNs



$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \boxed{\sum_{i=1}^{t} \left. \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} \right|_{(i)}}$$

Question: How do we calculate this?

Backpropagation for RNNs



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

Question: How do we calculate this?

Answer: Backpropagate over timesteps $i = t, \ldots, 0$, summing gradients as you go.
This algorithm is called **"backpropagation through time"**

Generating text with a RNN Language Model

- Let's have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on Obama speeches:

SEED: Jobs
"Good afternoon. God bless you.

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done. …

Thank you very much. God bless you, and God bless the United States of America."

Source: https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0

Generating text with a RNN Language Model

- Let’s have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on Harry Potter:

“I’m afraid I’ve definitely been suspended from power, no chance — indeed?” said Snape. He put his head back behind them and read groups as they crossed a corner and fluttered down onto their ink lamp, and picked up his spoon. The doorbell rang. It was a lot cleaner down in London.

Hermione yelled. The party must be thrown by Krum, of course.

Harry collected fingers once more, with Malfoy. “Why, didn’t she never tell me. … ” She vanished. And then, Ron, Harry noticed, was nearly right.

Source: https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

Generating text with a RNN Language Model

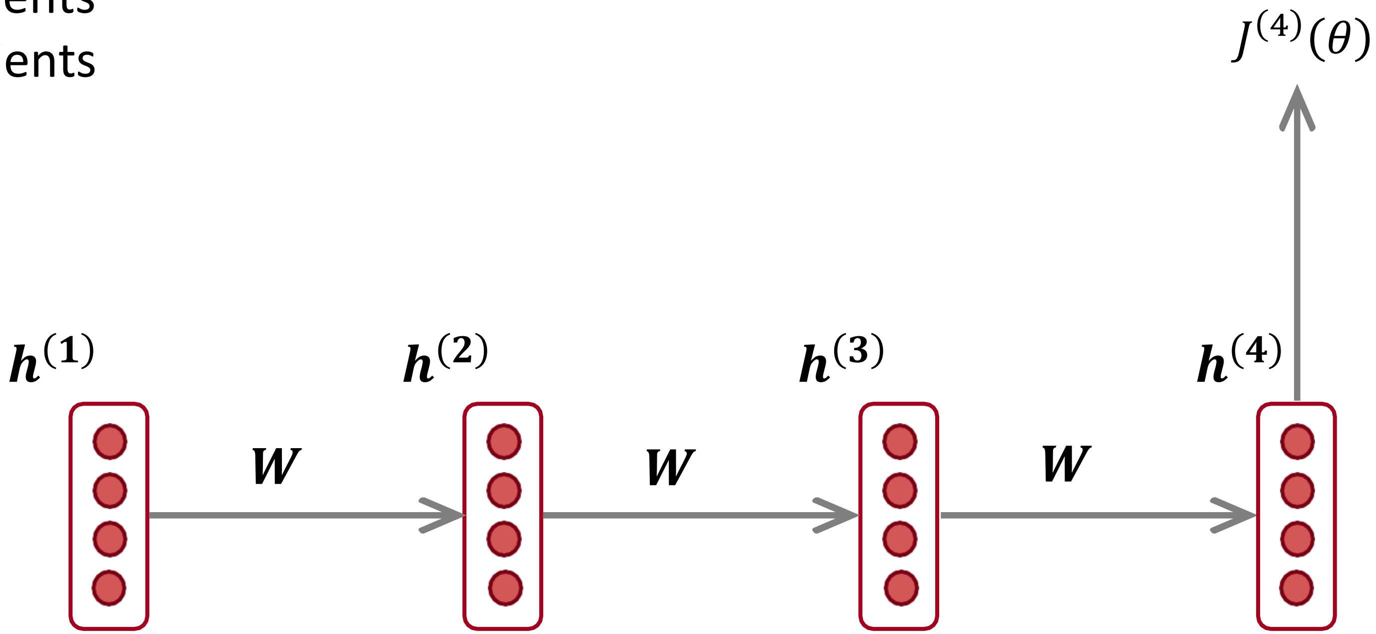
- Let's have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on recipes:

```
     Title: CHOCOLATE RANCH BARBECUE
Categories: Game, Casseroles, Cookies, Cookies
     Yield: 6 Servings

     2 tb Parmesan cheese -- chopped
     1 c  Coconut milk
     3    Eggs, beaten

Place each pasta over layers of lumps. Shape mixture into the moderate oven and simmer
until firm. Serve hot in bodied fresh, mustard, orange and cheese.

Combine the cheese and salt together the dough in a large skillet; add the ingredients
and stir in the chocolate and pepper.
```

