CS310 Natural Language Processing
自然语言处理
Lecture 03 - Recurrent Neural Networks and Language Modeling

Instructor: Yang Xu
主讲人：徐炀
xuyang@sustech.edu.cn

Overview

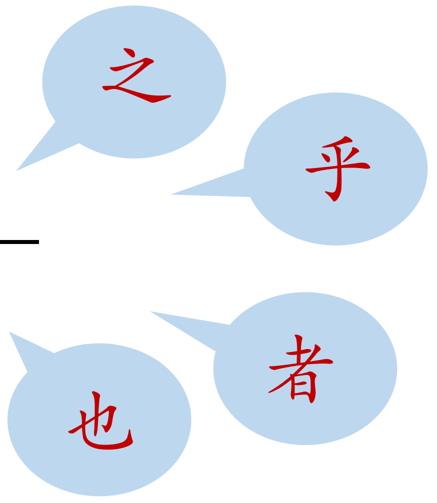
- **Language Modeling**
- Neural Language Models
- Recurrent Neural Networks for LM
- Evaluate LMs
- Long Short-Term Memory RNNs (LSTMs)

Language Modeling

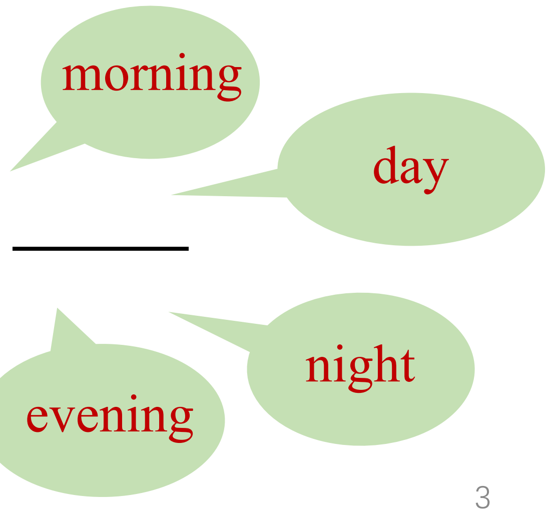
- LM is the task of predicting **what the next word is**, given the preceding ones.
- Formally: given a sequence of words $x^{\langle 1 \rangle}, x^{\langle 2 \rangle}, \ldots, x^{\langle t \rangle}$, compute the probability of $x^{\langle t+1 \rangle}$:

$$P(x^{\langle t+1 \rangle} | x^{\langle 1 \rangle}, \ldots, x^{\langle t \rangle})$$

子曰：学而时习 __

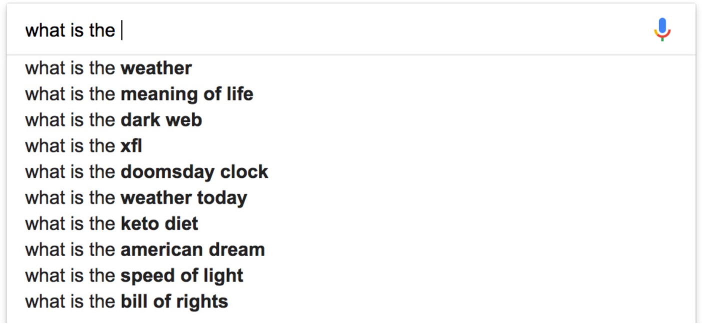


The Sun rises every _____



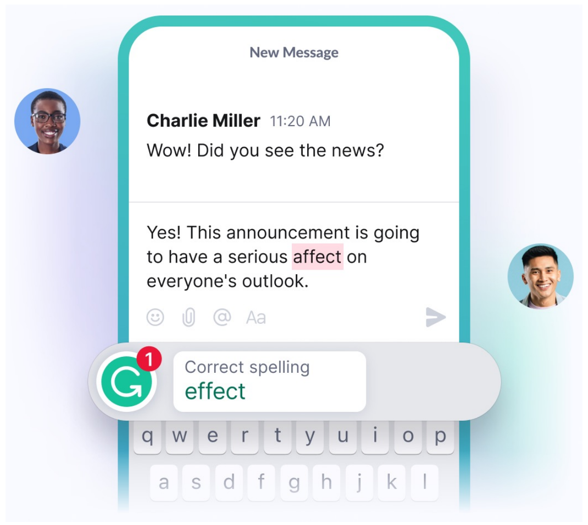
Motivation for LM

- Edit/input suggestion
- Speech recognition
- Grammar correction
- Dialogue system
- Code generation



what is the |

what is the **weather**
what is the **meaning of life**
what is the **dark web**
what is the **xfl**
what is the **doomsday clock**
what is the **weather today**
what is the **keto diet**
what is the **american dream**
what is the **speed of light**
what is the **bill of rights**

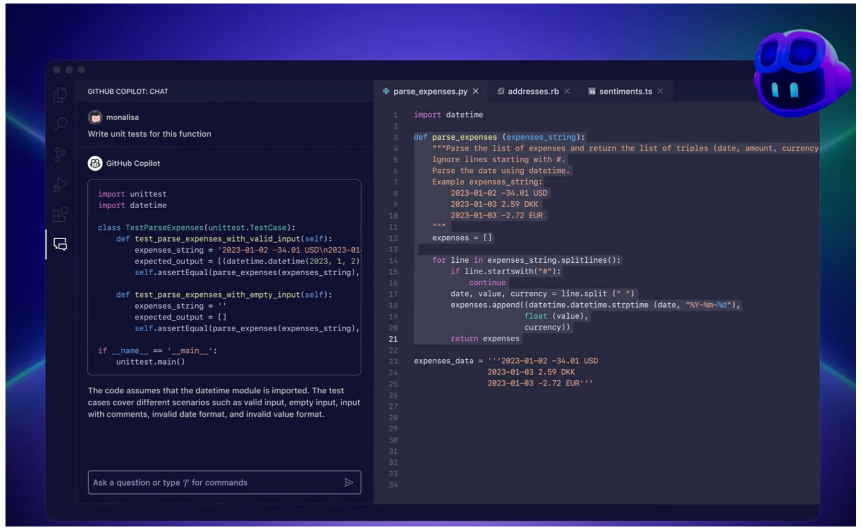


New Message

Charlie Miller 11:20 AM
Wow! Did you see the news?

Yes! This announcement is going to have a serious affect on everyone's outlook.

Correct spelling
effect



GITHUB COPILOT: CHAT

monalisa
Write unit tests for this function

GitHub Copilot

```
import unittest
import datetime

class TestParseExpenses(unittest.TestCase):
    def test_parse_expenses_with_valid_input(self):
        expenses_string = '2023-01-02 -34.01 USD\n2023-01
        expected_output = [(datetime.datetime(2023, 1, 2)
        self.assertEqual(parse_expenses(expenses_string),

    def test_parse_expenses_with_empty_input(self):
        expenses_string = ''
        expected_output = []
        self.assertEqual(parse_expenses(expenses_string),

if __name__ == '__main__':
    unittest.main()
```

The code assumes that the datetime module is imported. The test cases cover different scenarios such as valid input, empty input, input with comments, invalid date format, and invalid value format.

Ask a question or type '/' for commands

parse_expenses.py × addresses.rb × sentiments.ts ×

```
import datetime

def parse_expenses (expenses_string):
    """Parse the list of expenses and return the list of triples (date, amount, currency
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2023-01-02 -34.01 USD
        2023-01-03 2.59 DKK
        2023-01-03 -2.72 EUR
    """
    expenses = []

    for line in expenses_string.splitlines():
        if line.startswith("#"):
            continue
        date, value, currency = line.split (" ")
        expenses.append((datetime.datetime.strptime (date, "%Y-%m-%d"),
                        float (value),
                        currency))
        return expenses

expenses_data = '''2023-01-02 -34.01 USD
                2023-01-03 2.59 DKK
                2023-01-03 -2.72 EUR'''
```

Review of Probability

Event space (e.g., $\mathcal{X}$, $\mathcal{Y}$)—in this class, usually discrete

Random variables (e.g., X, Y)

Typical statement: "random variable X takes value $x \in \mathcal{X}$ with probability $p(X = x)$, or, in shorthand, $p(x)$"

Joint probability: $p(X = x, Y = y)$

Conditional probability: $p(X = x \mid Y = y)$
$$= \frac{p(X = x, Y = y)}{p(Y = y)}$$

Always true:
$$p(X = x, Y = y) = p(X = x \mid Y = y) \cdot p(Y = y)$$
$$= p(Y = y \mid X = x) \cdot p(X = x)$$

Sometimes true: $p(X = x, Y = y) = p(X = x) \cdot p(Y = y)$

Adapted from: https://nasmith.github.io/NLP-winter23/assets/slides/lm.pdf

How to Learn an LM?

- Pre- Neural network solution: ***n*-gram Language Model**
- **Def.** An *n*-gram is a chunk of *n* consecutive words

```
the Sun rises every ______
```

- **Uni**grams (*n*=1): “the”, “Sun”, “rises”, “every”
- **Bi**grams (*n*=2): “the Sun”, “Sun rises”, “rises every”
- **Tri**grams (*n*=3): “the Sun rises”, “Sun rises every”
- **Four**-grams (*n*=4): “the Sun rises every”

- **Idea**: Count the frequencies of different n-grams and use these to predict the next word

n-grams LM: Markov assumption

Andrey Andreyevich Markov
(14 June 1856 – 20 July 1922)

- **Markov assumption**: a word at only depends on its preceding $n-1$ words

$$P\left(x^{\langle t+1\rangle} \middle| x^{\langle 1\rangle},\ldots,x^{\langle t\rangle}\right) = P\left(x^{\langle t+1\rangle} \middle| \underbrace{x^{\langle t-n+2\rangle},\ldots,x^{\langle t\rangle}}_{n-1 \text{ words}}\right)$$

By the definition of conditional probability

Probability of a *n*-gram

Probability of a (*n-1*)-gram

$$= \frac{P(x^{\langle t-n+2\rangle},\ldots x^{\langle t\rangle},x^{\langle t+1\rangle})}{P(x^{\langle t-n+2\rangle},\ldots x^{\langle t\rangle})}$$

- **Question**: How to obtain the probabilities?
- **Answer**: By counting them from some large enough corpora (statistical approximation)

$$\approx \frac{\text{count}(x^{\langle t-n+2\rangle},\ldots x^{\langle t\rangle},x^{\langle t+1\rangle})}{\text{count}(x^{\langle t-n+2\rangle},\ldots x^{\langle t\rangle})}$$

n-gram LM: Example

- Goal: Learning a **4**-gram LM, i.e., considering 3 preceding words

discard

`~~in this peculiar game,~~ the Sun rises every` $\underline{\quad w \quad}$

$$P(w|\texttt{Sun rises every}) \approx \frac{\text{count}(\texttt{Sun rises every } w)}{\text{count}(\texttt{Sun rises every})}$$

Example, suppose in the corpus

- "`Sun rises every`" occurs **1000** times
- "`Sun rises every morning`" occurs **600** times
  - $\Rightarrow P(\textbf{morning}|\texttt{Sun rises every}) = \mathbf{0.6}$
- "Sun rises every day" occurs **300** times
  - $\Rightarrow P(\textbf{day}|\texttt{Sun rises every}) = \mathbf{0.3}$

Question: What's the problem of this method?

