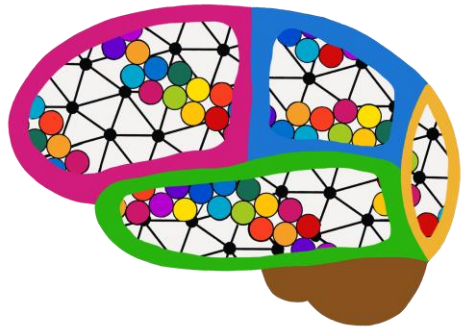
University of Northampton



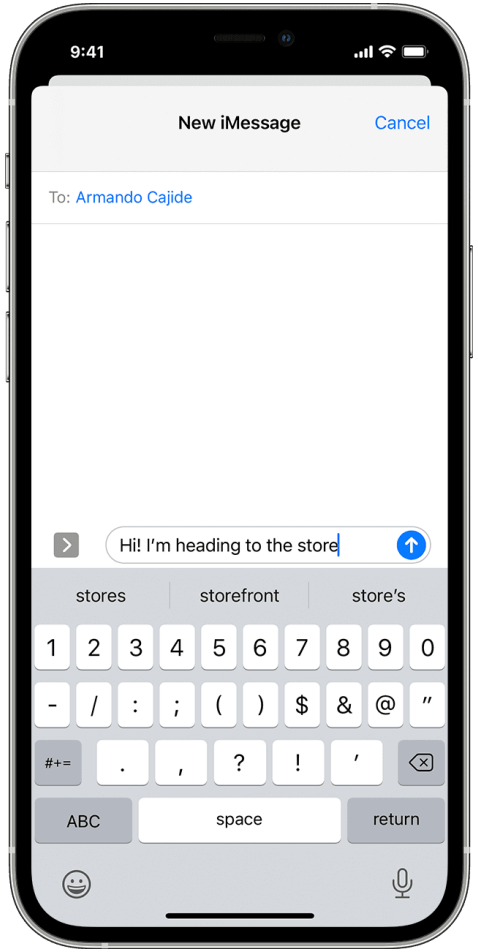
Week 6

Neural Language Models

CSY3055 – Natural Language Processing

Dr Oluseyi Oyedeji

When your phone finishes your sentences, or when ChatGPT writes a paragraph that makes sense.



how does it know what word comes next?

This lecture explores how **RNNs**, **LSTMs**, and **GRUs** handle sequences and generate text

University of Northampton

Learning Goals for Week 6

By the end of week six, you should be able to:

- Explain how neural networks can model language sequences.
- Describe the architecture of Recurrent Neural Networks (RNNs).
- Understand vanishing and exploding gradients.
- Explain how Long Short-Term Memory (LSTM) networks address these issues.
- Understand how text generation works using RNN-based models.

Why Study Neural Language Models?

Language models underpin:

- Predictive text and autocorrect.
- Speech recognition.
- Grammar and style correction.
- Machine translation.
- Chatbots and generative text (like GPT).

From Traditional to Neural LMs

- Traditional n-gram models:
  - Count word co-occurrences.
  - Limited by fixed window size.

- Neural models
  - Learn distributed representations.
  - Capture longer dependencies.
  - Generalise to unseen contexts.

Today's Session

- Introduction
- Neural Networks
- Recurrent Neural Network (RNN)
- Long Short-Term Memory
- Gated Recurrent Unit (GRU)
- Practical Session

FEEDFORWARD NEURAL NETWORKS

What is Neural Network?

- A **neural network** is a type of **machine learning model** designed to recognize patterns and make decisions in a way that loosely mimics how the human brain works.
- Consists of layers of “neurons” that transform input signals.

$$y = f(Wx + b)$$

- Learns patterns from data via weight adjustments.

Basic Structure of Neural Networks

- Single Neuron Equation

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

- In vector form

$$z = \mathbf{w}^\top \mathbf{x} + b$$

$$a = f(z)$$

Where: $\mathbf{x} = [\mathbf{x_1}, \mathbf{x_2}, \ldots \mathbf{x_n}]$ **are the inputs**

$\mathbf{w} = [\mathbf{w_1}, \mathbf{w_2}, \ldots \mathbf{w_n}]$ **are the weights**

b = bias

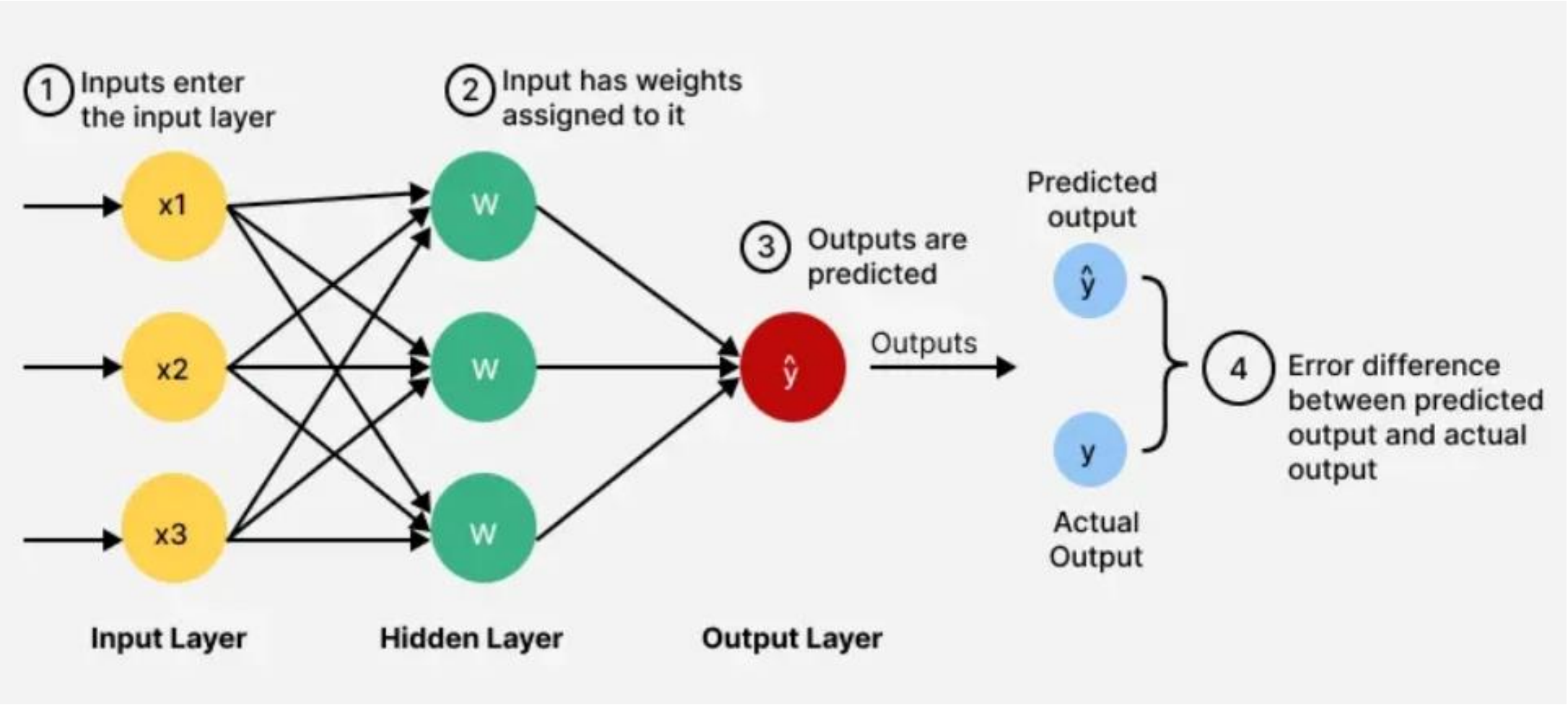
$\mathbf{f(\cdot)}$ **introduces non-linearity (so the network can model complex functions)**

Basic Structure of Neural Networks

Layer	Purpose
Input layer	Takes in the raw data (e.g., pixels from an image, words from text, numbers from a dataset).
Hidden layers	Perform mathematical transformations to extract patterns or relationships in the data.
Output layer	Produces the final prediction or classification (e.g., “cat” vs “dog,” or a numerical value).

