DSA4213 Natural Language Processing for Data Science

Doudou Zhou (ddzhou@nus.edu.sg)

Lecture 2 Language Modeling

Lecture Plan

Lecture 2: Language Modeling

Recap: the first lecture

- Concept and applications of NLP
- Word embedding
  - Word2vec: Skip-gram
  - Co-occurrence and SVD
  - SPPMI-SVD
  - Glove
- Word embedding evaluation
- Word sense ambiguity

Language Modeling

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

Language Modeling

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

Example

The students opened their _____

Language Modeling

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

Example

The students opened their _____

- Possible completions: books, laptops, exams, minds

Language Modeling

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

Example

The students opened their _____

- Possible completions: books, laptops, exams, minds

- More formally: Given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$, compute the probability distribution of the next word $x^{(t+1)}$:

$$P(x^{(t+1)} \mid x^{(t)}, \ldots, x^{(1)}),$$

where $x^{(t+1)}$ can be any word in the vocabulary $V = \{w_1, \ldots, w_{|V|}\}$

Language Modeling

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

Example

The students opened their _____

- Possible completions: books, laptops, exams, minds

- More formally: Given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$, compute the probability distribution of the next word $x^{(t+1)}$:

$$P(x^{(t+1)} \mid x^{(t)}, \ldots, x^{(1)}),$$

  where $x^{(t+1)}$ can be any word in the vocabulary $V = \{w_1, \ldots, w_{|V|}\}$
- A system that does this is called a **Language Model**

Language Modeling

- You can also think of a **Language Model** as a system that **assigns a probability to a piece of text**

Language Modeling

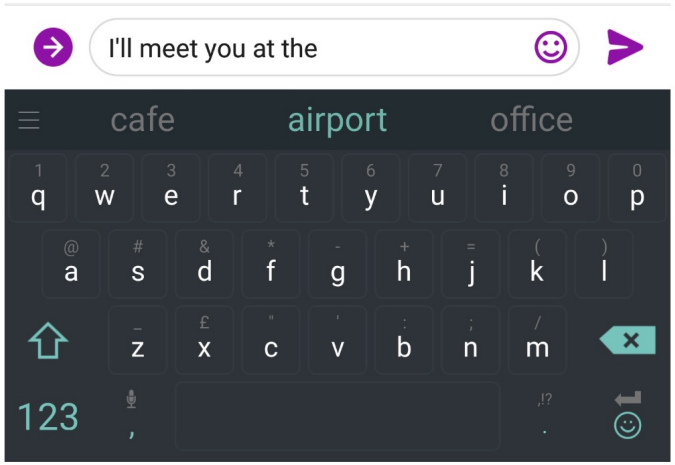
- You can also think of a **Language Model** as a system that **assigns a probability to a piece of text**
- For example, if we have some text $x^{(1)}, \ldots, x^{(T)}$, then the probability of this text (according to the Language Model) is:

$$P(x^{(1)}, \ldots, x^{(T)}) = P(x^{(1)}) \times P(x^{(2)} \mid x^{(1)}) \times \cdots \times P(x^{(T)} \mid x^{(T-1)}, \ldots, x^{(1)})$$

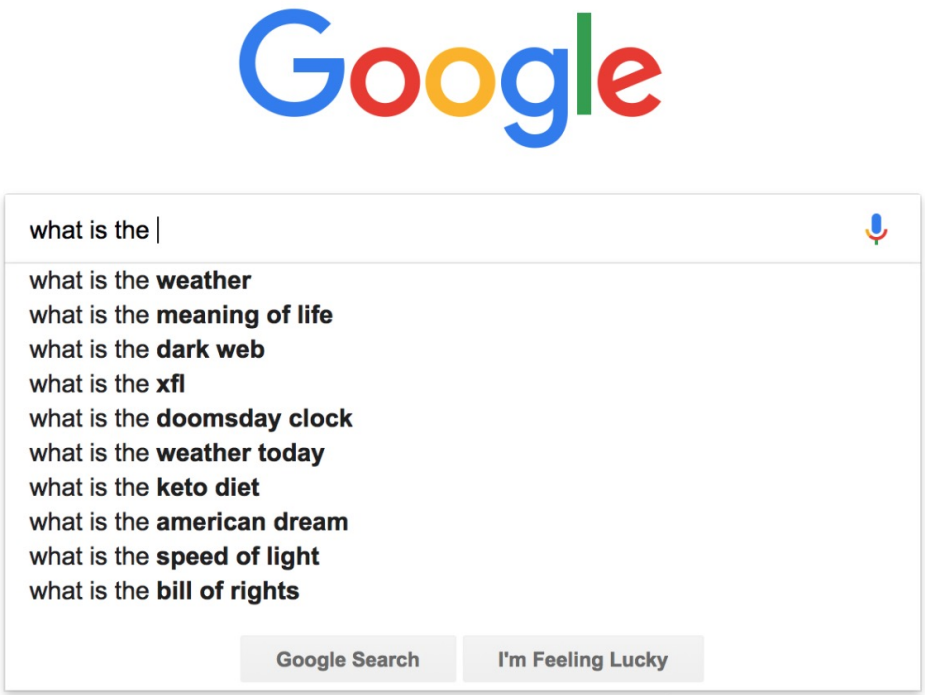
$$= \prod_{t=1}^{T} P(x^{(t)} \mid x^{(t-1)}, \ldots, x^{(1)})$$

This is what our Language Model provides

You use Language Models every day!



You use Language Models every day!



Example

The students opened their _____

- **Question:** How to learn a Language Model?

Example

The students opened their _____

- **Question:** How to learn a Language Model?
- **Answer (pre-Deep Learning):** Learn an **n-gram Language Model**
- **Definition:** An *n-gram* is a chunk of *n* consecutive words.
  - **unigrams:** "the", "students", "opened", "their"
  - **bigrams:** "the students", "students opened", "opened their"
  - **trigrams:** "the students opened", "students opened their"
  - **four-grams:** "the students opened their"
- **Idea:** Collect statistics about how frequent different *n-grams* are and use these to predict the next word.

n-gram language models

- First, we make a **Markov assumption**: $x^{(t+1)}$ depends only on the preceding $n-1$ words.

$$P(x^{(t+1)} \mid x^{(t)}, \ldots, x^{(1)}) = P(x^{(t+1)} \mid x^{(t)}, \ldots, x^{(t-n+2)}) \quad \text{(assumption)}$$

$$= \frac{P(x^{(t+1)}, x^{(t)}, \ldots, x^{(t-n+2)})}{P(x^{(t)}, \ldots, x^{(t-n+2)})} \quad \text{(definition of conditional probability)}$$

- **Question:** How do we get these n-gram and (n-1)-gram probabilities?

n-gram language models

- First, we make a **Markov assumption**: $x^{(t+1)}$ depends only on the preceding $n-1$ words.

$$P(x^{(t+1)} \mid x^{(t)}, \ldots, x^{(1)}) = P(x^{(t+1)} \mid x^{(t)}, \ldots, x^{(t-n+2)}) \quad \text{(assumption)}$$

$$= \frac{P(x^{(t+1)}, x^{(t)}, \ldots, x^{(t-n+2)})}{P(x^{(t)}, \ldots, x^{(t-n+2)})} \quad \text{(definition of conditional probability)}$$

- **Question:** How do we get these n-gram and (n-1)-gram probabilities?
- **Answer:** By **counting** them in some large corpus of text!

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

Example Corpus for Bigram Model

- **Mini-corpus** (3 short sentences):
  - "I love data"
  - "I love statistics"
  - "I love machine learning"

Example Corpus for Bigram Model

- **Mini-corpus** (3 short sentences):
  - "I love data"
  - "I love statistics"
  - "I love machine learning"
- Add special tokens <s> (start) and </s> (end):
  - `<s> I love data </s>`
  - `<s> I love statistics </s>`
  - `<s> I love machine learning </s>`

n-gram language models: example

- Suppose that we are learning a **4-gram** Language Model.

~~as the proctor started the clock,~~ $\underbrace{\textit{the students opened their}}_{\text{condition on this}}$ ____

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

n-gram language models: example

- Suppose that we are learning a **4-gram** Language Model.

~~as the proctor started the clock,~~ $\underbrace{\textit{the students opened their}}_{\text{condition on this}}$ ____

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

- For example, suppose that in the corpus:
  - "students opened their" occurred 1000 times
  - "students opened their books" occurred 400 times
    - $\Rightarrow P(\text{ books} \mid \text{students opened their}) = 0.4$
  - "students opened their exams" occurred 100 times
    - $\Rightarrow P(\text{exams} \mid \text{students opened their}) = 0.1$

n-gram language models: example

- Suppose that we are learning a **4-gram** Language Model.

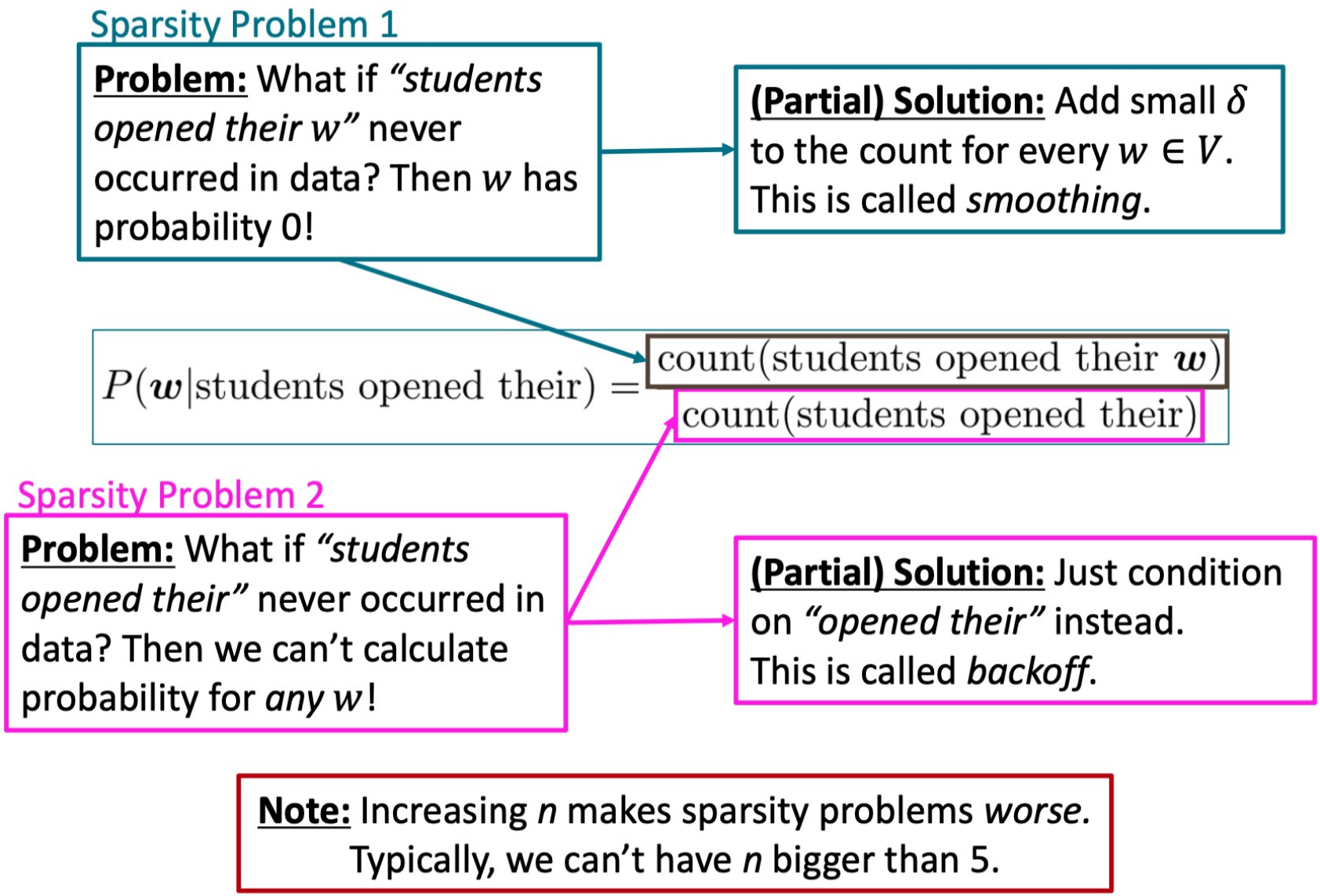
~~as the proctor started the clock,~~ $\underbrace{\textit{the students opened their}}_{\text{condition on this}}$ ____

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

- For example, suppose that in the corpus:
  - "students opened their" occurred 1000 times
  - "students opened their books" occurred 400 times
    - $\Rightarrow P(\text{ books} \mid \text{students opened their}) = 0.4$
  - "students opened their exams" occurred 100 times
    - $\Rightarrow P(\text{exams} \mid \text{students opened their}) = 0.1$

Should we have discarded the "proctor" context?

Sparsity problems with n-gram language models



Storage problems with n-gram language models

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

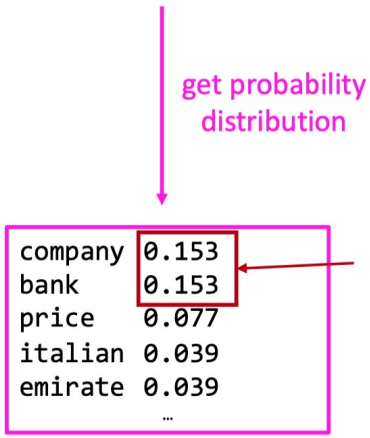
$$P(w \mid \text{students opened their}) = \frac{\boxed{\text{count}(\text{students opened their } w)}}{\text{count}(\text{students opened their})}$$

- Increasing n or increasing corpus increases model size!

n-gram language models in practice

- You can build a simple trigram Language Model over a 1.7 million word corpus (Reuters, Business and financial news) in a few seconds on your laptop

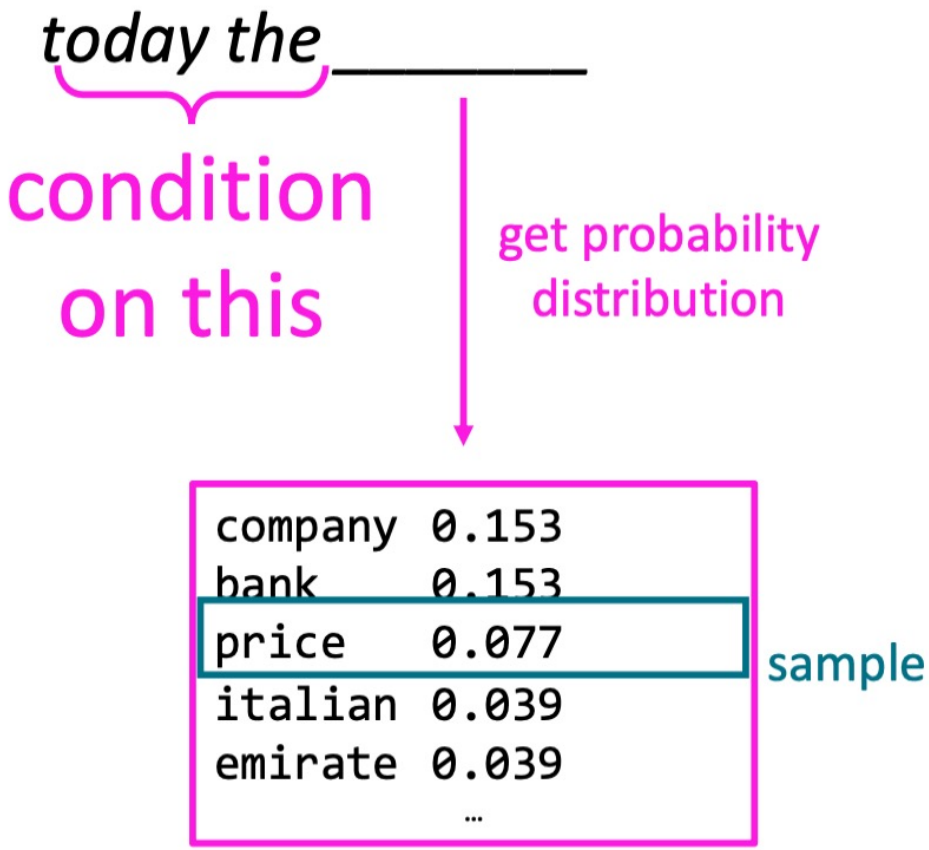
today the ____



- Sparsity problem: not much granularity in the probability distribution
- Otherwise, seems reasonable!

