

Words, N-grams and Language Models

Bonan Min

bonanmin@gmail.com

Some slides are based on class materials from Chris Manning, Ralph Grishman, Thien Huu Nguyen

Word Segmentation

Why word segmentation: words are

- The basic unit for applications such as Information retrieval
- The basic building block for syntactic and semantic analysis

Word segmentation: segment a sentence into words

- For English, it's also known as tokenization

Word Segmentation

Examples

- *Dogs eat vegetables.* → *Dogs/eat/vegetables/.*

Words in English can be separated by (mostly) white spaces & punctuations, except for

- Acronyms: *N.A.T.O., i.e., m.p.h, Mr., AT&T*
- Contraction & possessive: *I'm, He'd, don't, Tom's*
- Numbers, dates, IDs: *128,236, +32.56, -40.23, 02/02/94, 02-02-94, D-4, T-1-A, B.1.2*
- ...

Word segmentation

Word segmentation in some languages (e.g., Chinese) is not trivial

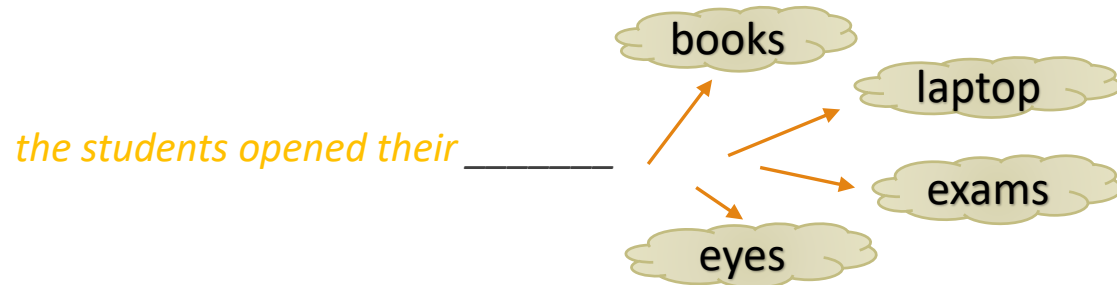
Example: 鱼在长江中游。 → 鱼/在/长江/中/游/。

Methods:

- Rule-based: (forward/backward) maximum matching with a lexicon
 - Unable to resolve ambiguity
- Statistical tagger (e.g., Hidden Markov Models, Conditional Random Fields) trained with annotated corpora
 - Usually done jointly with Part-of-Speech tagging

Language Modeling

Language Modeling is the task of predicting what word comes next given a sequence of previous words.



More formally, given a sequence of words x_1, x_2, \dots, x_i , compute the probability distribution of the following word:

$$P(x_{i+1} | x_i, x_{i-1}, \dots, x_1)$$

where x_{i+1} can be any word in the vocabulary $V = \{w_1, \dots, w_{|V|}\}$

A system that can do this is called a language model

Language Modeling

A language model can also be viewed as a system that assigns probability to a piece of text (i.e., estimating how likely the piece of text is).

For instance, give the text x_1, x_2, \dots, x_N , the probability of this text can be computed based on the probabilities from the language model by:

$$P(x_1, x_2, \dots, x_N) = P(x_1)P(x_2|x_1) \dots P(x_N|x_{N-1}, \dots, x_1)$$

apply chain-rule

$$= \prod_{i=1}^N P(x_i|x_{i-1}, \dots, x_1)$$

Example: $P(\text{John read a book}) = P(\text{John}) P(\text{read}|\text{John}) P(\text{a}|\text{John read}) P(\text{book}|\text{John read a})$

Why Should We Care About Language Modeling?

Language Modeling is a **benchmark task** that helps us **measure our progress** on understanding language

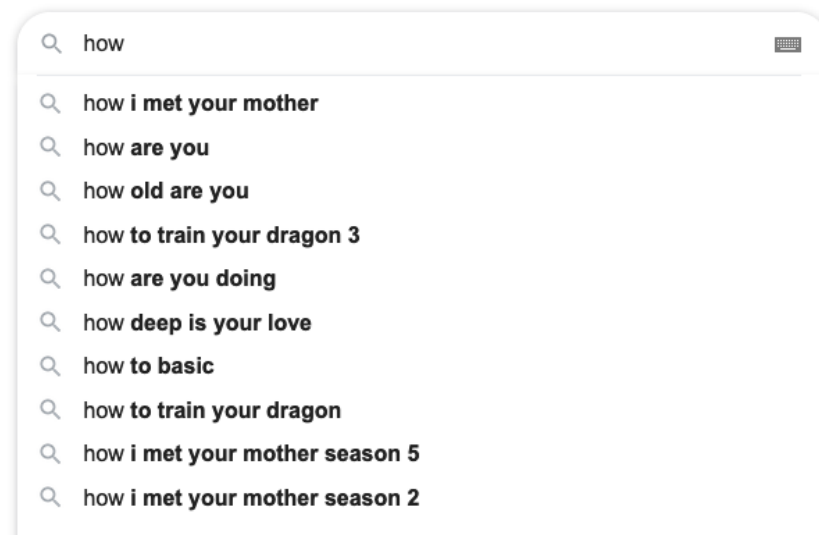
Language Modeling is a **subcomponent** of many NLP tasks, especially those involving **generating text** or **estimating the probability of text**:

- Predictive typing
- Speech recognition
- Handwriting recognition
- Spelling/grammar correction
- Authorship identification
- Machine translation
- Summarization
- Dialogue
- etc.

E.g., MT: select the most likely translated sentence among many possibilities


Language Modeling


Remember this?

The Google logo is displayed in its standard multi-colored font: blue 'G', red 'o', yellow 'o', blue 'g', green 'l', and red 'e'.

Language Modeling

Remember this?



using | 

using **essential oils safely**

using **giveaway**

using **linkedin**

using **innovative products**

using **technology**

using **hydraulic equipment**

using **social media**

using

[See all results for using](#)

N-gram Language Models

the students opened their _____

Question: How do we learn a language model?

Answer (before deep learning): use n-gram language model

Definition: a n-gram is a sequence of n consecutive words

- unigrams: “the”, “students”, “opened”, “their”
- bigrams: “the students”, “students opened”, “opened their”
- trigrams: “the students opened”, “students opened their”
- 4-grams: “the students opened their”

Main idea: collect the statistics about the frequency of the n-gram in some corpus, and use it to estimate the probability

N-gram Language Models

The **simplifying assumption**: x_i only depends on the $n - 1$ preceding words, i.e.,

$$P(x_i | x_{i-1}, \dots, x_1) = P(x_i | \overbrace{x_{i-1}, \dots, x_{i-n+1}}^{n-1 \text{ words}}) \quad (\text{assumption})$$

probability of n-gram \longrightarrow

probability of (n-1)-gram \longrightarrow

$$= \frac{P(x_{i-n+1}, x_{i-n}, \dots, x_i)}{P(x_{i-n+1}, x_{i-n}, \dots, x_{i-1})}$$

Question: how to obtain the probabilities of such n-grams?

