STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

CS 584 Natural Language Processing

Language Modeling

Ping Wang
Department of Computer Science
Stevens Institute of Technology

Outlines

- Language Modeling (LM)
- Development of LM
- N-gram LM
- Neural LM

Language modeling

- LANGUAGE is a prominent ability in human beings to express and communicate, which develops in early childhood and evolves over a lifetime.
- Machines, however, cannot naturally grasp the abilities of understanding and communicating in the form of human language, unless equipped with powerful artificial intelligence (AI) algorithms.
- It has been a longstanding research challenge to achieve this goal, to enable machines to read, write, and communicate like humans.

A Survey of Large Language Models: https://arxiv.org/pdf/2303.18223.pdf

Language modeling

- In general, language modeling is the task of predicting what word comes next.
- For example, “the students opened their [blank]” (books? laptops? minds? exams?)

- Formally, given a sequence of words $x^{(1)},x^{(2)},...,x^{(t)}$, compute the probability distribution of the next word $x^{(t+1)}$:

$$p(x^{(t+1)} = w_j | x^{(t)},...,x^{(1)})$$

where w_j is a word in the vocabulary $V = \{w_1, ..., w_{|V|}\}$.

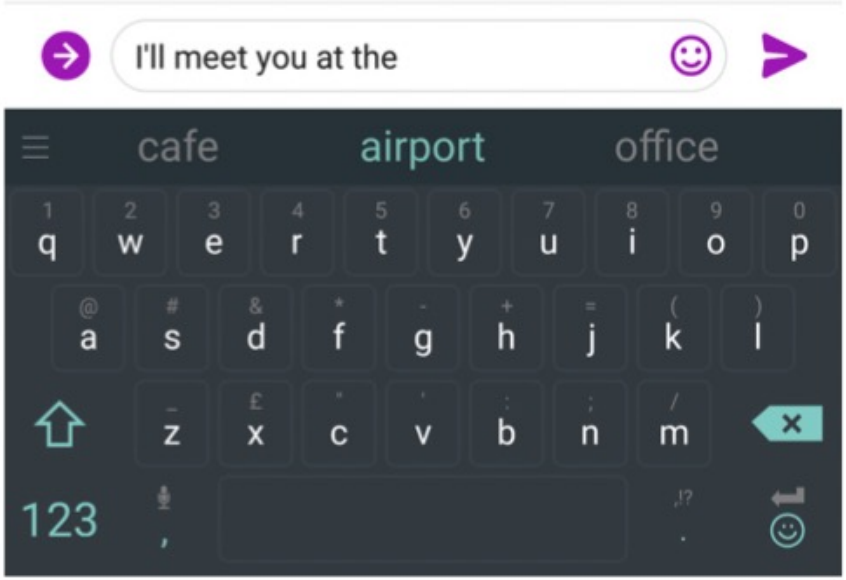
- A system that does this is called a language model.

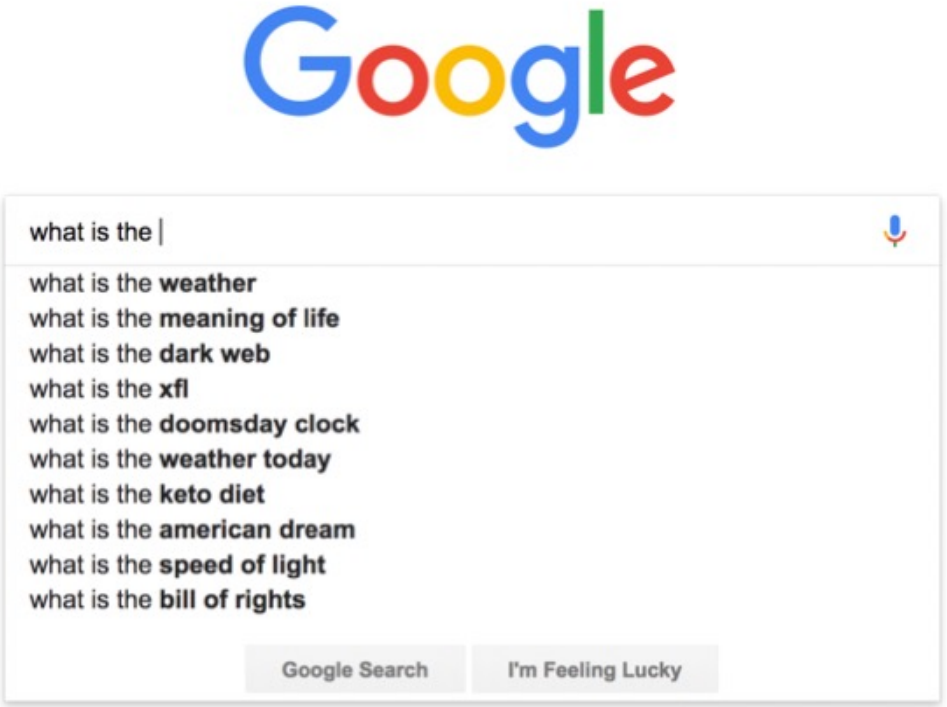
Why should we care about LM?

- LM is a benchmark task that helps us measure our progress on understanding language.

- LM 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.

We use Language Models every day!





Development of LM 1: statistical LM

- Developed based on statistical learning methods in 1990s.
- N-gram LMs, such as bigram and trigram.
- Limitations:
  - Difficult to accurately estimate high-order language models since an exponential number of transition probabilities need to be estimated.
  - Specially designed smoothing strategies are used to alleviate the data sparsity problem.

Development of LM 2: Neural LM

- Characterize the probability of word sequences by neural networks, e.g. RNN

- Introduced the concept of distributed representation of words, such as word2vec, which were demonstrated to be very effective across a variety of NLP tasks.

- These studies have initiated the use of language models for representation learning, having an important impact on the field of NLP.

Development of LM 3: Pre-trained LM (PLM)

- **Pre-training first and then fine-tuning** on specific downstream task, allows to capture context-aware word representation.

- BERT: pretraining with specially designed tasks on large-scale unlabeled data, which is very effective to serve a general-purpose semantic features and improves the performance of NLP tasks.

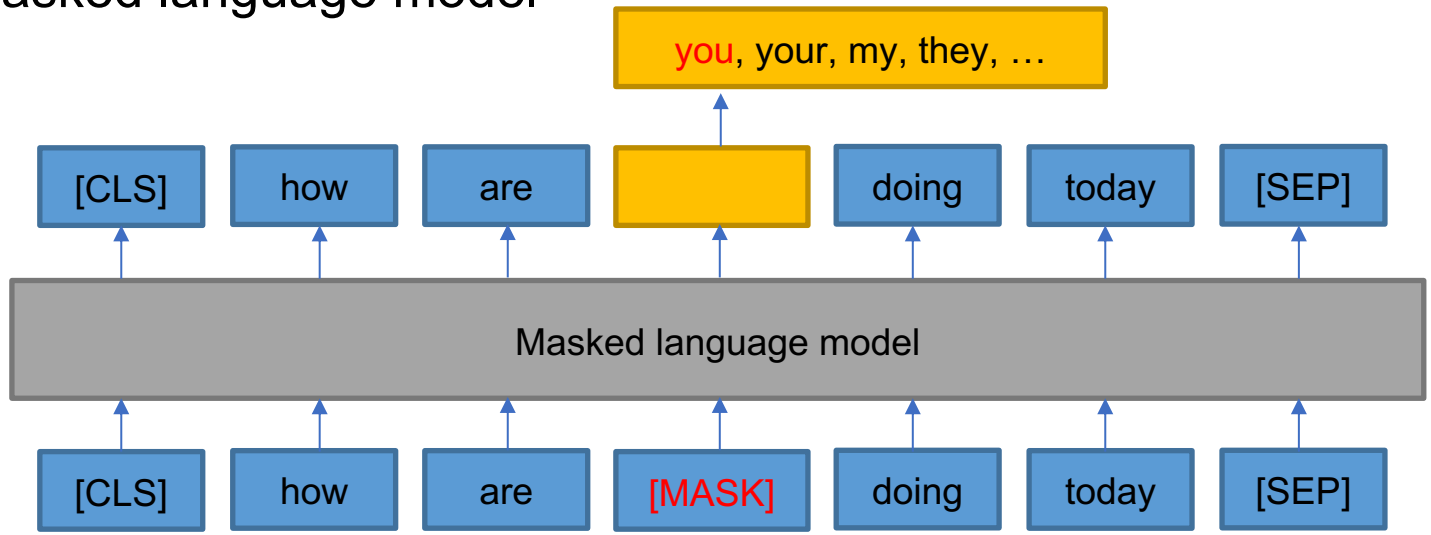
- Inspired many follow-up work which sets the "pre-training and fine-tuning" learning paradigm: with different architectures or improved pre-training strategies.

