

CS310 Natural Language Processing

自然语言处理

Lecture 03 - Recurrent Neural Networks and Language Modeling

Instructor: Yang Xu

主讲人：徐炆

xuyang@sustech.edu.cn

Overview

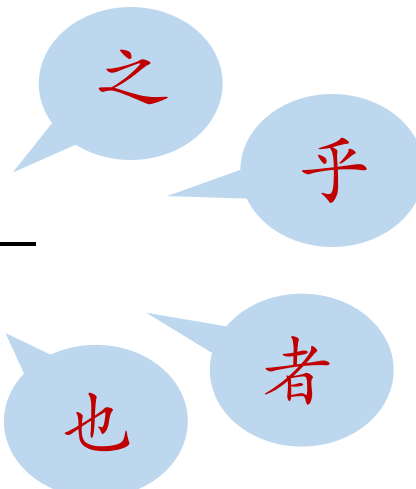
- **Language Modeling**
- Neural Language Models
- Recurrent Neural Networks for LM
- Evaluate LMs
- Long Short-Term Memory RNNs (LSTMs)

Language Modeling

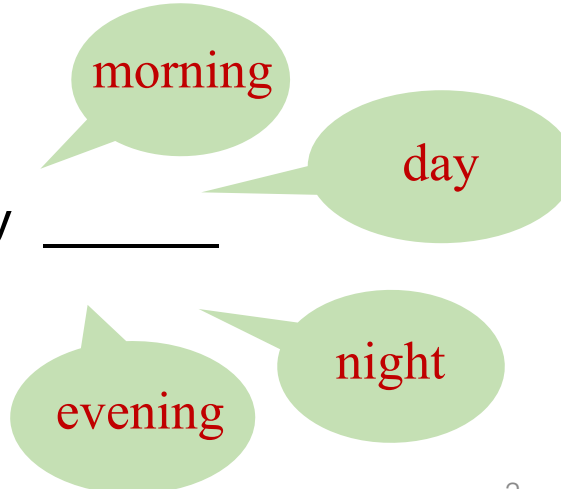
- LM is the task of predicting **what the next word is**, given the preceding ones.
- Formally: given a sequence of words $x^{\langle 1 \rangle}, x^{\langle 2 \rangle}, \dots, x^{\langle t \rangle}$, compute the probability of $x^{\langle t+1 \rangle}$:

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

子曰：学而时习 ____

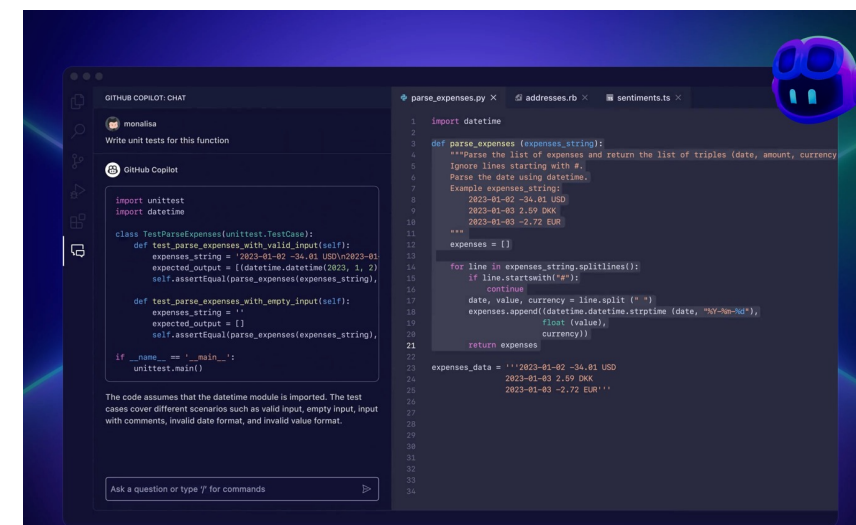
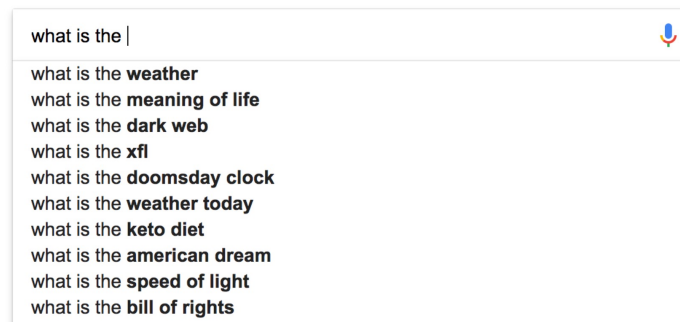


The Sun rises every ____



Motivation for LM

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



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:

$$\begin{aligned} p(X = x, Y = y) &= p(X = x \mid Y = y) \cdot p(Y = y) \\ &= p(Y = y \mid X = x) \cdot p(X = x) \end{aligned}$$

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

How to Learn an LM?

- Pre- Neural network solution: ***n*-gram Language Model**
- **Def.** An *n*-gram is a chunk of *n* consecutive words n-gram是由n个连续词组成的单元

the Sun rises every _____

- **Unigrams** ($n=1$): “the”, “Sun”, “rises”, “every”
 - **Bigrams** ($n=2$): “the Sun”, “Sun rises”, “rises every”
 - **Trigrams** ($n=3$): “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 统计不同n-gram的频率，并使用这些频率来预测下一个词

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

n -grams LM: Markov assumption

马尔可夫假设：该假设认为一个词只依赖于其前面的 $n-1$ 个词

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

$$P(x^{(t+1)} | x^{(1)}, \dots, x^{(t)}) = P(x^{(t+1)} | \underbrace{x^{(t-n+2)}, \dots, x^{(t)}}_{n-1 \text{ words}})$$

Probability of a n -gram

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

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

By the
definition of
conditional
probability

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

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

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 w

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

Example, suppose in the corpus

- “Sun rises every” occurs **1000** times
- “Sun rises every morning” occurs **600** times
 - $\Rightarrow P(\text{morning}|\text{Sun rises every}) = 0.6$
- “Sun rises every day” occurs **300** times
 - $\Rightarrow P(\text{day}|\text{Sun rises every}) = 0.3$

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

对每个词添加一个小的常数 δ 到词频中，避免出现频率为零的情

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

Sparsity problem 2:

用较短的 n -gram来计算概率

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

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

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*

today the _____

Business and financial news

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 **trigram** LM

Today the _____

$P(* | \text{Today the})$

| | |
|---------|-------|
| company | 0.153 |
| bank | 0.153 |
| price | 0.077 |
| italian | 0.039 |
| emirate | 0.039 |
| ... | |


sample

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

Generate text with n -gram LM

Today the price _____

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




| | |
|-----|-------|
| 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 _____

$P(* | \text{price of})$



| | |
|------|-------|
| 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

通过一个固定大小的上下文窗口来处理文本中的词汇 (这个例子中, 一次会处理4个词)

concatenated word embeddings

$$e = \left[e^{(1)}; e^{(2)}; e^{(3)}; e^{(4)} \right] \left. \vphantom{\left[e^{(1)}; e^{(2)}; e^{(3)}; e^{(4)} \right]} \right\} d$$

$4 \times d$

Input tokens: $x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$

(window size = 4)

