A 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

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

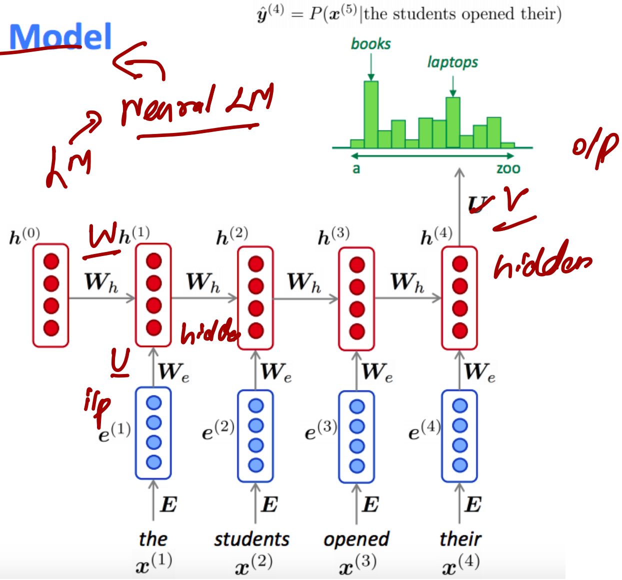
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

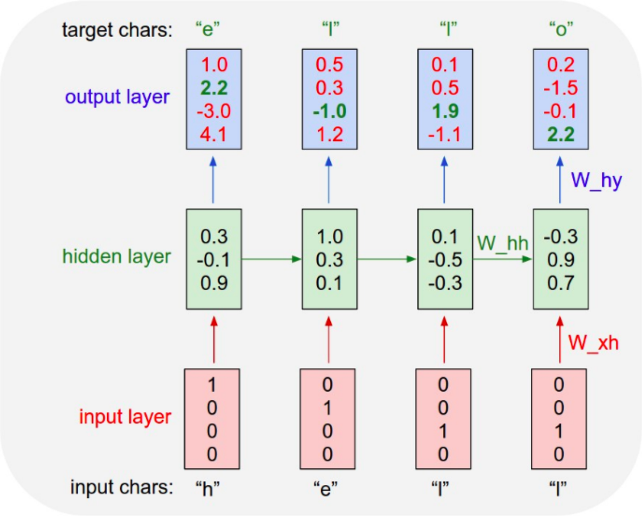
$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$