Sparsity Problem with *n*-gram LM

Sparsity problem 1:
What if "`Sun rises every` w" never occurred in data? Then the probability is 0

Partial solution: **Smoothing** => Add small δ to the count for every word in V

$$P(w|\texttt{Sun rises every}) \approx \frac{\text{count}(\texttt{Sun rises every } w)}{\text{count}(\texttt{Sun rises every})}$$

Sparsity problem 2:
What if "`Sun rises every`" never occurred in data? Then the probability is not computable

Partial solution: **Backoff** => Count "`rises every`" instead, i.e., shorter conditional context

Adapted from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Storage Problem with *n*-gram LM

- **Storage**: Need to store count for all *n*-grams in the corpus
- Larger *n* or larger corpus means larger model size

$$P(w|\texttt{Sun rises every}) \approx \frac{\text{count}(\texttt{Sun rises every}\ w)}{\text{count}(\texttt{Sun rises every})}$$

Every term needs be stored

n-gram LM in Practice

- Implementations on Github: https://github.com/kpu/kenlm

You can build a simple trigram Language Model over a
1.7 million word corpus (Reuters) in a few seconds on your laptop*

Business and financial news

today the _______

get probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
…	

Sparsity problem: not much granularity in the probability distribution

slide credit to: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Generate text with *n*-gram LM

Example: Using a **tri**gram LM

`Today the ________`

$P(* \mid \texttt{Today the})$

company	0.153	
bank	0.153	
price	0.077	sample
italian	0.039	
emirate	0.039	
…		

Note: Sampling strategies will affect which next word get sampled (not necessarily the highest probability one)

Adapted from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Generate text with *n*-gram LM

`Today the price ________`

$P(* \mid \text{the price})$

of	0.308
for	0.050
it	0.046
to	0.046
is	0.031
…	

sample

Generate text with *n*-gram LM

`Today the price of ________`

$P(* \mid \texttt{price of})$

the	0.072
18	0.043
oil	0.043
its	0.036
gold	0.018
…	

sample

Generate text with *n*-gram LM

- A complete example

```
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 .
```

Grammatical but **not consistent**

We need longer context (more than three words) to model language well

but that means sparsity and storage problems …

n-gram LM Recap

- **Pros:**

☐ Easy to understand

☐ Cheap to implement

☐ Decent performance in application when training data is scarce

- **Cons:**

☐ Fixed vocabulary assumption

☐ Markov assumption is linguistically inaccurate

☐ Sparsity and storage problems

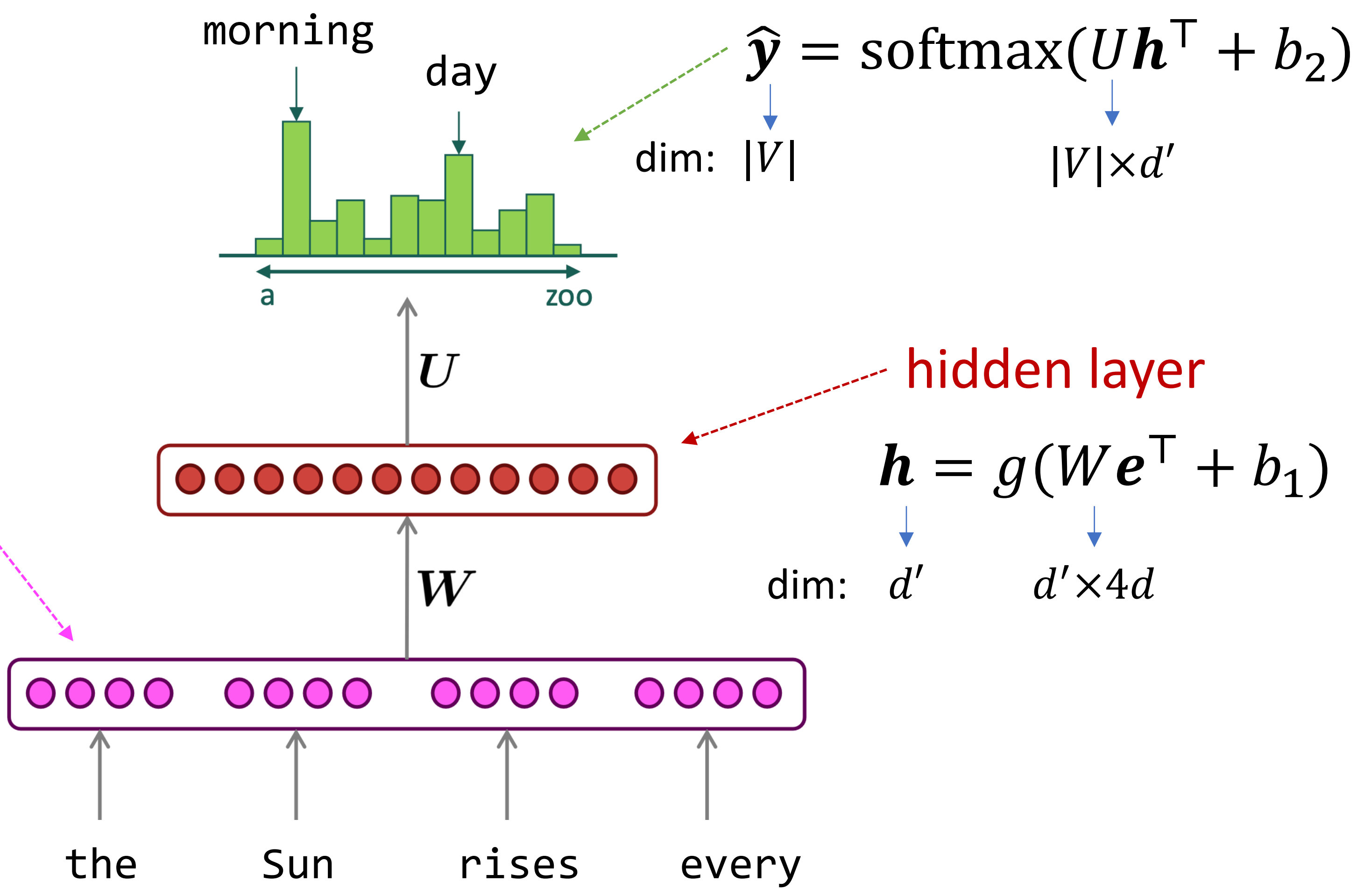
Adapted from: https://nasmith.github.io/NLP-winter23/assets/slides/lm.pdf

Overview

- Language Modeling
- **Neural Language Models**
- Recurrent Neural Networks for LM
- Evaluate LMs
- Long Short-Term Memory RNNs (LSTMs)

Fixed-Window Neural Language Model

- **Idea**: Represent words with embedding vectors; predict the next word using the concatenated embeddings from a fixed context window



concatenated word embeddings

$$\boldsymbol{e} = \underbrace{[\boldsymbol{e}^{\langle 1\rangle}; \boldsymbol{e}^{\langle 2\rangle}; \boldsymbol{e}^{\langle 3\rangle}; \boldsymbol{e}^{\langle 4\rangle}]}_{4\times d} \} \, d$$

Input tokens: $x^{\langle 1\rangle}, x^{\langle 2\rangle}, x^{\langle 3\rangle}, x^{\langle 4\rangle}$

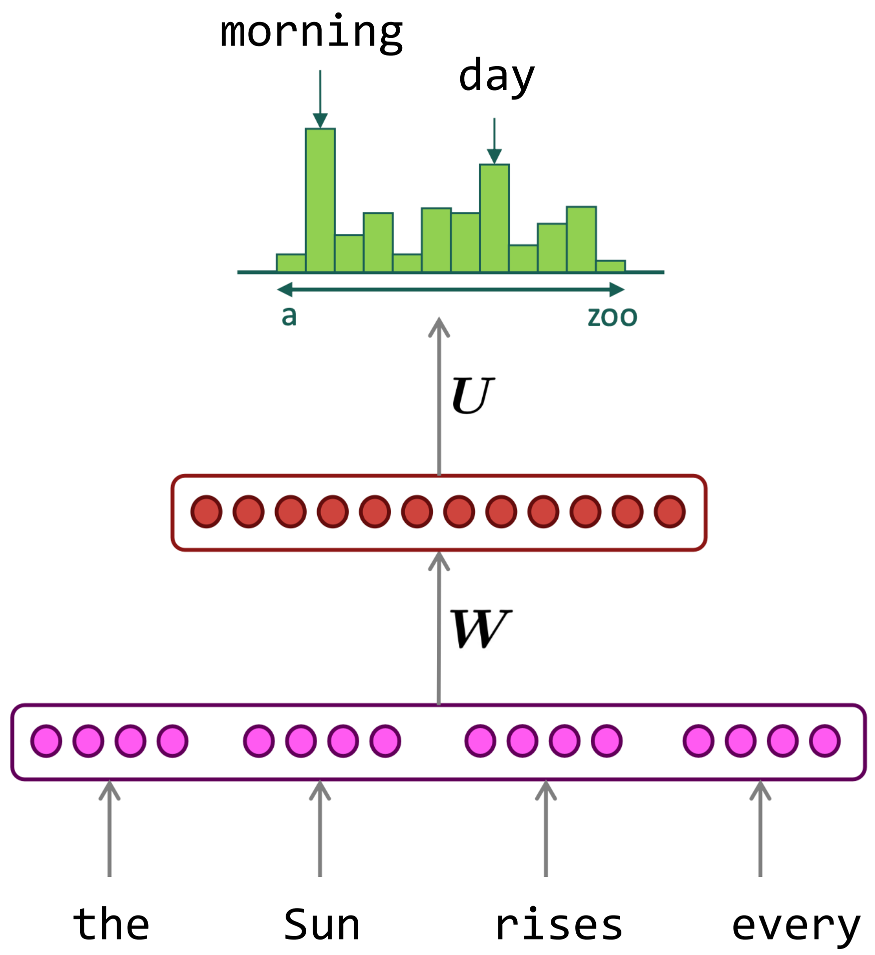
(window size = 4)

Figure from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Fixed-Window Neural LM

- **Improvements** over n-gram LM:
- No sparsity problem;
- No need to store all observed n-grams

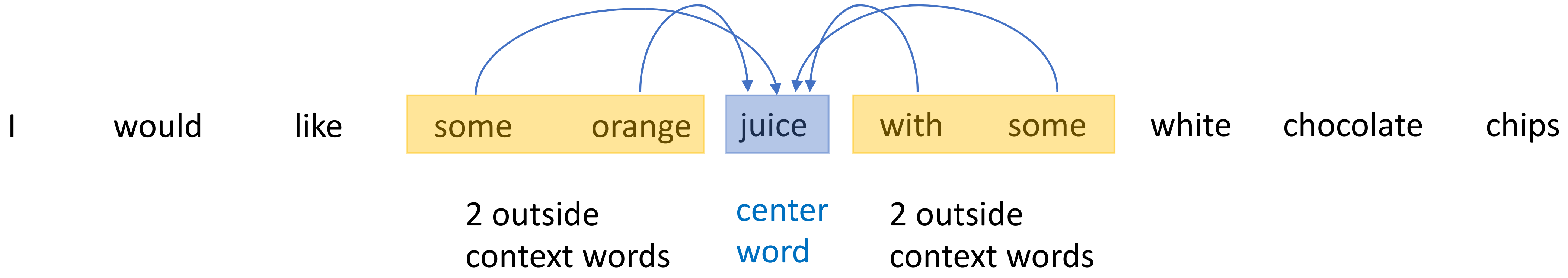
- Remaining **problems**:
- Fixed window can be small
- Increasing window size also increase W
- Need a neural architecture that can process *any input length*



Word2vec (CBOW) is also a neural LM (generic)

Aggregate all context words
as if they a bag of words

$P(w_t|w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2})$

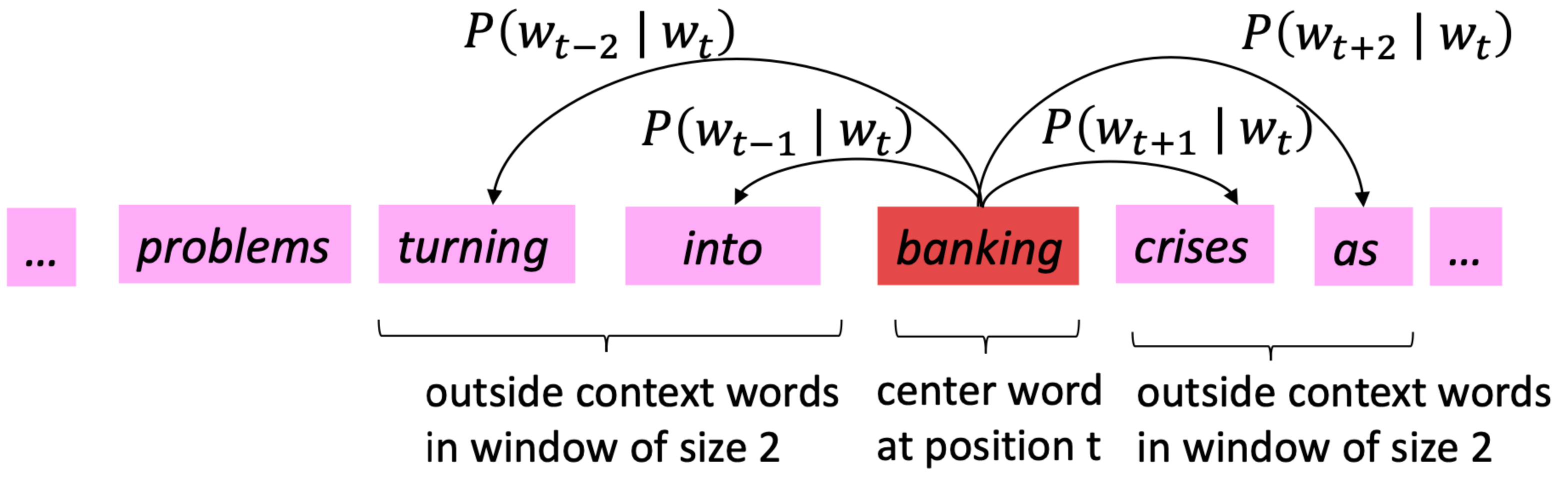


Compute only one
probability at position t:
$P(w_t|w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2})$
, for **window size 2**