https://gist.github.com/nylki/1efbaa36635956d35bcc

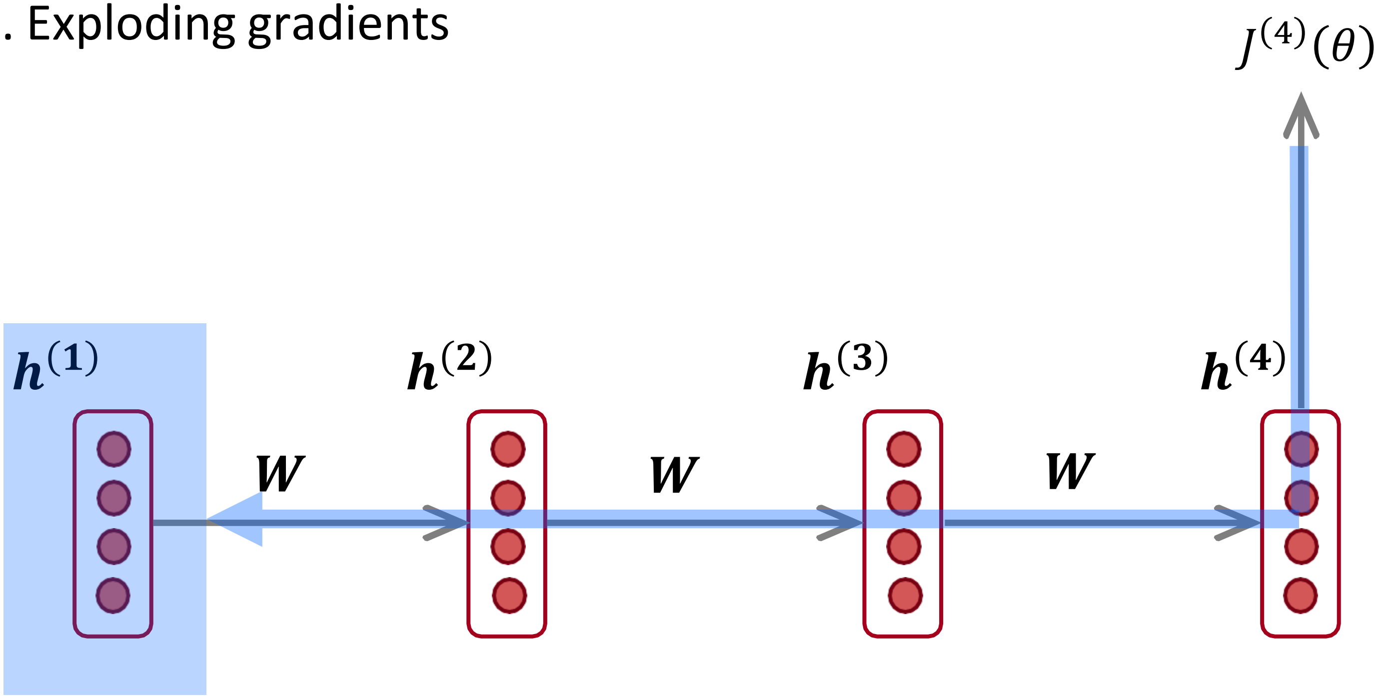
Problems with RNNs

- 1. Vanishing gradients
- 2. Exploding gradients



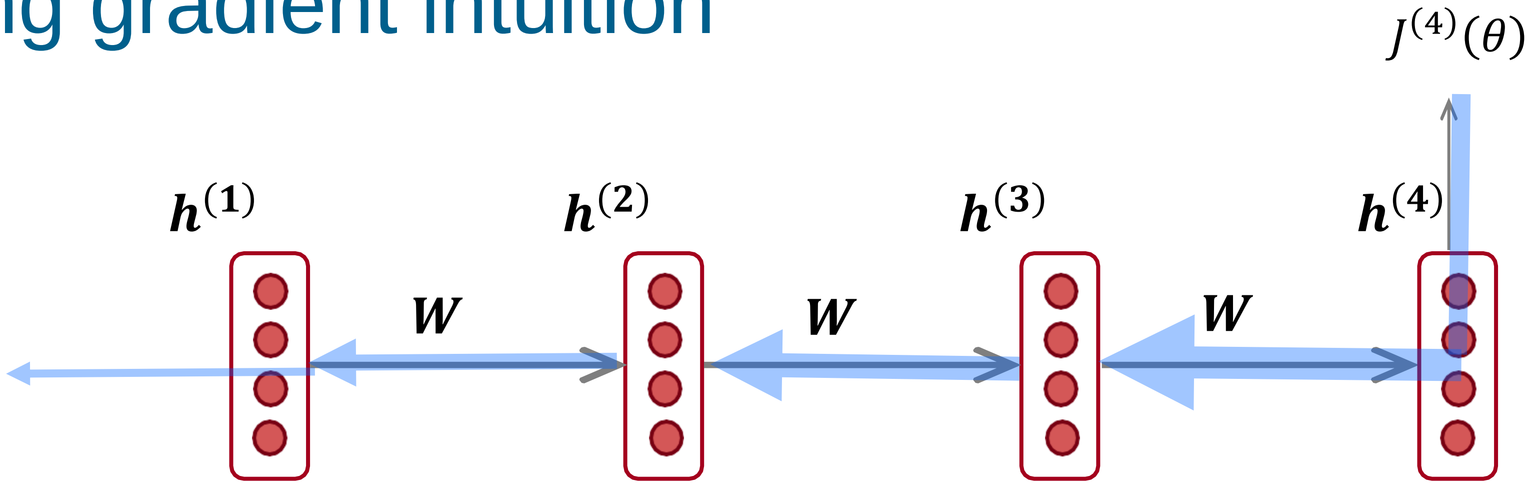
Problems with RNNs

1. Vanishing gradients
2. Exploding gradients



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \ ?$$

Vanishing gradient intuition

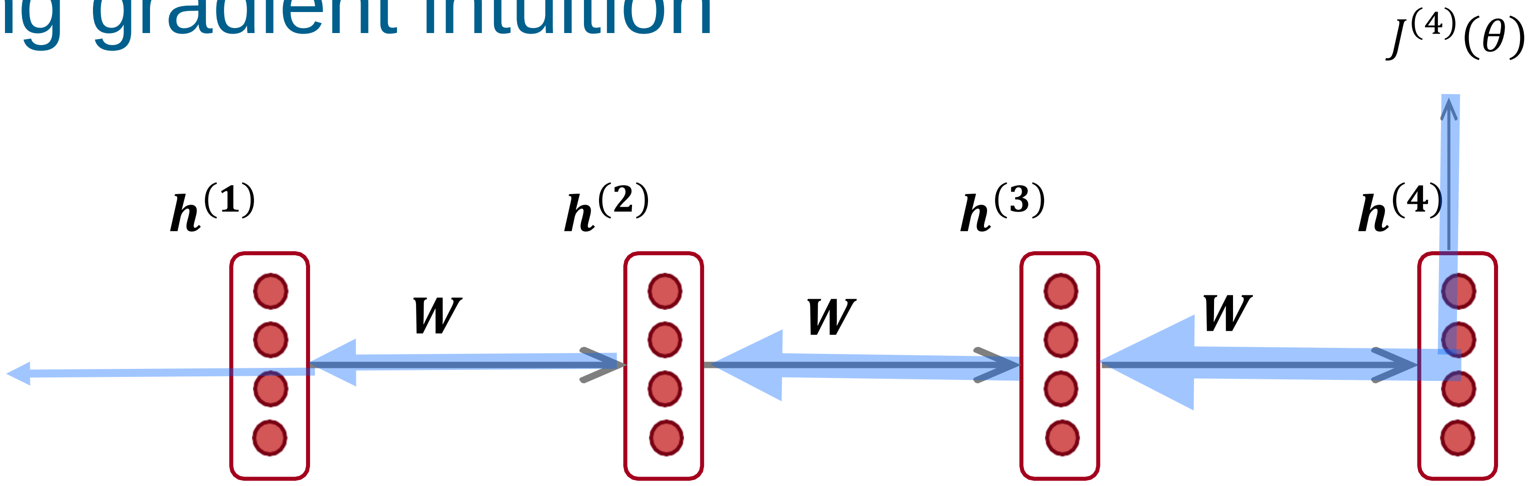


$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \boxed{\frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

What happens if
these are small?