words / one-hot vectors

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

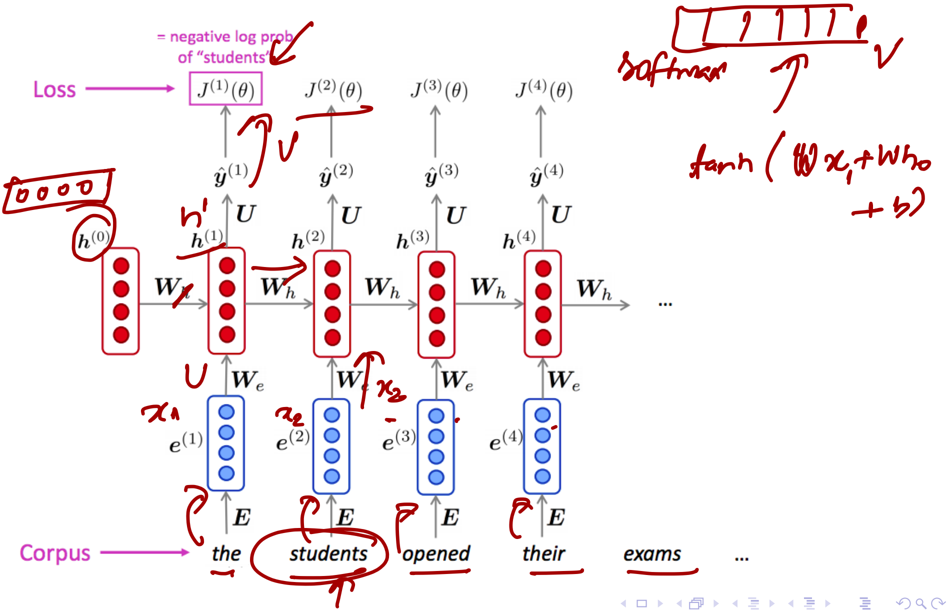


Example RNN

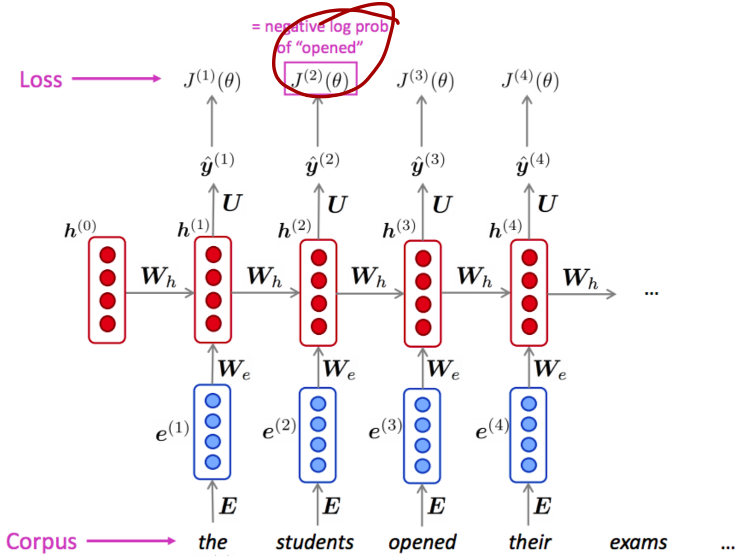


target chars:
"e"
"l"
"l"
"o"
output layer
1.0
2.2
-3.0
4.1
0.5
0.3
-1.0
1.2
0.1
0.5
1.9
-1.1
0.2
-1.5
-0.1
2.2
W_hy
hidden layer
0.3
-0.1
0.9
1.0
0.3
0.1
0.1
-0.5
-0.3
W_hh
-0.3
0.9
0.7
W_xh
input layer
1
0
0
0
0
1
0
0
0
0
1
0
0
0
1
0
input chars:
"h"
"e"
"l"
"l"

