CSCI 567: Machine Learning

Vatsal Sharan

Spring 2024

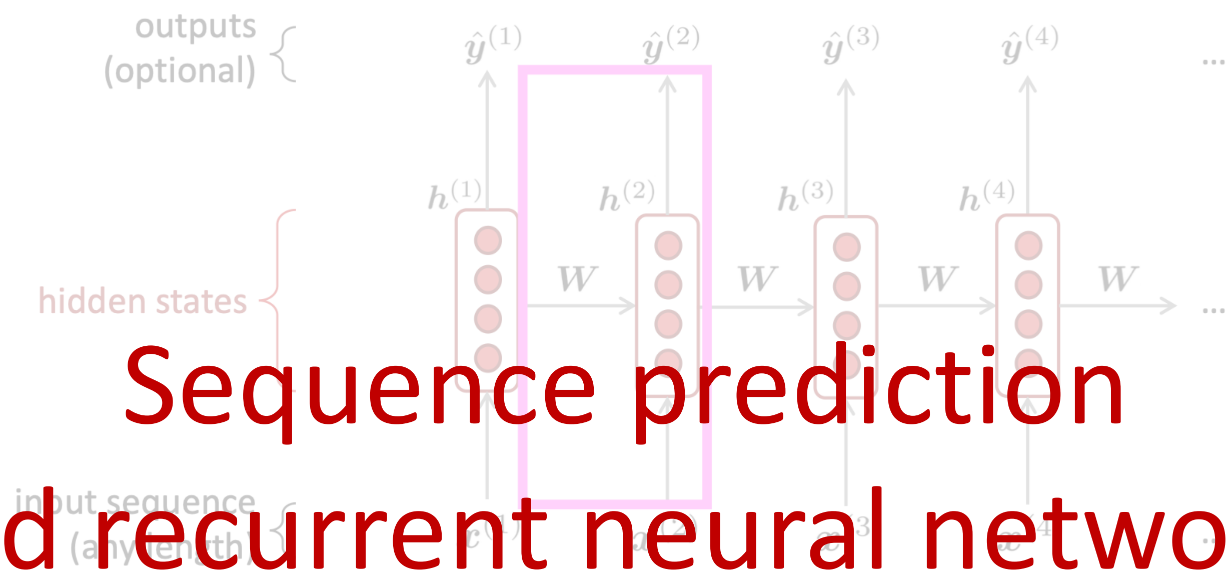
Lecture 8, March 8

USC University of Southern California

Administrivia

- HW3 due in less than 3 weeks
- No office hours next week due to spring break
- Project proposals due today on Gradescope & Google form

- Today's plan:
  - Sequential prediction, Markov models, recurrent neural networks, attention & Transformers



Sequence prediction and recurrent neural networks

Acknowledgements

We borrow heavily from:

- Stanford's CS224n: https://web.stanford.edu/class/cs224n/

Sequential prediction

Given observations $\underbrace{x_1, x_2, \ldots, x_{t-1}}_{\text{input}}$ what is $\underbrace{x_t}_{\text{output}}$?

Examples:

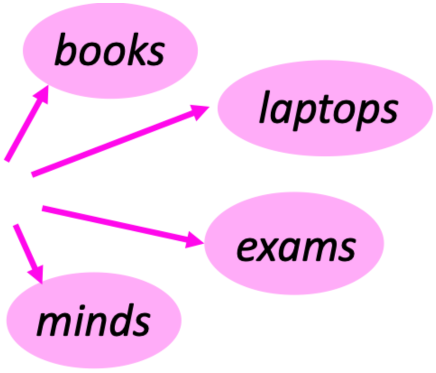
- text or speech data
- stock market data
- weather data
- …

In this lecture, we will mostly focus on text data (**language modelling**).

Language modelling

Language modelling is the task of predicting what word comes next:

the students opened their ______



r.v. over the randomness in the sentence, context etc.

More formally, let X_i be the random variable for the i-th word in the sentence, and let x_i be the value taken by the random variable. Then the goal is to compute

$$P(X_{t+1}|X_t = x_t, \ldots, X_1 = x_1).$$

A system that does this is known as a **Language Model.**

Language modelling

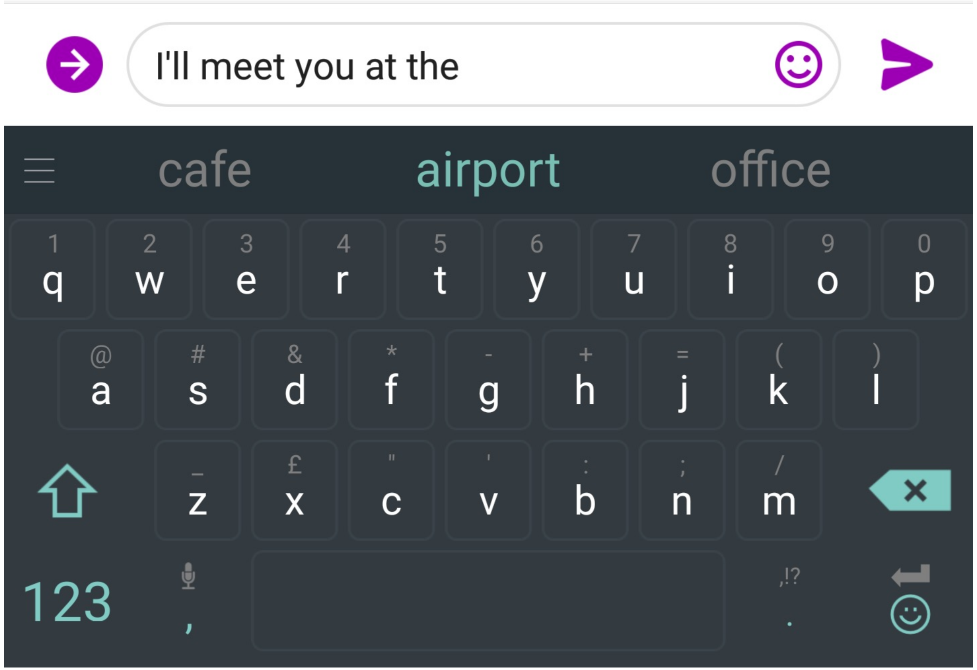
We can also think of a Language Model as a system that *assigns a probability to a piece of text.*

For example, if we have some text $x_1, \ldots, x_T$, then the probability of this text (according to the Language Model) is:

$$
\begin{aligned}
P(X_1 = x_1, \ldots, X_T = x_T) &= P(X_1 = x_1) \times P(X_2 = x_2 | X_1 = x_1) \\
&\quad \times \cdots \times P(X_T = x_T | X_{T-1} = x_{T-1}, \ldots, X_1 = x_1) \\
&= \Pi_{t=1}^{T} P(X_t = x_t | X_{t-1} = x_{t-1}, \ldots, X_1 = x_1).
\end{aligned}
$$

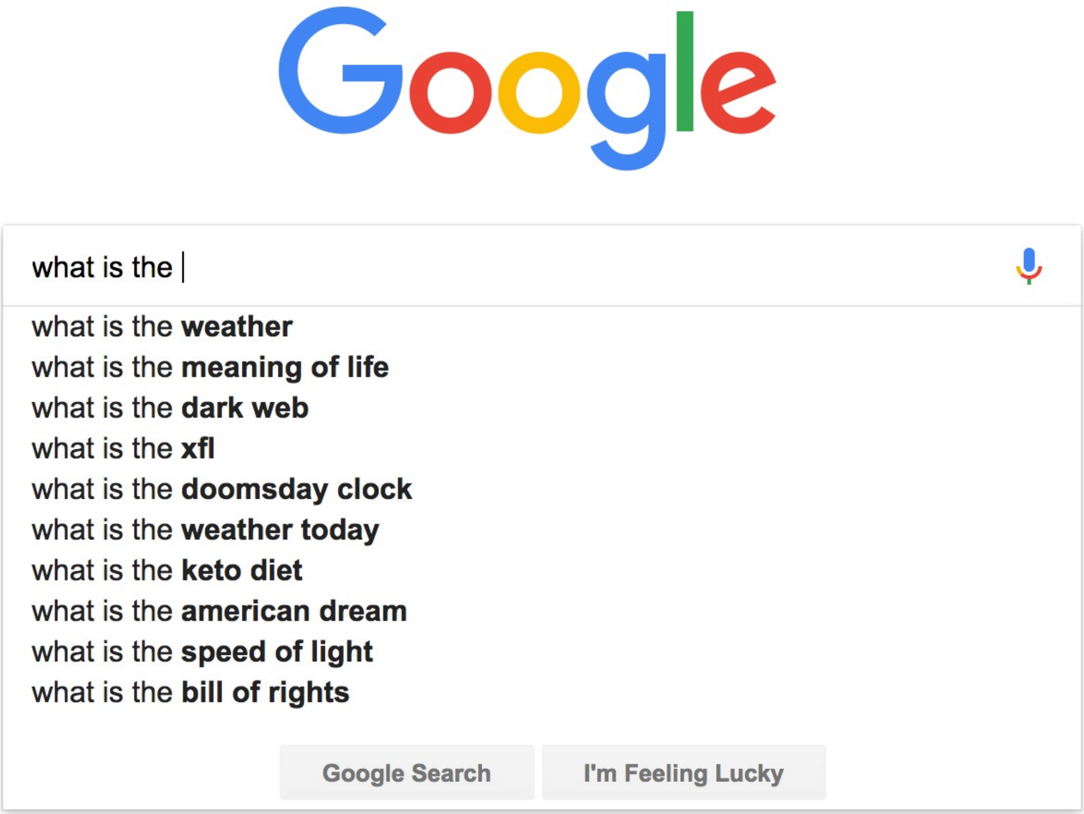
Slide adapted from CS224n by Chris Manning (Lecture 5)

You use Language Models every day!



Slide adapted from CS224n by Chris Manning (Lecture 5)

You use Language Models every day!



Slide adapted from CS224n by Chris Manning (Lecture 5)

n-gram Language Models

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 next word.

Slide adapted from CS224n by Chris Manning (Lecture 5)

n-gram language model: A type of Markov model

A **Markov model or Markov chain** is a sequence of random variables with the **Markov property**: a sequence of random variables $X_1, X_2, \cdots$ s.t.

$$P(X_{t+1} \mid X_{1:t}) = P(X_{t+1} \mid X_t) \qquad \text{(Markov property)}$$

i.e. *the next state only depends on the most recent state* (notation $X_{1:t}$ denotes the sequence $X_1, \ldots, X_t$). This is a *bigram model*.

We will consider the following setting:

- All X_t's take value from the same discrete set $\{1, \ldots, S\}$ → the size of dictionary of all possible words
- $P(X_{t+1} = s' \mid X_t = s) = a_{s,s'}$, known as transition probability
- $P(X_1 = s) = \pi_s$ → initial probability
- $(\{\pi_s\}, \{a_{s,s'}\}) = (\boldsymbol{\pi}, \boldsymbol{A})$ are parameters of the model (s, s') entry of A is $a_{s,s'}$

$$P(X_1, \ldots, X_T) = P(X_1) \cdot P(X_2 \mid X_1) \cdot P(X_3 \mid X_2) \cdots P(X_T \mid X_{T-1})$$

Markov model: examples

- Example 1 (**Language model**)
  States $[S]$ represent a dictionary of words,

$$a_{\text{ice,cream}} = P(X_{t+1} = \text{cream} \mid X_t = \text{ice})$$

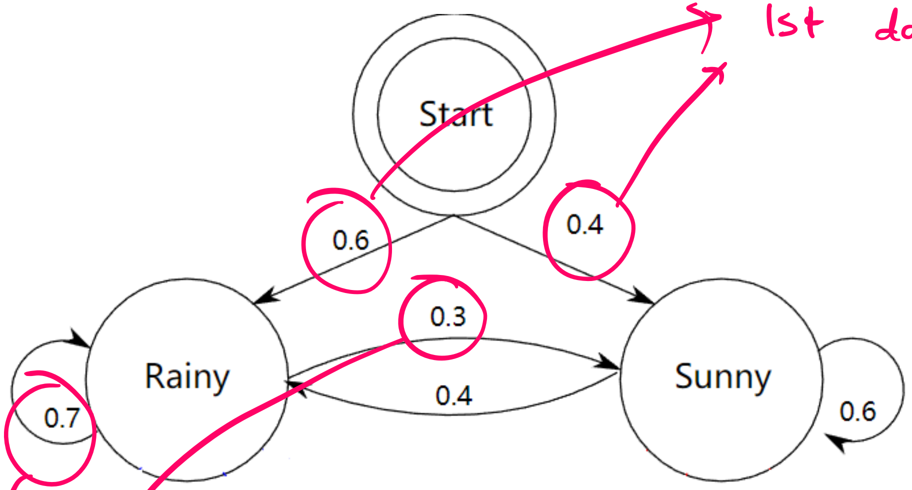
  is an example of the transition probability.