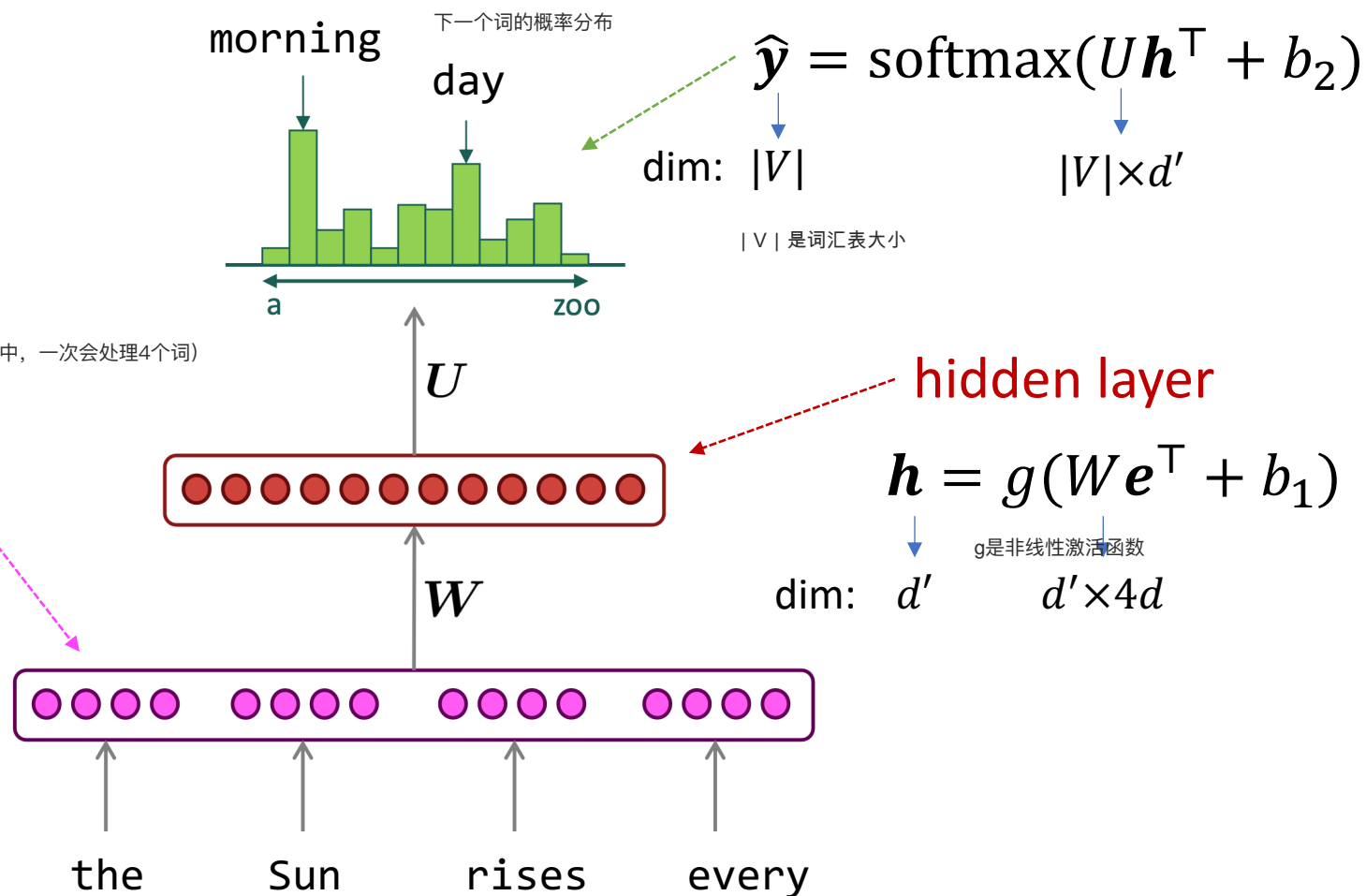
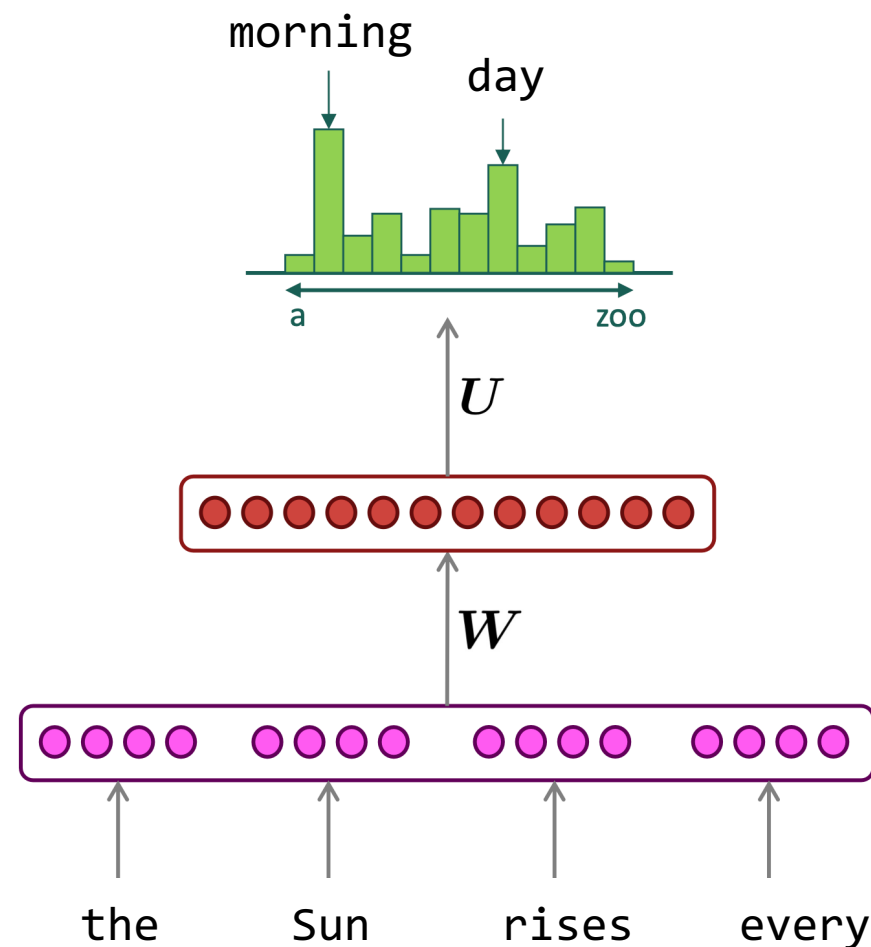


Figure from: <https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/>

Fixed-Window Neural LM

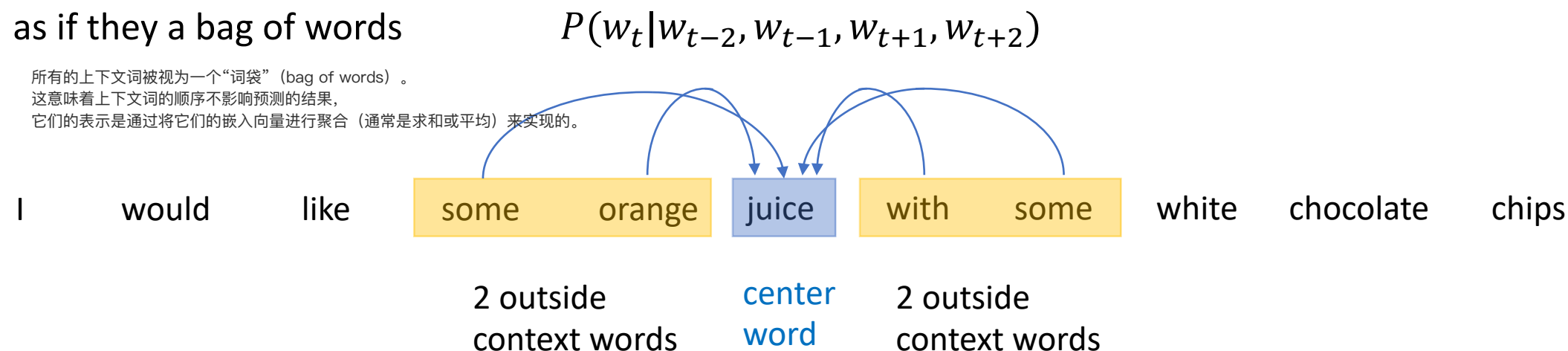
- **Improvements** over n-gram LM:
- No sparsity problem;
- No need to store all observed n-grams
- Remaining **problems**:
- Fixed window can be small
- Increasing window size also increase W
- Need a neural architecture that can process *any input length*



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

Aggregate all context words
as if they a bag of words

所有的上下文词被视为一个“词袋” (bag of words)。
这意味着上下文词的顺序不影响预测的结果，
它们的表示是通过将它们的嵌入向量进行聚合（通常是求和或平均）来实现的。