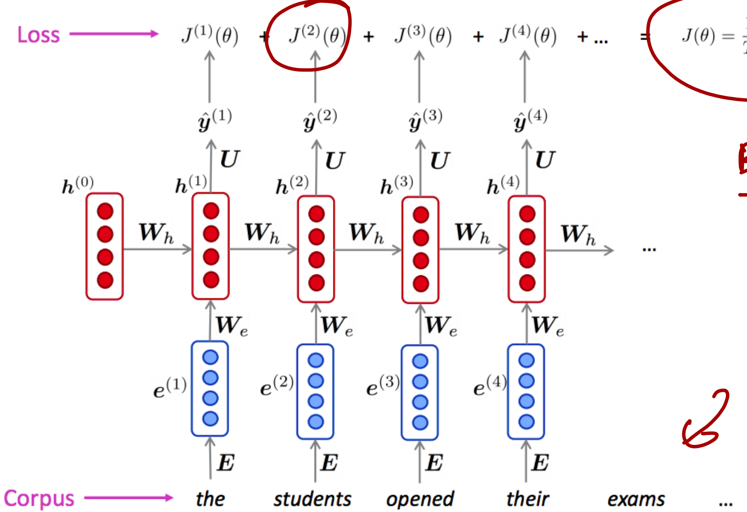
Training a RNN language model



Training a RNN language model



Training a RNN language model



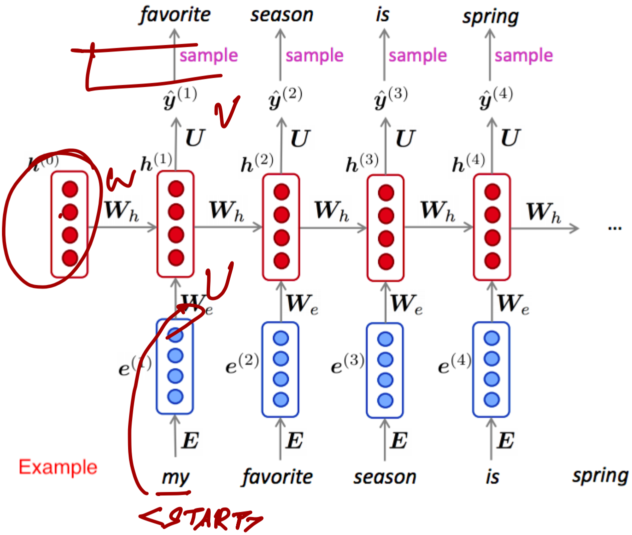
$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta)$$

BPTT

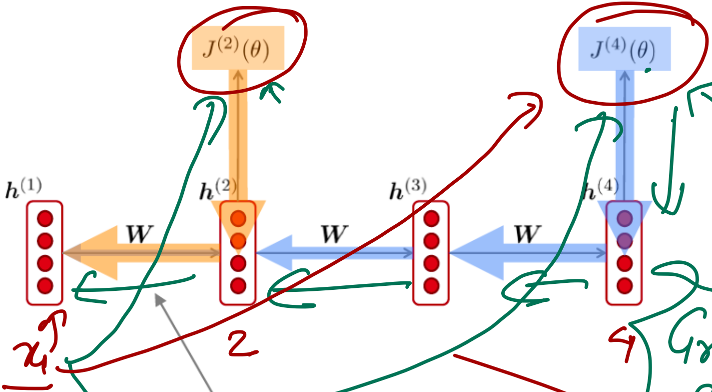
teacher-forcing


5 7
7
10
7
8
20
10
10
Sort by length

Generating text with a RNN Language Model



Vanishing Gradient Problem with RNNs



Gradient clipping

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.

Gradient Explosion

Gradient Vanishing

bigger problem

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

Effect of vanishing gradient on RNN LM

- **LM task:** *The writer of the books ___* are is → *is* / *are*
- **Correct answer**: *The writer of the books is planning a sequel*
- **Syntactic recency:** *The writer of the books is* (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]

sub. singula

sub. singular

h_1 h_2 $\rightarrow$ h_3 $\rightarrow$ h_4 $\rightarrow$ h_5

x_1 x_2 x_3 x_4 x_5

the   write   of   the   books

$$h_3 = \tanh\left(Ux + \boxed{W h_2} + b\right)$$

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

$$\boxed{h^{(t)} = tanh(Wh^{(t-1)} + Ux^{(t)} + b)}$$

- How about better RNN units?

0.5 -0.1 0.3

1 0 ~~0~~ 0 0 0 0 0 0

h_2

keep as it is

0.5

0, 1

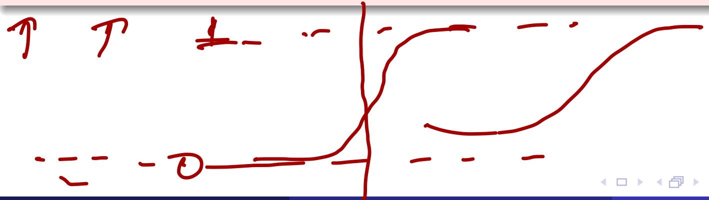
h_3

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



RNN → GRU (2014) LSTM (1997)