Difference from fixed-window neural-LM:
The prediction is **bidirectional**

Word2vec (skip-gram) is also a neural LM (generic)

Skip-gram: Compute probability $P(w_{t+j}|w_t)$, for $j \in \{-2,-1,1,2\}$ when window size is 2



Difference from fixed-window neural-LM:

- The prediction is **bidirectional**
- Window size is minimal: 1

Example from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Overview

- Language Modeling
- Neural Language Models
- **Recurrent Neural Networks for LM**
- Evaluate LMs
- Long Short-Term Memory RNNs (LSTMs)

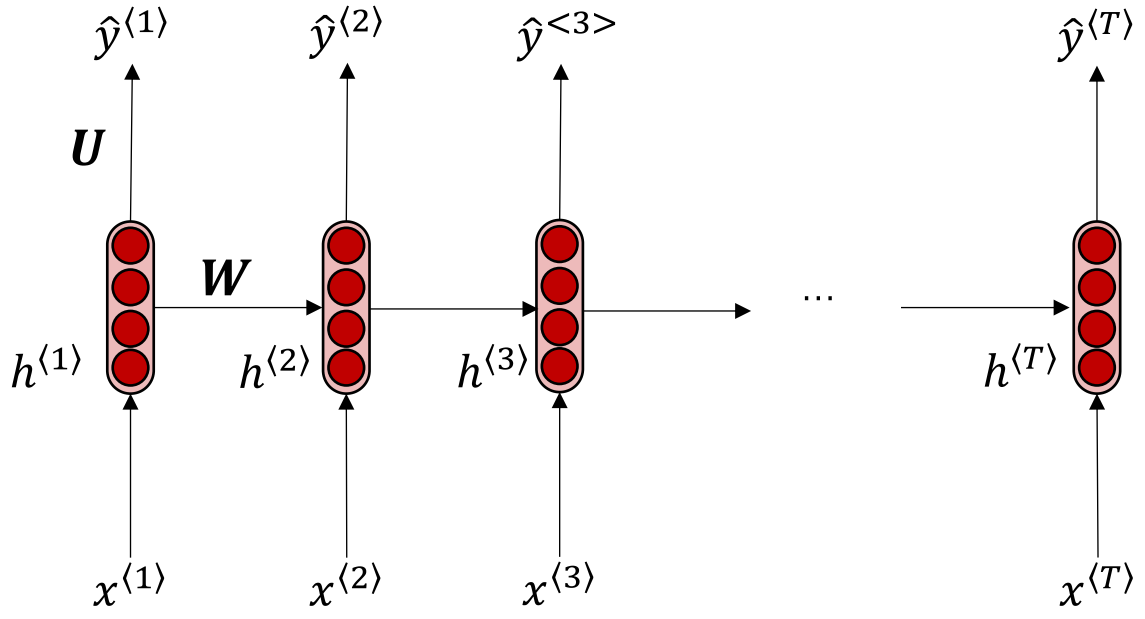
Recurrent Neural Networks

- "recurrent": *adj*. occurring repeatedly
- **Core idea**: apply the same weights W/U repeatedly at different time steps

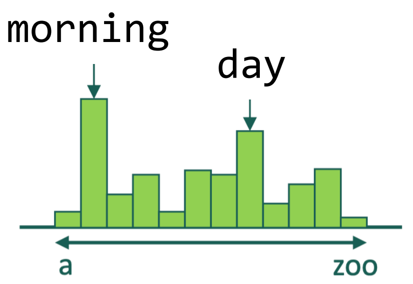
output sequence (optional)

hidden state

Input sequence (of any length T)



A Simple RNN Language Model



$\hat{y}^{\langle 4\rangle} = P(x^{\langle 5\rangle}|\texttt{the Sun rises every})$

output (optional)

$$\hat{\boldsymbol{y}}^{\langle t\rangle} = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h}^{\langle t\rangle} + b_2)$$

hidden state

$$\boldsymbol{h}^{\langle t\rangle} = g(\boldsymbol{W}_h\boldsymbol{h}^{\langle t-1\rangle} + \boldsymbol{W}_e\boldsymbol{e}^{\langle t\rangle} + b_1)$$

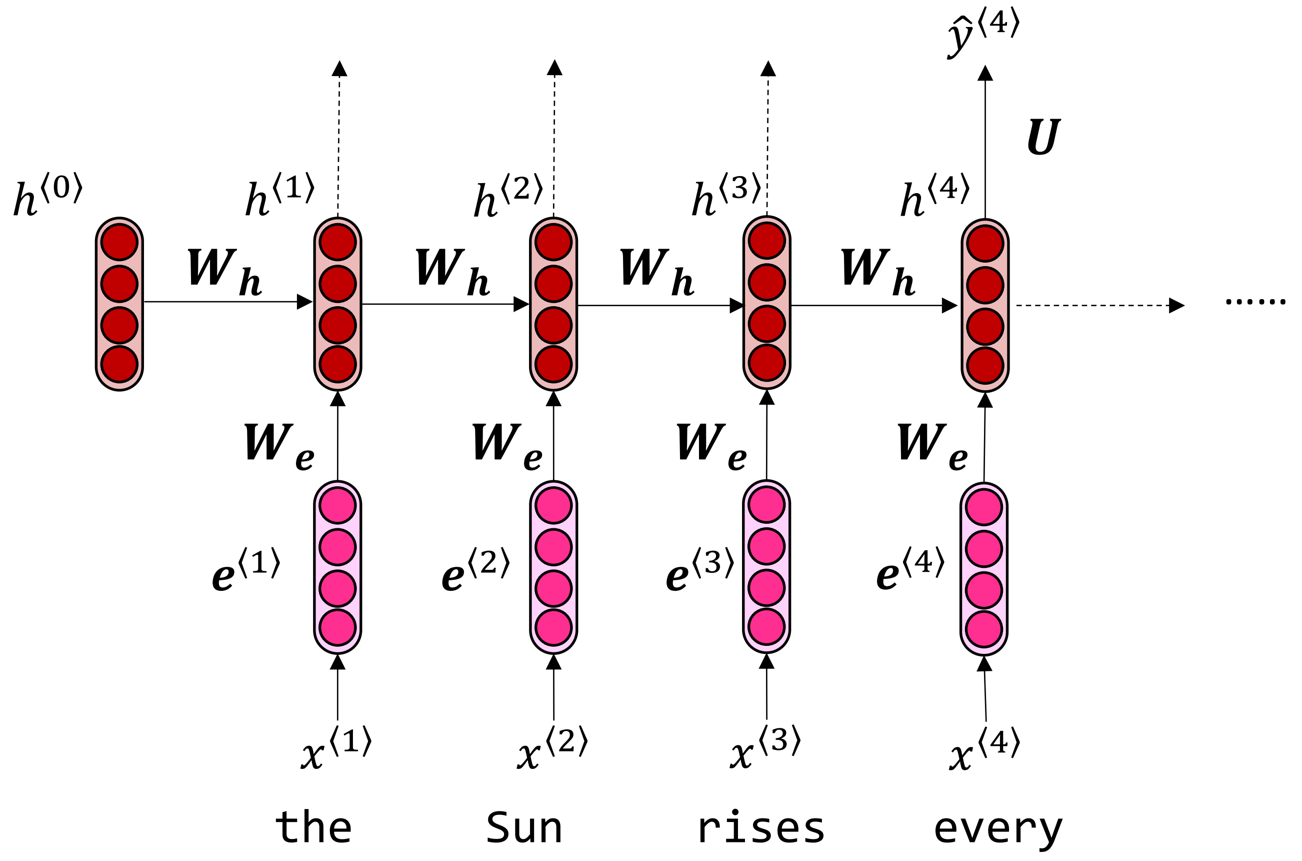
($\boldsymbol{h}^{\langle 0\rangle}$ is the initial hidden state)

input embedding

$$\boldsymbol{e}^{\langle t\rangle} \in \mathbb{R}^d$$

input sequence

$$\boldsymbol{x}^{\langle t\rangle}$$



Training an RNN LM: Objective and loss

- **Next token prediction task:** Given a sequence of T tokens $x^{\langle 1 \rangle}, \dots, x^{\langle T \rangle}$
- Feed them as input to RNN-LM; compute the output probability for **every time step t, $\hat{\boldsymbol{y}}^{\langle t \rangle}$**
- **Loss function:** The cross-entropy between the predicted probability $\hat{\boldsymbol{y}}^{\langle t \rangle}$ and the true next word (ground truth) $\boldsymbol{y}^{\langle t \rangle}$, (that is, $\boldsymbol{x}^{\langle t+1 \rangle}$!)

(negative log likelihood)

$$J^{\langle t \rangle}(\theta) = \mathrm{cross-entropy}\left(\hat{\boldsymbol{y}}^{\langle t \rangle}, \boldsymbol{y}^{\langle t \rangle}\right) = -\sum_{w \in V} \boldsymbol{y}_{\boldsymbol{w}}^{\langle t \rangle} \log \hat{\boldsymbol{y}}_{\boldsymbol{w}}^{\langle t \rangle} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}^{\langle t+1 \rangle}}^{\langle t \rangle}$$

this term is 1 only for $w = \boldsymbol{y}^{\langle t \rangle}$; all zeros for other words

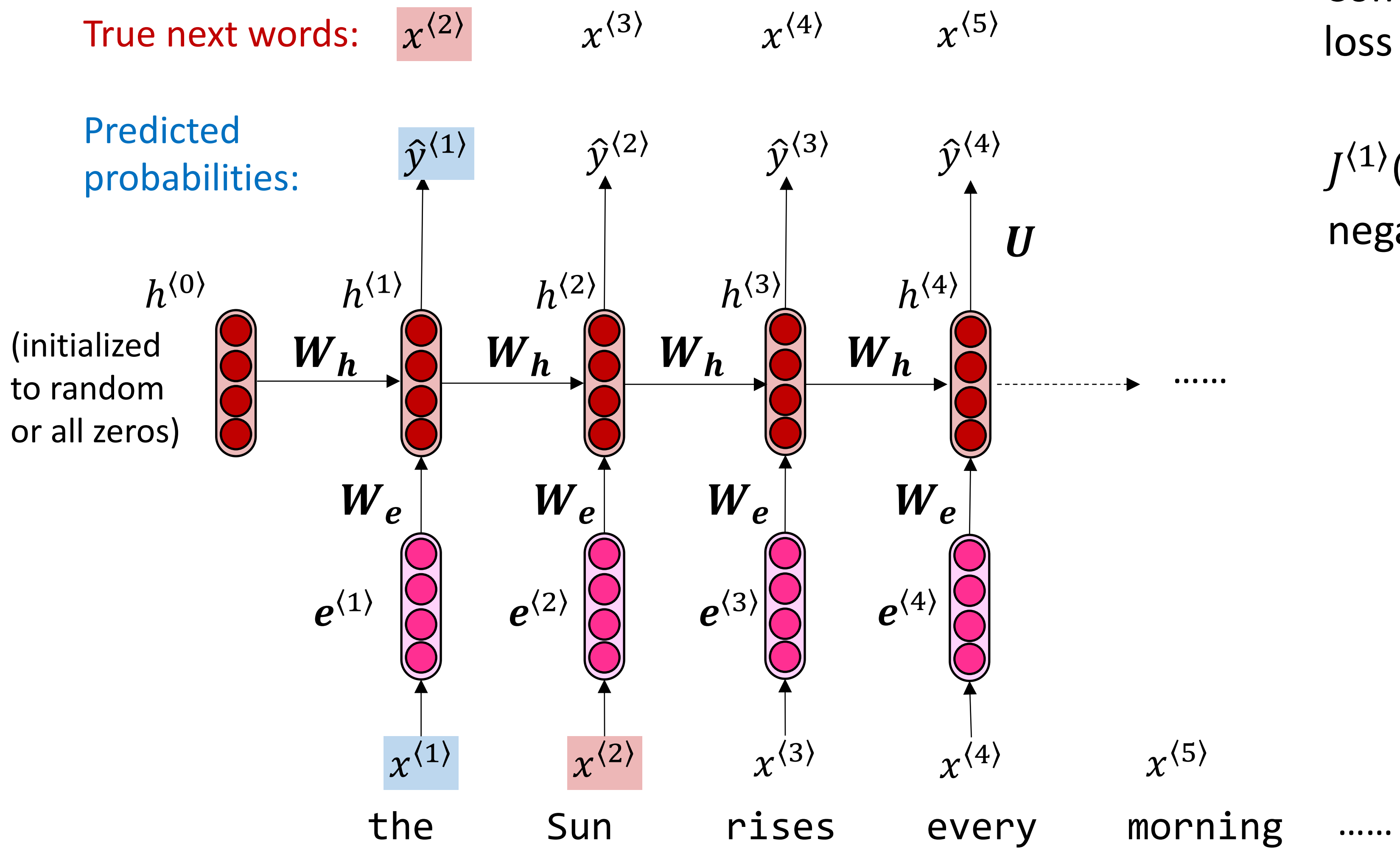
$P(\boldsymbol{x}^{\langle t+1 \rangle} | \text{all previous words})$