Answer: by **counting** their appearance in some large corpus of text:

$$= \frac{\text{count}(x_{i-n+1}, x_{i-n}, \dots, x_i)}{\text{count}(x_{i-n+1}, x_{i-n}, \dots, x_{i-1})} \quad \text{statistical approximation}$$

N-gram Language Models: An Example

Suppose we are learning a 4-gram language model

as the proctor started the clock, the students opened their _____
discard condition on this

For example, suppose that in the corpus:

- “students opened their” occurred 1000 times
- “students opened their books” occurred 400 times
→ $P(\text{books} \mid \text{students opened their}) = 0.4$
- “students opened their exams” occurred 100 times
→ $P(\text{exams} \mid \text{students opened their}) = 0.1$

Should we have
discarded the “proctor”
context?

Sparsity Problems With N-gram Language Models

Use larger n-grams makes predictions more accurate but causes the sparsity problems

Sparsity Problem 1

Problem: What if “students opened their w ” never occurred in data? Then w has probability 0!

(Partial) Solution: Add small δ to the count for every $w \in V$. This is called *smoothing*.

$$P(w|\text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})}$$

Sparsity Problem 2

Problem: What if “students opened their” never occurred in data? Then we can’t calculate probability for *any* w !


(Partial) Solution: Just condition on “opened their” instead. This is called *backoff*.

Note: Increasing n makes sparsity problems worse. Typically we can’t have n bigger than 5.

Back-off

In given corpus, we may have never seen

- Scottish beer drinkers
- Scottish beer eaters



Smoothing gives
same probability

We may have seen “beer drinker” more often than “beer eater”

Better: back-off to bigram

Interpolation

Higher and lower order n-gram models have different strengths and weaknesses

- high-order n-grams are sensitive to more context, but have sparse counts
- low-order n-grams consider only very limited context, but have robust counts

Combine them:

$$\begin{aligned} p_I(w_3|w_1, w_2) = & \lambda_1 p_1(w_3) \\ & \times \lambda_2 p_2(w_3|w_2) \\ & \times \lambda_3 p_3(w_3|w_1, w_2) \end{aligned}$$

Storage Problems With N-gram Language Models

Storage: Need to store count for all n -grams you saw in the corpus.

$$P(\mathbf{w} | \text{students opened their}) = \frac{\text{count}(\text{students opened their } \mathbf{w})}{\text{count}(\text{students opened their})}$$

Uni-gram ($n=1$)	$P(w_i)$	N parameters
Bi-gram ($n=2$)	$P(w_i w_{i-1})$	N^2 parameters
Tri-gram ($n=3$)	$P(w_i w_{i-2} w_{i-1})$	N^3 parameters
Four-gram/quadrigram* ($n=4$)	$P(w_i w_{i-3} w_{i-2} w_{i-1})$	N^4 parameters

Common n -grams (quadrigram are rarely used because it's not practical)

Increasing n or increasing corpus increases model size!

Storage Problems With N-gram Language Models

Storage: Need to store count for all n -grams you saw in the corpus.

$$P(w|\text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})}$$

	Large n	Small n
Pros	More context, helps predicting the next word	fewer parameters, inexpensive to estimate, more reliable parameters even with smaller corpus
Cons	exponentially more parameters, computationally expensive, require a large corpus to estimate parameters (still likely unreliable due to sparsity)	Lack context for predicting next word

N-gram Language Models In Practice

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 _____

get probability
distribution

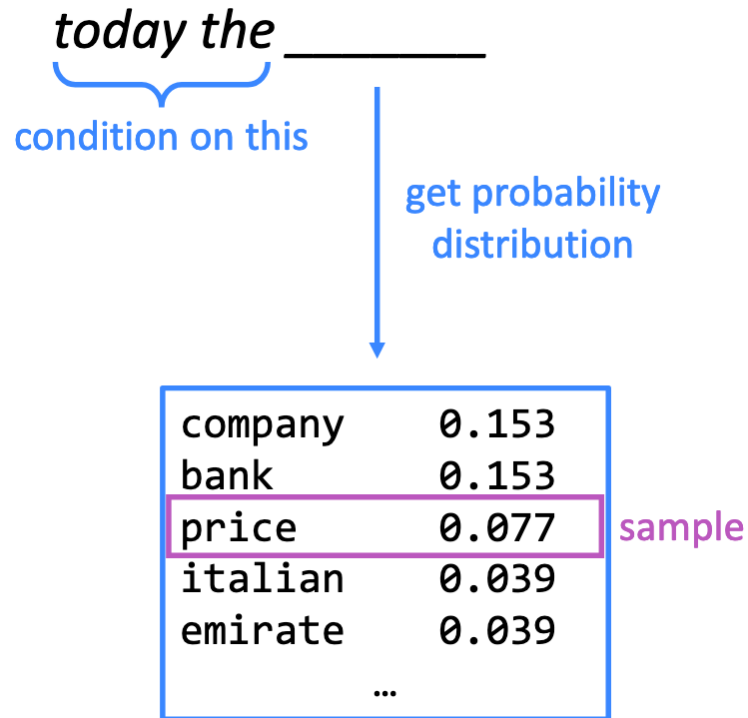
company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
...	

Sparsity problem:
not much granularity
in the probability
distribution

Otherwise, seems reasonable!

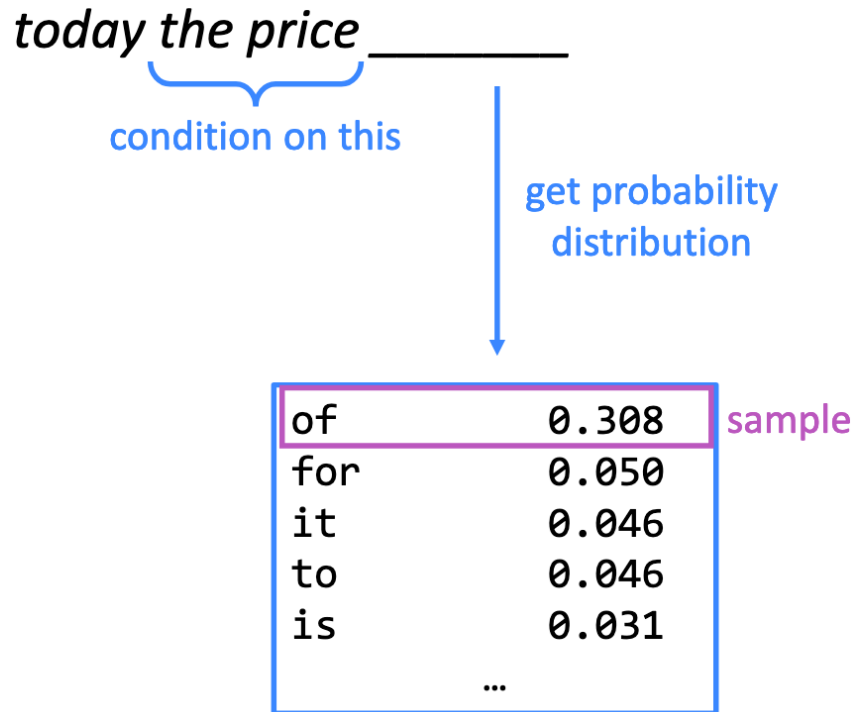
Generating Text With A N-gram Language Model

You can also use a Language Model to generate text.