- Example 2 (**Weather**)
  States $[S]$ represent weather at each day

$$a_{\text{sunny,rainy}} = P(X_{t+1} = \text{rainy} \mid X_t = \text{sunny})$$

Markov model: Graphical representation

A Markov model is nicely represented as a **directed graph**



1st day is { Rainy, 60% prob.
Sunny, 40% prob.

if today is Rainy, tomorrow is { Rainy, 70% prob.
Sunny, 30% prob.

Learning Markov models

Now suppose we have observed n **sequences of examples**:

- $x_{1,1}, \ldots, x_{1,T}$
- $\cdots$
- $x_{i,1}, \ldots, x_{i,T}$
- $\cdots$
- $x_{n,1}, \ldots, x_{n,T}$

(rainy, sunny, ..., rainy)

⋮

where

- for simplicity we assume each sequence has the same length T
- lower case $x_{i,t}$ represents the value of the random variable $X_{i,t}$

From these observations how do we *learn the model parameters* $(\boldsymbol{\pi}, \boldsymbol{A})$?

Learning Markov models: MLE

Same story, find the **MLE**. The log-likelihood of a sequence $x_1, \ldots, x_T$ is

$$\ln P(X_{1:T} = x_{1:T})$$

$$= \sum_{t=1}^{T} \ln P(X_t = x_t \mid X_{1:t-1} = x_{1:t-1}) \qquad \text{(always true)}$$

$$= \sum_{t=1}^{T} \ln P(X_t = x_t \mid X_{t-1} = x_{t-1}) \qquad \text{(Markov property)}$$

$$= \ln \pi_{x_1} + \sum_{t=2}^{T} \ln a_{x_{t-1}, x_t}$$

$P(X_1 = x_1) = \pi_{x_1}$

Prob. of transitioning from $x_{t-1} \rightarrow x_t$

$$= \sum_{s} \mathbb{I}[x_1 = s] \ln \pi_s + \sum_{s,s'} \left(\sum_{t=2}^{T} \mathbb{I}[x_{t-1} = s, x_t = s'] \right) \ln a_{s,s'}$$

This is over one sequence, can sum over all.

Learning Markov models: MLE

s, s' entry is $a_{s,s'}$

So MLE is

$$\underset{\boldsymbol{\pi}, \boldsymbol{A}}{\operatorname{argmax}} \sum_s (\textbf{\#initial states with value } s) \ln \pi_s + \sum_{s,s'} (\textbf{\#transitions from } s \textbf{ to } s') \ln a_{s,s'}$$

This is an optimization problem, and can be solved by hand (though we'll skip in class).
The solution is:

$$\pi_s = \frac{\text{\#initial states with value } s}{\text{\#initial states}}$$

$$a_{s,s'} = \frac{\text{\#transitions from } s \text{ to } s'}{\text{\#transitions from } s \text{ to any state}}$$

Learning Markov models: Another perspective

Let's first look at the transition probabilities. By the Markov assumption,

$$P(X_{t+1} = x_{t+1} \mid X_t = x_t, \ldots, X_1 = x_1) = P(X_{t+1} = x_{t+1} \mid X_t = x_t)$$

Using the definition of conditional probability,

$$P(X_{t+1} = x_{t+1} \mid X_t = x_t) = \frac{P(X_{t+1} = x_{t+1}, X_t = x_t)}{P(X_t = x_t)}$$

We can estimate this using data,

$$\frac{P(X_{t+1} = x_{t+1}, X_t = x_t)}{P(X_t = x_t)} \approx \frac{\text{\#times } (x_t, x_{t+1}) \text{ appears}}{\text{\# times } (x_t) \text{ appears (and is not the last state)}}$$

~~/ # observations~~

~~/ # observations~~

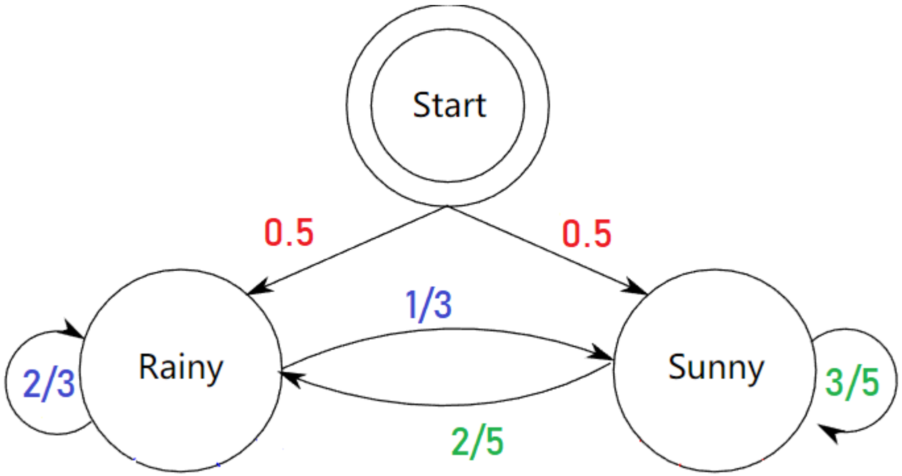
The initial state distribution follows similarly,

$$P(X_1 = s) \approx \frac{\text{\#times } s \text{ is first state}}{\text{\#sequences}}$$

Learning Markov models: Example

Suppose we observed the following 2 sequences of length 5

- sunny, sunny, rainy, rainy, rainy
- rainy, sunny, sunny, sunny, rainy



Higher-order Markov models

Is the Markov assumption reasonable? Not so in many cases, such as for language modeling.

Higher order Markov chains make it a bit more reasonable, e.g.

$$P(X_{t+1} \mid X_t, \ldots, X_1) = P(X_{t+1} \mid X_t, X_{t-1}) \qquad \text{(\textbf{second-order Markov assumption})}$$

i.e. the current word only depends on the last two words. This is a *trigram model*, since we need statistics of three words at a time to learn. In general, we can consider a n-th Markov model (or a *$(n+1)$-gram model*):

$$P(X_{t+1} \mid X_t, \ldots, X_1) = P(X_{t+1} \mid \overbrace{X_t, X_{t-1}, \ldots, X_{t-n+2}}^{\text{previous n observations}}) \quad \text{(\textbf{n-th order Markov assumption})}$$

Learning higher order Markov chains is similar, but more expensive.

$$\begin{aligned} P(X_{t+1} = x_{t+1} \mid X_t = x_t, \ldots, X_1 = x_1) &= P(X_{t+1} = x_{t+1} \mid X_t = x_t, X_{t-1} = x_{t-1}, \ldots, X_{t-n+2} = x_{t-n+2}) \\ &= \frac{P(X_{t+1} = x_{t+1}, X_t = x_t, X_{t-1} = x_{t-1}, \ldots, X_{t-n+2} = x_{t-n+2})}{P(X_t = x_t, X_{t-1} = x_{t-1}, \ldots, X_{t-n+2} = x_{t-n+2})} \\ &\approx \frac{\text{count}(x_{t-n+2}, \ldots, x_{t-1}, x_t, x_{t+1}) \text{ in the data}}{\text{count}(x_{t-n+2}, \ldots, x_{t-1}, x_t) \text{ in the data}} \end{aligned}$$

n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

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

discard

$\underbrace{\textit{students opened their}}_{\text{condition on this}}$

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

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

Slide adapted from CS224n by Chris Manning (Lecture 5)

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

Business and financial news

today the _______

get probability distribution

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

Notice that there isn't that much granularity in the distribution, because *"today the"* doesn't appear too often in corpus.
Most two-grams won't appear too often.

Generating text with a n-gram Language Model

You can also use a Language Model to generate text

today the ________

condition on this

get probability distribution

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

sample

Slide adapted from CS224n by Chris Manning (Lecture 5)

Generating text with a n-gram Language Model

You can also use a Language Model to generate text

today $\underbrace{\textit{the price}}_{\text{condition on this}}$ ________

get probability distribution

of	0.308
for	0.050
it	0.046
to	0.046
is	0.031
…	

sample

Generating text with a n-gram Language Model

You can also use a Language Model to generate text

today the price of ________

condition on this

get probability distribution

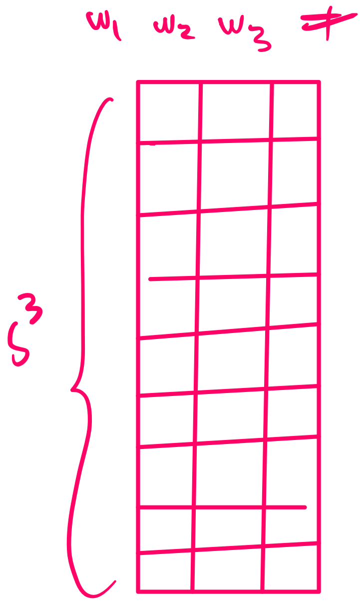
the	0.072
18	0.043
oil	0.043
its	0.036
gold	0.018
…	

sample

Slide adapted from CS224n by Chris Manning (Lecture 5)

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.

However, larger n increases model size and requires too much data to learn

Slide adapted from CS224n by Chris Manning (Lecture 5)

How to build a *neural* Language Model?

$x^{(t)}$ refer to both r.v. & the value it takes

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

- How about a window-based neural model?

Slide adapted from CS224n by Chris Manning (Lecture 5)

A fixed-window neural Language Model

~~*as the proctor started the clock*~~ *the students opened their* ______

discard

$\underbrace{\textit{the students opened their}}_{\text{fixed window}}$

Use a fixed window of previous words, and train a vanilla fully-connected neural network to predict the next word? → This is a standard supervised learning task.

Neural networks take vectors as inputs, how to give a word as input?

Approach 1: one-hot (sparse) encoding

Suppose vocabulary is of size S

'the' = [1, 0, 0, ... 0] → S-dim vector

'students' = [0, 1, 0, ... 0] → S-dim vector

① high-dimensional

② each representation is orthogonal, even similar words have orthogonal representations

Approach 2: **word embeddings/word vectors**

Word embeddings/vectors

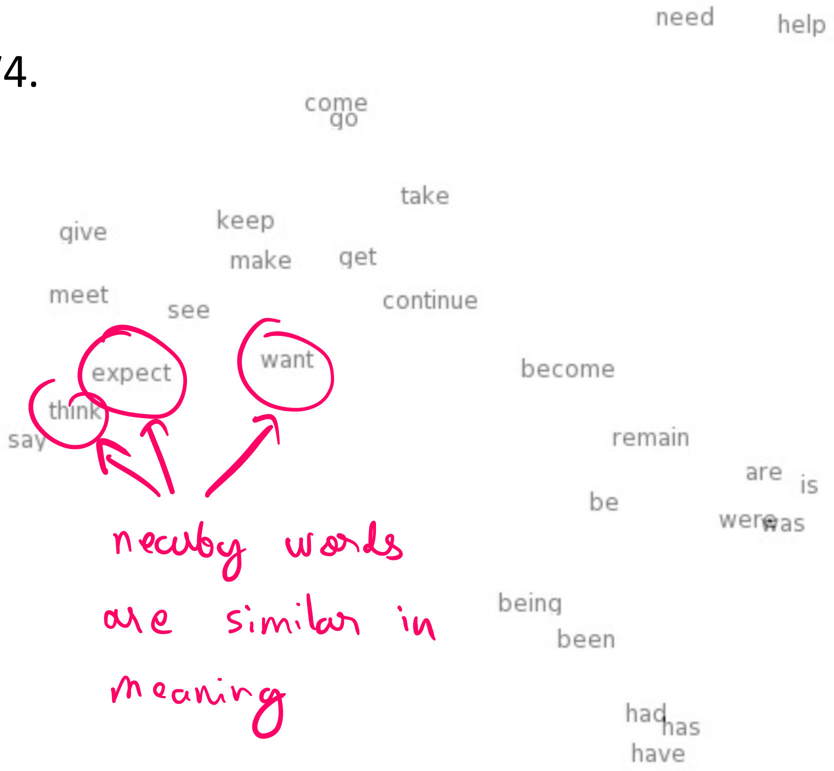
A word embedding is a (dense) mapping from words, to vector representations of the words.

