CS 378 Lecture 16

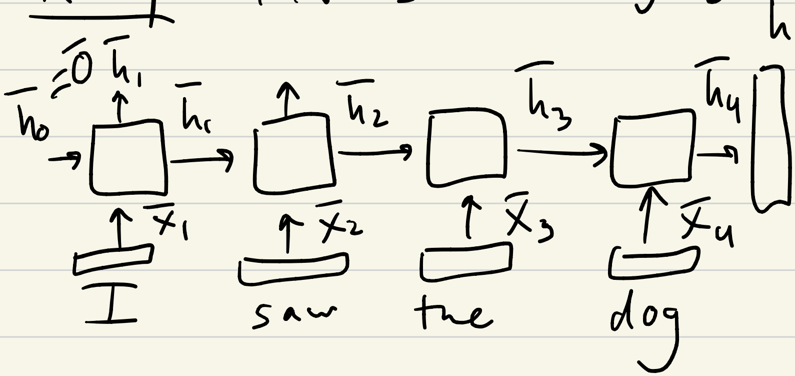
Today

- RNNs
- LSTMs (the type of RNN you will be using)
- Implementation

Announcements

- A4 out later today
- Midterm back Monday

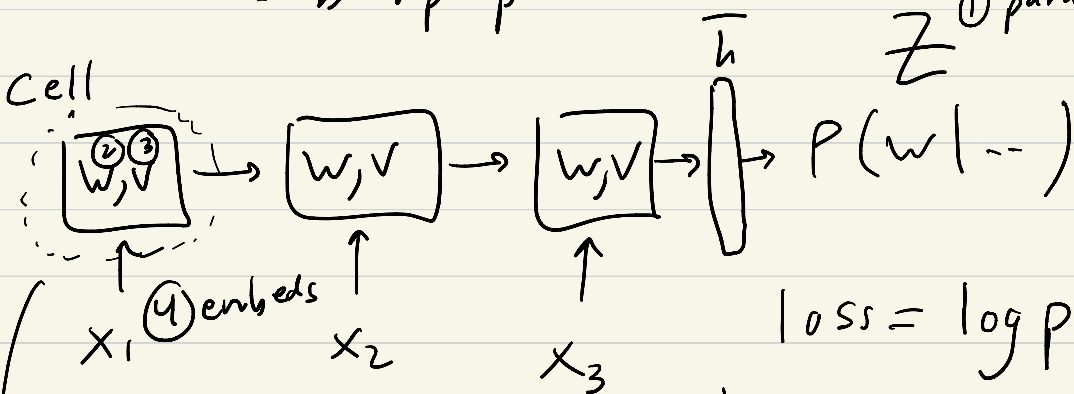
Recap RNNs + Language modeling



$$P(w \mid \text{I saw the dog}) = \text{softmax}(z \cdot \bar{h})$$

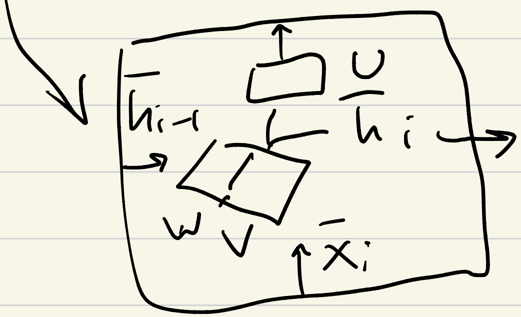
Training "Backpropagation through time"

= backprop



loss = log p(w^* ...

backprop to get gradient



<u>This</u> <u>still</u> <u>works</u>!

Multiple updates for W, V, sum them together

Backprop: loss → gradient for each param

LSTMs

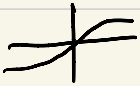
What we've seen so far is a
recipe for RNNs
Many types of RNNs

Long short-term memory (LSTM)
1998

RNN state is a "short-term memory"
LSTMs are better at remembering
info for more timesteps

Problem with Elman networks:
vanishing gradient problem
(exploding)

Elman: $\bar{h}_3 = \tanh(W\bar{x}_3 + V \cdot$
$\tanh(W\bar{x}_2 + V \cdot$
$(\tanh(W\bar{x}_1)))$

tanh 

LSTM definition

Elman: $\bar{h}_i = \tanh(W\bar{x}_i + V\bar{h}_{i-1})$

Gated: $\bar{h}_i = \bar{h}_{i-1} \odot \bar{f} + \text{function}(\bar{x}_i, \bar{h}_{i-1}) \odot \bar{i}$

