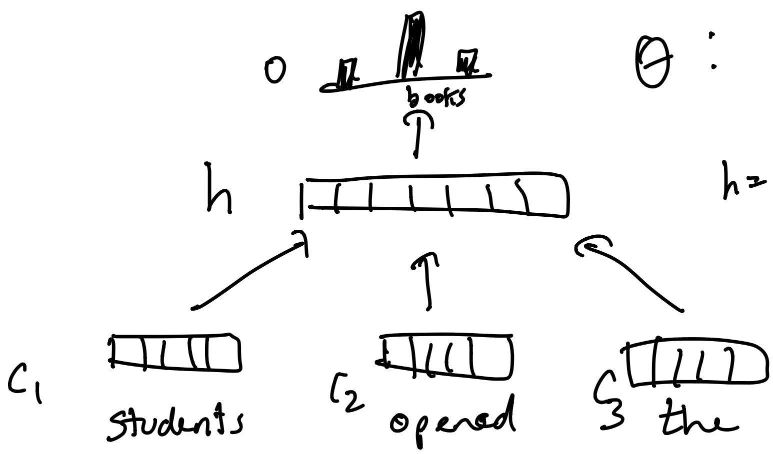
How do we <u>train</u> a neural language model?

how to adjust the params of our model to better predict the next word

---

NLM (concatenation):



θ : params: W_1, W_2, c_1, c_2, c_3

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

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

Steps to train this model:

1. define a LOSS FN $L(\theta)$, this tells us how bad the model currently is at predicting the next word
   - ↳ smooth, differentiable

2. Given $L(\theta)$, we compute the gradient of L with respect to θ
   - ↳ gradient gives us the direction

of steepest ascent of L

↳ same dimensionality as θ

↳ for each param j in θ, gradient tells you how much L would change if you increase j by a very small amount $\frac{dC}{d\theta}$

↳ for concat LM

$$\frac{dC}{d\theta} = \left\{ \frac{dL}{dW_1}, \frac{dL}{dW_2}, \frac{dL}{dc_1} \ldots \right\}$$

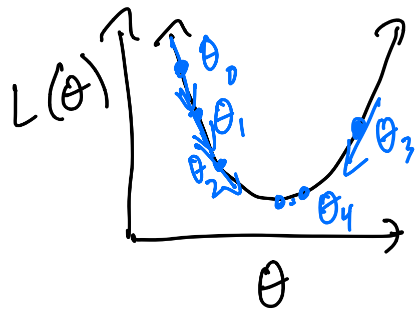
3. Given gradient $\frac{dL}{d\theta}$, we take a step in the direction of the negative gradient

↳ this minimize L

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

↳ gradient

→ learning rate, controls step size



optimizer:
- stochastic grad. descent
- Adam
- Adafactor
- ..

important hyperparams:

- learning rate η
- batch size
  - how many training examples do you use to estimate $\frac{dL}{d\theta}$ before taking a step

---

Loss function used to train NLMs

↳ cross-entropy loss

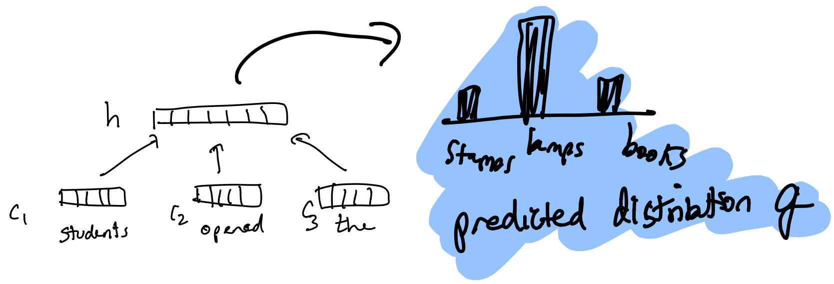
$\underbrace{\text{students opened their}}_{\text{training prefix}} \Rightarrow \underbrace{\text{books}}_{\text{target}}$, $|V|$ labels

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

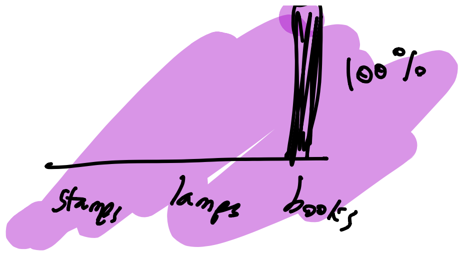
minimize negative log prob of "books"

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

why "cross-entropy" loss?



training data distribution p:



def of cross-entropy between p and q

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

$\hookrightarrow$ 1 when w = books
0 otherwise

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

neg. log prob. of the correct word

---

backpropagation: algorithm to compute gradient $\frac{dL}{d\theta}$ in an efficient manner

$$x \xrightarrow{W_1} h \xrightarrow{W_2} o$$

$\hookrightarrow$ model's prediction

training inputs: (x, y) e.g. $(3, 9.4)$

$\hookrightarrow$ target
$\hookrightarrow$ input

$h = \tanh(W_1 x)$
$o = \tanh(W_2 h)$

1. compute loss fn L

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

square loss / L2 loss
good for regression problems

$\hookrightarrow$ target $\hookrightarrow$ prediction

2. compute gradient

$$\frac{dL}{d\theta} : \left\{ \frac{dL}{dw_1}, \frac{dL}{dw_2} \right\} \quad \text{2 params}$$

Chain rule of calculus

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

$$L = \frac{1}{2}(y-o)^2$$
$$o = \tanh(a)$$
$$a = w_2 h$$
$$h = \tanh(b)$$
$$b = w_1 x$$

intermediate vars:

$$a = w_2 h$$
$$b = w_1 x$$

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

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

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

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

backpropagation: chain rule of calculus
+ caching prev. computed derivatives

3. updating params

$$w_{1_{new}} = w_{1_{OLD}} - \eta \frac{dL}{dw_1}$$

$$w_{2_{new}} = w_{2_{OLD}} - \eta \frac{dL}{dw_2}$$