RPTU

Vanishing gradient intuition

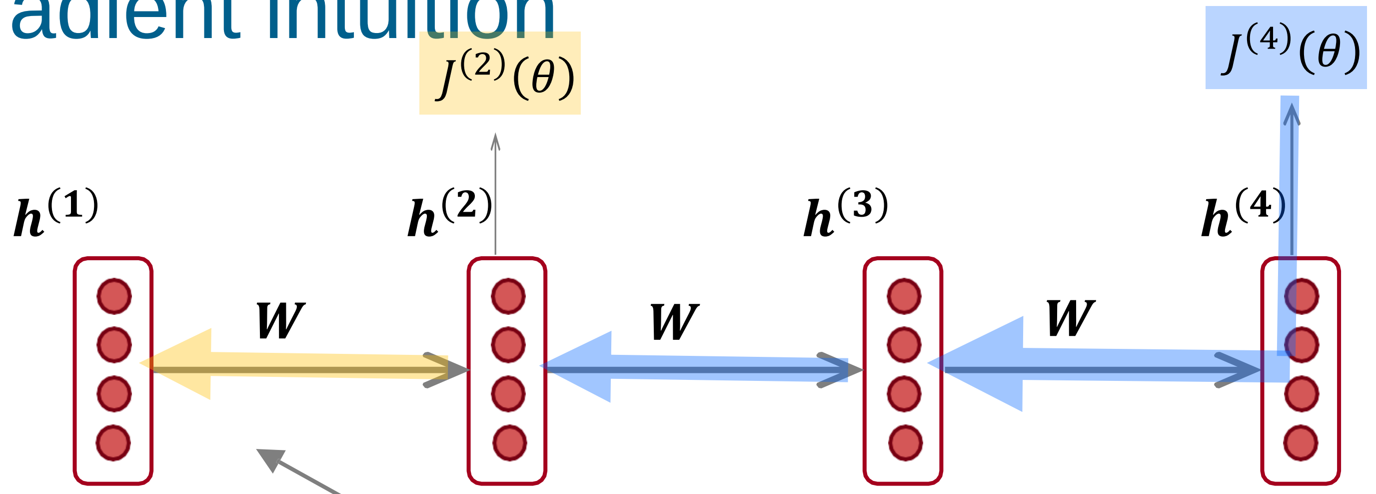


$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \boxed{\frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

What happens if these are small?

Vanishing gradient problem: When these are small, the gradient signal gets smaller and smaller as it backpropagates further

Vanishing gradient intuition



Gradient signal from far away is lost because it's much smaller than gradient signal from close-by.

So, model weights are updated only with respect to near effects, not long-term effects.

Effect of vanishing gradient on RNN-LM

LM task: *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her* ____

- To learn from this training example, the RNN-LM needs to model the dependency between *"tickets"* on the 7^{th} step and the target word *"tickets"* at the end.

- But if gradient is small, the model can't learn this dependency
  - So, the model is unable to predict similar long-distance dependencies at test time
- Other example: „The writer of the books _", Possible answers: is/are

Why is exploding gradient a problem?

- If the gradient becomes too big, then the SGD update step becomes too big:

$$\theta^{new} = \theta^{old} - \overbrace{\alpha}^{\text{learning rate}} \underbrace{\nabla_\theta J(\theta)}_{\text{gradient}}$$

- This can cause bad updates: we take too large a step and reach a weird and bad parameter configuration (with large loss)

- Worst case, this will result in Inf or NaN in your network (then you will have to restart training from an earlier checkpoint)

Gradient clipping

- A solution for exploding gradient!

- Gradient clipping: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

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 a step in the same direction, but a smaller step

- In practice, remembering to clip gradients is important, but exploding gradients are an easy problem to solve

Source: “On the difficulty of training recurrent neural networks”, Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf

How to fix the vanishing gradient problem?

- The main problem is that *it's too difficult for the RNN to learn to preserve information over many timesteps*.

- In a vanilla RNN, the hidden state is constantly being rewritten

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}\right)$$

- How about a RNN with separate memory?

Long Short-Term Memory RNNs (LSTMs)

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing gradients problem.

- On step t, there is a hidden state $h^{(t)}$ *and* a cell state $c^{(t)}$

  - Both are vectors length n
  - The cell stores long-term information
  - The LSTM can read, erase, and write information from the cell
    - The cell becomes conceptually rather like RAM in a computer

"Long short-term memory", Hochreiter and Schmidhuber, 1997. https://www.bioinf.jku.at/publications/older/2604.pdf "Learning to Forget: Continual Prediction with LSTM", Gers, Schmidhuber, and Cummins, 2000. https://dl.acm.org/doi/10.1162/089976600300015015

Long Short-Term Memory RNNs (LSTMs)

- The selection of which information is erased/written/read is controlled by three corresponding gates
  - The gates are also vectors of length n
  - On each timestep, each element of the gates can be open (1), closed (0), or somewhere in-between
  - The gates are dynamic: their value is computed based on the current context

Long Short-Term Memory RNNs (LSTMs)

We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$ .On timestep t:

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

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

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

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

Long Short-Term Memory RNNs (LSTMs)

We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$ .On timestep t:

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

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

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

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

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

Long Short-Term Memory RNNs (LSTMs)

We have a sequence of inputs $x^{(\mathrm{t})}$, and we will compute a sequence of hidden states $h^{(\mathrm{t})}$ and cell states $c^{(t)}$ .On timestep t:

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

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

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

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

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

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

Long Short-Term Memory RNNs (LSTMs)

We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$ .On timestep t:

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

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

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

Sigmoid function: all gate values are between 0 and 1

$$
\left.
\begin{aligned}
\boldsymbol{f}^{(t)} &= \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right) \\
\boldsymbol{i}^{(t)} &= \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right) \\
\boldsymbol{o}^{(t)} &= \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right) \\
\tilde{\boldsymbol{c}}^{(t)} &= \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right) \\
\boldsymbol{c}^{(t)} &= \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)} \\
\boldsymbol{h}^{(t)} &= \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}
\end{aligned}
\right\} \text{All these are vectors of same length } n
$$

Long Short-Term Memory RNNs (LSTMs)

New cell content: this is the new content to be written to the cell

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

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

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

Long Short-Term Memory RNNs (LSTMs)

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

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

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

Long Short-Term Memory RNNs (LSTMs)