Pre-training BERT

- Next sentence prediction

Sentence 1	Sentence 2	Next sentence?
I am going outside.	I will be back after 6.	Yes
I am going outside.	You are welcome.	No

- Masked language model



Devlin J, et al., Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

Development of LM 4: Large LM (LLM)

- Scaling PLM in model size often leads to an emergent model capacity.
- For example, GPT-3 can solve few-shot tasks through in-context learning, while GPT-2 cannot do well.
- The term LLM is coined for the large-sized PLMs.
- An application of LLMs is ChatGPT that adapts the LLMs from the GPT series for dialogue.
- Followed with a sharp release of LLMs: Bard, Claude 2, LLaMA, Falcon, etc.

Statistical Language Modeling

N-gram language models

- 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.
  - Uni-grams: “the”, “students”, “opened”, “their”
  - Bi-grams: “the students”, “students opened”, “opened their”
  - Tri-grams: “the students opened”, “students opened their”
  - 4-grams: “the students opened their”

- Idea: Collect statistics about how frequent different n-grams are, and use these to predict next word.

N-gram language models

- We make a simplifying assumption: the next word only depends on the preceding n-1 words
- Thus, the probability can be written as:

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

Prob. of a n-gram

Prob. of a (n-1)-gram

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

N-gram language models

- Statistical approximation: By counting them in large text corpus!

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

- Assuming a 4-gram language model:

$$p(w_j|\text{“students opened their”}) = \frac{\text{count(students opened their } w_j)}{\text{count(students opened their)}}$$

For example, suppose that in this 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

How to compute joint probability

- How to compute this joint probability:
  - P(its, water, is, so, transparent, that)

- Intuition: let's rely on the Chain Rule of Probability

Reminder: the chain rule

- Recall the definition of conditional probability
- More variables:
  - $P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$
- The Chain Rule in General
  - $P(x_1,x_2,x_3,\ldots,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1,\ldots,x_{n-1})$

The chain rule for joint probability

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

P("its water is so transparent") =

P(its) × P(water|its) × P(is|its water)

× P(so|its water is) × P(transparent|its water is so)

How to estimate these probabilities

Can we just simply count and divide?

$$P(\text{“the”} \mid \text{“its water is so transparent that”}) = \frac{\text{count (its water is so transparent that the)}}{\text{count (its water is so transparent that)}}$$

No! Too many possible sentences!

We’ll never see enough data for estimating these.

Markov Assumption

- The probability of moving to the next state depends only on the present state and not on the previous states.
- Simplifying assumption:

P(“the” | “its water is so transparent that”) = P(“the”|”that”)

- Or maybe:

P(“the” | “its water is so transparent that”)

= P(“the”|“transparent that”)

Simplest case: unigram model

$$P(w_1 ... w_n) \approx \prod_i P(w_i)$$

- Some automatically generated sentences from a unigram model:
  - *fifth, an, of, futures, the, an, incorporated, a, a, the, inflation, most, dollars, quarter, in, is, mass*
  - *thrift, did, eighty, said, hard, 'm, july, bullish'*
  - *that, or, limited, the*

Bigram model

- Condition on the previous word:

$$P(w_1, ...w_n) \approx \prod_i P(w_i|w_{i-1})$$

- Some automatically generated sentences from a bigram model:
  - *texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen*
  - *outside, new, car, parking, lot, of, the, agreement, reached*
  - *this, would, be, a, record, november*

N-gram model

- We can extend to trigrams, 4-grams, 5-grams
- In general, this is an insufficient model of language because language has long-distance dependencies:
  - "*The computer which I had just put into the machine room on the fifth floor crashed*"
- But we can often get away with N-gram models

Estimating probabilities

- The maximum likelihood estimate (MLE)
  - of some parameter of a model M from a training set T
  - maximizes the likelihood of the training set T given the model M

- Suppose the word “bagel” occurs 400 times in a corpus of a million words
- What is the probability that a random word from some other text will be “bagel”?
- MLE estimate is 400/1,000,000 = .0004

- This may be a bad estimate for some other corpus
  - But it is the estimate that makes it most likely that “bagel” will occur 400 times in a million word corpus.

Estimating bigram probabilities

- The Maximum Likelihood Estimate
- To compute a particular bigram probability of a word w_i given a previous word w_{i-1}:

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)}{\sum_w c(w_{i-1}w)}$$

- It is equivalent to the following calculation since the sum of all bigram counts that start with a given word w_{i-1} must be equal to the unigram count for that word w_{i-1}.

$$P(w_i|w_{i-1}) = \frac{c(w_{i-1},w_i)}{c(w_{i-1})}$$

An example

- <s> I am Sam </s>
- <s> Sam I am </s>
- <s> I do not like green eggs and ham </s>

$$P(w_i|w_{i-1}) = \frac{c(w_{i-1},w_i)}{c(w_{i-1})}$$

P(I | <s>) = ⅔ = 0.67
P(Sam | <s>) = ⅓ = 0.33
P(am | I) = ⅔=0.67
P(</s> | Sam)= ½=0.5
P(Sam | am) = ½ =0.5
P(do | I) = ⅓=0.33

Practical Issues

- For computing efficiency, we do everything in log space
  - Avoid underflow: the probabilities become so close to zero that they cannot be accurately represented.
  - Also adding is faster than multiplying

$$\log (p_1 \times p_2 \times p_3 \times p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4$$

Language Modeling Toolkits

- SRILM - The SRI Language Modeling Toolkit
  - http://www.speech.sri.com/projects/srilm/

- The CMU-Cambridge Statistical Language Modeling Toolkit v2
  - http://www.cs.cmu.edu/~dorcas/toolkit_documentation.html

Google N-gram release (2006)

- All Our N-gram are belong to you

  "We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times."

- Google N-gram viewer: A tool that displays a graph showing how those phrases have occurred in a corpus of books.

Sparsity problems in N-grams

Problem: what if "its water is so transparent that w" never occurred in data? Then w has probability 0!

(Partial) solution: add small value to the count for every w in the vocabulary. This is called **smoothing**.

$$P(w \mid \text{"its water is so transparent that"}) = \frac{\text{count (its water is so transparent that w)}}{\text{count (its water is so transparent that)}}$$

Problem: what if "its water is so transparent that" never occurred in data? Then we cannot calculate probability for any w.

(Partial) solution: Just condition on "so transparent that" instead. This is called **backoff**.

Note: increasing n makes sparsity problems worse. Typically, we cannot have n bigger than 5.

Storage problems with n-gram

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

$$P(w \mid \text{“its water is so transparent that”}) = \frac{\text{count (its water is so transparent that w)}}{\text{count (its water is so transparent that)}}$$

Note: increasing n or increasing corpus makes model size bigger!

Statistical Language Modeling

Goal: assign a probability to a sentence

- Machine Translation:
  - P(high winds tonite), P(large winds tonite)
- Spell Correction
  - The office is about fifteen minuets from my house
    - P(about fifteen minutes from), P(about fifteen minuets from)
- Speech Recognition
  - P(I saw a van), P(eyes awe of an)
- + Summarization, question-answering, etc., etc.!

Generating text with an n-gram LM

You can 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.

Extrinsic evaluation of N-gram models

- Does our language model prefer good sentences to bad ones?
  - Assign higher probability to “real” or “frequently observed” sentences than “ungrammatical” or “rarely observed” sentences?

- Best evaluation for comparing models A and B: extrinsic evaluation
  - Put each model in a downstream task
    - spelling corrector, speech recognizer, machine translation
  - Run the task, get an accuracy for A and for B
    - How many misspelled words corrected properly
    - How many words translated correctly
  - Compare accuracy for A and B

  - Limitation: Time-consuming; can take days or weeks

Intrinsic evaluation of N-gram models

- Sometimes use intrinsic evaluation: perplexity
  - Bad approximation
    - Unless the test data looks just like the training data
    - So generally, only useful in pilot experiments
  - But it is helpful to think about.

Perplexity

- The best language model is one that best predicts an unseen test set
  - Gives the highest P(sentence)

$$P(W) = P(w_1 w_2 ... w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_N)}}$$

Chain rule:

$$= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 ... w_{i-1})}}$$

For bigrams:

$$P(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}}$$

- Perplexity is the inverse probability of the test set, normalized by the number of words
  - minimizing perplexity is the same as maximizing probability

Perplexity as a branching factor

- Let's suppose a sentence '0,1,2,3,4,5,6,7,8,9' consisting of random digits. How hard is the task of recognizing digits?
- Or what is the perplexity of this sentence according to a model that assign P = 1/10 to each digit?

$$
\begin{aligned}
P(W) &= P(w_1 w_2 ... w_N)^{-\frac{1}{N}} \\
&= \left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}} \\
&= \frac{1}{10}^{-1} \\
&= 10
\end{aligned}
$$

- Perplexity 10 since each digit is equally likely.