Ideally, this mapping has the property that words similar in meaning have representations which are close to each other in the vector space.

You'll see a simple way to construct these in HW4.

$$\textit{expect} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

10-dim



nearby words are similar in meaning

Slide adapted from CS224n by Chris Manning (Lecture 1)

A fixed-window neural Language Model

same architecture as in HW3

output distribution

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

hidden layer

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

f: non-linearity (ReLU)

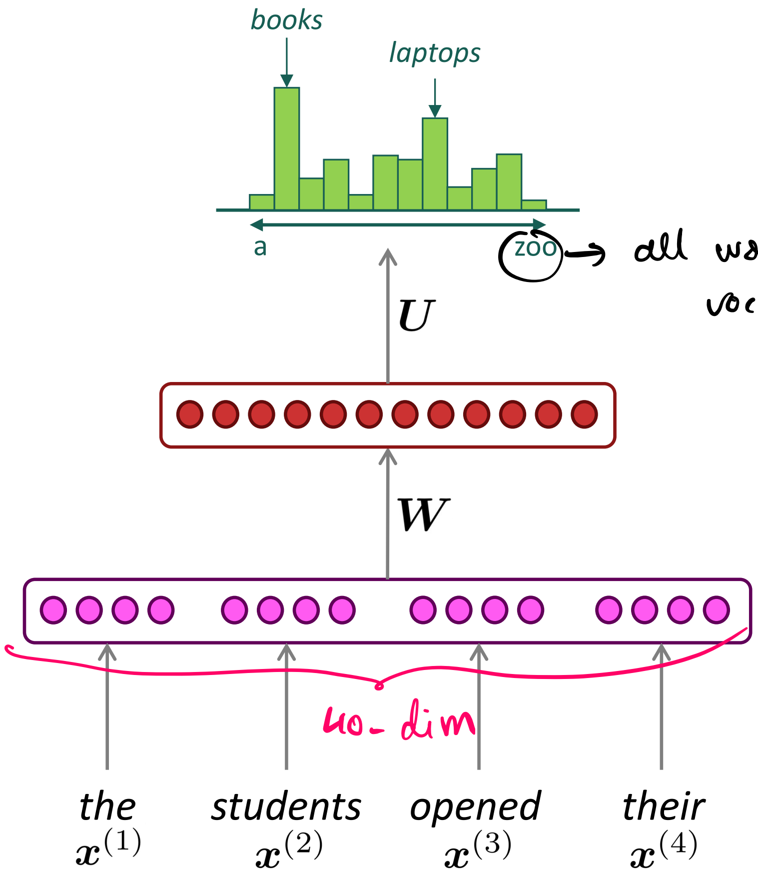
concatenated word embeddings

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

suppose each is 10-dim

words / one-hot vectors

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



all words in vocabulary

Slide adapted from CS224n by Chris Manning (Lecture 5)

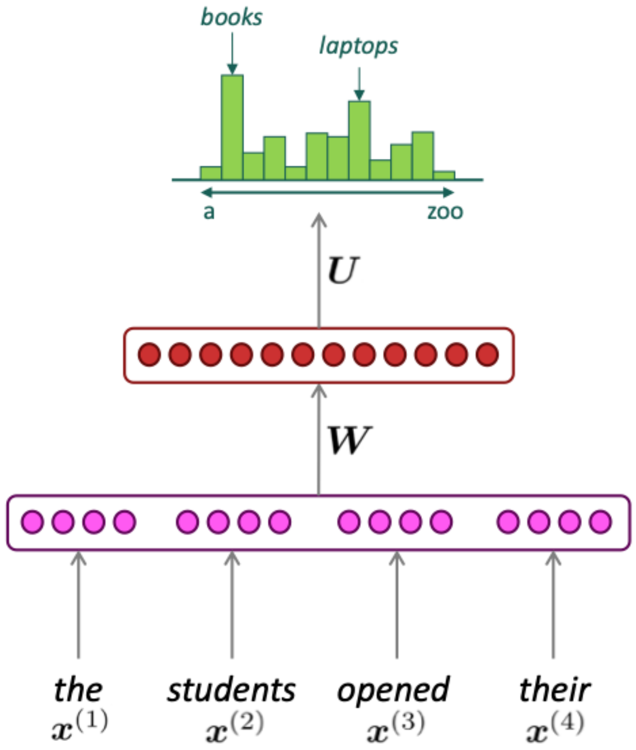
The problem with this architecture

- Uses a fixed window, which can be too small.
- Enlarging this window will enlarge the size of the weight matrix $\boldsymbol{W}$.
- The inputs $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ are multiplied by completely different weights in $\boldsymbol{W}$.

  No symmetry in how inputs are processed!

As with CNNs for images before, we need an architecture which has similar symmetries as the data.

In this case, ***can we have an architecture that can process any input length?***



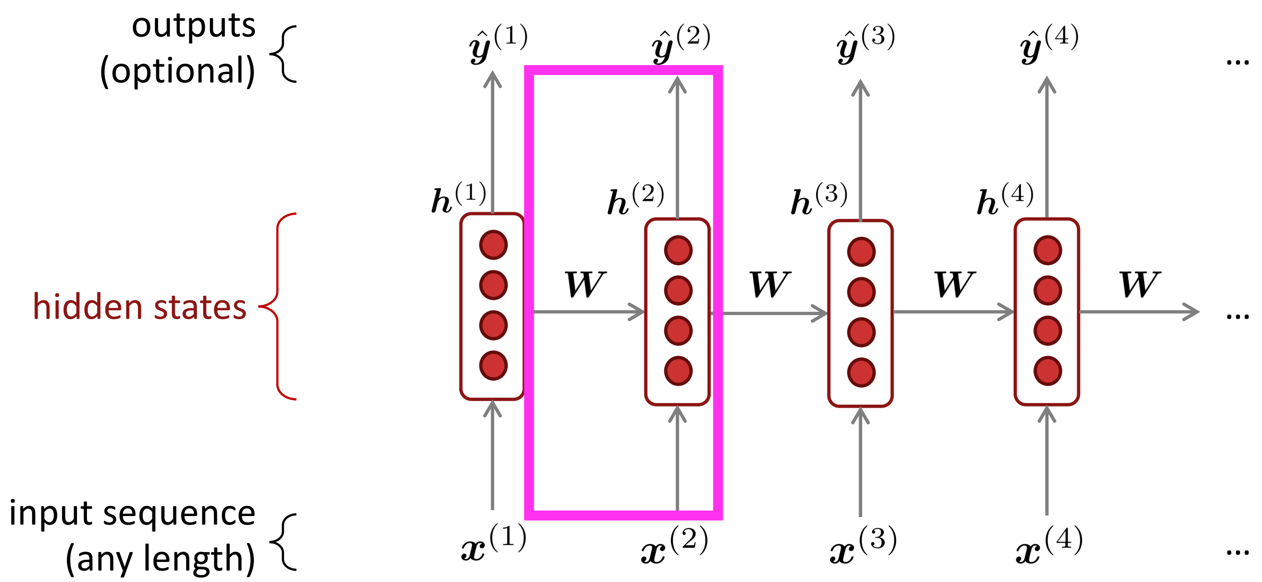
Slide adapted from CS224n by Chris Manning (Lecture 5)

Recurrent Neural Networks (RNN)

A family of neural architectures

Core idea: Apply the same weights W *repeatedly*

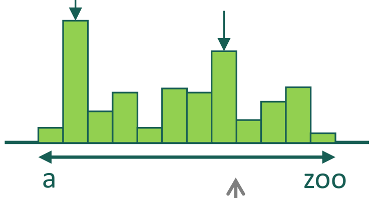
similar to filters



Slide adapted from CS224n by Chris Manning (Lecture 5)

A Simple RNN Language Model

$\hat{\boldsymbol{y}}^{(4)} = P(\boldsymbol{x}^{(5)} | \text{the students opened their})$



output distribution

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

hidden states

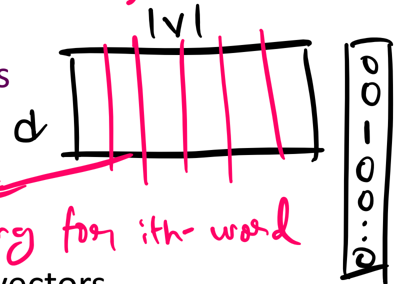
$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e \boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

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

σ : Activation (ReLU)

word embeddings

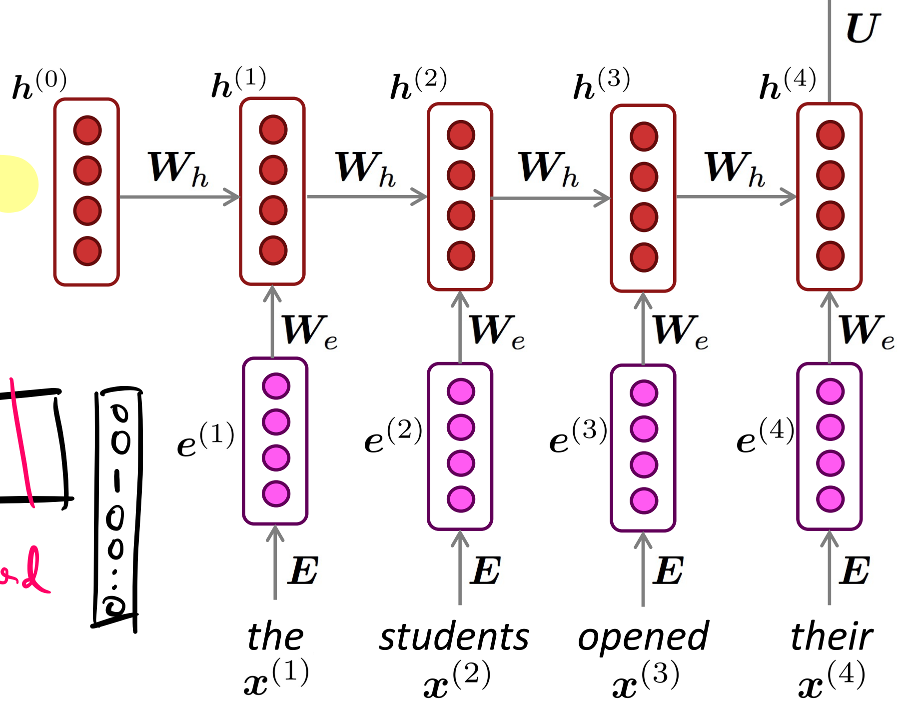
$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$



i-th column is embedding for ith- word

words / one-hot vectors

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



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

Slide adapted from CS224n by Chris Manning (Lecture 5)

Training an RNN Language Model

- Get a big corpus of text which is a sequence of words $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{\boldsymbol{y}}^{(t)}$ for *every step t*.
  - i.e. predict probability dist of *every word*, given words so far

- Loss function on step *t* is cross-entropy between predicted probability distribution $\hat{\boldsymbol{y}}^{(t)}$, and the true next word $\boldsymbol{y}^{(t)}$ (one-hot for $\boldsymbol{x}^{(t+1)}$):