- Average over entire training set:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{\langle t \rangle}(\theta) = \frac{1}{T}\sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}^{\langle t+1 \rangle}}^{\langle t \rangle}$$

θ denotes all model parameters: U, W_e, W_h, b_1, b_2

Training an RNN LM: Example



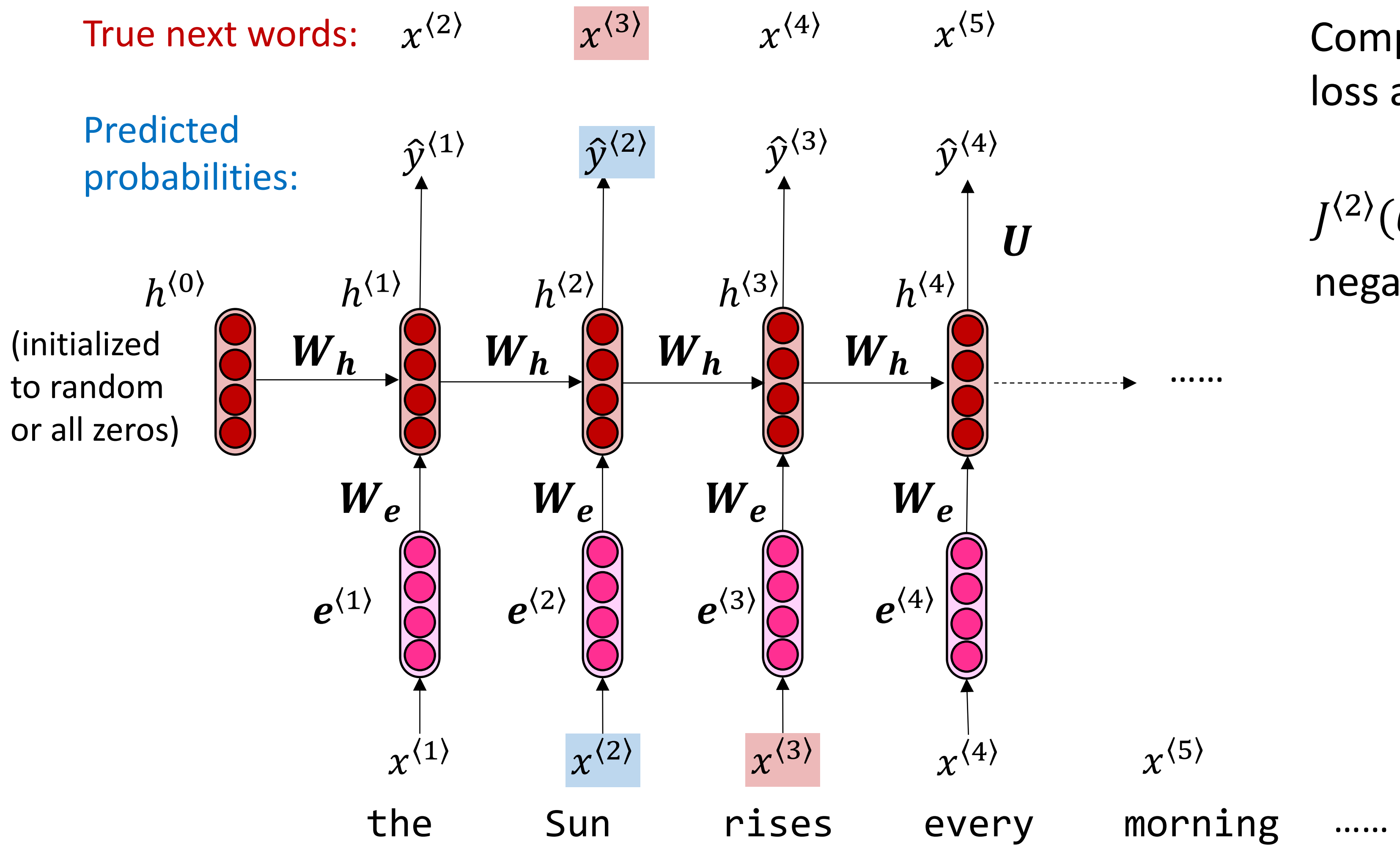
True next words:
$x^{\langle 2\rangle}$ $x^{\langle 3\rangle}$ $x^{\langle 4\rangle}$ $x^{\langle 5\rangle}$
Predicted probabilities:
$\hat{y}^{\langle 1\rangle}$ $\hat{y}^{\langle 2\rangle}$ $\hat{y}^{\langle 3\rangle}$ $\hat{y}^{\langle 4\rangle}$
U
$h^{\langle 0\rangle}$ $h^{\langle 1\rangle}$ $h^{\langle 2\rangle}$ $h^{\langle 3\rangle}$ $h^{\langle 4\rangle}$
(initialized to random or all zeros)
W_h
W_e
$e^{\langle 1\rangle}$ $e^{\langle 2\rangle}$ $e^{\langle 3\rangle}$ $e^{\langle 4\rangle}$
$x^{\langle 1\rangle}$ $x^{\langle 2\rangle}$ $x^{\langle 3\rangle}$ $x^{\langle 4\rangle}$ $x^{\langle 5\rangle}$
the Sun rises every morning ……
……

Compute the loss at step 1:

$$J^{\langle 1\rangle}(\theta) = \mathrm{CE}\left(\hat{y}^{\langle 1\rangle}, x^{\langle 2\rangle}\right) = -\log \hat{y}^{\langle 1\rangle}_{\text{Sun}},$$

negative log-probability of "Sun"

Training an RNN LM: Example

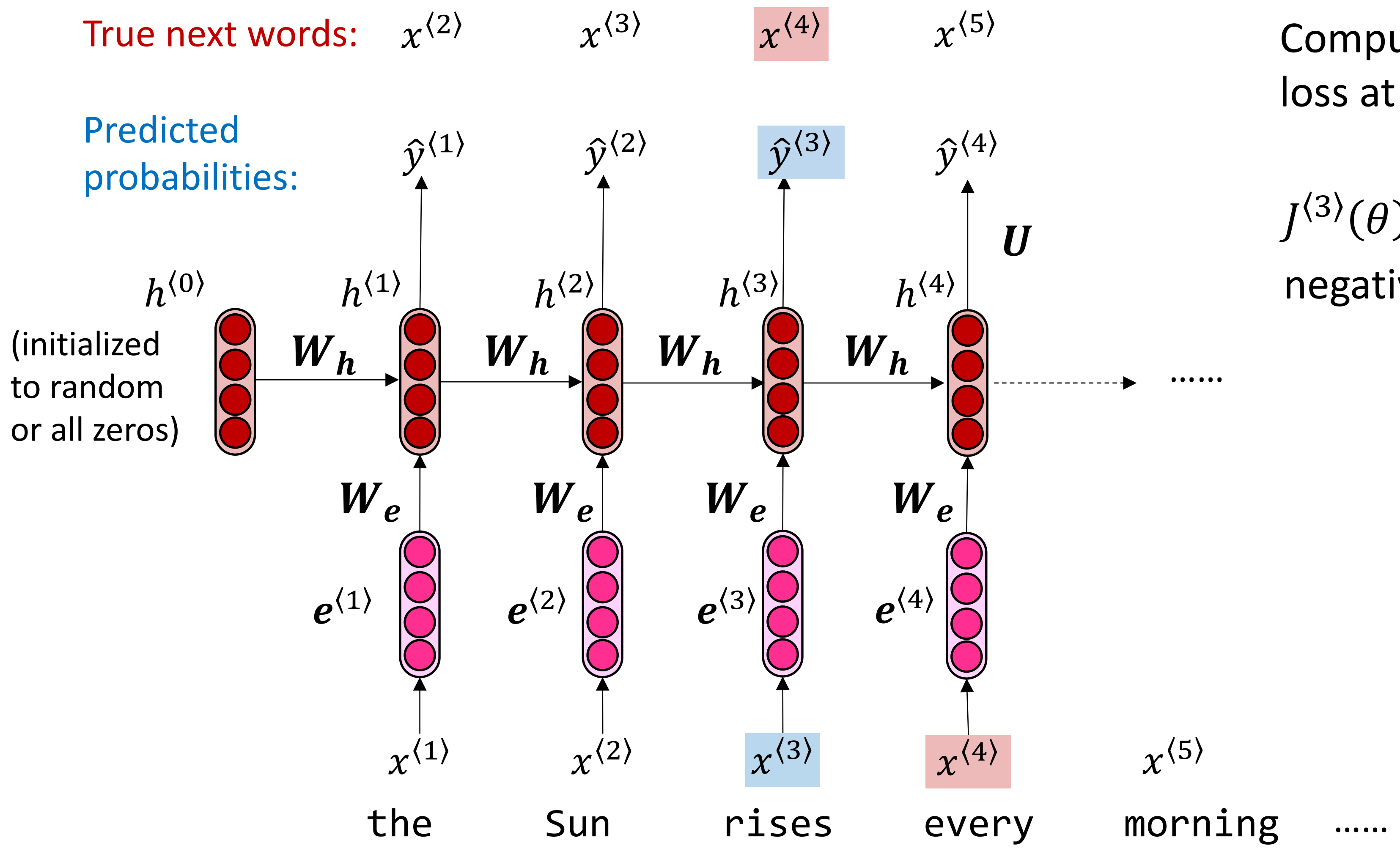


Compute the loss at step 2:

$$J^{\langle 2\rangle}(\theta) = \mathrm{CE}\left(\hat{y}^{\langle 2\rangle}, x^{\langle 3\rangle}\right) = -\log \hat{y}^{\langle 2\rangle}_{\texttt{rises}},$$

negative log-probability of “`rises`”

