

# Recurrent Neural Networks (RNNs) and LSTM

# Sequential Data

Sometimes the sequence of data matters

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



# 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**:

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

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

**Problem:**

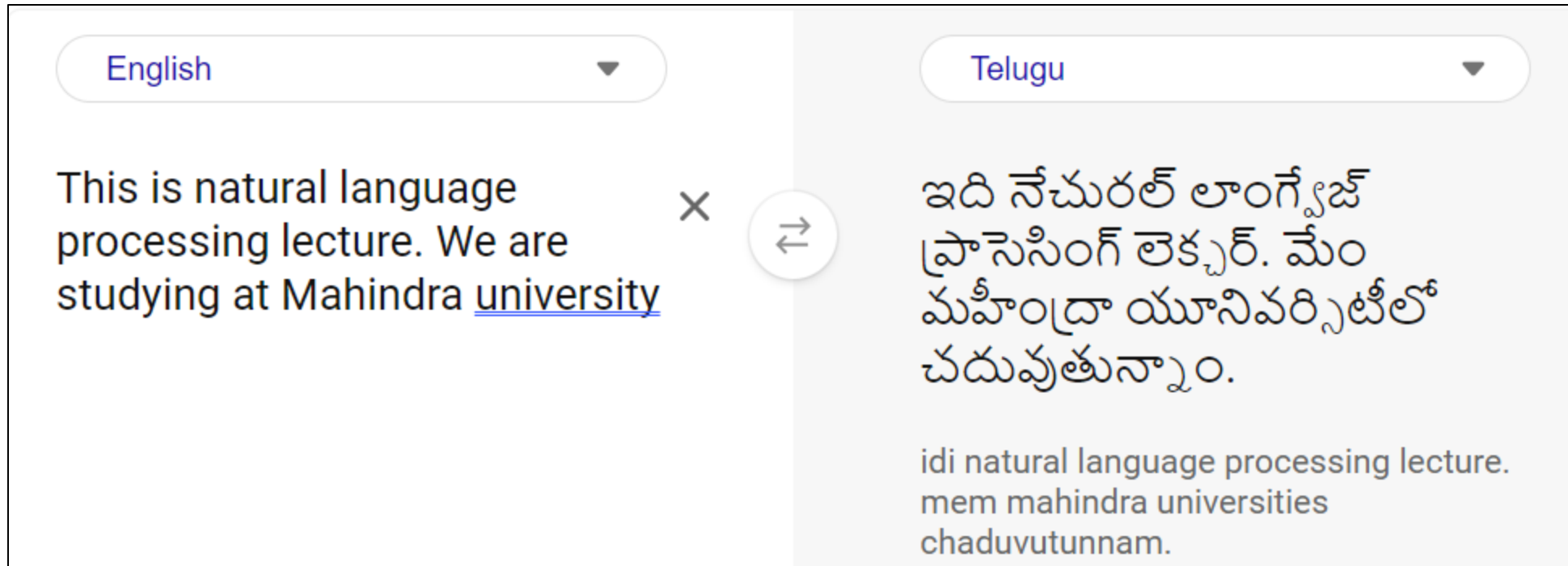
But sometimes **long-distance context** can be important:

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



The screenshot displays a machine translation interface with two main panels. The left panel, labeled 'English', contains the text: 'This is natural language processing lecture. We are studying at Mahindra university'. The right panel, labeled 'Telugu', shows the translated text in Telugu script: 'ఇది నేచురల్ లాంగ్వేజ్ ప్రాసెసింగ్ లెక్చర్. మేం మహింద్రా యూనివర్సిటీలో చదువుతున్నాం.' Below this, a less accurate machine translation is shown in English: 'idi natural language processing lecture. mem mahindra universities chaduvutunnam.' A central icon with a double-headed arrow and a cross indicates the translation process.

English	Telugu
This is natural language processing lecture. We are studying at Mahindra <u>university</u>	ఇది నేచురల్ లాంగ్వేజ్ ప్రాసెసింగ్ లెక్చర్. మేం మహింద్రా యూనివర్సిటీలో చదువుతున్నాం.
	idi natural language processing lecture. mem mahindra universities chaduvutunnam.

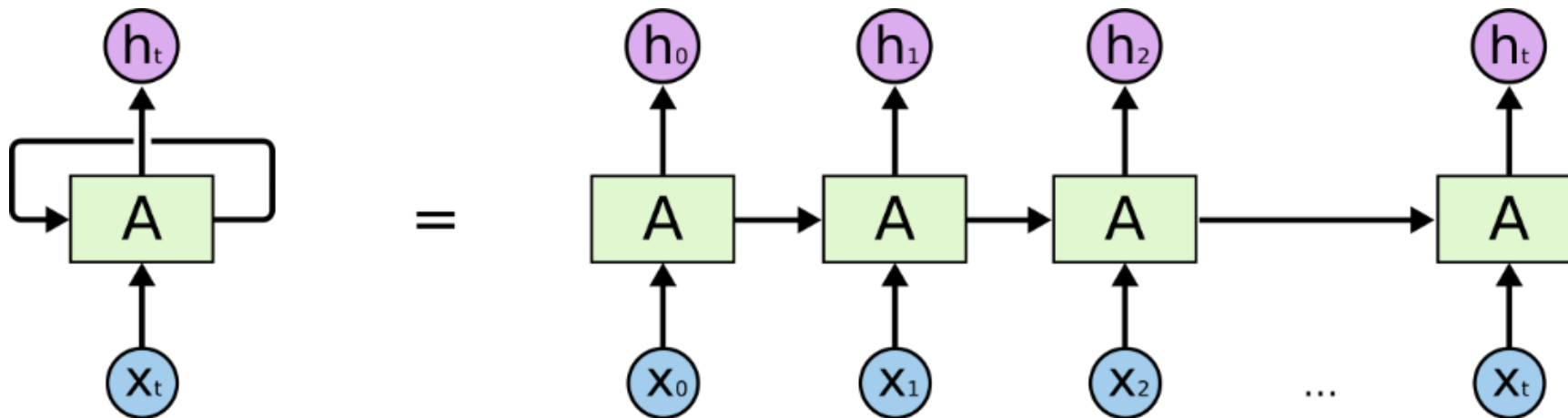
# Difference/Problems

A key difference with labeling:

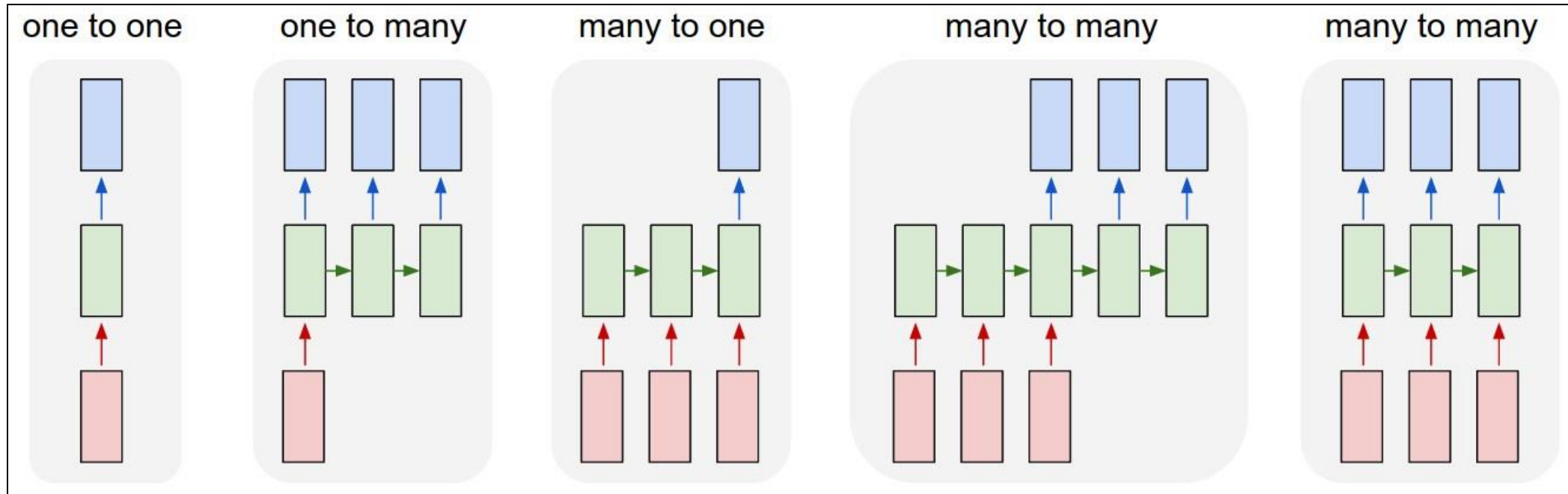
- Input and output sequences may have different lengths and “orders”
- We do not just “find the Telugu word corresponding to the English word”
- We probably don’t know the output length

