Natural Language Processing

CS 3216/UG, AI 5203/PG

Week-7

Recurrent Neural Networks

Recap

- NLP
- Applications
- Regular expressions
- Tokenization
- Stemming
    - Porter Stemmer
- Lemmatization
- Normalization
- Stopwords
- Bag-of-Words
- TF-IDF
- NER
- POS tagging
- Semantics, Distributional semantics, Word2vec
- Language models
- Neural Networks and Neural language modeling

Last Lecture

- Neural Networks
- Feed- forward Neural Networks
- Neural language models

Sequential Data

Sometimes the sequence of data matters

- Text generation
- Stock price prediction
- Machine translation
- Speech recognition

Image Source: https://www.analyticsvidhya.com/blog/2019/07/openai-gpt2-text-generator-python/

Sentence

The clouds are in the .... ?

Sentence

The clouds are in the .... ?

SKY

Sequence data

- The clouds are in the .... ?
  SKY
- Simple solution: N-grams?

Sequence data

- The clouds are in the .... ?

  SKY

- Simple solution: N-grams?
- Hard to represent patterns with more than a few words (possible patterns increases exponentially

Sequence data

- The clouds are in the .... ?
- SKY
- Simple solution: N-grams?
  - Hard to represent patterns with more than a few words (possible patterns increases exponentially
- Simple solution: Neural networks?
  - Fixed input/output size Fixed number of steps

Where is sequence in language?

Spoken language is a sequence of acoustic events over time

The temporal nature of language is reflected in the metaphors

- Flow of conversations
- News feeds
- Twitter streams

Motivation

- Not all problems can be converted into one with **fixed length inputs and outputs**

Another Motivation

Recall that we made a **Markov assumption**:

$$p(w_i \mid w_1, \ldots, w_{i-1}) = p(w_i \mid w_{i-3}, w_{i-2}, w_{i-1}).$$

This means the model is memoryless, i.e., it has no memory of anything before the last few words.

Problem:

But sometimes long-distance context can be important:

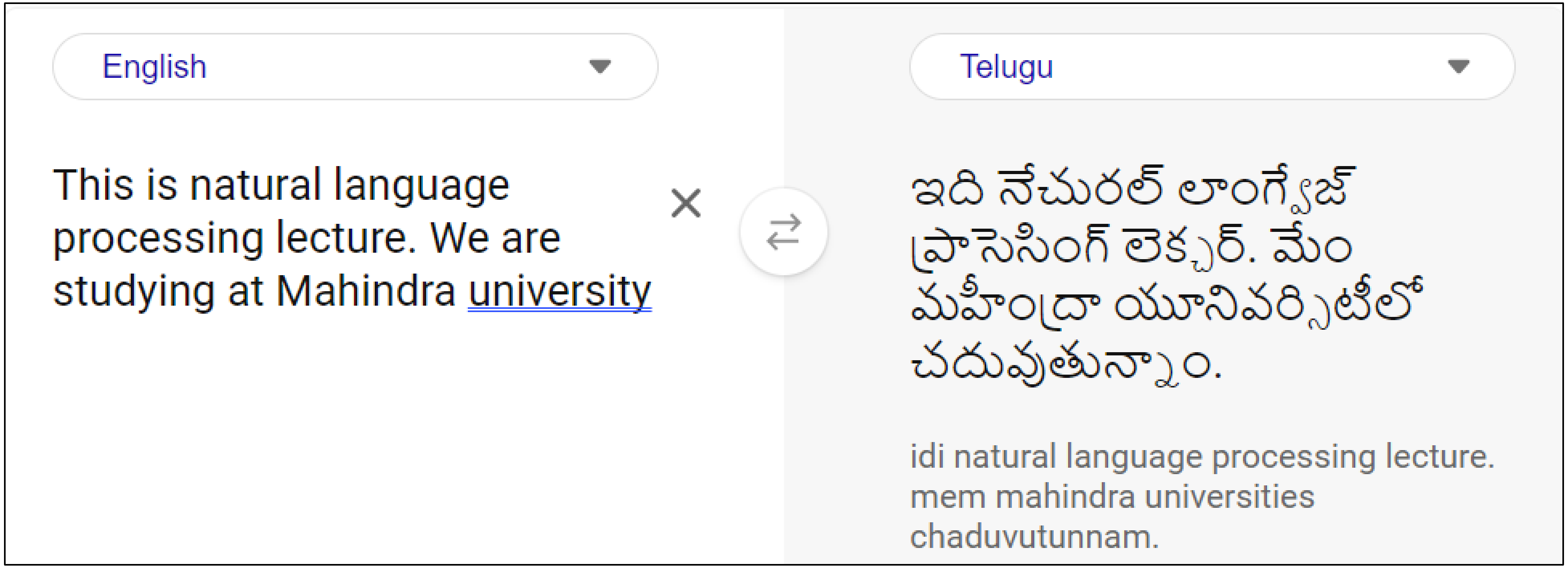
Rob Ford told the flabbergasted reporters assembled at the press conference that ________.

Motivation: Machine Translation

Consider the problem of machine translation:

– Input is text from one language

– Output is text from another language with the same meaning



Difference/Problems

A key difference with labeling:

- Input and output sequences may have different lengths and “orders”
- We do not just “find the Telugu word corresponding to the English word”
- We probably don’t know the output length

Time will explain.

Jane Austen, Persuasion


Time is what we want most,
but what we use worst.
William Penn
quotefancy

Finding structure in time

ELSEVIER

Cognitive Science

Volume 14, Issue 2, April–June 1990, Pages 179-211

Finding structure in time ☆

Jeffrey L. Elman[†]

Show more

+ Add to Mendeley Share Cite

https://doi.org/10.1016/0364-0213(90)90002-E

Recurrent Neural Networks (RNN)

- Any network that contains a cycle within its network connections, meaning that the value of some unit is directly, or indirectly, dependent on its own earlier outputs as an input.

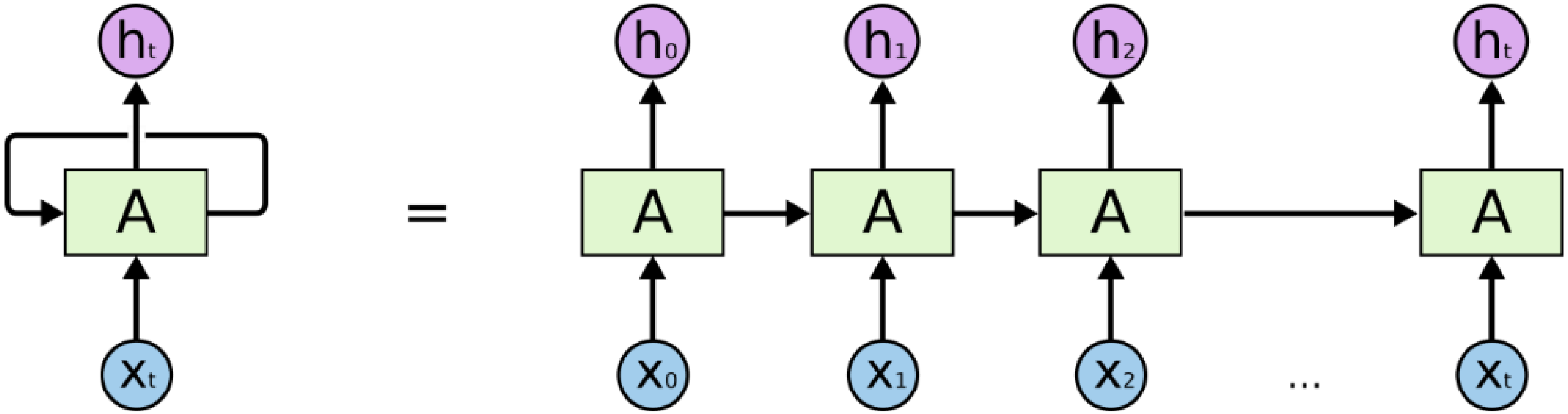


Image source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Networks (RNN)

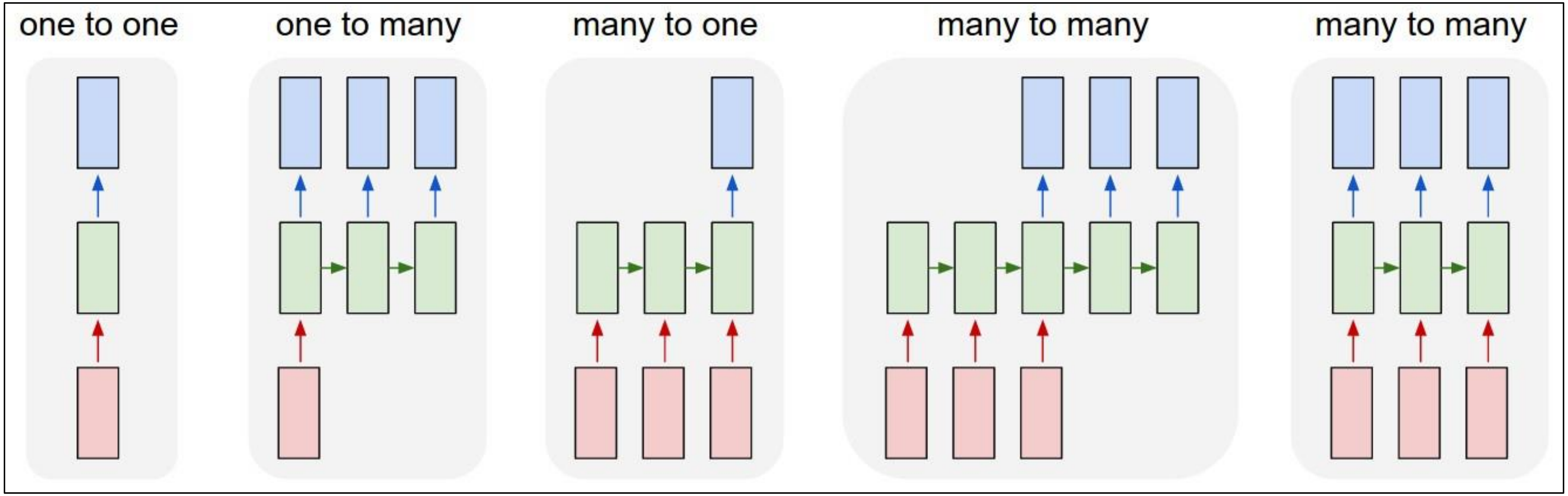
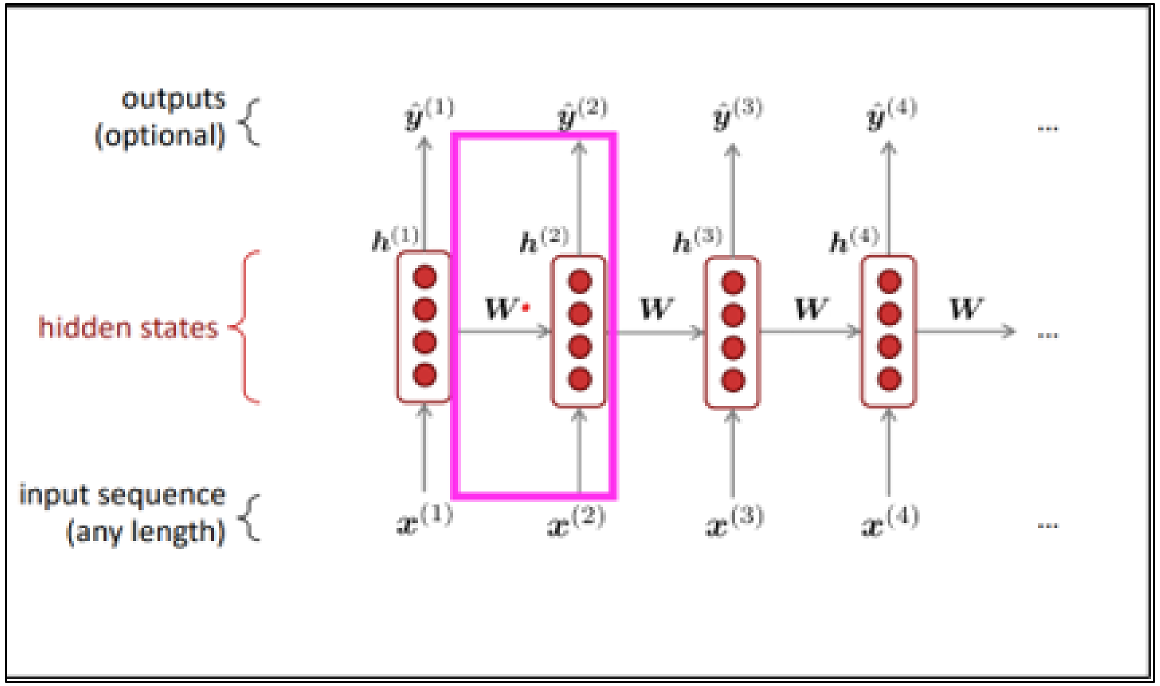