Training an RNN LM: Example

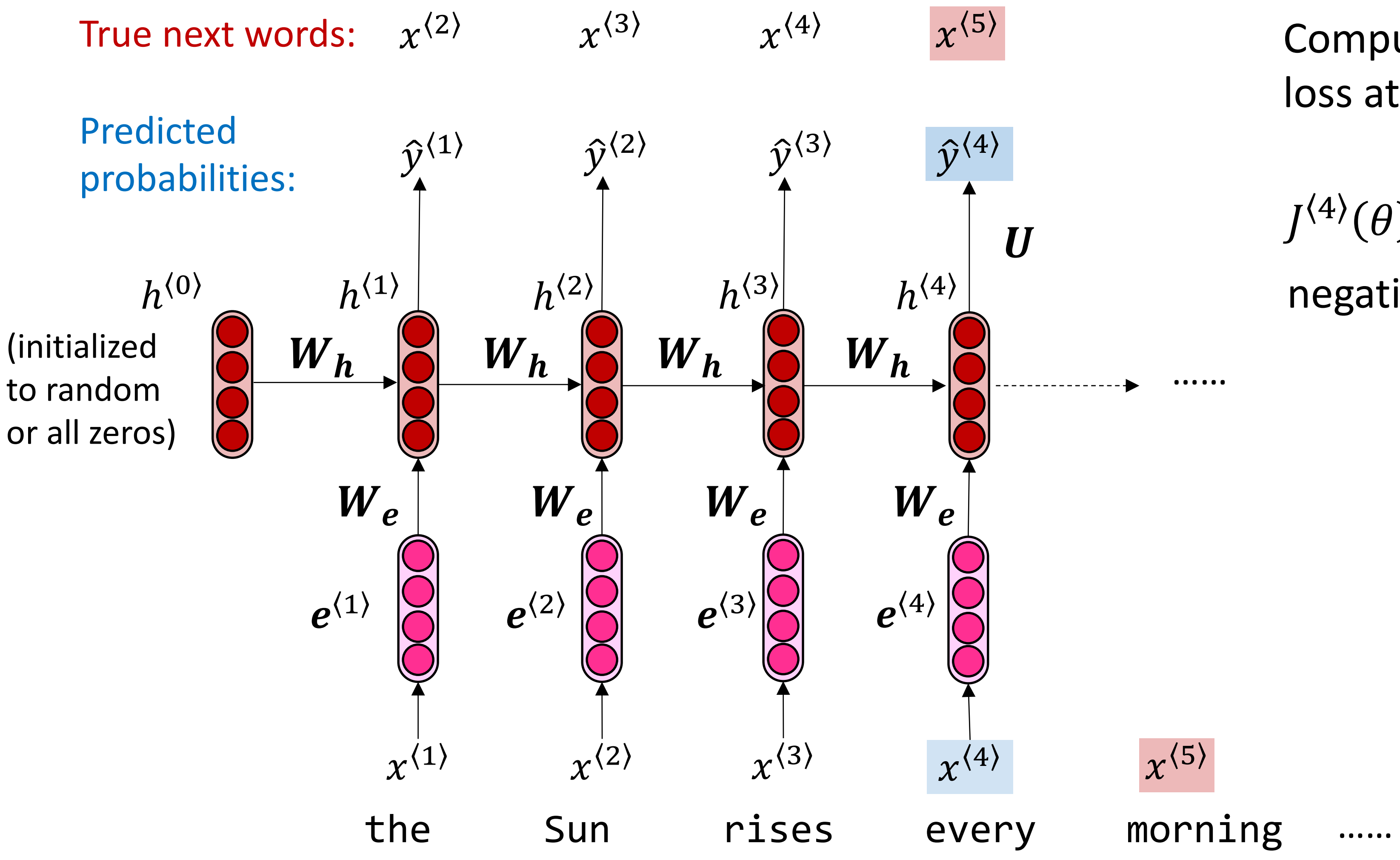


Compute the loss at step 3:

$$J^{\langle 3 \rangle}(\theta) = \mathrm{CE}\left(\hat{y}^{\langle 3 \rangle}, x^{\langle 4 \rangle}\right) = -\log \hat{y}^{\langle 3 \rangle}_{\texttt{every}},$$

negative log-probability of "every"

Training an RNN LM: Example



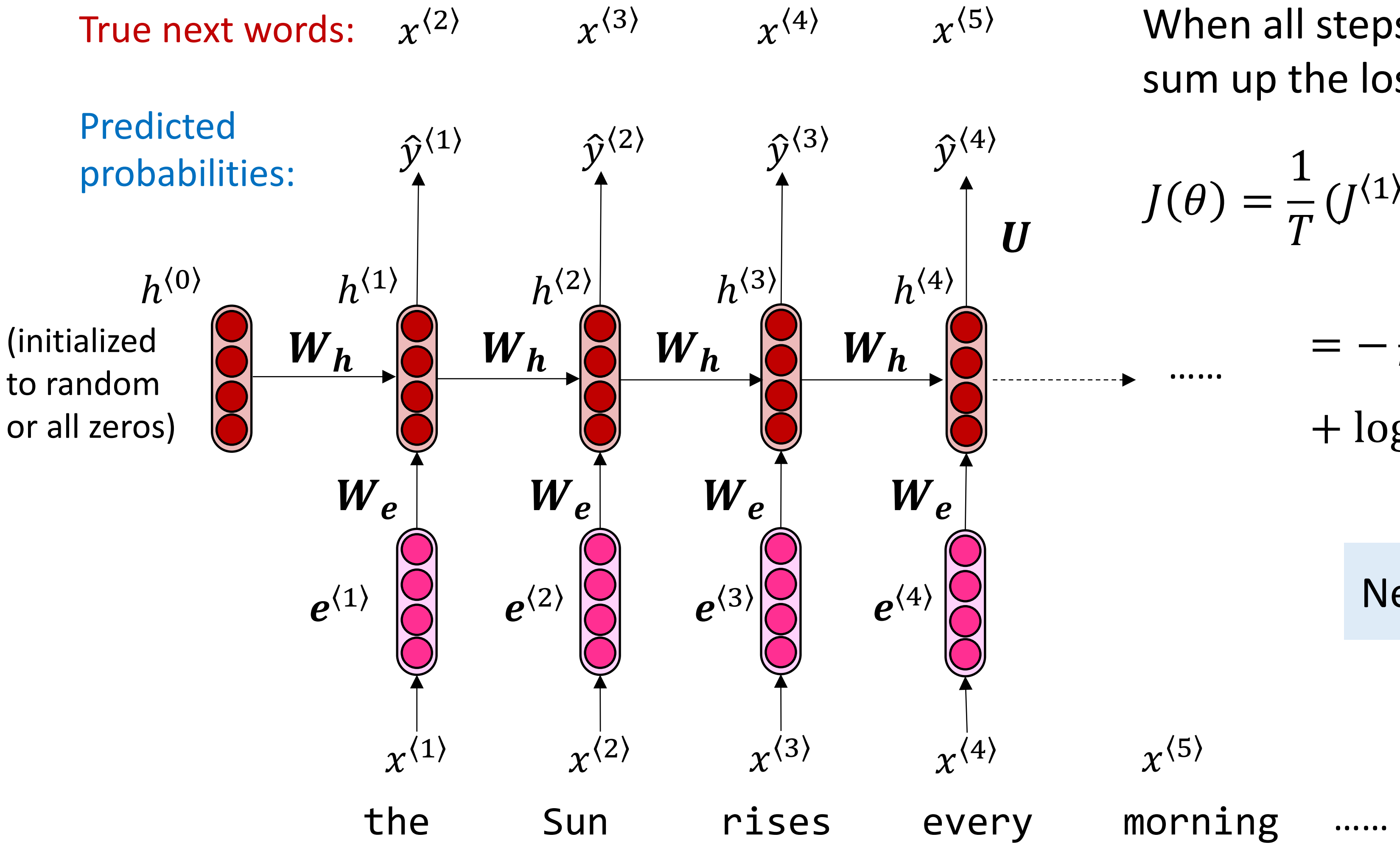
True next words: $x^{\langle 2\rangle}$ $x^{\langle 3\rangle}$ $x^{\langle 4\rangle}$ $x^{\langle 5\rangle}$
Predicted probabilities: $\hat{y}^{\langle 1\rangle}$ $\hat{y}^{\langle 2\rangle}$ $\hat{y}^{\langle 3\rangle}$ $\hat{y}^{\langle 4\rangle}$
U
$h^{\langle 0\rangle}$ $h^{\langle 1\rangle}$ $h^{\langle 2\rangle}$ $h^{\langle 3\rangle}$ $h^{\langle 4\rangle}$
(initialized to random or all zeros)
W_h W_h W_h W_h
……
W_e W_e W_e W_e
$e^{\langle 1\rangle}$ $e^{\langle 2\rangle}$ $e^{\langle 3\rangle}$ $e^{\langle 4\rangle}$
$x^{\langle 1\rangle}$ $x^{\langle 2\rangle}$ $x^{\langle 3\rangle}$ $x^{\langle 4\rangle}$ $x^{\langle 5\rangle}$
the Sun rises every morning ……

Compute the loss at step 4:

$$J^{\langle 4\rangle}(\theta) = \mathrm{CE}\big(\hat{y}^{\langle 4\rangle}, x^{\langle 5\rangle}\big) = -\log \hat{y}^{\langle 4\rangle}_{\texttt{morning}},$$

negative log-probability of “morning”

Training an RNN LM: Example


True next words:
$x^{\langle 2\rangle}$ $x^{\langle 3\rangle}$ $x^{\langle 4\rangle}$ $x^{\langle 5\rangle}$
Predicted probabilities:
$\hat{y}^{\langle 1\rangle}$ $\hat{y}^{\langle 2\rangle}$ $\hat{y}^{\langle 3\rangle}$ $\hat{y}^{\langle 4\rangle}$
U
$h^{\langle 0\rangle}$ $h^{\langle 1\rangle}$ $h^{\langle 2\rangle}$ $h^{\langle 3\rangle}$ $h^{\langle 4\rangle}$
(initialized to random or all zeros)
W_h
W_e
$e^{\langle 1\rangle}$ $e^{\langle 2\rangle}$ $e^{\langle 3\rangle}$ $e^{\langle 4\rangle}$
$x^{\langle 1\rangle}$ $x^{\langle 2\rangle}$ $x^{\langle 3\rangle}$ $x^{\langle 4\rangle}$ $x^{\langle 5\rangle}$
the Sun rises every morning ……
……

When all steps have been predicted, sum up the loss:

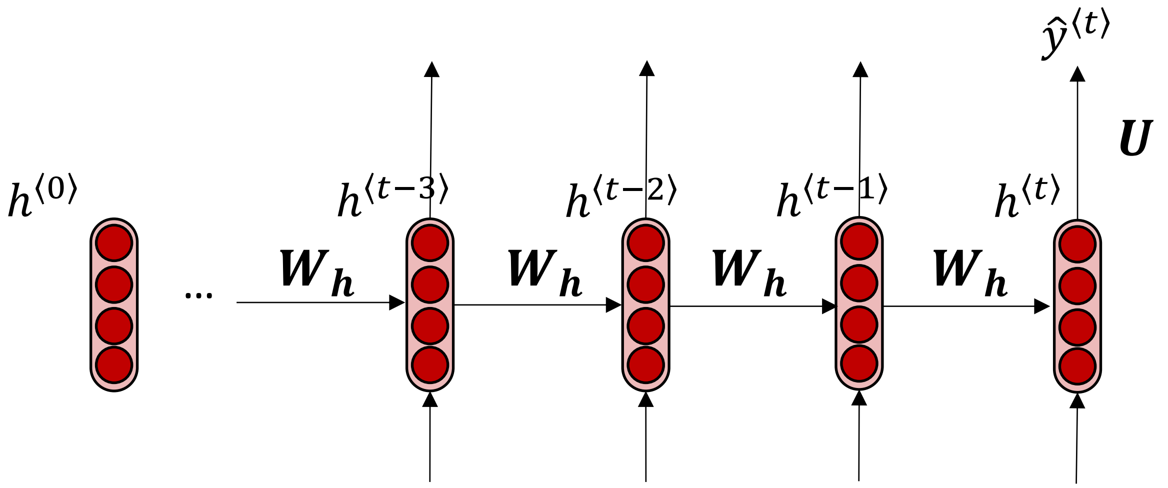
$$J(\theta) = \frac{1}{T}\left(J^{\langle 1\rangle}(\theta) + J^{\langle 2\rangle}(\theta) + J^{\langle 3\rangle}(\theta) + J^{\langle 4\rangle}(\theta) \ldots\right)$$

$$= -\frac{1}{T}\left(-\log \hat{y}^{\langle 1\rangle}_{\texttt{Sun}} + \log \hat{y}^{\langle 2\rangle}_{\texttt{rises}} + \log \hat{y}^{\langle 3\rangle}_{\texttt{every}} + \log \hat{y}^{\langle 4\rangle}_{\texttt{morning}} + \cdots\right)$$

Next, compute gradients: $\frac{\partial J(\theta)}{\partial \theta}$

Backpropagation for RNN

- **Question**: How to compute $\frac{\partial J^{\langle t \rangle}(\theta)}{\partial \theta}$? Here $\theta \coloneqq \{\boldsymbol{U}, \boldsymbol{W_e}, \boldsymbol{W_h}, \boldsymbol{b_1}, \boldsymbol{b_2}\}$
- For simplification, how to compute $\frac{\partial J^{\langle t \rangle}}{\partial \boldsymbol{W_h}}$?