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**

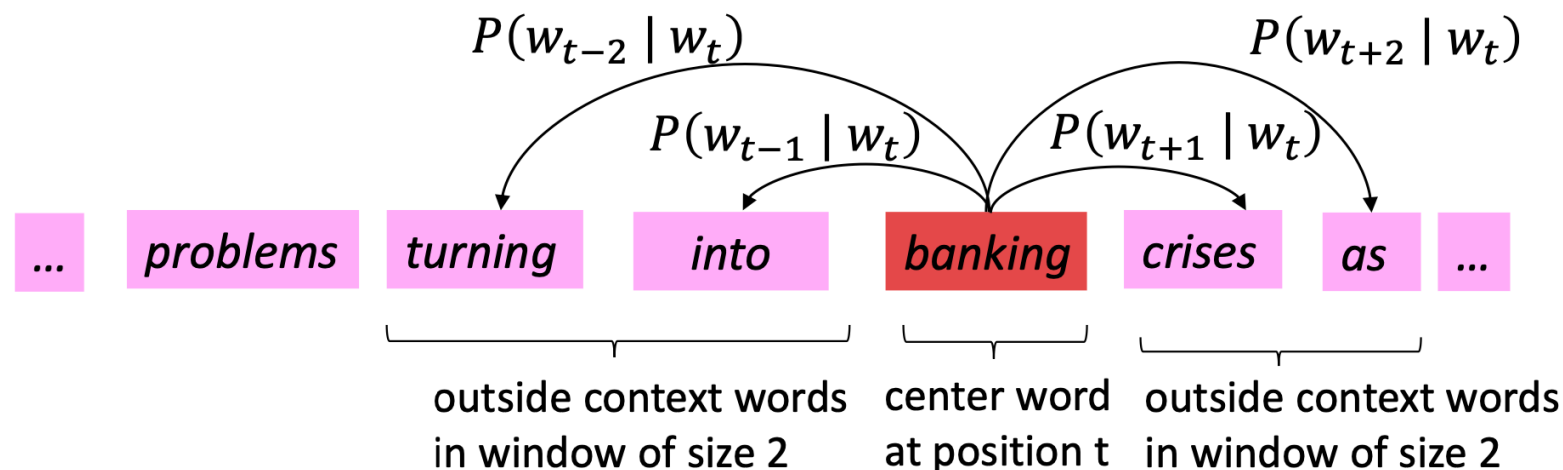
Difference from fixed-window neural-LM:

The prediction is **bidirectional**

同时考虑前后文

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)

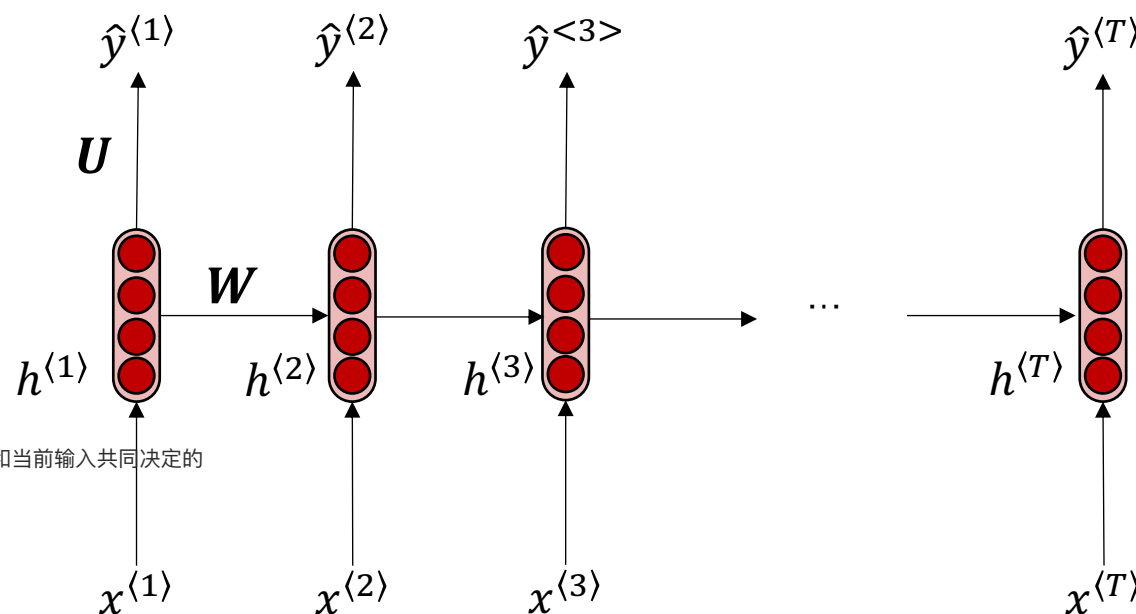
Recurrent Neural Networks

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

在不同的时间步 (time steps) 上反复应用相同的权重矩阵

output sequence
(optional)

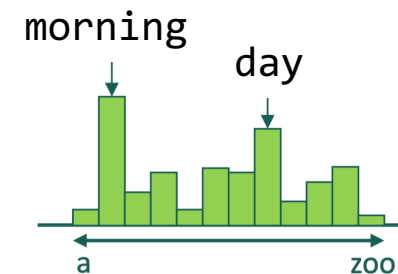
hidden state



RNN的每个时间步都有一个隐藏状态，该隐藏状态是上一时间步的隐藏状态和当前输入共同决定的

Input sequence
(of any length T)

A Simple RNN Language Model



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

output (optional)

$$\hat{y}^{(t)} = \text{softmax}(\mathbf{U}\mathbf{h}^{(t)} + b_2)$$

hidden state

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

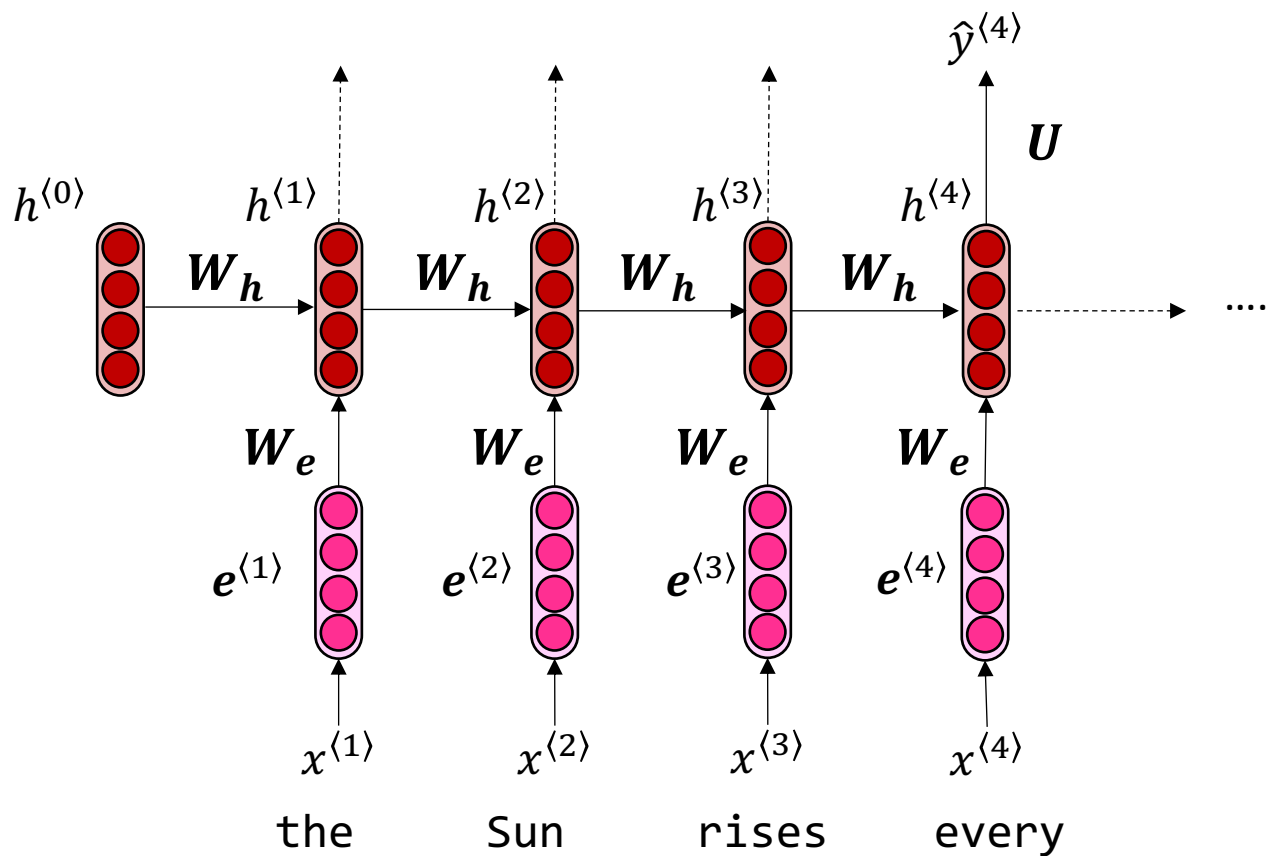
($\mathbf{h}^{(0)}$ is the initial hidden state)