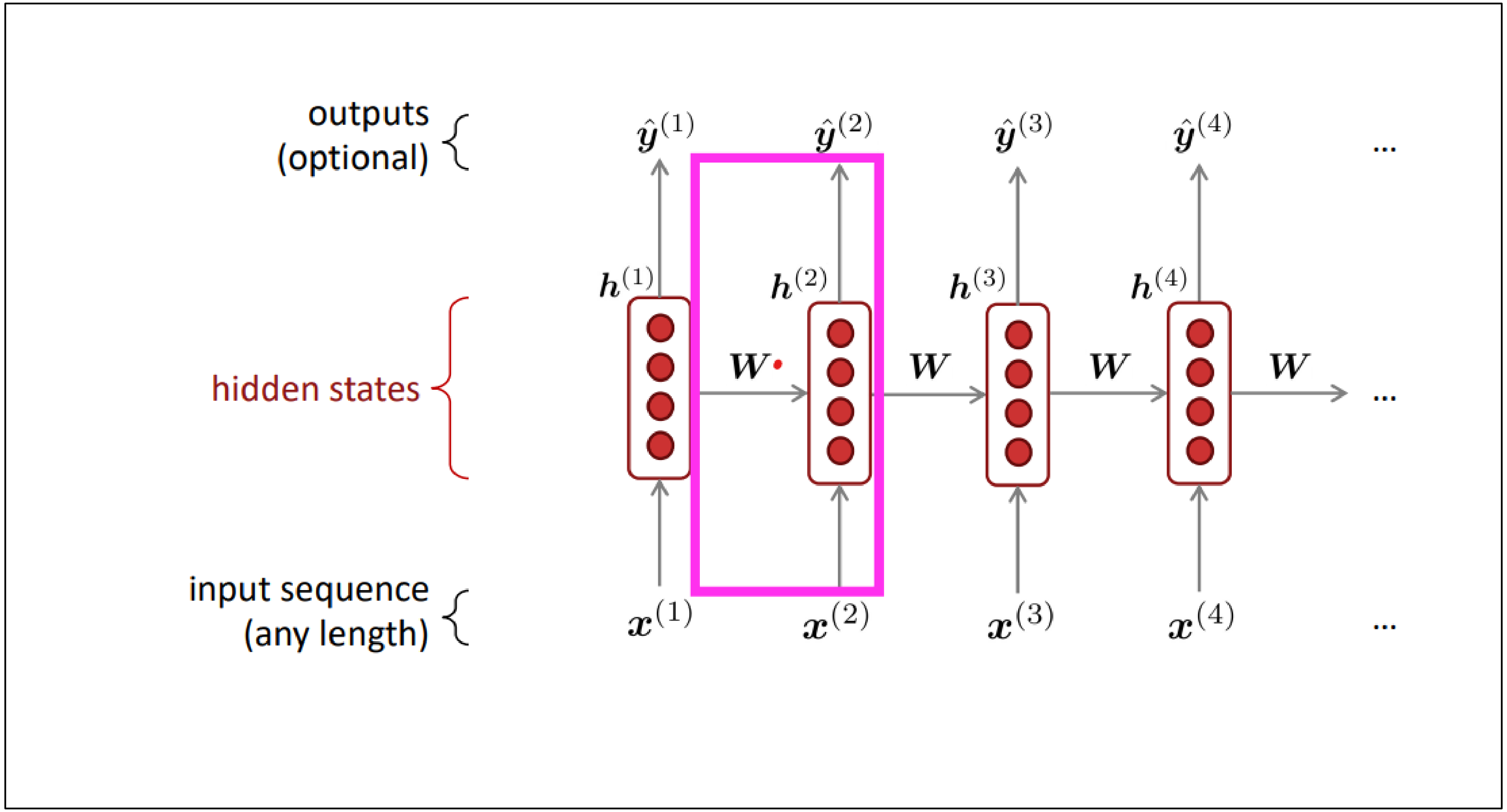
Image source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Recurrent Neural Networks

- Have memory that keeps track of information observed so far
- Maps from the entire history of previous inputs to each output
- Handle sequential data



Idea: Apply same weights repeatedly



A simple RNN Language Model

$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

some function with parameters W

RNN Language Model

output distribution

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

hidden states

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

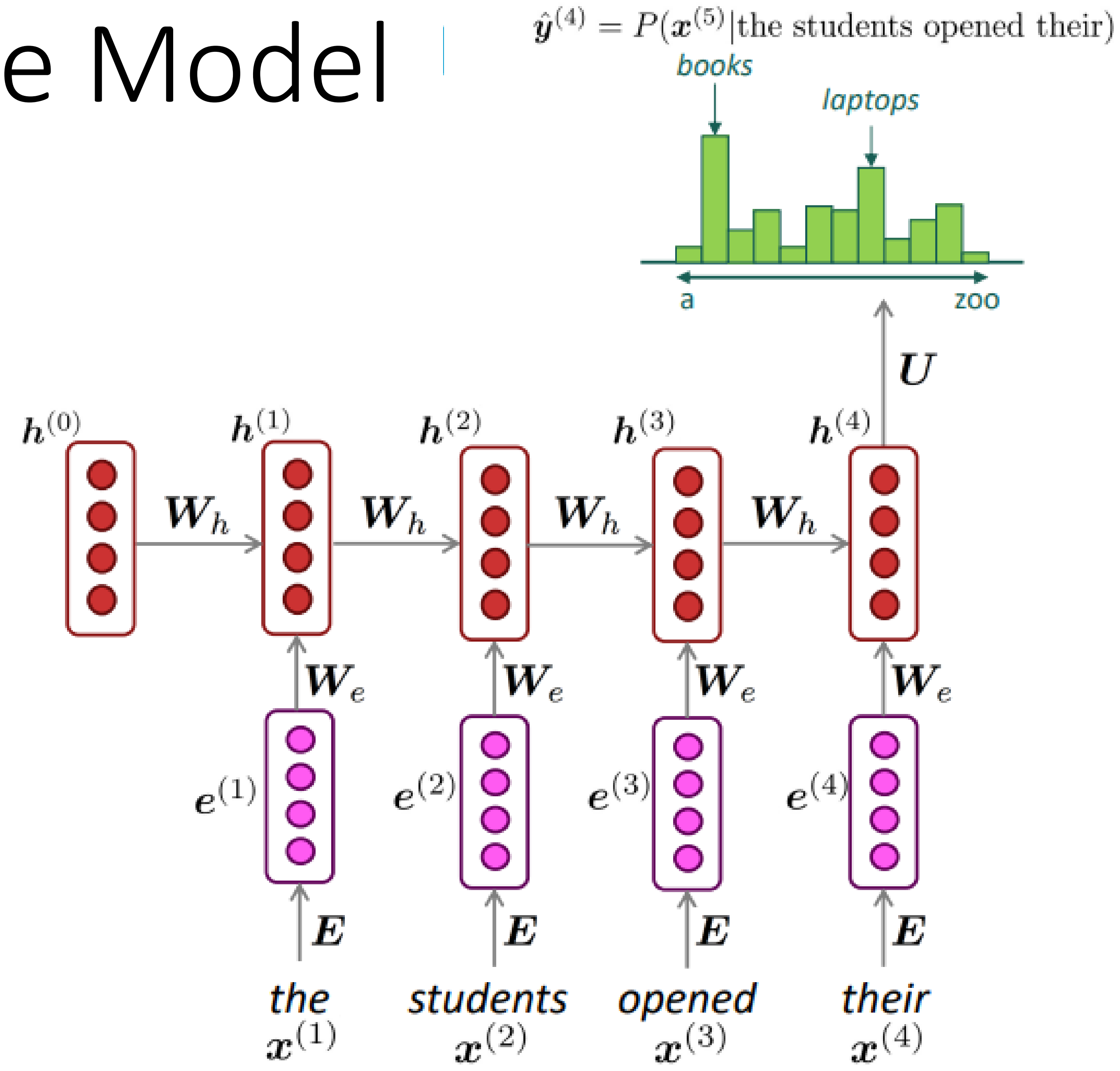
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

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

words / one-hot vectors

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



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

Credits: Slide adapted from [3]