Low perplexity = better model

- Training 38 millions words, test 1.5 million words (WSJ)

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Overfitting

- N-grams only work well for word prediction if the test corpus looks like the training corpus
  - In real life, it often doesn't
  - We need to train robust models that generalize!

  - One kind of generalization: Zeros!
    - Things that don't ever occur in the training set
    - But occur in the test set

Zeros: Zero probability bigrams

- Training set:

... denied the allegations
... denied the reports
... denied the claims
... denied the request

- Test set:

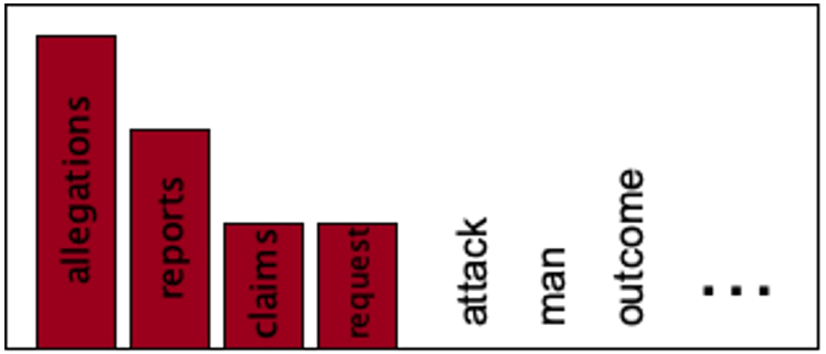
... denied the offer
... denied the loan

P("offer" | denied the) = 0
Perplexity=∞

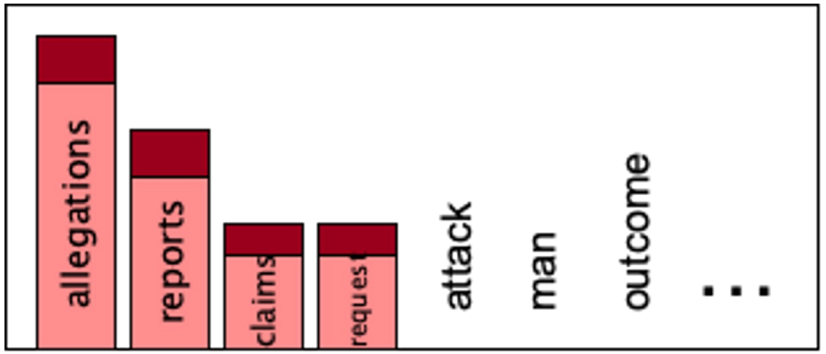
- Bigram with zero probability: means that we will assign 0 probability to the test set
- Hence, we cannot compute perplexity (divided by 0!)
- Solution: Laplace smoothing

The intuition of smoothing

- When we have sparse statistics
  - P(w | “denied the”)
  - 3 allegations
  - 2 reports
  - 1 claims
  - 1 request
  - 7 total



- Steal probability mass to generalize better
  - P(w | “denied the”)
  - 2.5 allegations
  - 1.5 reports
  - 0.5 claims
  - 0.5 request
  - 2 others
  - 7 total



Laplace smoothing

- Also called add-one estimation
- Pretend we saw each word one more time than we did
- Just add one to all the counts

- MLE estimate:

$$P_{\text{MLE}}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

- Add-1 estimate:

$$P_{\text{Add}-1}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

Berkeley Restaurant Corpus: Raw bigram counts vs. Laplace smoothed bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Add-1 estimation is a blunt instrument

- Primary limitations of Add-1 estimation:
  - **Overestimation of probabilities**, especially for less common n-grams
  - **Loss of discrimination**: It tends to smooth out differences between n-grams, leading to a loss of discrimination between more and less likely sequences.
  - **Sensitivity to vocabulary size**: The level of smoothing in "add-1" depends on the size of the vocabulary and the length of the n-gram. In cases where the vocabulary is large or the n-gram length is high, the added "pseudocount" of 1 may have a smaller impact on the estimated probabilities.
- So add-1 isn't used for N-grams
  - But add-1 is used to smooth other NLP models in domains where the number of zeros isn't so huge

Backoff and interpolation

- Backoff: Sometimes it helps to use less context
  - Condition on less context for contexts you haven't learned much about
  - Use trigram if you have sufficient evidence
  - Otherwise bigram, otherwise unigram

- Interpolation: mixing the probability estimates from all n-gram estimators; weighting and combining the unigram, bigram, and trigram counts works better.

Linear Interpolation

- Simple interpolation

$$\begin{aligned}\hat{P}(w_n|w_{n-1}w_{n-2}) &= \lambda_1 P(w_n|w_{n-1}w_{n-2}) \\ &+ \lambda_2 P(w_n|w_{n-1}) \\ &+ \lambda_3 P(w_n) \\ &\sum_i \lambda_i = 1\end{aligned}$$

- Lambdas conditional on context:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1(w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) \\ &+ \lambda_2(w_{n-2}^{n-1})P(w_n|w_{n-1}) \\ &+ \lambda_3(w_{n-2}^{n-1})P(w_n)\end{aligned}$$

Now lambdas are dependent on what the previous two words were.

How to set the lambdas?

- Use a held-out corpus

- Choose lambdas to maximize the probability of held-out data:
  - fix the N-gram probabilities (on the training data)
  - then search for lambdas that give largest probability to held-out set.

Unknown words:

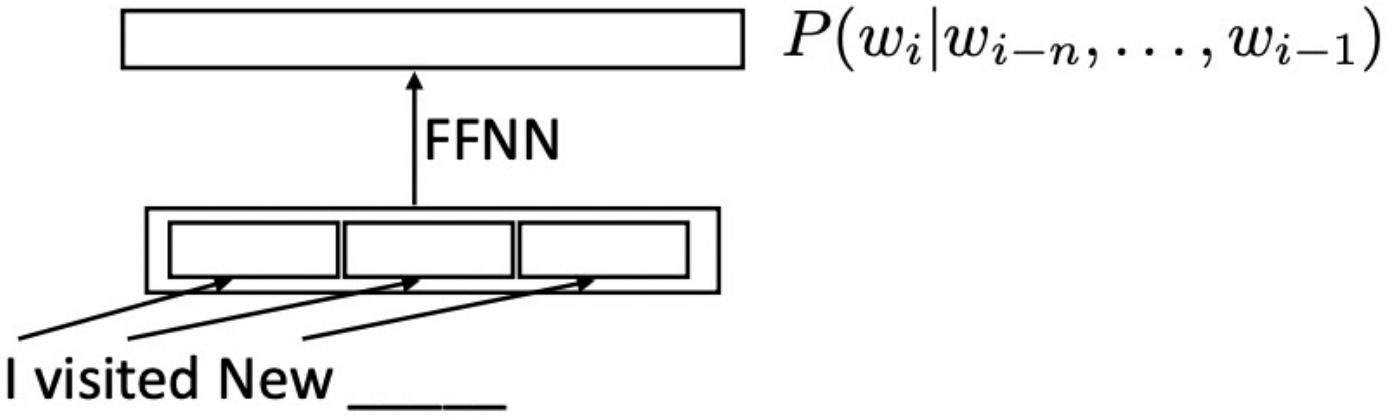
open versus closed vocabulary task

- If we know all the words in advanced
  - Vocabulary V is fixed
  - Closed vocabulary task
- Often we don't know this
  - Out Of Vocabulary = OOV words
  - Open vocabulary task
- Instead: create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - Create a fixed lexicon L of size V
    - At text normalization phase, any training word not in L changed to <UNK>
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training

Neural Language Modeling

How to build a neural language model?

- Early work: feedforward neural networks looking at context



- Disadvantages: Slow to train over lots of data; considering limited context; …

Bengio et al. (2003)

How to build a neural language model?