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

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

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

Long Short-Term Memory RNNs (LSTMs)

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

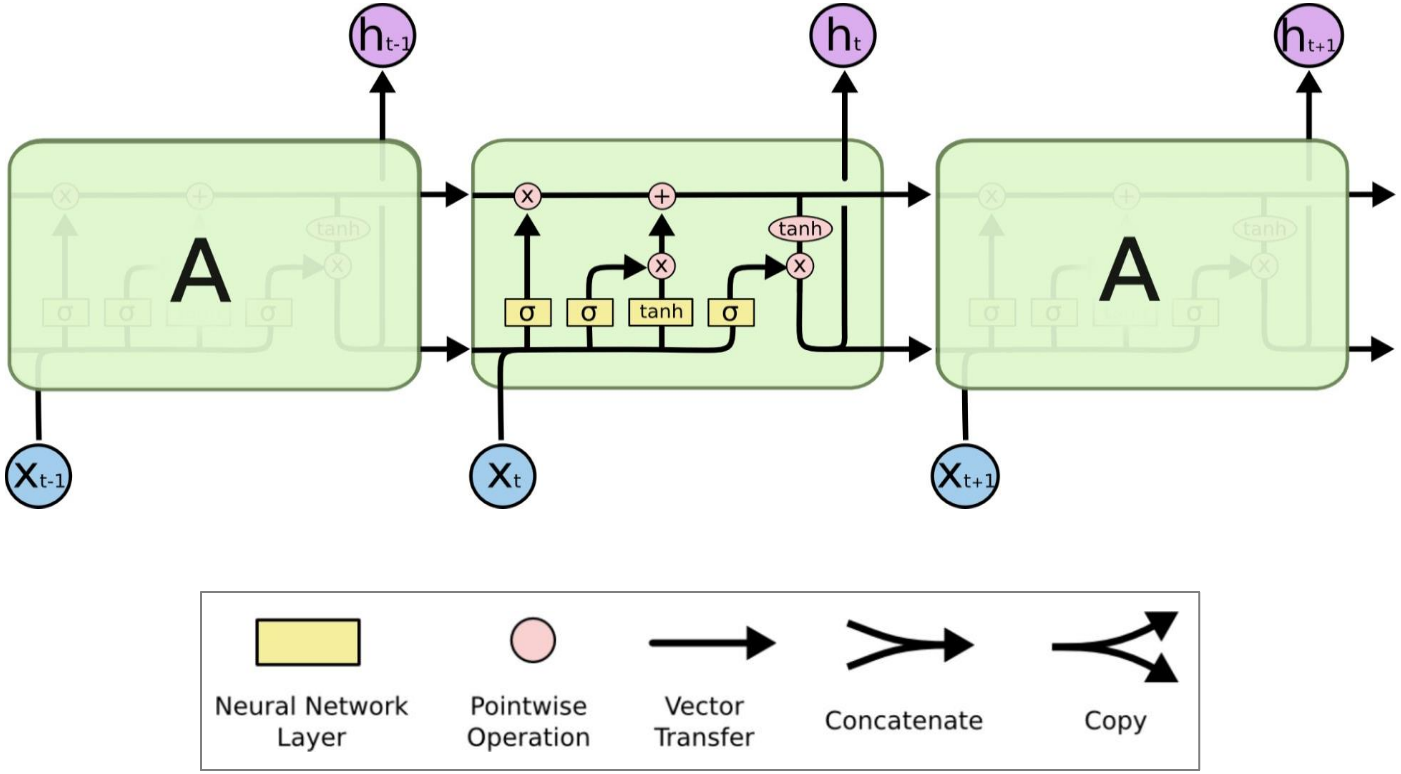
$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

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

Gates are applied using element-wise (or Hadamard) product: $\odot$

Long Short-Term Memory RNNs (LSTMs)

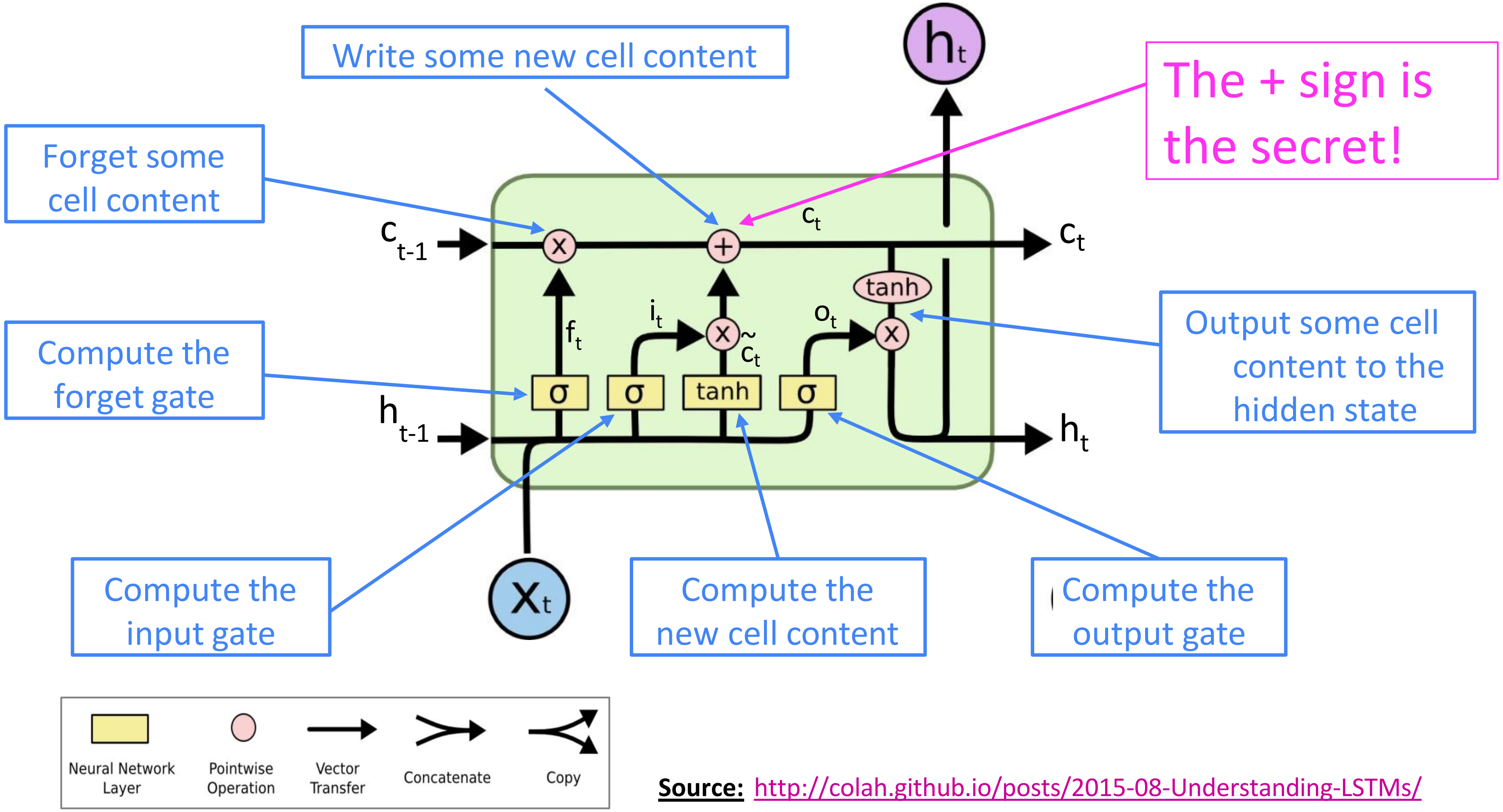
You can think of the LSTM equations visually like this:



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

Long Short-Term Memory RNNs (LSTMs)

You can think of the LSTM equations visually like this:



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

How does LSTM solve vanishing gradients?

- The LSTM architecture makes it easier for the RNN to preserve information over many timesteps
  - e.g., if the forget gate is set to 1 for a cell dimension and the input gate set to 0, then the information of that cell is preserved indefinitely.
  - In contrast, it's harder for a vanilla RNN to learn a recurrent weight matrix W_h that preserves info in the hidden state
  - In practice, you get about 100 timesteps rather than about 7
- LSTM doesn't *guarantee* that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

LSTMs: real-world success

- In 2013–2015, LSTMs started achieving state-of-the-art results
  - Successful tasks include handwriting recognition, speech recognition, machine translation, parsing, and image captioning, as well as language models
  - LSTMs became the dominant approach for most NLP tasks
- Since ca. 2017, other approaches (e.g., Transformers) have become dominant for many tasks
  - For example, in WMT (a Machine Translation conference + competition):
  - In WMT 2016, the summary report contains “RNN” 44 times
  - In WMT 2019: “RNN” 7 times, ”Transformer” 105 times

Source: "Findings of the 2016 Conference on Machine Translation (WMT16)", Bojar et al. 2016, http://www.statmt.org/wmt16/pdf/W16-2301.pdf **Source:** "Findings of the 2018 Conference on Machine Translation (WMT18)", Bojar et al. 2018, http://www.statmt.org/wmt18/pdf/WMT028.pdf **Source:** "Findings of the 2019Conference on Machine Translation (WMT19)", Barrault et al. 2019, http://www.statmt.org/wmt18/pdf/WMT028.pdf

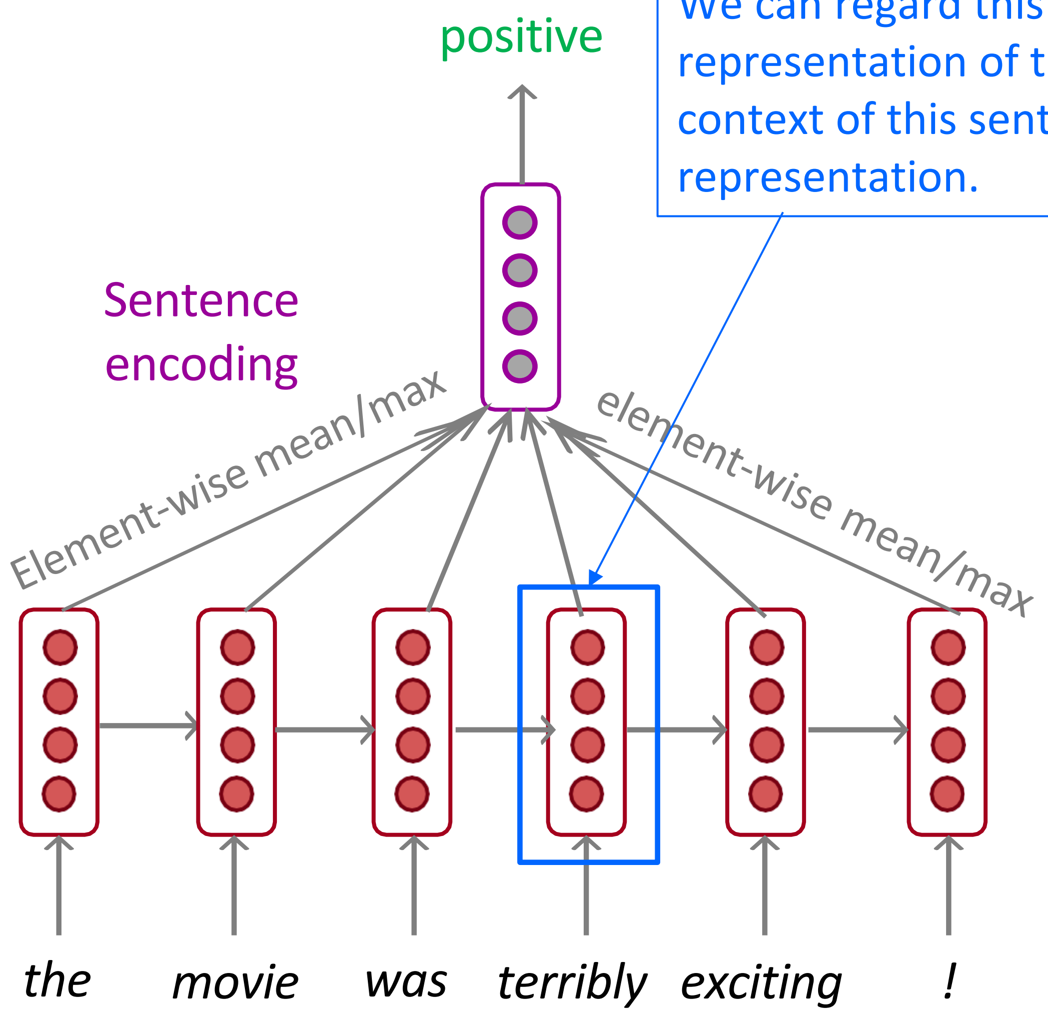
Is vanishing/exploding gradient just an RNN problem?