- Each connection between neurons has a **weight**, and each neuron has a **bias**. Together they determine how much influence one neuron’s output has on another.

Structure of Neural Networks



basic feedforward diagram — input → hidden → output

How the Network Learns

- Each weight starts as a small random number.
- We show the model examples and measure how wrong it is using a **loss function** (often *cross-entropy loss*).
- Then we update weights using **backpropagation**, a clever use of the **chain rule** from calculus.
    1. Forward pass → make a prediction
    2. Compare to correct answer → compute loss
    3. Backward pass → adjust weights slightly
- After thousands of small corrections, the network "figures out" good weights.

Limitations of Feedforward Networks

- Feed-forward networks see each input separately, they forget what came before.
  - But language is **sequential**
- Feedforward networks:
  - Handle fixed-size inputs only.
  - Forget previous inputs , **no memory**.
  - Poor for sequential data (text, audio, etc.).

"I bought a loaf of ___" → depends on previous words

Even Word2Vec is only able to create word similarities, there is no contextual memory

RECURRENT NEURAL NETWORKS (RNNs)

Intuition: Adding Memory

- An **RNN** introduces a hidden state $\boldsymbol{h_t}$ that carries information across timesteps.

$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

$$y_t = g(W_{hy}h_t + b_y)$$

- Each step processes the **current input** *and* the **past hidden state**, a form of learned memory.

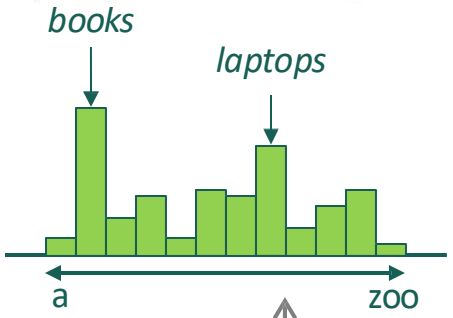
Simple RNN Language Model

- **If Goal**: predict the next word given previous ones.
  - Input: sequence of embeddings ($\boldsymbol{x_1, x_2, \ldots\ldots x_t}$)
  - Output: probability distribution for next word

$$P(w_t|w_1, \ldots, w_{t-1}) = \text{softmax}(W_{hy}h_t)$$

The 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

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

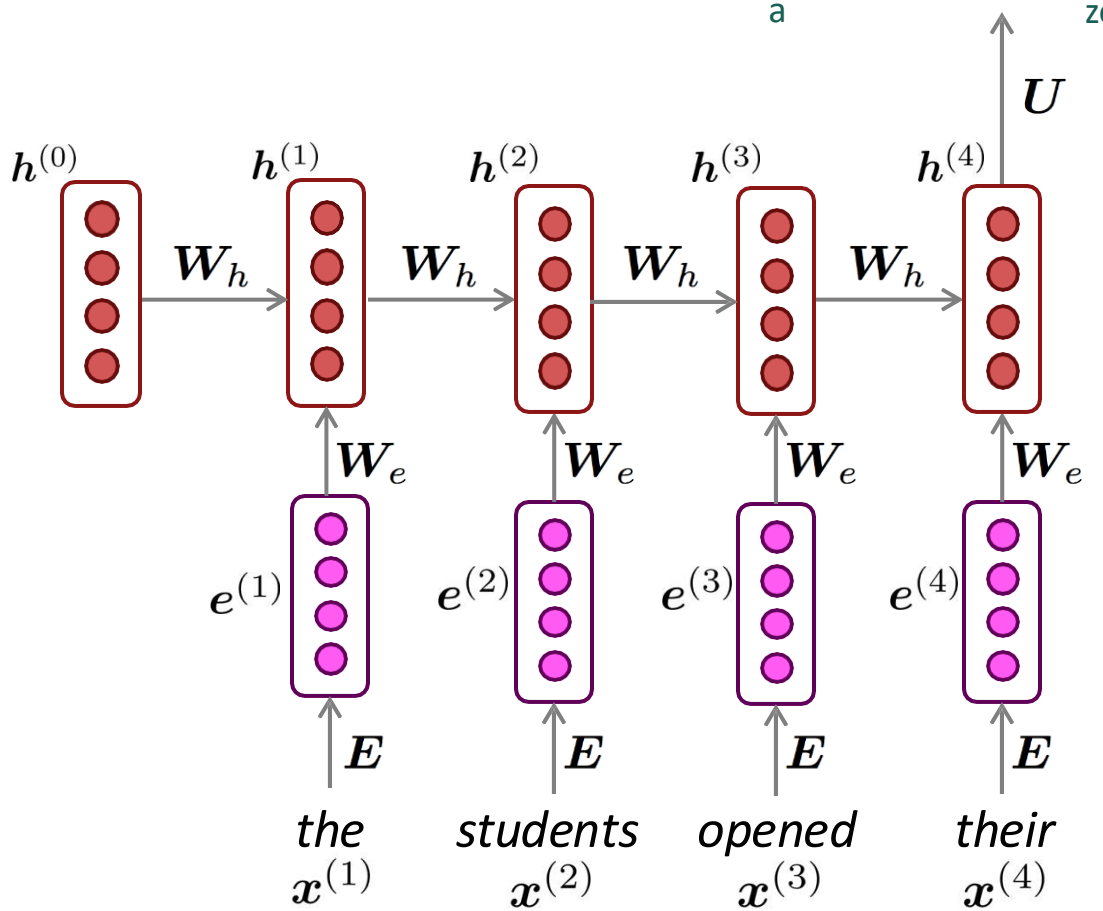
$\boldsymbol{h}^{(0)}$ is the initial hidden state