- How about a window-based neural model?
- Considered a fixed-sized window of words as input
- Predict the next word in the sequence

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

discard

fixed window

A fixed-window neural language model

- output distribution

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

- hidden layer

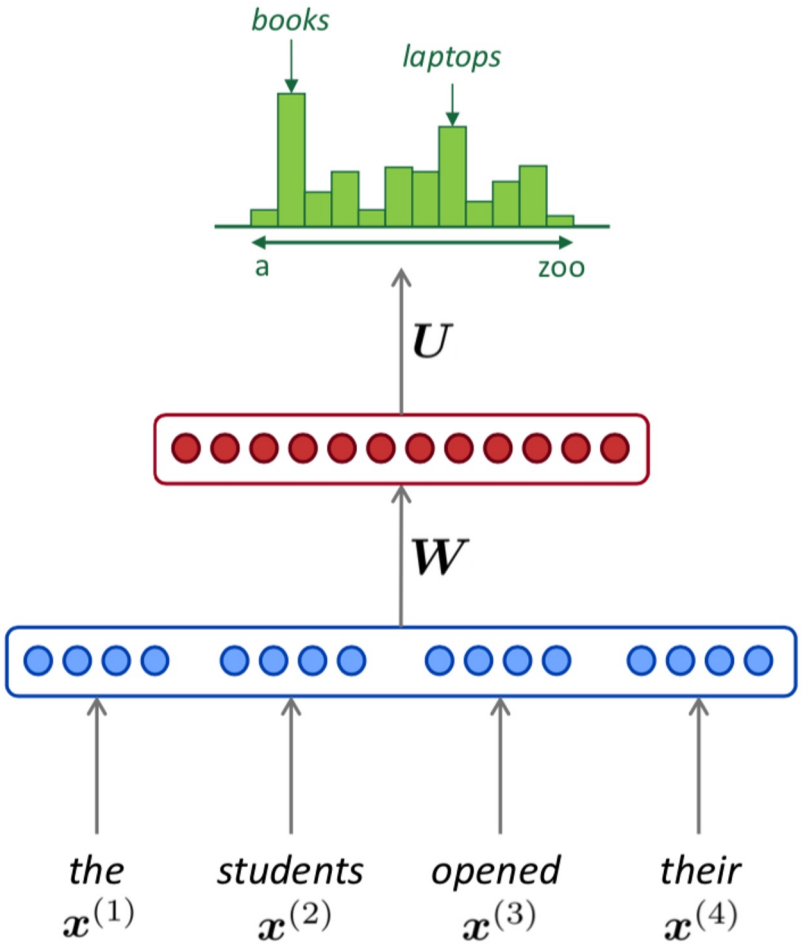
$$h = f(We + b_1)$$

- concatenated word embeddings

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

- words/one-hot vectors:

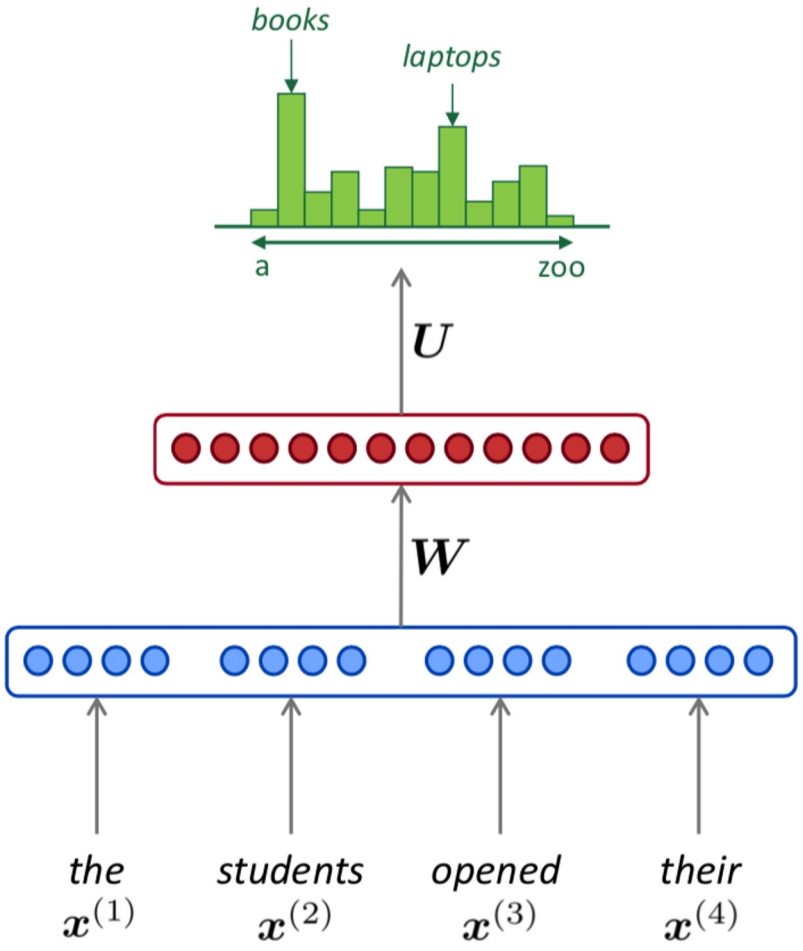
$$x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$$



A fixed-window neural language model
Motivate RNN

- Improvements over n-gram LM:
  - No sparsity problem
  - Model size is smaller
- Remaining problems:
  - Fixed window is too small
  - Enlarging window enlarges W
  - Window can never be large enough!
  - Each x(i) uses different rows of W. We don't share weights across window.

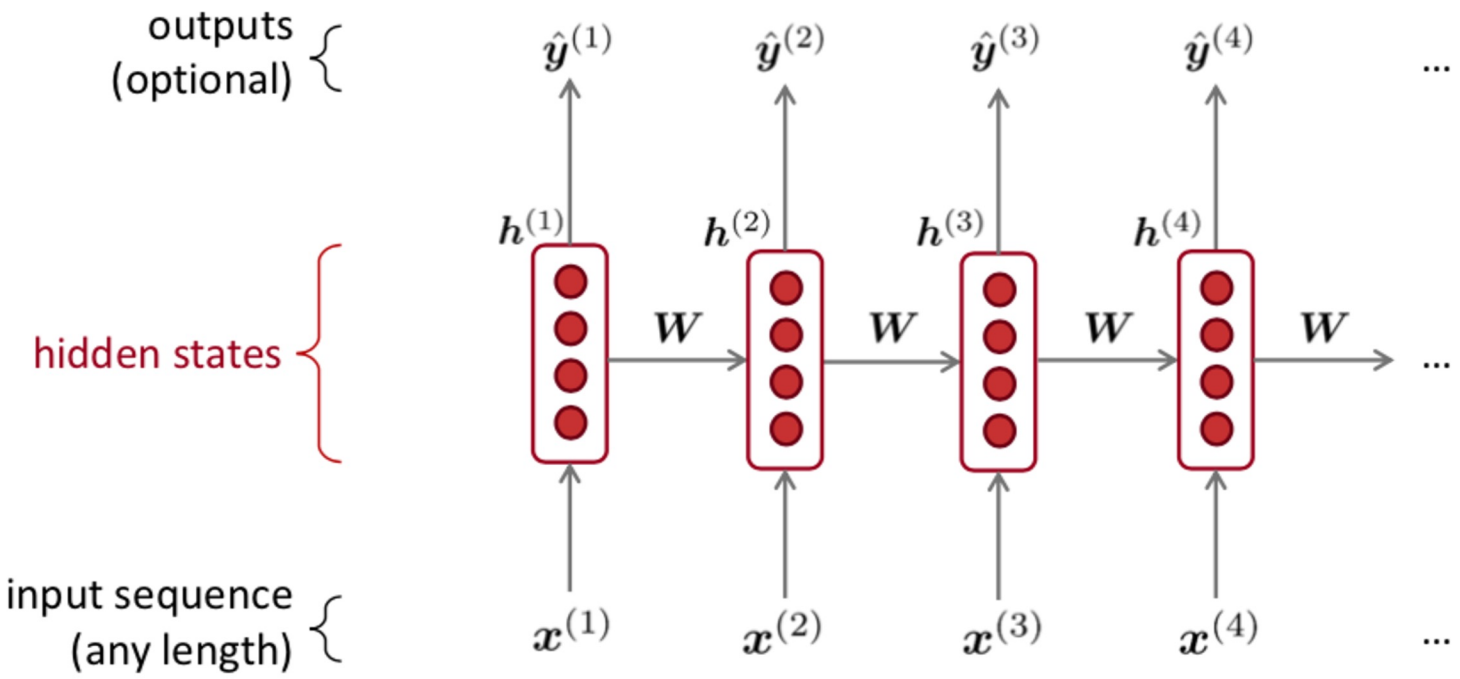
We need a neural architecture that can process any length input



Recurrent Neural Networks (RNN)