# 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.

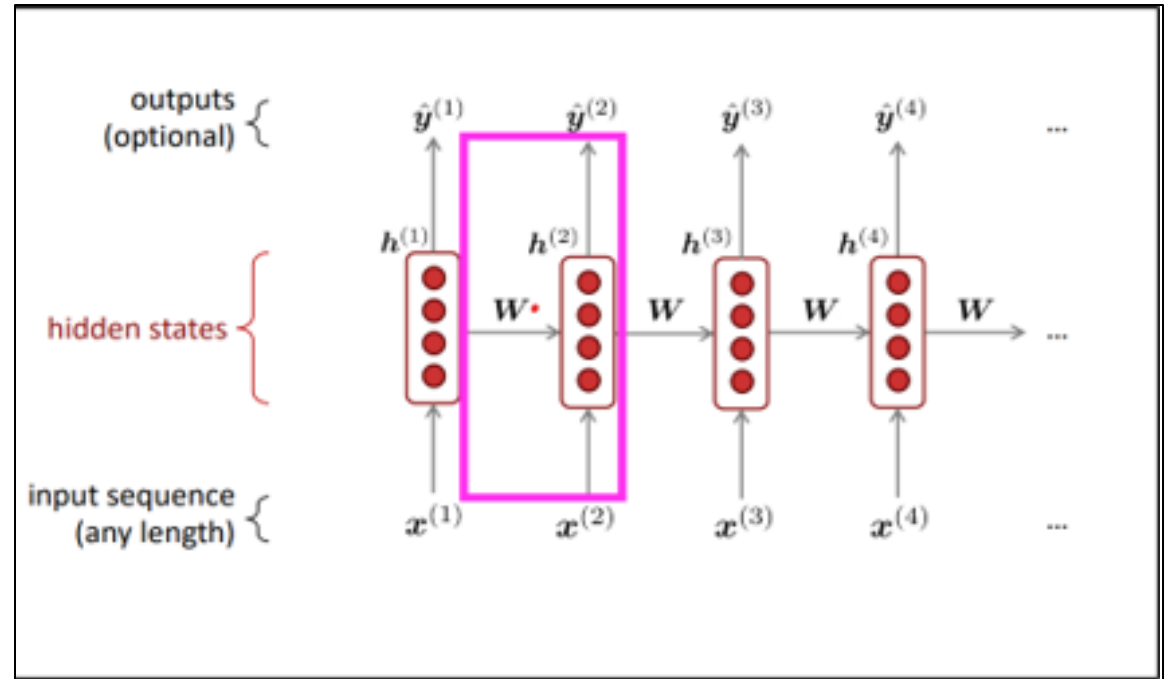


# Recurrent Neural Networks (RNN)

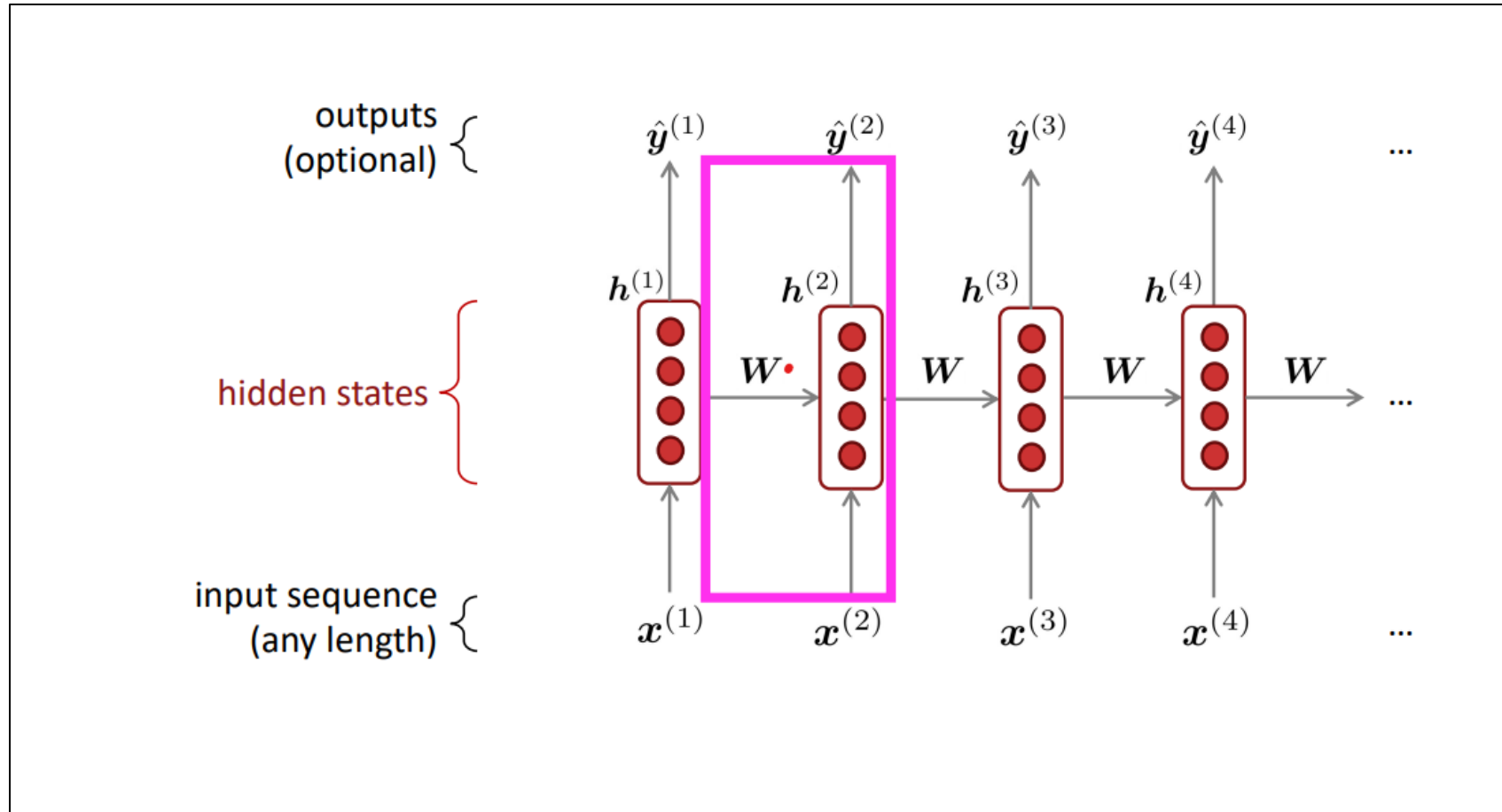


# Recurrent Neural Networks

- 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

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

hidden states

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

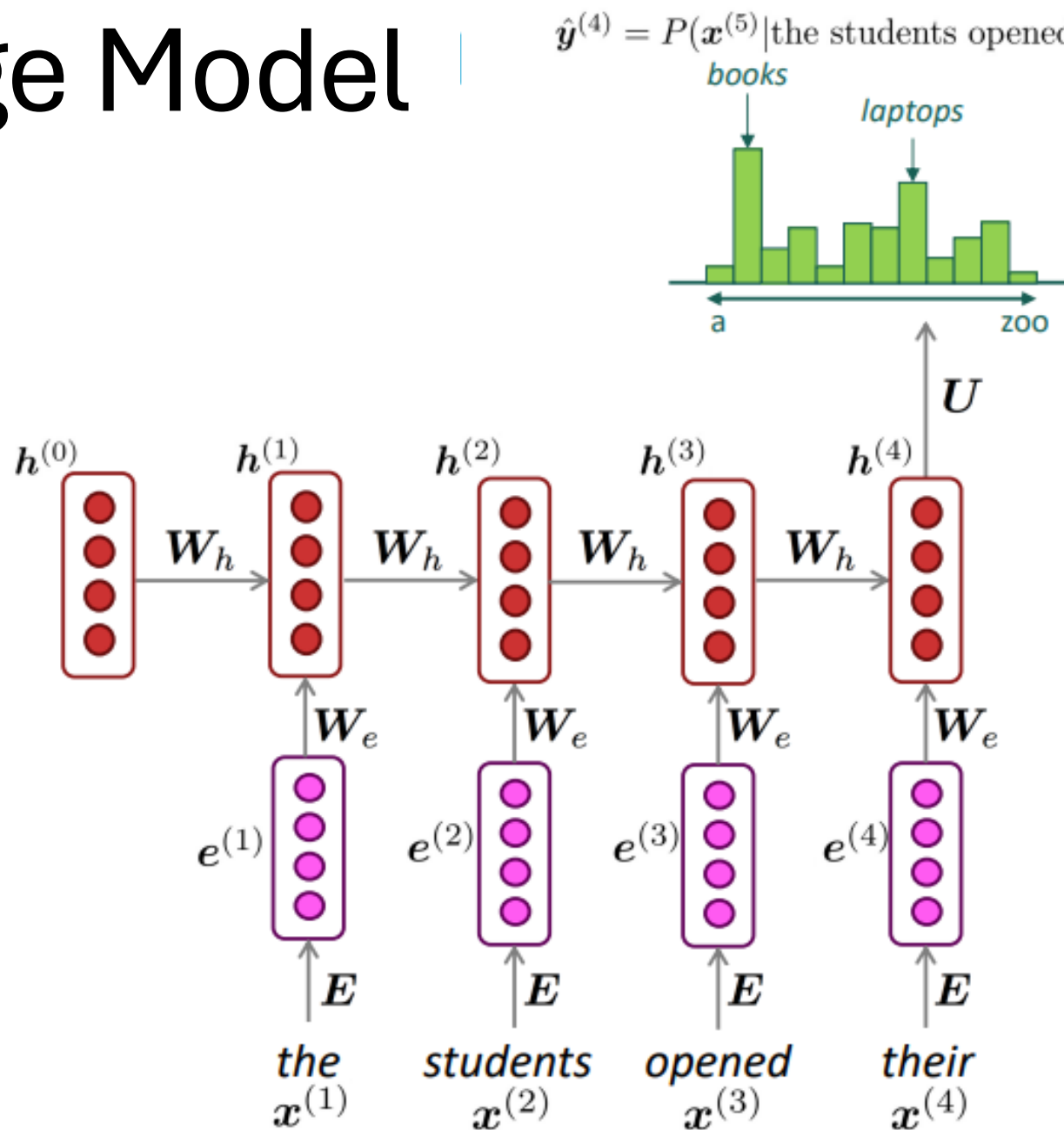
$h^{(0)}$  is the initial hidden state

word embeddings

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

words / one-hot vectors

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



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

# RNN Language Models

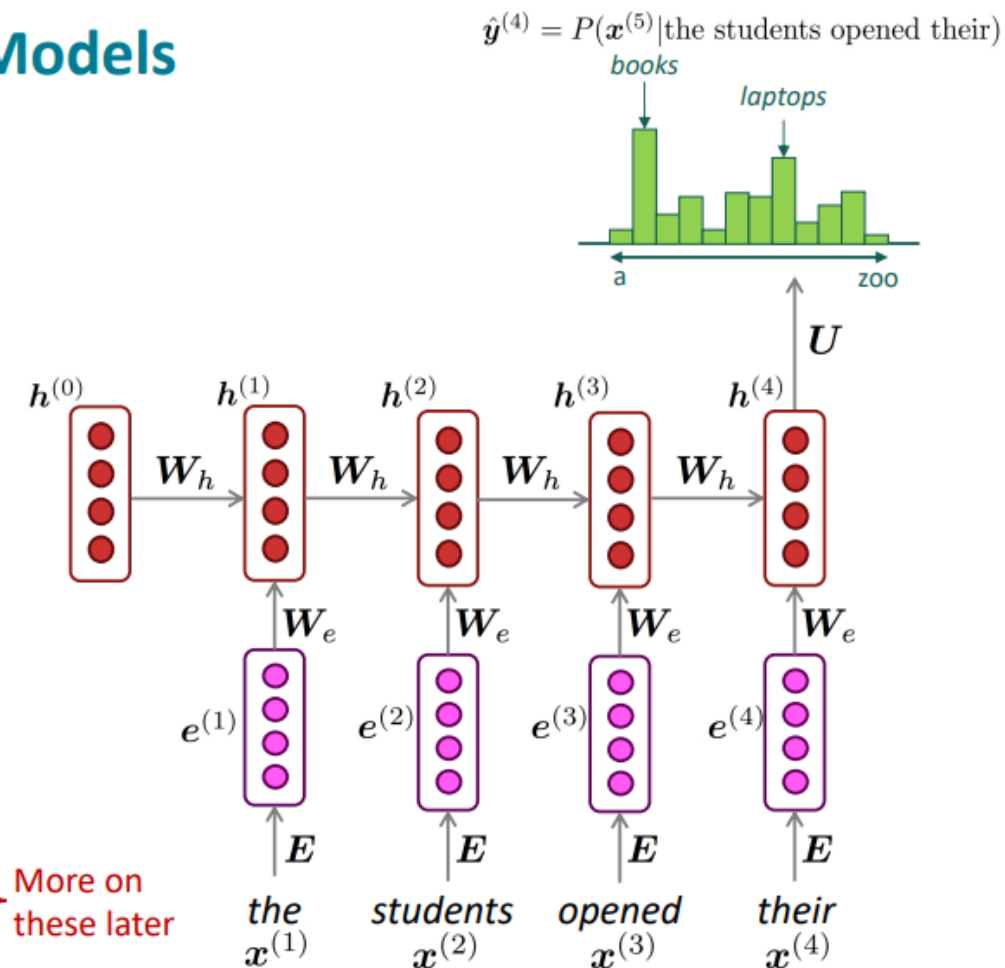
## RNN Language Models

### RNN Advantages:

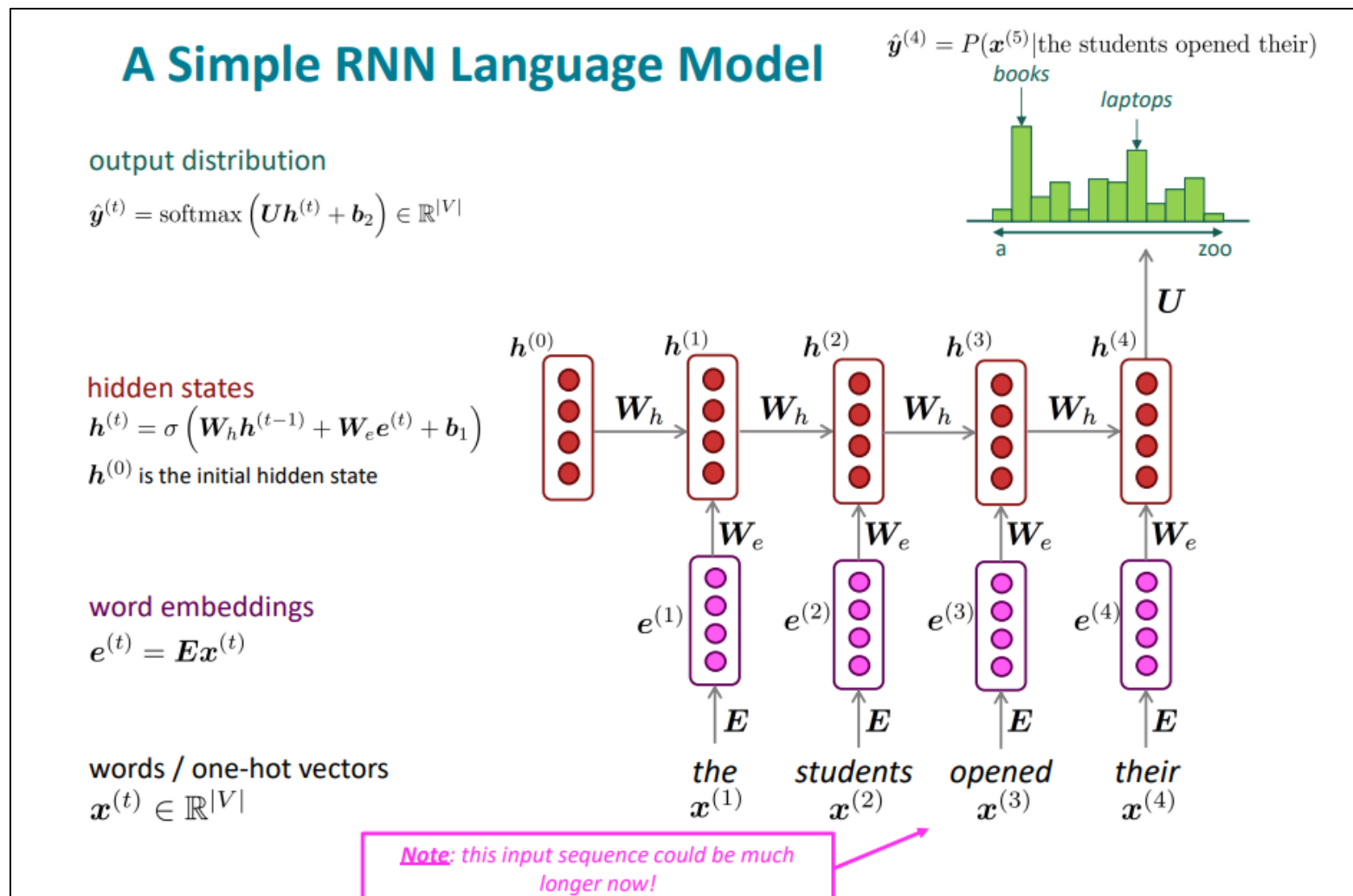
- Can process **any length** input
- Computation for step  $t$  can (in theory) use information from **many steps back**