- No! It can be a problem for all neural architectures (including feed-forward and convolutional), especially very deep ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus, lower layers are learned very slowly (hard to train)
- Solution: lots of new deep feedforward/convolutional architectures that add more direct connections (thus allowing the gradient to flow)
  - Called residual connections or skip-connections

- Conclusion: Though vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to the repeated multiplication by the same weight matrix [Bengio et al, 1994]

”Learning Long-Term Dependencies with Gradient Descent is Difficult", Bengio et al. 1994, http://ai.dinfo.unifi.it/paolo//ps/tnn-94-gradient.pdf

Bidirectional and Multi-layer RNNs: motivation



We can regard this hidden state as a representation of the word "terribly" in the context of this sentence. We call this a contextual representation.

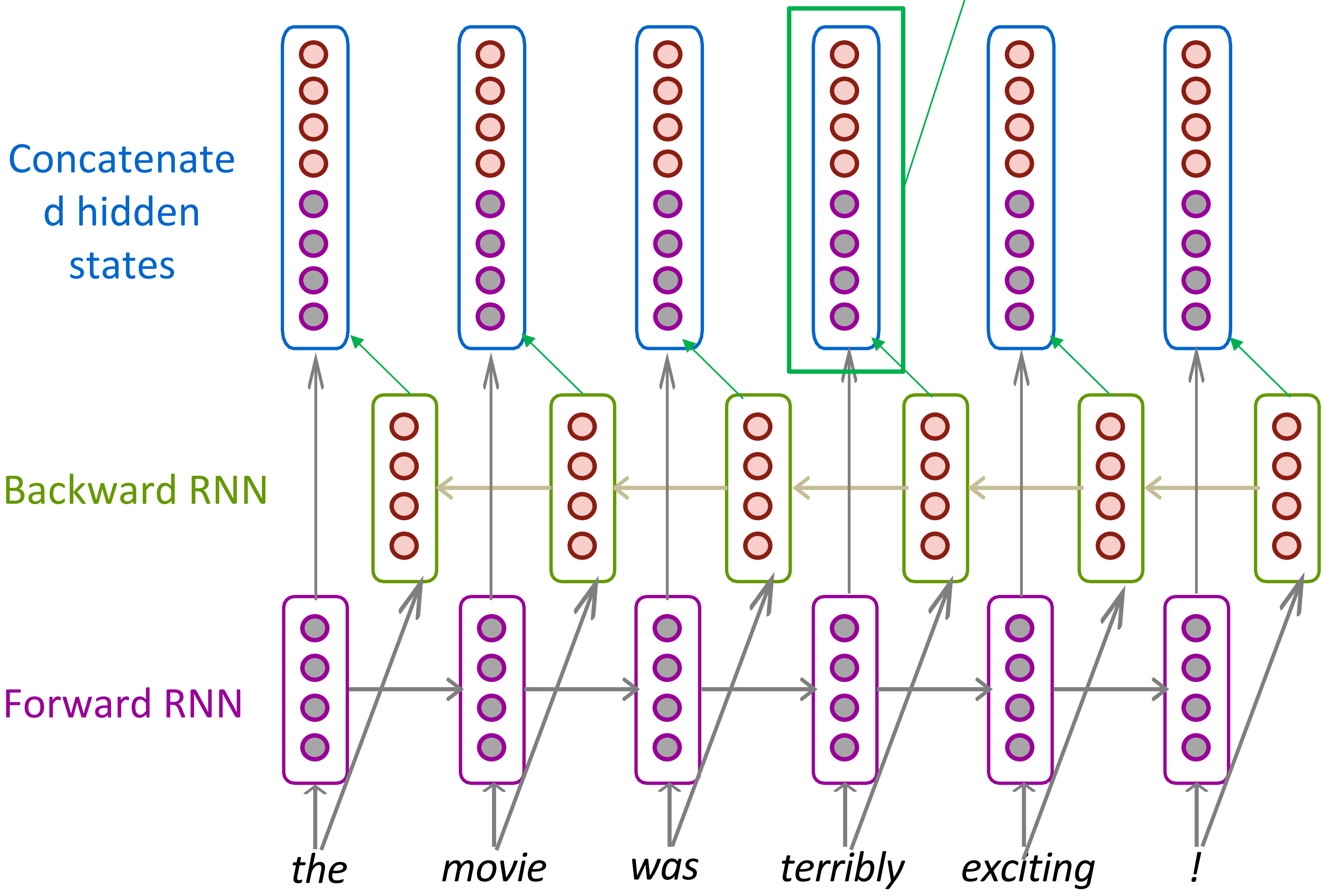
These contextual representations only contain information about the left context (e.g.) "the movie was".

What about right context?

In this example "exciting" is in the right context and this modifies the meaning of "terribly" (from negative to positive).

Bidirectional and Multi-layer RNNs: motivation

This contextual representation of 'terribly' has both left and right context



Bidirectional RNNs

- On timestep t:

- Forward RNN $\overrightarrow{h^{(t)}} = RNN_{FW}(\overrightarrow{h^{(t-1)}}, x^{(t)})$
- Backward RNN $\overleftarrow{h^{(t)}} = RNN_{BW}(\overleftarrow{h^{(t+1)}}, x^{(t)})$

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

RPTU

Bidirectional RNNs

- Note: bidirectional RNNs are only applicable if you have access to the entire input sequence
  - They are not applicable to Language Modeling, because in LM you only have left context available.

- If you do have entire input sequence (e.g., any kind of encoding), bidirectionality is powerful (you should use it by default).

- For example, BERT (Bidirectional Encoder Representations from Transformers) is a powerful pretrained contextual representation system built on bidirectionality.
  - You will learn more about transformers including BERT in a couple of weeks!