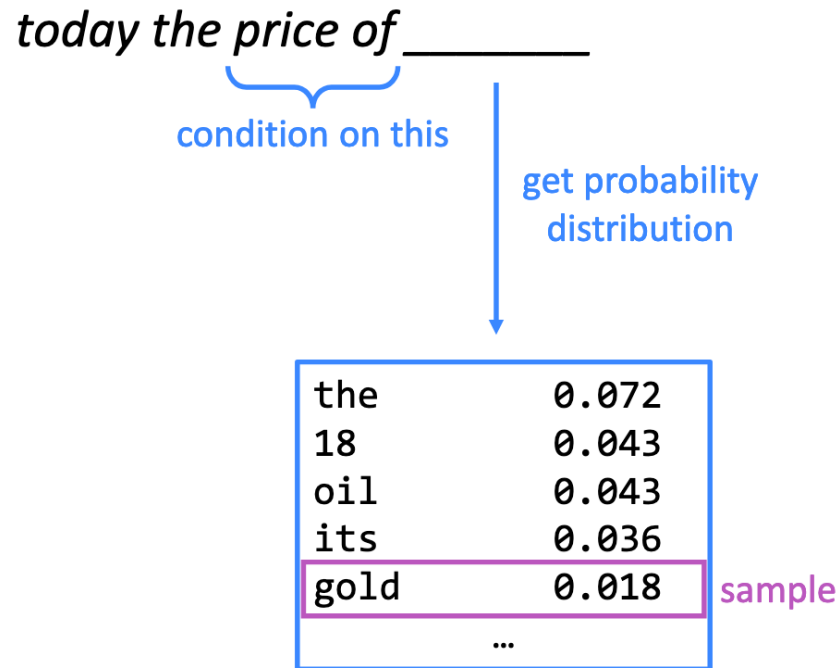
Generating Text With A N-gram Language Model

You can also use a Language Model to generate text.



Generating Text With A N-gram Language Model

You can also use a Language Model to generate text.



Generating Text With A 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 .

Surprisingly grammatical!

...but incoherent. We need to consider more than three words at a time if we want to model language well.

But increasing n worsens sparsity problem, and increases model size...

Neural Language Model

Recall:

- Given the previous words x_{i-1}, \dots, x_1
- We need to compute $P(x_i | x_{i-1}, \dots, x_1)$

How about a window-based neural model?

as the proctor started the clock, the students opened their _____

discard fixed-window

A Fixed-window Neural Language Model

$$\hat{y}_4 = P(x_5 | \text{the students opened their})$$

output distribution

$$y_i = \text{softmax}(Uh + b_2)$$

hidden layer

$$h = f(We + b_1)$$

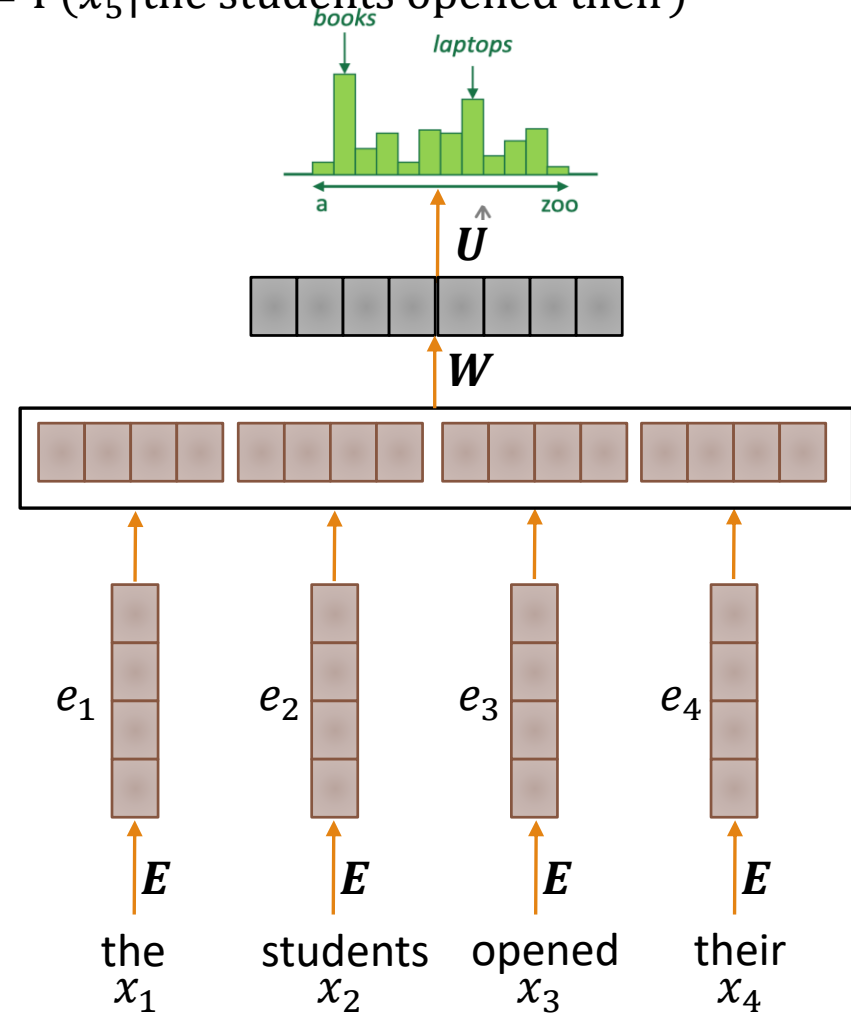
concatenated word embeddings:

$$e = [e_1, e_2, e_3, e_4]$$

word embeddings

words/one-hot vectors

$$x_i \in \{0,1\}^{|V|}$$



A Fixed-window Neural Language Model

$$\hat{y}_4 = P(x_5 | \text{the students opened their})$$

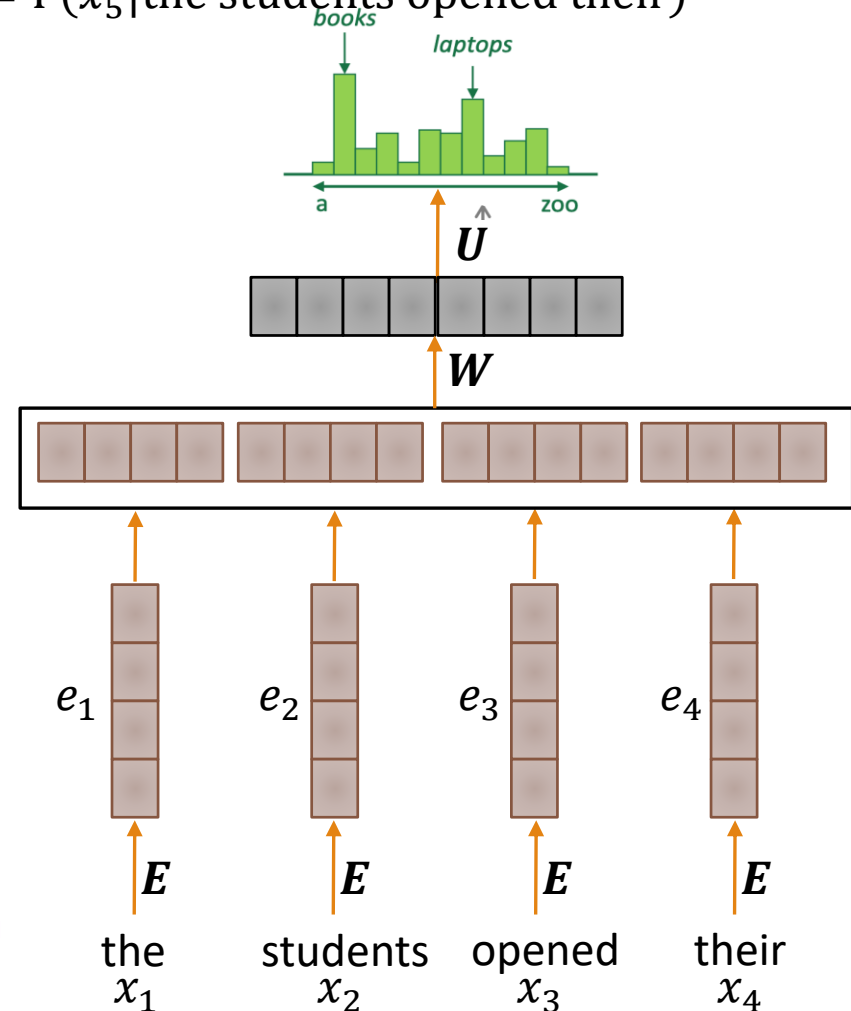
Improvements over n -gram LM:

- No sparsity problem
- Don't need to store all observed n -grams

Remaining problems

- Fixed window is **too small**
- Enlarging window enlarges W
- Window can never be large enough!
- x_1 and x_2 are multiplied by completely different weights in W .
- **No symmetry** in how the inputs are processed.

We need a neural architecture that can process any length input



A RNN Language Model

output distribution

$$y_i = \text{softmax}(Uh_t + b_2)$$

hidden states

$$h_i = \sigma(W_h h_{i-1} + W_e e_i + b_1)$$

h_0 is the initial hidden state

using LSTM or GRU is more common

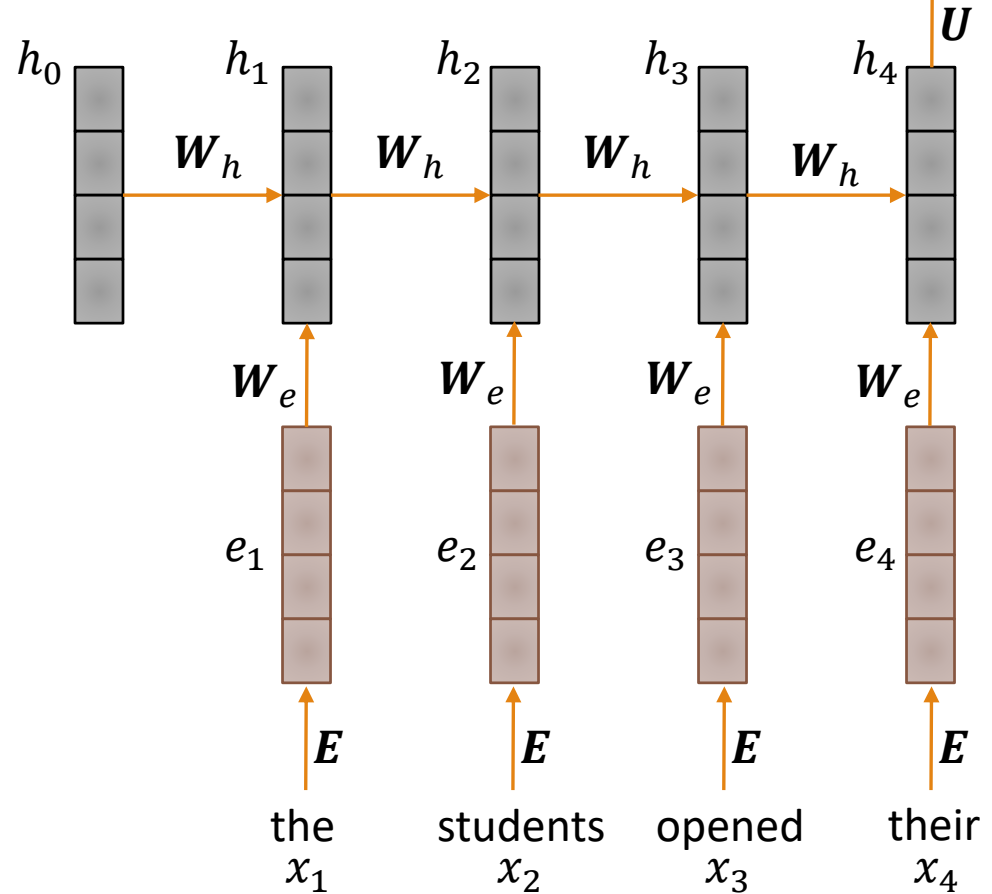
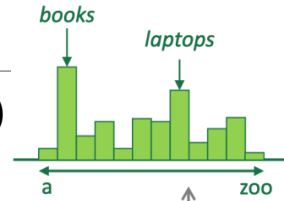
word embeddings