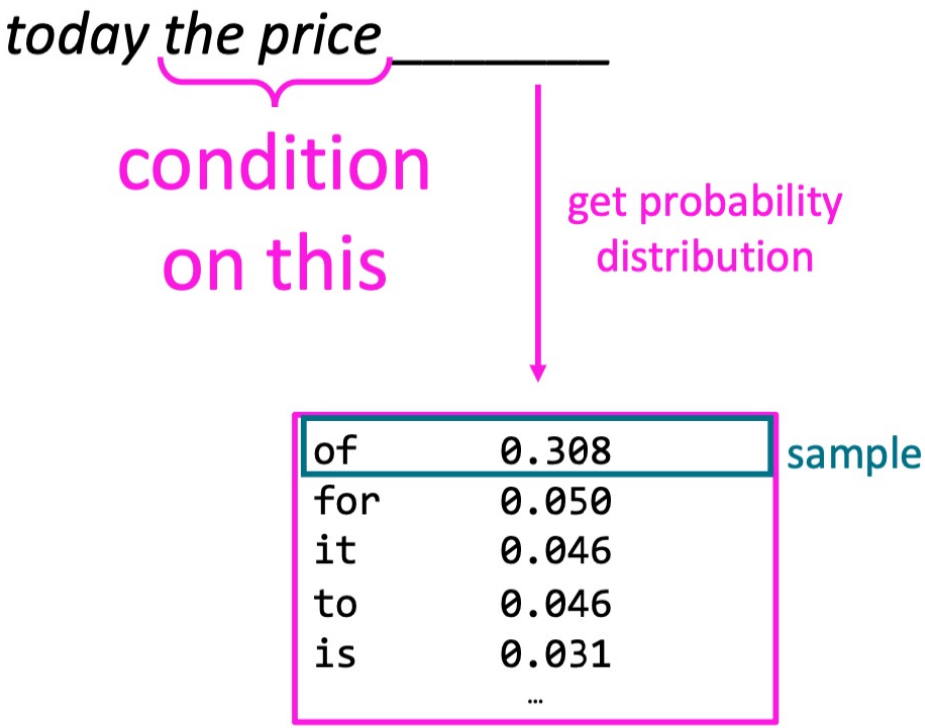
Generating text with a n-gram language model

- You can also use a Language Model to generate text.



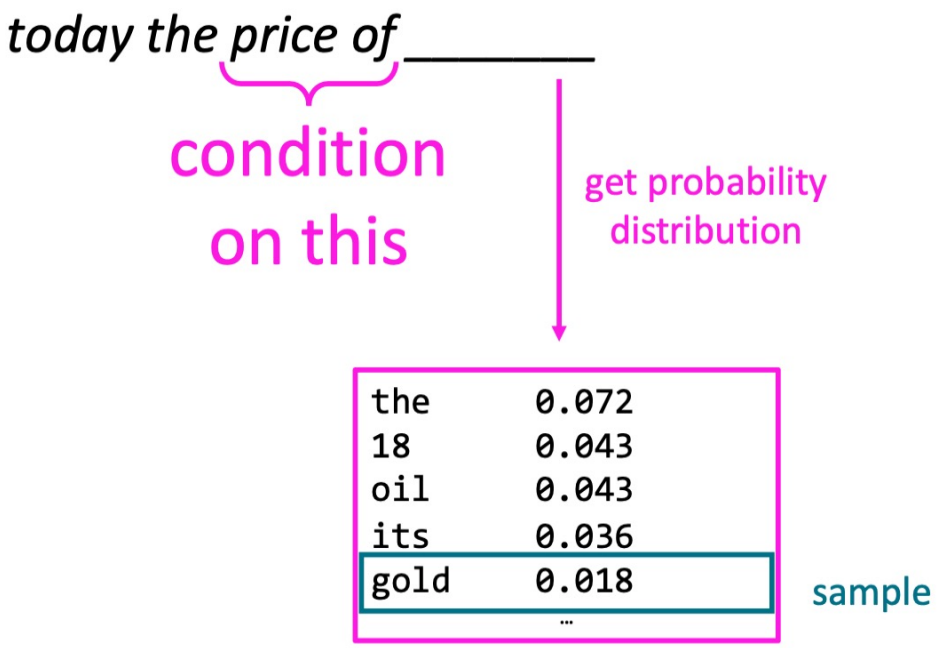
Generating text with a n-gram language model

- 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...

How to build a neural language model?

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

- How about a **window-based neural model**?
  - A fixed window neural language model

~~as the proctor started the clock,~~ *the students opened their* ____

~~as the proctor started the clock,~~: discard

the students opened their: fixed window

A fixed-window neural language model

Output distribution

$$\hat{y} = \text{softmax}(Uh + b_2) \in \mathbb{R}^{|V|}$$

Hidden layer

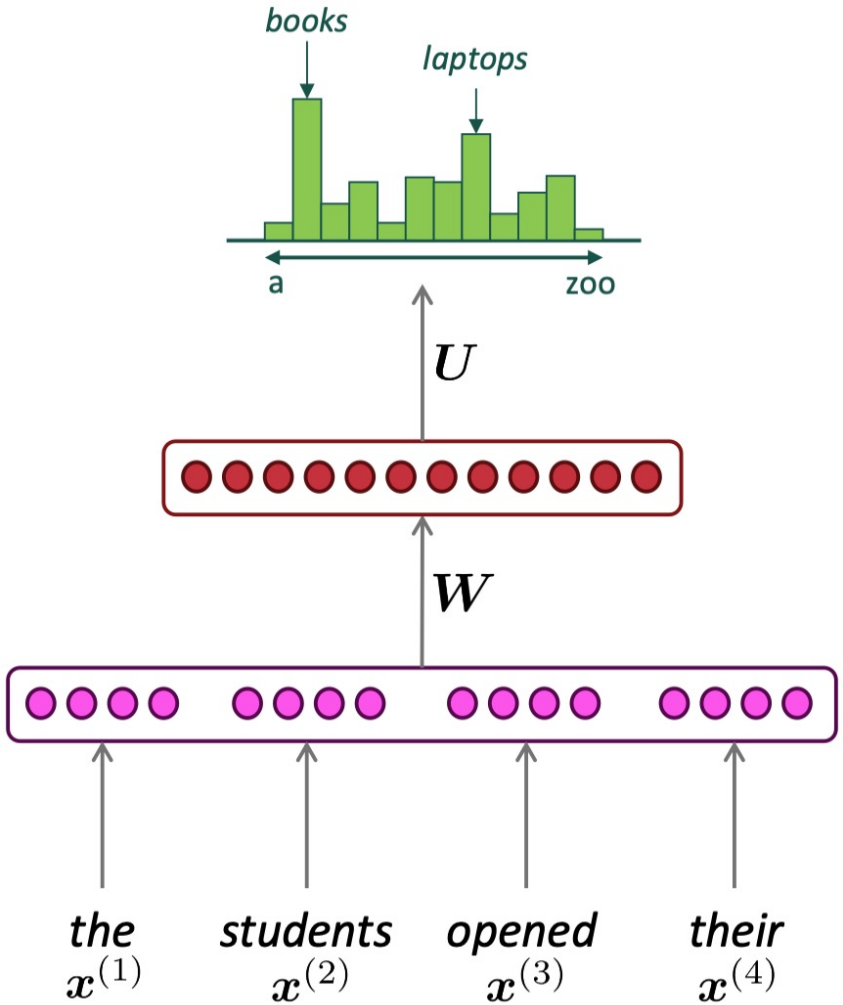
$$h = f(We + b_1)$$

Concatenated word embeddings

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

Words / One-hot vectors

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



A fixed-window neural language model

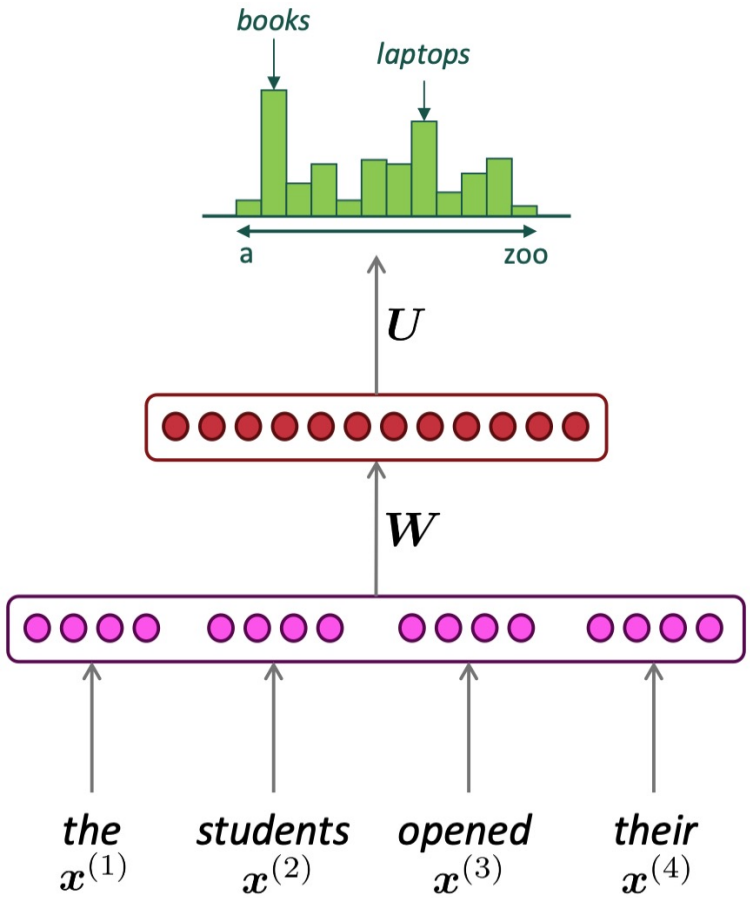
Approximately: Bengio et al. ((2000)): A Neural Probabilistic Language Model

Improvements over *n*-gram LM:

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

Remaining problems:

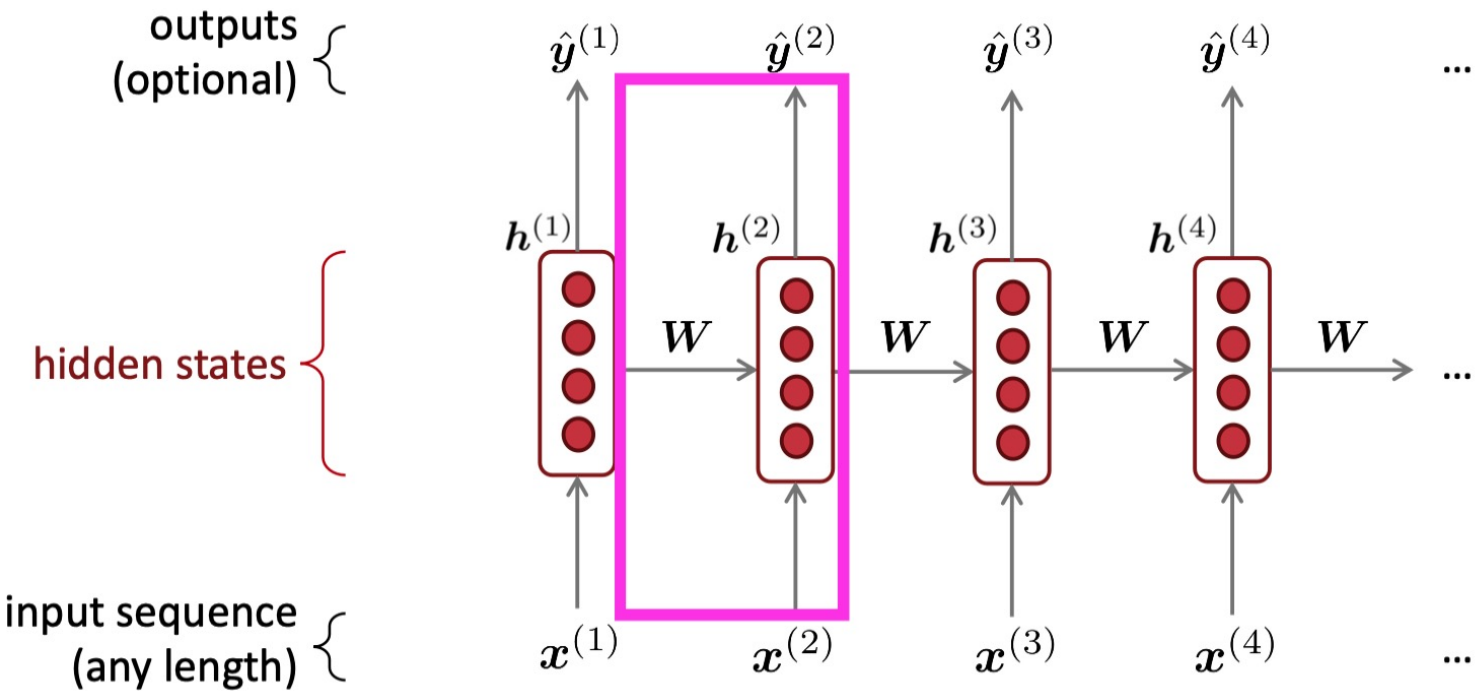
- Fixed window is too small
- Enlarging window enlarges **W**
- Window can never be large enough!
- $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in **W**.
- **No symmetry** in how the inputs are processed.