Multi-layer RNNs

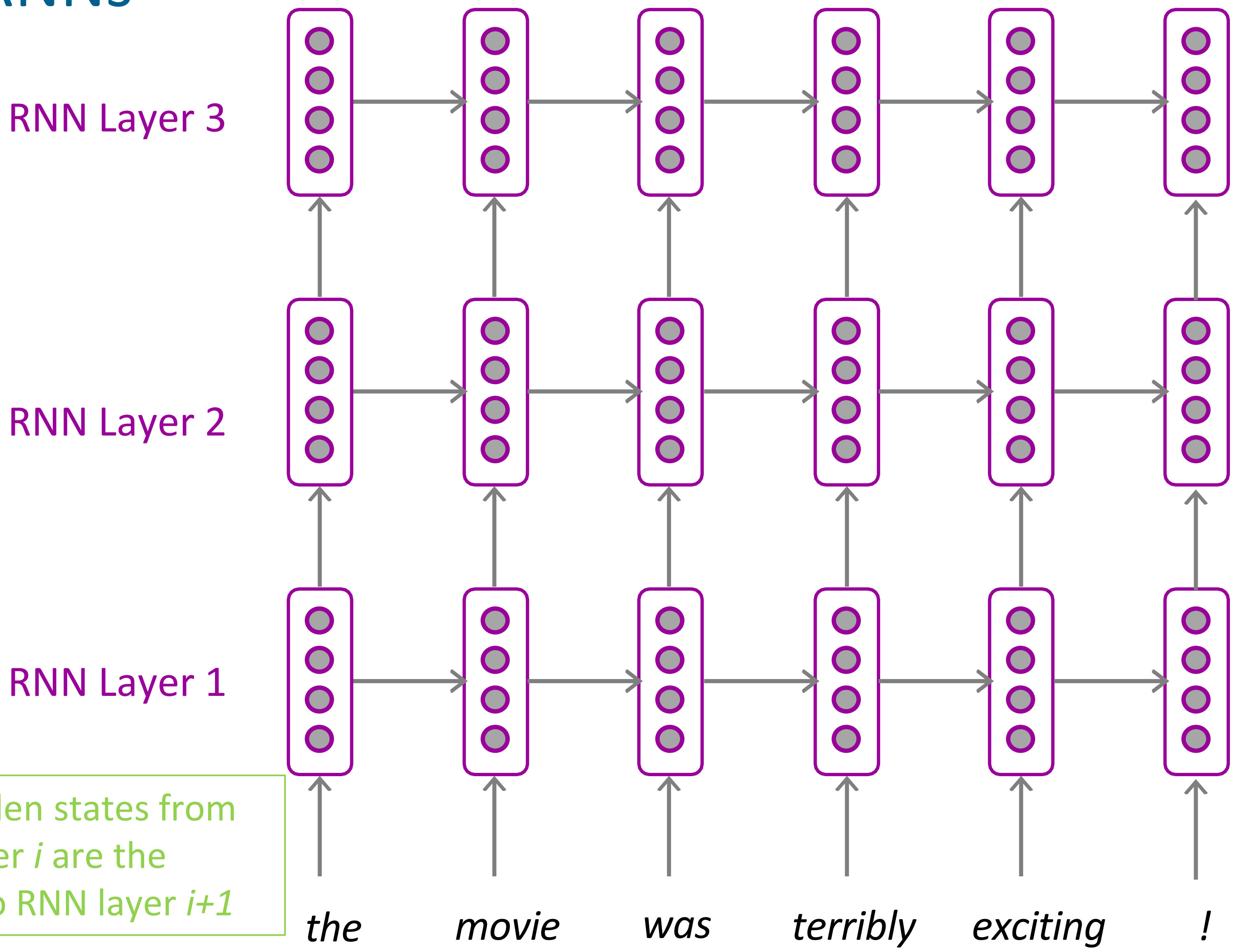
- RNNs are already “deep” on one dimension (they unroll over many timesteps)

- 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.*

Multi-layer RNNs



The hidden states from RNN layer *i* are the inputs to RNN layer *i+1*

Multi-layer RNNs in practice

- High-performing RNNs are often multi-layer (but aren't as deep as convolutional or feed-forward networks)

- For example: In a 2017 paper, Britz et al find that for Neural Machine Translation, 2 to 4 layers is best for the encoder RNN, and 4 layers is best for the decoder RNN
  - Usually, skip-connections/dense-connections are needed to train deeper RNNs (e.g., 8 layers)

- Transformer-based networks (e.g., BERT) are usually deeper, like 12 or 24 layers.

"Massive Exploration of Neural Machine Translation Architecutres", Britz et al, 2017. https://arxiv.org/pdf/1703.03906.pdf

What Problems are Handled?

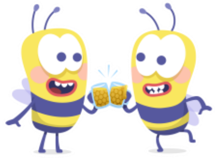
- Cannot share strength among similar words

she bought a car	she bought a bicycle
she purchased a car	she purchased a bicycle

- solved, and similar contexts as well!

- Cannot condition on context with intervening words

Dr. Jane Smith	Dr. Gertrude Smith



- Solved!

Problems and solutions?

- Cannot handle long-distance dependencies

For tennis class he wanted to buy his own raquet
for programming class he wanted to buy his own computer

- Solved!

CNNs

A 1D convolution for text

Tentative	0.2	0.1	-0.3	0.4
Deal	0.5	0.2	-0.3	-0.1
Reached	-0.1	-0.3	-0.2	0.4
To	0.3	-0.3	0.1	0.1
Keep	0.2	-0.3	0.4	0.2
Government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3

T,d,r	-1.0
D,r,t	-0.5
R,t,k	-3.6
T,k,g	-0.2
K,g,o	0.3

Filter/kernel

3	1	2	-3
-1	2	1	-3
1	1	-1	1

Padding

Ø	0.0	0.0	0.0	0.0
Tentative	0.2	0.1	-0.3	0.4
Deal	0.5	0.2	-0.3	-0.1
Reached	-0.1	-0.3	-0.2	0.4
To	0.3	-0.3	0.1	0.1
Keep	0.2	-0.3	0.4	0.2
Government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

Filter/kernel

3	1	2	-3
-1	2	1	-3
1	1	-1	1

Ø,t,d	-0.6
T,d,r	-1.0
D,r,t	-0.5
R,t,k	-3.6
T,k,g	-0.2
K,g,o	0.3
G,o, Ø	-0.5

Multiple filters

Ø	0.0	0.0	0.0	0.0
Tentative	0.2	0.1	-0.3	0.4
Deal	0.5	0.2	-0.3	-0.1
Reached	-0.1	-0.3	-0.2	0.4
To	0.3	-0.3	0.1	0.1
Keep	0.2	-0.3	0.4	0.2
Government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

Ø,t,d	-0.6	…	
T,d,r	-1.0	…	
D,r,t	-0.5	…	
R,t,k	-3.6		
T,k,g	-0.2		
K,g,o	0.3		
G,o, Ø	-0.5		