$$e_i = Ex_i$$

words/one-hot vectors

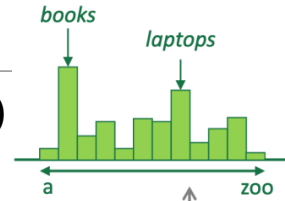
$$x_i \in \{0,1\}^{|V|}$$

$$\hat{y}_4 = P(x_5 | \text{the students opened their})$$



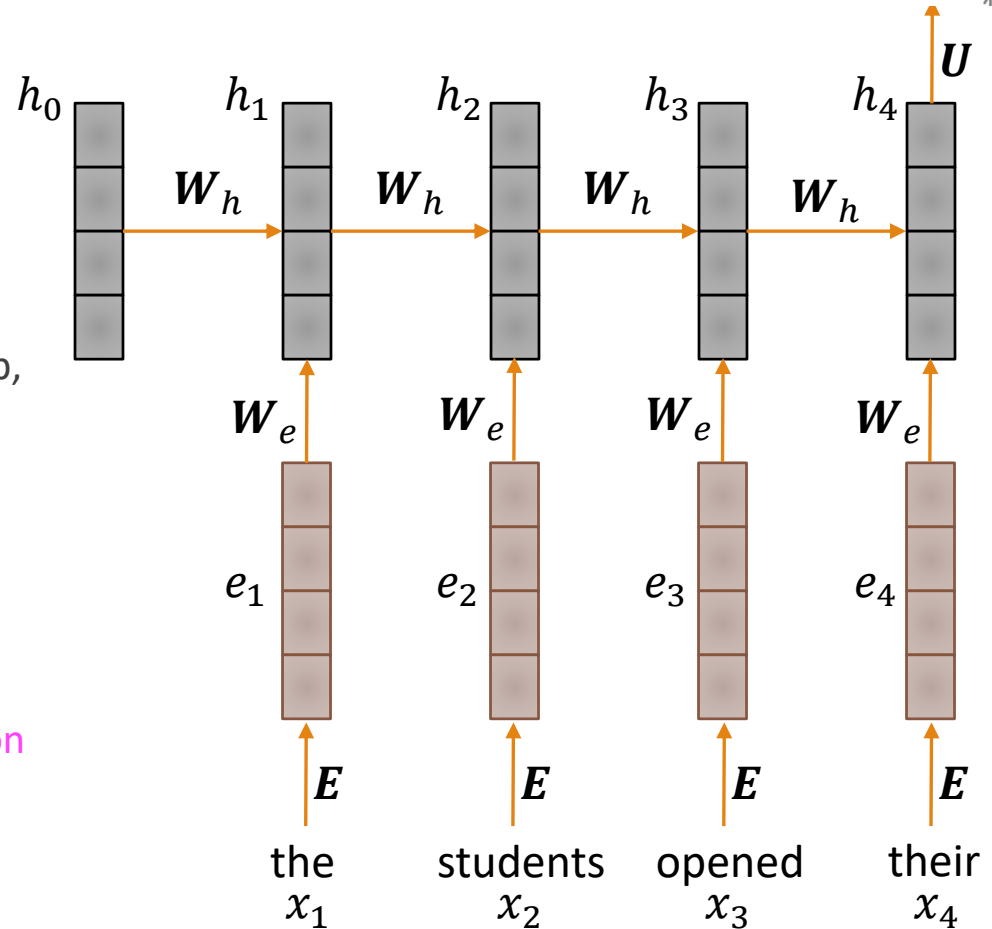
A RNN Language Model

$$\hat{y}_4 = P(x_5 | \text{the students opened their})$$



RNN advantages:

- Can process **any length** input
- Computation for step i can (in theory) use information from **many steps back**
- **Model size doesn't increase** for longer input
- Same weights applied on every timestep, so there is **symmetry** in how inputs are processed.



RNN disadvantages:

- Recurrent computation is **slow**
- In practice, **difficult to access information from many steps back** (although LSTM can help a bit)

Training A RNN Language Model

Get a **big corpus of text** which is a sequence of words x_1, \dots, x_T

Feed into RNN-LM; compute output distribution \hat{y}_i **for every step i**

- i.e., predict probability distribution of every word, given words so far

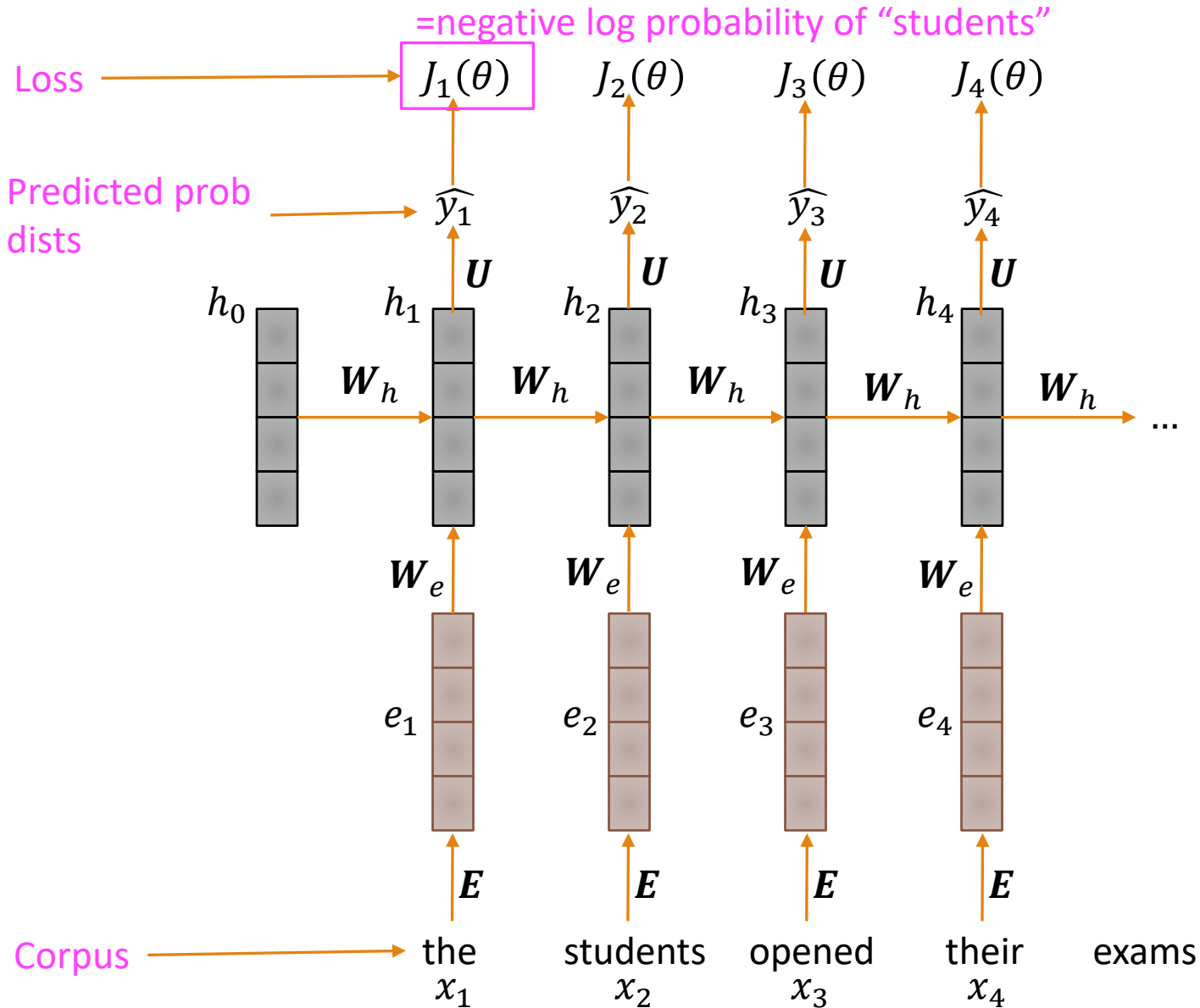
Loss function on step i is **cross-entropy** between predicted probability distribution \hat{y}_i , and the true next word y_i (one-hot distribution for x_{i+1}):

$$J_i(\theta) = CE(y_i, \hat{y}_i) = - \sum_{w \in V} y_i(w) \log \hat{y}_i(w) = - \log \hat{y}_i(x_{i+1})$$