$\tanh(U x_t + W h_{t-1} + b)$

r_t Reset Gate

u_t Update Gate

$\boxed{0 \mid 0 \mid 0\ 0\ 0\ 0\ 0\ 0}$ h_{t-1}

$\boxed{0 \mid 0\ 0\ 0\ 0\ 0\ 0}$ h_t

$\boxed{1 \mid 0}$ u_t

$\boxed{0.5 \mid 0}$ r_t

$u_t = 1$

$r_t =$ anything

GRU

$$r_t = \sigma(U_r x_t + W_r h_{t-1} + b_r)$$

$$u_t = \sigma(U_u x_t + W_u h_{t-1} + b_u)$$

$$\tilde{h}_t = \tanh(U_h x_t + W_h (r_t \circ h_{t-1}) + b_h)$$

elem wise

$$h_t = u_t \circ h_{t-1} + (1 - u_t) \circ \tilde{h}_t$$

Gated Recurrent Units (GRU)

$[u_t = 1, r_t = 0]$ forget

$u_t = 0$ remember

- Proposed by Cho et al. in 2014 as a simpler alternative to the LSTM.
- On each timestep t we have input $\boldsymbol{x}^{(t)}$ and hidden state $\boldsymbol{h}^{(t)}$ (no cell state).

Update gate: controls what parts of hidden state are updated vs preserved

Reset gate: controls what parts of previous hidden state are used to compute new content

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

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

gates

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

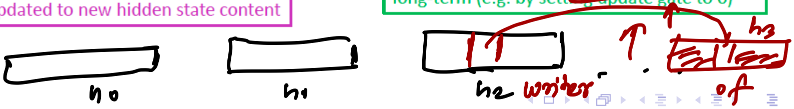
$$\tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}_h(\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}) + \boldsymbol{U}_h \boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right)$$

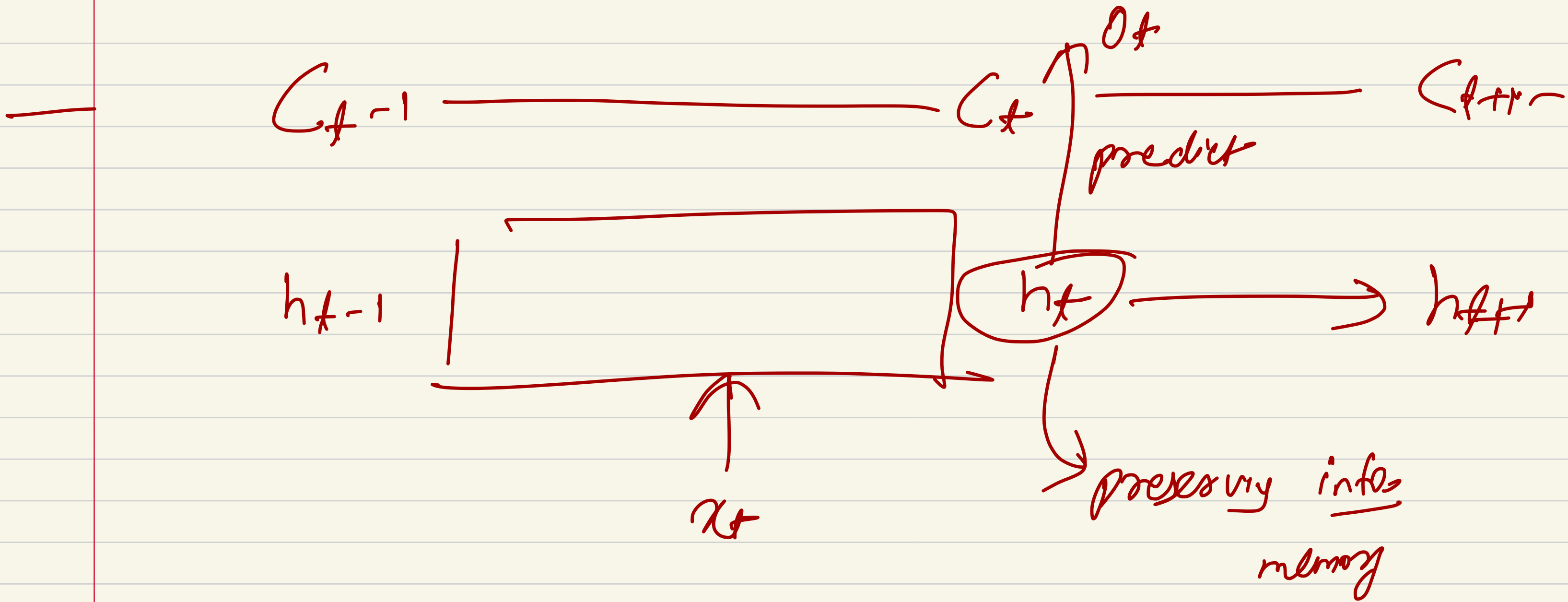
$$\boldsymbol{h}^{(t)} = \underbrace{\boxed{(1 - \boldsymbol{u}^{(t)}) \circ \boldsymbol{h}^{(t-1)}}}_{\approx 1} + \underbrace{\boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}}_{\approx 0}$$