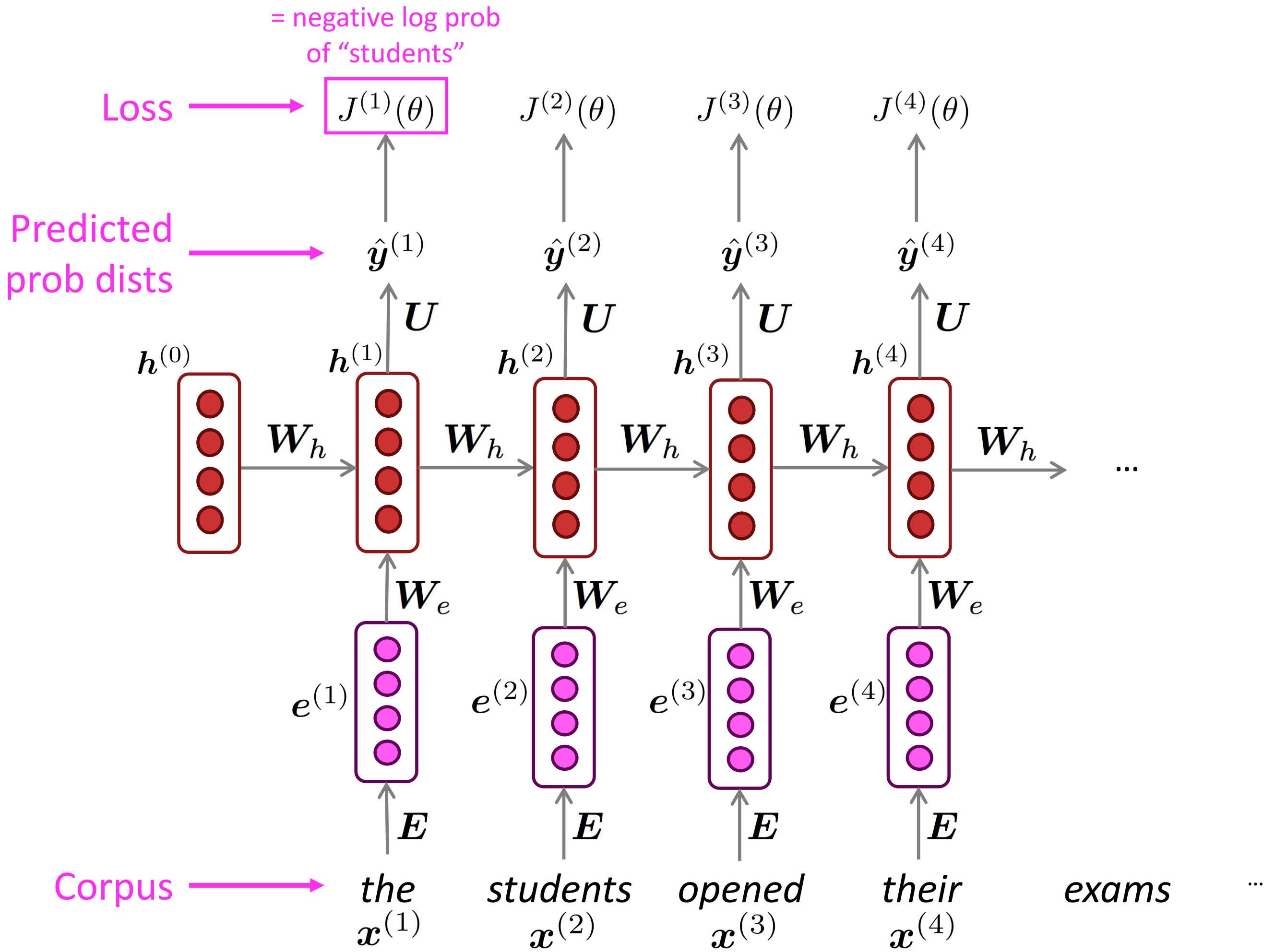
$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

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

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

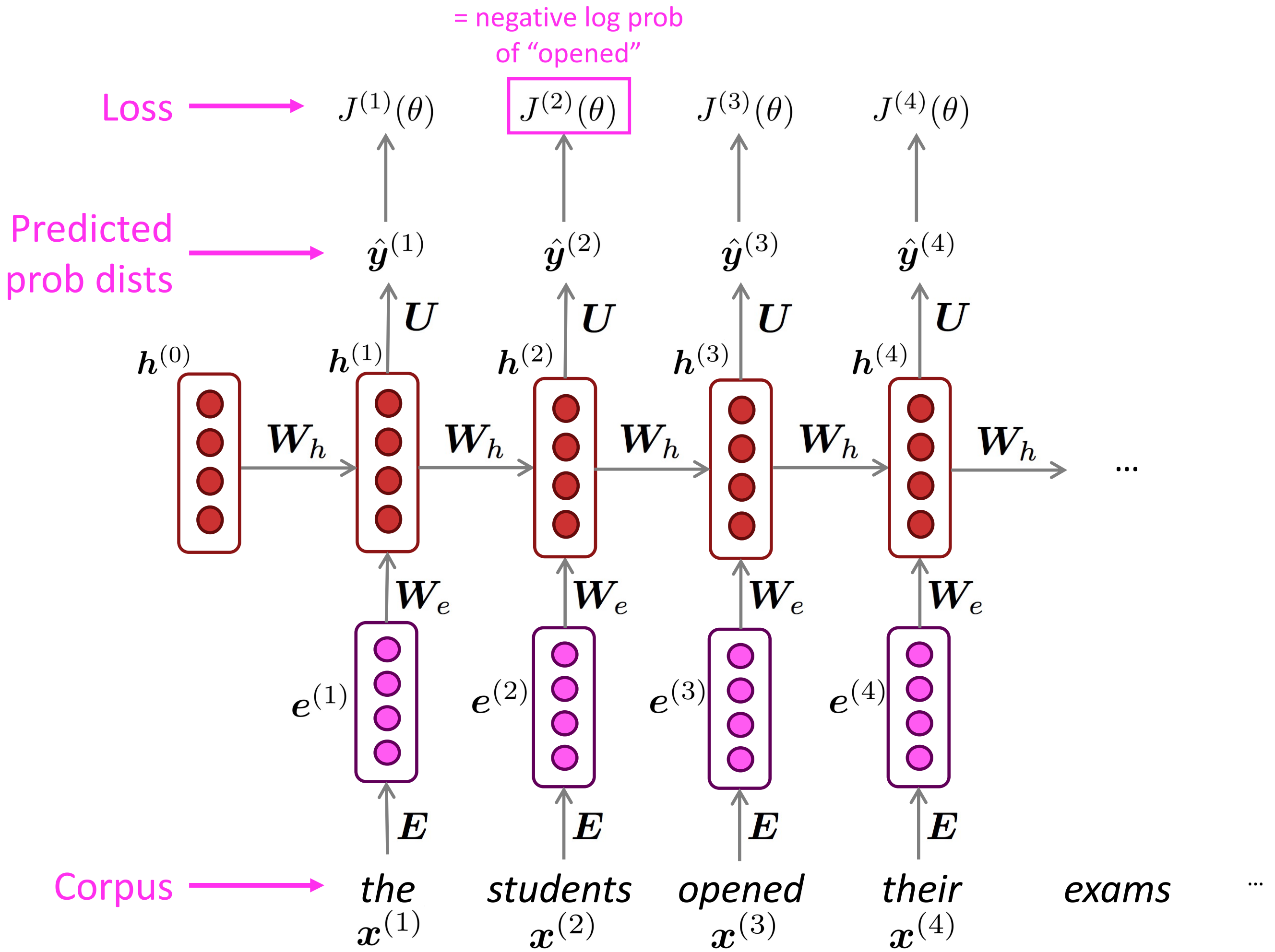
Slide adapted from CS224n by Chris Manning (Lecture 5)

Training an RNN Language Model



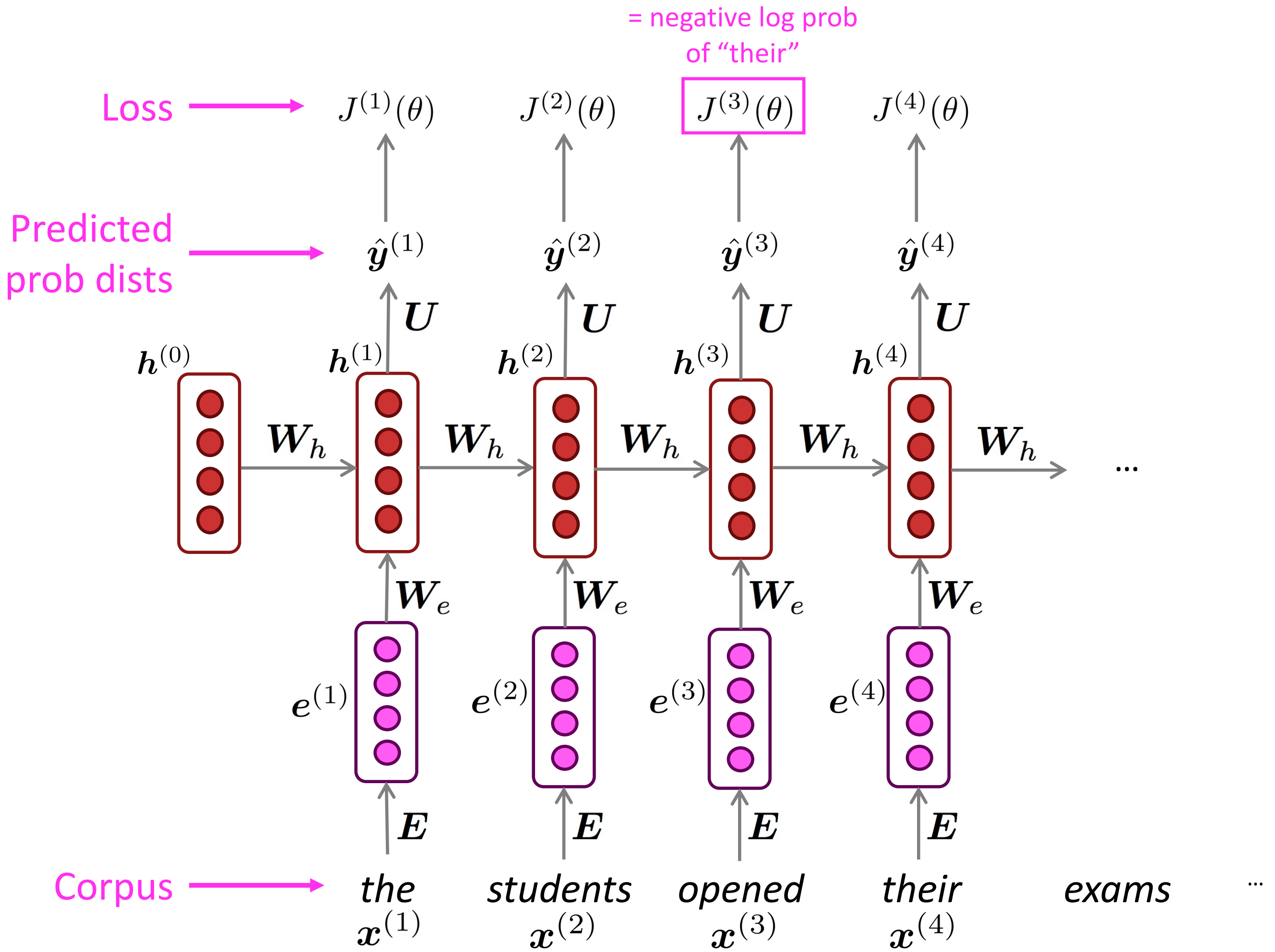
Slide adapted from CS224n by Chris Manning (Lecture 5)