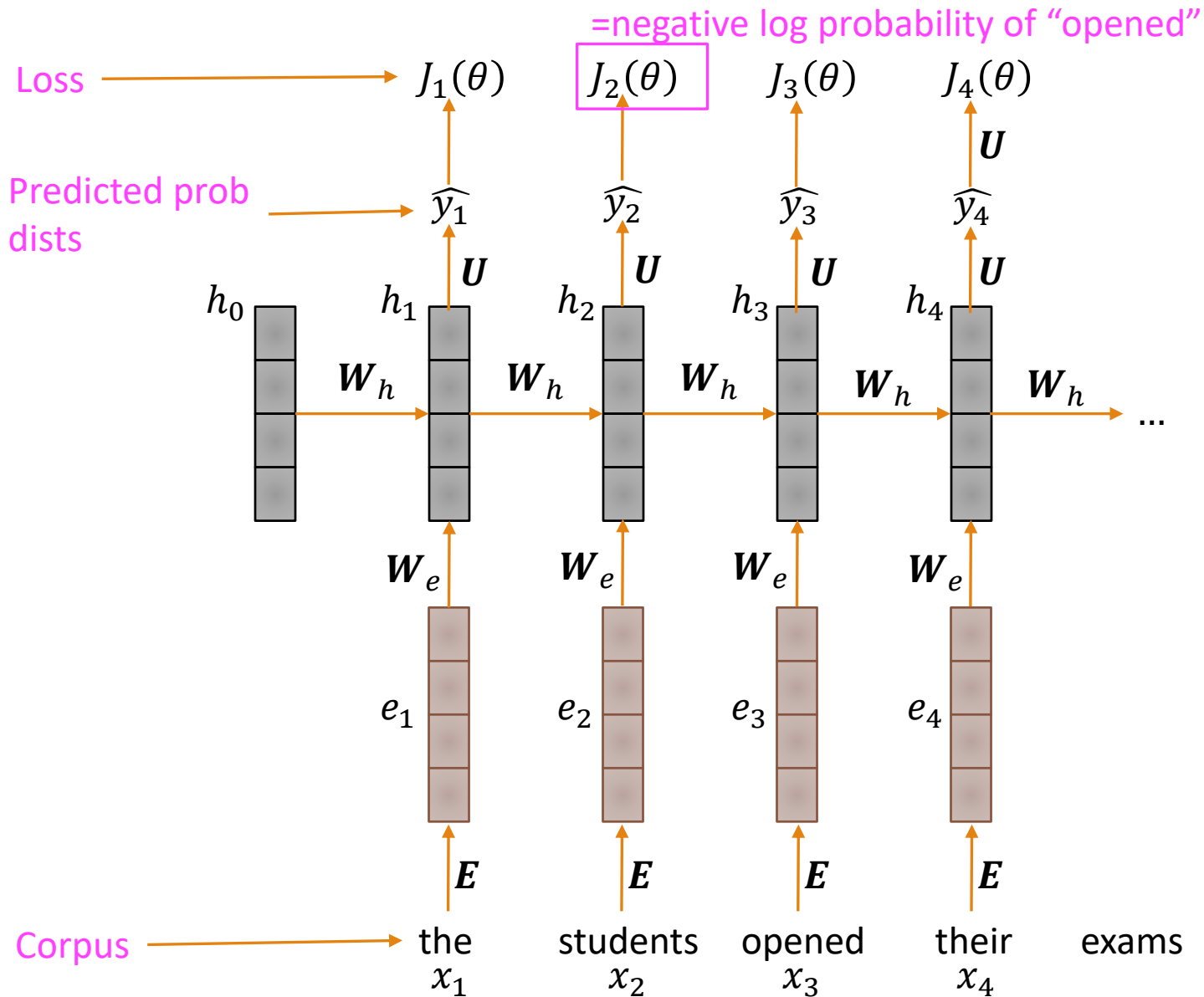
Average this to get **overall loss** for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{i=1}^T J_i(\theta) = \frac{1}{T} \sum_{i=1}^T - \log \hat{y}_i(x_{i+1})$$