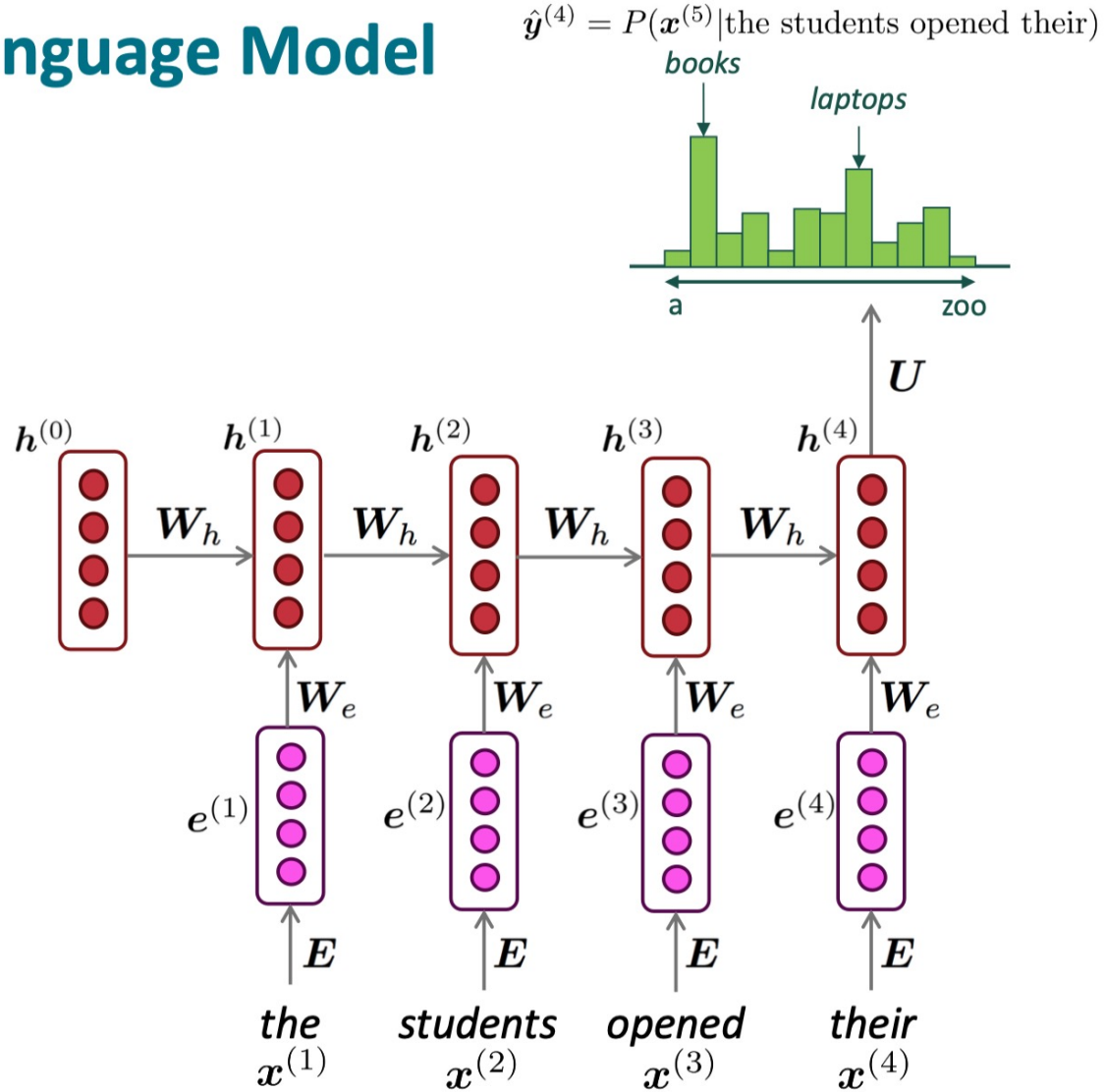
We need a neural architecture that can process *any length* input

Recurrent Neural Networks (RNN)

- A family of neural architectures
- Core idea: Apply the same weights **W** repeatedly



A Simple RNN Language Model



output distribution

$$\hat{y}^{(t)} = \text{softmax}\left(Uh^{(t)} + b_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_e e^{(t)} + b_1\right)$$

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

word embeddings

$$e^{(t)} = Ex^{(t)}$$

words / one-hot vectors

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

***Note**: this input sequence could be much longer now!*

RNN language models

RNN Advantages:

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

RNN language models

RNN Advantages:

- Can process **any length** input
- Computation for step t can (in theory) use information from **many steps back**
- Model size **doesn't increase** for longer input context
- 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**

More on these later

Training an RNN Language Model

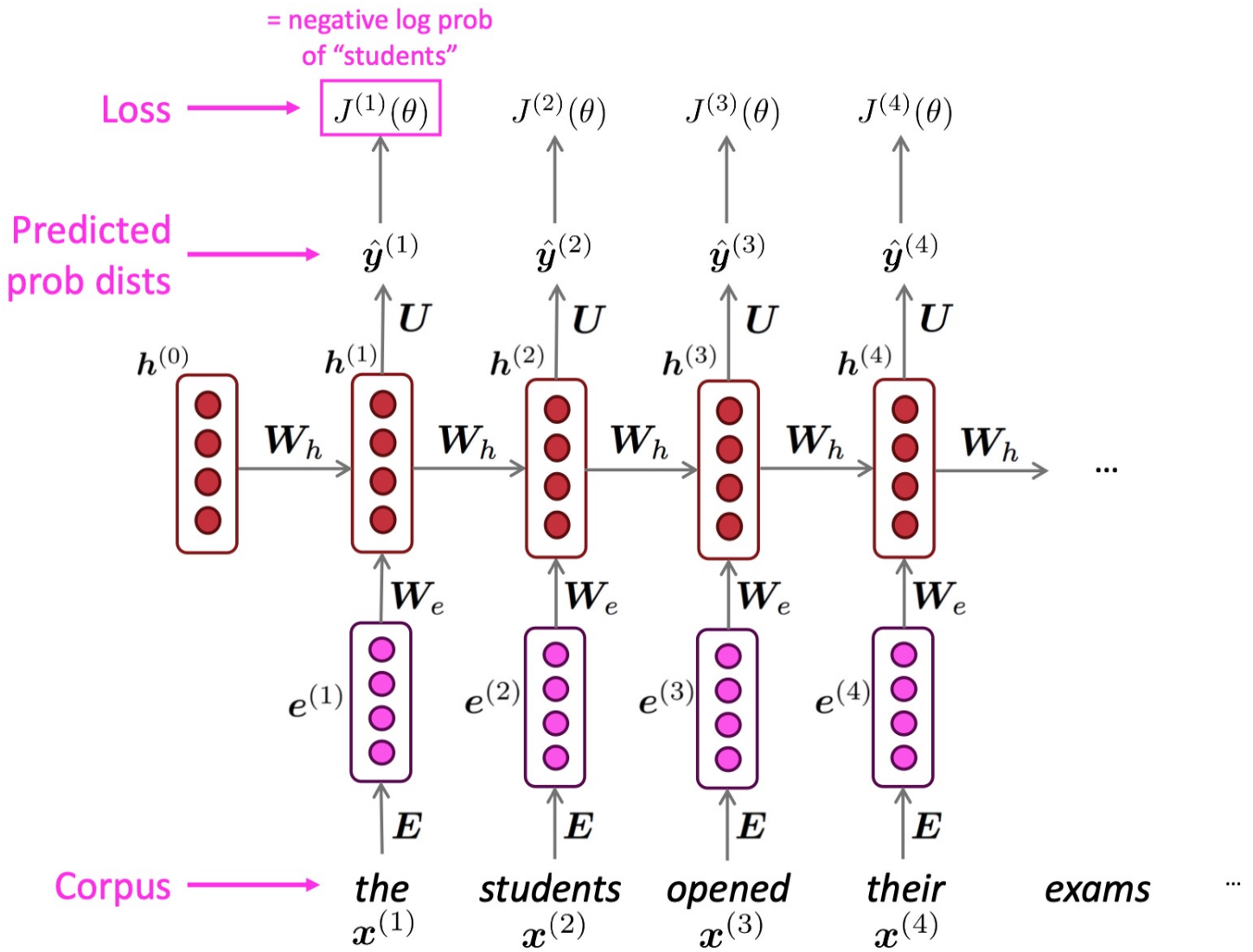
- Get a **big corpus of text** which is a sequence of words $x^{(1)}, \ldots, x^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{y}^{(t)}$ **for every step** t.
  - Predict probability distribution of **every word**, given words so far
- **Loss function** on step t is cross-entropy between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

$$J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = -\sum_{w \in V} y_w^{(t)} \log \hat{y}_w^{(t)} = -\log \hat{y}_{x_{t+1}}^{(t)}$$

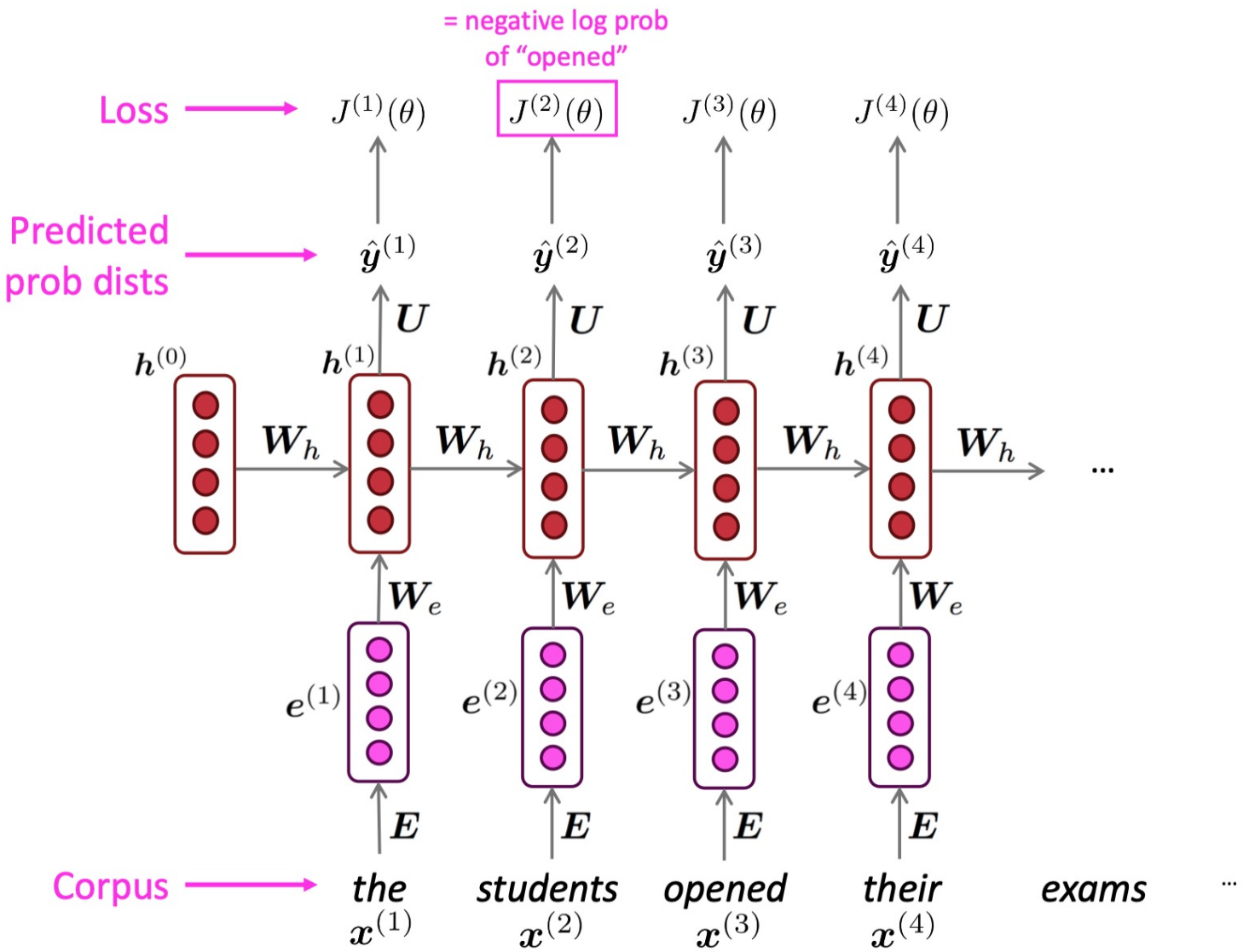
- Average this to get **overall loss** for entire training set:

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

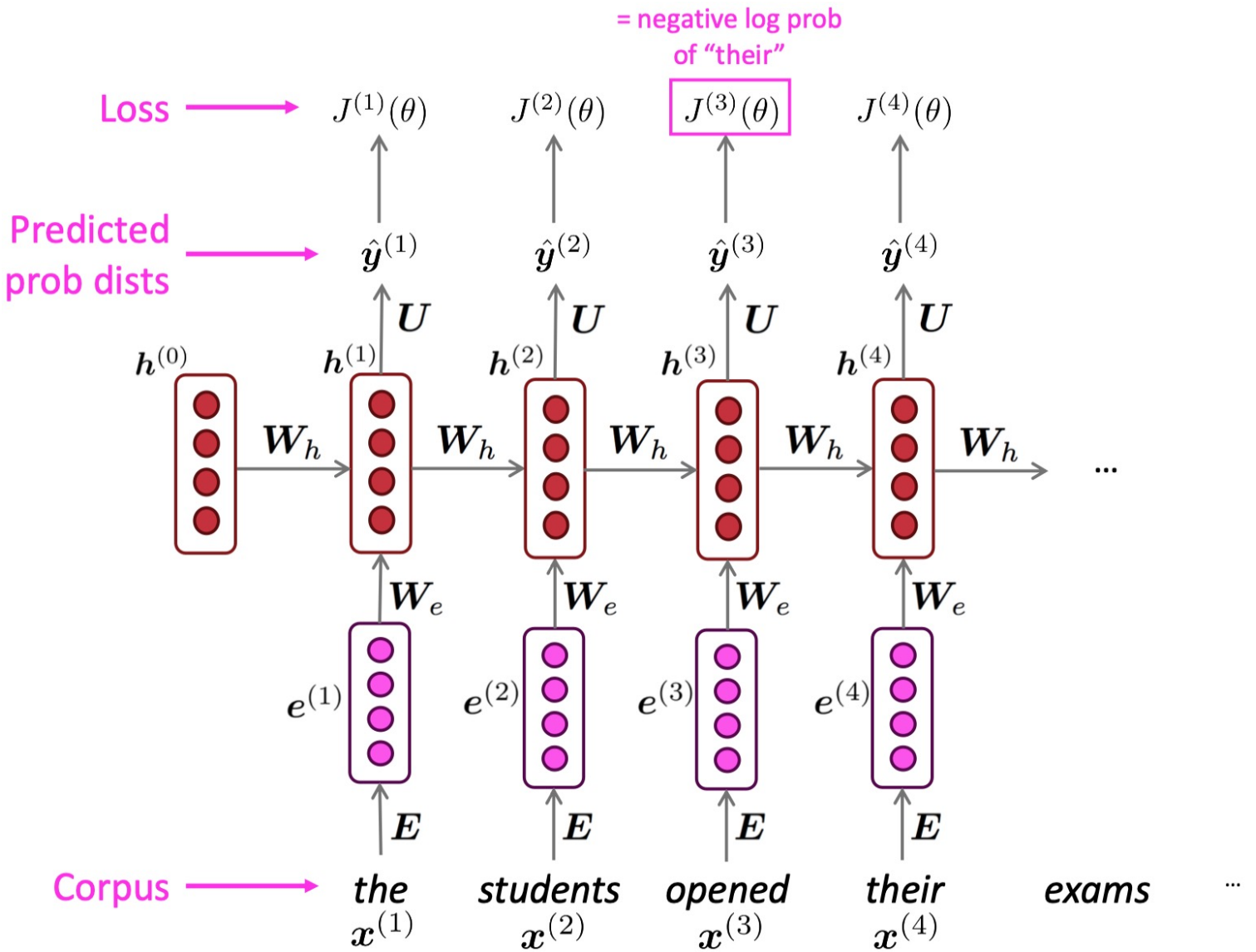
Training an RNN language model



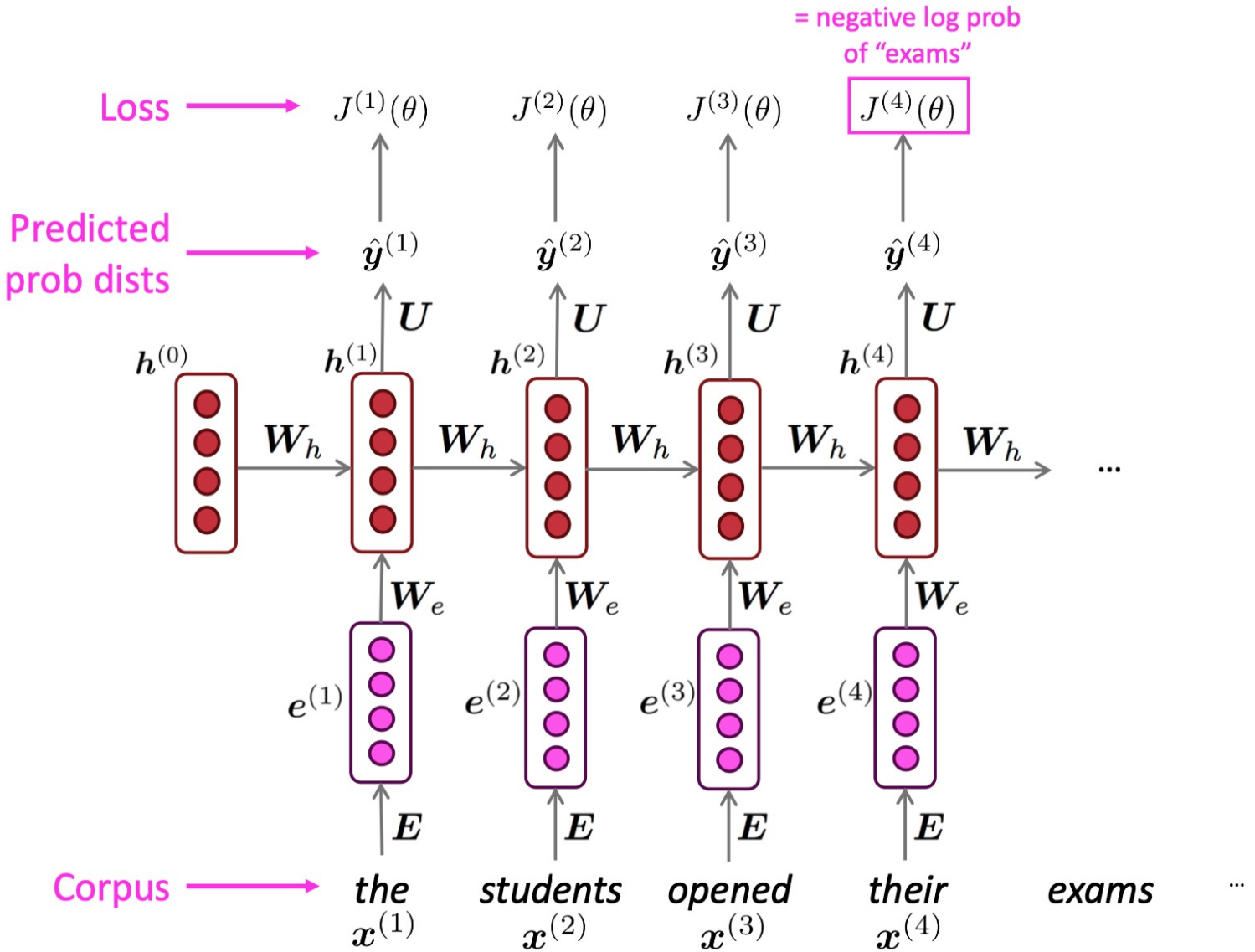
Training an RNN language model



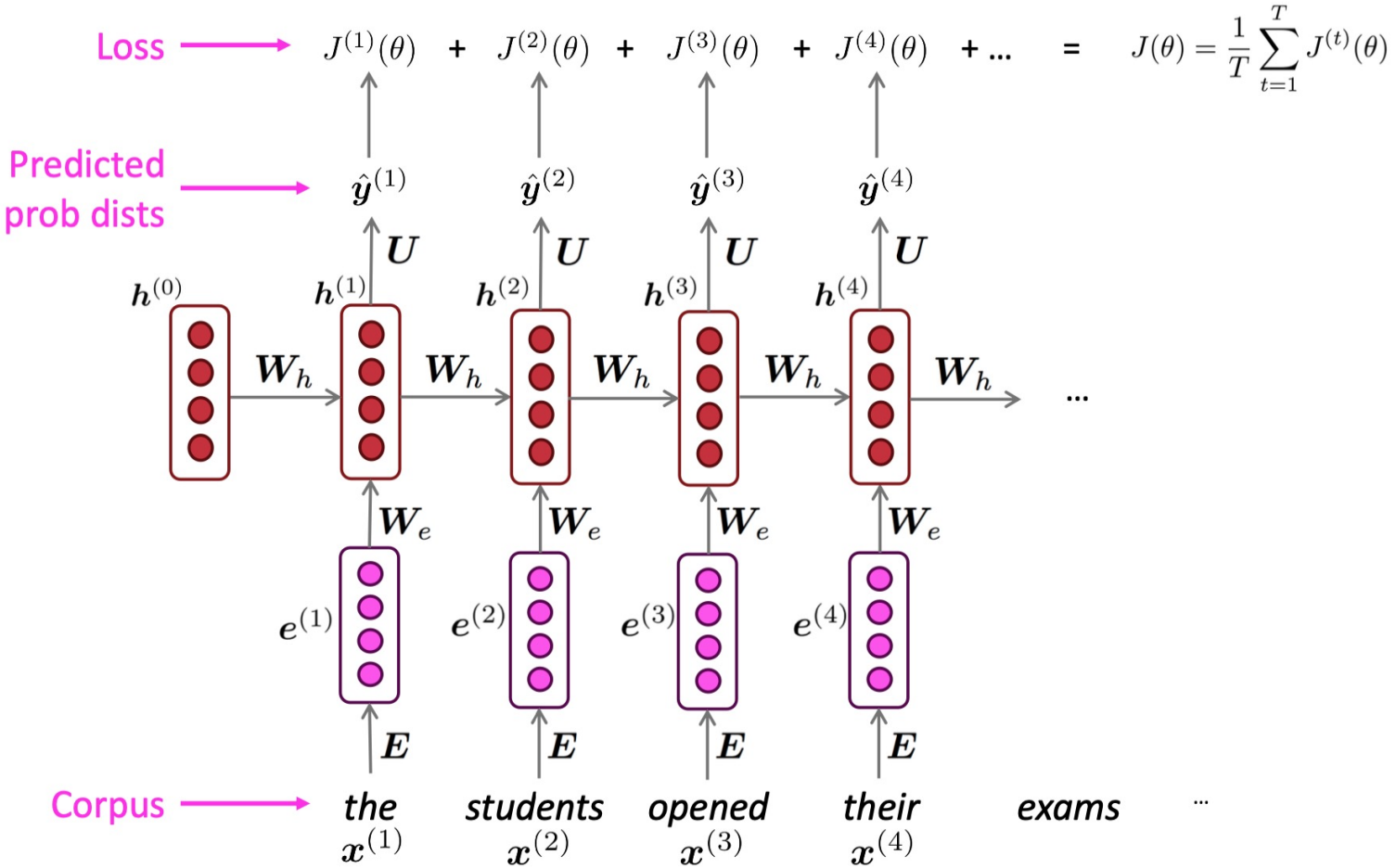
Training an RNN language model



Training an RNN language model



Training an RNN language model



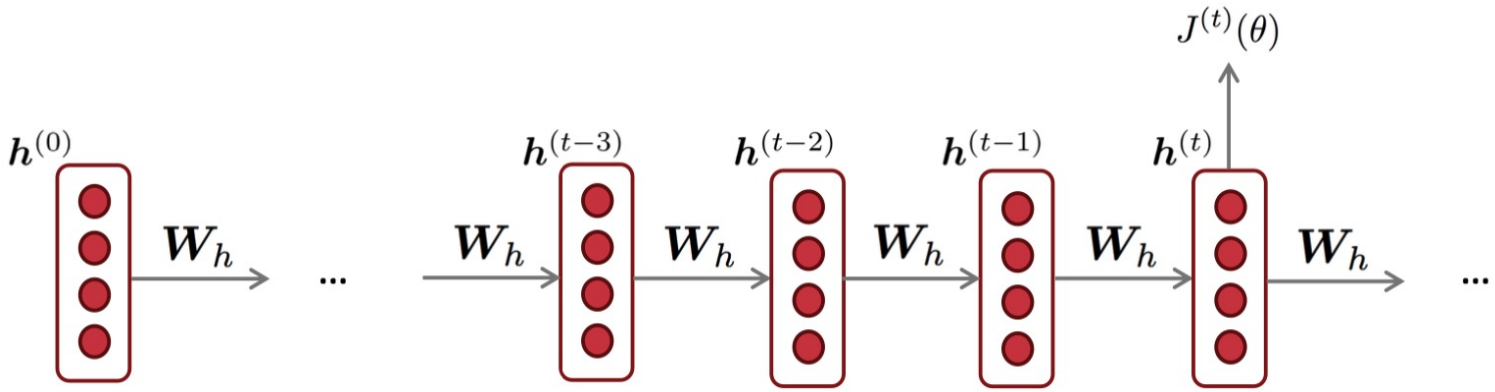
Training a RNN Language Model

- However: computing loss and gradients across **entire corpus** $x^{(1)}, \ldots, x^{(T)}$ at once is **too expensive** (memory-wise)!

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

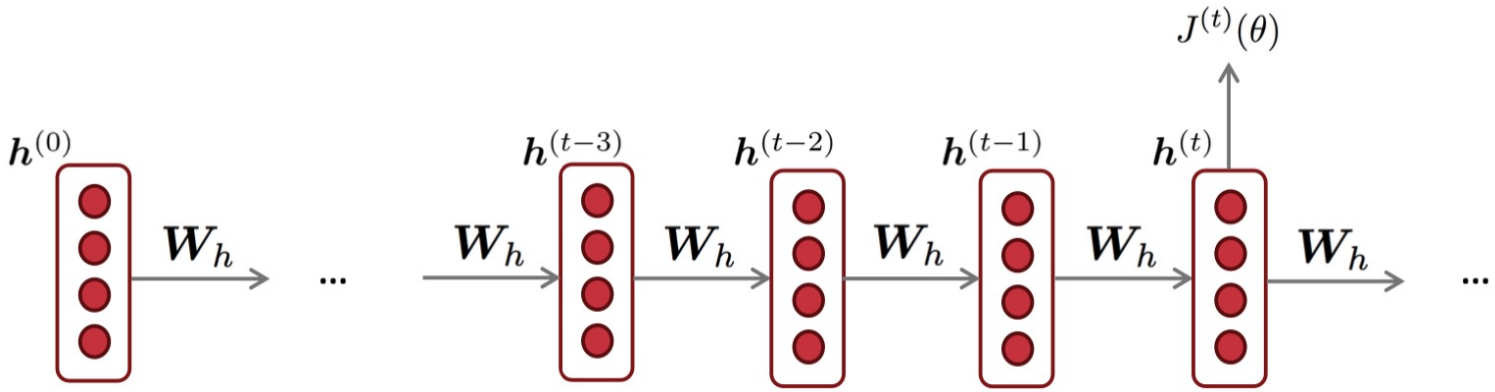
- In practice, consider $x^{(1)}, \ldots, x^{(T)}$ as a **sentence** (or a document)
- **Recall**: **Stochastic Gradient Descent** allows us to compute loss and gradients for small chunk of data, and update.
- Compute loss $J(\theta)$ for a sentence (actually, a batch of sentences), compute gradients and update weights. Repeat on a new batch of sentences.

Backpropagation for RNNs



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

Backpropagation for RNNs



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

Answer:

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

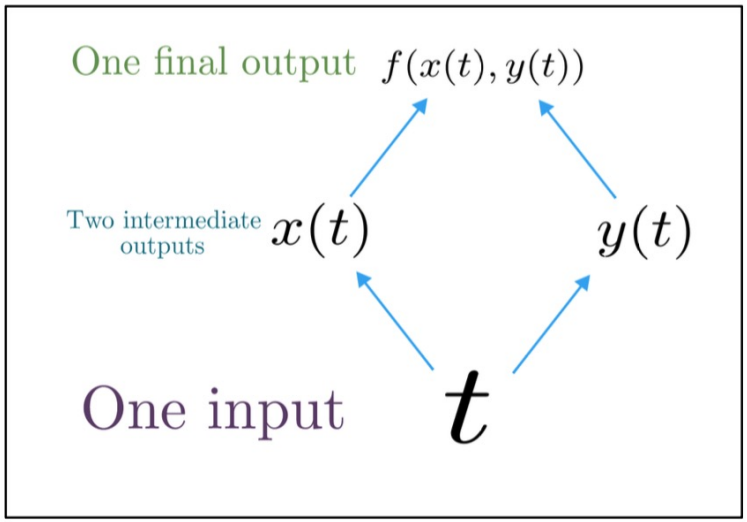
"The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears"

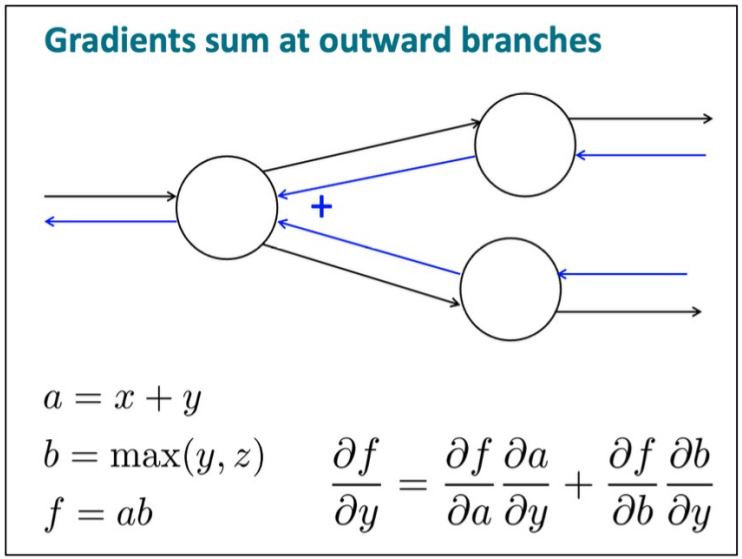
Why?

