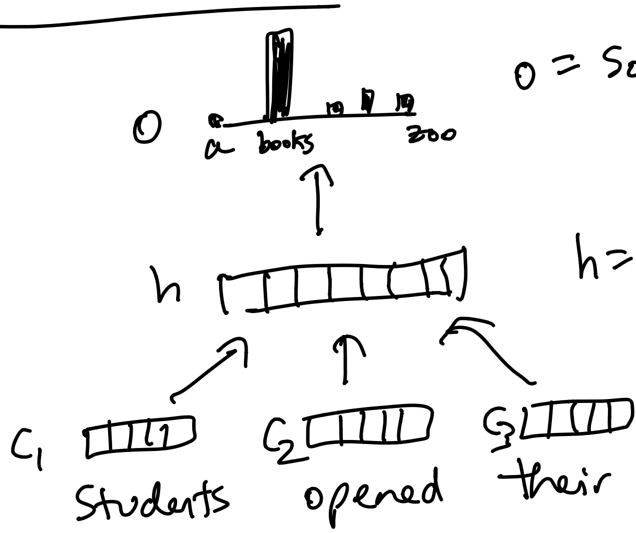
Training neural language models

- NLMs contain parameters (e.g. W_1, W_2, c_1, c_2, c_3 ...)
  - → params are randomly initialized
    - → thus, $p(w_n \mid w_1, w_2 \dots w_{n-1})$ is also random at the start
  - → by training the NLM, we adjust its params to maximize the likelihood of the training data

---



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

concatenation operation

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

params:

$$\theta = \{W_1, W_2, c_1, c_2, c_3\}$$

Steps to train an NLM:

1. define a loss function $L(\theta)$
   - → tells us how bad the model is at predicting the next word

→ smooth, differentiable

2. Given loss $L(\theta)$, we compute the gradient of L wrt θ.

↳ gradient gives us the direction of steepest ascent

↳ same dimensionality as θ

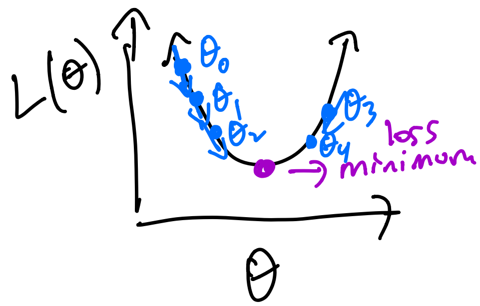
$$\frac{dL}{d\theta} = \left\{ \frac{dL}{dW_1}, \frac{dL}{dW_2}, \frac{dL}{dc_1}, \frac{dL}{dc_2} \dots \right\}$$

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

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

↳ minimize L

$$\theta_{new} = \theta_{OLD} - \underbrace{\eta}_{\text{learning rate "step size"}} \underbrace{\frac{dL}{d\theta}}_{\text{gradient}}$$



optimizer:

- SGD
- Adam (more common)
- Sophia (LLMs)

hyperparameters of gradient descent

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

Loss fn: cross-entropy loss

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

goal: maximize p("books" | "students opened their")

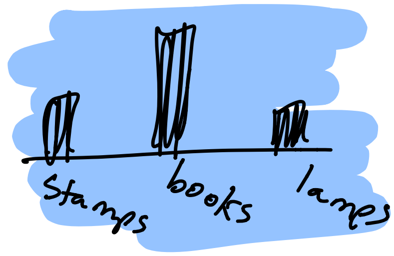
$\Rightarrow$ minimizing log prob of books | ...

$L = -\log(p(\text{books} \mid \text{prefix}))$

neg. log prob of the correct next token

why "cross entropy loss"?

NLM("students opened their") ⇒



model's predicted distribution q

training data distribution: ⇒

def of cross entropy

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

$p(w)$: 1 when w = books, 0 otherwise

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

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

input

$$x \xrightarrow{w_1} h \xrightarrow{w_2} o$$

model's prediction

training: (x, y) $(1.5, 3.4)$

input target

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

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

params: $\{w_1, w_2\}$

gradient $\left\{ \frac{dL}{dw_1}, \frac{dL}{dw_2} \right\}$

1. compute loss L

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

target prediction

square loss / L2 loss
regression problems

2. compute $\frac{dL}{dw_1}, \frac{dL}{dw_2}$

chain rule of calculus

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

$$\frac{dL}{dw_2}$$

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

intermediate vars:

$a = w_2 h$

$b = w_1 x$

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

$$\frac{dL}{dw_2} = \frac{dL}{do} \cdot \frac{do}{da} \cdot \frac{da}{dw_2}$$
$$-(y-o) \cdot (1-o^2) \cdot h$$

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

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

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

3. update params:

$$w_{1_{new}} = w_{1_{OLD}} - \eta \frac{dL}{dw_1} \quad \Big| \quad w_{2_{new}} = w_{2_{OLD}} - \eta \frac{dL}{dw_2}$$

Pytorch : model

loss = -log(model(books | " students opened their "))

loss.backwards()