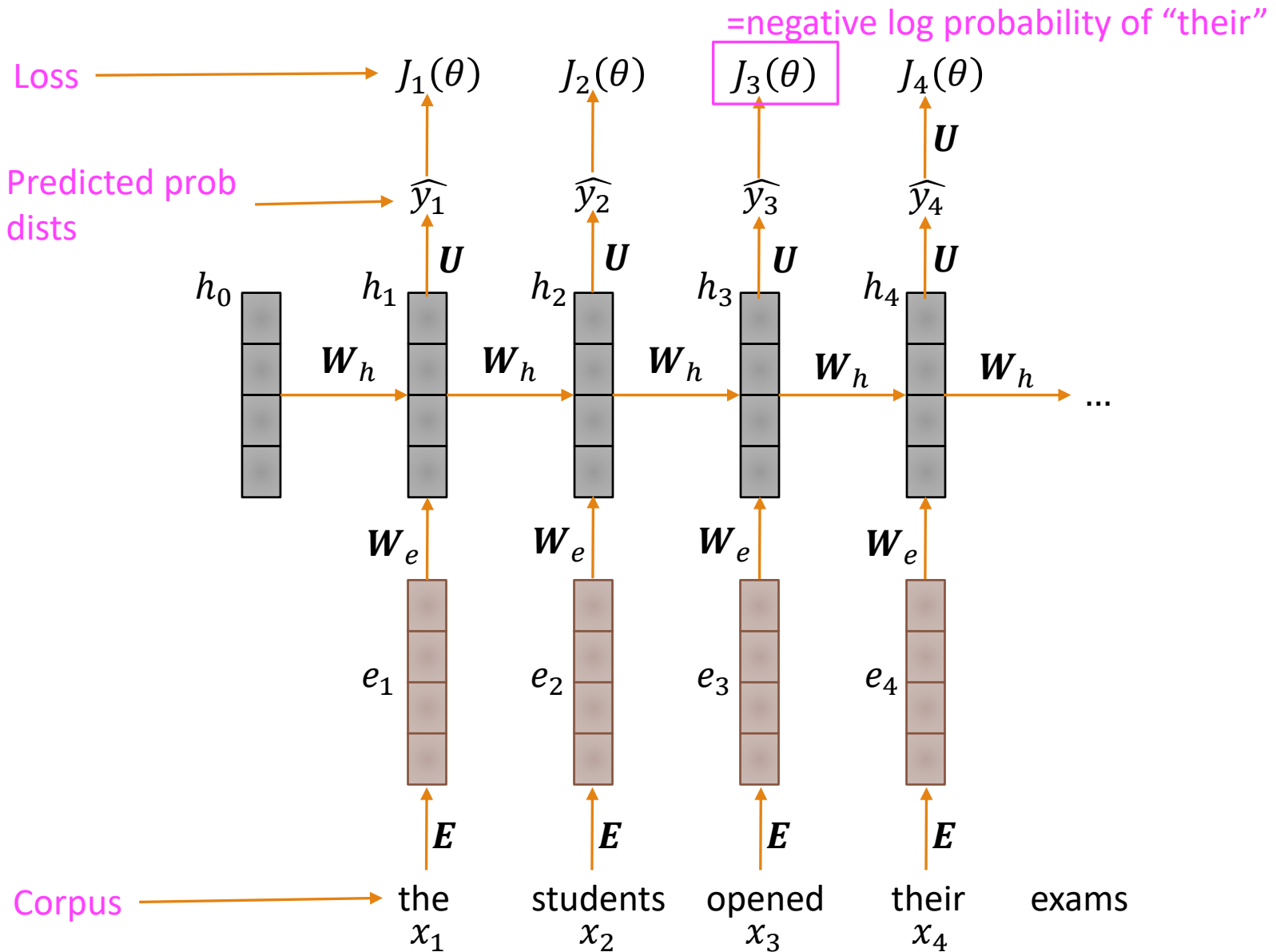
A RNN language model



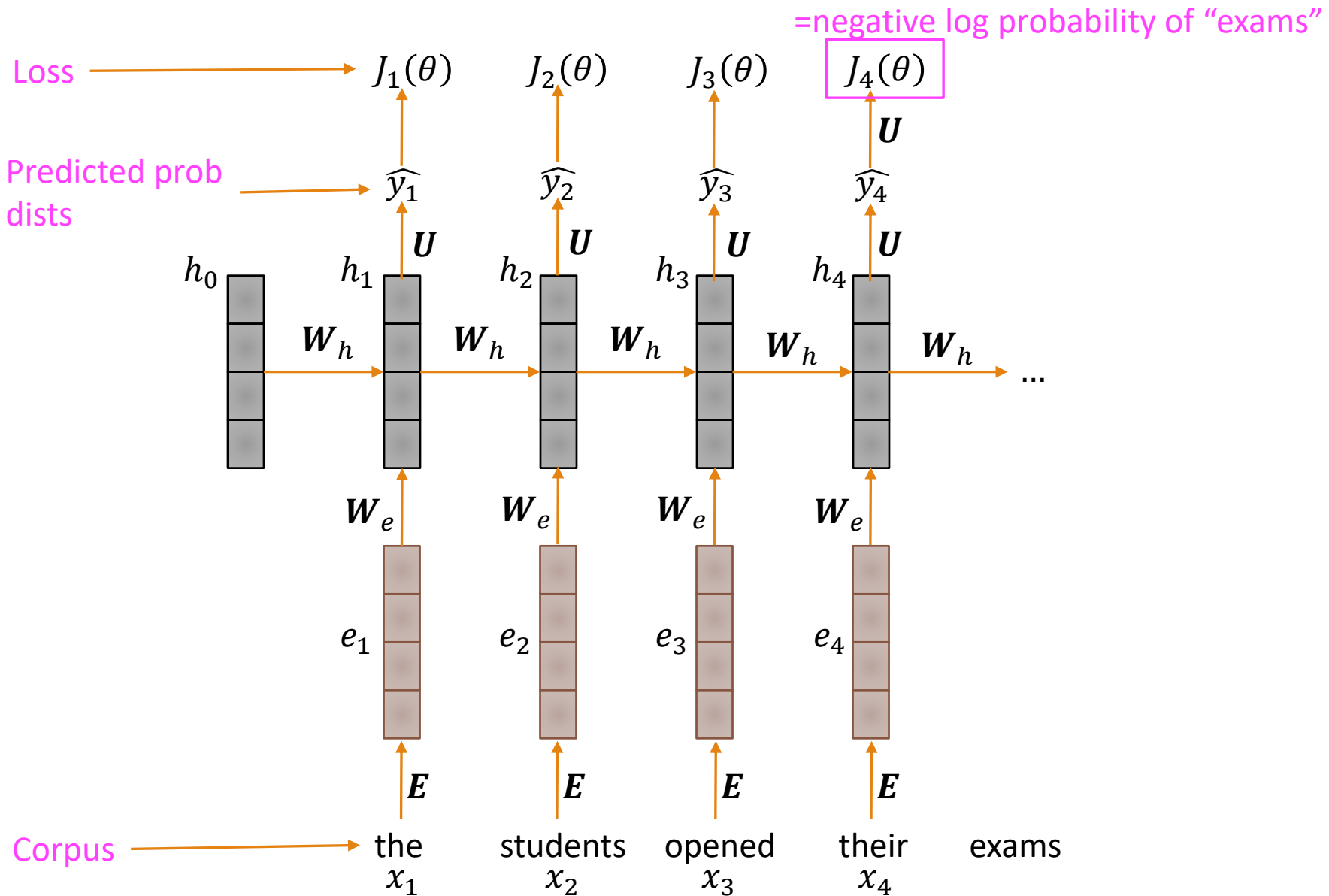
A RNN language model



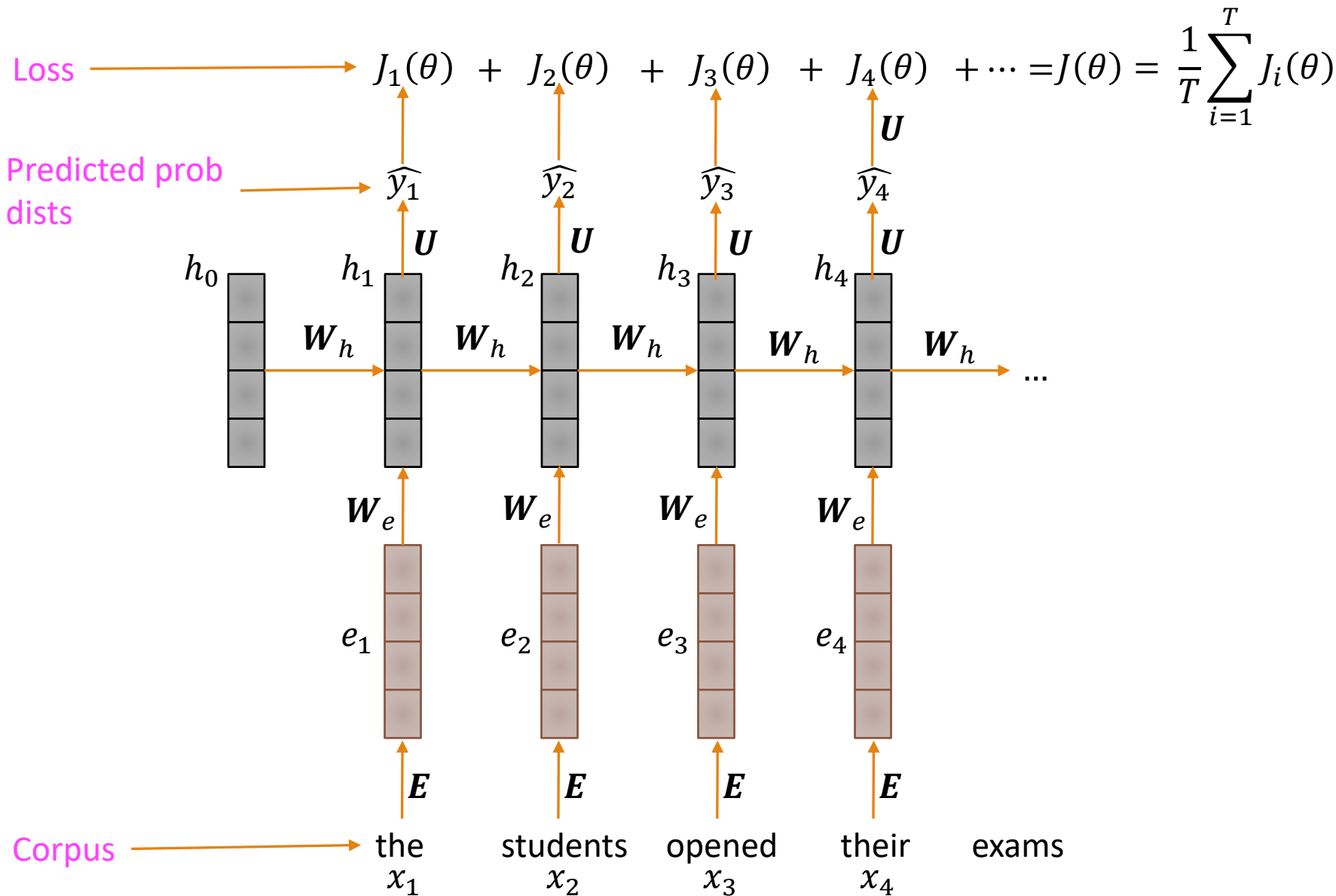
A RNN language model



A RNN language model



A RNN language model



Training A RNN Language Model

However: Computing loss and gradients across **entire corpus** x_1, \dots, x_T is **too expensive!**