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!*

Training an RNN Language Model

- Get a **big corpus of text** which is a sequence of words

$$\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$$

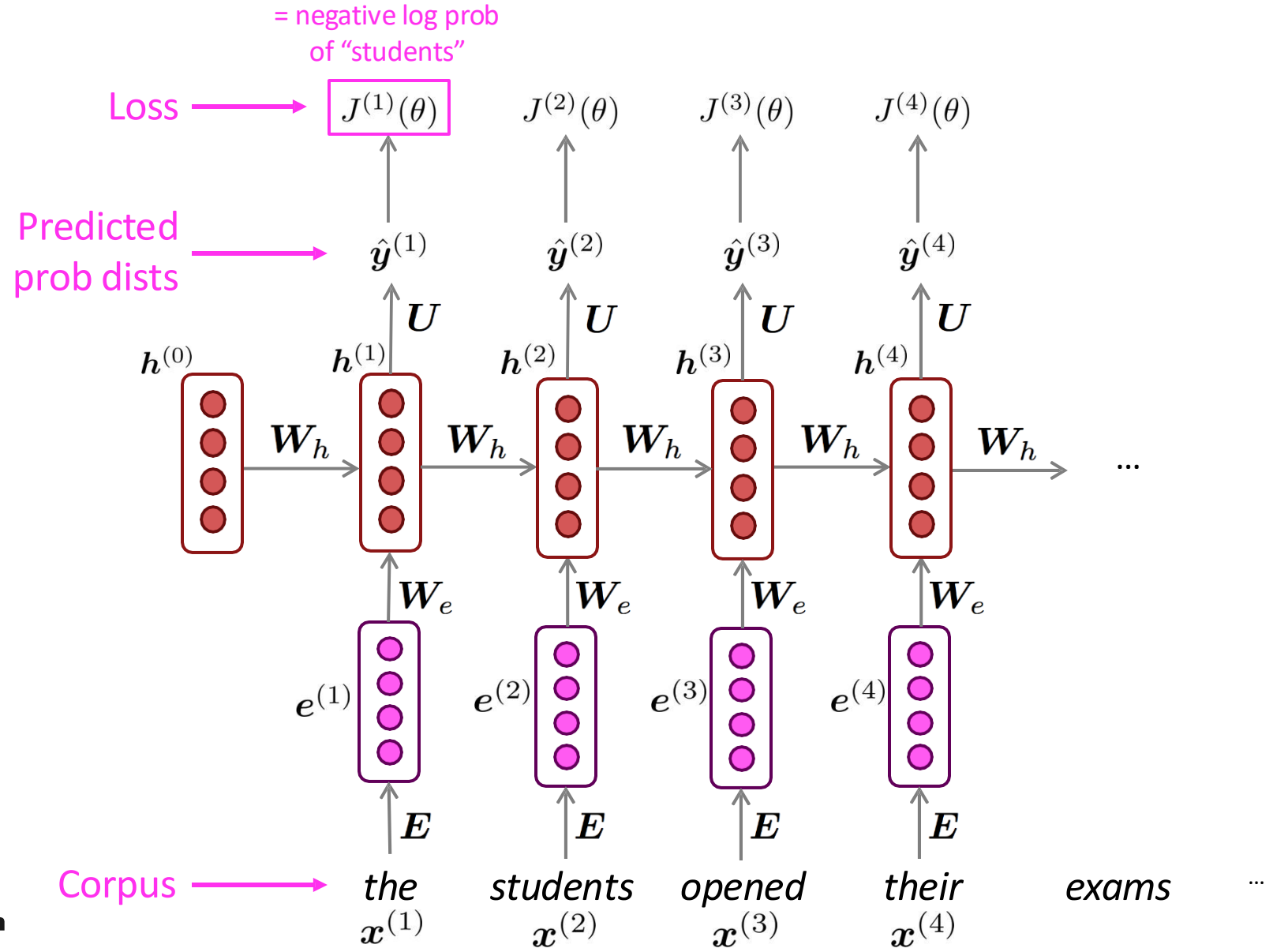
- Feed into RNN-LM; compute output distribution $\boldsymbol{y}^{(t)}$ for *every step t*.
  - i.e., predict probability dist of *every word*, given words so far
  - Loss function on step *t* is cross-entropy between predicted probability distribution, $\hat{\boldsymbol{y}}^{(t)}$ and the true next word (one-hot for $\boldsymbol{y}^{(t)}$ $\boldsymbol{x}^{(t+1)}$)

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

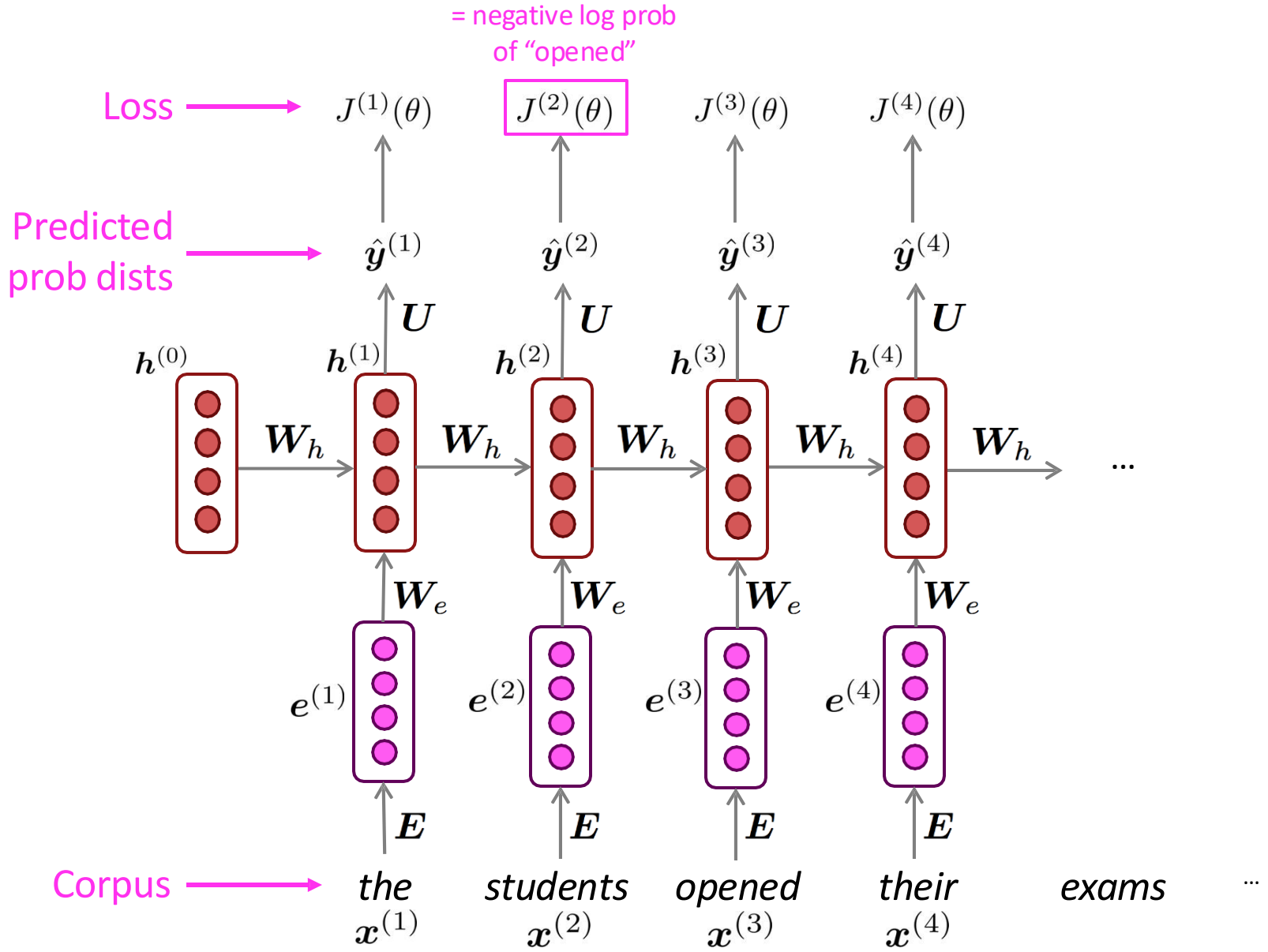
  - Average this to get **overall loss** for entire training set:

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

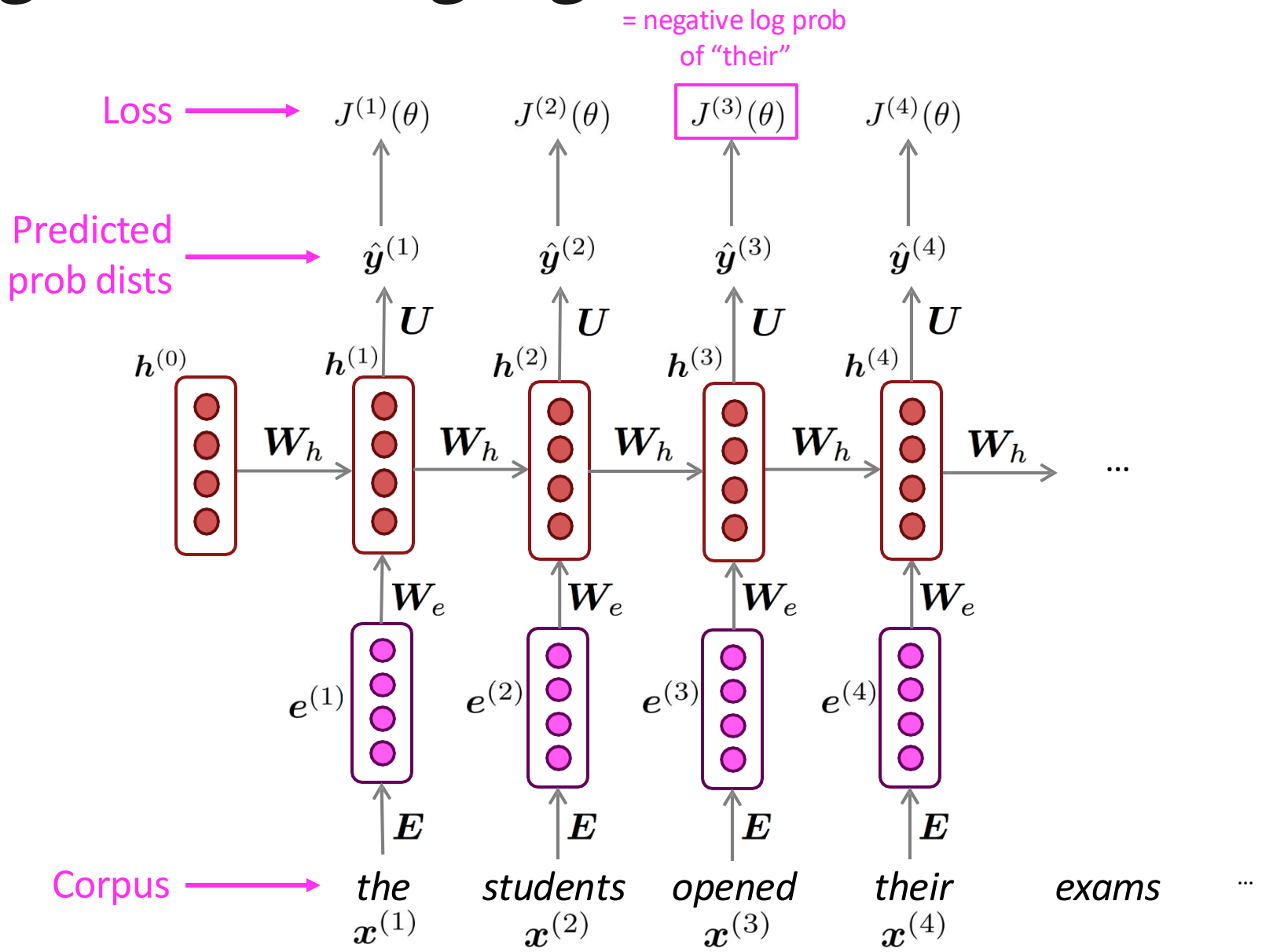
Training an RNN Language Model



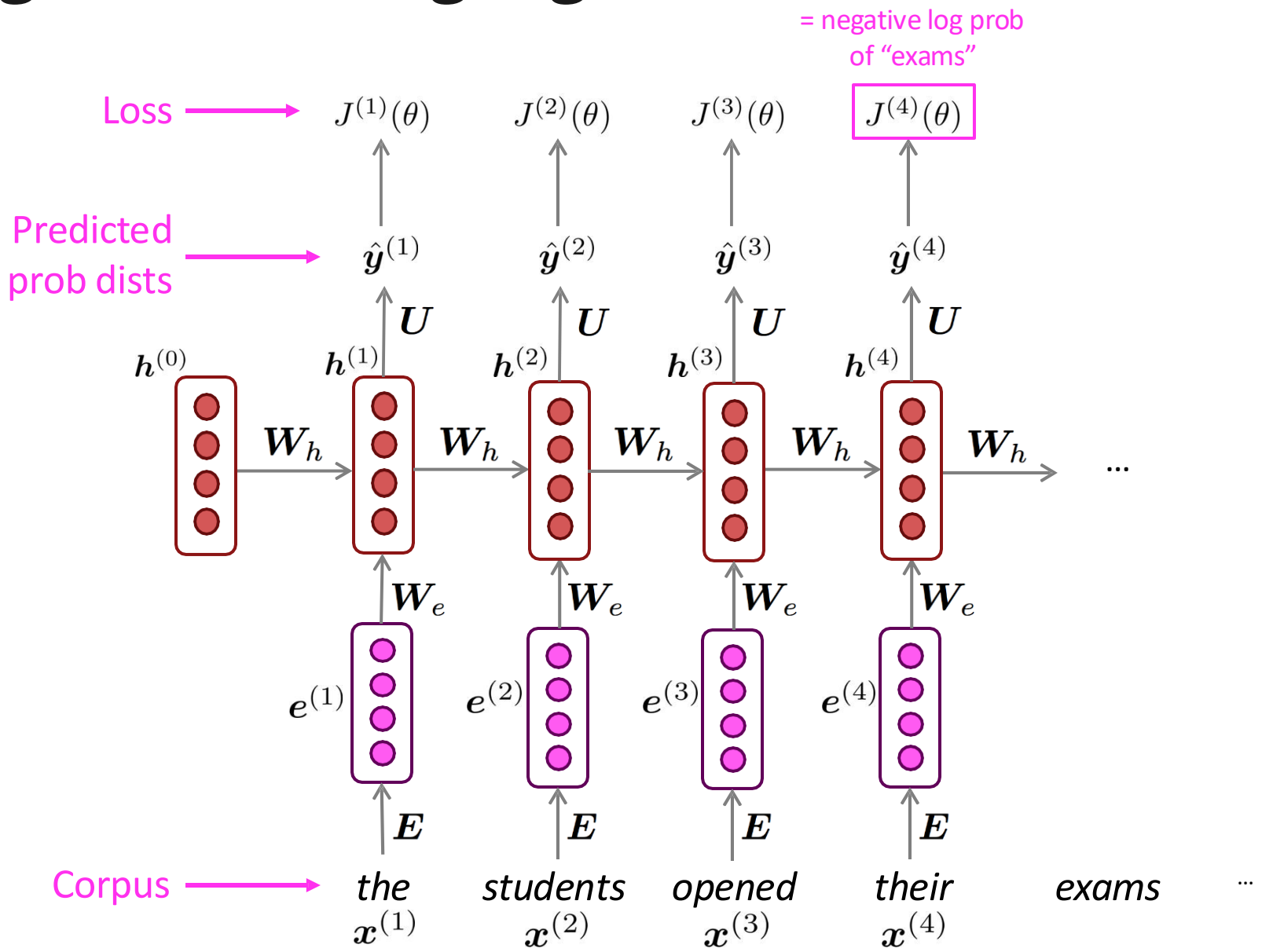
Training an RNN Language Model



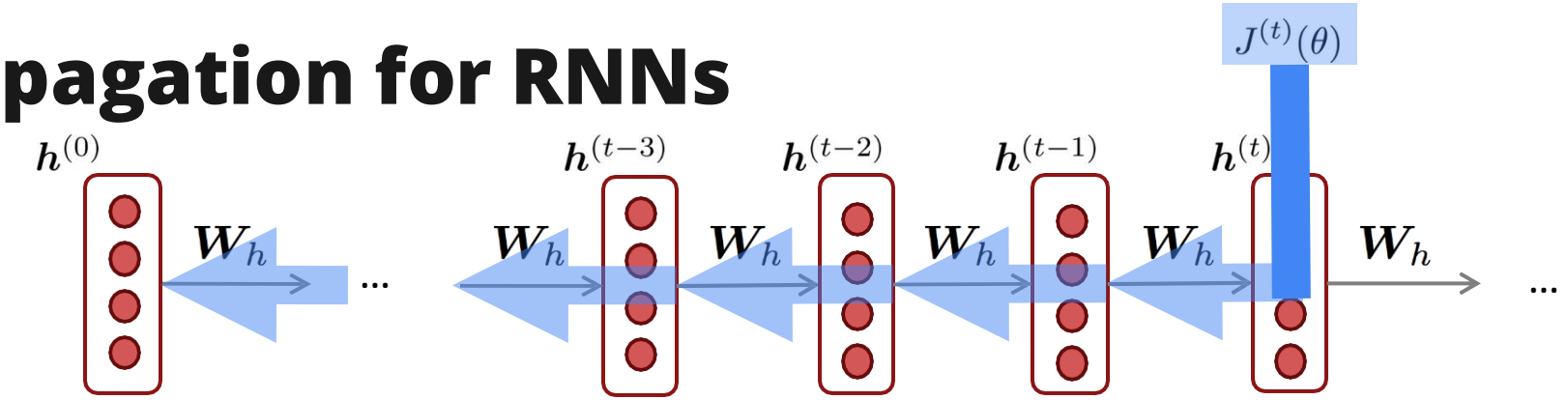
Training an RNN Language Model



Training an RNN Language Model



Backpropagation for RNNs



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