RNN Language Models

RNN Language Models

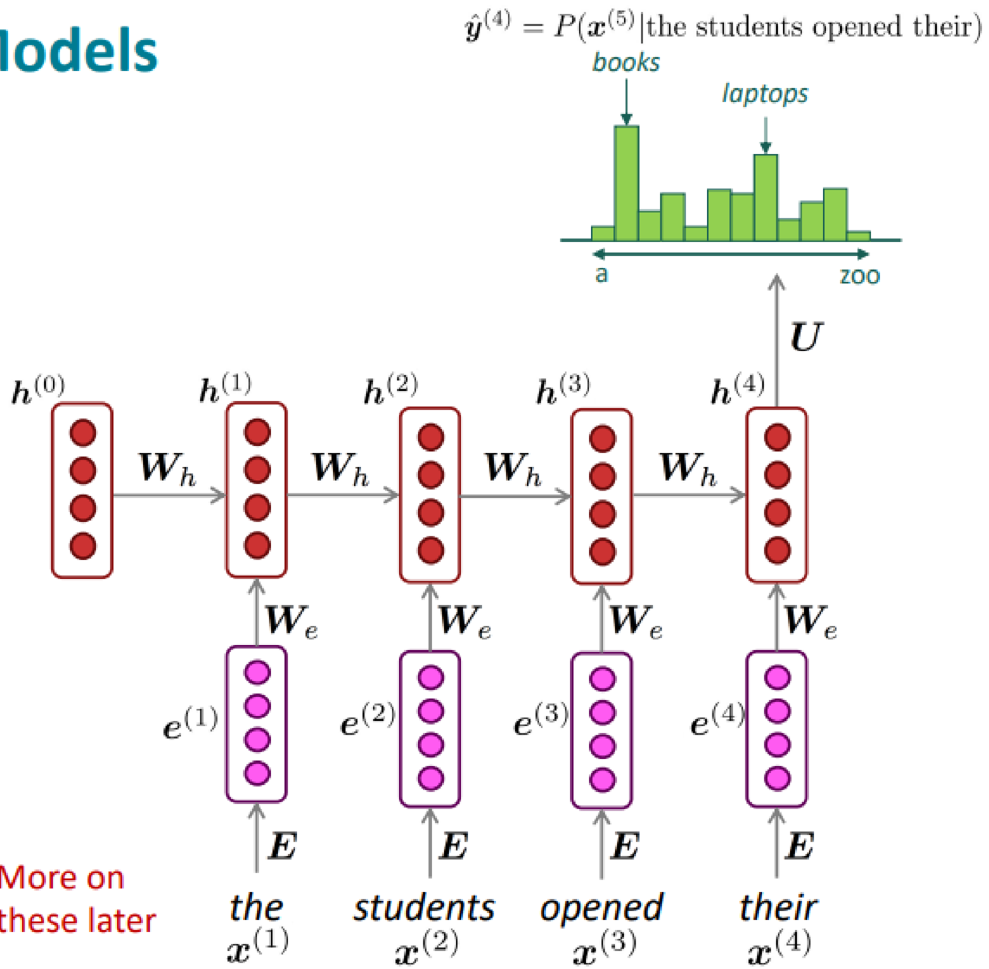
RNN **Advantages**:

- 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**:

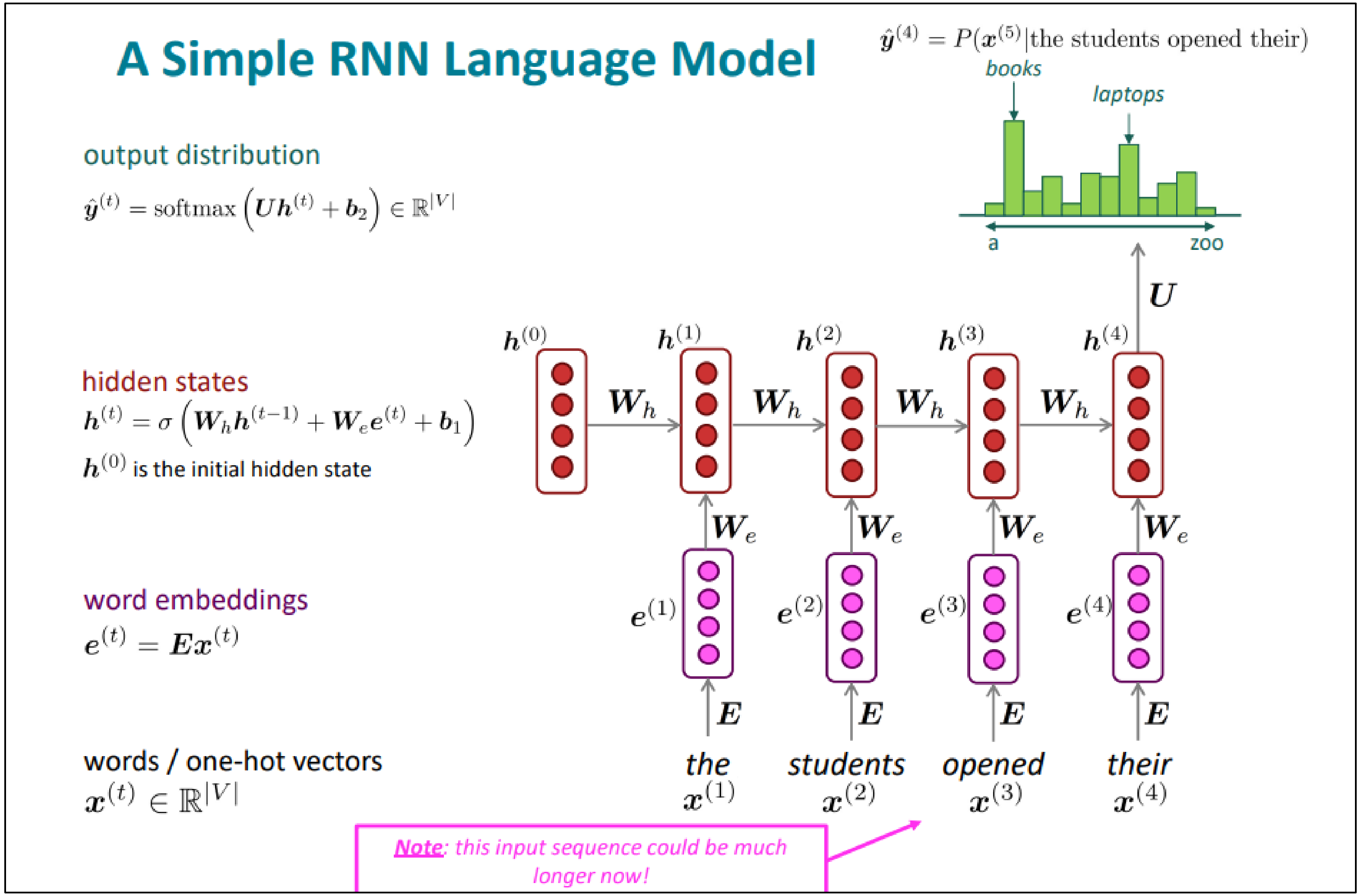
- Recurrent computation is slow
- In practice, difficult to access information from many steps back

More on these later



Credits: Slide adapted from [3]

A simple RNN Language Model



Credits: Slide adapted from [3]

Training a 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 $\boldsymbol{x}^{(t+1)}$):

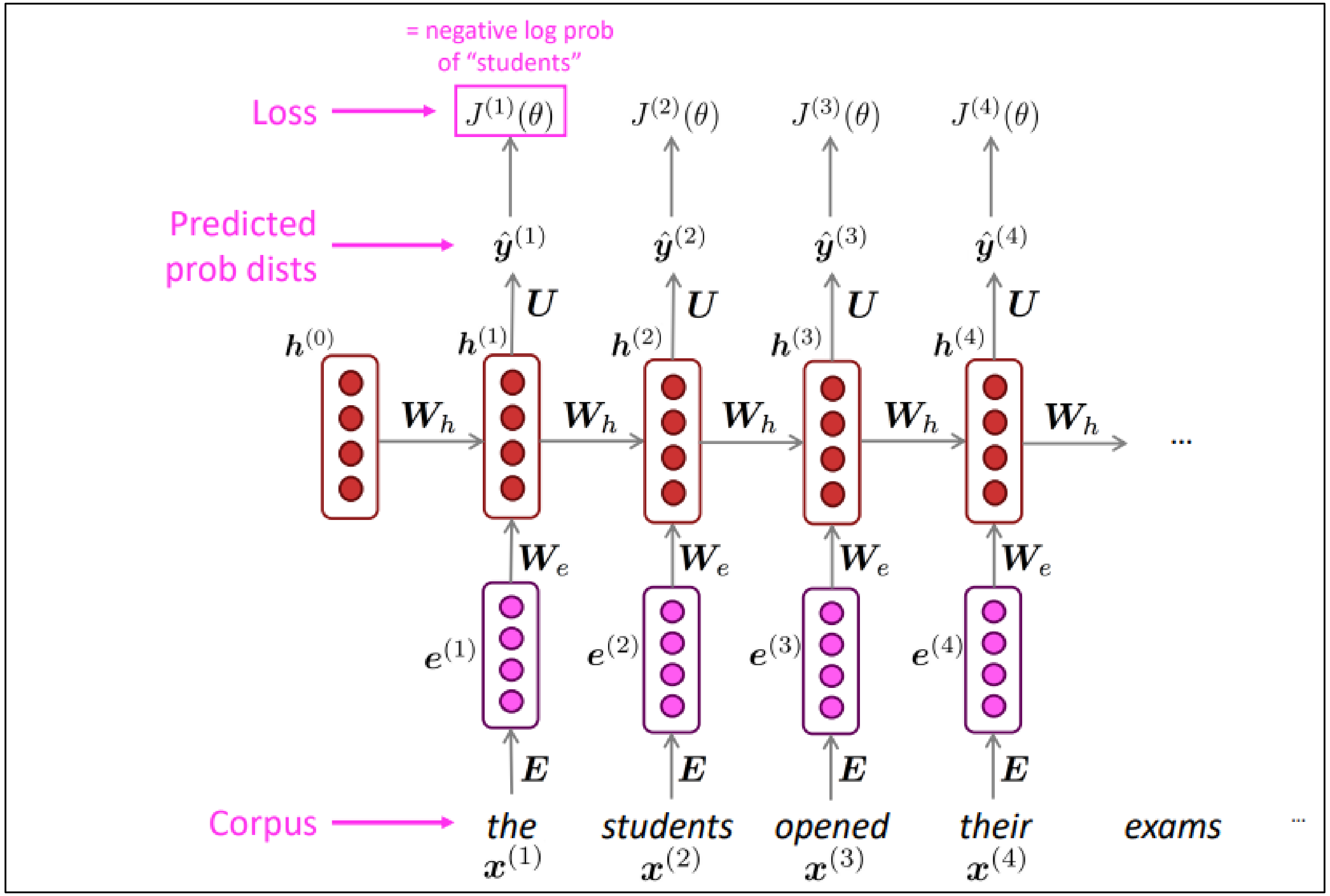
$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

Average this to get overall loss for entire training set:

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

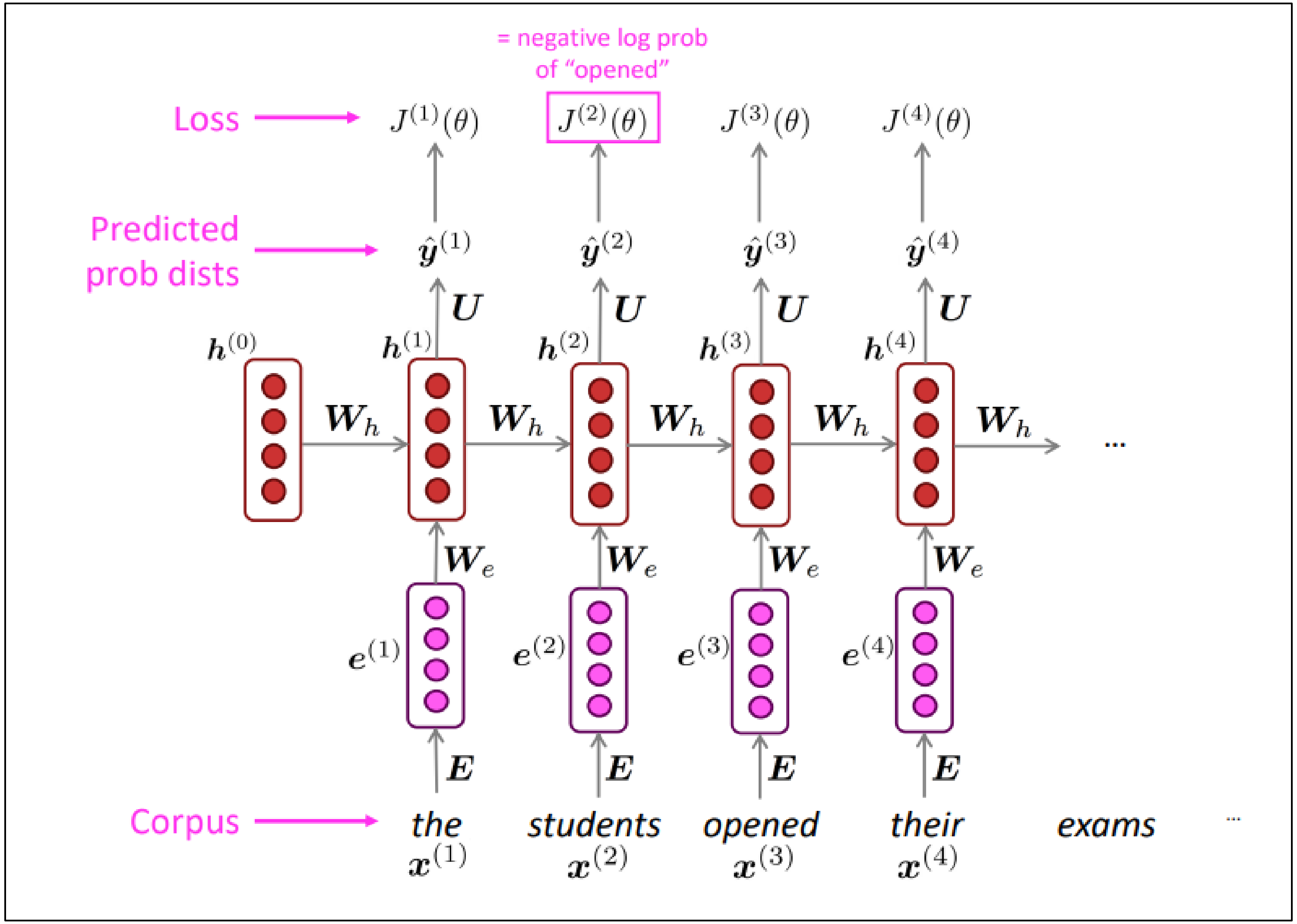
Credits: Slide adapted from [3]

Training a RNN language model



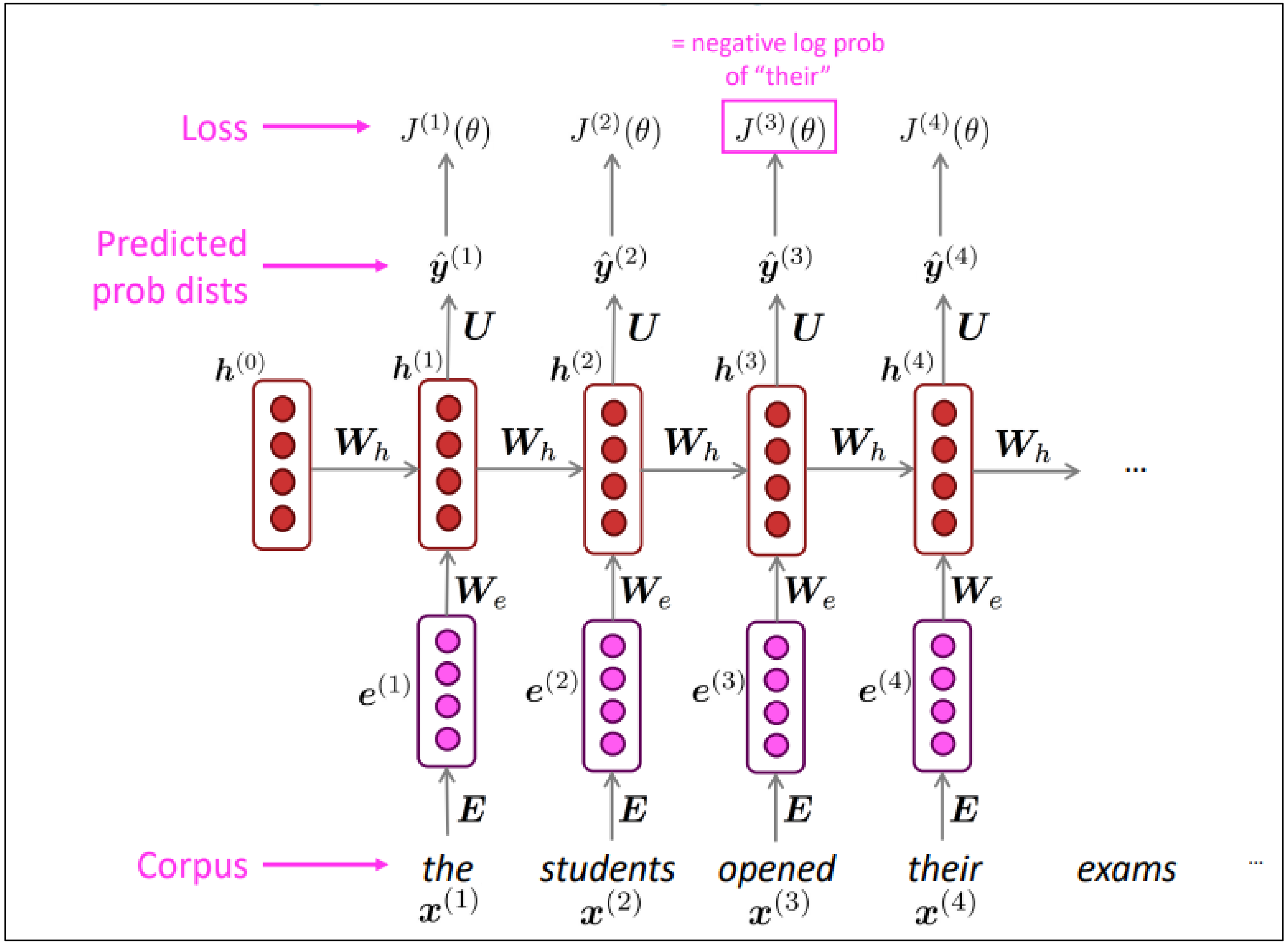
Credits: Slide adapted from [3]

Training a RNN language model



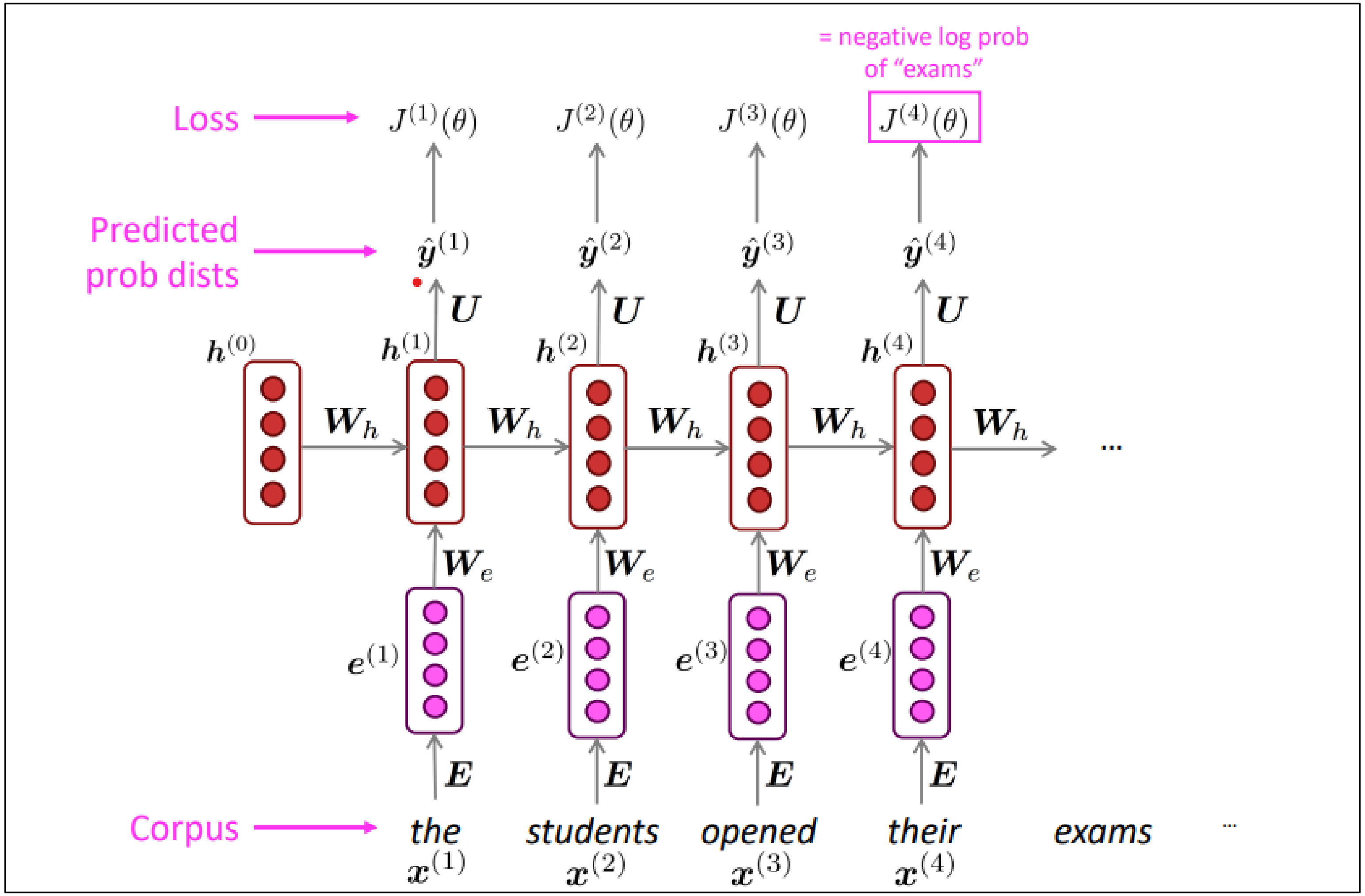
Credits: Slide adapted from [3]

