

## RNNs: Other Applications, LSTMs

Pawan Goyal

CSE, IIT Kharagpur

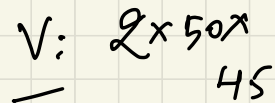
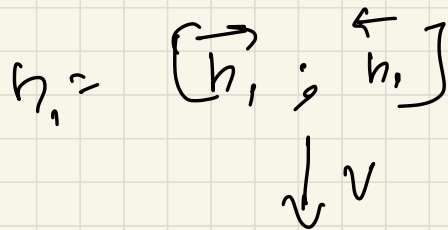
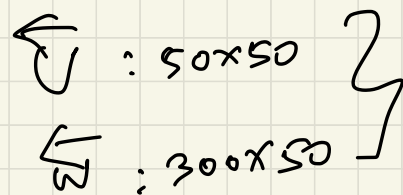
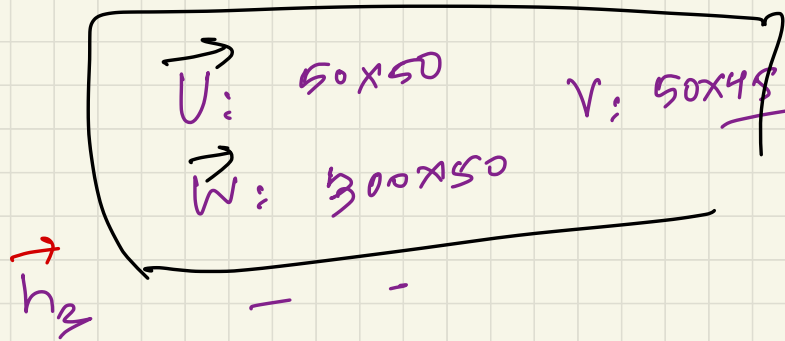
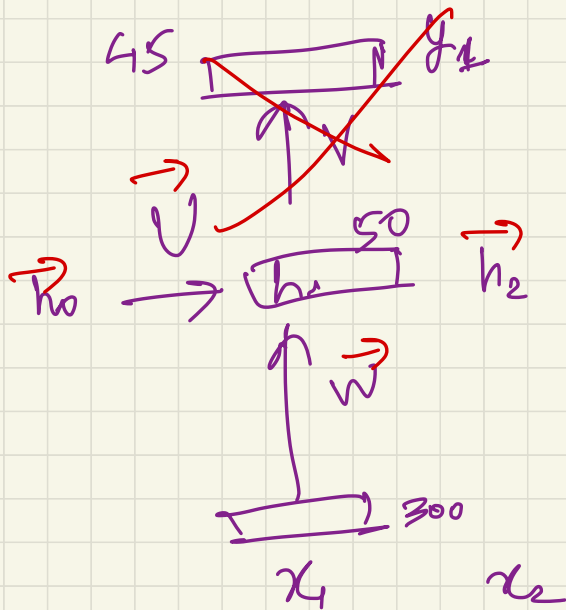
CS60010

300-d word  
embedds

hidden - 50 dim

# of parameters?

Using bi-RNN  
for POS tags  
45 POS tags



5-6 weeks of classes

25% - Mid Sem  
40% - End Sem

GAN

VAE

Diffusion

Transformers

Pretraining (MLP)