Multivariable chain rule

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

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

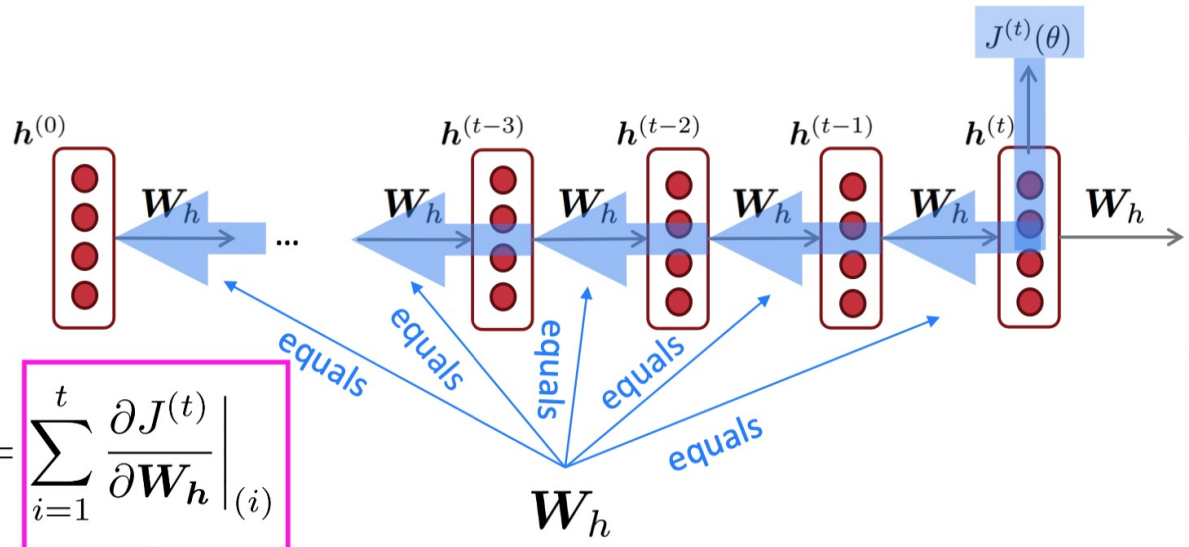




source:

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

Training the parameters of RNNs: backpropagation for RNNs



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

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

Question: How do we calculate this?

Answer: Backpropagate over timesteps $i = t, \ldots, 0$, summing gradients as you go. This algorithm is called **"backpropagation through time"** [Werbos, P.G., 1988, *Neural Networks* **1**, and others]

Apply the multivariable chain rule:

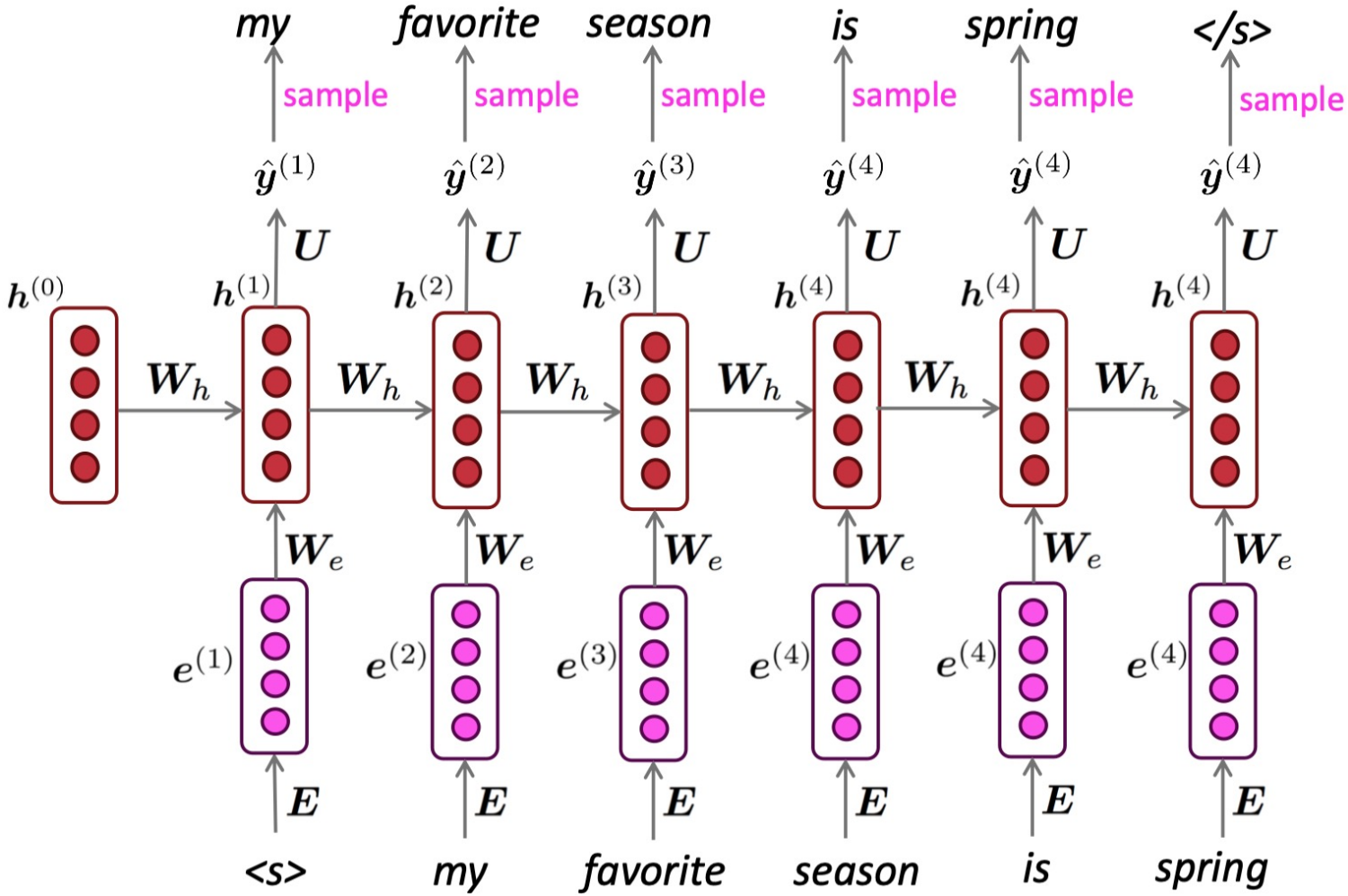
$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}\bigg|_{(i)} \frac{\partial \boldsymbol{W_h}|_{(i)}}{\partial \boldsymbol{W_h}}$$

($\frac{\partial \boldsymbol{W_h}|_{(i)}}{\partial \boldsymbol{W_h}} = 1$)

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

Generating with an RNN language model ("generating roll outs")

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



Generating text with an RNN language model

Let's have some fun!

- You can train an 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.

source:

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

Generating text with an RNN language model

Let's have some fun!

- You can train an RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on Harry Potter:

"Sorry," Harry shouted, panicking — "I'll leave those brooms in London, are they?"
"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.

source:

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

Generating text with an RNN language model

Let's have some fun!

- You can train an 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.

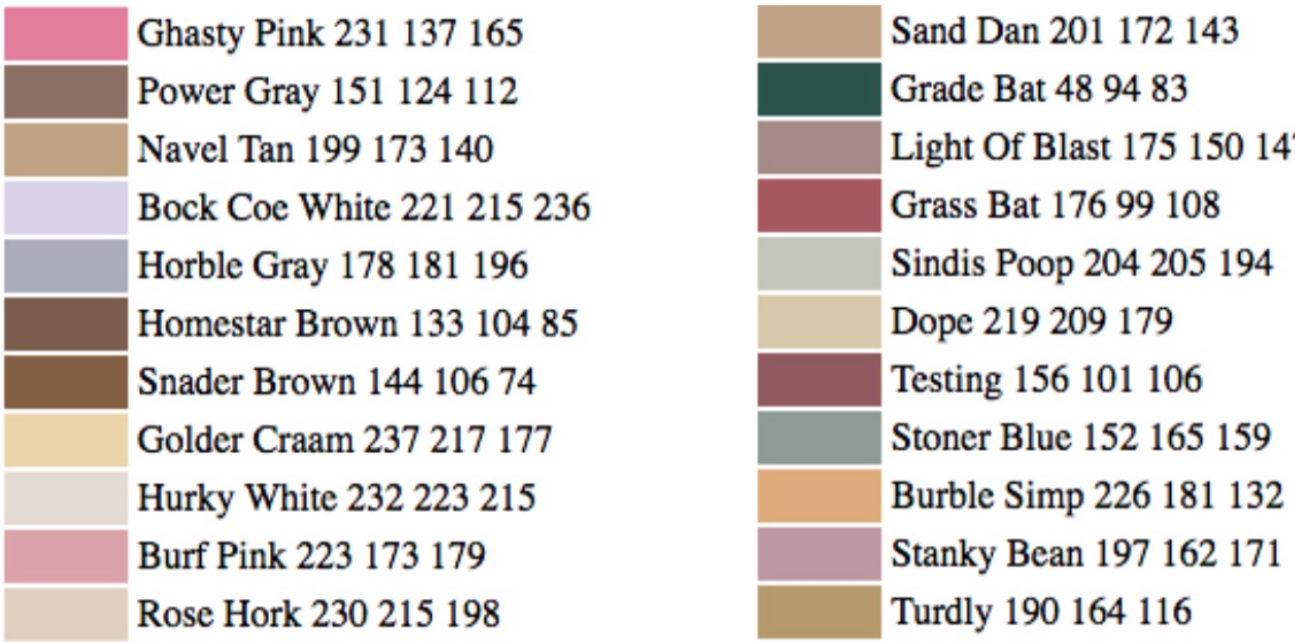
Combine the cheese and salt together the dough in a large skillet; add the ingredients and stir in the chocolate and pepper.

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

Generating text with a RNN language model

Let's have some fun!

- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on paint color names:



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

Source:http://aiweirdness.com/post/160776374467/new-paint-colors-invented-by-neural-

Evaluating language models

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

$$\text{perplexity} = \prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}\left(\mathbf{x}^{(t+1)} \mid \mathbf{x}^{(t)}, \ldots, \mathbf{x}^{(1)}\right)} \right)^{1/T}$$

  - Inverse probability of corpus, according to Language Model
  - Normalized by number of words

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

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

Lower perplexity is better!

RNNs greatly improved perplexity over what came before

n-gram model →

Increasingly complex RNNs ↓

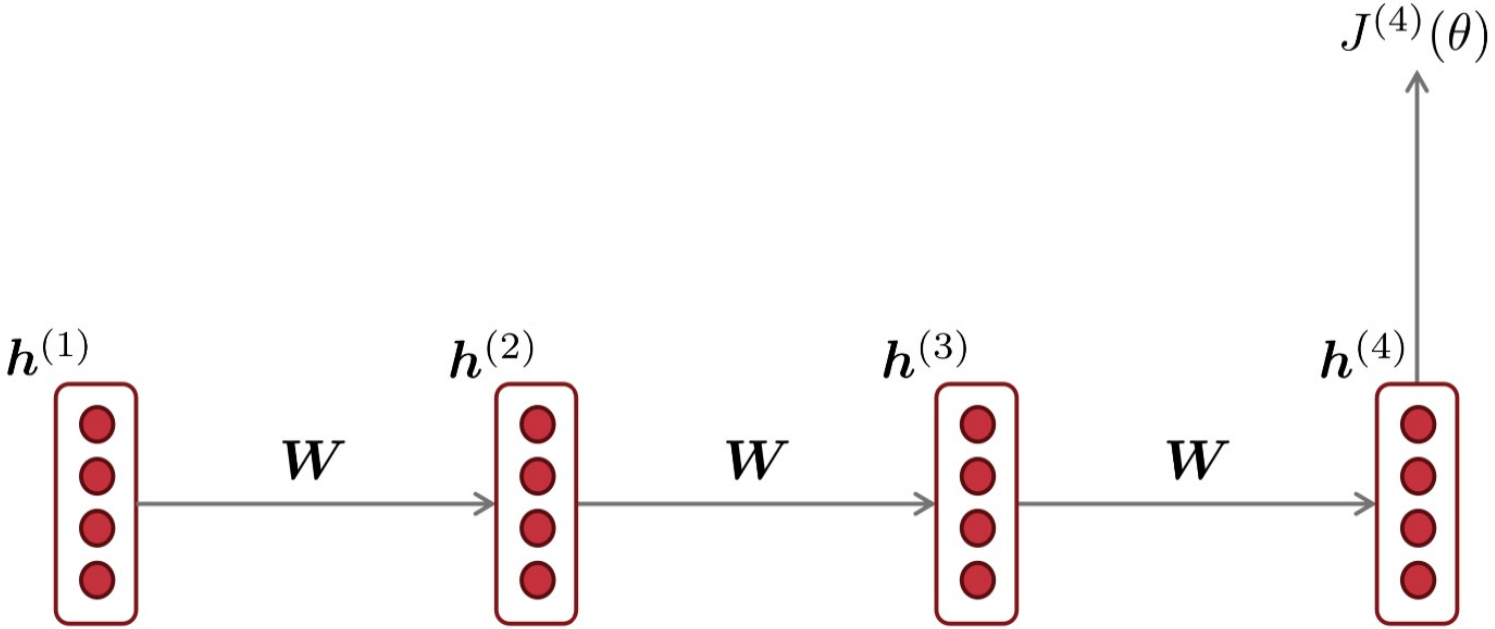
Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192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

↓ Perplexity improves (lower is better)