Training a RNN language model



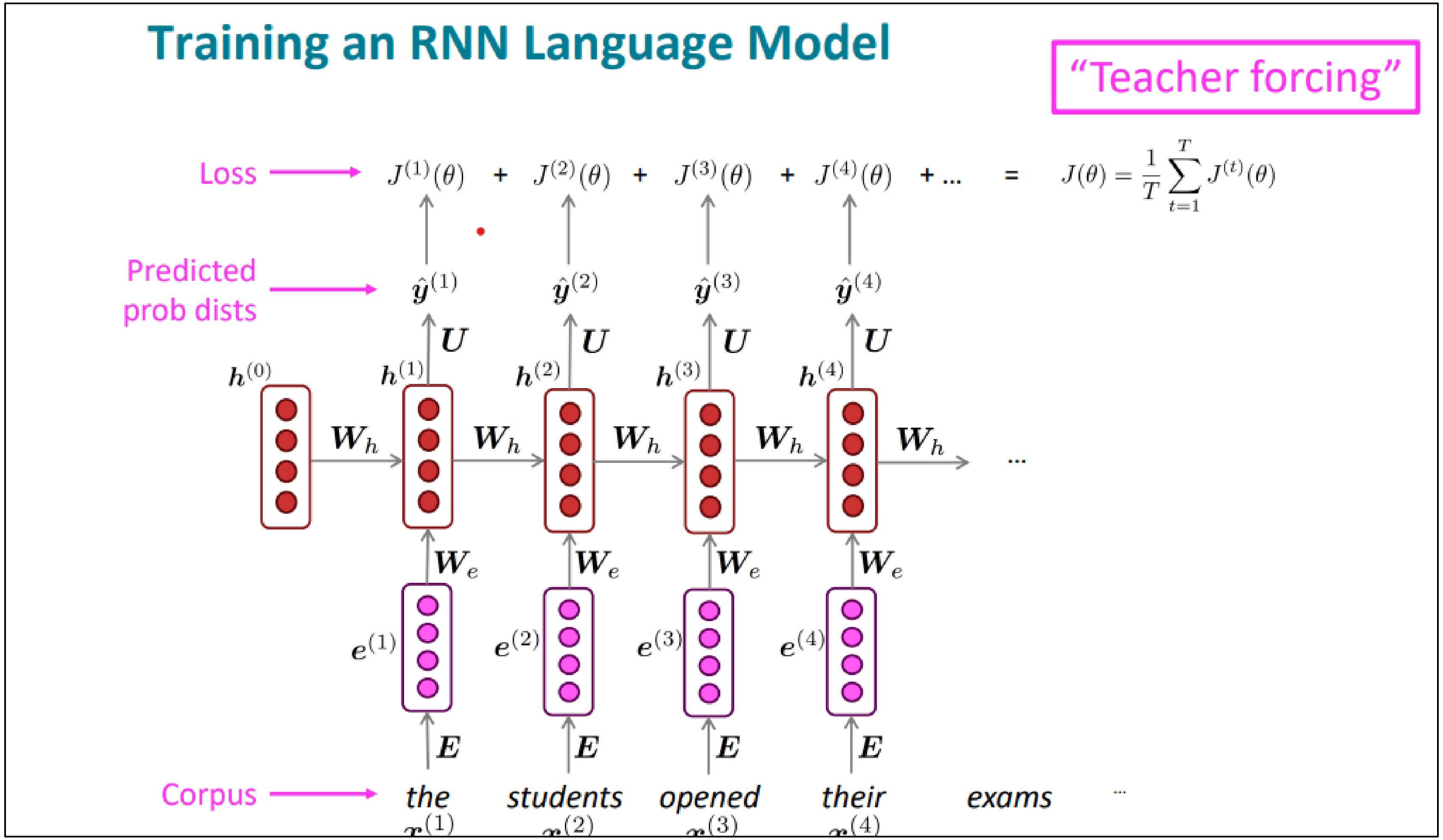
Credits: Slide adapted from [3]

Training a RNN language model



Credits: Slide adapted from [3]

Training a RNN language model



Credits: Slide adapted from [3]

Training a RNN language model

- However: *Computing loss and gradients across $\{x_1, x_2, \ldots, x_t\}$, the entire corpus at once is too expensive (memory-wise)!*

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

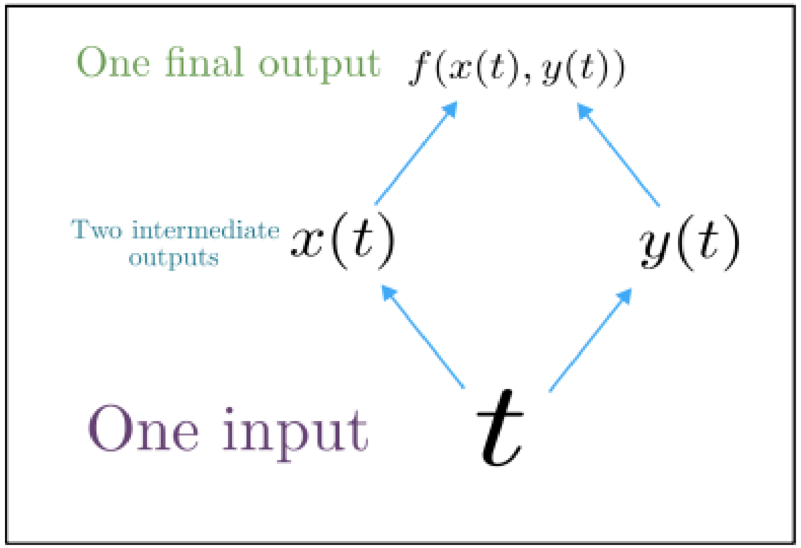
- Consider as a sentence (or a document)
- Recall: Stochastic Gradient Descent allows us to compute loss $J(\theta)$ and gradients for small chunk of data, and update.
- Compute loss $J(\theta)$, for a sentence (actually, a batch of sentences), compute gradients and update weights. Repeat on a new batch of sentences.

Credits: Slide adapted from [3]

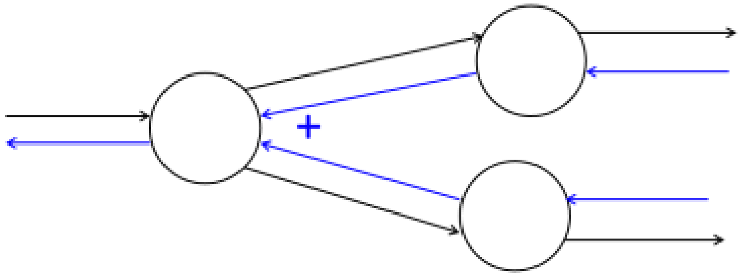
Multivariable Chain Rule

- 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:

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



Gradients sum at outward branches

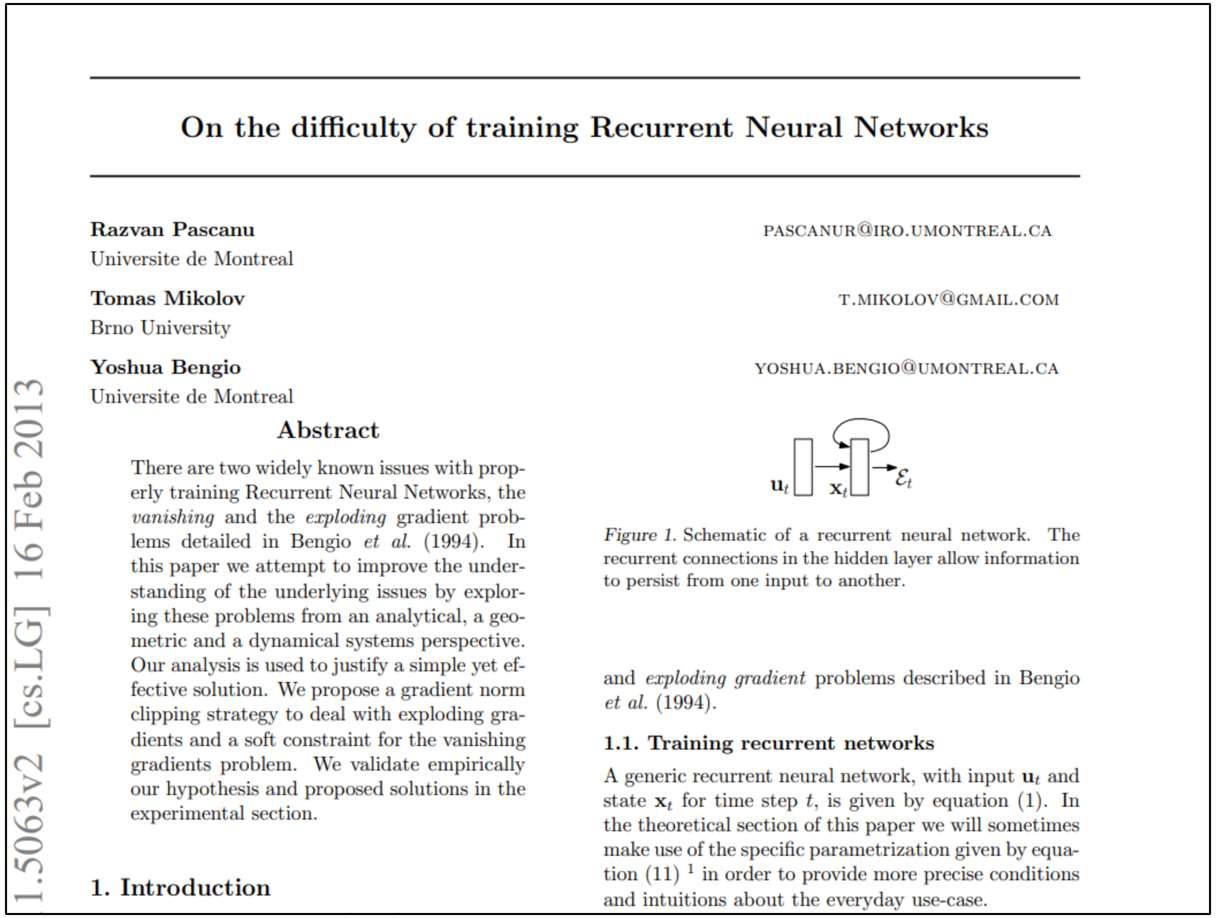


$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial y} + \frac{\partial f}{\partial b}\frac{\partial b}{\partial y}$$

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

Issues with RNN: Vanishing and Exploding Gradients



On the difficulty of training Recurrent Neural Networks

Razvan Pascanu PASCANUR@IRO.UMONTREAL.CA
Universite de Montreal

Tomas Mikolov T.MIKOLOV@GMAIL.COM
Brno University

Yoshua Bengio YOSHUA.BENGIO@UMONTREAL.CA
Universite de Montreal

1.5063v2 [cs.LG] 16 Feb 2013

Abstract

There are two widely known issues with properly training Recurrent Neural Networks, the *vanishing* and the *exploding* gradient problems detailed in Bengio *et al.* (1994). In this paper we attempt to improve the understanding of the underlying issues by exploring these problems from an analytical, a geometric and a dynamical systems perspective. Our analysis is used to justify a simple yet effective solution. We propose a gradient norm clipping strategy to deal with exploding gradients and a soft constraint for the vanishing gradients problem. We validate empirically our hypothesis and proposed solutions in the experimental section.

1. Introduction

$\mathbf{u}_t$ $\mathbf{x}_t$ $\mathcal{E}_t$

Figure 1. Schematic of a recurrent neural network. The recurrent connections in the hidden layer allow information to persist from one input to another.