Question: How do we calculate this?

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

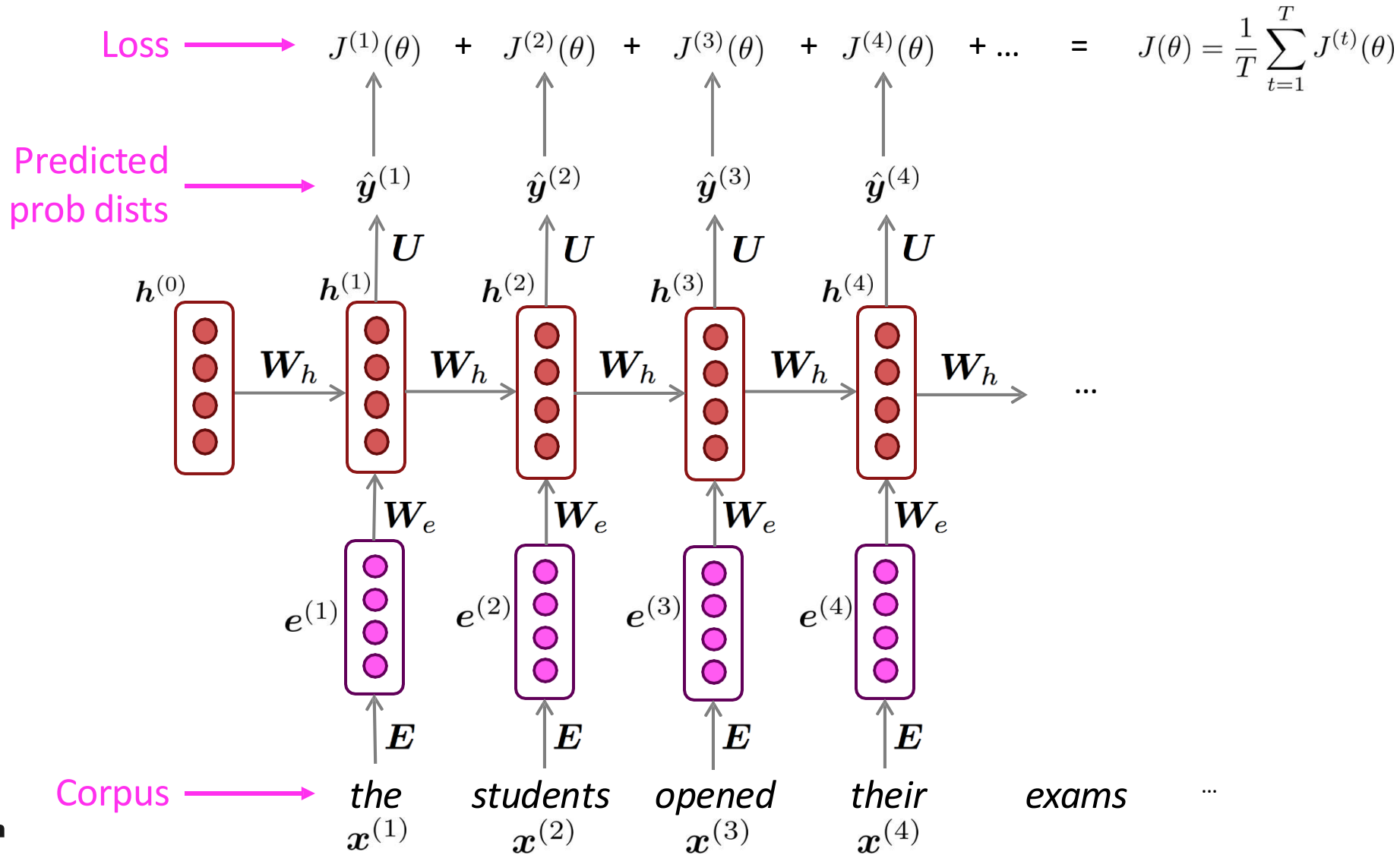
- **BPTT** is just backprop across multiple timesteps, we reuse the same weights, so gradients accumulate. [**Backpropagation through time**]

Teacher Forcing

- During training, feed the **true previous word** (not the model's prediction)
  - **Speeds convergence and stabilises learning.**