Training an RNN Language Model



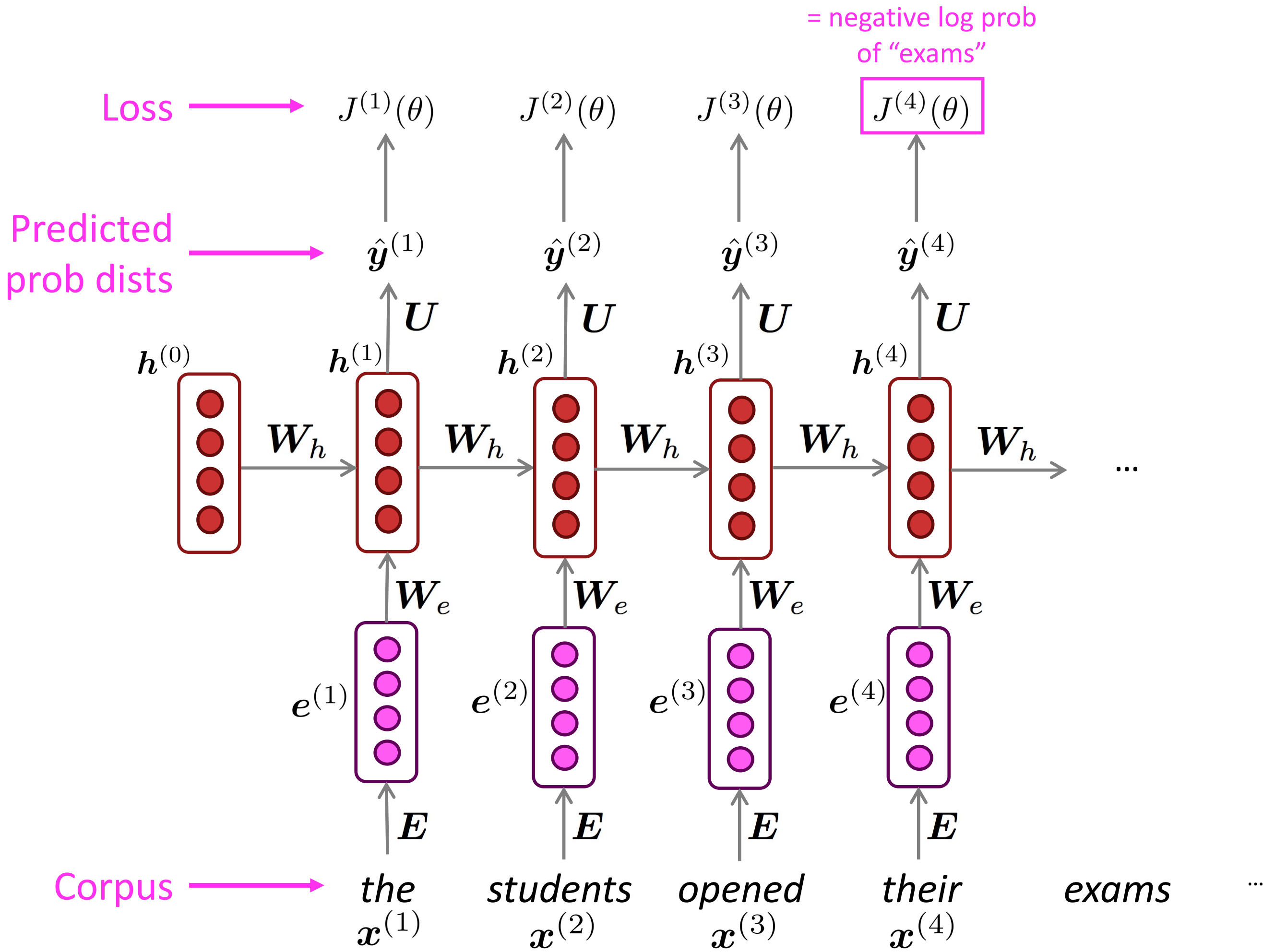
Slide adapted from CS224n by Chris Manning (Lecture 5)

Training an RNN Language Model



Slide adapted from CS224n by Chris Manning (Lecture 5)

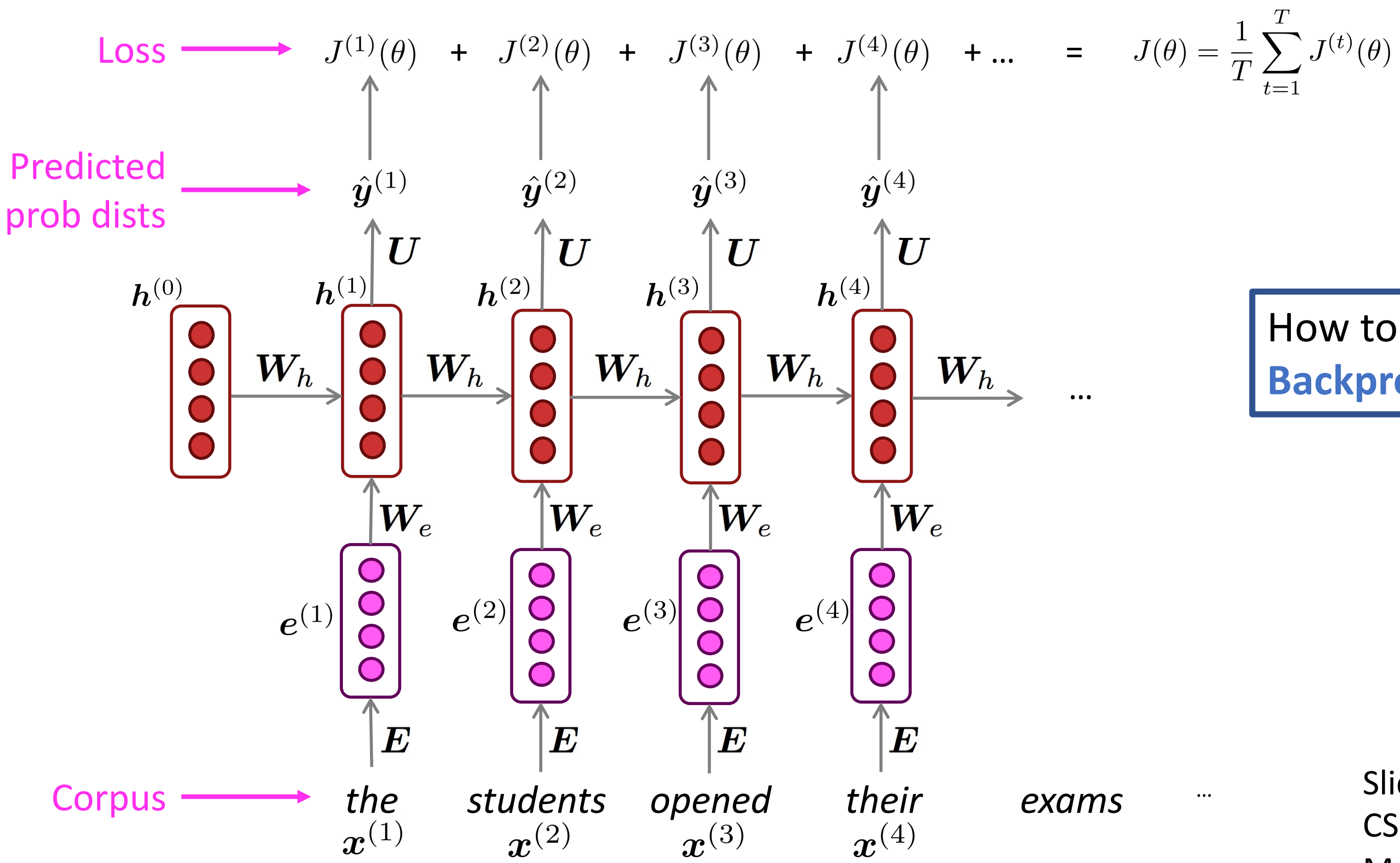
Training an RNN Language Model



Slide adapted from CS224n by Chris Manning (Lecture 5)

Training an RNN Language Model

"Teacher forcing"

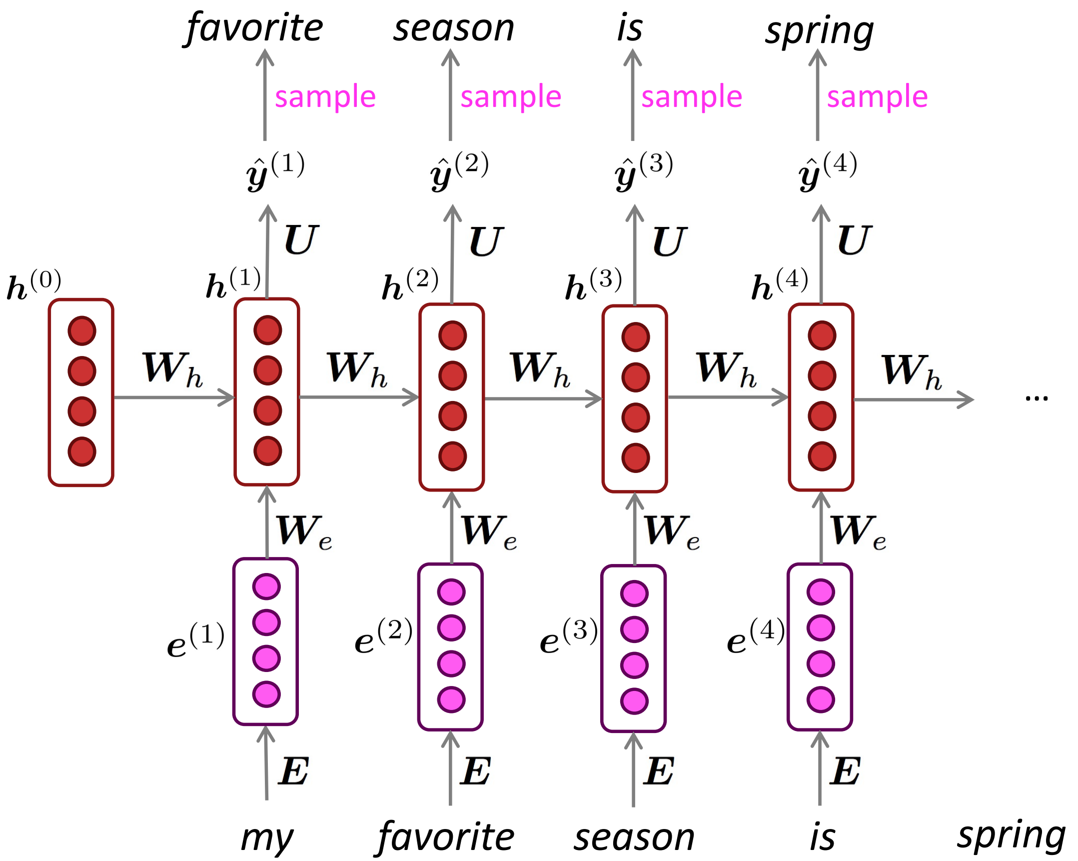


How to train this?
Backprop + SGD

Slide adapted from CS224n by Chris Manning (Lecture 5)

Generating text with a RNN Language Model

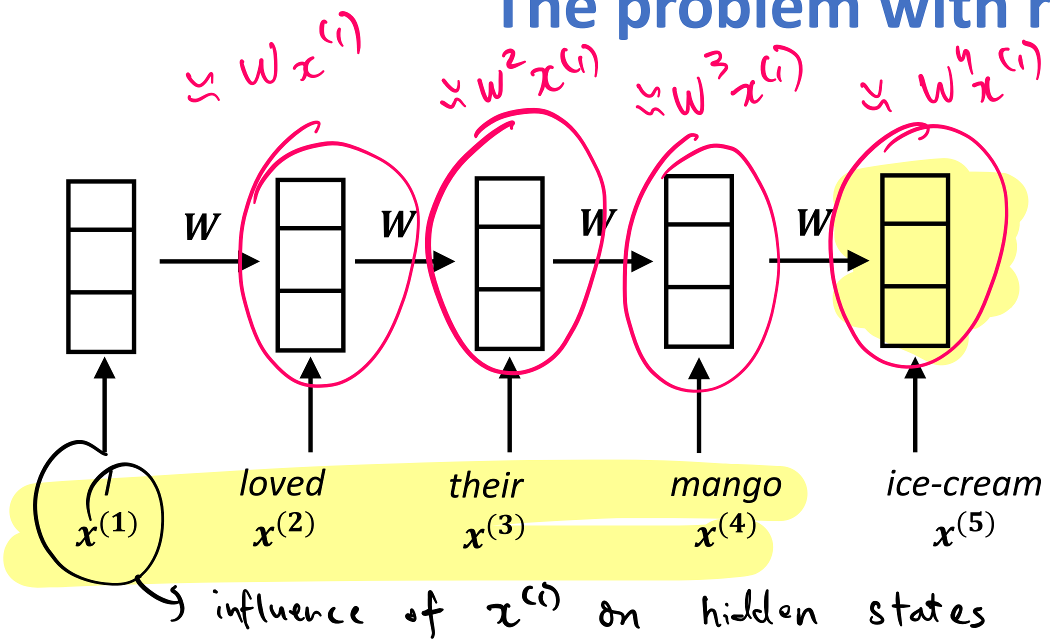
Just like a n-gram Language Model, you can use a RNN Language Model to generate text by repeated sampling. Sampled output becomes next step's input.