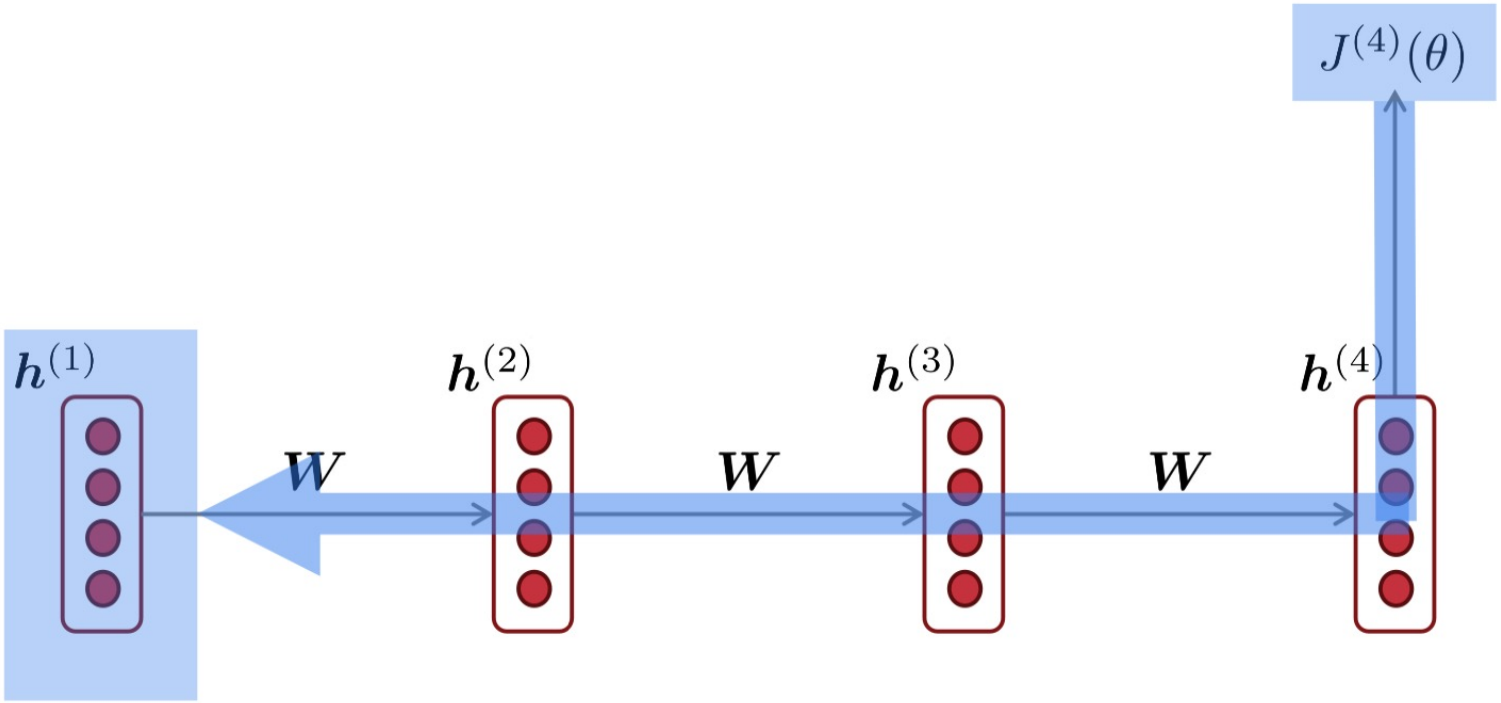
Source:

https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/

Problems with RNNs: vanishing and exploding gradients

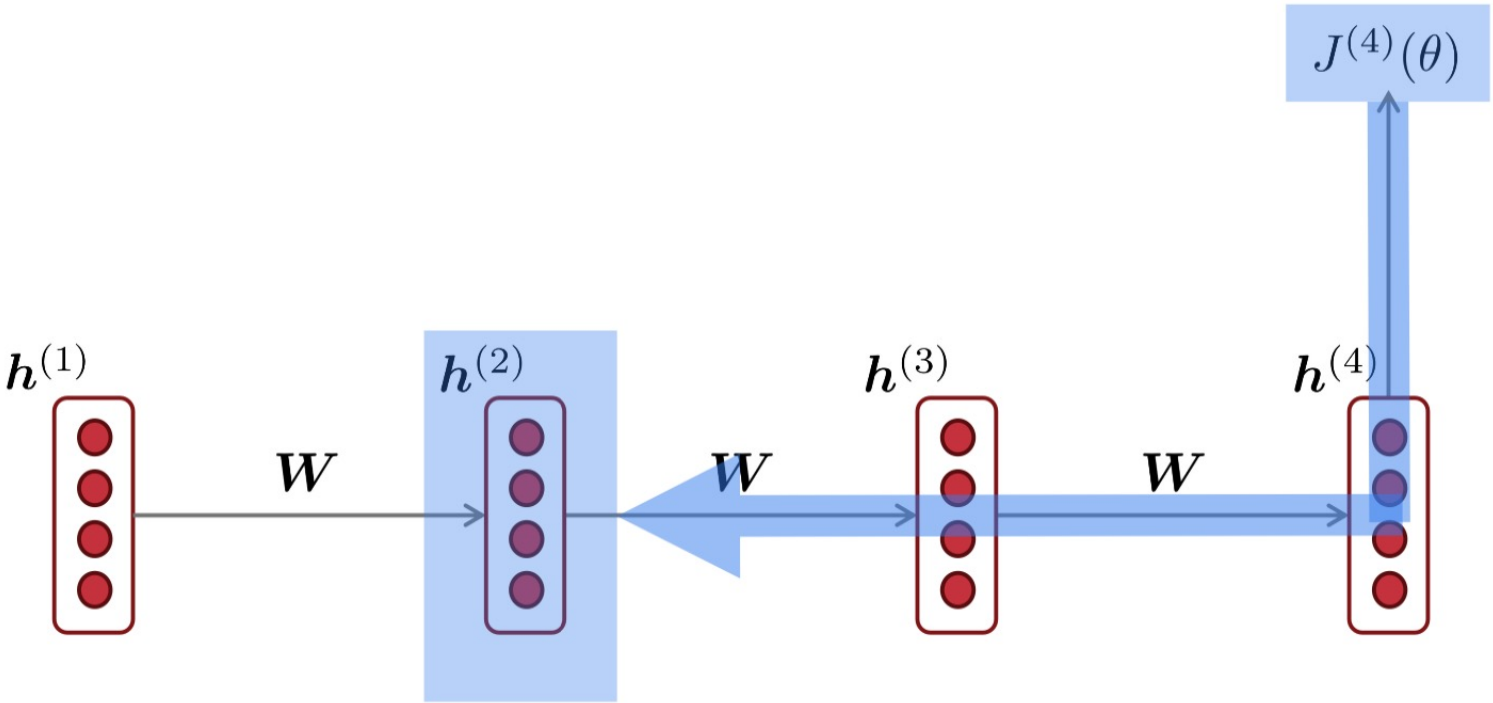


Vanishing gradient intuition



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

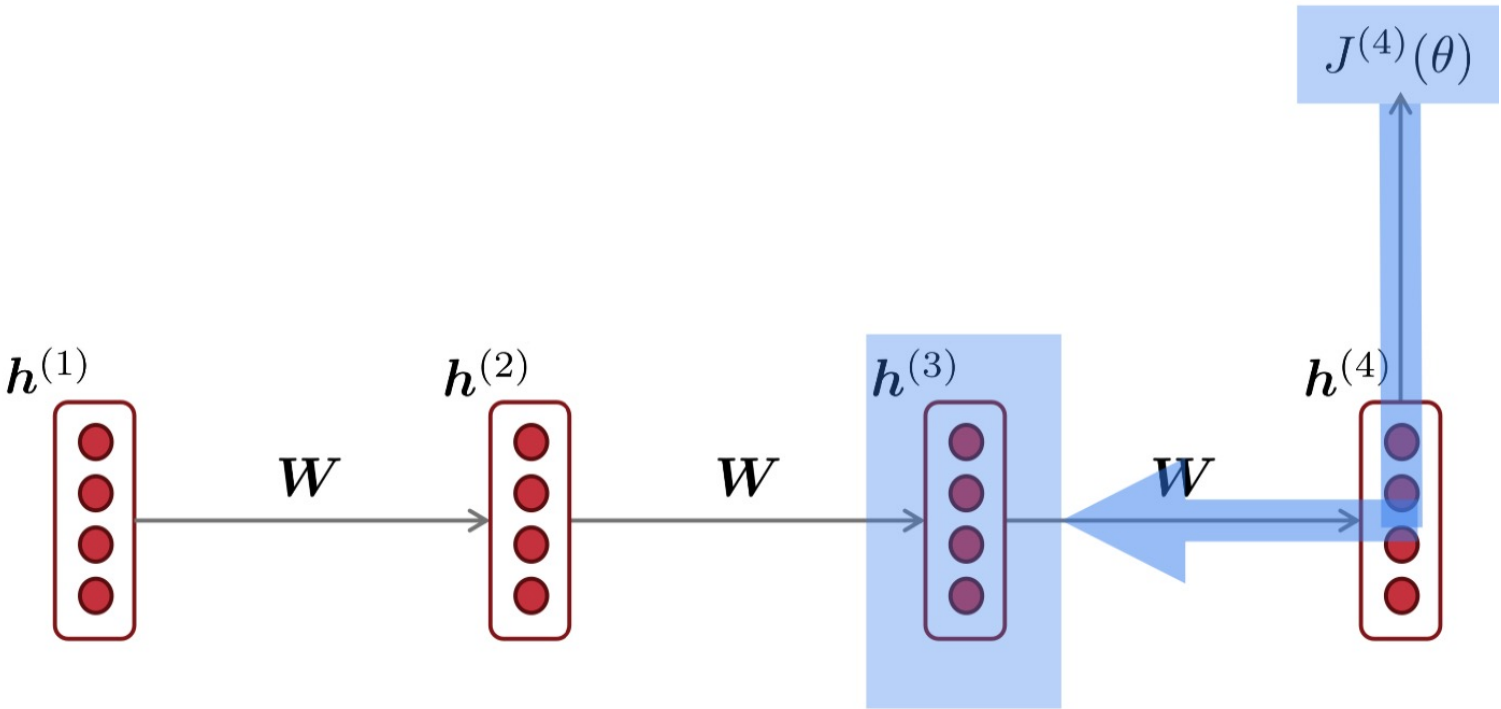
Vanishing gradient intuition



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

chain rule!

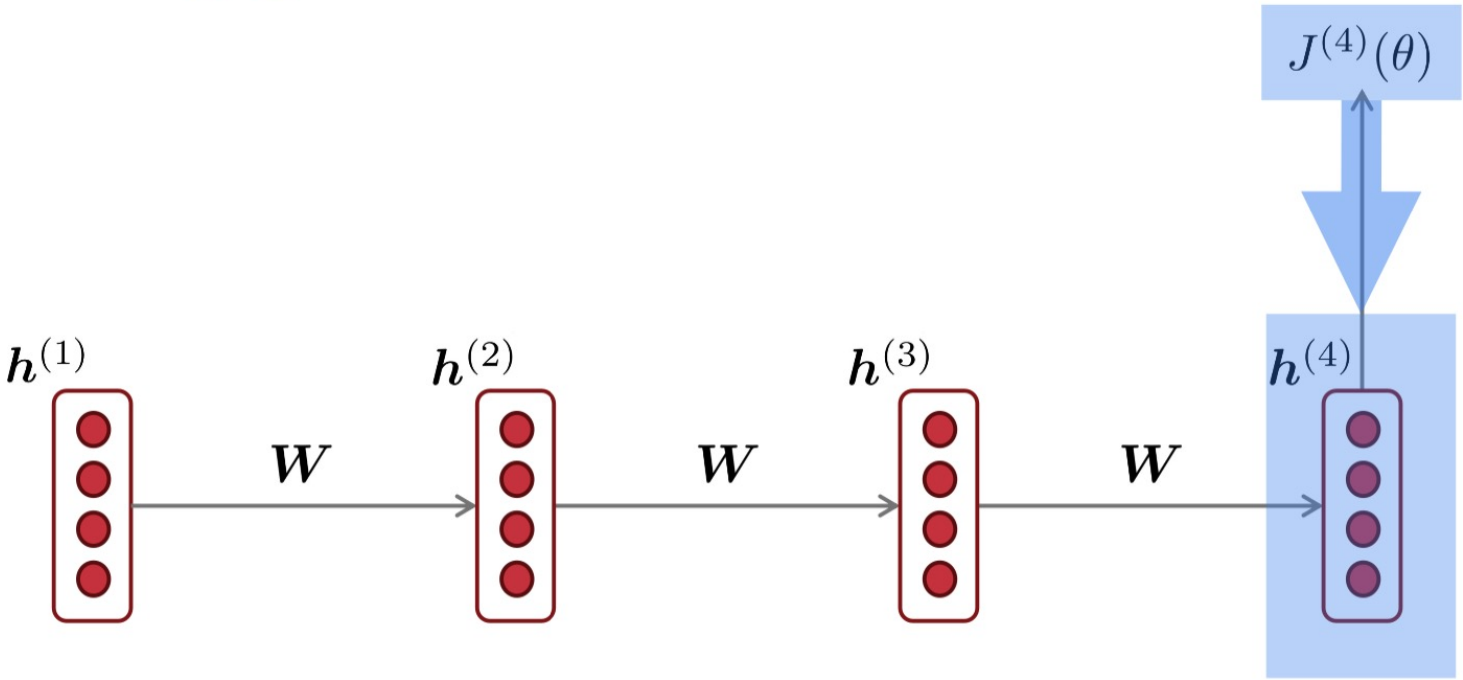
Vanishing gradient intuition



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

chain rule!