- **Model size doesn't increase** for longer input context
- Same weights applied on every timestep, so there is **symmetry** in how inputs are processed.

### RNN Disadvantages:

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



# A simple RNN Language Model



# Training a RNN language model

Get a **big corpus of text** which is a sequence of words  $x^{(1)}, \dots, x^{(T)}$

Feed into RNN-LM; compute output distribution  $\hat{y}^{(t)}$  **for every step  $t$** .

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

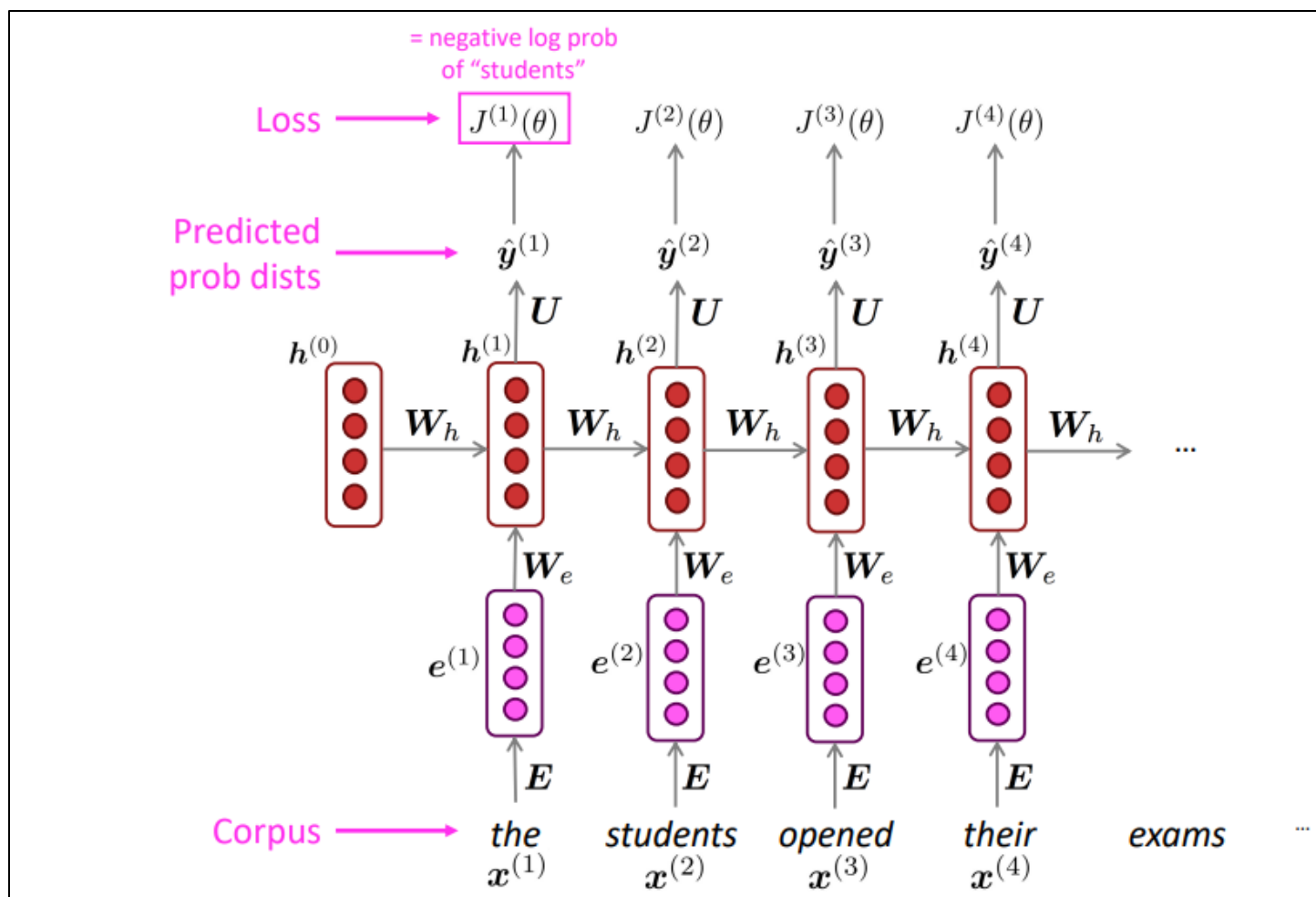
**Loss function** on step  $t$  is **cross-entropy** between predicted probability distribution  $\hat{y}^{(t)}$ , and the true next word  $y^{(t)}$  (one-hot for  $x^{(t+1)}$ ):

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

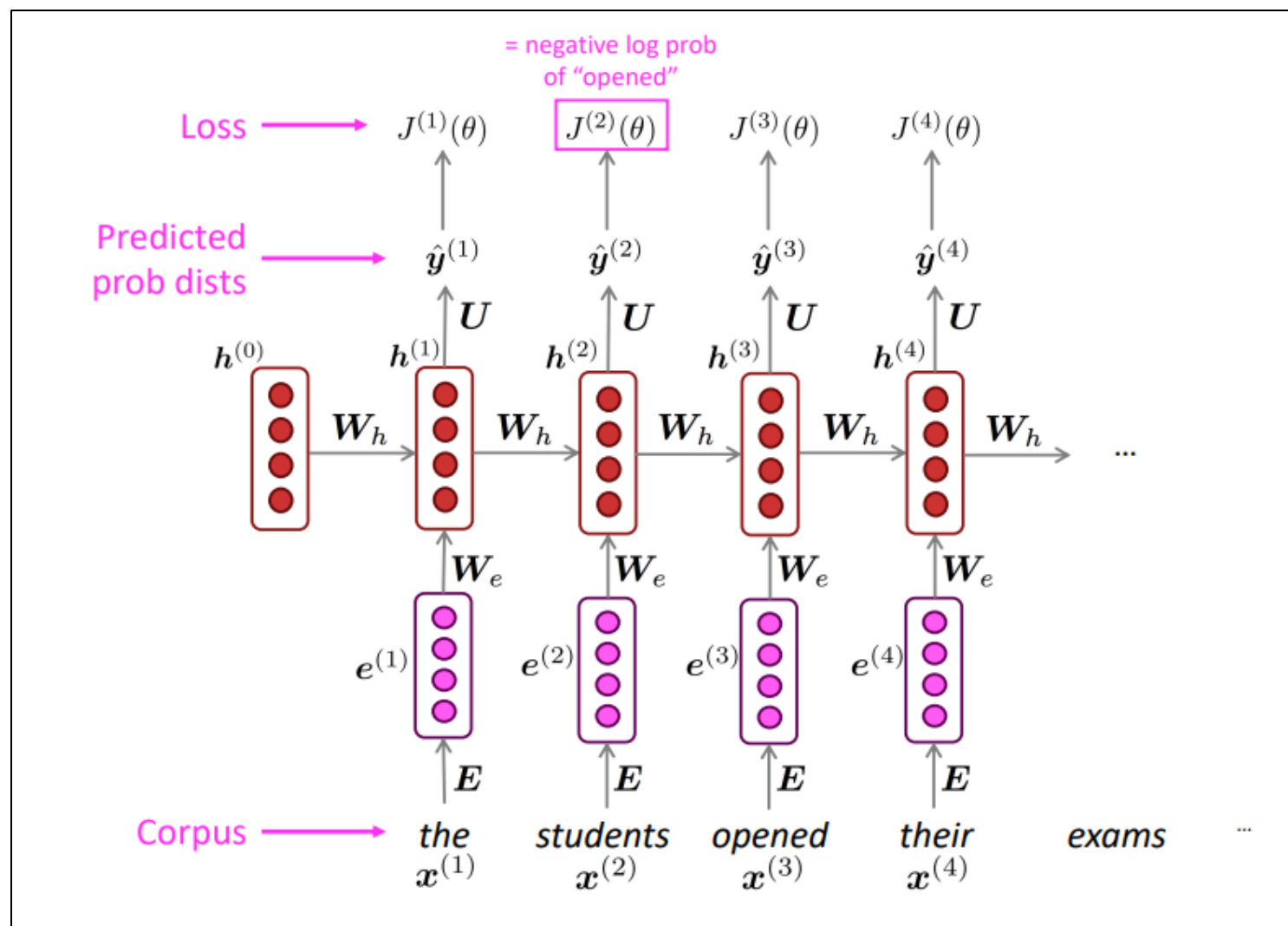
Average this to get **overall loss** for entire training set:

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

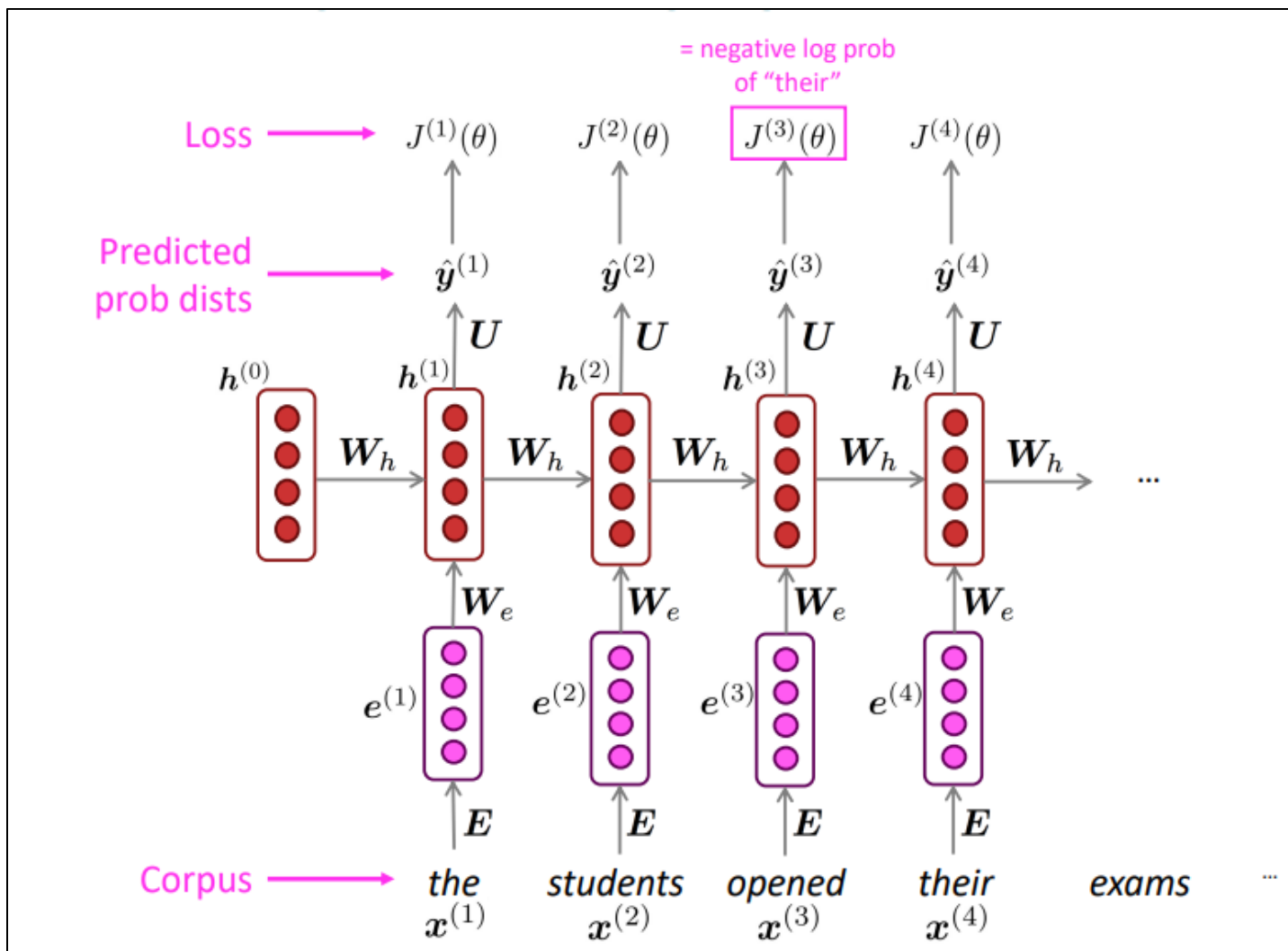
# Training a RNN language model



# Training a RNN language model

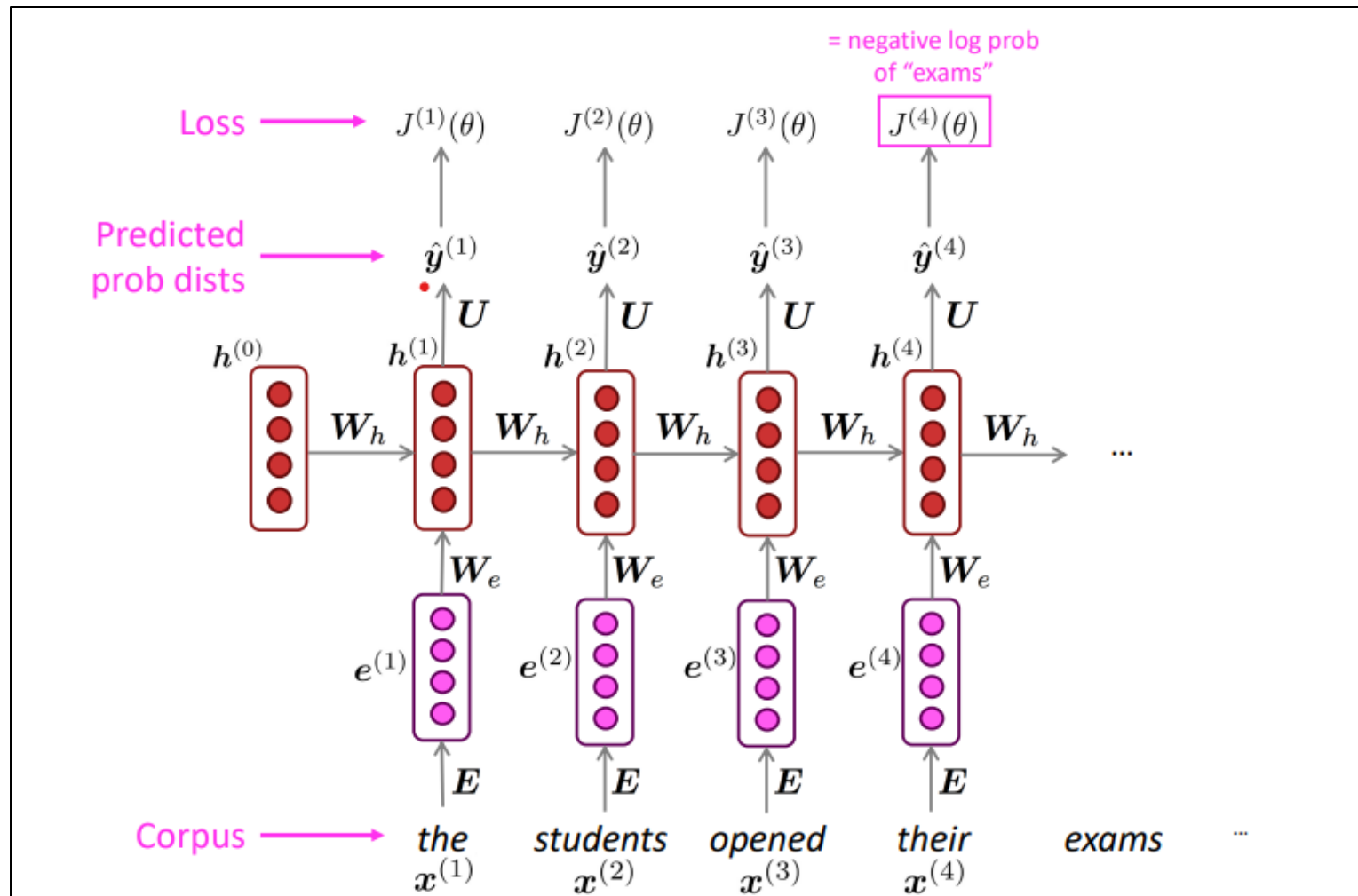


# Training a RNN language model

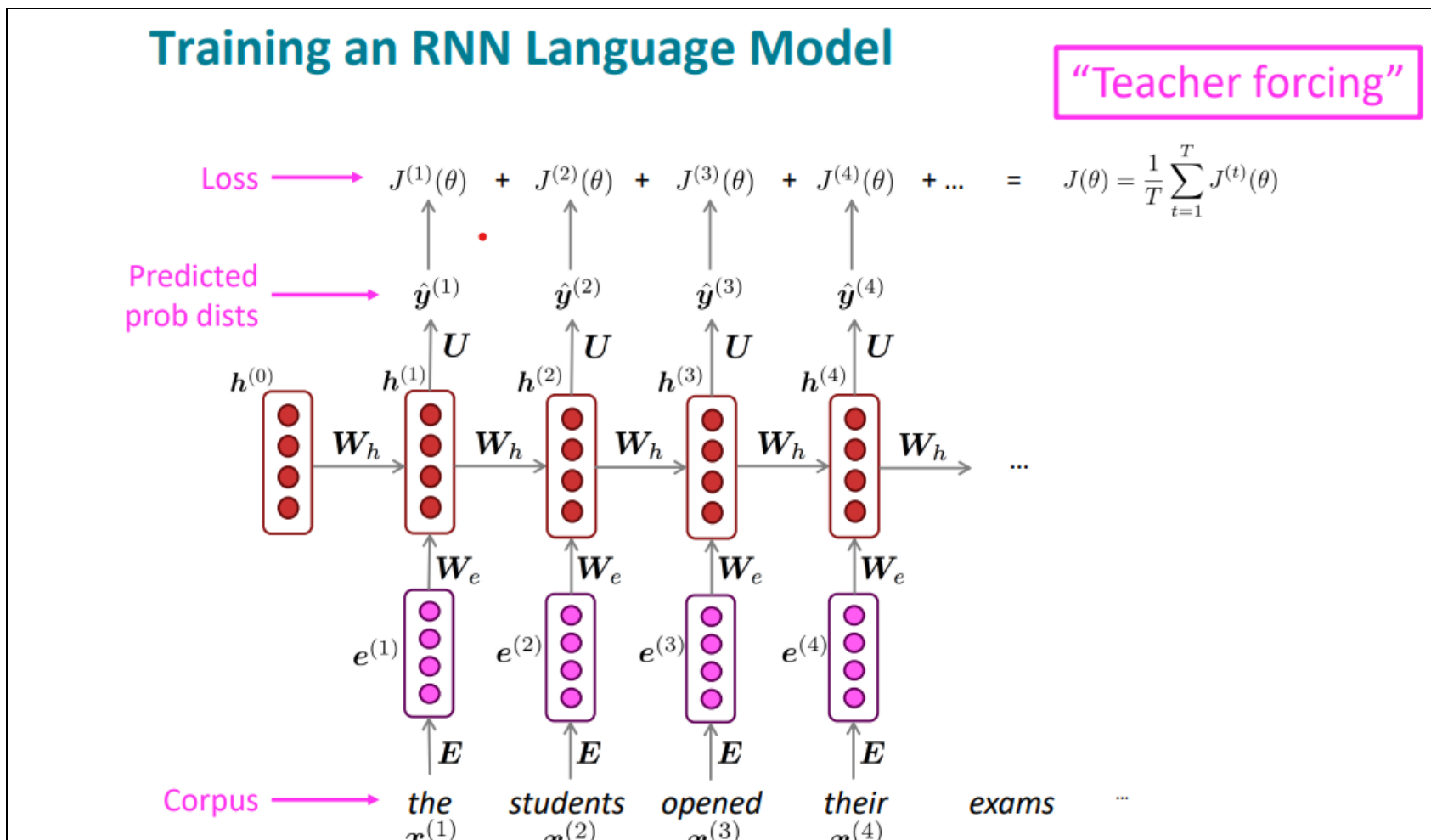




# Training a RNN language model



# Training a RNN language model



# Training a RNN language model

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

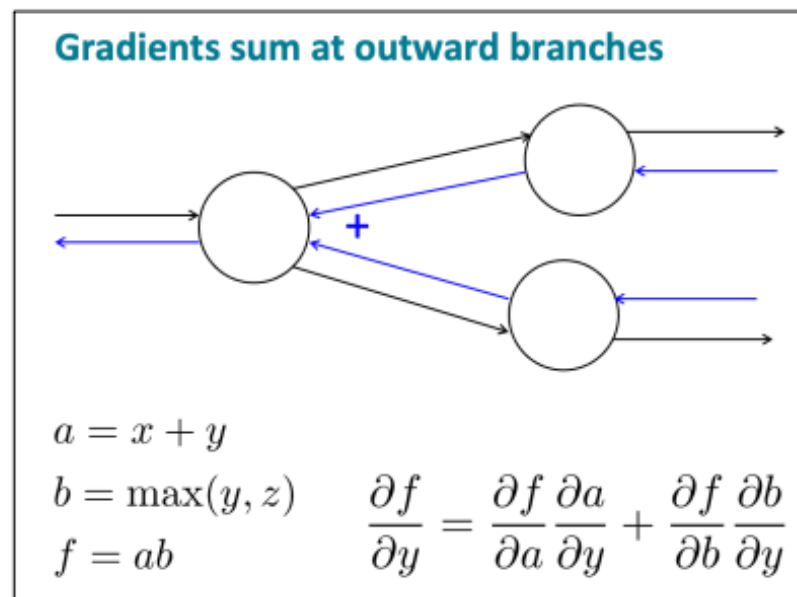
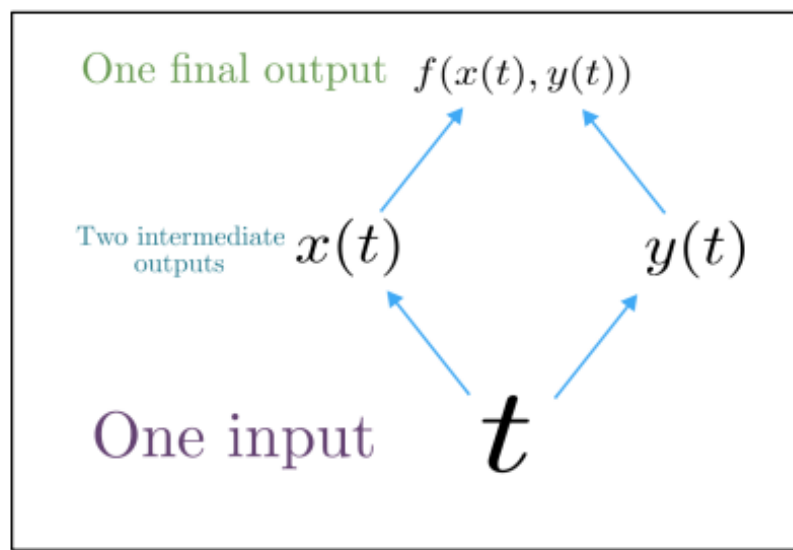
$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta)$$

- Consider as a sentence (or a document)
- Recall: Stochastic Gradient Descent allows us to compute  $J(\theta)$  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.

# 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:

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