input embedding

$$\mathbf{e}^{(t)} \in \mathbb{R}^d$$

input sequence

$$\mathbf{x}^{(t)}$$



Training an RNN LM: Objective and loss

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

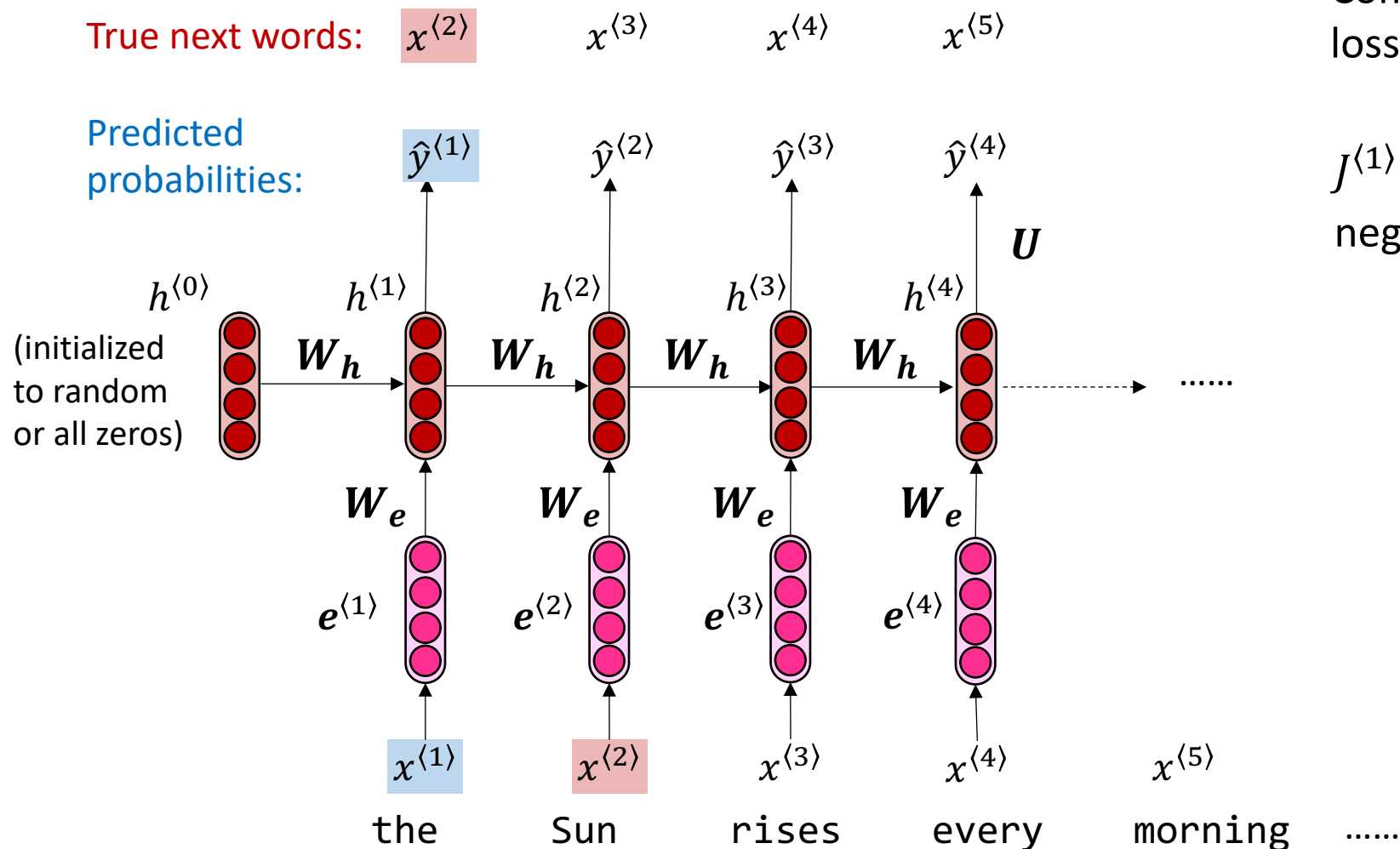
$$J^{(t)}(\theta) = \text{cross-entropy}(\hat{y}^{(t)}, y^{(t)}) = - \sum_{w \in V} \underbrace{y_w^{(t)}}_{\substack{\text{this term is 1 only for } w = y^{(t)}; \\ \text{all zeros for other words}}} \log \hat{y}_w^{(t)} = - \log \hat{y}_{x^{(t+1)}}^{(t)} \quad (\text{negative log likelihood})$$

- Average over entire training set:

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

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

Training an RNN LM: Example

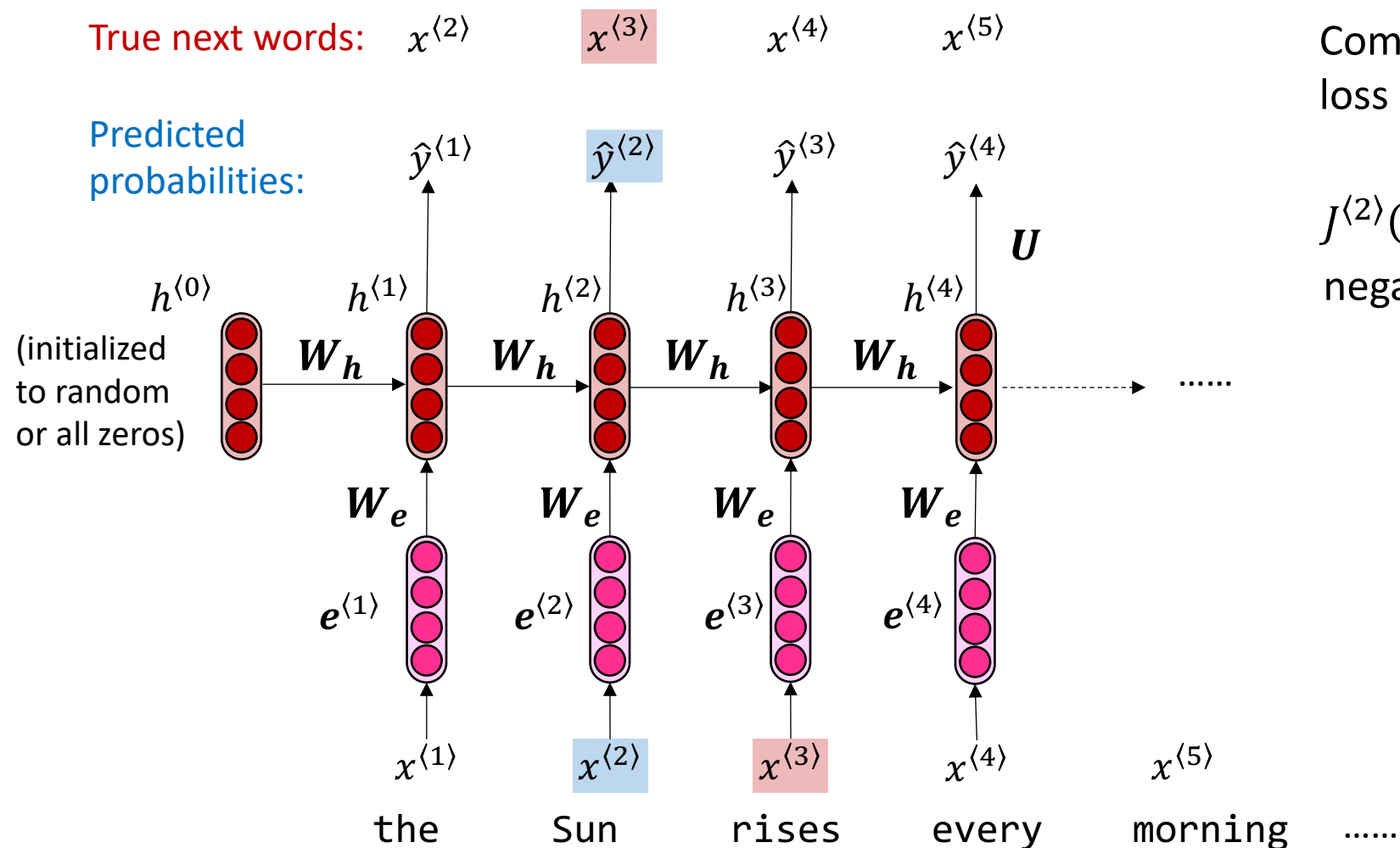


Compute the
loss at step 1:

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

negative log-probability of “Sun”

Training an RNN LM: Example

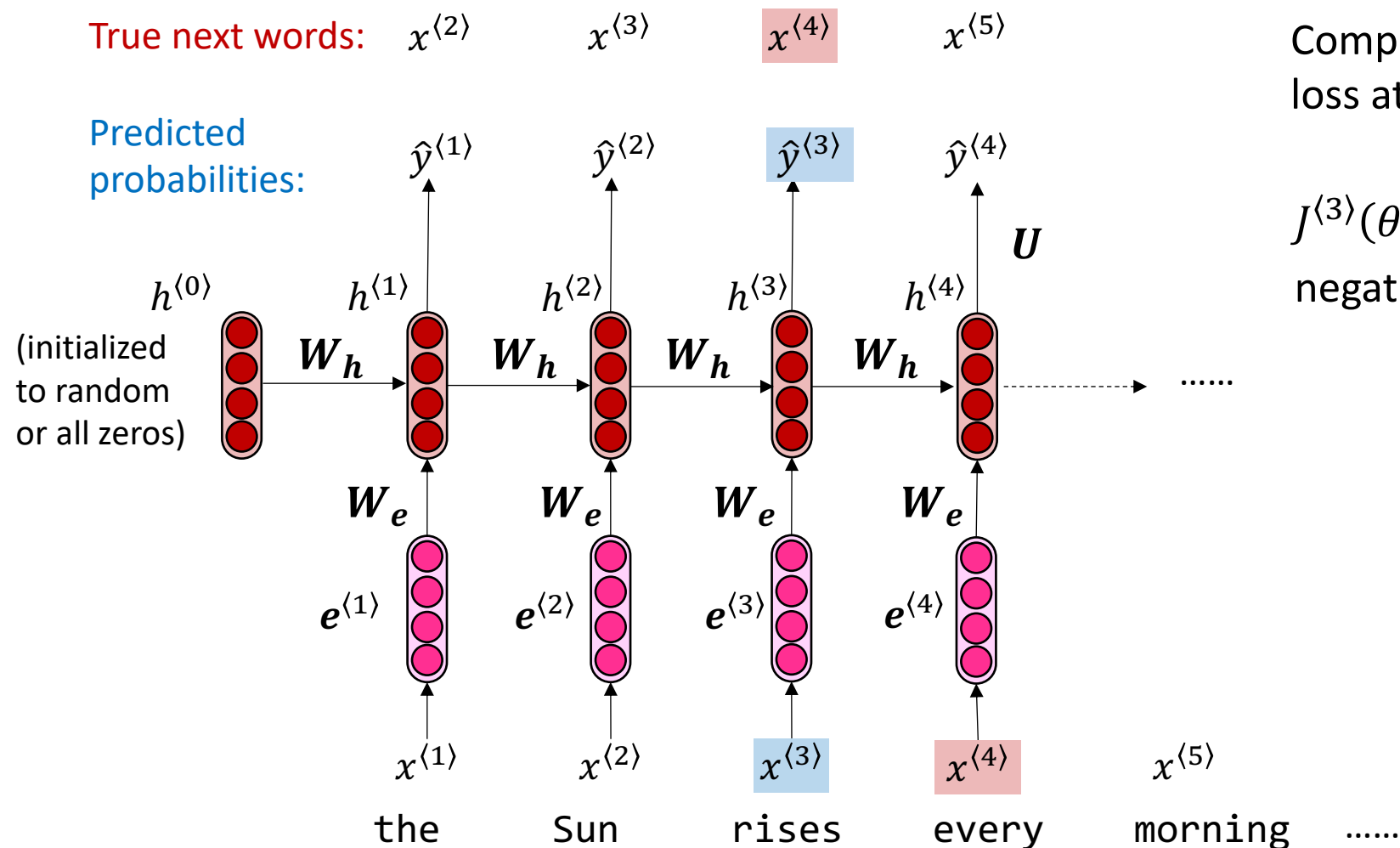


Compute the
loss at step 2:

$$J^{(2)}(\theta) = \text{CE}(\hat{y}^{(2)}, x^{(3)}) = -\log \hat{y}_{\text{rises}}^{(2)},$$

negative log-probability of “rises”

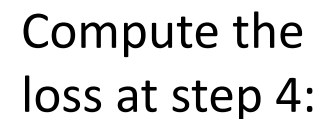
Training an RNN LM: Example



Compute the
loss at step 3:

$$J^{(3)}(\theta) = \text{CE}(\hat{y}^{(3)}, x^{(4)}) = -\log \hat{y}_{\text{every}}^{(3)},$$

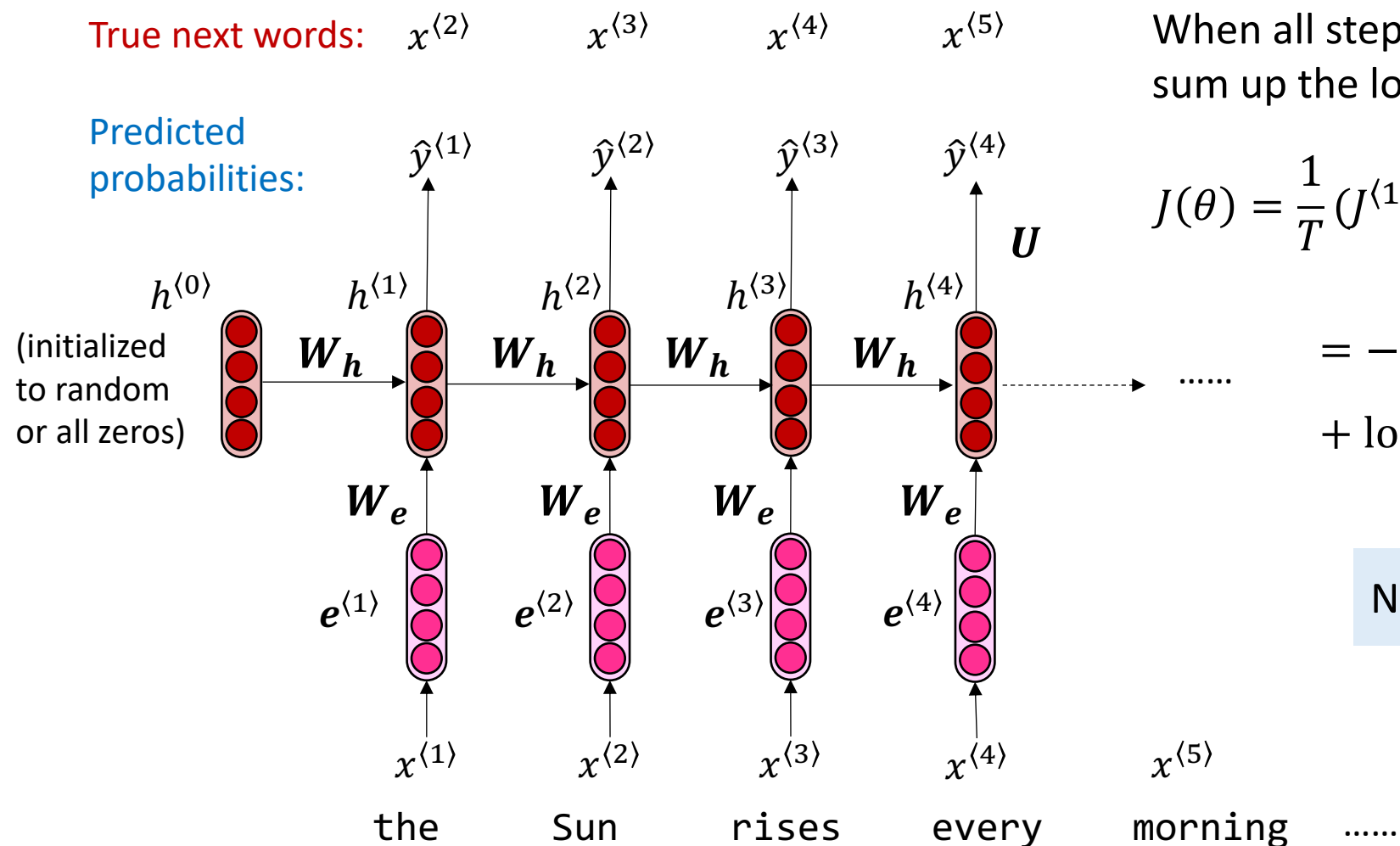
negative log-probability of “every”



