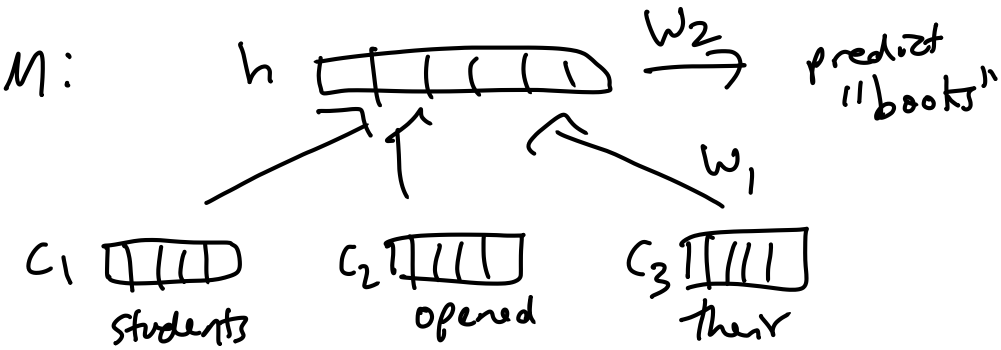
This class:

- gradient descent ⇒ cross-entropy loss
- backpropagation
  - ↳ feed-forward NN } single neurons
  - ↳ recurrent NN
  - ↳ see F2020 video for backprop thru linear layer

NLM:



params: W_1, W_2, c_1, c_2, c_3

$$h = f(W_1[c_1; c_2; c_3])$$

$$o = \text{Softmax}(W_2 h)$$

h, o are intermediate vars

---

how do we train this model to make better predictions ⇒ gradient descent!

1. define loss fn $L(\theta)$ ← all params, that tells us how bad the model is doing at predicting the next word
   - ↳ cross entropy loss

↳ say we have a training ex

$\underbrace{\text{students opened their}}_{\text{input prefix}} \Rightarrow \underbrace{\text{books}}_{\text{target}}$

$p(\text{books} \mid \text{"students opened their"})$

↳ maximize this probability

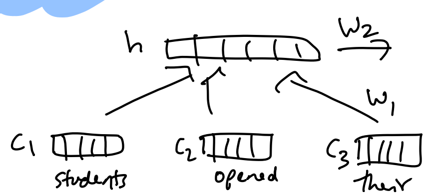
in practice,

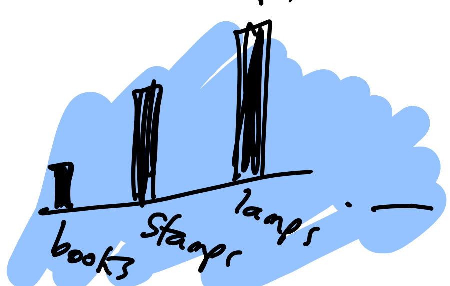
we minimize the negative log prob of "books"

$L = -\log(p(\text{books} \mid \text{"students opened their"}))$

↳ cross entropy loss

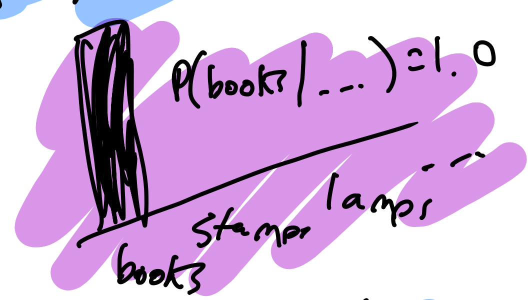
model:





data:

students opened their



Cross-entropy loss of two dists p and q

$$-\sum_{w \in V} p(w) \log q(w)$$

p ↳ data
1 for books, 0 for everything else

q ↳ model

$= -\log q(\text{books} \mid \text{students opened their})$
(neg. log prob of correct word)

---

okay, so we have a loss fn $L(\theta)$
need to compute gradient of L WRT θ
↳ model params

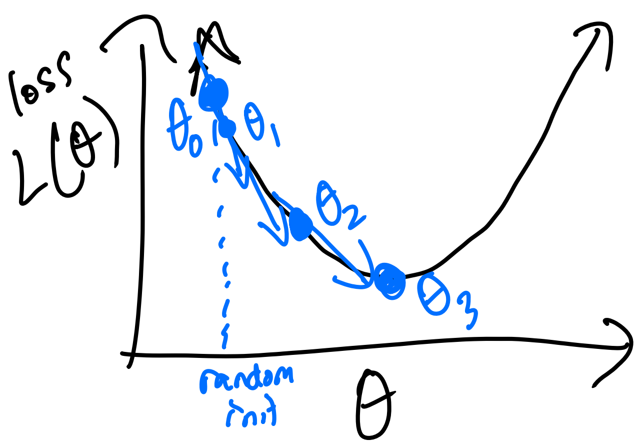
$$\frac{dL}{d\theta}$$

(↳ gradient tells us the direction of steepest ascent of L, intuitively, tells us how L changes when we modify θ