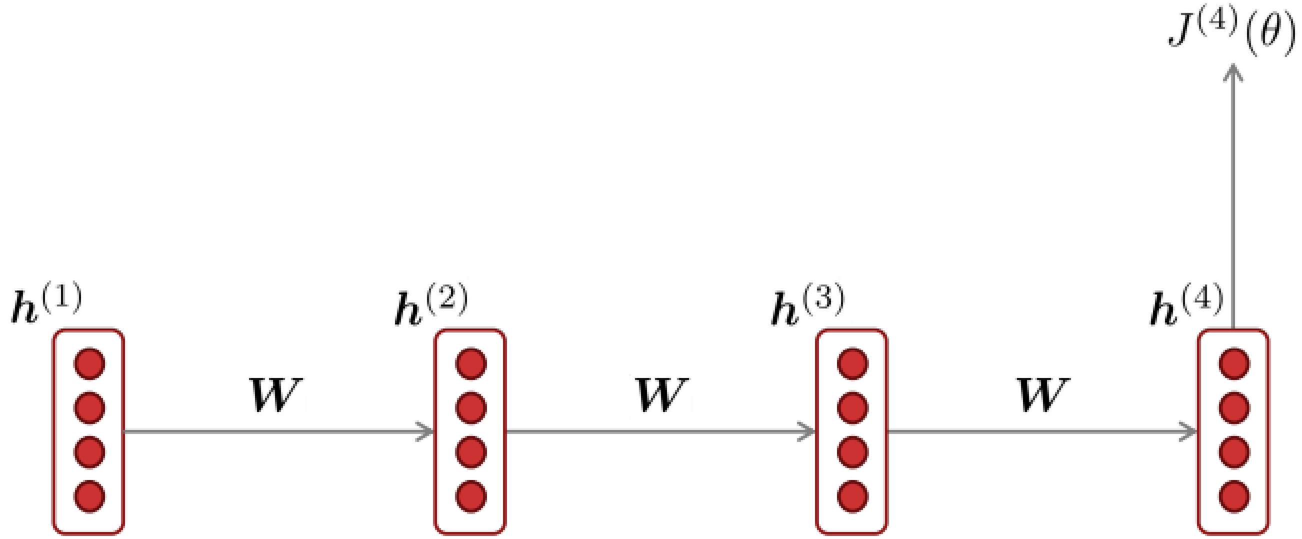
and *exploding gradient* problems described in Bengio *et al.* (1994).

1.1. Training recurrent networks

A generic recurrent neural network, with input $\mathbf{u}_t$ and state $\mathbf{x}_t$ for time step t, is given by equation (1). In the theoretical section of this paper we will sometimes make use of the specific parametrization given by equation (11) [1] in order to provide more precise conditions and intuitions about the everyday use-case.

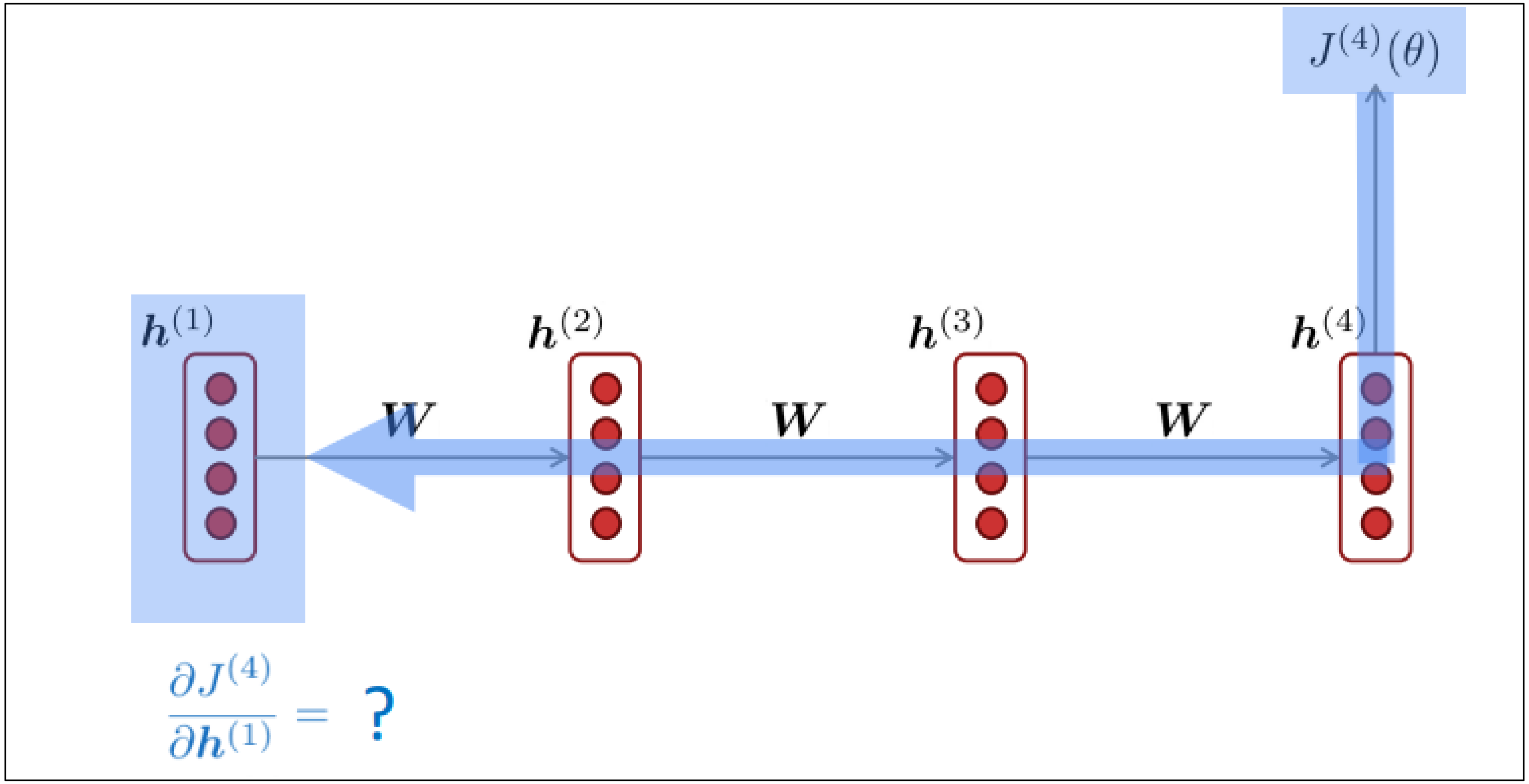
[On the difficulty of training recurrent neural networks, Pascanu et al., 2013](#)

Problems with RNNs: Vanishing and Exploding Gradients



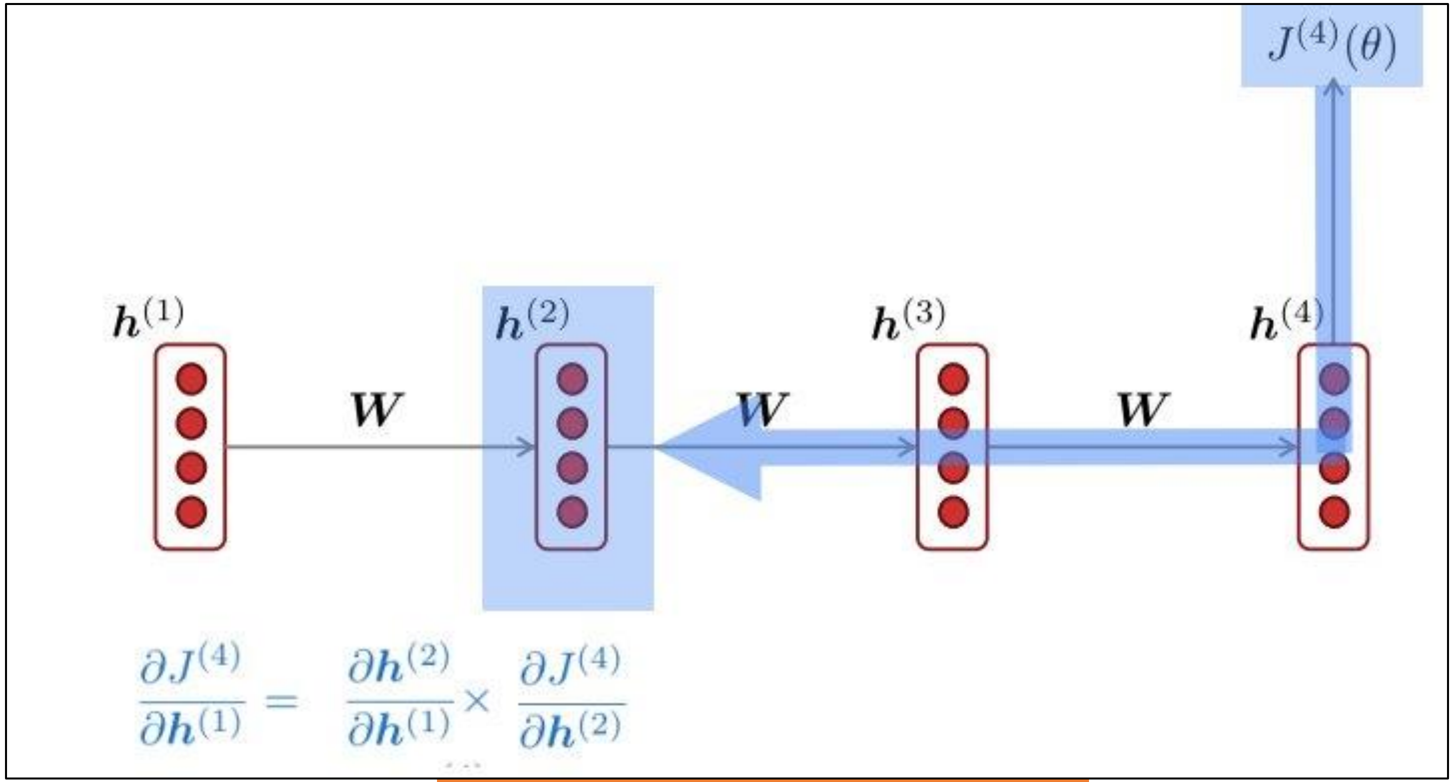
Credits: Slide adapted from [3]

Vanishing Gradient Intuition



Credits: Slide adapted from [3]