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

negative log-probability of “morning”

Training an RNN LM: Example



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

$$J(\theta) = \frac{1}{T} (J^{(1)}(\theta) + J^{(2)}(\theta) + J^{(3)}(\theta) + J^{(4)}(\theta) \dots)$$

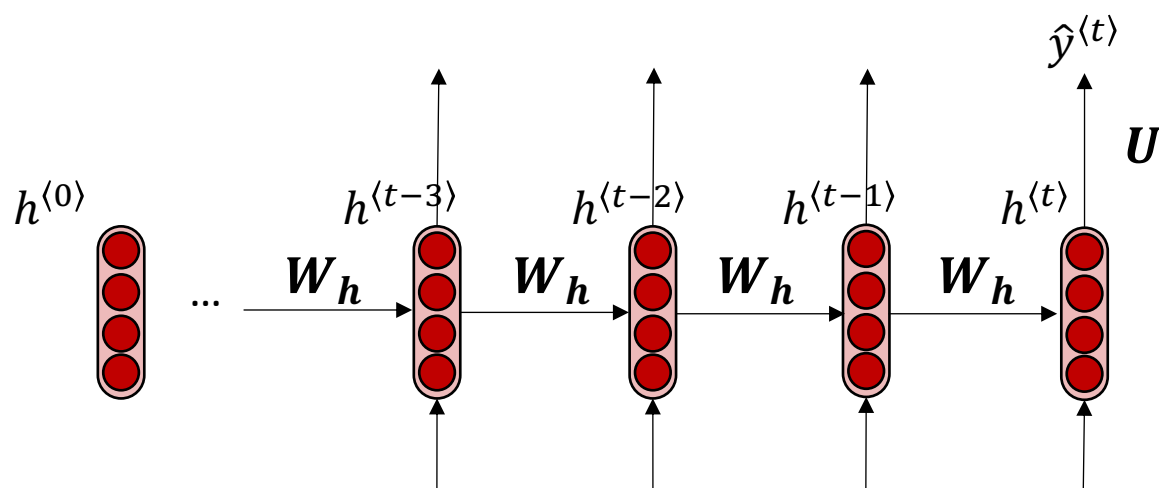
$$= -\frac{1}{T} (-\log \hat{y}_{\text{Sun}}^{(1)} + \log \hat{y}_{\text{rises}}^{(2)} + \log \hat{y}_{\text{every}}^{(3)} + \log \hat{y}_{\text{morning}}^{(4)} + \dots)$$

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

Backpropagation for RNN

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

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

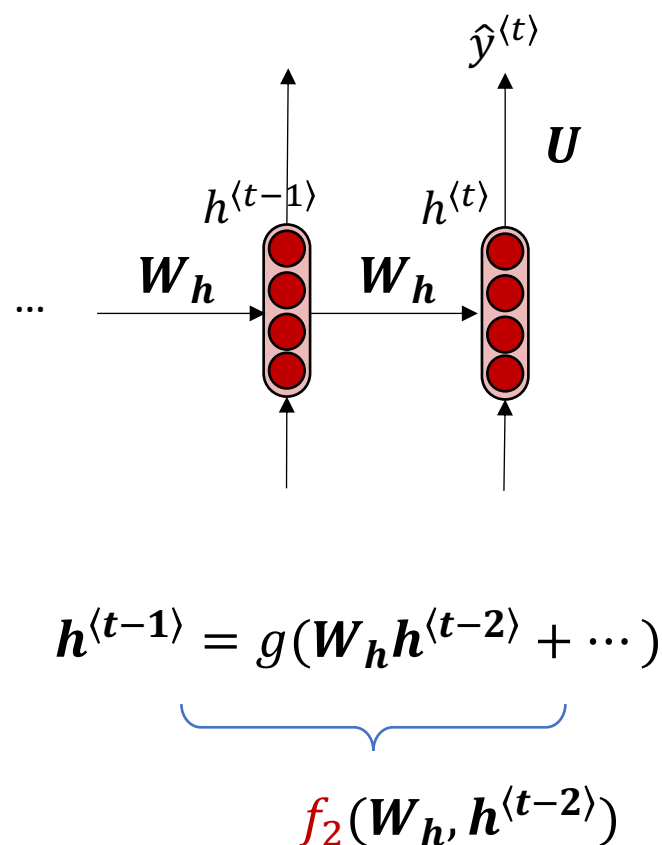


$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial W_h} \Big|_{(i)}$$

gradients contributed by time step i

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

Backpropagation for RNN

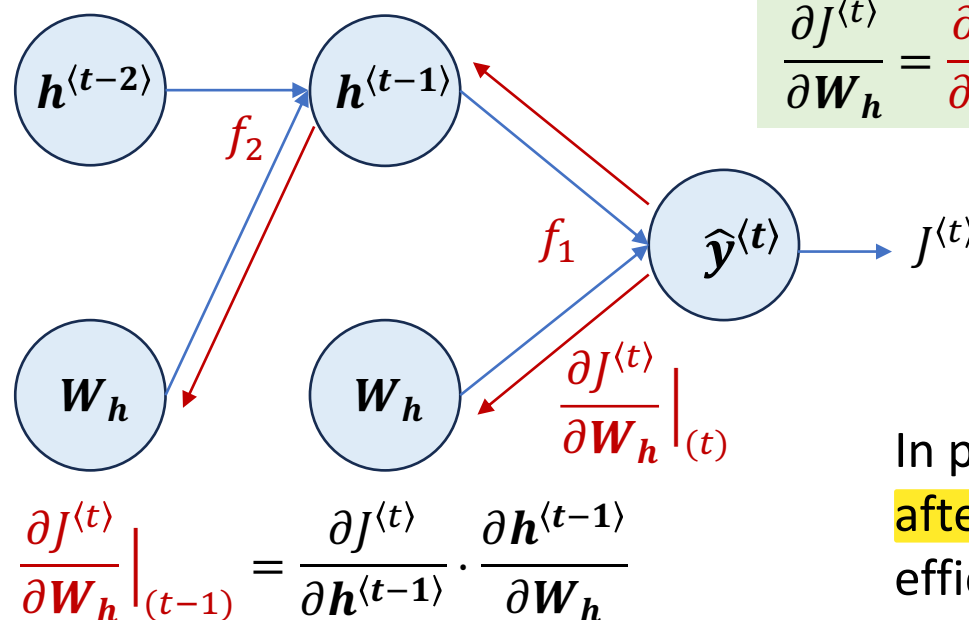


因为权重矩阵在每个时间步都被使用（参数共享），所以 $J(t)$ 对 $W(h)$ 的梯度是所有时间步贡献的总和。

$$\hat{y}^{(t)} = \text{softmax}(U h^{(t)} + b_2) \quad h^{(t)} = g(W_h h^{(t-1)} + W_e e^{(t)} + b_1)$$