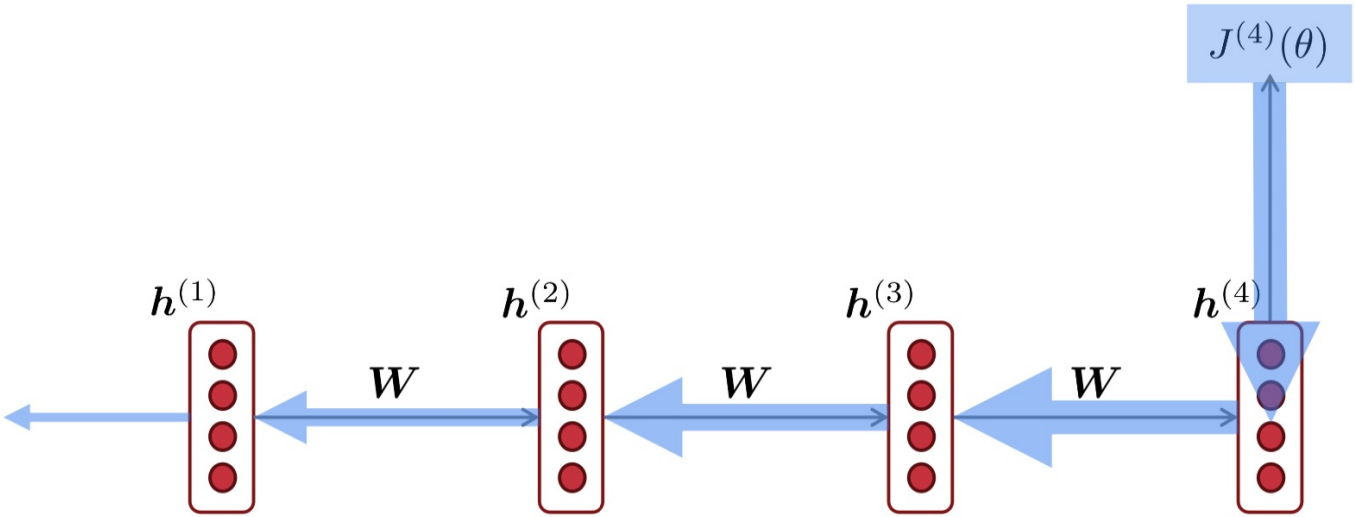
Vanishing gradient intuition



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

chain rule!

Vanishing gradient intuition

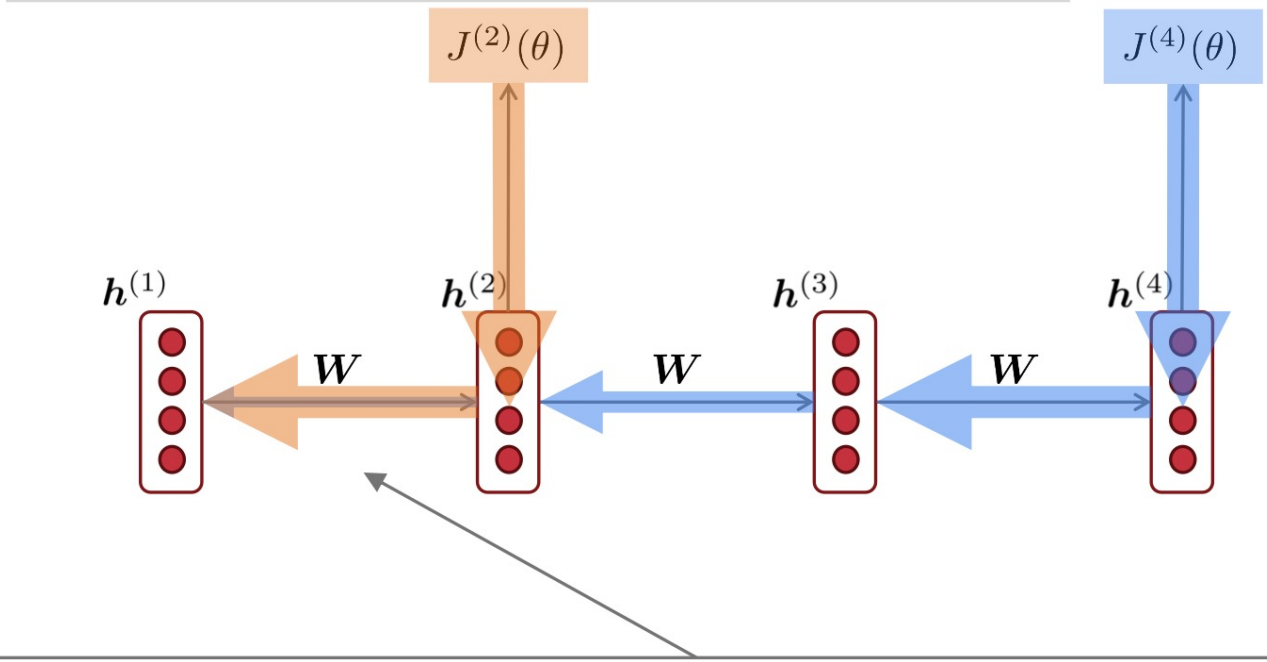


$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

What happens if these are small?

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

Why is vanishing gradient a problem?



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

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

Effect of vanishing gradient on RNN-LM

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

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

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

Why is exploding gradient a problem?

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

$$\theta_{\text{new}} = \theta_{\text{old}} - \alpha \underbrace{\nabla_\theta J(\theta)}_{\text{gradient}}$$

learning rate gradient

- This can cause **bad updates**: we take too large a step and reach a weird and bad parameter configuration (with large loss)
  - You think you've found a hill to climb, but suddenly you're in Iowa.
- In the worst case, this will result in **Inf** or **NaN** in your network:
  - Then you have to restart training from an earlier checkpoint.

Gradient clipping: solution for exploding gradient

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

Algorithm 1 Pseudo-code for norm clipping

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

- **Intuition**: take a step in the same direction, but a smaller step.
- In practice, **remembering to clip gradients is important**, but exploding gradients are an easy problem to solve.

How to fix the vanishing gradient problem?

- **The main problem** is that **it's too difficult for the RNN to learn to preserve information over many timesteps.**
- In a vanilla RNN, the hidden state is constantly being **rewritten**:

$$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_x x^{(t)} + b\right)$$

- **First off next time:** How about an RNN with separate **memory** which is added to?
  - **LSTMs**
- **And then:** Creating more direct and linear pass-through connections in model:
  - **Attention, residual connections, etc.**

Recap

- Language Model: A system that predicts the next word
- Recurrent Neural Network: A family of neural networks that:
  - Take sequential input of any length
  - Apply the same weights on each step
  - Can optionally produce output on each step
- **Recurrent Neural Network $\neq$ Language Model**

- We've shown that RNNs are a great way to build a LM (despite some problems). But:
  - RNNs are also useful for much more!
  - There are other models for building LMs (esp. Transformers!)

Why should we care about language modeling?

Why should we care about language modeling?

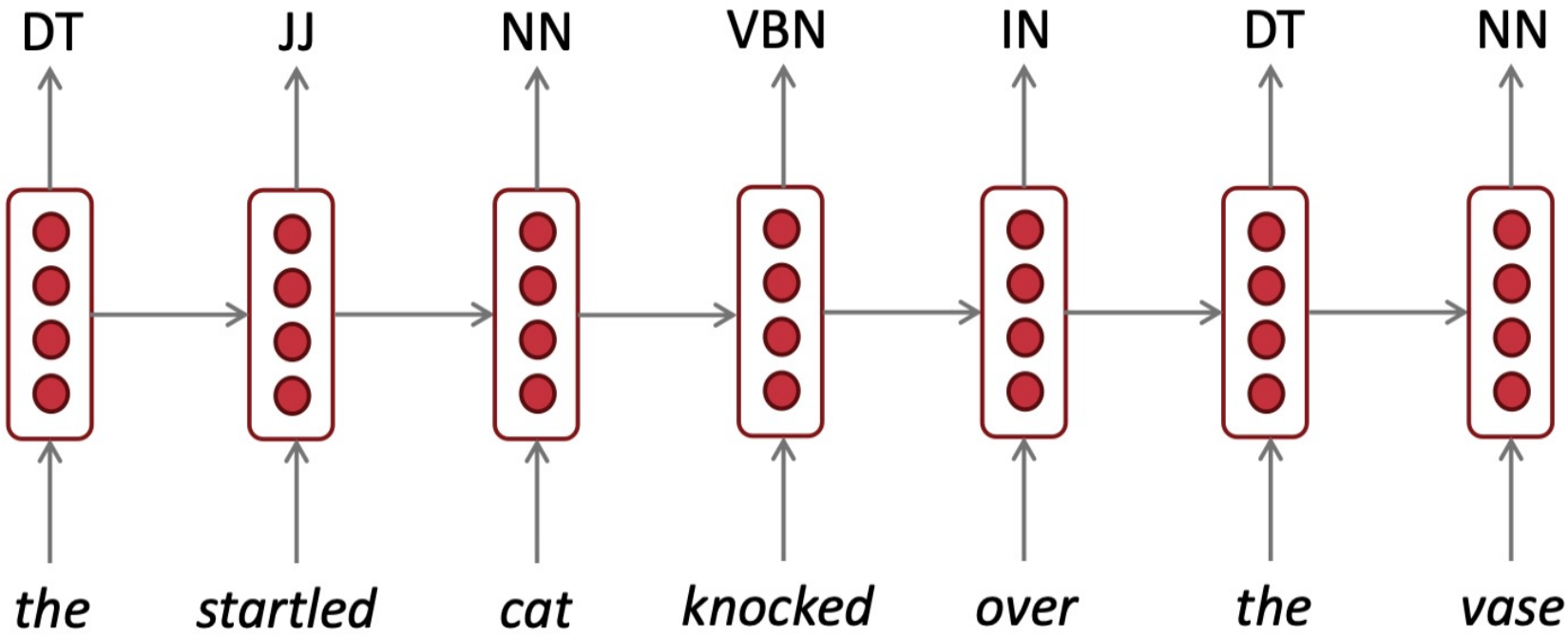
- **Old answer:**
  - Language Modeling is a benchmark task that helps us measure our progress on predicting language use
  - Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
    - Predictive typing, Speech recognition, Handwriting recognition, Spelling/grammar correction
    - Authorship identification, Machine translation, Summarization, Dialogue
    - etc.

Why should we care about language modeling?

- **Old answer:**
  - Language Modeling is a benchmark task that helps us measure our progress on predicting language use
  - 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.
- **New answer:**
  - Everything in NLP has now been rebuilt upon Language Modeling!
    - GPT-3 is an LM! GPT-4 is an LM! Claude Opus is an LM! Gemini Ultra is an LM!
    - We can now instruct LMs to do language understanding and reasoning tasks for us

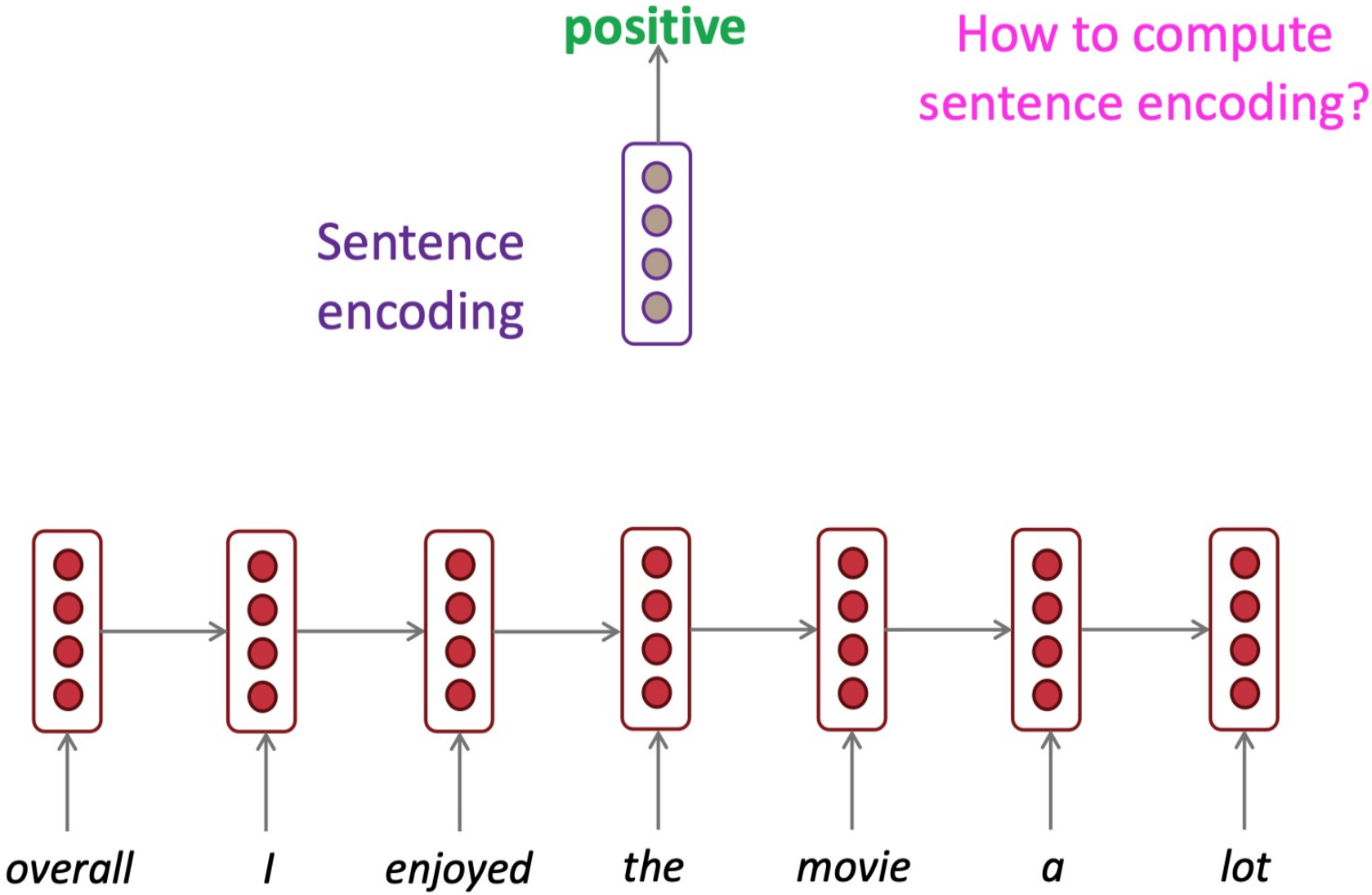
Other RNN uses: RNNs can be used for sequence tagging