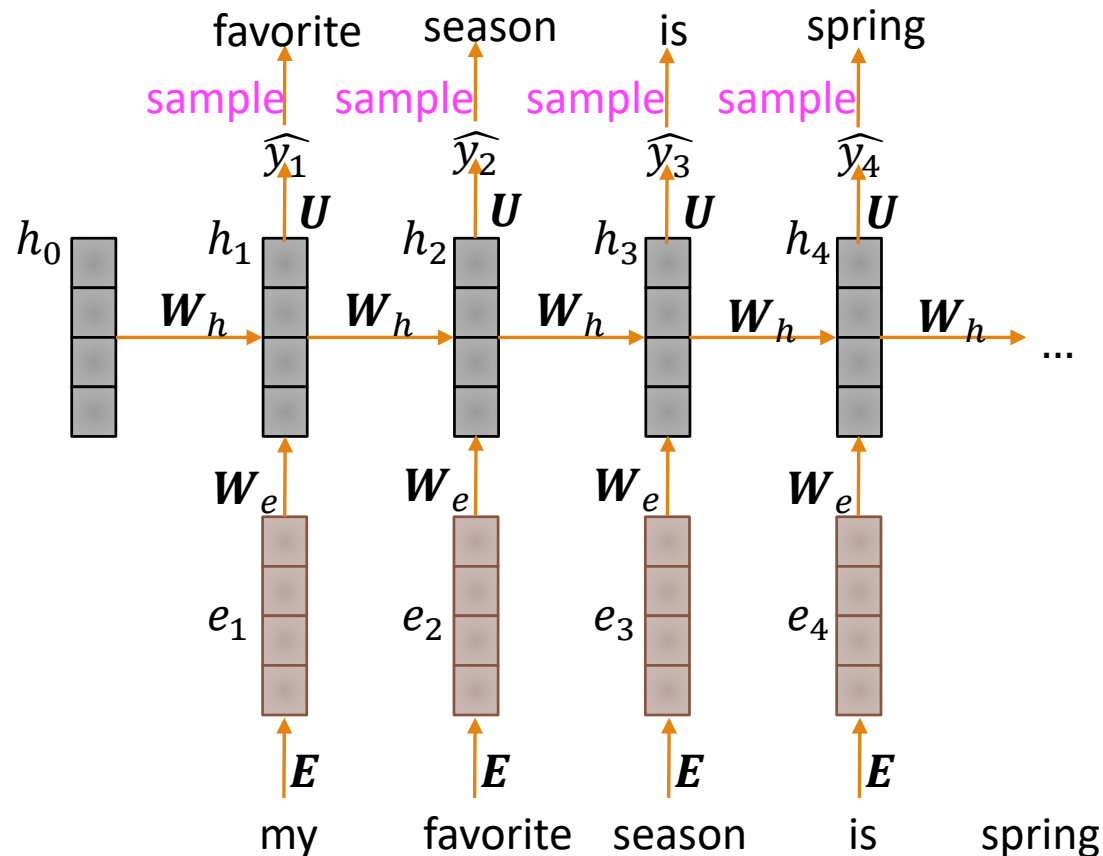
$$J(\theta) = \frac{1}{T} \sum_{i=1}^T J_i(\theta)$$

In practice, consider x_1, \dots, x_T as a **sentence** (or a **document**)

Using **Stochastic Gradient Descent**: compute the loss $J(\theta)$ and gradients for a sentence (usually a batch of sentences), update the weights. Repeat.

Generating Text With A RNN Language Model

Just like a n-gram Language Model, you can use a RNN language model to **generate text** by **repeated sampling**. Sampled output is next step's input.



Generating Text With A RNN Language Model

You can train a RNN-LM on any kind of text, then generate text in that style.

RNN-LM trained on Obama speeches:

SEED: Democracy

Good morning. One of the borders will be able to continue to be here today. We have to say that the partnership was a partnership with the American people and the street continually progress that is a process and distant lasting peace and support that they were supporting the work of concern in the world. They were in the streets and communities that could have to provide steps to the people of the United States and Afghanistan. In the streets — the final decade of the country that will include the people of the United States of America. Now, humanitarian crisis has already rightly achieved the first American future in the same financial crisis that they can find reason to invest in the world.

Thank you very much. God bless you. God bless you. Thank you.



Generating Text With A RNN Language Model

You can train a RNN-LM on any kind of text, then generate text in that style.

RNN-LM trained on **Obama speeches**:



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.

Generating Text With A RNN Language Model

You can train a RNN-LM on any kind of text, then generate text in that style.

RNN-LM trained on **Harry Potter**:



“No idea,” said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry’s shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn’t felt it seemed. He reached the teams too.

Generating Text With A RNN Language Model

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>

Generating Text With A RNN Language Model

You can train a RNN-LM on any kind of text, then generate text in that style.

RNN-LM trained on **paint color names**:

	Ghasty Pink 231 137 165		Sand Dan 201 172 143
	Power Gray 151 124 112		Grade Bat 48 94 83
	Navel Tan 199 173 140		Light Of Blast 175 150 147
	Bock Coe White 221 215 236		Grass Bat 176 99 108
	Horble Gray 178 181 196		Sindis Poop 204 205 194
	Homestar Brown 133 104 85		Dope 219 209 179
	Snader Brown 144 106 74		Testing 156 101 106
	Golder Craam 237 217 177		Stoner Blue 152 165 159
	Hurky White 232 223 215		Burple Simp 226 181 132
	Burf Pink 223 173 179		Stanky Bean 197 162 171
	Rose Hork 230 215 198		Turdly 190 164 116

This is an example of a character-level RNN-LM(predicts what character comes next)

Evaluating Language Models

The standard **evaluation metric** for Language Models is **perplexity**.

$$\textit{perplexity} = \prod_{i=1}^T \left(\underbrace{\frac{1}{P_{LM}(x_{i+1}|x_1, \dots, x_i)}} \right)^{1/T}$$

Normalized by the number of words

Inverse probability of corpus, according to language model

This is equal to the exponential of the cross-entropy loss $J(\theta)$

$$= \prod_{i=1}^T \left(\frac{1}{\hat{y}_i(x_{i+1})} \right)^{1/T} = \exp \left(\frac{1}{T} \sum_{i=1}^T -\log \hat{y}_i(x_{i+1}) \right) = \exp(J(\theta))$$

RNNs Have Greatly Improved Perplexity

	Model	Perplexity
<i>n</i> -gram model	Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
	RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
	RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
	Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
	LSTM-2048 (Jozefowicz et al., 2016)	43.7
	2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
	Ours small (LSTM-2048)	43.9
	Ours large (2-layer LSTM-2048)	39.8

<https://research.fb.com/blog/2016/10/building-an-efficient-neural-language-model-over-a-billion-words/>

Perplexity improves
(lower is better)