A family of neural architectures

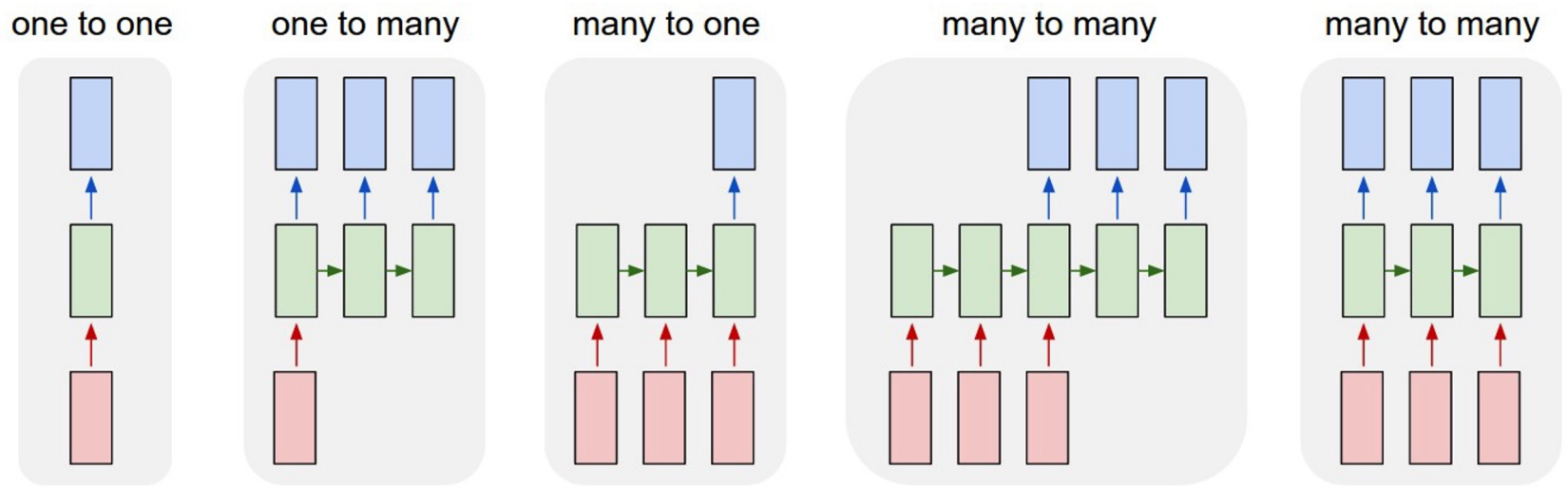
- Core idea: apply the same weights W repeatedly



Recurrent Neural Networks (RNN)

A family of neural architectures

- Core idea: apply the same weights W repeatedly



A RNN language model

- output distribution

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

- hidden layer

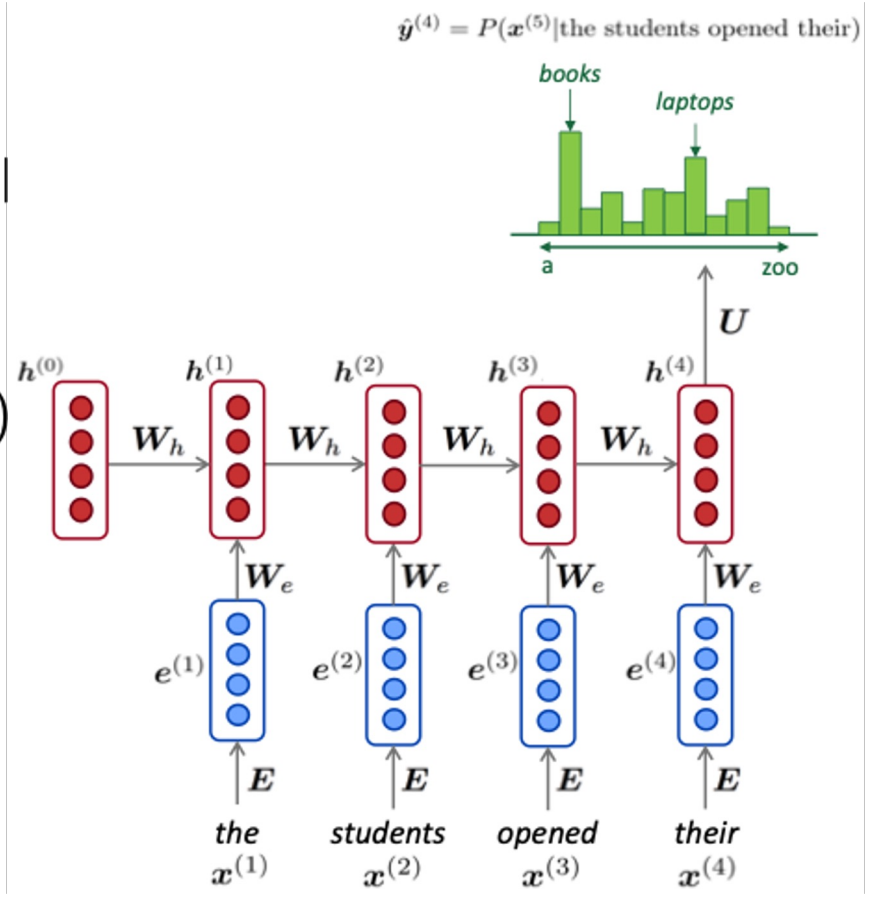
$$h^{(t)} = \tanh(W_h h^{(t-1)} + W_e e^{(t)} + b_1)$$

- word embeddings:

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

- words/one-hot vectors:

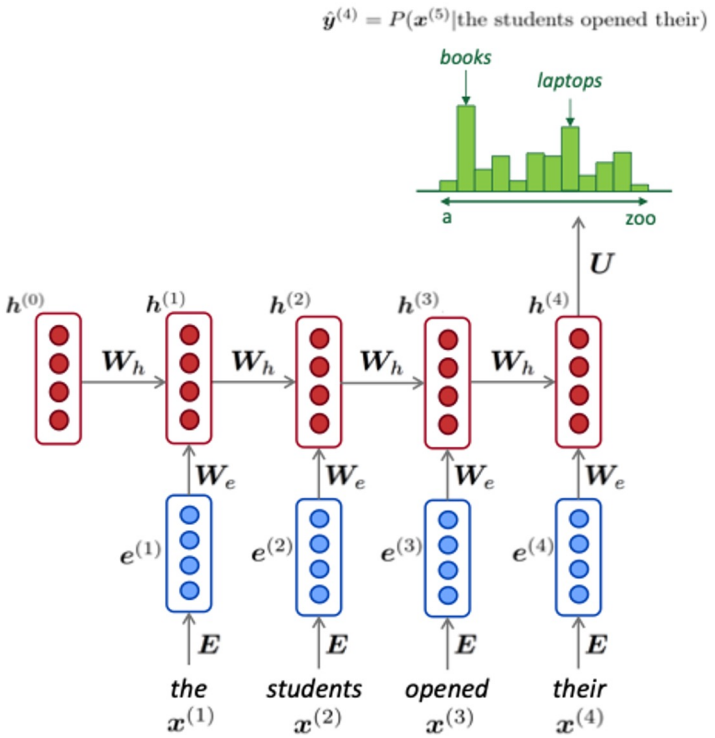
$$x^{(t)} \in R^{|V|}$$



Note: this input sequence could be much longer

A RNN language model

- Advantages:
  - Can process any length input
  - Model size doesn't increase for longer input
  - Computation for step t can use information from many steps back (in theory)
  - Weights are shared across time stamps: representation are shared
- Disadvantages:
  - Recurrent computation is slow
  - In practice, difficult to access information from multiple steps back



Training A RNN language model

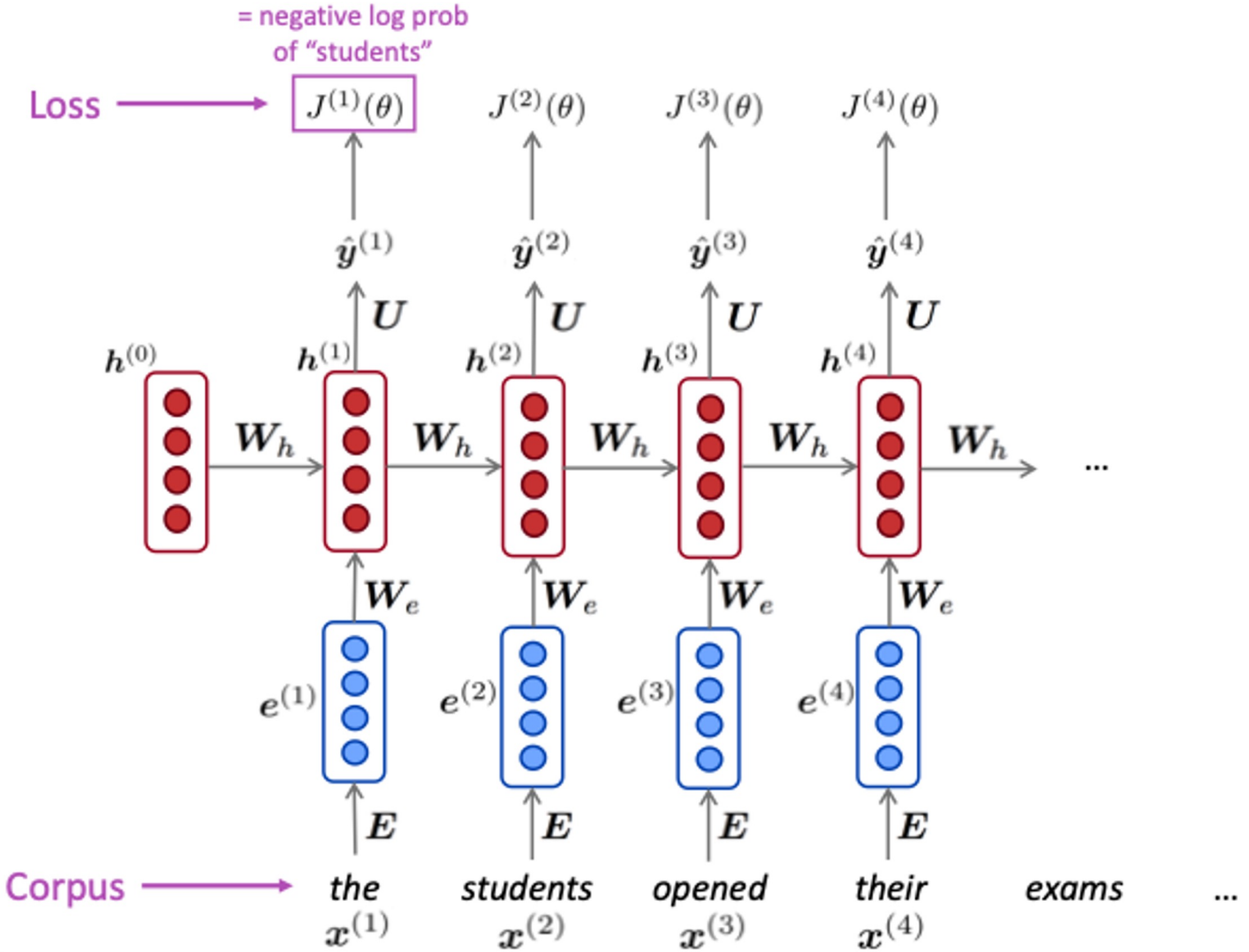
- Get a big corpus of text which is a sequence of words $x^{(1)}, x^{(2)}, ..., x^{(t)}$
- Feed into RNN-LM; compute output distribution $\hat{y}^{(t)}$ for every step t.
- Loss function on step t is usual cross-entropy between out predicted probability $\hat{y}^{(t)}$ and the true next word $y^{(t)} = x^{(t+1)}$:

$$J^{(t)}(\theta) = \text{CE}(y^{(t)}, \hat{y}^{(t)}) = -\sum_{j=1}^{|V|} y_j^{(t)} \log \hat{y}_j^{(t)}$$

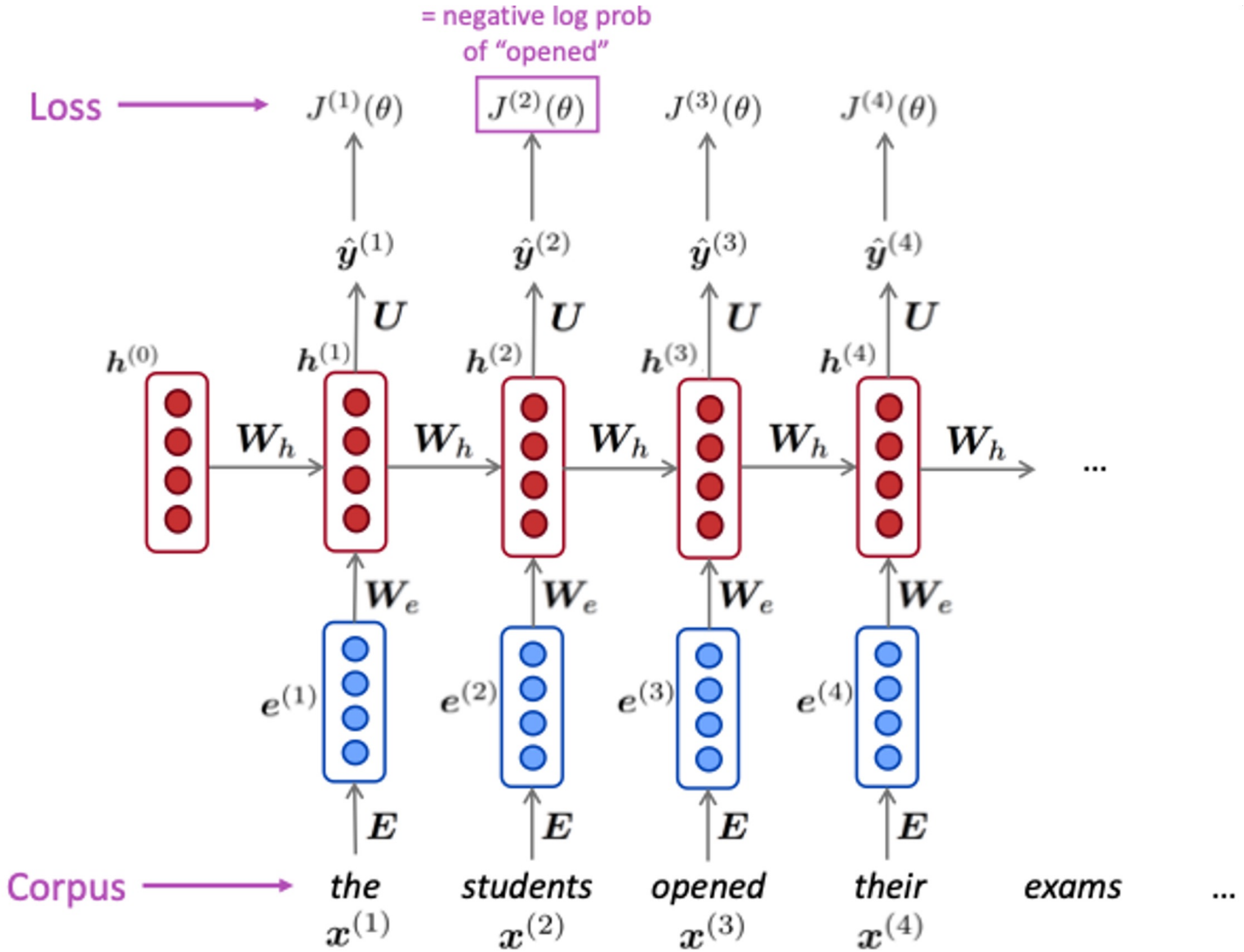
- Average this to get overall loss for entire training steps