$\hat{y}^{(t)} = f_1(W_h, h^{(t-1)})$

损失 $J(t)$ 不仅依赖于当前时间步 t ，还依赖于之前的所有时间步



Backpropagate all the way to the very first step

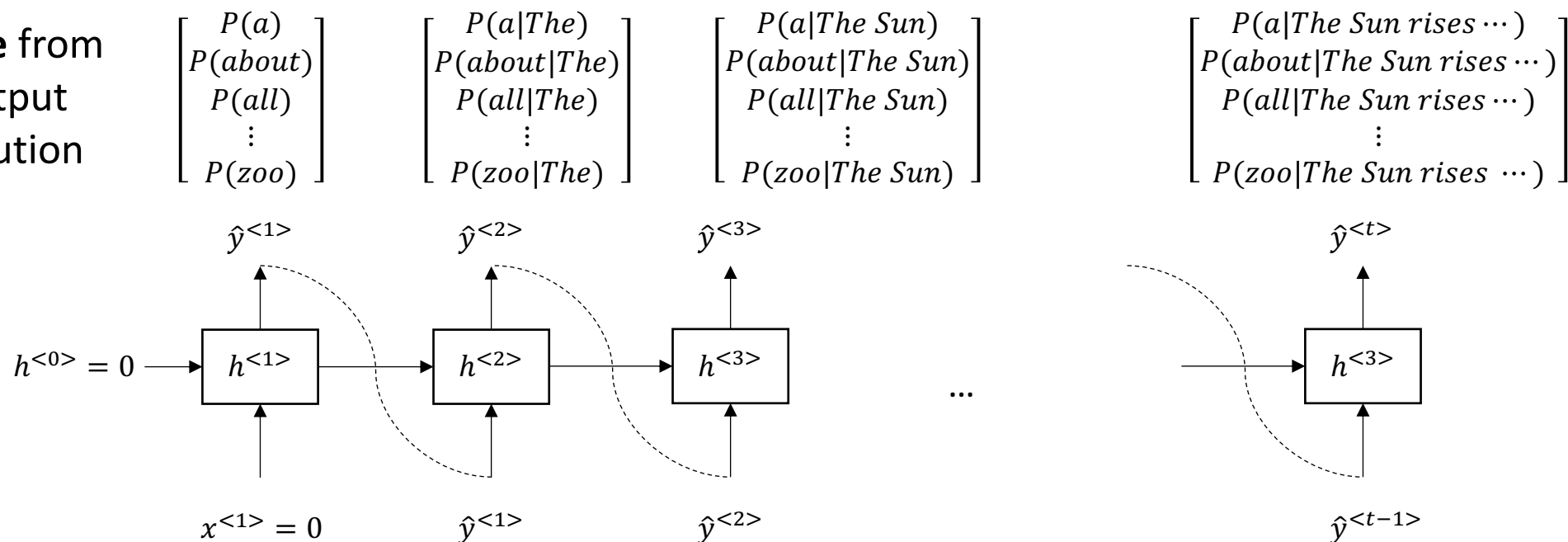
In practice, often **truncated** after **~20 timesteps** for efficiency reasons

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

sample from
the output
distribution



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

```
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 $X_{\text{étale}}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{\text{morph}_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

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

Lemma 0.2. *This is an integer \mathbb{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

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

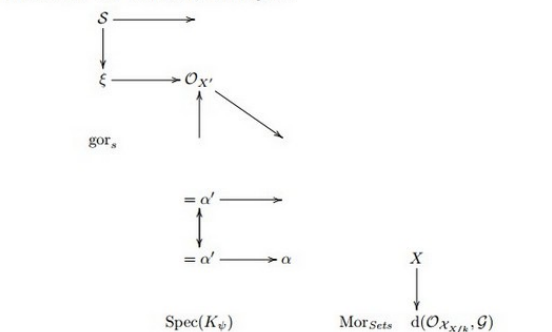
be a morphism of algebraic spaces over S and Y .

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



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 $X = \text{Spec}(R)$ 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"

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_x^{-1}(\mathcal{O}_{X_{\text{étale}}}) \longrightarrow \mathcal{O}_{X_x}^{-1} \mathcal{O}_{X_x}(\mathcal{O}_{X_x}^{\overline{\mathcal{F}}})$$

is an isomorphism of covering of \mathcal{O}_{X_x} . 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_x} 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 | | Perplexity |
|---|---------------------------|---------------|--------|------------|
| | | [hours] | [CPUs] | |
| 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

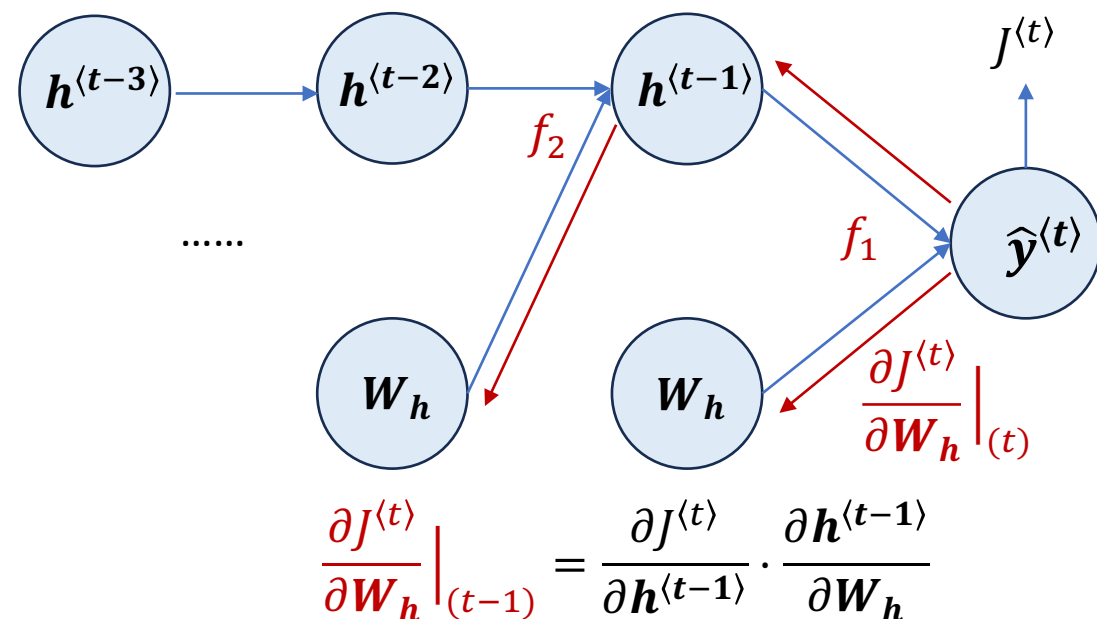
- Vanishing gradient issue

$$\frac{\partial J^{(t)}}{\partial \mathbf{W}_h} \Big|_{(t-2)} = \frac{\partial J^{(t)}}{\partial \mathbf{h}^{(t-1)}} \cdot \frac{\partial \mathbf{h}^{(t-1)}}{\partial \mathbf{h}^{(t-2)}} \cdot \frac{\partial \mathbf{h}^{(t-2)}}{\partial \mathbf{W}_h}$$

$$\frac{\partial J^{(t)}}{\partial \mathbf{W}_h} \Big|_{(t-3)} = \frac{\partial J^{(t)}}{\partial \mathbf{h}^{(t-1)}} \cdot \frac{\partial \mathbf{h}^{(t-1)}}{\partial \mathbf{h}^{(t-2)}} \cdot \frac{\partial \mathbf{h}^{(t-2)}}{\partial \mathbf{h}^{(t-3)}} \cdot \frac{\partial \mathbf{h}^{(t-3)}}{\partial \mathbf{W}_h}$$

.....

Becomes a long chain of products



Problems with RNN-LM: Vanishing gradient

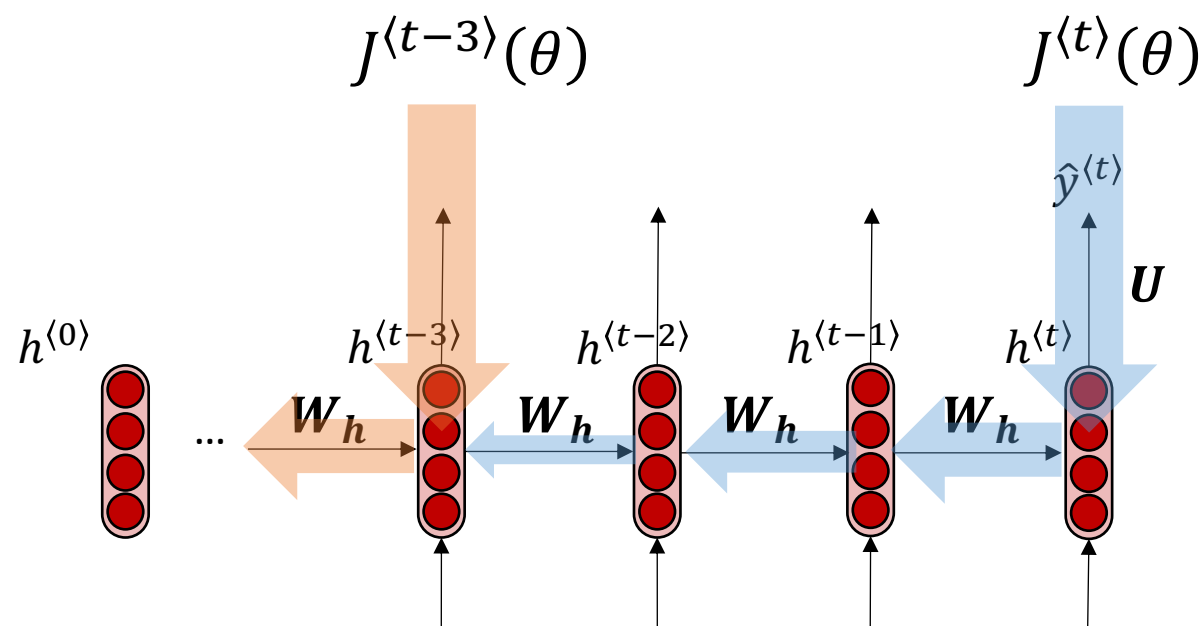
- Recall: $\mathbf{h}^{(t)} = g(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_e \mathbf{e}^{(t)} + b_1)$
- if g is an identity function, $g(x) = x$, then by chain rule: $\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} = g' \cdot \mathbf{W}_h = \mathbf{W}_h$
- Consider the loss at step i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

$$\begin{aligned} \frac{\partial J^{(i)}}{\partial \mathbf{h}^{(j)}} &= \frac{\partial J^{(i)}}{\partial \mathbf{h}^{(i)}} \cdot \frac{\partial \mathbf{h}^{(i)}}{\partial \mathbf{h}^{(i-1)}} \cdot \frac{\partial \mathbf{h}^{(i-1)}}{\partial \mathbf{h}^{(i-2)}} \cdots \frac{\partial \mathbf{h}^{(j+1)}}{\partial \mathbf{h}^{(j)}} \\ &= \frac{\partial J^{(i)}}{\partial \mathbf{h}^{(i)}} \prod_{j < t \leq i} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} = \frac{\partial J^{(i)}}{\partial \mathbf{h}^{(i)}} \prod_{j < t \leq i} \mathbf{W}_h = \frac{\partial J^{(i)}}{\partial \mathbf{h}^{(j)}} \mathbf{W}_h^\ell \end{aligned}$$

If \mathbf{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 \Rightarrow 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  
   $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$   缩放到范数是阈值大小  
end if
```