↓  
⋮  
↓

L2M

Assignment 1

+ Mini-Project

Groups - 3 members  
1 month

+ 1 Quiz

+ 2 quizzes

+ Attendance

35%

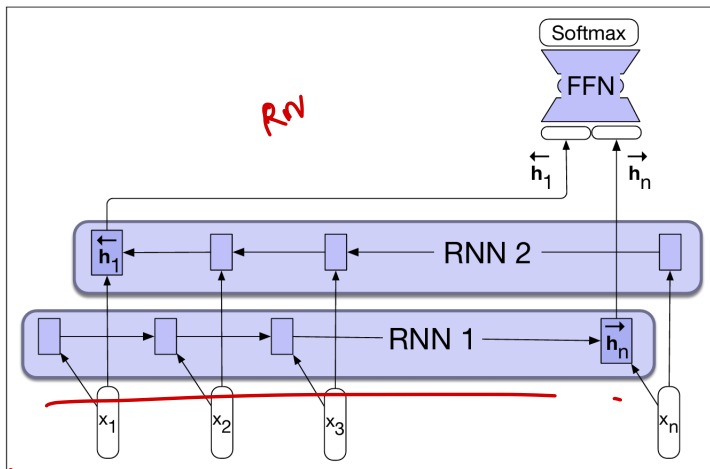
10

20-25

5

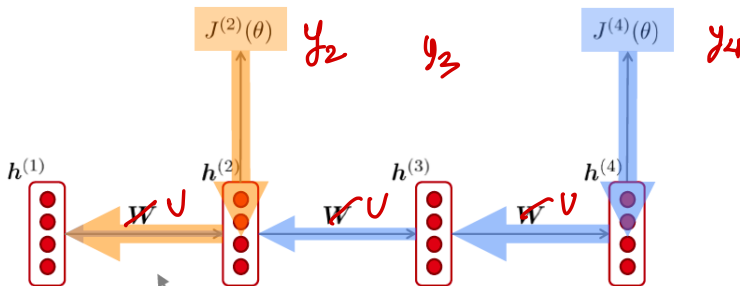
5

# Using Bidirectional RNNs for Sequence Classification



**Figure 9.12** A bidirectional RNN for sequence classification. The final hidden units from the forward and backward passes are combined to represent the entire sequence. This combined representation serves as input to the subsequent classifier.

# Need for better units: Vanishing Gradient



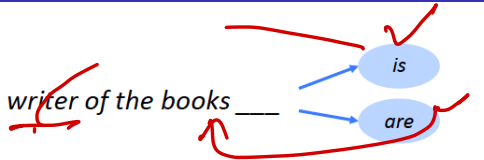

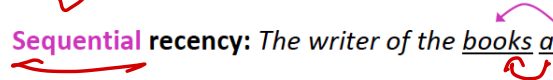
Gradient signal from faraway is lost because it's much smaller than gradient signal from close-by.

So model weights are only 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 7<sup>th</sup> 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

# Effect of vanishing gradient on RNN LM

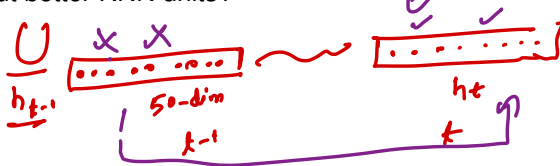
- **LM task:** *The writer of the books \_\_\_\_*  

- **Correct answer:** *The writer of the books is planning a sequel*
- **Syntactic recency:** *The writer of the books is* Syntax (correct)  

- **Sequential recency:** *The writer of the books are* (incorrect)  

- Due to vanishing gradient, RNN-LMs are better at learning from **sequential recency** than **syntactic recency**, so they make this type of error more often than we'd like [Linzen et al 2016]

# How to fix vanishing gradient problem?

- The main problem is that it is too difficult for the RNN to learn to preserve information over many timesteps.
- In a vanilla RNN, the hidden state is constantly being rewritten

$$h^{(t)} = \underbrace{\tanh}_{\text{non-linear}} \left( \underbrace{U}_{\text{matrix}} \underbrace{h^{(t-1)}}_{\text{vector}} + \underbrace{W}_{\text{matrix}} \underbrace{x^{(t)}}_{\text{vector}} \right) + \alpha h^{(t-1)}$$

- How about better RNN units?