# Issues with RNN: Vanishing and Exploding Gradients

1.5063v2 [cs.LG] 16 Feb 2013

---

## On the difficulty of training Recurrent Neural Networks

---

**Razvan Pascanu**  
Universite de Montreal

**Tomas Mikolov**  
Brno University

**Yoshua Bengio**  
Universite de Montreal

PASCANUR@IRO.UMONTREAL.CA

T.MIKOLOV@GMAIL.COM

YOSHUA.BENGIO@UMONTREAL.CA

### 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.

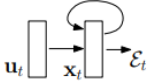


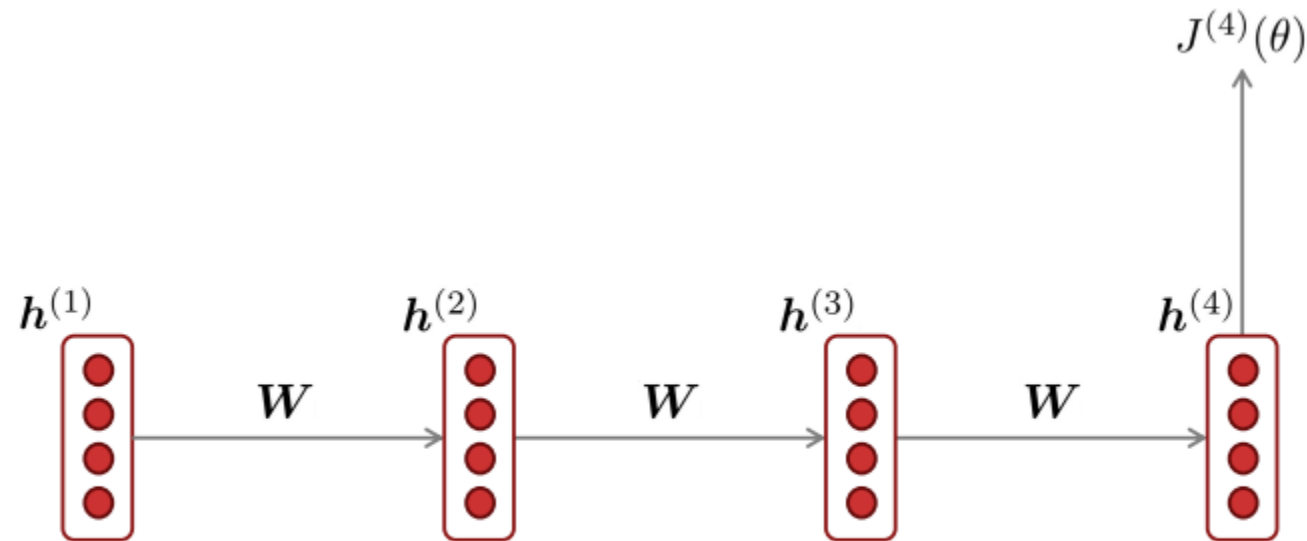
Figure 1. Schematic of a recurrent neural network. The recurrent connections in the hidden layer allow information to persist from one input to another.

### 1. Introduction

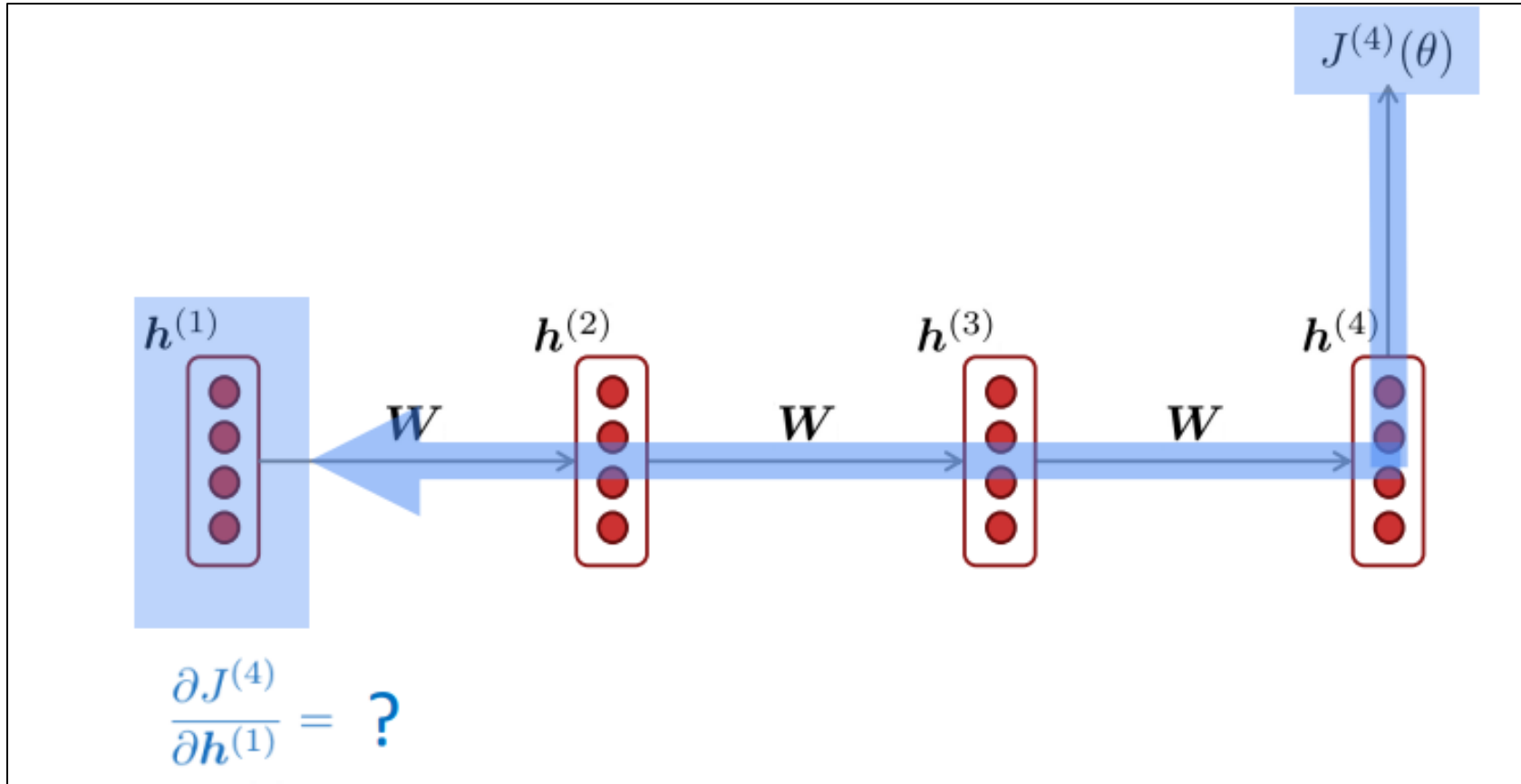
#### 1.1. Training recurrent networks

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) <sup>1</sup> in order to provide more precise conditions and intuitions about the everyday use-case.

