Training neural language models

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

\[ o = \text{Softmax} (W_2 h) \]
\[ h = f(W_2 [c_1 ; c_2 ; c_3]) \]

Steps to train an NLM:
1. define a loss function \( L(\theta) \)
  - tells us how bad the model is
    at predicting the next word
2. Given loss $L(\theta)$, we compute the gradient of $L$ wrt $\theta$.

Gradient gives us the direction of steepest ascent.

$\mathbf{\theta}$ same dimensionality as $\mathbf{\theta}$

$$\frac{dL}{d\theta} = \sum \frac{dL}{d\mathbf{w}_i} \frac{d\mathbf{w}_i}{d\theta_1} \frac{d\mathbf{w}_2}{d\theta_2} \cdots$$

For each param $j$ in $\theta$,

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 to minimize $L$.

$$\mathbf{\theta}_{\text{new}} = \mathbf{\theta}_{\text{old}} - \eta \frac{dL}{d\theta}$$
$L(\theta) \uparrow \uparrow \text{loss} \downarrow \downarrow \rightarrow \text{minimum}$

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

hyperparameters of gradient descent
- learning rate $\eta$
- batch size
  - how many training examples do you use to estimate $\frac{dC}{d\theta}$ before taking a step.

Loss fn: cross-entropy loss

$\text{students opened their \Rightarrow books \over \text{target, } |V| \text{ labels}}$

training prefix

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

$\Rightarrow$ minimizing log prob of books $\ldots$

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

neg. log prob of the correct next token
Why "cross entropy loss"?

\[ NLM(" \text{students opened their}") \Rightarrow \]

![Diagram](image)

Model's predicted distribution

training data distribution: \( \equiv \)

![Diagram](image)

\[
\text{def of cross entropy} = - \sum_{w \in V} p(w) \log q(w)
\]

\[
\uparrow 1 \text{ when } w = \text{books} \\
0 \text{ otherwise}
\]

\[
= - \log q(\text{books} | "\text{students opened their}")
\]
**backpropagation**: algo to compute $\frac{dl}{d\theta}$ in an efficient manner

\[
\begin{align*}
\text{input: } & w_1 \quad w_2 \\
\text{model's prediction: } & \theta (x, y) \\
\text{target: } & (1.5, 3.4)
\end{align*}
\]

\[h = \tanh(w_1 x)\]
\[o = \tanh(w_2 h)\]
\[\text{params: } \theta_{w_1}, w_2\]
\[\text{gradient: } \frac{dl}{dw_1}, \frac{dl}{dw_2}\]

1. compute loss $L$
   \[L = \frac{1}{2} (y - o)^2\]
2. compute $\frac{dl}{dw_1}, \frac{dl}{dw_2}$

**chain rule of calculus**
\[
\frac{d}{dx} g(f(x)) = \frac{dg}{df} \cdot \frac{df}{dx}
\]

\[
\frac{dl}{dw_2}
\]
\[ L = \frac{1}{2} (y - o)^2 \]
\[ o = \tanh(a) \]
\[ a = w_2 h \]
\[ h = \tanh(b) \]
\[ b = \odot_1 x \]

Intermediate vals:
\[ a = w_2 h \]
\[ b = w_1 x \]
\[ \frac{d}{dx} \tanh(x) = 1 - \tanh^2(x) \]

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

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

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

\[ \text{backpropagation: chain rule of calculus, caching prev. computed derivatives} \]

3. update params:
\[ w_1_{new} = w_1_{old} - \eta \frac{dL}{dw_1} \]
\[ w_2_{new} = w_2_{old} - \eta \frac{dL}{dw_2} \]
Pytorch: model

\[
\text{loss} = \log(\text{model}(\text{books | "students opened their"}))
\]

loss.backward()