How do we train a neural language model?

\[ L \]

\( \text{how to adjust the params of our model to better predict the next word} \)

NLM (concatenation):

\[
\begin{align*}
\theta : & \quad \text{params } (W_1, W_2) \\
\Theta : & \quad (c_1, c_2, c_3)
\end{align*}
\]

\[
\begin{align*}
h = & \, f(W_1 [c_1, c_2, c_3]) \\
o = & \, \text{softmax}(W_2 h)
\end{align*}
\]

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

   \( L \) smooth, differentiable

2. Given \( L(\theta) \), we compute the gradient of \( L \) with respect to \( \theta \)

   \( \theta \) gradient gives us the direction
of steepest ascent of $L$

$L$) same dimensionality as $\Theta$

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

$L$) for concat LM

$$\frac{dL}{d\Theta} = \sum \frac{dL}{dW_1}, \frac{dL}{dW_2}, \frac{dL}{dW_3}...$$

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

$L$) this minimize $L$

$$\Theta_{new} = \Theta_{old} - \eta \frac{dL}{d\Theta}$$

$L$) gradient

$L$) learning rate

$L$) controls step size

optimizer:
- stochastic grad. descent
- Adam
- Adafactor
important hyperparams:
- learning rate $\eta$
- 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

$\Rightarrow$ cross-entropy loss

students opened their $\Rightarrow$ books

training prefix $\Rightarrow$ target, $|V|$ labels

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

minimize negative log prob of "books"

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

why "cross-entropy" loss?
def of cross-entropy between \( p \) and \( q \):

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

\( \Rightarrow 1 \) when \( w = \text{books} \)

\( \Rightarrow 0 \) otherwise

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

reg. log prob. of the correct word

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

\[
\begin{array}{ccc}
\odot & W_1 & W_2 \\
\odot & h & h \\
\odot & 0 & 0 \\
\end{array}
\]

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

\( 0 = \tanh(w_2 h) \)

1. compute loss function \( L \)

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

square loss / L2 loss

good for regression problems
2. Compute gradient

\[
\frac{dl}{d\theta} = \sum \frac{dl}{dw_1}, \frac{dl}{dw_2}
\]

Chain rule of calculus

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

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

Intermediate vars:

\[
\begin{align*}
& a = w_2 h \\
& b = w_1 x \\
& \frac{dl}{dw_2} = \frac{dl}{do} \cdot \frac{do}{da} \cdot \frac{da}{dw_2} \\
& \frac{dl}{dw_1} = \frac{dl}{do} \cdot \frac{do}{da} \cdot \frac{da}{dh} \cdot \frac{dh}{db} \cdot \frac{db}{dw_1}
\end{align*}
\]

Backpropagation: Chain rule of calculus + Caching prev. computed derivatives
3. updating params

\[ w_{1\text{new}} = w_{1\text{old}} - \eta \frac{\partial \ell}{\partial w_1} \]
\[ w_{2\text{new}} = w_{2\text{old}} - \eta \frac{\partial \ell}{\partial w_2} \]