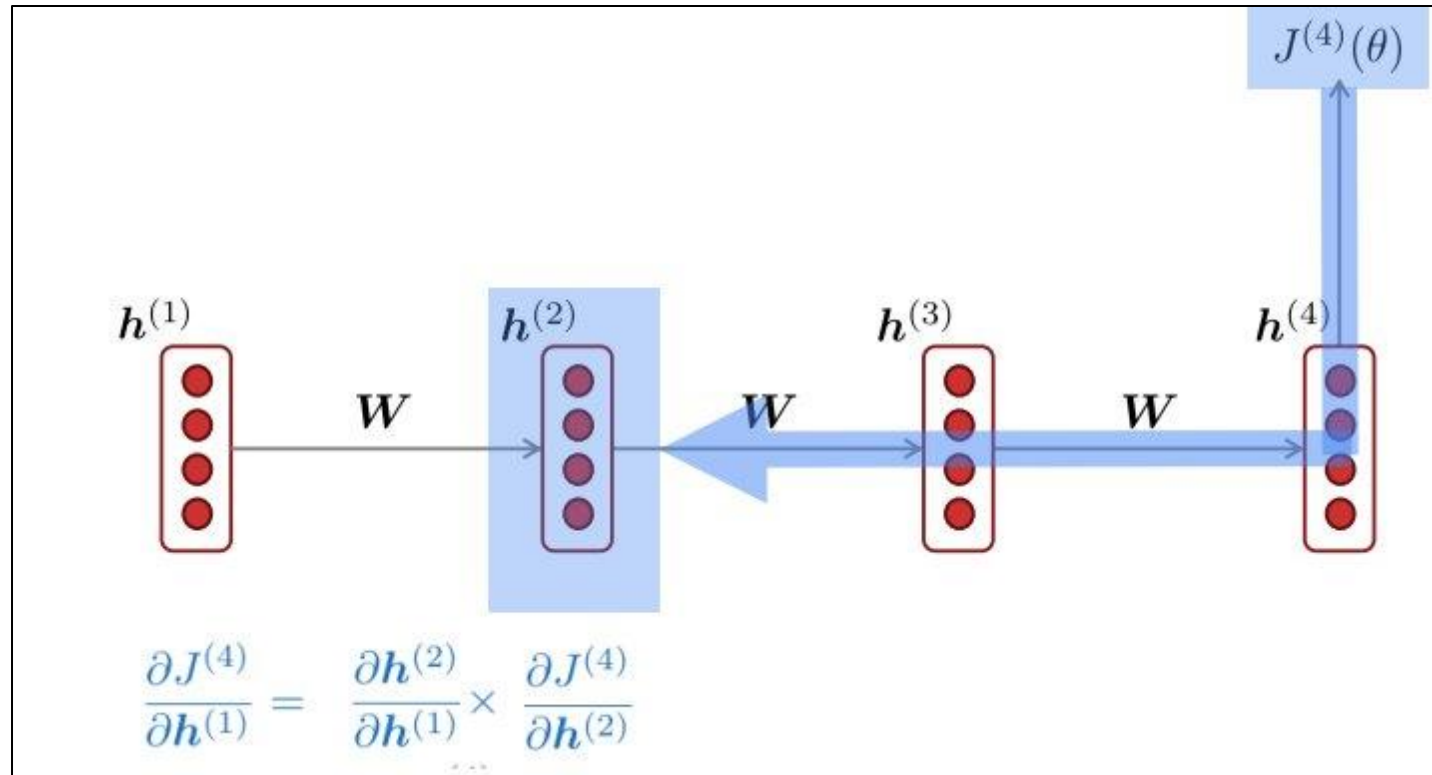
# Problems with RNNs: Vanishing and Exploding Gradients