$u_t = 0$

How does this solve vanishing gradient?
Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)

0 1





LSTM

cell state

LSTM

three gates $\qquad \underline{C_{t-1}} \qquad\qquad \underline{C_t}$

$\qquad\qquad\qquad h_{t-1} \qquad\qquad \underline{h_t}$

i/p gate $\quad i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$

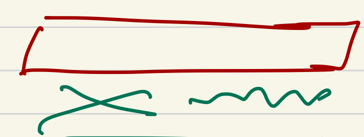
o/p gate $\quad O_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$

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

1. Update cell content $\qquad \underline{\tilde{C}_t} = \tanh(W_c h_{t-1} + U_c x_t + b_c)$

$$\rightarrow \underline{C_t} = f_t \circ C_{t-1} + i_t \circ \underline{\tilde{C}_t}$$

$$\rightarrow \underline{h_t} = \underline{O_t \circ \tanh(C_t)}$$



$O_t \; [0\,0\,0\,0\,0\,1\,1\,1\,1\,1]$

Long Short Term Memory (LSTM)

We have a sequence of inputs $\boldsymbol{x}^{(t)}$, and we will compute a sequence of hidden states $\boldsymbol{h}^{(t)}$ and cell states $\boldsymbol{c}^{(t)}$. 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

Sigmoid function: all gate values are between 0 and 1

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

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

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

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

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

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

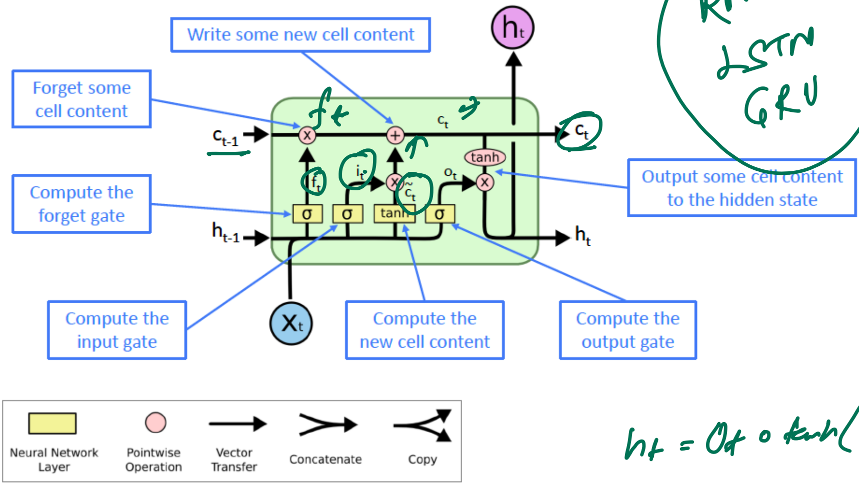
$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

Gates are applied using element-wise product

All these are vectors of same length *n*

Long Short Term Memory (LSTM)

You can think of the LSTM equations visually like this:



RNN
LSTM
GRU

$h_t = O_t \circ \tanh(C_t)$