Slide adapted from CS224n by Chris Manning (Lecture 5)

Transformers

The problem with recurrence



1. Must always compress all necessary information into one hidden state representation

2. Cannot capture long-range dependencies in input ("vanishing gradients problem")

Inputs from sufficiently far away do not contribute to hidden state representation:

Suppose $$\boldsymbol{W} = \begin{pmatrix} 0.8 & 0.2 \\ -0.6 & 0.9 \end{pmatrix}$$

Then $$\boldsymbol{W}^5 = \begin{pmatrix} -0.31 & 0.35 \\ -1.06 & -0.13 \end{pmatrix}, \quad \boldsymbol{W}^{10} = \begin{pmatrix} -0.28 & -0.16 \\ 0.47 & -0.36 \end{pmatrix}, \quad \boldsymbol{W}^{50} = \begin{pmatrix} 0.01 & 0.00 \\ -0.01 & 0.01 \end{pmatrix}$$

A solution: Attention

Attention Is All You Need

USC
connection :)

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

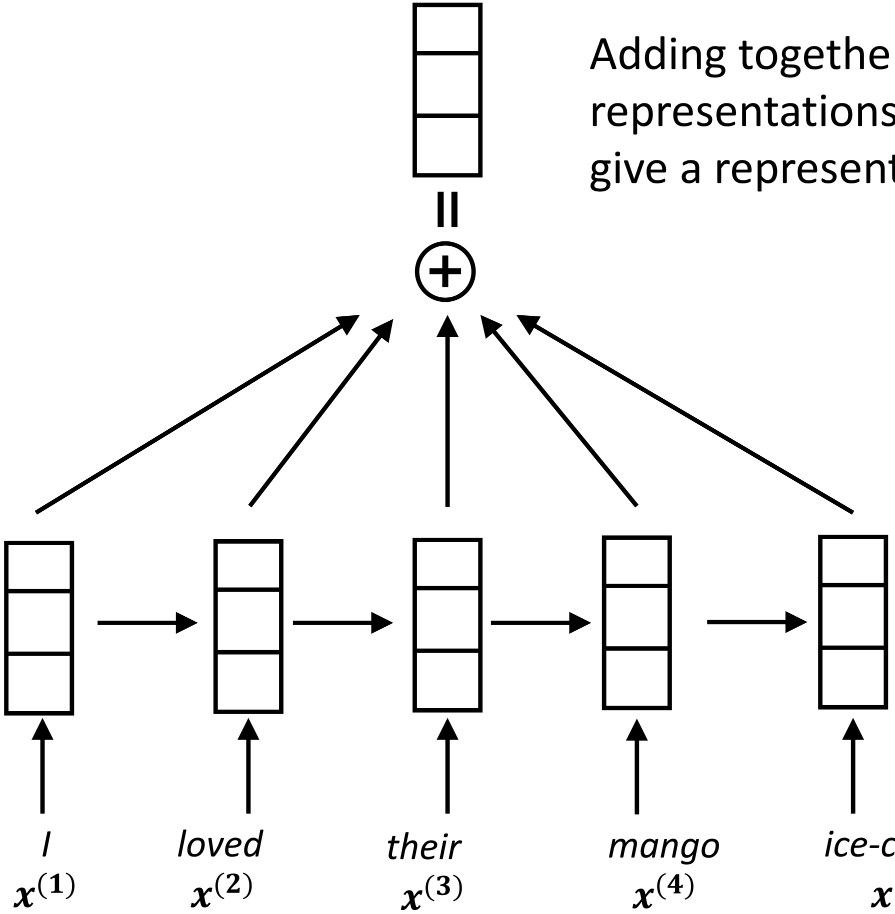
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

Starting point: Averaging word representations

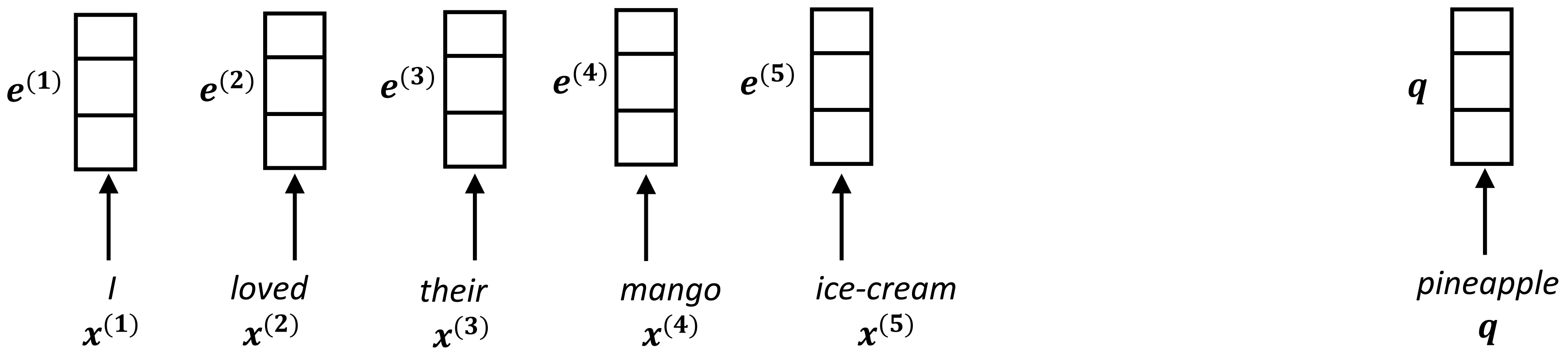


Adding together/averaging the representations of all words in sentence will give a representation of the sentence