# Vanishing Gradient Intuition



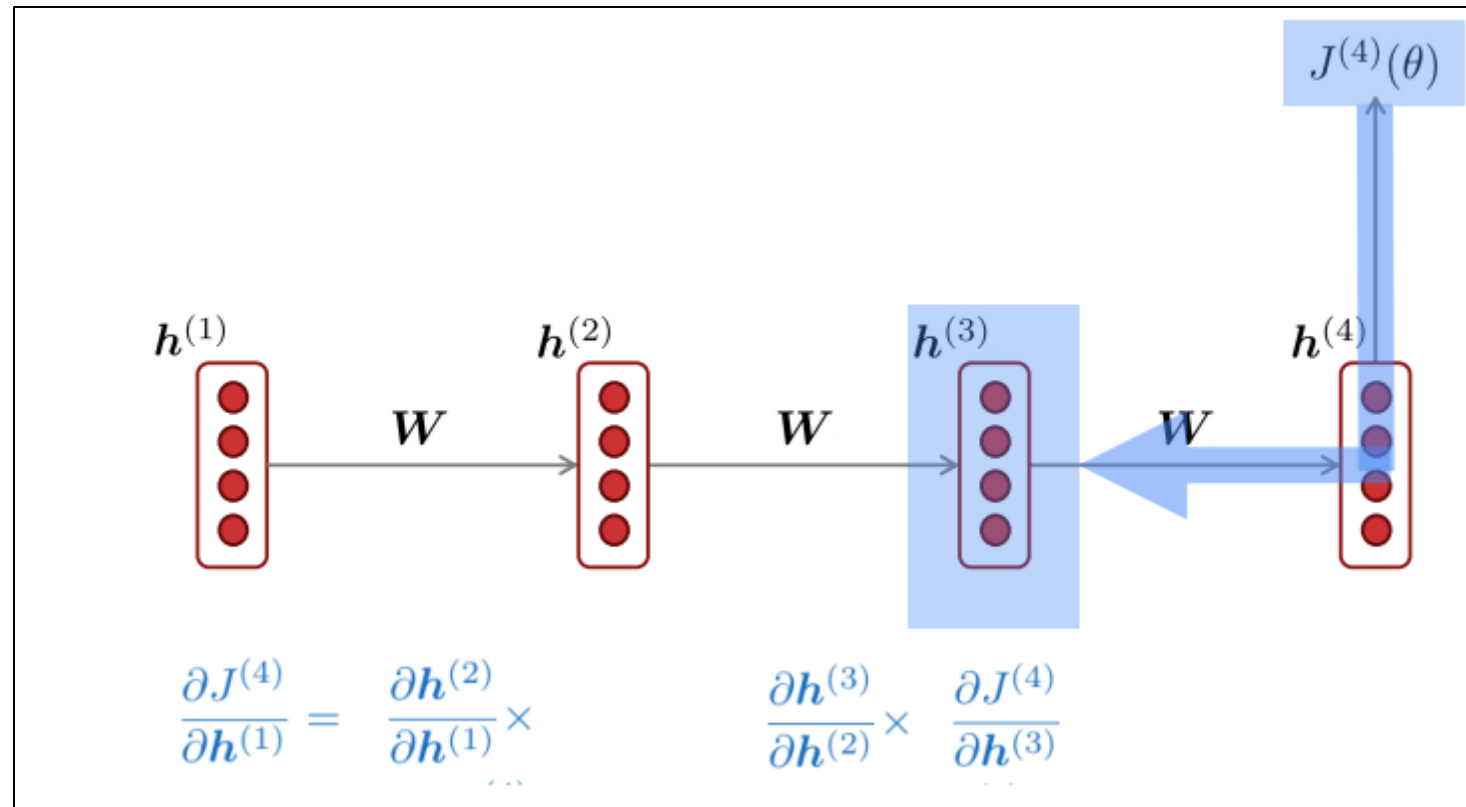
# Vanishing Gradient Intuition



**Chain Rule!!!!!!**

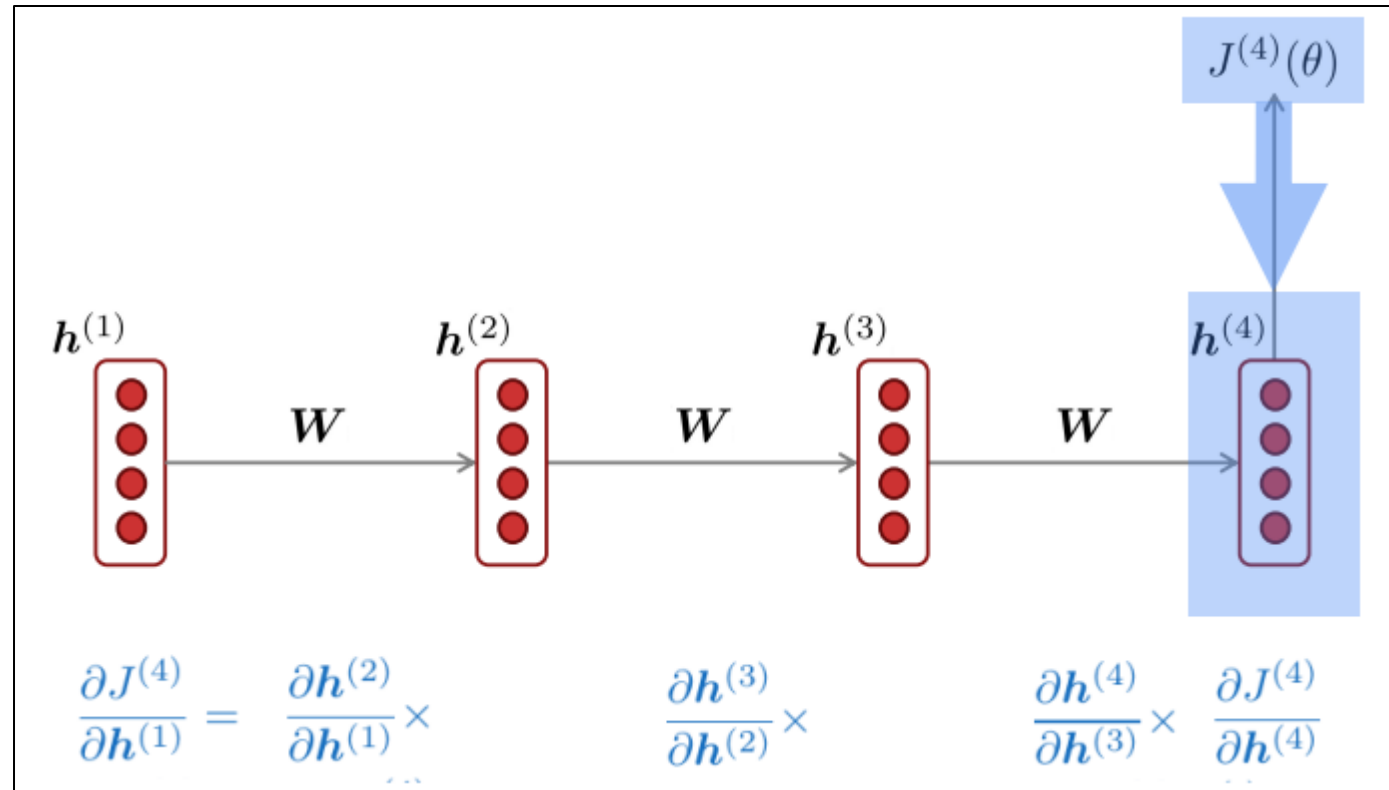


# Vanishing Gradient Intuition



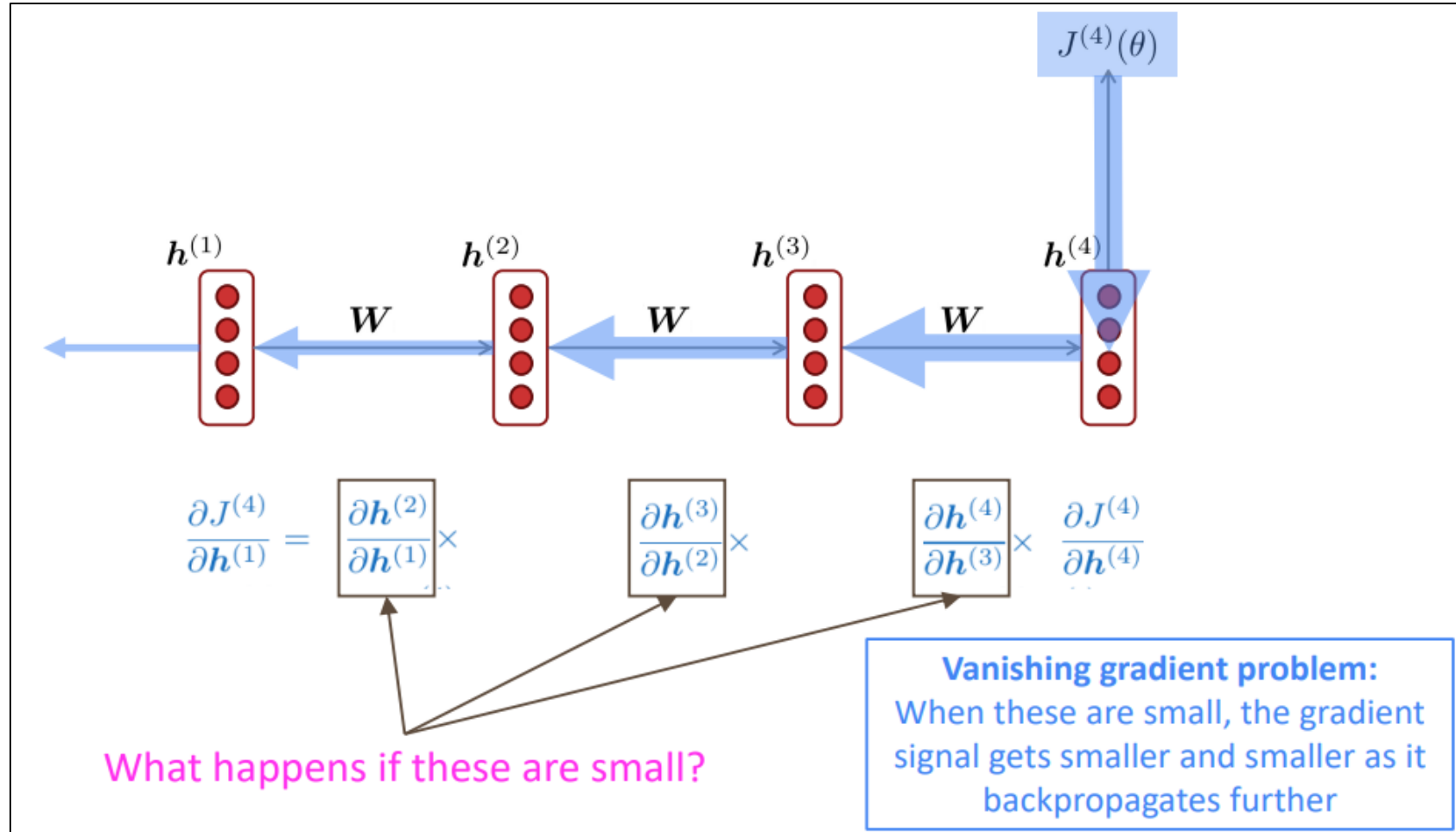
**Chain Rule!!!!!!**

# Vanishing Gradient Intuition



**Chain Rule!!!!!!**

# Vanishing Gradient Intuition



# Vanishing gradient proof sketch

Recall:

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

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

$$\begin{aligned} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} &= \text{diag} \left( \sigma' \left( \mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b}_1 \right) \right) \mathbf{W}_h && \text{(chain rule)} \\ &= \mathbf{I} \mathbf{W}_h = \mathbf{W}_h \end{aligned}$$

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

$$\begin{aligned} \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(j)}} &= \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \prod_{j < t \leq i} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} && \text{(chain rule)} \\ &= \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \prod_{j < t \leq i} \mathbf{W}_h = \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \boxed{\mathbf{W}_h^\ell} && \text{(value of } \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} \text{)} \end{aligned}$$

If  $\mathbf{W}_h$  is “small”, then this term gets exponentially problematic as  $\ell$  becomes large

# Vanishing gradient proof sketch

What's wrong with  $W_h^\ell$  ?

Consider if the eigenvalues of  $W_h$  are all less than 1:

$$\lambda_1, \lambda_2, \dots, \lambda_n < 1$$

$q_1, q_2, \dots, q_n$  (eigenvectors)

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

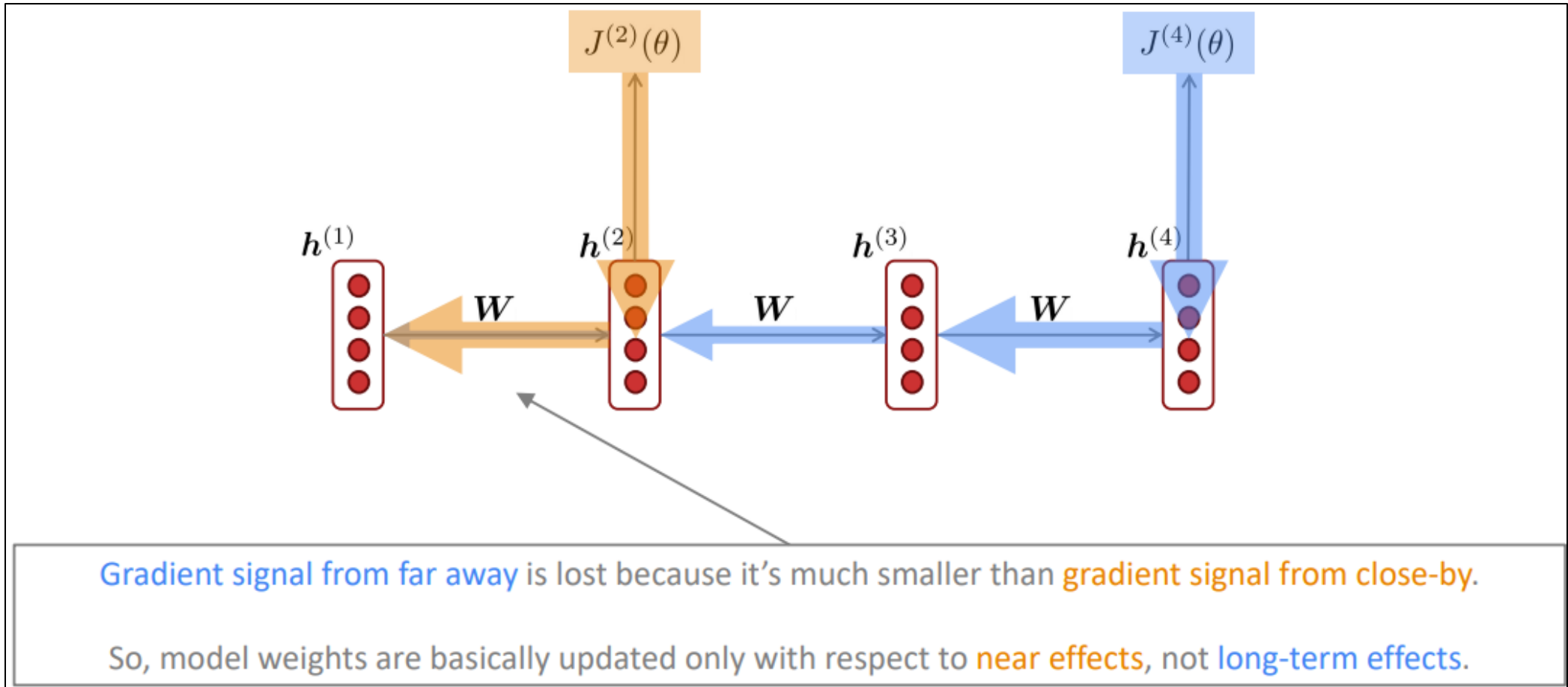
$$\frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} W_h^\ell = \sum_{i=1}^n c_i \lambda_i^\ell q_i \approx \mathbf{0} \text{ (for large } \ell \text{)}$$

Approaches 0 as  $\ell$  grows, so gradient vanishes

What about nonlinear activations  $\sigma$  (i.e., what we use?)

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

# Why is vanishing gradient a problem?



# 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]**

# 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

---

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

---

- **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



# 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)

# 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

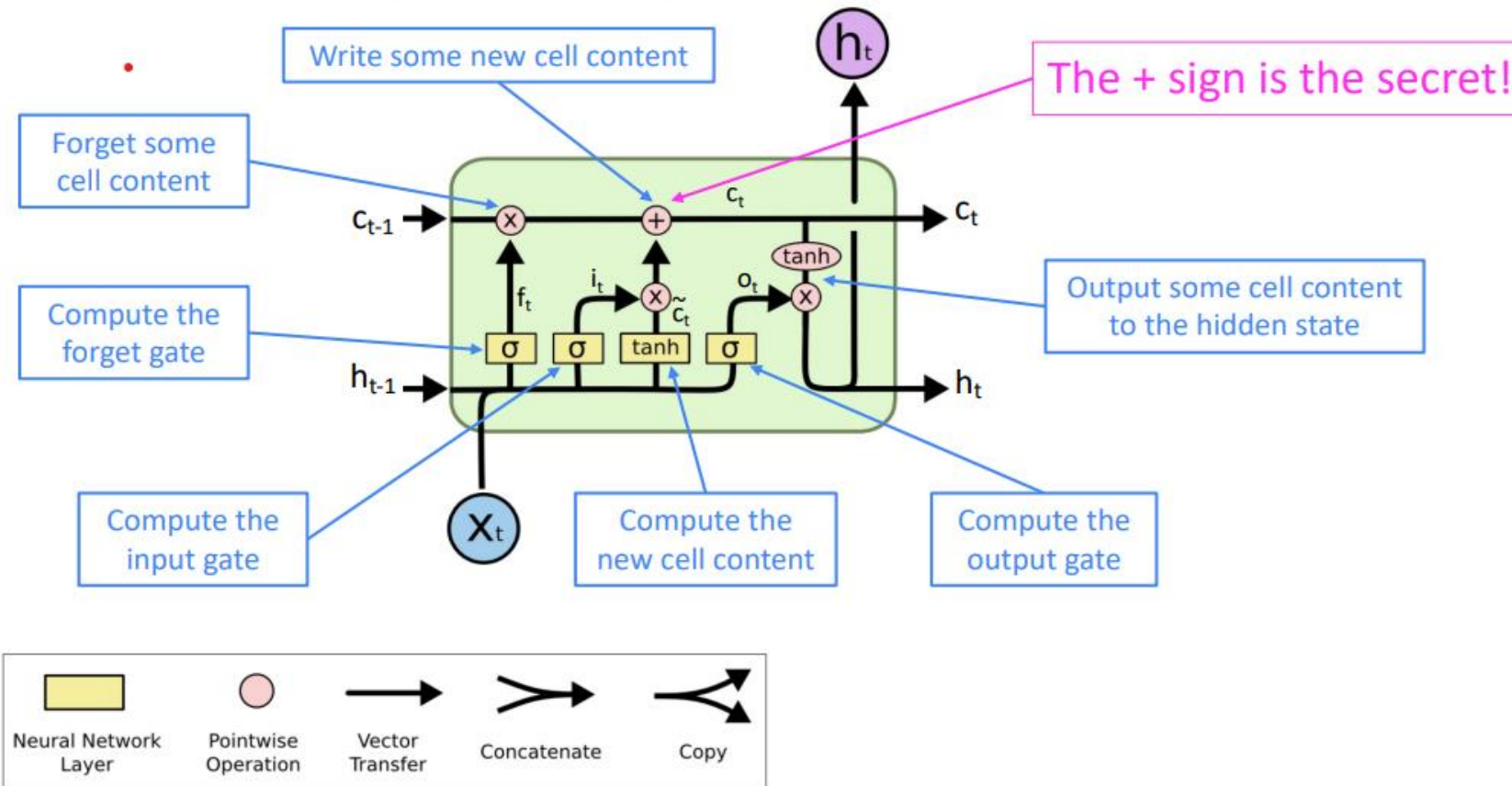
Learning to store information over extended time intervals via recurrent backpropagation takes a very long time, mostly due to insufficient, decaying error back flow. We briefly review Hochreiter's 1991 analysis of this problem, then address it by introducing a novel, efficient, gradient-based method 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 carousels" 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**