Teacher Forcing

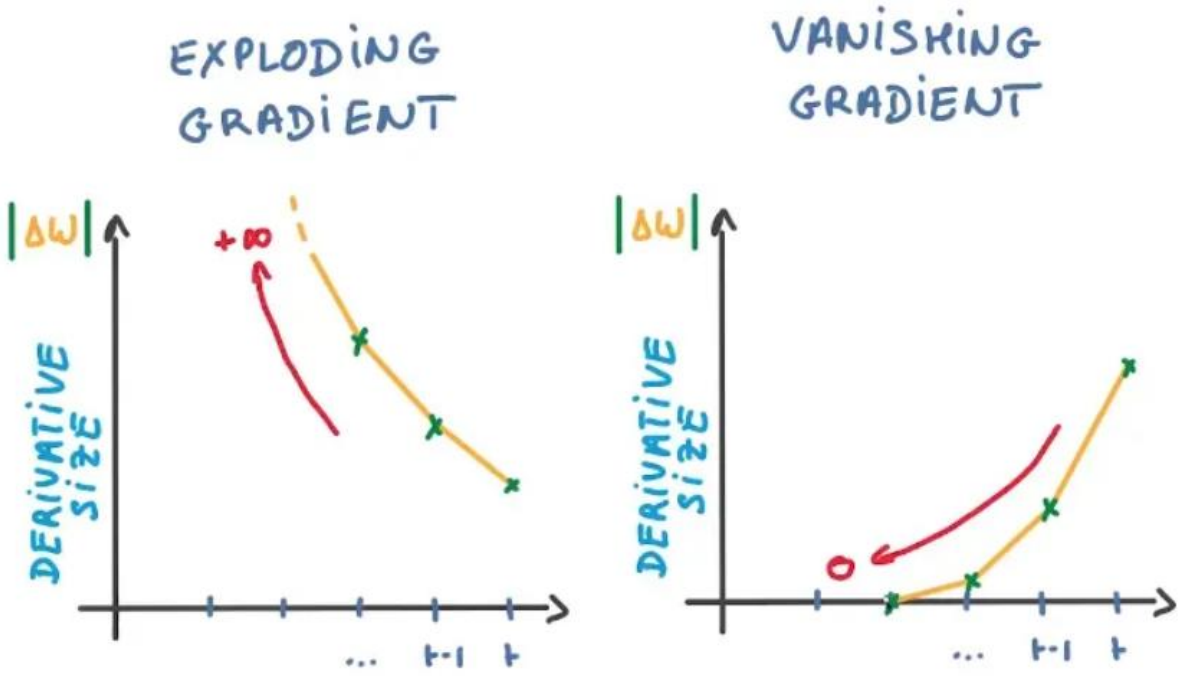
"Teacher forcing"



GRADIENT CHALLENGES

The BPTT Problem

- **Back propagation** through time (**BPTT**) occurs in **RNN**
- In long sequences
  - **Gradients can vanish (model forgets earlier context).**
  - **Or explode (unstable learning).**



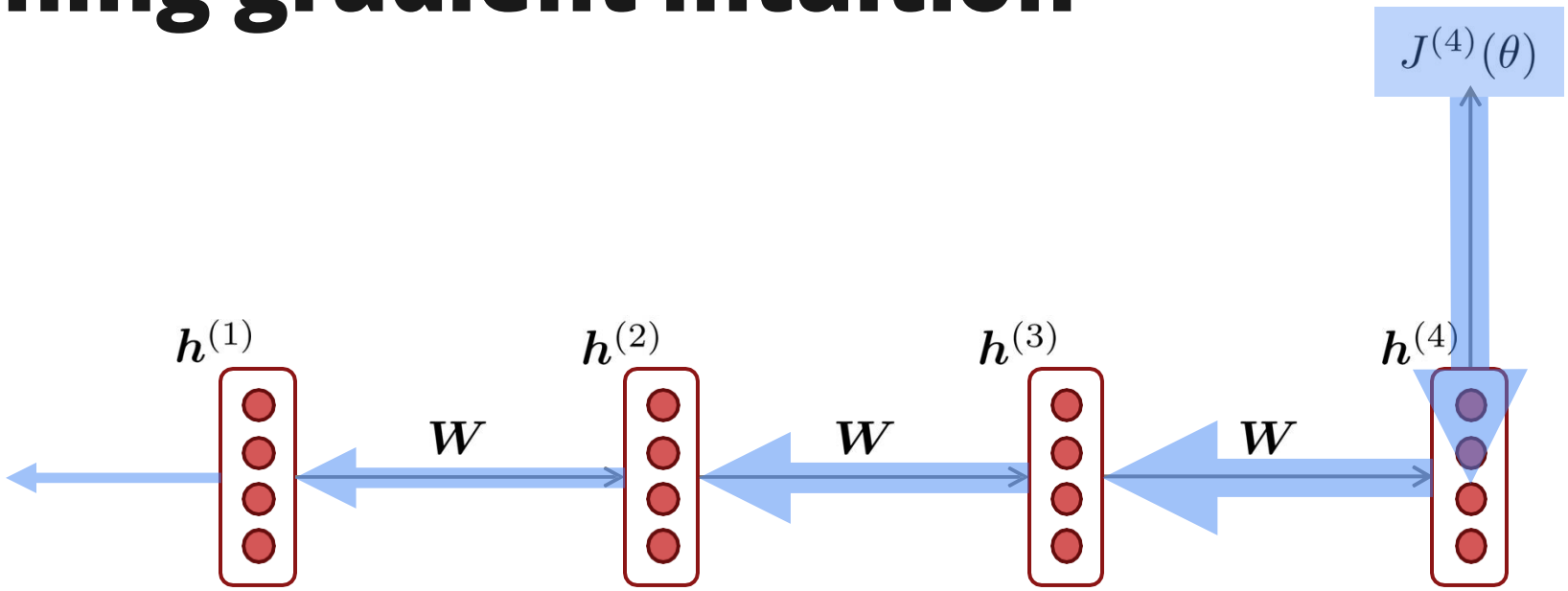
Vanishing Gradient (Concept)

- Each timestep multiplies gradients by W_h repeatedly:

$$\frac{\partial L}{\partial W_h} \propto (W_h)^t$$

- If eigenvalues of $W_h < 1$,gradients **shrink** exponentially, Gradient vanishes
- **Earlier signals vanish**; the network **"forgets"** distant words like the subject of a long sentence.