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

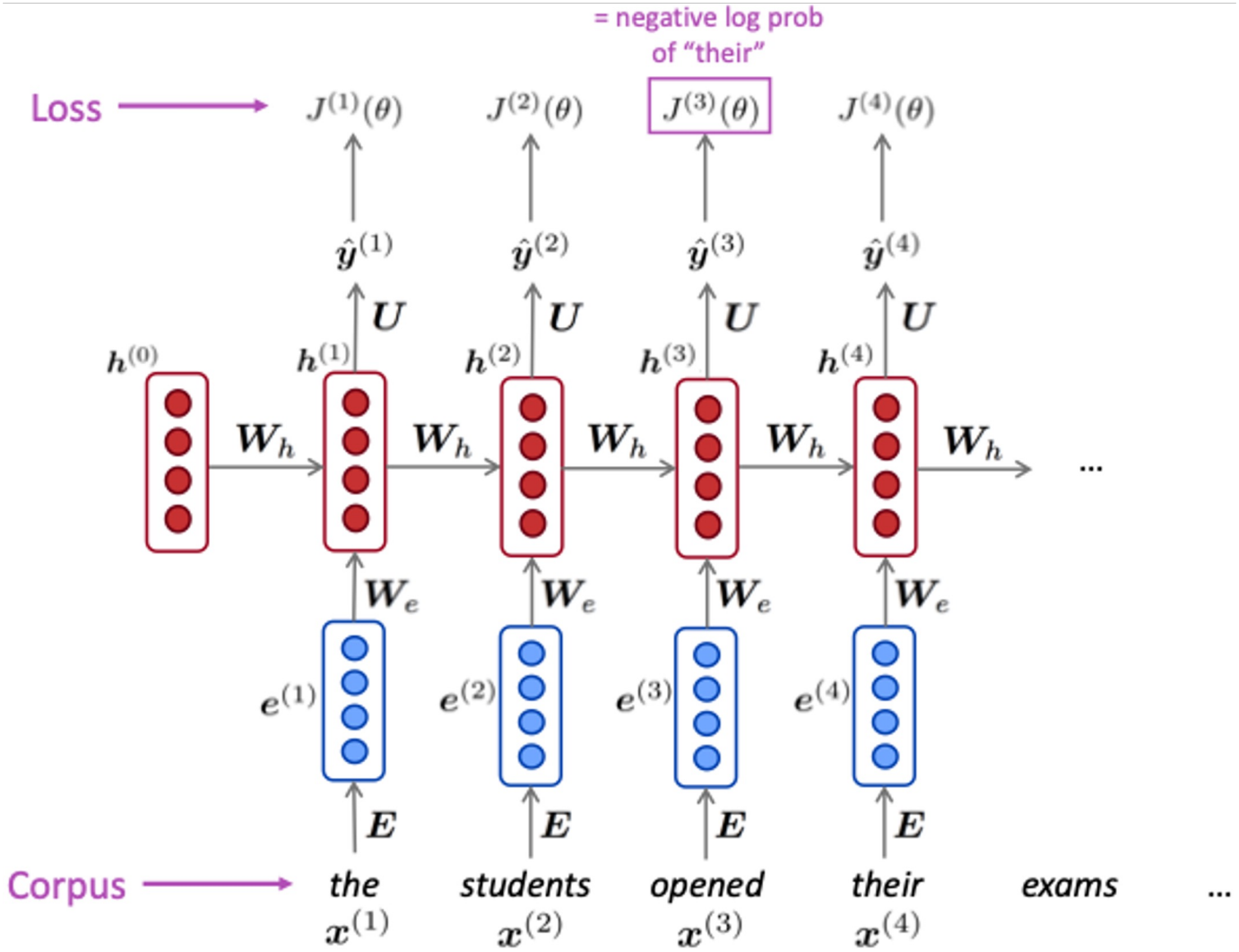
Training A RNN language model



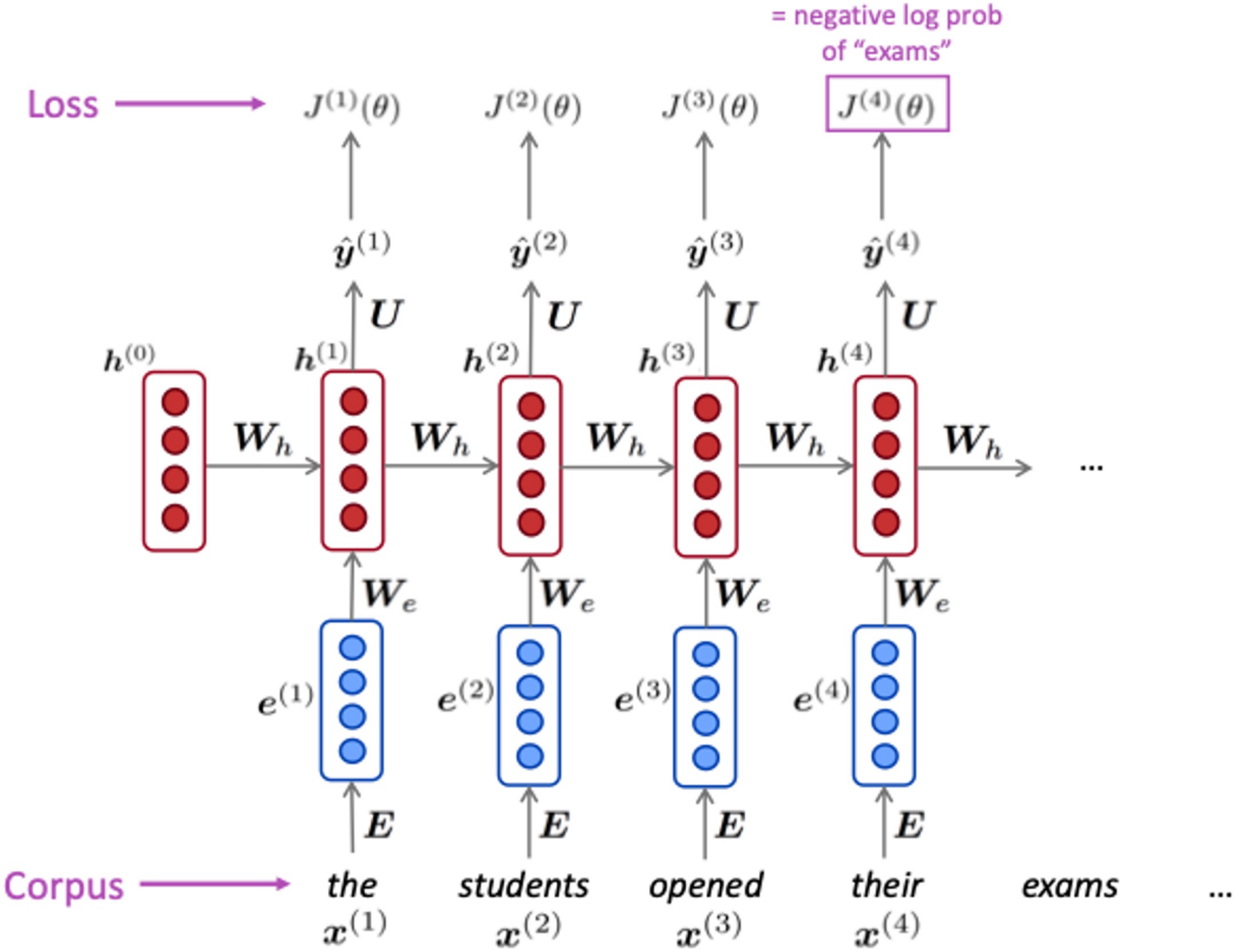
Training A RNN language model



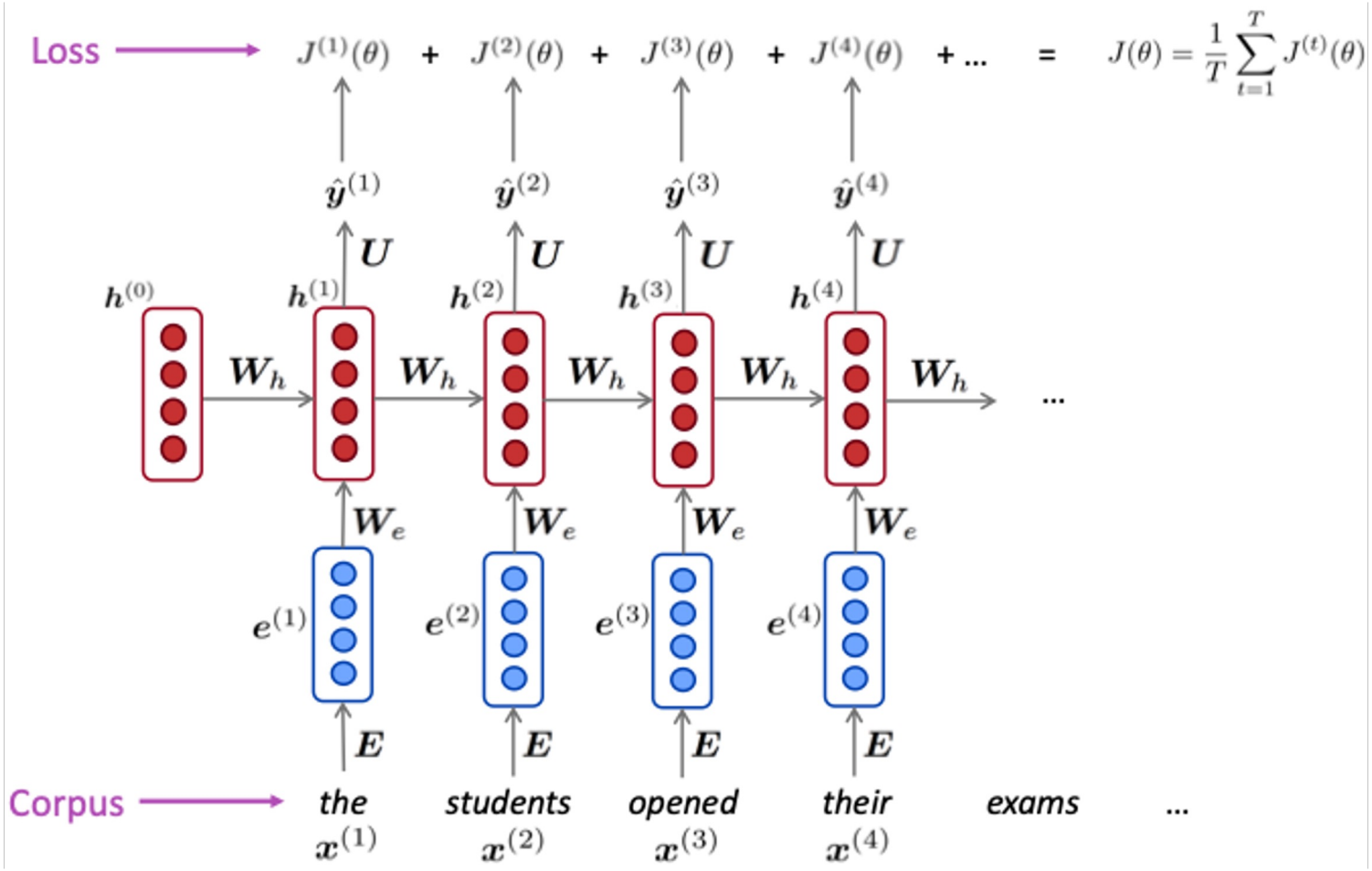
Training A RNN language model



Training A RNN language model



Training A RNN language model



Training A RNN language model

- However, computing loss and gradients across entire corpus is too expensive!
- Recall: Stochastic Gradient Descent (SGD) allows us to compute loss and gradient for small set of data and update
- In practice, consider x(1),x(2),...,x(t) as a sentence:

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

- Compute loss J(θ) for a sentence, compute gradients and update weights. Repeat.

