- **Vanilla GD** \( L = \sum_{i=1}^{N} \ell(f(x_i; \theta), y_i) \)

- **Stochastic GD** \( L = \sum_{i=1}^{n} \ell(f(x_i; \theta), y_i) \) \( n \ll N \) = “minibatch”

- Divide up data into minibatches
- Evaluate gradient loss & gradient on each
- Update parameters mini-batch by mini-batch

- Minibatch size \( n \) is another key hyperparameter!
- Some tradeoff btw \( n \) & \( y \)

- Loopy over full data once = “1 epoch”

---

Further improvements to SGD:

- “Momentum” SGD with fixed \( y \) could mean
  - steepest = “raving”
  - shallow = “small”

  Can lead to slow convergence
  - Allowing shallow direction

  Idea: “Momentum” like in physics = acceleration
\[
\delta \theta = v + \gamma \frac{\nabla f(\theta) - \nabla f(\theta + 1)}{\nabla f(\theta)}
\]

Push parameters in direction of previous gradient
- if gradients oscillate (bouncy and steep direction)
  momentum cancels out
- if gradients pointed along same dir \( \pm 1 \),
  then they add \( \rightarrow \) acceleration

- Adaptive Learning Rates
  Idea: large gradients \( \rightarrow \) slow down
  small gradients \( \rightarrow \) speed up

\[
\delta \theta = -\eta \frac{\nabla f(\theta)}{\sqrt{\sum_{i} \delta \theta_{i}^2}} \quad G \sim \Sigma (\frac{\partial f(\theta)}{\partial \theta})^2 \quad \text{timesteps}
\]

"Adadelta" & "Adam" the popular examples

restrict \( \eta \) to finite
\# of time steps or decaying avg.

most common choice (\~Adadelta w/momentum)
Backpropagation

A brief aside on how gradients of NNs are actually calculated (motivated by explaining vanishing gradient problem)

NNs have a recursive structure

\[ f^{(L)}(x) = A (w_L A (w_{L-1} A \ldots A (w_1 x)) \ldots) \]

\[ L = \sum L(f(x; w), x_i) \]

\[ \frac{\partial L}{\partial w} \rightarrow \sum \frac{\partial f^{(L)}}{\partial w} \]

\[ f^{(L)}(x) = x_L = A(w_L x_{L-1}) \]

\[ x_{L-1} = A(w_{L-1} x_{L-2}) \]

\[ x_{L-2} = A(w_{L-2} x_{L-3}) \]

\[ \ldots \]

\[ x_0 = x \]

\[ x_1 = A(w_1 x) \]

\[ x_2 = A(w_2 x_1) \]

\[ x_3 = A(w_3 x_2) \]

\[ \ldots \]

\[ x_L = A(w_L x_{L-1}) \]

\[ \sum \frac{\partial f^{(L)}}{\partial w} \]

\[ \frac{\partial f^{(L)}}{\partial w} = A'(w_{L-1} x_{L-1}) x_{L-1} \]

\[ 3 \text{ mm} \]

\[ \frac{\partial f^{(L)}}{\partial w_{L-1}} = A'(w_{L-1} x_{L-1}) w_L A'(w_{L-2} x_{L-2}) x_{L-2} \]

\[ 5 \text{ mm} \]

\[ \frac{\partial f^{(L)}}{\partial w_{L-2}} = A'(w_{L-2} x_{L-2}) w_L A'(w_{L-3} x_{L-3}) w_{L-1} A'(w_{L-4} x_{L-4}) x_{L-4} \]

\[ 7 \text{ mm} \]

\[ \vdots \]

To compute all gradients, need \( O(L^2) \) matrix multiplications — expensive!

Backprop: reuse matrix multiplications \( L \rightarrow L-1 \rightarrow L-2 \ldots \)
Let $G_{\theta_{L}} = A'(w_{L-1}x_{L-1})w_{L-1}A'(w_{L-2}x_{L-2})...w_{2}A'(w_{1}x_{1})$

So

$$\frac{\partial f_{NN}}{\partial w_{L}} = G_{L}x_{L-1}$$
$$\frac{\partial f_{NN}}{\partial w_{L-1}} = G_{L-1}x_{L-2}$$

- have it from forward pass
- reduced to
- each step only needs 2 matrix multiplications $\rightarrow O(L)$ needed

- $\Rightarrow$ Backprop was key step (seems obvious)
- in making NNs trainable!

- also see gradients are products of many $A'(\cdot)$ derivatives!

- gradients of earlier layer weights can be very suppressed or enhanced
  “vanishing/exploding gradient problem”

- especially bad for sigmoid
- ameliorated by ReLU
One last thing to discuss before we can do a demo of NN training:

**Overfitting & train/test split**

How do we detect & prevent overfitting? Or underfitting?

"Variance" vs "Bias" generalization error

"Bias-variance tradeoff"  

Want model to generalize well to unseen data following same distr. as training data

- Set aside validation set

Loss vs Val vs Train

Stop training! "Early stopping" or "model selection"
Then possibly biased on val set

→ final metrics on another holdout dataset

"test set"