Vanishing gradient intuition

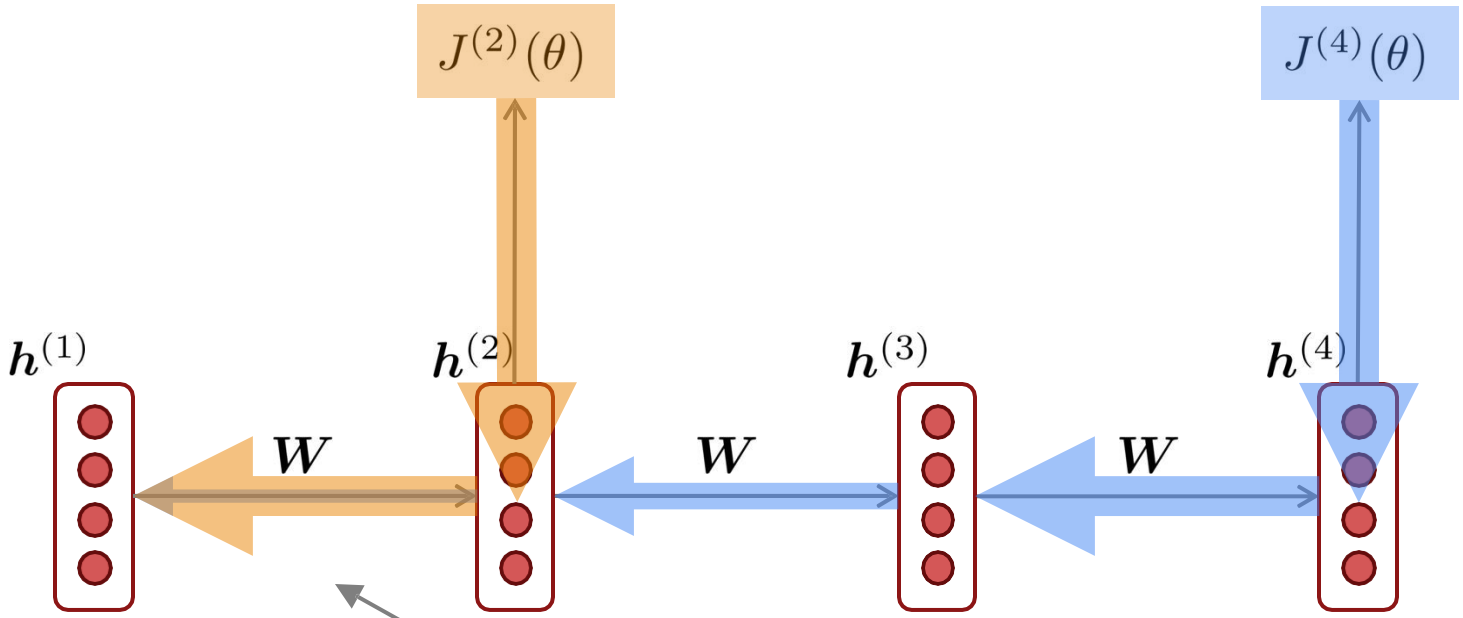


$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

What happens if these are small?

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

Why is vanishing gradient a problem?



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

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

Effect of vanishing gradient on RNN-LM

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

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

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

Exploding Gradient (Concept)

- If eigenvalues of $W_h > 1$,gradients grow exponentially.
  - Weight updates become huge.
  - Training diverges.

Why is exploding gradient a problem?

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

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

- This can cause bad updates: we take **too large a step** and reach a weird and bad parameter configuration (**with large loss**)
  - *You think you've found a hill to climb, but suddenly you're in a different country*

- In the worst case, this will result in Inf or NaN in your network (then you have to restart training from an earlier checkpoint)

How to fix the vanishing gradient problem?

- The main problem is that *it's too difficult for the RNN to learn to preserve information over many timesteps*.

- In a vanilla RNN, the hidden state is **constantly being rewritten**

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

- How about a RNN with separate memory?

LONG SHORT-TERM MEMORY (LSTM)

Long Short-Term Memory RNNs (LSTMs)

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing gradients problem.
  - Everyone cites that paper but really a crucial part of the modern LSTM is from Gers et al. (2000)

- On step *t*, there is a hidden state $\boldsymbol{h}^{(t)}$ *and* a cell state $\boldsymbol{c}^{(t)}$
  - Both are vectors length *n*
  - The cell stores long-term information
  - The LSTM can read, erase, and write information from the cell
    - The cell becomes conceptually rather like RAM in a computer

- The selection of which information is erased/written/read is controlled by three corresponding gates
  - The gates are also vectors length *n*
  - On each timestep, each element of the gates can be open (1), closed (0), or somewhere in-between
  - The gates are dynamic: their value is computed based on the current context

- "Long short-term memory", Hochreiter and Schmidhuber, 1997. https://www.bioinf.jku.at/publications/older/2604.pdf "Learning to Forget: Continual Prediction with LSTM", Gers, Schmidhuber, and Cummins, 2000. https://dl.acm.org/doi/10.1162/089976600300015015

Long Short-Term Memory (LSTM)

We have a sequence of inputs $x^{(')}$, and we will compute a sequence of hidden states $h^{(')}$ and cell states $c^{(')}$. On timestep *t*:

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

Sigmoid function: all gate values are between 0 and 1

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

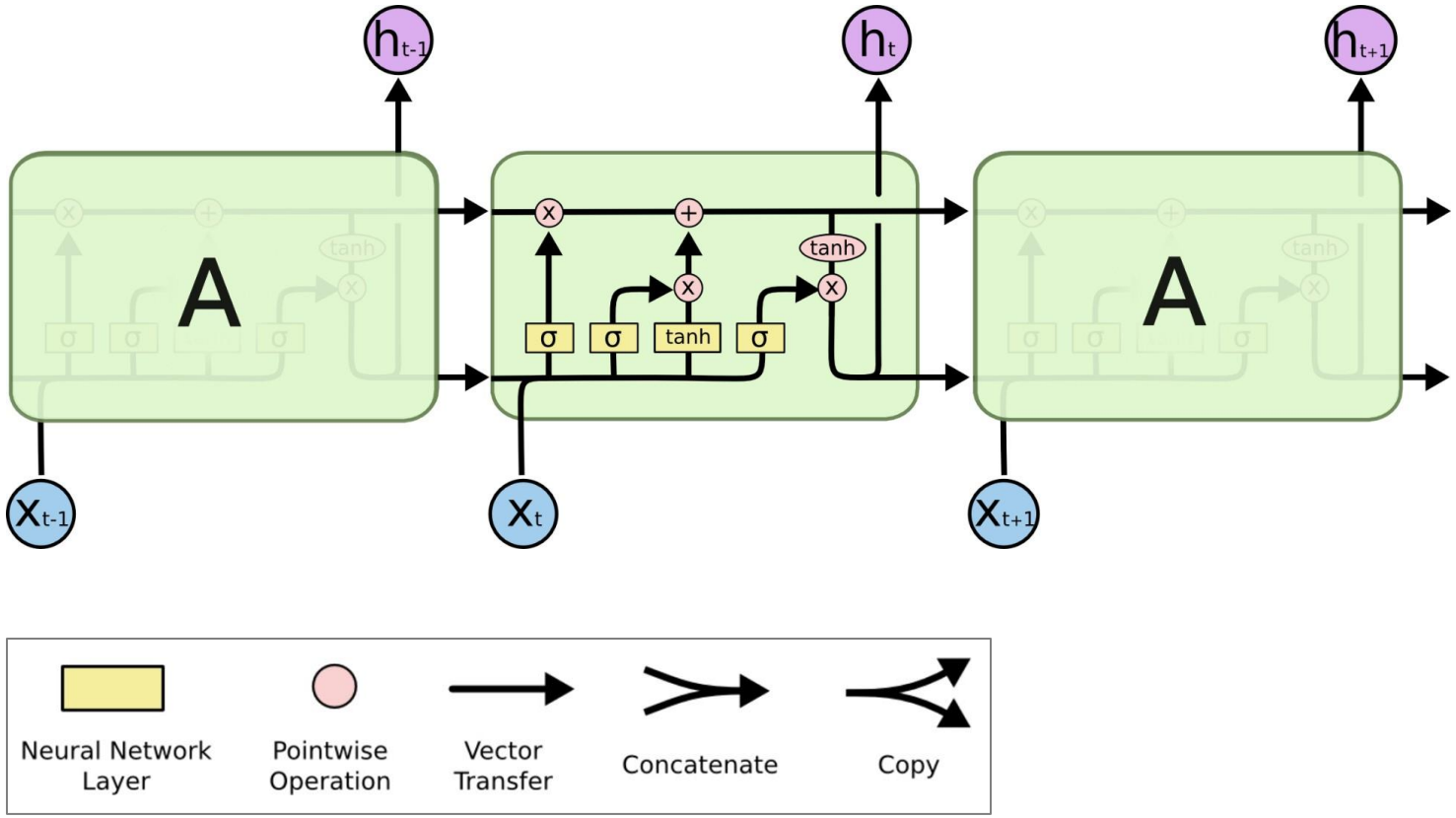
$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