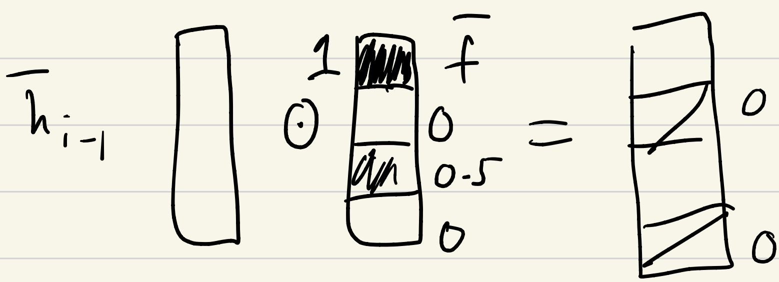
prev state

elementwise multiplication

f: forget gate

$\bar{i}$: input gate

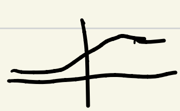
vector of values in $[0, 1]$



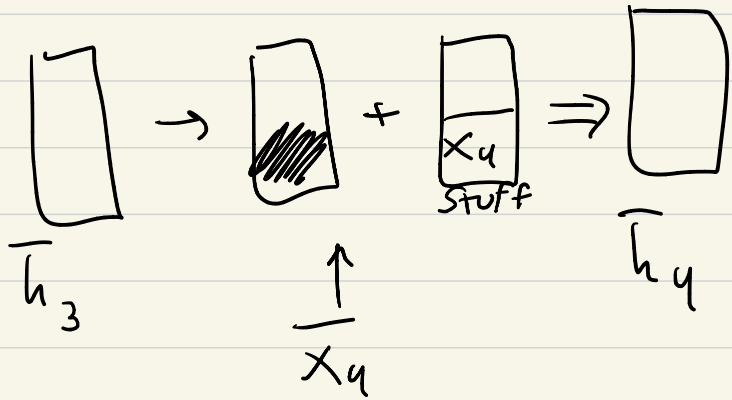
$$f = \text{sigmoid}(W^{(1)}\bar{x}_i + W^{(2)}\bar{h}_{i-1})$$

$$i = \text{sigmoid}(W^{(3)}\bar{x}_i + W^{(4)}\bar{h}_{i-1})$$

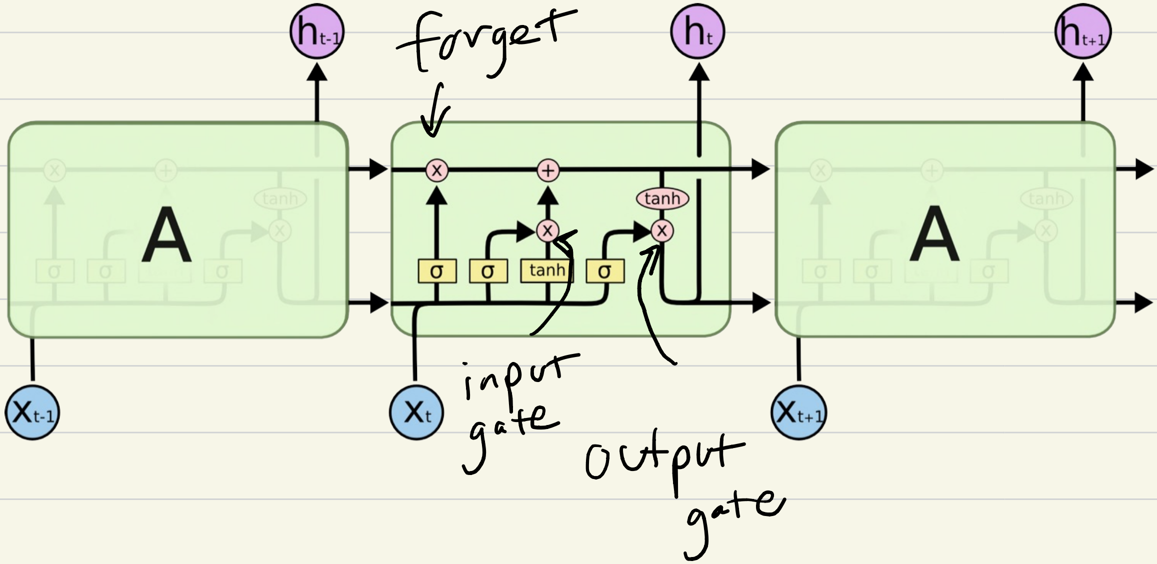
$$\sigma : \frac{e^x}{1+e^x}$$



$\bar{h}$ is "memory"



Real LSTM: next page

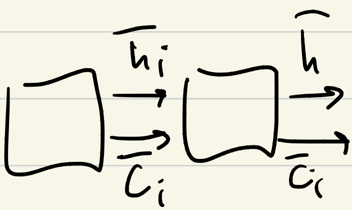


Chris Olah blog

LSTM: 8 weight matrices

$\bar{h}$, $\bar{c}$
hidden state, cell state

in Pytorch, these are dealt with as a tuple



Poll:

forget bias

$$f = \text{sigmoid}\left(W^{(1)}\bar{x}_i + W^{(2)}\bar{h}_{i-1} + b_{forget}\right)$$

$$i = \text{sigmoid}\left(W^{(3)}\bar{x}_i + W^{(4)}\bar{h}_{i-1} + b_{input}\right)$$

[0, 1, 2] closer to [1, 1, 2)

than [0, 1, 3]

Most recent stuff is more relevant

Forget gate high: changes further back matter more