e.g., part-of-speech tagging, named entity recognition



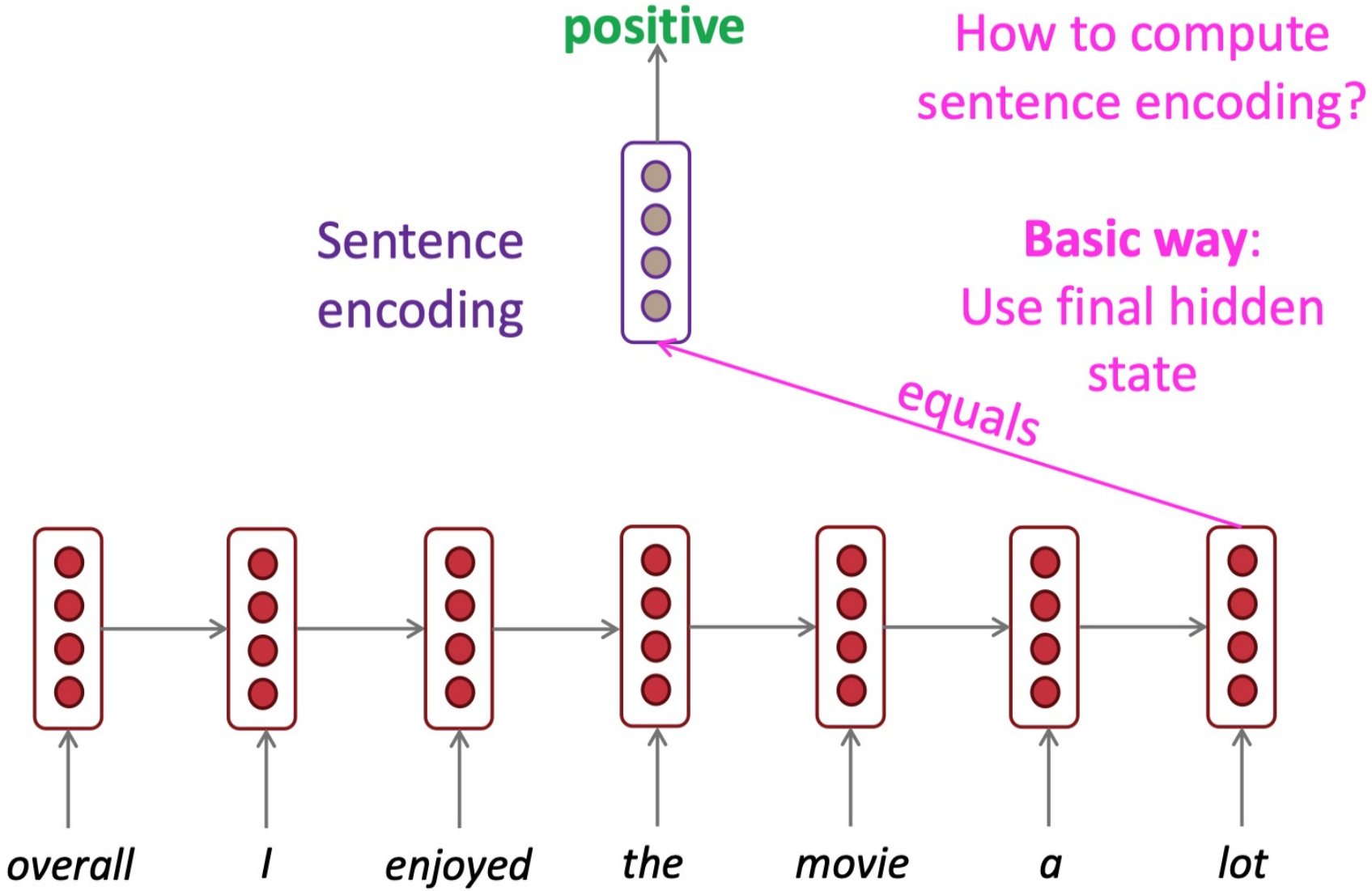
RNNs can be used for sentence classification

e.g., sentiment classification



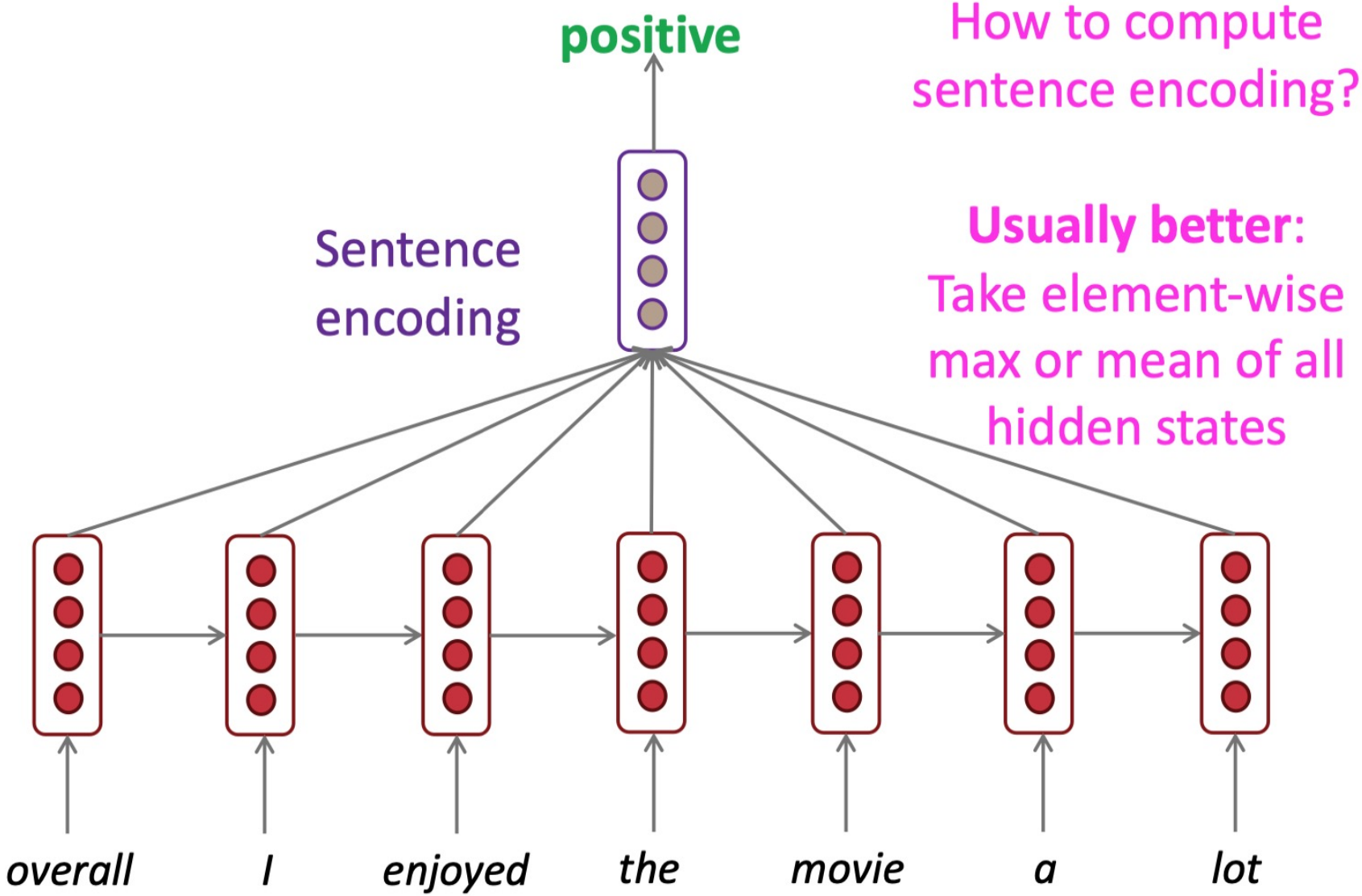
RNNs can be used for sentence classification

e.g., sentiment classification



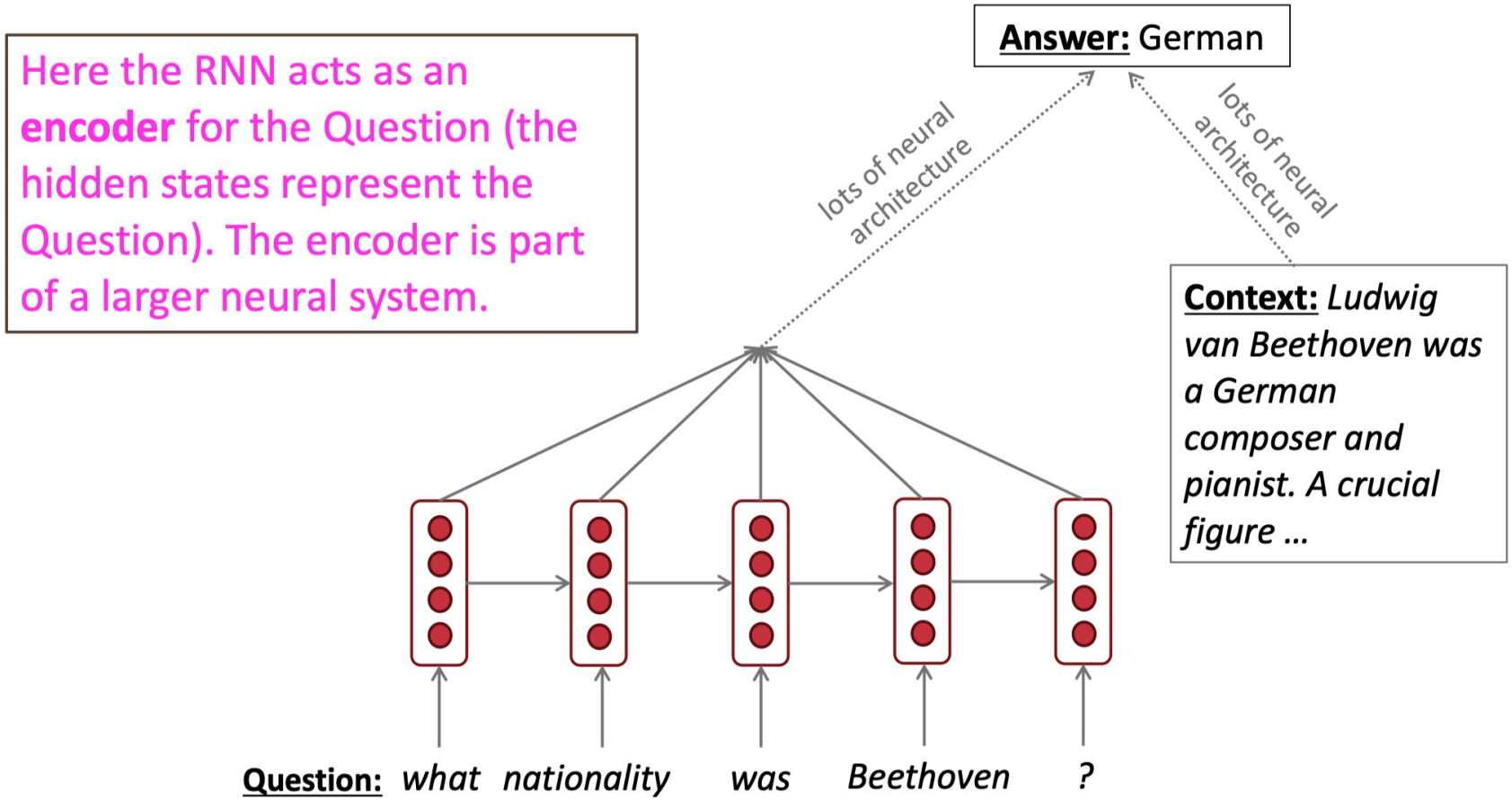
RNNs can be used for sentence classification

e.g., sentiment classification



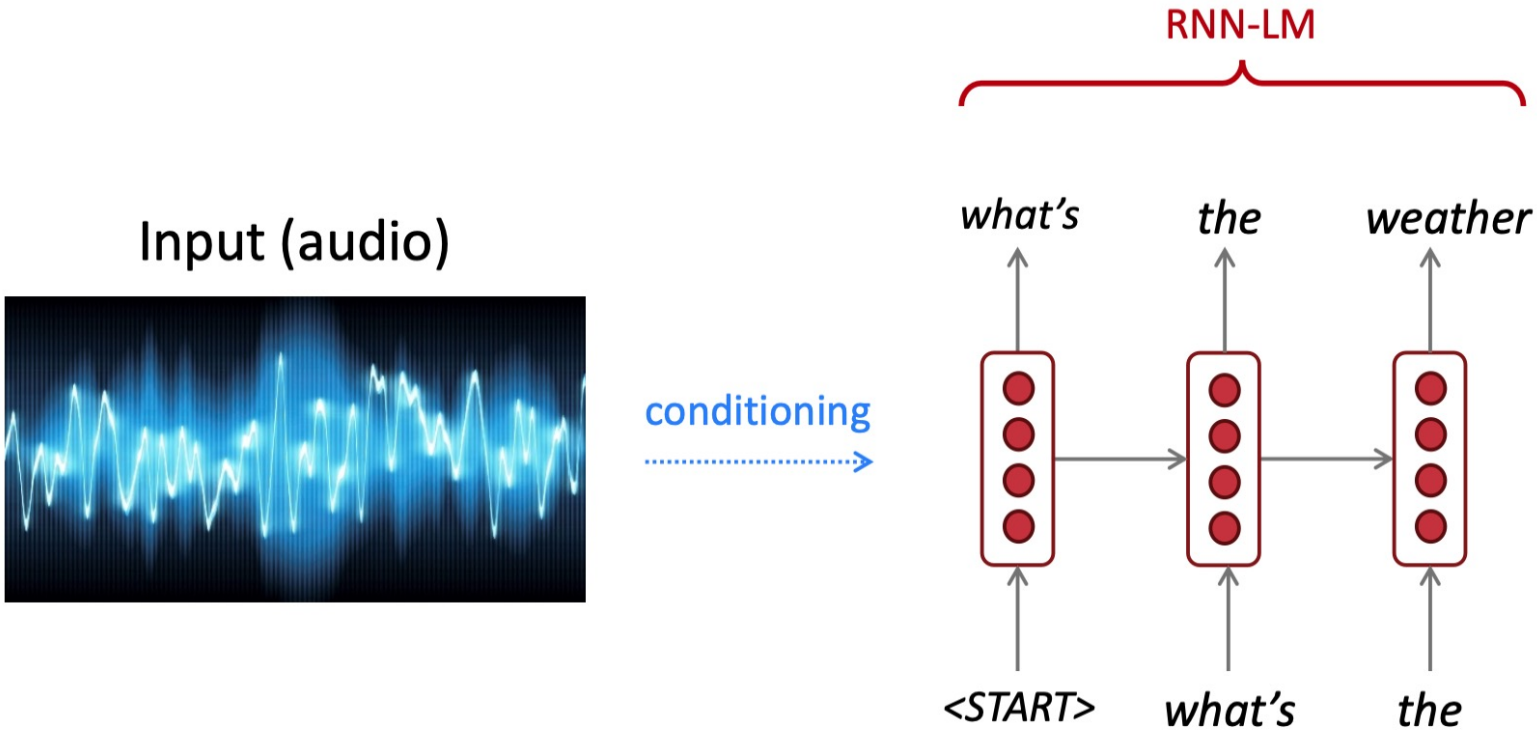
RNNs can be used as an encoder module

e.g., question answering, machine translation, many other tasks!



RNN-LMs can be used to generate text

e.g., speech recognition, machine translation, summarization



This is an example of a *conditional language model*.
We'll see Machine Translation in much more detail starting next lecture.

Terminology and a look forward

e.g., speech recognition, machine translation, summarization

The RNN described in this lecture = **simple**/vanilla/**Elman** RNN 

Next lecture: You will learn about other RNN flavors

like LSTM  and GRU  and multi-layer RNNs 

By the end of the course: You will understand phrases like
"stacked bidirectional LSTMs with residual connections and self-attention"

References I

Y. Bengio, R. Ducharme, and P. Vincent. A neural probabilistic language model. *Advances in neural information processing systems*, 13, 2000.