Backpropagation for RNNs

- What is the derivative of $J^{(t)}(\theta)$ w.r.t the repeated weight matrix W_h?
- Answer:

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

- Backpropagate over time steps i=t,...,0, summing gradients as you go. This algorithm is called "backpropagation through time"

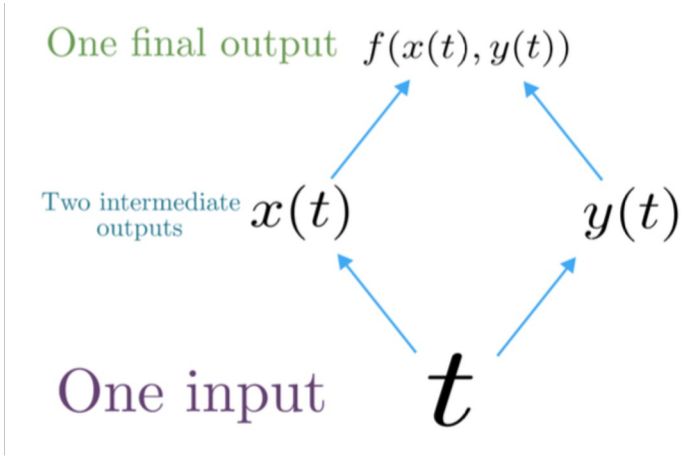
"The gradient w.r.t a repeated weight is the sum of the gradient w.r.t each time it appears"

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:

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



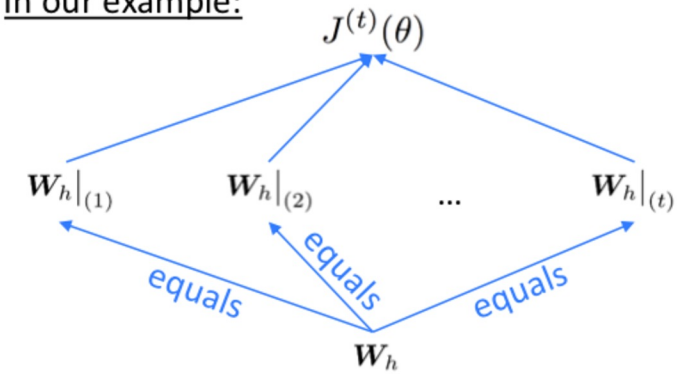
https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/differentiating-vector-valued-functions/a/multivariable-chain-rule-simple-version

Backpropagation for RNNs: proof sketch

- 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:

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

In our example:

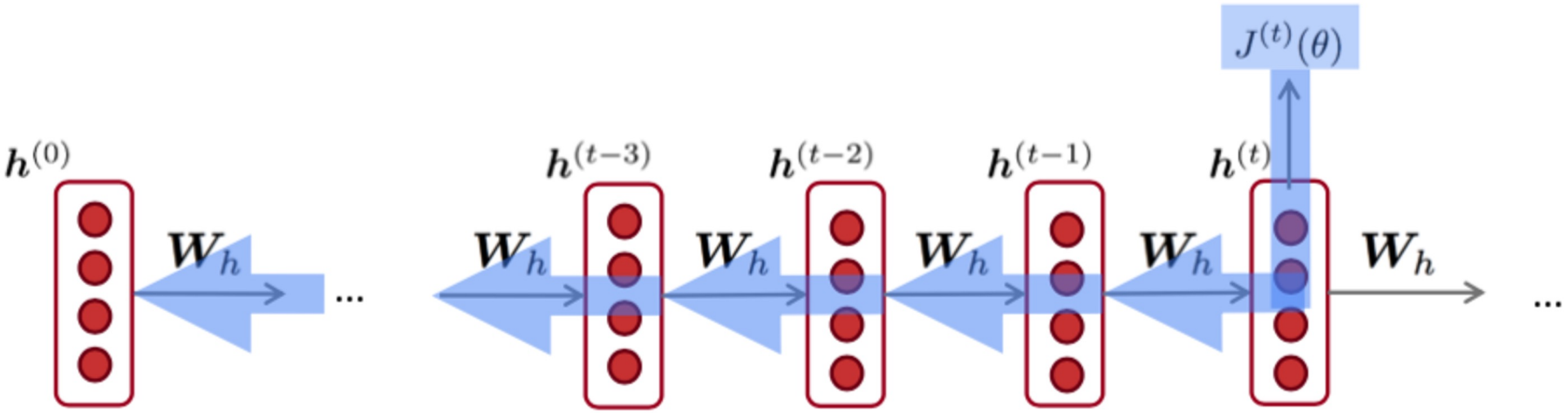


Apply the multivariable chain rule:

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

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

Backpropagation for RNNs



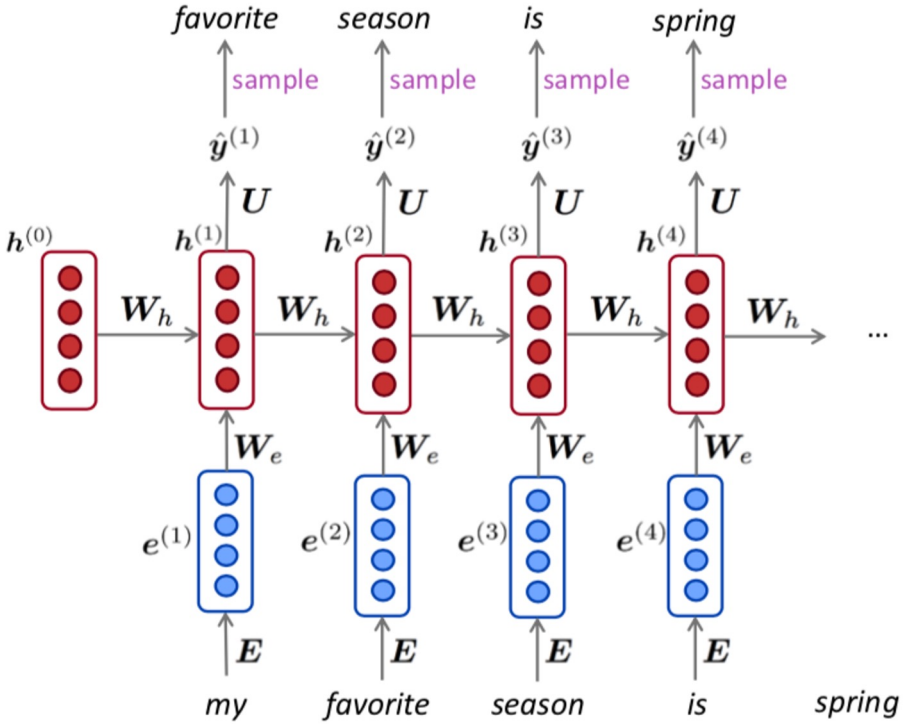
$$\frac{\partial J^{(t)}}{\partial \mathbf{W}_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \mathbf{W}_h}\Big|_{(i)}$$

Q: how do we calculate this?

Answer: backpropagate over time steps i = t, … ,0, summing gradients as you go.
This is called "backpropagation through time"

Generating text with a RNN language model

- Just like a n-gram LM, we can use a RNN LM to generate text by repeated sampling. Sampled output is next step's input.



Evaluating language models

- The standard evaluation metric for language models is perplexity

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

Normalized by number of words

Inverse probability of corpus, according to LM

- This is equal to the exponential of the cross-entropy loss J(θ)

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

Lower perplexity is better

RNNs have greatly improved perplexity

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

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

A note on terminology

- RNN described in this lecture = “vanilla RNN”

- Next lecture: learn about other RNN flavors like GRU  and LSTM  and multi-layer RNNs 

- By the end of the course, you will understand phrases like “stacked bidirectional LSTM with residual connections and self-attention”

Summary

- Language Model: A system that predicts the next word
- Statistical LM: developed based on statistical learning
- 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
- RNNs ⇏ Language Model
- We've shown that RNNs are a great way to build a LM
- But RNNs are useful for much more!

Readings

- CH 9, 13 SLP; CH 14 NNLP
- A Survey of Large Language Models: https://arxiv.org/pdf/2303.18223.pdf
- N-gram language models: https://web.stanford.edu/~jurafsky/slp3/3.pdf
- Afshine Amidi and Shervine Amidi. Recurrent Neural Networks cheatsheet. 2019
- Implementing RNN from scratch: https://github.com/pangolulu/rnn-from-scratch

Acknowledgements

Slides adapted from Dr. Dan Jurafsky's Introduction to Natural Language Processing at Stanford.

STEVENS

INSTITUTE of TECHNOLOGY

THE INNOVATION UNIVERSITY®

stevens.edu

Thank You