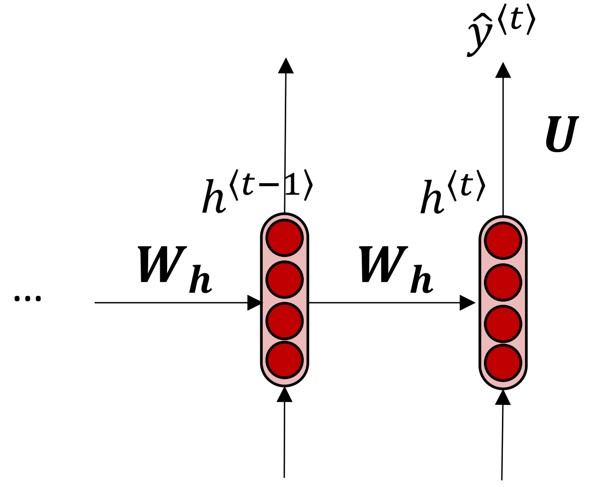
Solution: Backpropagation through time (BPTT) (Werbos, 1990)

$$\frac{\partial J^{\langle t \rangle}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \frac{\partial J^{\langle t \rangle}}{\partial \boldsymbol{W_h}} \bigg|_{(i)}$$

gradients contributed by time step i

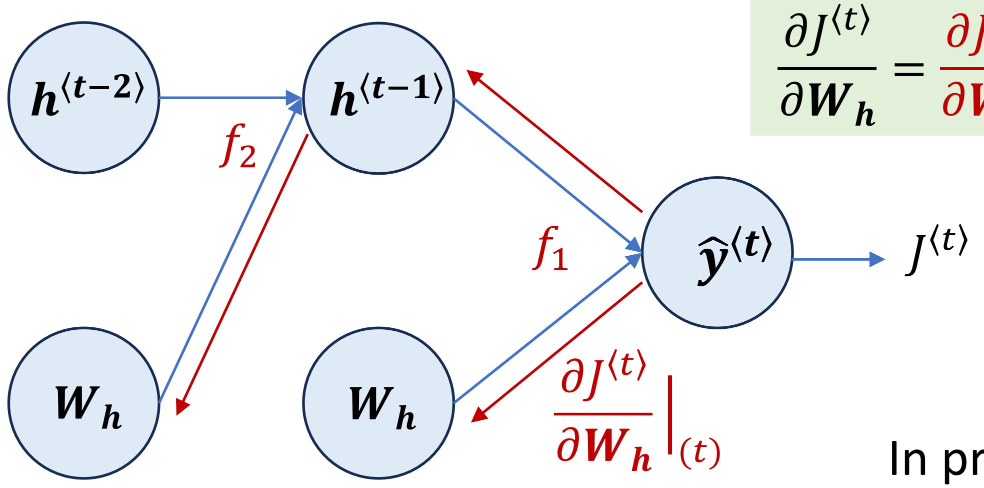
Sum up the gradients from each time step the weight has appeared

Backpropagation for RNN



$$\underbrace{\hat{\boldsymbol{y}}^{\langle t\rangle} = \text{softmax}(\boldsymbol{U}\boldsymbol{h}^{\langle t\rangle} + b_2) \qquad \boldsymbol{h}^{\langle t\rangle} = g(\boldsymbol{W_h}\boldsymbol{h}^{\langle t-1\rangle} + \boldsymbol{W_e}\boldsymbol{e}^{\langle t\rangle} + b_1)}_{\hat{\boldsymbol{y}}^{\langle t\rangle} = f_1(\boldsymbol{W_h}, \boldsymbol{h}^{\langle t-1\rangle})}$$

$$\frac{\partial J^{\langle t\rangle}}{\partial \boldsymbol{W_h}} = \left.\frac{\partial J^{\langle t\rangle}}{\partial \boldsymbol{W_h}}\right|_{(t)} + \left.\frac{\partial J^{\langle t\rangle}}{\partial \boldsymbol{W_h}}\right|_{(t-1)} + \cdots$$



$\boldsymbol{h}^{\langle t-2\rangle}$
$\boldsymbol{h}^{\langle t-1\rangle}$
f_2
f_1
$\hat{\boldsymbol{y}}^{\langle t\rangle}$
$J^{\langle t\rangle}$
$\boldsymbol{W_h}$
$\boldsymbol{W_h}$
$\left.\frac{\partial J^{\langle t\rangle}}{\partial \boldsymbol{W_h}}\right|_{(t)}$

Backpropagate all the way to the very first step

$$\underbrace{\boldsymbol{h}^{\langle t-1\rangle} = g(\boldsymbol{W_h}\boldsymbol{h}^{\langle t-2\rangle} + \cdots)}_{f_2(\boldsymbol{W_h}, \boldsymbol{h}^{\langle t-2\rangle})}$$

$$\left.\frac{\partial J^{\langle t\rangle}}{\partial \boldsymbol{W_h}}\right|_{(t-1)} = \frac{\partial J^{\langle t\rangle}}{\partial \boldsymbol{h}^{\langle t-1\rangle}} \cdot \frac{\partial \boldsymbol{h}^{\langle t-1\rangle}}{\partial \boldsymbol{W_h}}$$

In practice, often **truncated** after ~20 timesteps for efficiency reasons

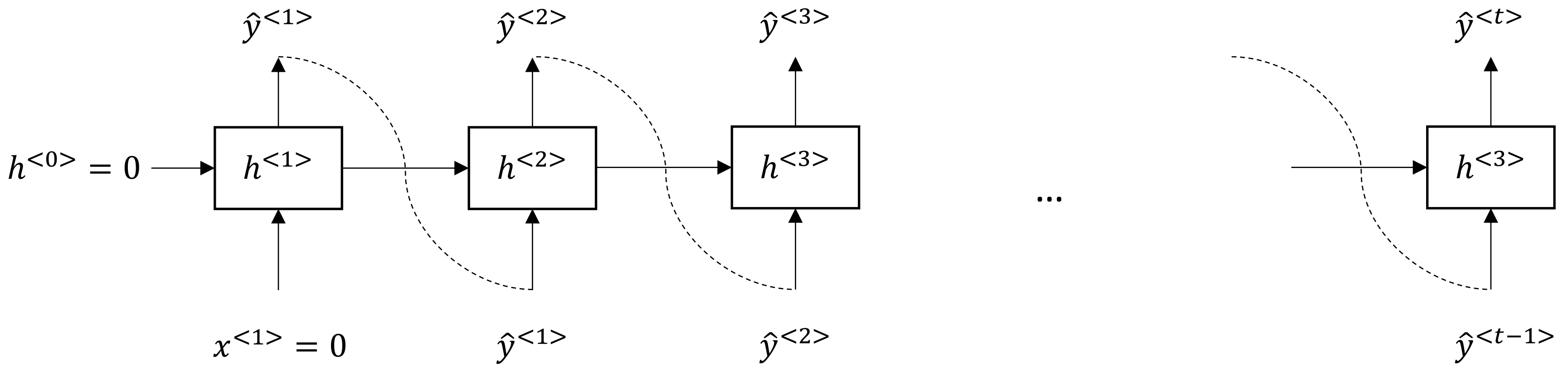
Generate text with RNN-LM

- Just like an *n*-gram Language Model, we can use an RNN-LM to generate text by **repeated sampling**. Sampled output becomes next step's input.

E.g., $\hat{y}^{<1>} = The$ $\quad \hat{y}^{<2>} = Sun$ $\quad \hat{y}^{<3>} = rises$ $\quad \hat{y}^{<t>} = <EOS>$

sample from the output distribution

$$\begin{bmatrix} P(a) \\ P(about) \\ P(all) \\ \vdots \\ P(zoo) \end{bmatrix} \quad \begin{bmatrix} P(a|The) \\ P(about|The) \\ P(all|The) \\ \vdots \\ P(zoo|The) \end{bmatrix} \quad \begin{bmatrix} P(a|The\ Sun) \\ P(about|The\ Sun) \\ P(all|The\ Sun) \\ \vdots \\ P(zoo|The\ Sun) \end{bmatrix} \quad \begin{bmatrix} P(a|The\ Sun\ rises \cdots) \\ P(about|The\ Sun\ rises \cdots) \\ P(all|The\ Sun\ rises \cdots) \\ \vdots \\ P(zoo|The\ Sun\ rises\ \cdots) \end{bmatrix}$$



Fun Examples of generated text

- RNN-LM trained on *Harry Potter*

Part 1
“The Malfoys!” said Hermione.

Harry was watching him. He looked like Madame Maxime. When she strode up the wrong staircase to visit himself.

“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.

Somewhat better than *n*-gram LM, but still not consistent content.

From a post in 2016: https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

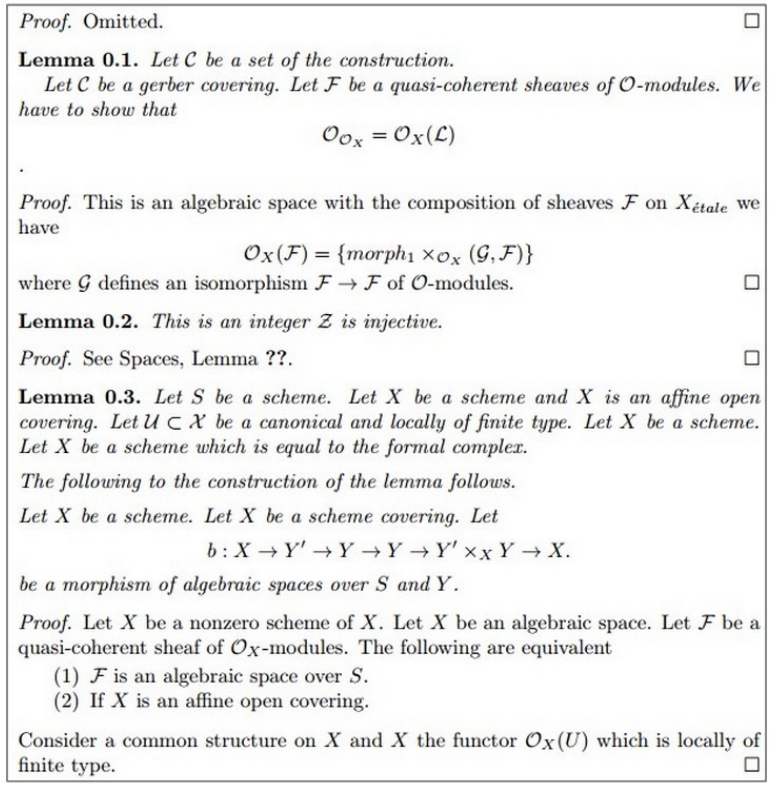
Fun Examples of generated text

Linux source code

```c
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
```

(fake code that does not compile)

Math text book → learned from Latex code, and almost compiled



Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{étale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

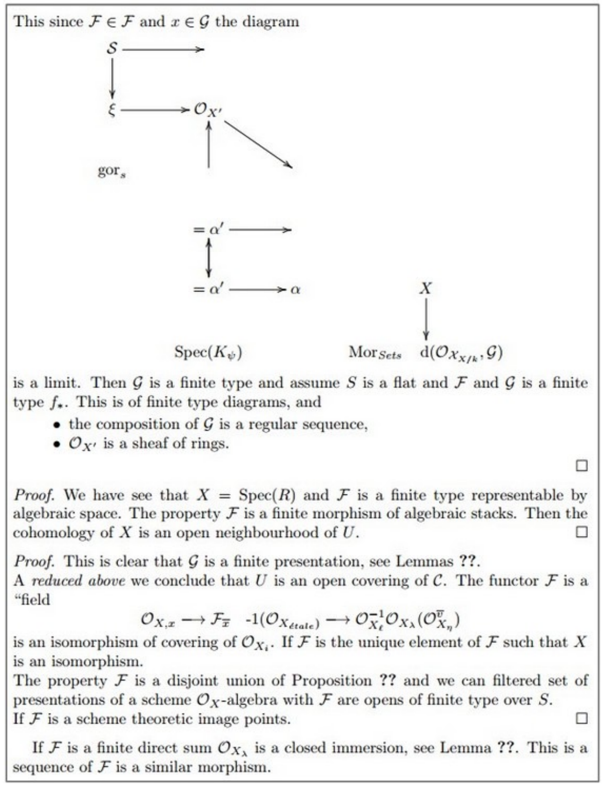
$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □



This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram

$$\begin{tikzcd}
\mathcal{S} \arrow[r] \arrow[d] & {} & & \\
\xi \arrow[r] & \mathcal{O}_{X'} \arrow[rd] & & \\
\mathrm{gor}_s & {} \arrow[u] & {} & \\
& =\alpha' \arrow[r] \arrow[d, leftrightarrow] & {} & \\
& =\alpha' \arrow[r] & \alpha & X \arrow[d] \\
& \mathrm{Spec}(K_\psi) & \mathrm{Mor}_{Sets} & d(\mathcal{O}_{\mathcal{X}_{X/k}}, \mathcal{G})
\end{tikzcd}$$

is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \ \text{-1}(\mathcal{O}_{X_{étale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