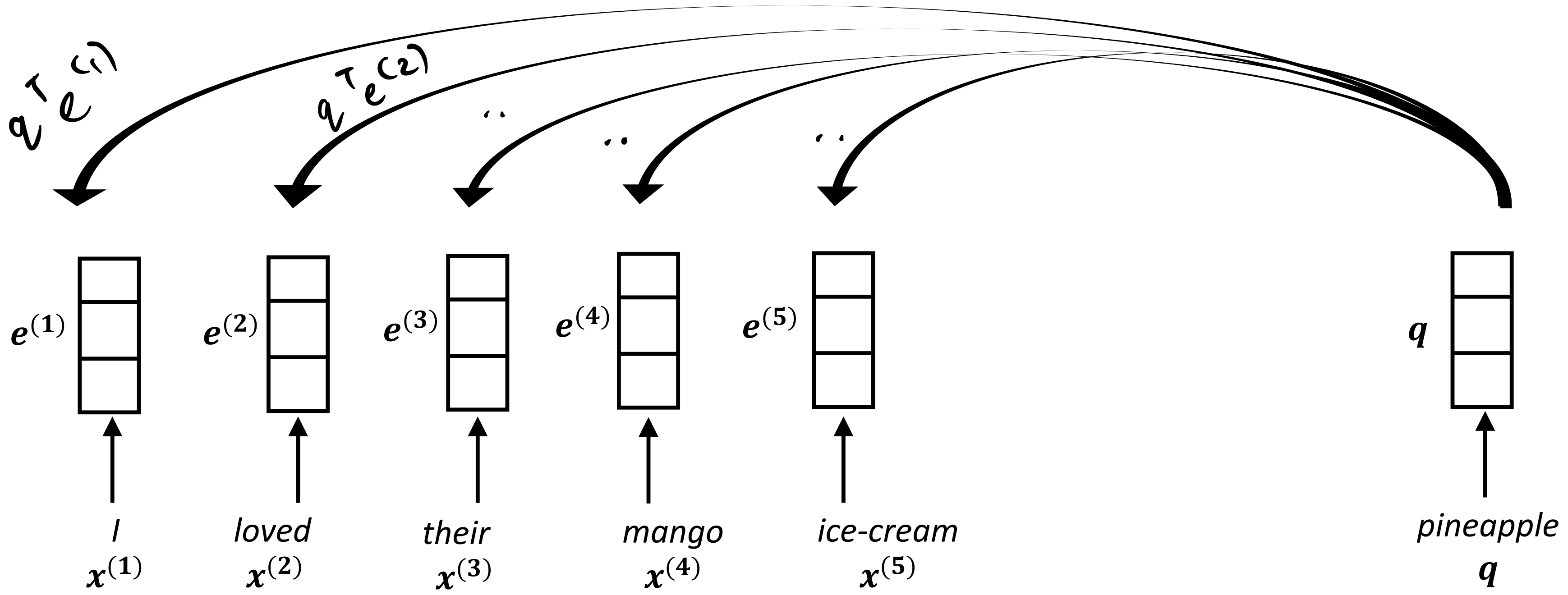
↓

used for sentiment analysis

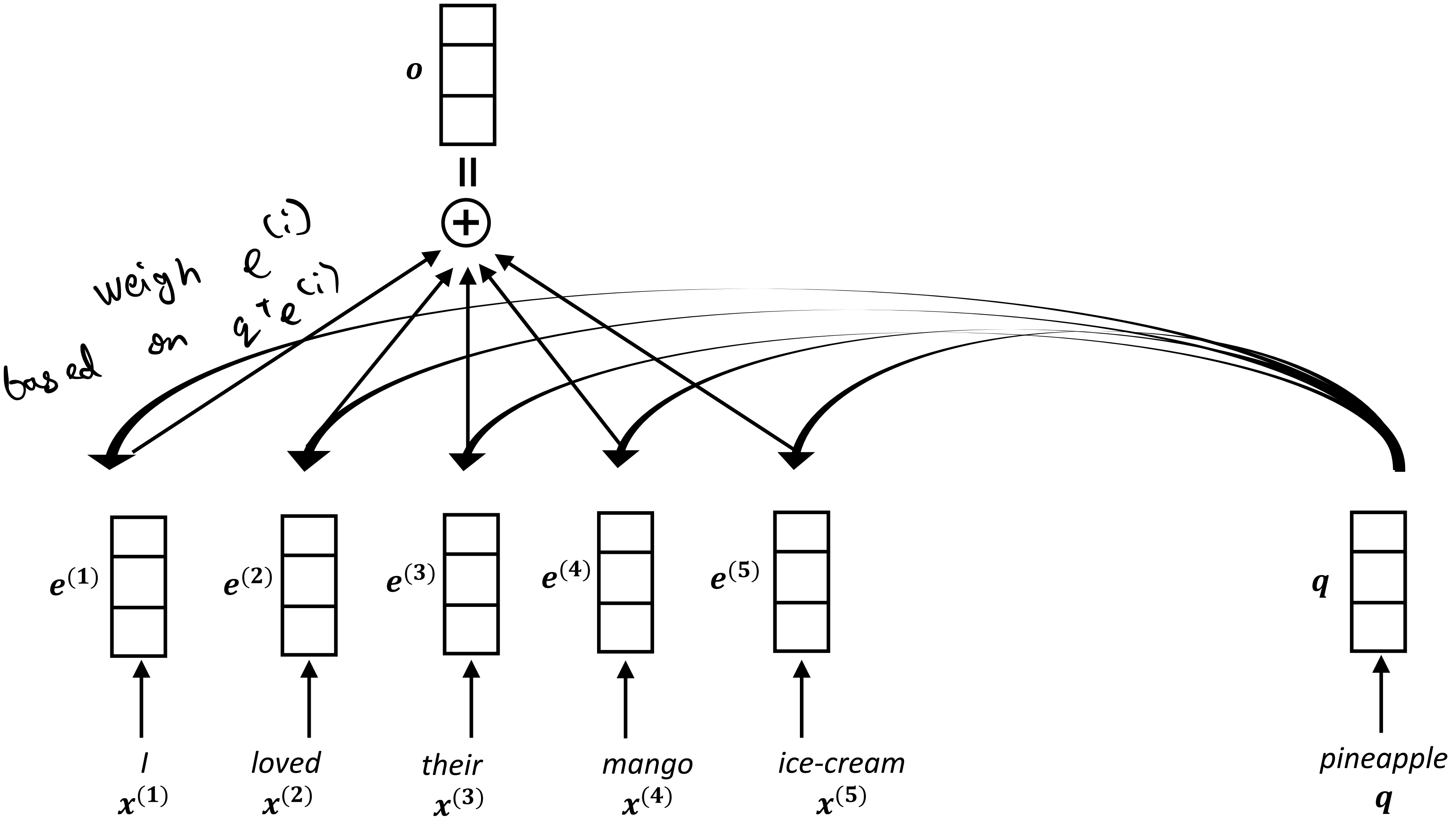
Attention: Weighted averaging



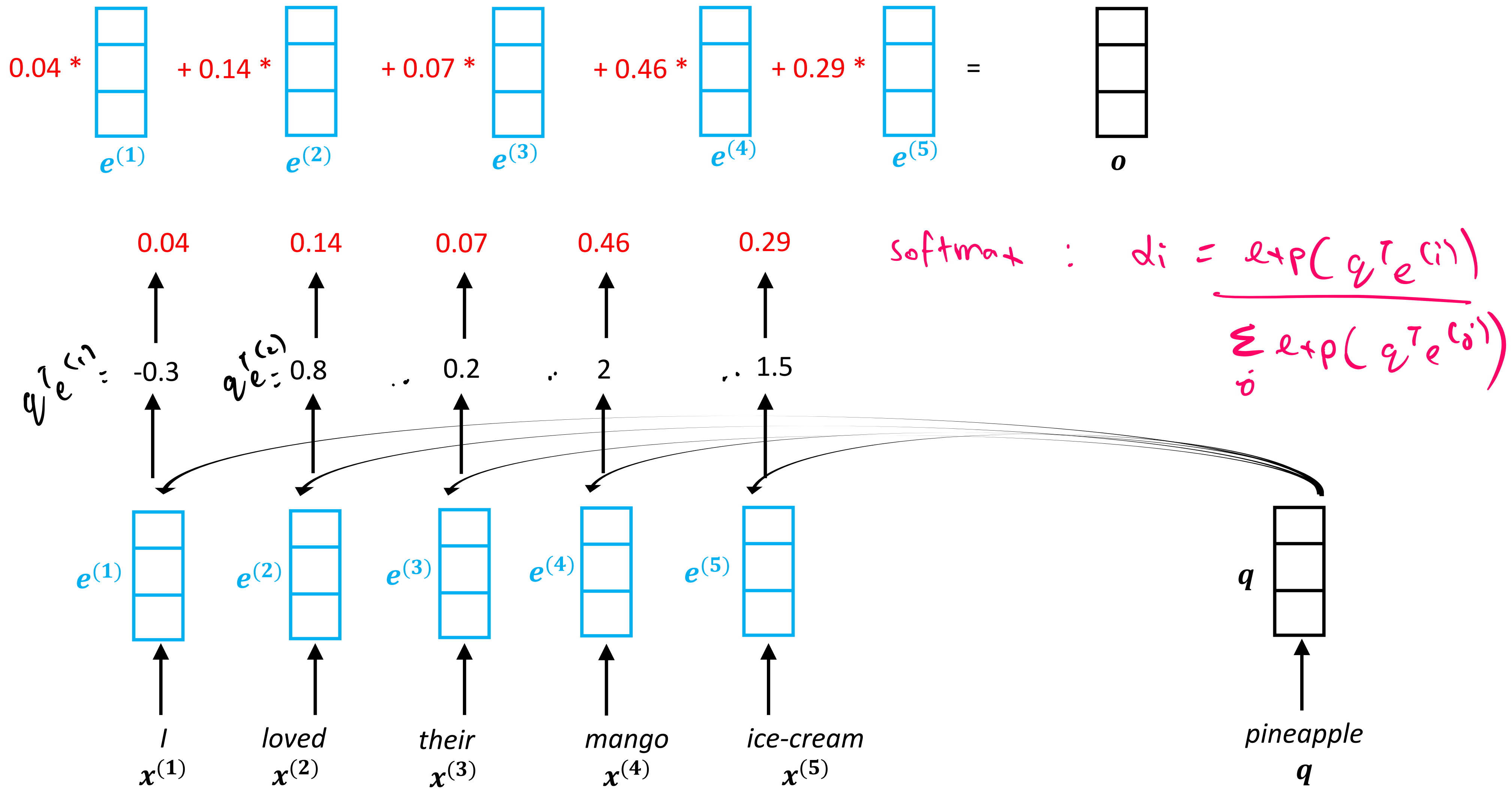
Attention: Weighted averaging



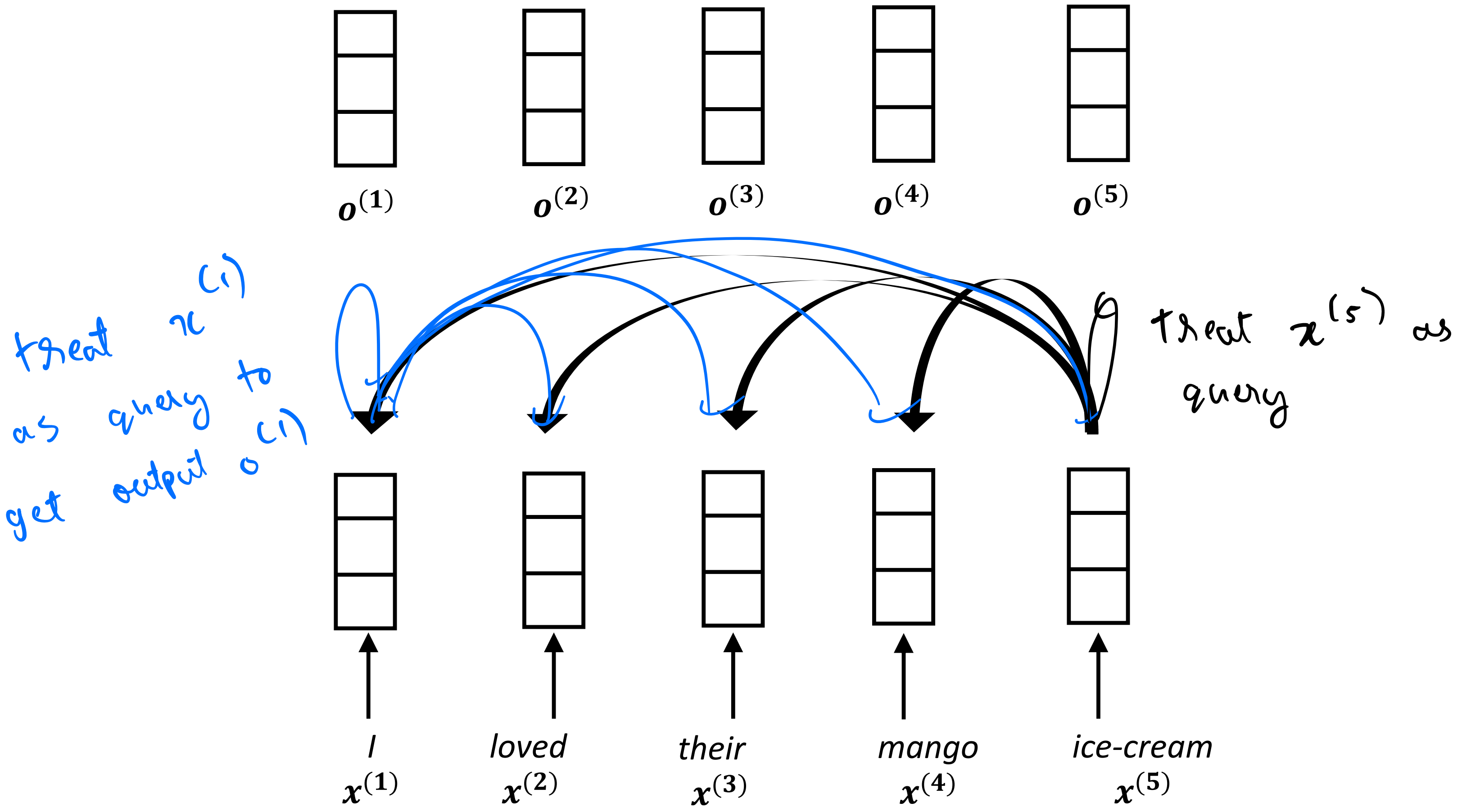
Attention: Weighted averaging



Attention: Weighted averaging



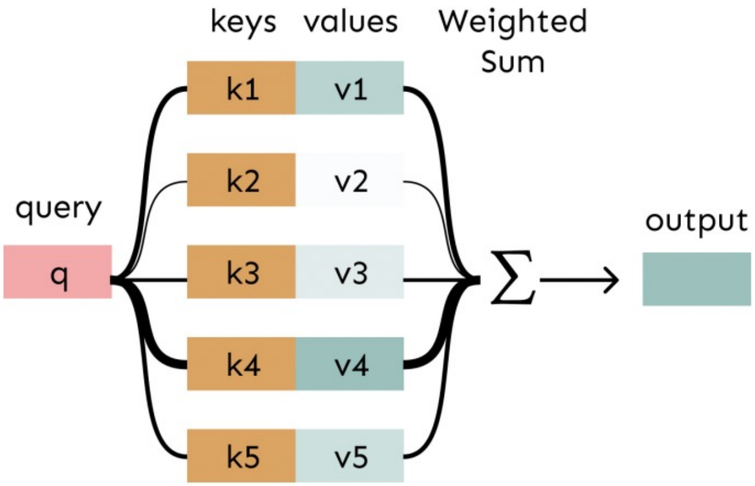
Self-attention

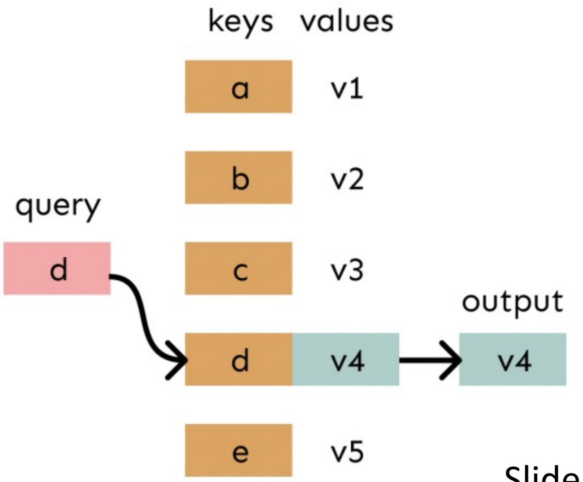


Attention as soft lookup

Attention: match query q to keys k1, k2, …, k5 to get weights between 0 and 1. Sum up values corresponding to each key with respective weight