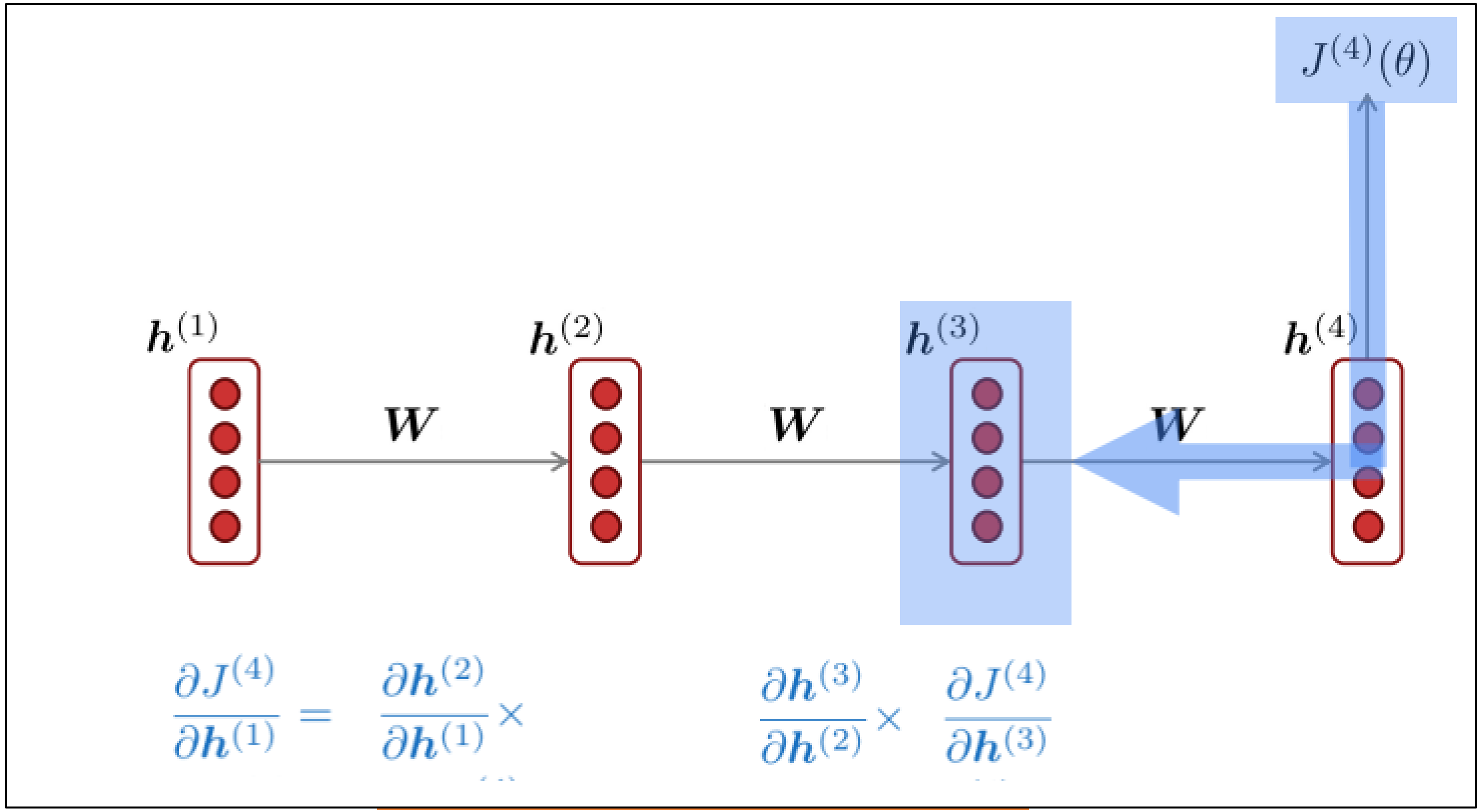
Vanishing Gradient Intuition



Chain Rule!!!!!!

Credits: Slide adapted from [3]

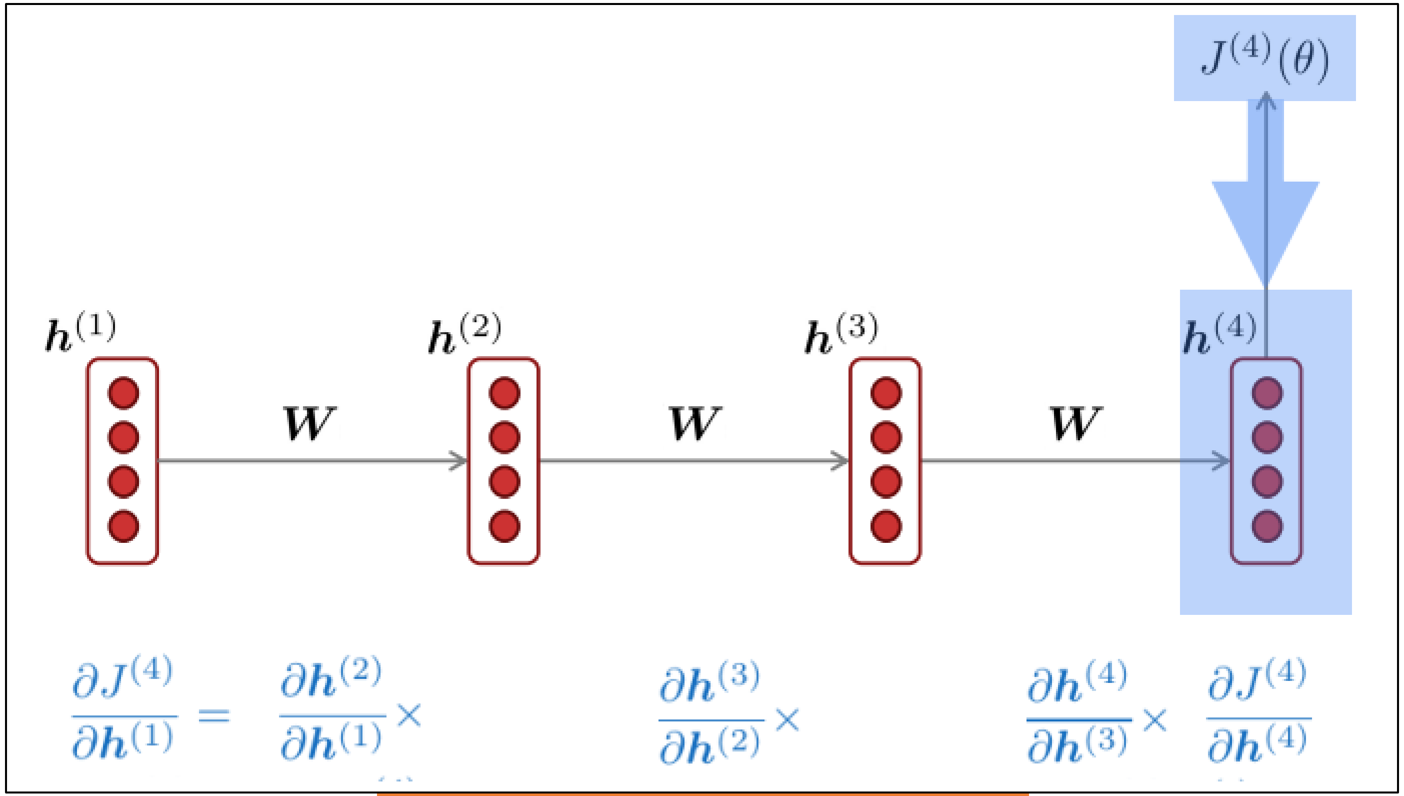
Vanishing Gradient Intuition



Chain Rule!!!!!!

Credits: Slide adapted from [3]

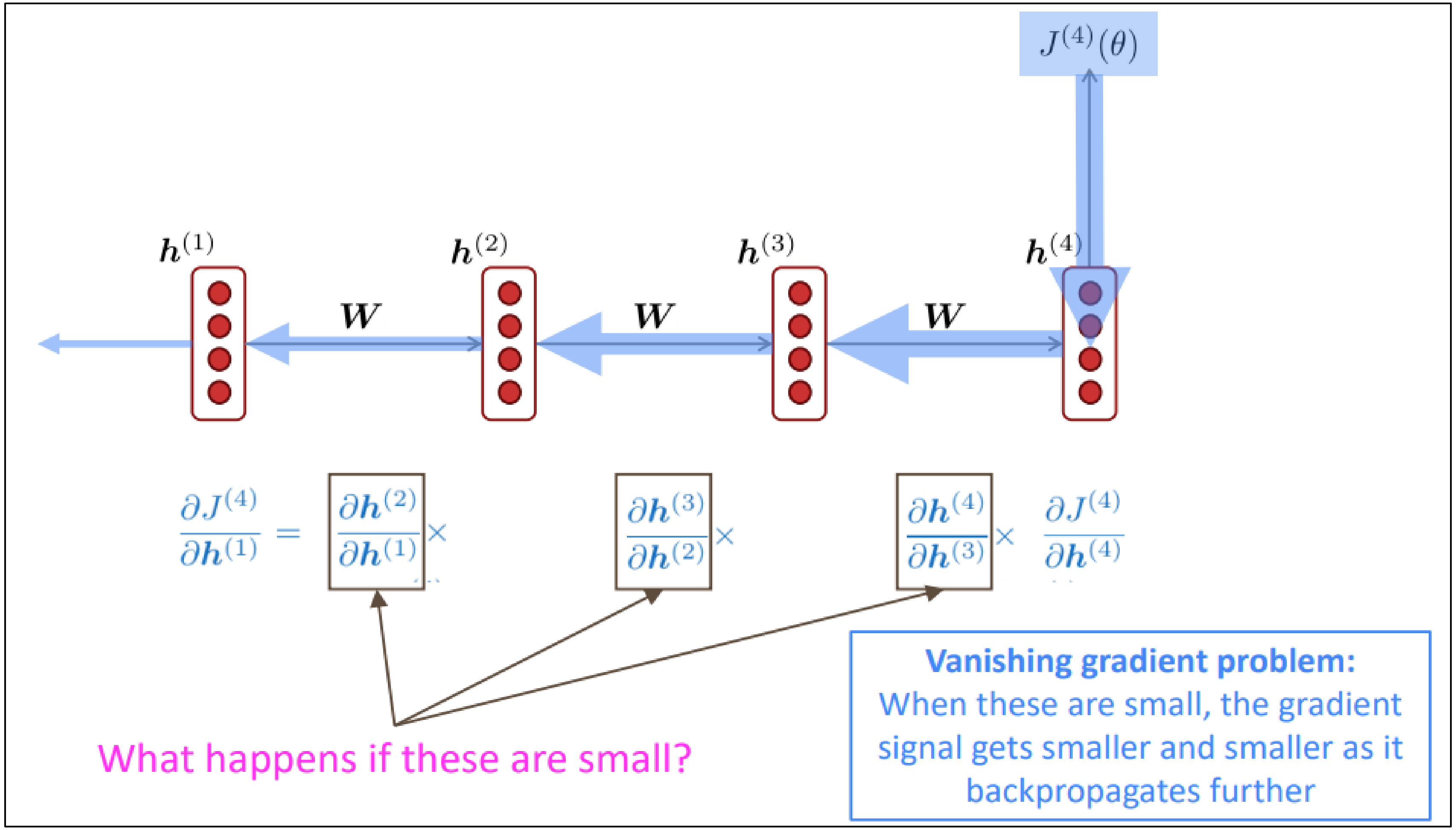
Vanishing Gradient Intuition



Chain Rule!!!!!!