3. after computing $\frac{dL}{d\theta}$, take a step in direction of negative gradient, minimizing L

$$\theta_{new} = \theta_{OLD} - \eta \frac{dL}{d\theta}$$

↳ gradient
↳ learning rate, controls step size



learning rate is a hyperparameter

batch size is important hyperparam

Simple example:

$$x \xrightarrow{w_1} h \xrightarrow{w_2} o \qquad o = \tanh(w_2 h) \qquad \text{input: } (x, y)$$

$$h = \tanh(w_1 x)$$

compute loss fn : $L = \frac{1}{2}(y - o)^2$ square loss

y ↳ target, o ↳ model prediction

compute gradient $\frac{dL}{d\theta}$: $\frac{dL}{dw_1}$, $\frac{dL}{dw_2}$

chain rule of calculus

$$\frac{d}{dx} g(f(x)) = \frac{dg}{df} \cdot \frac{df}{dx}$$

$$L = \frac{1}{2}(y - o)^2$$

$$o = \tanh(w_2 h)$$

intermediate var $a = w^2 h$, $b = w_1 x$

$$o = \tanh(a)$$

$$a = w_2 h, \quad h = \tanh(b), \quad b = w_1 x$$

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$$

$$\frac{dL}{dw_2} = \frac{dL}{do} \cdot \frac{do}{da} \cdot \frac{da}{dw_2}$$

$$-(y - o) \cdot (1 - o^2) \cdot h$$

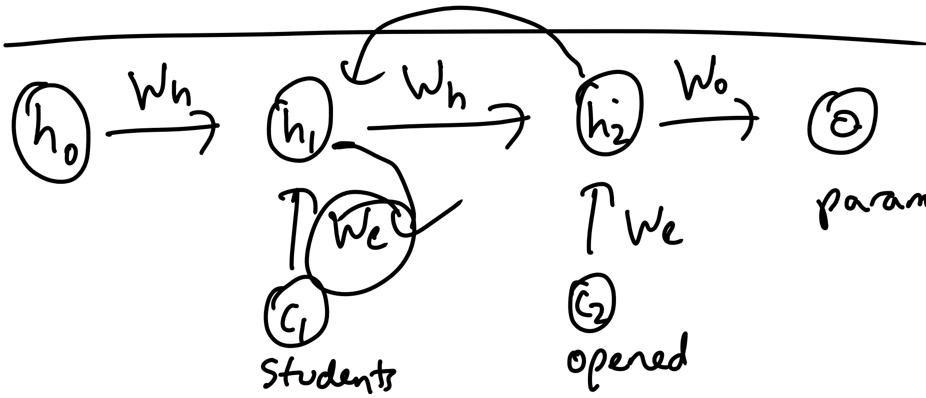
$$\frac{dL}{dw_1} = \frac{dL}{do} \cdot \frac{do}{da} \cdot \frac{da}{dh} \cdot \frac{dh}{db} \cdot \frac{db}{dw_1}$$

backprop = chain rule + caching prev. computed derivatives

Update params:

$$W_{2_{new}} = W_{2_{OLD}} - \eta \frac{dL}{dW_2}$$

$$W_{1_{new}} = W_{1_{OLD}} - \eta \frac{dL}{dW_1}$$



params: $W_h, W_e, W_o,$
c_1, c_2

$$L = \frac{1}{2}(y-o)^2$$

$$o = W_o h_2$$

$$h_2 = \tanh(\underbrace{\underline{W_e} c_2 + W_h h_1}_{a})$$

$$h_1 = \tanh(\underbrace{W_e c_1 + W_h h_0}_{b})$$

$$\frac{dL}{dW_o} = \frac{dL}{do} \cdot \frac{do}{dW_o} = -(y-o) \cdot h_2$$

$$\frac{dL}{dc_2} = \frac{dL}{do} \cdot \frac{do}{dh_2} \cdot \frac{dh_2}{da} \cdot \frac{da}{dc_2} = -(y-o) \cdot W_o \cdot (1-h_2^2) \cdot W_e$$

$$\frac{dL}{dc_1} = \frac{dL}{do} \cdot \frac{do}{dh_2} \cdot \frac{dh_2}{da} \cdot \frac{da}{dh_1} \cdot \frac{dh_1}{db} \cdot \frac{db}{dc_1}$$

$\frac{dL}{dW_e}$ and $\frac{dL}{dW_h}$ are trickier b/c of shared weights

"backpropagation thru time" allows us to compute these gradients by summing the contributions from diff. time steps

$$\frac{dL}{dW_e} = \frac{dL}{do} \cdot \frac{do}{dh_2} \cdot \frac{dh_2}{da} \cdot \frac{da}{dW_e} + \underset{\text{other timesteps}}{\cdots}$$

we accumulate these $\frac{dL}{dW_e}$ as we step back thru time.

the vanishing gradient problem occurs in RNNs when gradient contributions at faraway timesteps go to zero