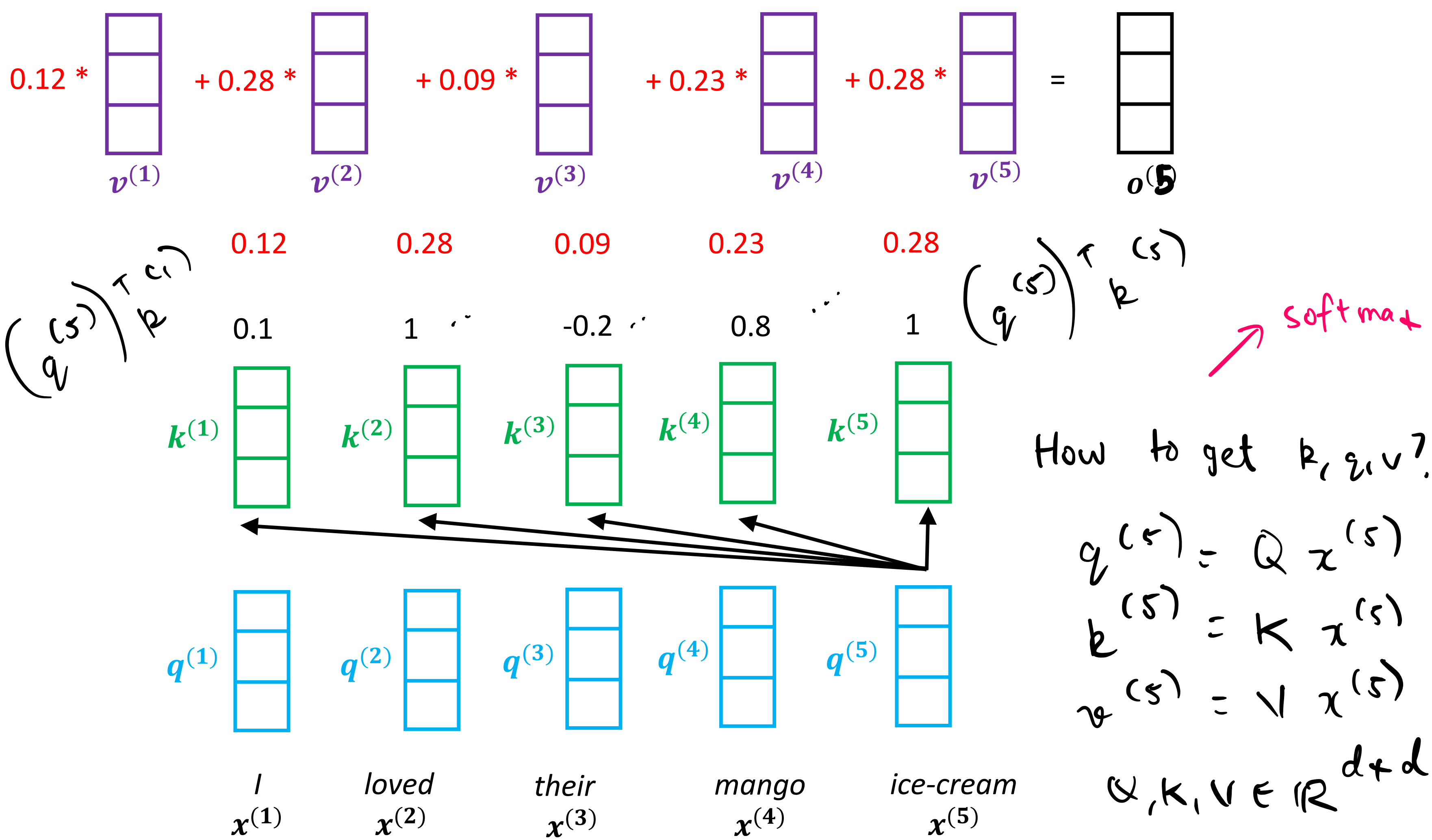
Lookup: find query in database, return value corresponding to its key





Slide adapted from CS224n by Chris Manning

Self-attention



How to get k, q, v?

$$q^{(5)} = Q\, x^{(5)}$$
$$k^{(5)} = K\, x^{(5)}$$
$$v^{(5)} = V\, x^{(5)}$$
$$Q, K, V \in \mathbb{R}^{d \times d}$$

Self-attention in matrix form

1. Transform each word embedding with weight matrices $\boldsymbol{Q}$, $\boldsymbol{K}$, $\boldsymbol{V}$, each in $\mathbb{R}^{d \times d}$

$$
\begin{aligned}
\boldsymbol{q}_i &= \boldsymbol{Q}\boldsymbol{x}_i \quad \text{(queries)} \\
\boldsymbol{k}_i &= \boldsymbol{K}\boldsymbol{x}_i \quad \text{(keys)} \\
\boldsymbol{v}_i &= \boldsymbol{V}\boldsymbol{x}_i \quad \text{(values)}
\end{aligned}
$$

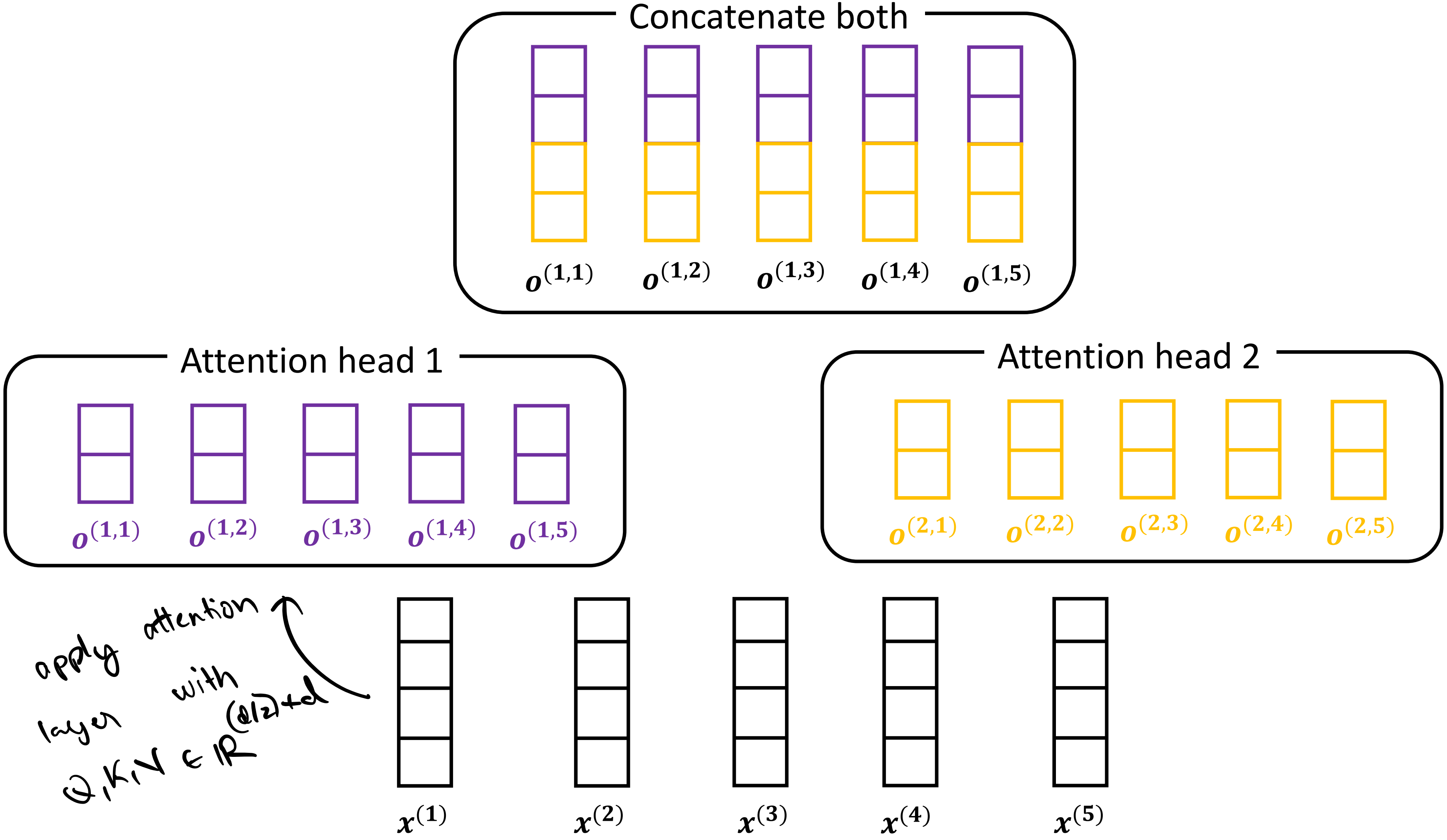
2. Compute pairwise similarities between keys and queries; normalize with softmax

$$
\begin{aligned}
\alpha_{ij} &= \boldsymbol{q}_i^\top \boldsymbol{k}_j \\
w_{ij} &= \frac{\exp(\alpha_{ij})}{\sum_{j'} \exp(\alpha_{ij'})}
\end{aligned}
$$

3. Compute output for each word as weighted sum of values

$$
\boldsymbol{o}_i = \sum_j w_{ij} \boldsymbol{v}_j
$$

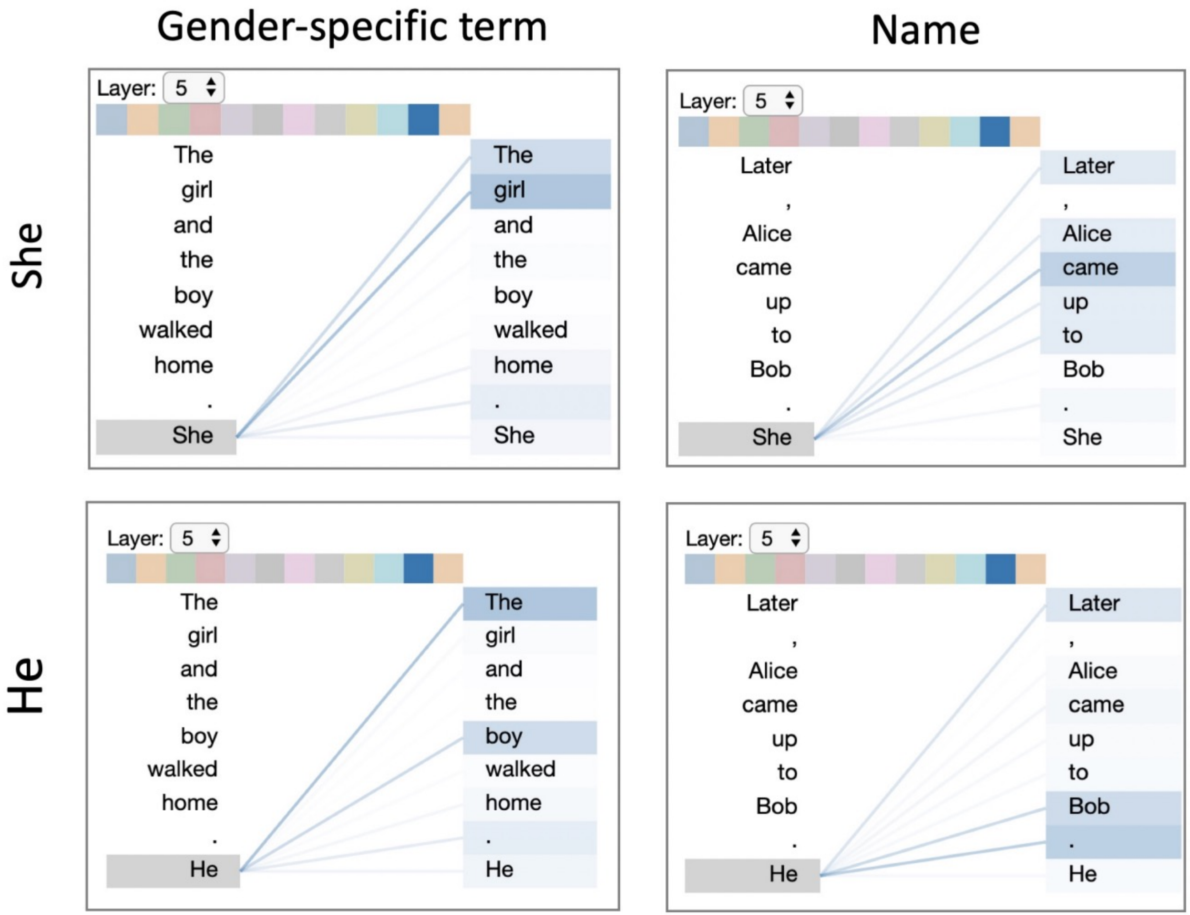
Multi headed self-attention



Multi headed self-attention

- Input: List of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$, each of size d
- Output: List of vectors $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_T$, each of size d
- Formula: For each head i:
  - Compute self attention output using $\boldsymbol{Q}_i$, $\boldsymbol{K}_i$, $\boldsymbol{V}_i$
  - Finally, concatenate results for all heads
- Parameters:
  - For each head i, parameter matrices $\boldsymbol{Q}_i$, $\boldsymbol{K}_i$, $\boldsymbol{V}_i$ of size $d_{\text{attn}} \times d$
  - # of heads n is hyperparameter, $d_{\text{attn}} = d/n$

What do attention heads learn?



A Multiscale Visualization of Attention in the Transformer Model, Vig 2019