From Andraj Karpathy's post : http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Overview

- Language Modeling
- Neural Language Models
- Recurrent Neural Networks for LM
- **Evaluate LMs**
- Long Short-Term Memory RNNs (LSTMs)

Evaluate Language Models

- Intrinsic evaluation metric: **perplexity (困惑度)**

$$\text{Perplexity} = \prod_{t=1}^{T} \left(\frac{1}{P(x^{\langle t+1 \rangle} | x^{\langle 1 \rangle}, \dots, x^{\langle t \rangle})} \right)^{1/T}$$

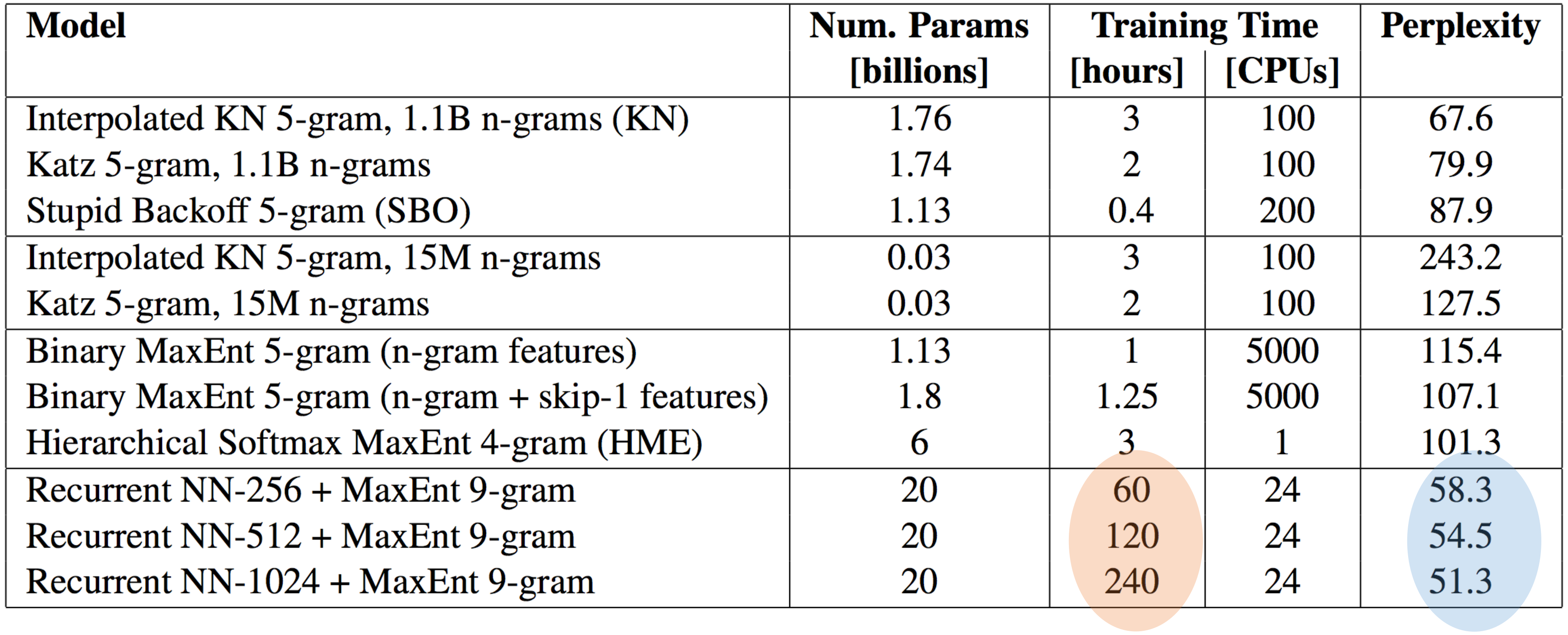
Inverse probability of all words in corpus, normalized by total word count

- Equivalent to the exponential of the cross-entropy loss

$$\log(\text{Perplexity}) = \frac{1}{T} \sum_{t=1}^{T} -\log P(x^{\langle t+1 \rangle} | x^{\langle 1 \rangle}, \dots, x^{\langle t \rangle}) = J(\theta)$$

Lower perplexity is better (in general) $\Rightarrow$ higher probability (likelihood) of words $\Rightarrow$ more *expected* words

Evaluate LMs with Perplexity



Model	Num. Params [billions]	Training Time [hours]	Training Time [CPUs]	Perplexity
Interpolated KN 5-gram, 1.1B n-grams (KN)	1.76	3	100	67.6
Katz 5-gram, 1.1B n-grams	1.74	2	100	79.9
Stupid Backoff 5-gram (SBO)	1.13	0.4	200	87.9
Interpolated KN 5-gram, 15M n-grams	0.03	3	100	243.2
Katz 5-gram, 15M n-grams	0.03	2	100	127.5
Binary MaxEnt 5-gram (n-gram features)	1.13	1	5000	115.4
Binary MaxEnt 5-gram (n-gram + skip-1 features)	1.8	1.25	5000	107.1
Hierarchical Softmax MaxEnt 4-gram (HME)	6	3	1	101.3
Recurrent NN-256 + MaxEnt 9-gram	20	60	24	58.3
Recurrent NN-512 + MaxEnt 9-gram	20	120	24	54.5
Recurrent NN-1024 + MaxEnt 9-gram	20	240	24	51.3

Table from: Chelba et al., 2013, One billion word benchmark for measuring progress in statistical language modeling

Problems with RNN-LM

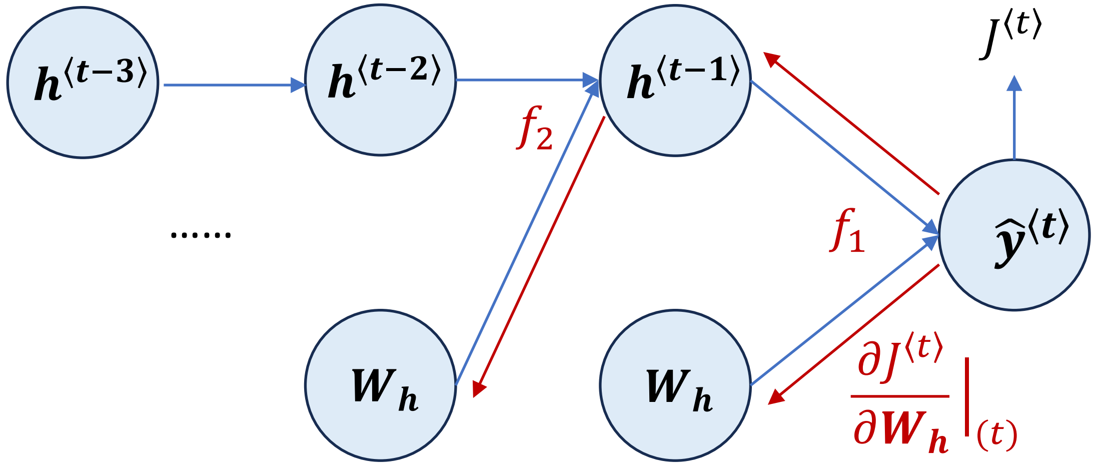
- Vanishing gradient issue

$$\left.\frac{\partial J^{\langle t\rangle}}{\partial \boldsymbol{W_h}}\right|_{(t-2)} = \frac{\partial J^{\langle t\rangle}}{\partial \boldsymbol{h}^{\langle t-1\rangle}} \cdot \frac{\partial \boldsymbol{h}^{\langle t-1\rangle}}{\partial \boldsymbol{h}^{\langle t-2\rangle}} \cdot \frac{\partial \boldsymbol{h}^{\langle t-2\rangle}}{\partial \boldsymbol{W_h}}$$

$$\left.\frac{\partial J^{\langle t\rangle}}{\partial \boldsymbol{W_h}}\right|_{(t-3)} = \frac{\partial J^{\langle t\rangle}}{\partial \boldsymbol{h}^{\langle t-1\rangle}} \cdot \frac{\partial \boldsymbol{h}^{\langle t-1\rangle}}{\partial \boldsymbol{h}^{\langle t-2\rangle}} \cdot \frac{\partial \boldsymbol{h}^{\langle t-2\rangle}}{\partial \boldsymbol{h}^{\langle t-3\rangle}} \cdot \frac{\partial \boldsymbol{h}^{\langle t-3\rangle}}{\partial \boldsymbol{W_h}}$$

……

Becomes a long chain of products



$\boldsymbol{h}^{\langle t-3\rangle}$
$\boldsymbol{h}^{\langle t-2\rangle}$
$\boldsymbol{h}^{\langle t-1\rangle}$
$J^{\langle t\rangle}$
f_2
f_1
$\hat{\boldsymbol{y}}^{\langle t\rangle}$
……
$\boldsymbol{W_h}$
$\boldsymbol{W_h}$
$\left.\frac{\partial J^{\langle t\rangle}}{\partial \boldsymbol{W_h}}\right|_{(t)}$

$$\left.\frac{\partial J^{\langle t\rangle}}{\partial \boldsymbol{W_h}}\right|_{(t-1)} = \frac{\partial J^{\langle t\rangle}}{\partial \boldsymbol{h}^{\langle t-1\rangle}} \cdot \frac{\partial \boldsymbol{h}^{\langle t-1\rangle}}{\partial \boldsymbol{W_h}}$$

Problems with RNN-LM: Vanishing gradient

- Recall: $\boldsymbol{h}^{\langle t\rangle} = g(\boldsymbol{W_h}\boldsymbol{h}^{\langle t-1\rangle} + \boldsymbol{W}_e\boldsymbol{e}^{\langle t\rangle} + b_1)$
- if g is an identity function, $g(x) = x$, then by chain rule: $\frac{\partial \boldsymbol{h}^{\langle t\rangle}}{\partial \boldsymbol{h}^{\langle t-1\rangle}} = g' \cdot \boldsymbol{W_h} = \boldsymbol{W_h}$
- Consider the loss at step i, $J^{\langle i\rangle}(\theta)$, and its gradient on step j: ($i > j$), let $\ell = i - j$

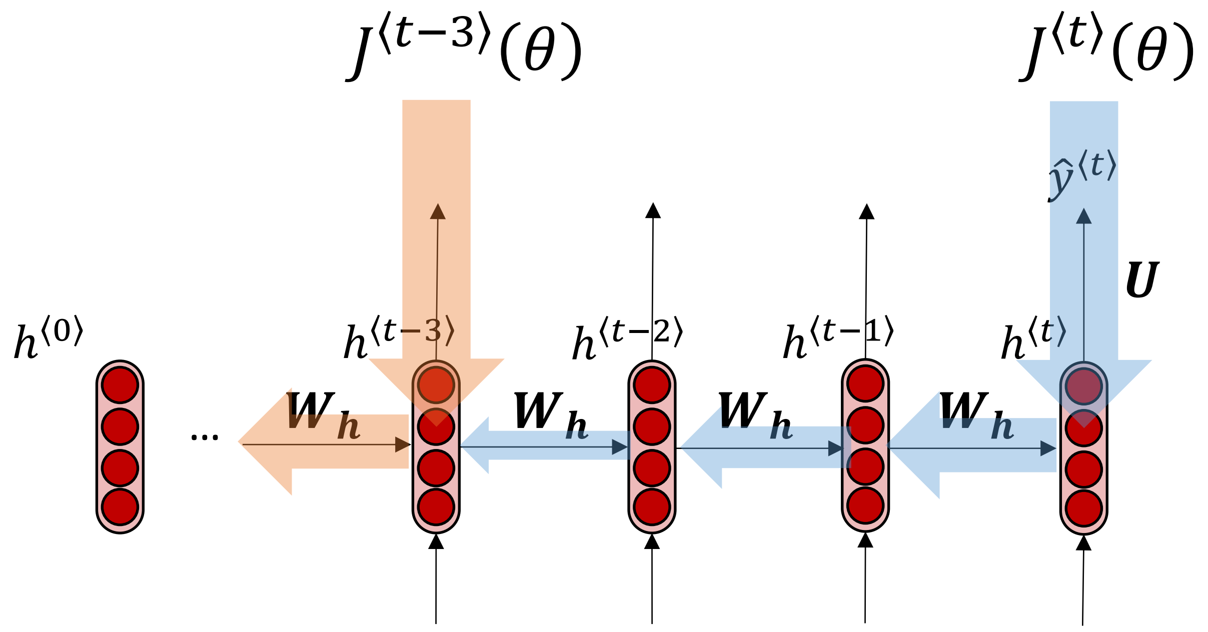
$$\frac{\partial J^{\langle i\rangle}}{\partial \boldsymbol{h}^{\langle j\rangle}} = \frac{\partial J^{\langle i\rangle}}{\partial \boldsymbol{h}^{\langle i\rangle}} \cdot \frac{\partial \boldsymbol{h}^{\langle i\rangle}}{\partial \boldsymbol{h}^{\langle i-1\rangle}} \cdot \frac{\partial \boldsymbol{h}^{\langle i-1\rangle}}{\partial \boldsymbol{h}^{\langle i-2\rangle}} \cdots \frac{\partial \boldsymbol{h}^{\langle j+1\rangle}}{\partial \boldsymbol{h}^{\langle j\rangle}}$$

$$= \frac{\partial J^{\langle i\rangle}}{\partial \boldsymbol{h}^{\langle i\rangle}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{\langle t\rangle}}{\partial \boldsymbol{h}^{\langle t-1\rangle}} = \frac{\partial J^{\langle i\rangle}}{\partial \boldsymbol{h}^{\langle i\rangle}} \prod_{j<t\leq i} \boldsymbol{W_h} = \frac{\partial J^{\langle i\rangle}}{\partial \boldsymbol{h}^{\langle j\rangle}} \boldsymbol{W}_{\boldsymbol{h}}^{\ell}$$

If $\boldsymbol{W_h}$ is small, then the gradient propagated to ℓ steps back becomes exponentially small, as ℓ becomes large!