# LSTM Equations

You can think of the LSTM equations visually like this:



# LSTM detailed visualization

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

# History of Neural models in NLP

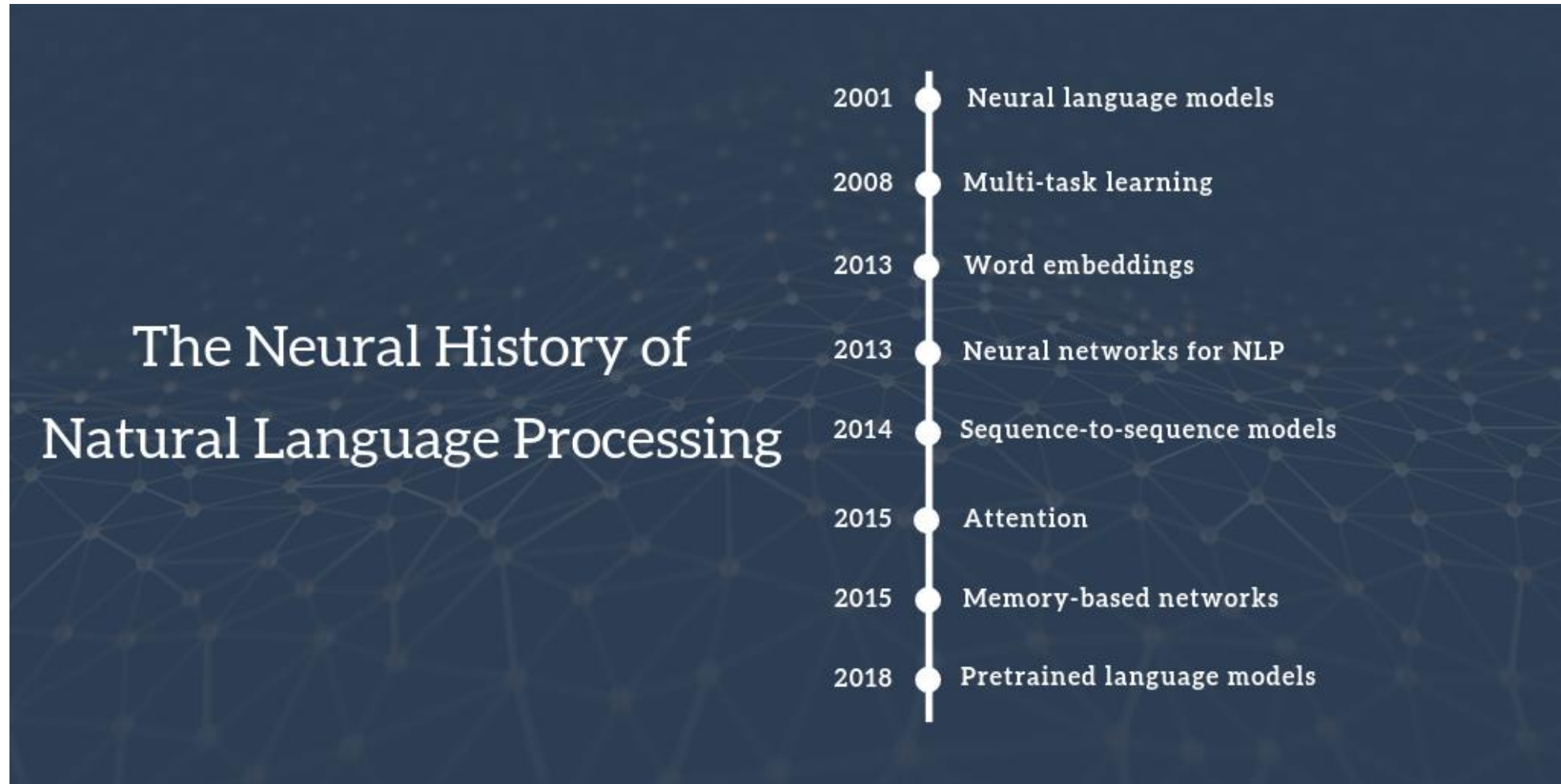


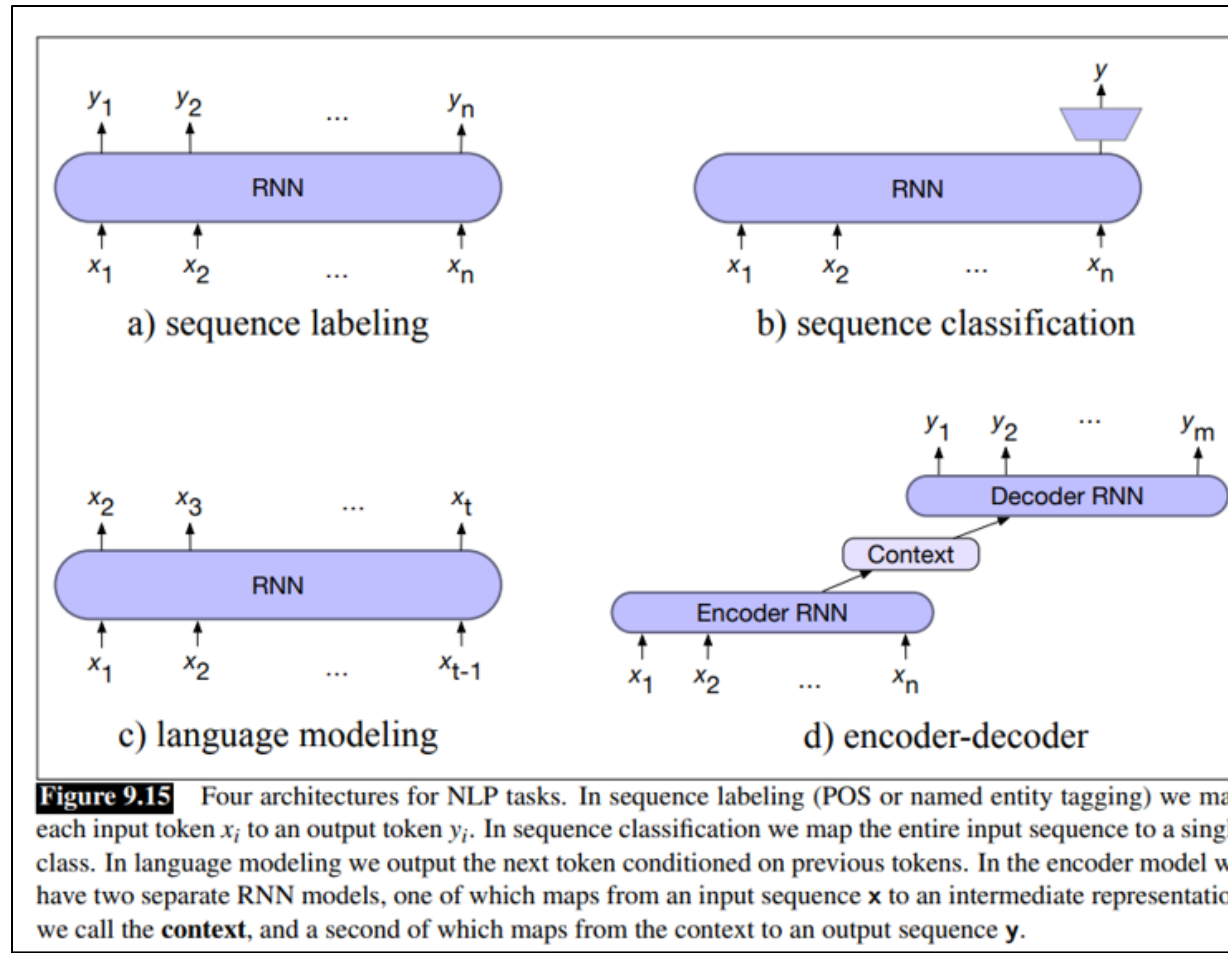
Image source : <https://www.ruder.io/a-review-of-the-recent-history-of-nlp/>

# Different variants of RNN

- Stacked RNN
- Bi-directional RNN
- Many more



# Sequence -to-Sequence learning



---

# Sequence to Sequence Learning with Neural Networks

---

Ilya Sutskever  
Google  
ilyasu@google.com

Oriol Vinyals  
Google  
vinyals@google.com

Quoc V. Le  
Google  
qvl@google.com

## Abstract

Deep Neural Networks (DNNs) are powerful models that have achieved excellent performance on difficult learning tasks. Although DNNs work well whenever large labeled training sets are available, they cannot be used to map sequences to sequences. In this paper, we present a general end-to-end approach to sequence learning that makes minimal assumptions on the sequence structure. Our method uses a multilayered Long Short-Term Memory (LSTM) to map the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector. Our main result is that on an English to French translation task from the WMT'14 dataset, the translations produced by the LSTM achieve a BLEU score of 34.8 on the entire test set, where the LSTM's BLEU score was penalized on out-of-vocabulary words. Additionally, the LSTM did not have difficulty on long sentences. For comparison, a phrase-based SMT system achieves a BLEU score of 33.3 on the same dataset. When we used the LSTM to rerank the 1000 hypotheses produced by the aforementioned SMT system, its BLEU score increases to 36.5, which is close to the previous best result on this task. The LSTM also learned sensible phrase and sentence representations that are sensitive to word order and are relatively invariant to the active and the pas-

# References

- [1] <https://www.cs.ubc.ca/~dsuth/440/23w2/slides/9-rnn.pdf>
- [2] [https://slazebni.cs.illinois.edu/spring17/lec02\\_rnn.pdf](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>
- [4] <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# 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