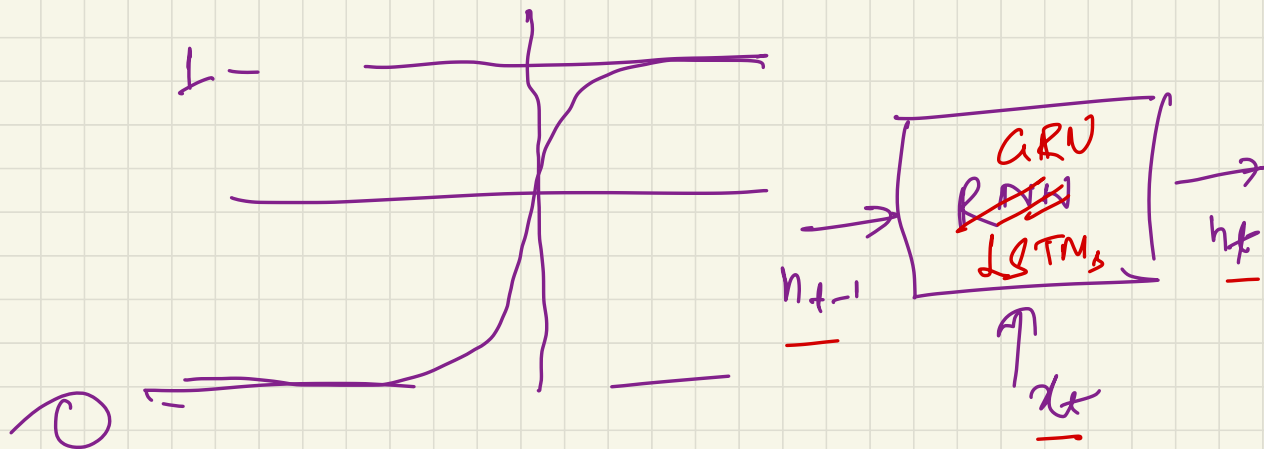




gate 1 0 0 1 1 1  
50

Neural Gate

Sigmoid



$$\underline{i_t} = \sigma \left( \underset{\substack{\uparrow \\ \text{more learnable} \\ \text{params}}}{U_i} h_{t-1} + \underset{\substack{\uparrow \\ \text{more learnable} \\ \text{params}}}{W_i} x_t \right)$$

50

## Using Gates for better RNN units

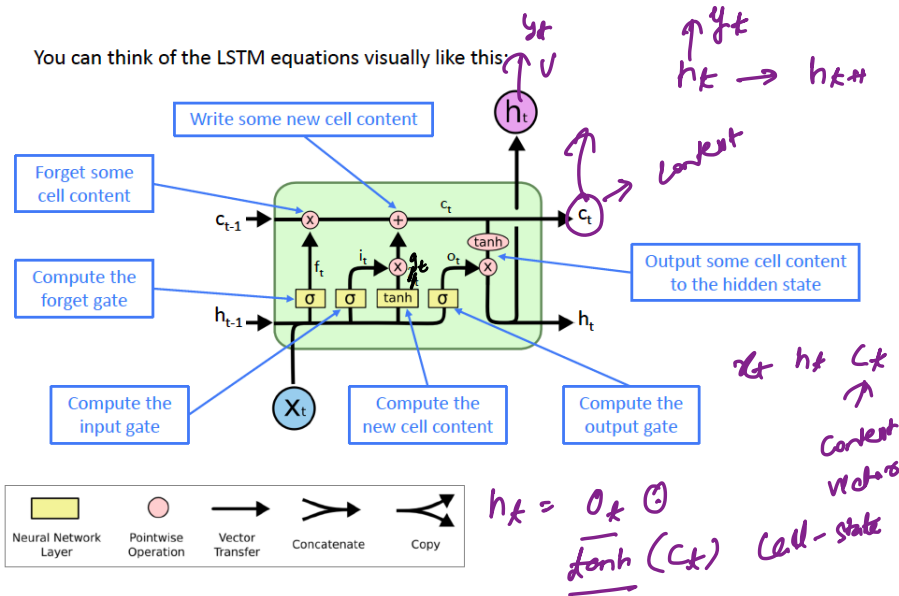
- The gates are also vectors
- On each timestep, each element of the gates can be open (1), close (0) or somewhere in-between.
- The gates are dynamic: their value is computed based on the current context.

*Two famous architectures*

GRUs, LSTMs

# Long Short Term Memory (LSTM)

You can think of the LSTM equations visually like this:



$$\underline{C_{t-1}}$$

$$h_{t-1}$$

$$x_t$$

$$g_t = \tanh(W_g x_t + U_g h_{t-1})$$

$$g_t =$$

$$\boxed{0.25 \mid -0.2 \mid 0.3}$$

$$f_t \odot C_{t-1} + i_t \odot \overrightarrow{g_t}$$

new state content

(like RNN)

$$i_t =$$

$$\boxed{0 \mid 0 \mid 1}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1})$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1})$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1})$$

- For context management, an explicit context layer is added to the architecture
- It makes use of specialized neural units (gates) to control the flow of information
- The gates share a common design feature, and choice of sigmoid pushes its output to 0 or 1, thus it works as a binary mask.