Problems with RNN-LM: Vanishing gradient

- Why is vanishing gradient a problem?



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

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

Effect of vanishing gradient on RNN-LM

step $i = 7$

- **Example:** 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 ______

step $j \gg 7$

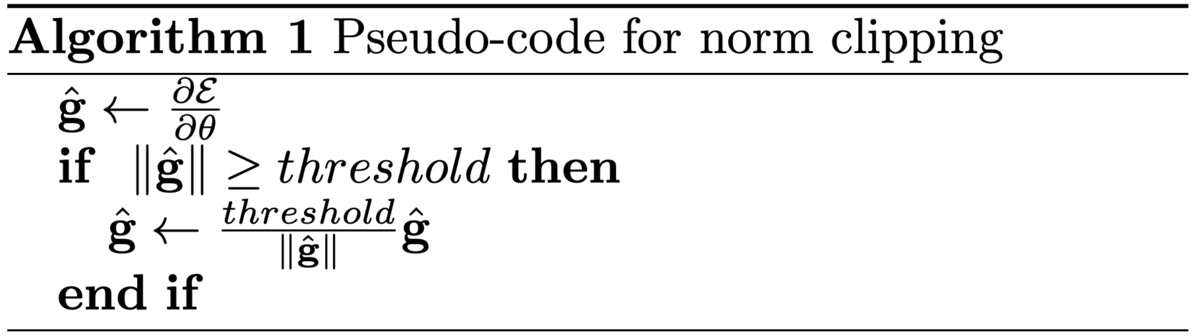
- To learn from this training example, the RNN-LM needs to model the **dependency** between "tickets" on the 7th step and the target word "tickets" at the end.

- But if the gradient is small, the model can't learn this dependency
- the model is unable to predict similar ***long-distance dependencies*** at test time

Adapted from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Opposite Issue: Exploding gradient

- If the gradient becomes too big, then the SGD update step becomes too big
- This can cause **bad updates**: we take too large a step and reach a weird and bad parameter configuration (with large loss)
- This will result in Inf or NaN in the model
- **Solution**: 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**
```

Table from: Pascanu et al. (2013) https://proceedings.mlr.press/v28/pascanu13.pdf

How to fix vanishing gradient?

- Exploding gradient is easier so solve than vanishing gradient
- Main problem of the latter: it's too difficult for the RNN to learn to **preserve information over many timesteps**.
- In vanilla RNN, the hidden state is constantly being rewritten

$$\boldsymbol{h}^{\langle t\rangle} = g(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{h}^{\langle \boldsymbol{t-1}\rangle} + \boldsymbol{W}_{e}\boldsymbol{e}^{\langle t\rangle} + b_1)$$

- **Idea**:
- How about an RNN with separate memory? -- Long short-term memory (LSTM)
- More advanced: Creating direct and linear pass-through connections in model--Attention, residual connections etc.

Adapted from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Overview

- Language Modeling
- Neural Language Models
- Recurrent Neural Networks for LM
- Evaluate LMs
- **Long Short-Term Memory RNNs (LSTMs)**

How to fix the vanishing gradient problem?

- Main problem: it’s too difficult for the RNN to preserve information over many timesteps.
- Because in vanilla RNN the hidden state is constantly being rewritten

$$\boldsymbol{h}^{\langle t\rangle} = g(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{h}^{\langle \boldsymbol{t-1}\rangle} + \boldsymbol{W}_e\boldsymbol{e}^{\langle t\rangle} + b_1)$$

- **Idea**: Design an RNN with separate memory added, besides the constantly updated hidden state

Long Short-Term Memory RNNs (LSTMs)

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997; and a modern version with crucial improvement from Gers etal.(2000)

- Only started to be recognized as promising through the work of S’s student Alex Graves in 2006

联结主义 vs. 符号主义

- Hist work: CTC(*connectionist* temporal classification) for speech recognition

- But only really became well-known after Geoffrey Hinton brought it to Google in 2013

Adapted from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Core Design of LSTMs

- Each step has two states: hidden state $\boldsymbol{h}^{\langle t \rangle}$ and cell state $\boldsymbol{c}^{\langle t \rangle}$
  - They are vectors of same length n
  - The cell $\boldsymbol{c}^{\langle t \rangle}$ stores **long-term** information
  - Can read, erase, and write from/to the cell; like RAM in computer

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

Adapted from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Long Short-Term Memory (LSTM)

$$\boldsymbol{i}^{\langle t\rangle} = \sigma(W_i \boldsymbol{h}^{\langle t-1\rangle} + U_i \boldsymbol{x}^{\langle t\rangle} + b_i)$$

Input gate: determines how much of the input should be added to the current cell

$$\boldsymbol{f}^{\langle t\rangle} = \sigma(W_f \boldsymbol{h}^{\langle t-1\rangle} + U_f \boldsymbol{x}^{\langle t\rangle} + b_f)$$

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

$$\boldsymbol{o}^{\langle t\rangle} = \sigma(W_o \boldsymbol{h}^{\langle t-1\rangle} + U_o \boldsymbol{x}^{\langle t\rangle} + b_o)$$

Output gate: determines what part of cell should influence the output at current step

$$\tilde{\boldsymbol{c}}^{\langle t\rangle} = \tanh(W_c \boldsymbol{h}^{\langle t-1\rangle} + U_c \boldsymbol{x}^{\langle t\rangle} + b_c)$$

New cell content: new content to be written to cell

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

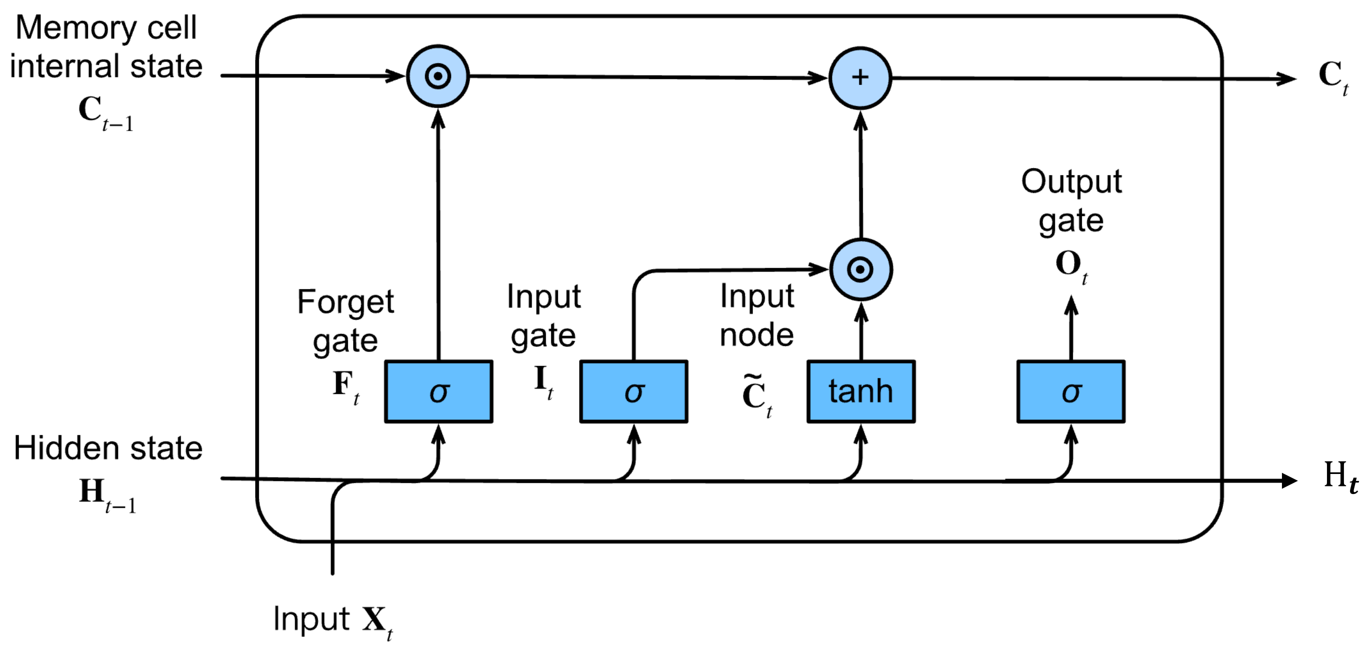
Updated cell state: “forget” some content from the previous cell and write some new content

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

Hidden state: read some content from the cell

$\odot$ for element-wise product

LSTM Computational Graph



Memory cell internal state $\mathbf{C}_{t-1}$
$\odot$
+
$\mathbf{C}_t$
Output gate $\mathbf{O}_t$
Forget gate $\mathbf{F}_t$
Input gate $\mathbf{I}_t$
Input node $\tilde{\mathbf{C}}_t$
σ
tanh
Hidden state $\mathbf{H}_{t-1}$
H_t
Input $\mathbf{X}_t$

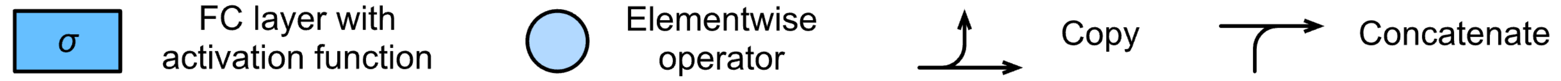


Figure from: https://d2l.ai/chapter_recurrent-modern/lstm.html

LSTM solves vanishing gradients

- LSTM makes it much easier for an RNN to preserve information over many steps
- If the forget gate $\boldsymbol{f}^{\langle t\rangle}$ is set to 1 (for a cell dimension) and the input gate $\boldsymbol{i}^{\langle t\rangle}$ set to 0, then the information (of that cell dimension) is preserved indefinitely.

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

LSTMs: History of Success

- In 2013–2015, **LSTMs** started achieving state-of-the-art results
  - Tasks include: language modeling, handwriting recognition, speech recognition, machine translation, parsing, and image captioning
  - LSTMs became the **dominant approach** for most NLP tasks

- For 2019--2023, **Transformers** have become dominant for all tasks
  - For example, in WMT (a Machine Translation conference + competition)
  - WMT2014 0 neural machine translation systems(!)
  - WMT2016 the summary report contains “RNN” 44 times
  - WMT2019: “RNN” 7 times, “Transformer” 105 times

Adapted from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Recap

- **Language Model**: Model for predicting next word
- **Recurrent Neural Network**: A family of neural networks that
  - Take sequential input of any length
  - Apply the same weights on each step
- RNNs $\neq$ Language Model
- RNNs are also useful for much more!
- LSTM can overcome the short-comes of vanilla RNNs

To-Do List

- Read Chapter 9 - RNNs and LSTMs, Chapter 8 - Sequence Labeling

References

- Bengio, Y., Ducharme, R., & Vincent, P. (2000). A neural probabilistic language model. *Advances in neural information processing systems*, *13*.
- Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550-1560.
- Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., & Robinson, T. (2013). One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv*:1312.3005.
- Pascanu, R., Mikolov, T., & Bengio, Y. (2013, May). On the difficulty of training recurrent neural networks. In *International conference on machine learning* (pp. 1310-1318). *PMLR*.