All these are vectors of same length *n*

Gates are applied using element-wise (or Hadamard) product: $\odot$

Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:

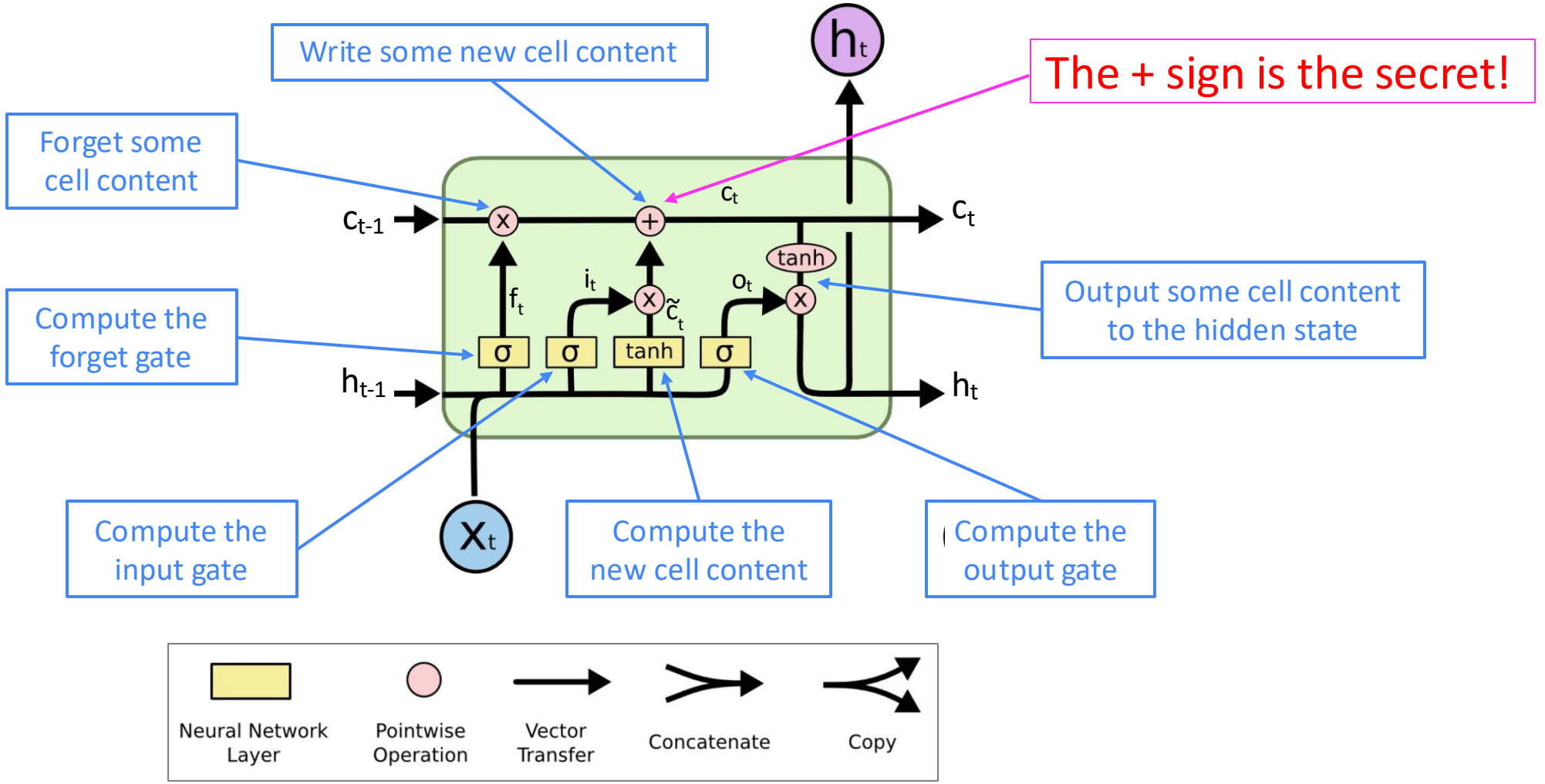


University of Northampton

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

Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



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

Why LSTMs Work

- Additive cell state enables stable gradient flow.
- Gates control what’s remembered or forgotten.
- Learn long-term dependencies (100+ timesteps).

LSTMs: real-world success

- In 2013–2015, LSTMs started achieving state-of-the-art results
  - Successful tasks include **handwriting recognition**, **speech recognition**, **machine translation**, **parsing**, and **image captioning**, as well as language models
  - LSTMs became the dominant approach for most NLP tasks

- Now (2021-2025), other approaches (e.g., Transformers) have become dominant for many tasks
  - For example, in **WMT** (a Machine Translation conference + competition):
  - In WMT 2016, the summary report contains "RNN" 44 times
  - In WMT 2019: "RNN" 7 times, "Transformer" 105 times

Source: "Findings of the 2016 Conference on Machine Translation (WMT16)", Bojar et al. 2016, http://www.statmt.org/wmt16/pdf/W16-2301.pdf
Source: "Findings of the 2018 Conference on Machine Translation (WMT18)", Bojar et al. 2018, http://www.statmt.org/wmt18/pdf/WMT028.pdf
Source: "Findings of the 2019Conference on Machine Translation (WMT19)", Barrault et al. 2019, http://www.statmt.org/wmt18/pdf/WMT028.pdf

GRU Overview

- Gated Recurrent Unit (Cho et al., 2014)
- Simpler alternative to LSTM
- Combines forget & input gates [update gate].
- Merges cell and hidden states.
- Fewer parameters [**faster training**]

LSTM vs GRU Summary

Feature	LSTM	GRU
Gates	3	2
Memory cell	Yes	No (merged)
Complexity	Higher	Lower
Best for	Long sequences	Smaller datasets

Generating text with an RNN Language Model

Let's have some fun!

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

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

"No idea," said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry's shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn't felt it seemed. He reached the teams too.

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

Generating a TV show Episode with LSTM

Let's have some fun!

- You can train an LSTM on any kind of text, then generate text in that style.
- LSTM trained on tv series script:

JARED

It wasn't in the last bonus compound. I've built a deal. They're simply four years who stole his IP options. Keep a promising right through the bike billion valuation.

DINESH

Oh, you believe your opinion, much? If Blaine wants it, until it crashed the same brain!

RICHARD

Whoa! So security arrangements in the flesh. I would love the highest bitrat to you to pay everyone in the old arbitration model, and we should dig that six hours. He probably did this. All the apps from blackjack. All right?

ERLICH

Uh excuse me. I need you to want to skullfuck me. Yep. I'll see your way up, Baltimore. Bernice. Stupid. Know that's just having something if it worked Canadian?

Source: https://medium.com/deep-writing/silicon-valley-a-new-episode-written-by-ai-a8f832645bc2

Ethical Reflection

- Neural LMs can
  - Memorise sensitive data from training sets.
  - Reflect or amplify social bias.
  - Generate plausible but false text.
- Best practice
  - Use differential privacy, filtered datasets, and bias audits.

Summary

- Neural LMs predict the next token in context.
- RNNs capture sequential memory.
- LSTMs and GRUs solve gradient problems.
- Gradient clipping and gating = stable learning.
- Ethical AI = mindful data and deployment.

Practical & Wrap-Up

Thank you