# LSTM: In Equations

## Forget Gate

Controls what is kept vs forgotten from the context

$$f_t = \sigma(U_f h_{t-1} + W_f x_t)$$

## Input Gate

Controls what parts of new cell content are written to the context

$$i_t = \sigma(U_i h_{t-1} + W_i x_t)$$

## Output Gate

Controls what part of context are output to hidden state

$$o_t = \sigma(U_o h_{t-1} + W_o x_t)$$

**New Cell content:**  $g_t = \tanh(U_g h_{t-1} + W_g x_t)$

**New Context Vector:**  $c_t = i_t \odot g_t + f_t \odot c_{t-1}$

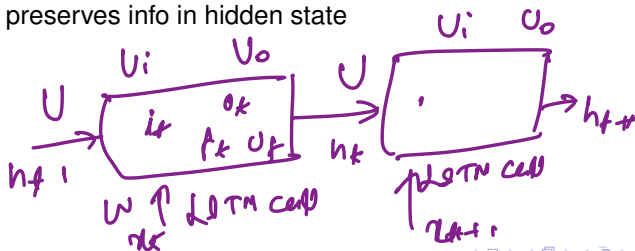
**New Hidden State:**  $h_t = o_t \odot \tanh(c_t)$

# How does LSTM solve vanishing gradients?

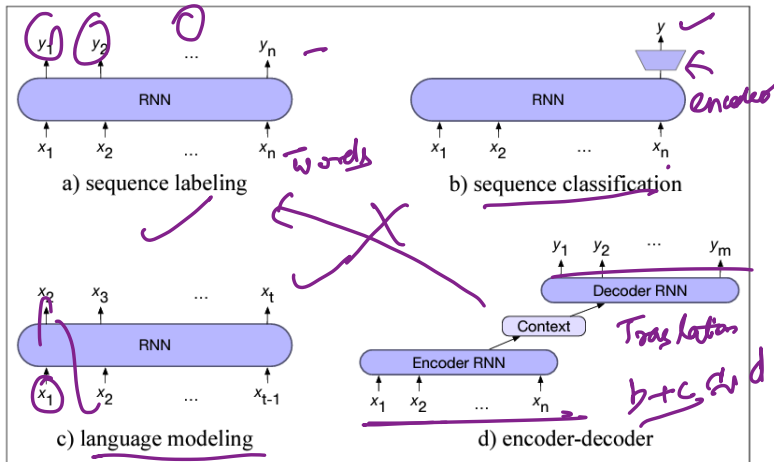


The LSTM architecture makes it easier for the RNN to preserve information over many timesteps

- e.g., if the forget gate is set to remember everything on every timestep, then the info in the cell is preserved indefinitely
- By contrast, it is harder for vanilla RNN to learn a recurrent weight matrix  $U$  that preserves info in hidden state



# Common RNN NLP Architectures



**Figure 9.15** Four architectures for NLP tasks. In sequence labeling (POS or named entity tagging) we map each input token  $x_i$  to an output token  $y_i$ . In sequence classification we map the entire input sequence to a single class. In language modeling we output the next token conditioned on previous tokens. In the encoder model we have two separate RNN models, one of which maps from an input sequence  $\mathbf{x}$  to an intermediate representation we call the **context**, and a second of which maps from the context to an output sequence  $\mathbf{y}$ .



# Encoder-decoder networks

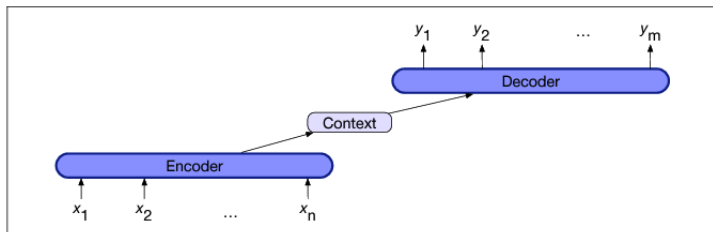
Also known as sequence-to-sequence networks, and are capable of generating contextually appropriate, arbitrary length output sequences given the input sequence.

## Three conceptual components

- An **encoder** that accepts an input sequence  $x_{1:n}$  and generates a corresponding sequence of contextualized representations  $h_{1:n}$
- A **context** vector,  $c$ , which is a function of  $h_{1:n}$  and conveys the essence of the input to the decoder
- A **decoder** which accepts  $c$  as input and generates an arbitrary length sequence of hidden states  $h_{1:m}$  from which the corresponding output states  $y_{1:m}$  can be obtained.

— 7

# Encoder-decoder networks



**Figure 9.16** The encoder-decoder architecture. The context is a function of the hidden representations of the input, and may be used by the decoder in a variety of ways.