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

How to fix vanishing gradient?

- Exploding gradient is easier to solve than vanishing gradient
- Main problem of the latter: it's too difficult for the RNN to learn to preserve information over many timesteps.
RNN很难学会在多个时间步上保持信息

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

$$h^{(t)} = g(W_h h^{(t-1)} + W_e e^{(t)} + b_1)$$

每次更新 $h^{(t)}$ 时, $h^{(t-1)}$ 的信息会被“重写”, 并且通过激活函数 (如 tanh 或 sigmoid) 压缩。

- **Idea:**

带有单独记忆的

可以选择性地记住或遗忘信息, 从而更好地捕捉长距离依赖

- 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.

在模型中创建直接且线性的传递连接——注意力机制 (Attention)、残差连接 (residual connections)

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

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

- **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 et al.(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 $\mathbf{h}^{\langle t \rangle}$ and cell state $\mathbf{c}^{\langle t \rangle}$
 - They are vectors of same length n
 - The cell $\mathbf{c}^{\langle t \rangle}$ 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

通过遗忘门、输入门和输出门控制信息的读取、擦除和写入

门的值是动态计算的，基于当前上下文，范围在 [0,1] 之间

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

Long Short-Term Memory (LSTM)

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

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

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

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

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

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

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

New cell content: new content to be written to cell

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

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

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

Hidden state: read some content from the cell

\odot for element-wise product

LSTM Computational Graph

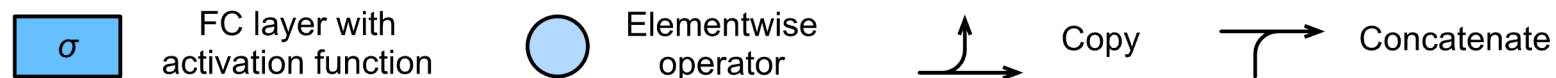
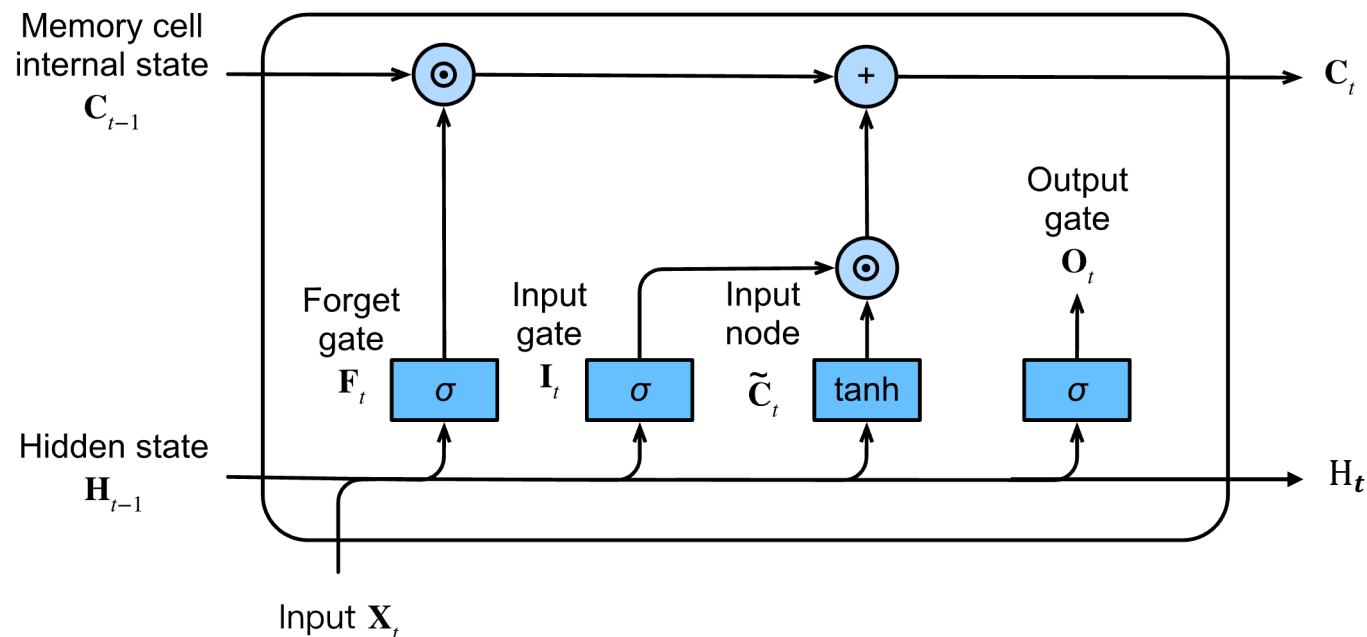


Figure from: https://d2l.ai/chapter_recurrent-modern/lstm.html

LSTM solves vanishing gradients

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

无限期保留

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

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*.