Credits: Slide adapted from [3]

Vanishing Gradient Intuition



Credits: Slide adapted from [3]

Vanishing gradient proof sketch

Recall: $$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x\boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)$$

What if σ were the identity function, $\sigma(x) = x$?

$$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} = \operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x\boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right)\boldsymbol{W}_h$$ (chain rule)

$$= \boldsymbol{I}\ \boldsymbol{W}_h = \boldsymbol{W}_h$$

Consider the gradient of the loss $J^{(i)}(\theta)$ on step i, with respect to the hidden state $\boldsymbol{h}^{(j)}$ on some previous step j. *Let* $\ell = i - j$

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$$ (chain rule)

$$= \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \boldsymbol{W}_h = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boxed{\boldsymbol{W}_h^{\ell}}$$ (value of $\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$)

If W_h is "small", then this term gets exponentially problematic as ℓ becomes large

Source: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf (and supplemental materials), at http://proceedings.mlr.press/v28/pascanu13-supp.pdf

Credits: Slide adapted from [3]

Vanishing gradient proof sketch

What's wrong with $\boldsymbol{W}_h^\ell$?

sufficient but not necessary

Consider if the eigenvalues of $\boldsymbol{W}_h$ are $\underbrace{\text{all less than}}_{\text{sufficient but not necessary}}$ 1:

$$\lambda_1, \lambda_2, \ldots, \lambda_n < 1$$
$$\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_n \text{ (eigenvectors)}$$

We can write $\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boldsymbol{W}_h^\ell$ using the eigenvectors of $\boldsymbol{W}_h$ as a basis:

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boldsymbol{W}_h^\ell = \sum_{i=1}^{n} c_i \boxed{\lambda_i^\ell} \boldsymbol{q}_i \approx \boldsymbol{0} \text{ (for large } \ell)$$

Approaches 0 as ℓ grows, so gradient vanishes

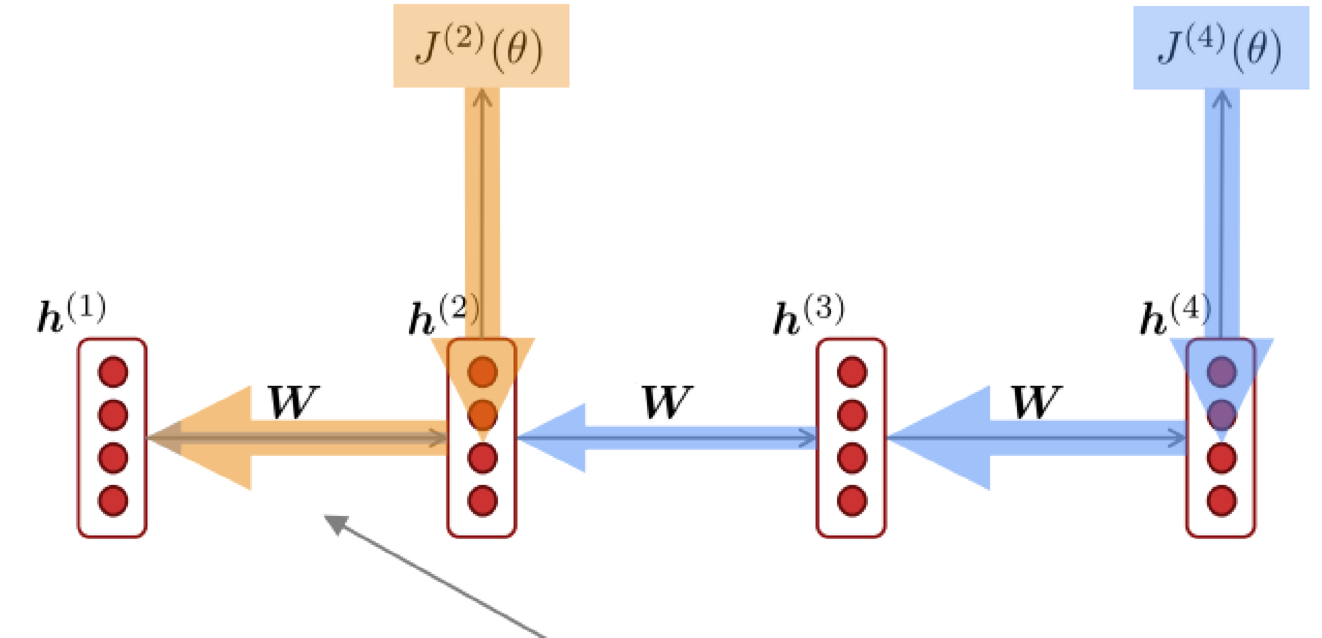
What about nonlinear activations σ (i.e., what we use?)

- Pretty much the same thing, except the proof requires $\lambda_i < \gamma$ for some γ dependent on dimensionality and σ

: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf (and supplemental materials), at http://proceedings.mlr.press/v28/pascanu13-supp.pdf

Credits: Slide adapted from [3]

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 basically updated only with respect to near effects, not long-term effects.

Effect of vanishing gradient on RNN

- **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 7th 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
- In practice a simple RNN will only condition ~7 tokens back [vague rule-of-thumb]

Credits: Slide adapted from [3]

Gradient Clipping: A 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

$$
\begin{aligned}
&\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta} \\
&\textbf{if} \ \ \|\hat{\mathbf{g}}\| \geq threshold \ \textbf{then} \\
&\quad \hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}} \\
&\textbf{end if}
\end{aligned}
$$

- **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

Source: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf

Is vanishing Gradient only a RNN problem?

- No! It can be a problem for all neural architectures (including feed-forward and convolutional), especially very deep ones.
- Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
- Thus, lower layers are learned very slowly (i.e., are hard to train)

Credits: Slide adapted from [3]

RNN improves 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

Perplexity improves (lower is better)

LSTMs (Long Short-Term Memory)

Long Short-Term Memory, Hochreiter et al., 1997

LONG SHORT-TERM MEMORY

NEURAL COMPUTATION 9(8):1735–1780, 1997

Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
http://www7.informatik.tu-muenchen.de/~hochreit

Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
http://www.idsia.ch/~juergen

Abstract

Learning to store information over extended time intervals via recurrent backpropagation takes a very long time, mostly due to insufficient, decaying error back flow. We briefly review Hochreiter's 1991 analysis of this problem, then address it by introducing a novel, efficient, gradient-based method called "Long Short-Term Memory" (LSTM). Truncating the gradient where this does not do harm, LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing *constant* error flow through "constant error carrousels" within special units. Multiplicative gate units learn to open and close access to the constant error flow. LSTM is local in space and time; its computational complexity per time step and weight is $O(1)$. Our experiments with artificial data involve local, distributed, real-valued, and noisy pattern representations. In comparisons with RTRL, BPTT, Recurrent Cascade-Correlation, Elman nets, and Neural Sequence Chunking, LSTM leads to many more successful runs, and learns much faster. LSTM also solves complex, artificial long time lag tasks that have never been solved by previous recurrent network algorithms.

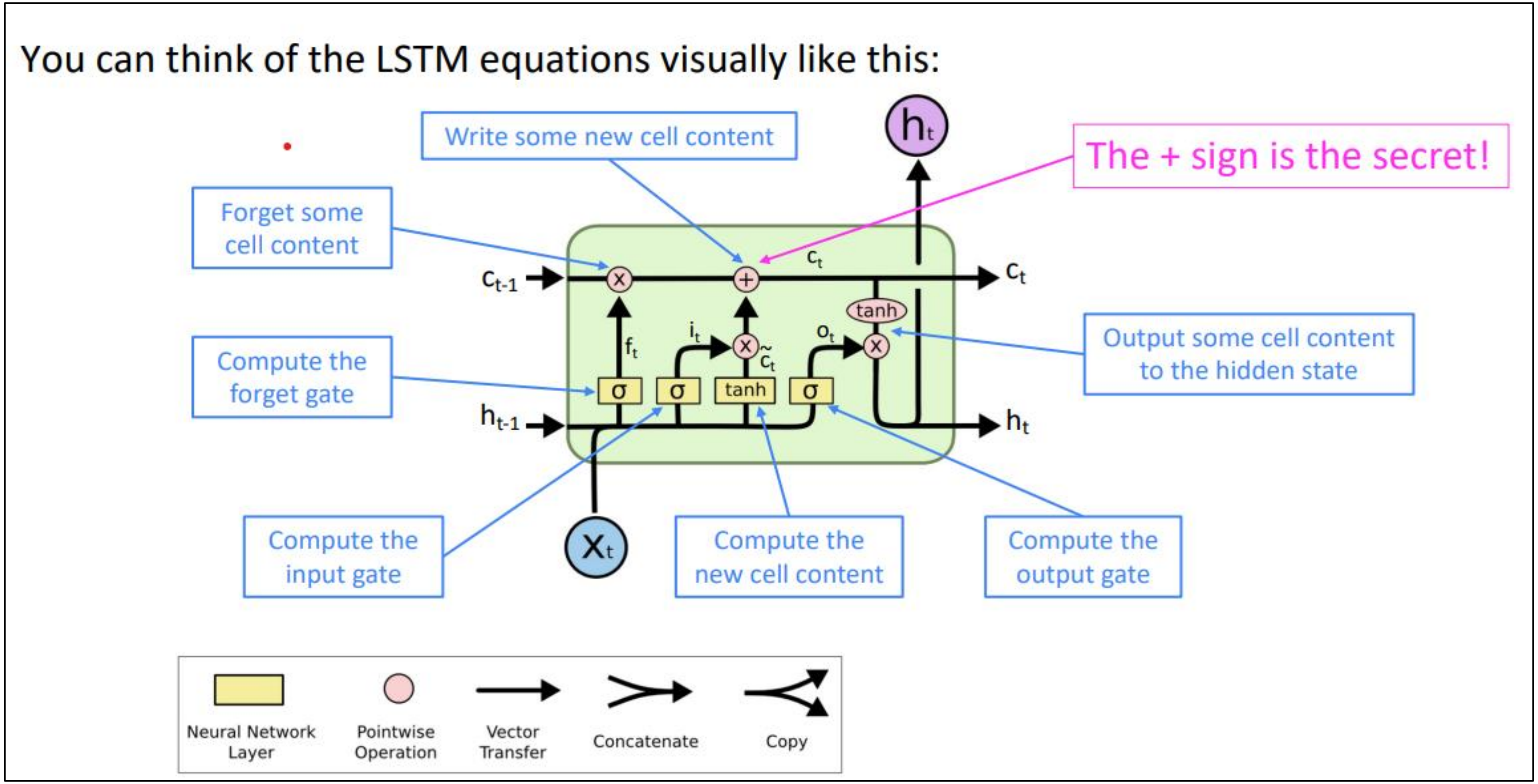
LSTM solve Vanishing Gradient Problem?

- The LSTM architecture makes it much easier for an RNN to preserve information over many timesteps
  - If the *forget gate* is set to *1* for a cell dimension and the *input gate* set to *0*, then the information of that cell is preserved indefinitely.
  - In contrast, it's harder for a *vanilla RNN* to learn a recurrent weight matrix W_h that preserves info in the hidden state
  - In practice, you get about *100 timesteps* rather than about *7*

- However, there are alternative ways of creating more direct and linear pass-through connections in models for long distance dependencies

Credits: Slide adapted from [3]

LSTM Equations



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

References

[1] https://www.cs.ubc.ca/~dsuth/440/23w2/slides/9-rnn.pdf

[2] https://slazebni.cs.illinois.edu/spring17/lec02_rnn.pdf

[3] https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1234/slides/cs224n-2023-lecture06-fancy-rnn.pdf

Reference materials

- https://vlanc-lab.github.io/mu-nlp-course/

- Lecture notes

- (A) Speech and Language Processing by Daniel Jurafsky and James H. Martin
- (B) Natural Language Processing with Python. (updated edition based on Python 3 and NLTK 3) Steven Bird et al. O'Reilly Media