3 filters

3	1	2	-3
-1	2	1	-3
1	1	-1	1

5	1	0.2	-13
-4	-2	1	4
-1	3	-1	1

1.3	1.1	2	-3
-10	-2	4.1	-3
2.1	-1	-1	1

Max pooling

Ø	0.0	0.0	0.0	0.0
Tentative	0.2	0.1	-0.3	0.4
Deal	0.5	0.2	-0.3	-0.1
Reached	-0.1	-0.3	-0.2	0.4
To	0.3	-0.3	0.1	0.1
Keep	0.2	-0.3	0.4	0.2
Government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

Channels/features

Ø,t,d	-0.6	…	
T,d,r	-1.0	…	
D,r,t	-0.5	…	
R,t,k	-3.6		
T,k,g	-0.2		
K,g,o	0.3		
G,o, Ø	-0.5		

Max pool	0.3	1.6	1.4

3 filters

3	1	2	-3
-1	2	1	-3
1	1	-1	1

5	1	0.2	-13
-4	-2	1	4
-1	3	-1	1

1.3	1.1	2	-3
-10	-2	4.1	-3
2.1	-1	-1	1

Average pooling

Ø	0.0	0.0	0.0	0.0
Tentative	0.2	0.1	-0.3	0.4
Deal	0.5	0.2	-0.3	-0.1
Reached	-0.1	-0.3	-0.2	0.4
To	0.3	-0.3	0.1	0.1
Keep	0.2	-0.3	0.4	0.2
Government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

Channels/features

Ø,t,d	-0.6	…	
T,d,r	-1.0	…	
D,r,t	-0.5	…	
R,t,k	-3.6		
T,k,g	-0.2		
K,g,o	0.3		
G,o, Ø	-0.5		

Average pool	-0.87	0.26	0.53

3 filters

3	1	2	-3
-1	2	1	-3
1	1	-1	1

5	1	0.2	-13
-4	-2	1	4
-1	3	-1	1

1.3	1.1	2	-3
-10	-2	4.1	-3
2.1	-1	-1	1

Stride=2

Ø	0.0	0.0	0.0	0.0
Tentative	0.2	0.1	-0.3	0.4
Deal	0.5	0.2	-0.3	-0.1
Reached	-0.1	-0.3	-0.2	0.4
To	0.3	-0.3	0.1	0.1
Keep	0.2	-0.3	0.4	0.2
Government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

Ø,t,d	-0.6	…	
D,r,t	-0.5	…	
T,k,g	-0.2		
G,o, Ø	-0.5		

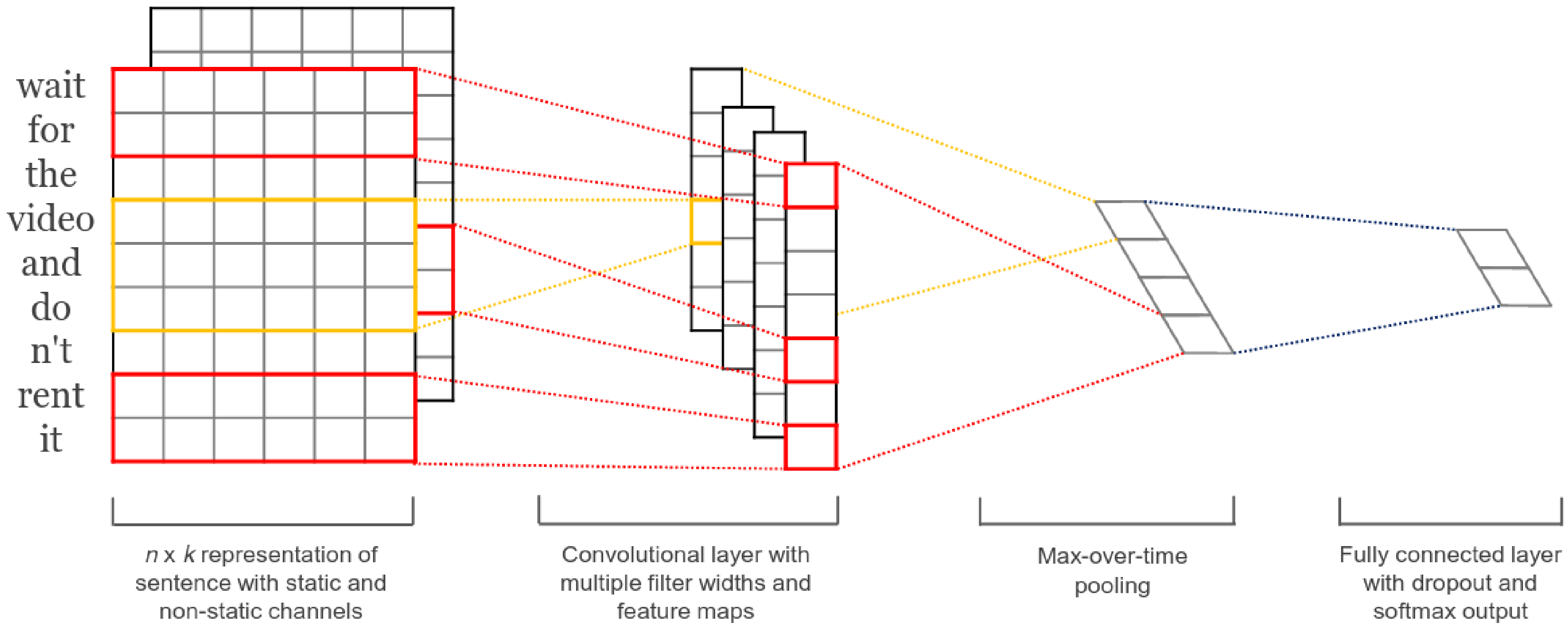
3 filters

3	1	2	-3
-1	2	1	-3
1	1	-1	1

5	1	0.2	-13
-4	-2	1	4
-1	3	-1	1

1.3	1.1	2	-3
-10	-2	4.1	-3
2.1	-1	-1	1

Kim (2014)



n words (possibly zero padded) and each word vector has k dimensions

Summary

- RNNs are capable of learning long sequences and prove to be useful for language modeling.
- RNNs suffer from vanishing/exploding gradients that affects learning.
- LSTMs are gated NNs with memory that solve vanishing/exploding gradients problem.
- Using CNNs, multiple feature maps of same word phrases can be obtained.

Next lecture
Attention and Transformers

References

- “Long short-term memory”, Hochreiter and Schmidhuber, 1997.

Acknowledgements

- Asmita Bhat
- Stanford CS224N, Lecture 6 and 7