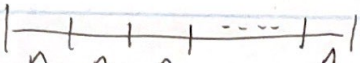
- Vanilla GD $L = \sum_{i=1}^{N} \mathcal{L}(f(x_i;\theta), x_i)$
- Stochastic GD $L = \sum_{i=1}^{n} \mathcal{L}(f(x_i;\theta), x_i)$

trainy data




$n << N$
"minibatch"

- divide up data into minibatches
- evaluate loss & gradients on each
- update parameters minibatch by minibatch

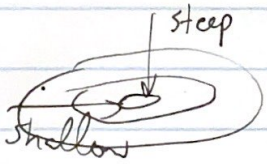
minibatch size $n$ is another key hyperparameter!
Some tradeoff btw $n$ & $\eta$

- loopy over full data once = "1 epoch"

---

Further improvements to SGD:

- "momentum" SGD w/ fixed $\eta$ could mean $\eta$ too large for large gradients "small" small "




- can lead to slow convergence allow shallow direction

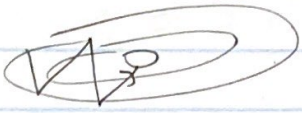
- idea: "momentum" like in physics → acceleration

$$v_t = \gamma v_{t-1} - \eta \frac{\partial L}{\partial \theta}$$

$$\delta\theta = v_t$$

Push parameters in direction of previous gradient

- if gradients oscillate (bouncy around steep direction), momentum cancels out
- if gradients pointed along same dir $t-1, t$ then they add → acceleration



- Adaptive learning rates

  idea: large gradients → slow down
  small gradients → speed up

  + diff learning rates ∀ parameter

$$\delta\theta_i = -\frac{\eta}{\sqrt{G_i}} \frac{\partial L}{\partial \theta_i} \qquad G \sim \sum_{\text{timesteps}} \left(\frac{\partial L}{\partial \theta_i}\right)^2$$

"Adadelta" ← restrict $G$ to finite # of time steps or decaying avg

& "Adam" two popular examples → 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

nested (ignore $b_i$ as for simplicity)

$$f_{NN}(x) = A\left(w_L \cdot A\left(w_{L-1} A\left(\cdots A(w_1 x)\right)\cdots\right)\right)$$

$$L = \sum \mathcal{L}(f_{NN}(x_i; w), x_i)$$

$$\frac{\partial L}{\partial w} \rightarrow \frac{\partial f_{NN}}{\partial w}$$

$$\left.\begin{aligned} x_0 &= x \\ x_1 &= A(w_1 x_0) \\ x_2 &= A(w_2 x_1) \\ x_{L-1} &= A(w_{L-1} x_{L-2}) \\ f_{NN}(A) = x_L &= A(w_L x_{L-1}) \end{aligned}\right\} \text{ forward pass}$$

just chain rule!

1 nn $\quad \dfrac{\partial f_{NN}}{\partial w_L} = A'(w_L x_{L-1})\, x_{L-1}$

3 nn $\quad \dfrac{\partial f_{NN}}{\partial w_{L-1}} = A'(w_L x_{L-1})\, w_L\, A'(w_{L-1} x_{L-2})\, x_{L-2}$

5 nn $\quad \dfrac{\partial f_{NN}}{\partial w_{L-2}} = A'(w_L x_{L-1})\, w_L\, A'(w_{L-1} x_{L-2})\, w_{L-1}\, A'(w_{L-2} x_{L-3})\, x_{L-3}$

$\vdots$

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

Backprop: reuse matrix multiplications $L \rightarrow L-1 \rightarrow L-2 \ldots$

Let $G_{\theta_h} = A'(w_L x_{L-1}) w_L A'(w_{L-1} x_{L-2}) \cdots w_{h+1} A'(w_h x_{h-1})$

So $\frac{\partial f_{NN}}{\partial w_L} = G_L x_{L-1}$

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

$\vdots$

have it from forward pass

$$G_{\theta_h} = G_{\theta_{h+1}} w_{h+1} A'(w_h x_{h-1})$$

each step only needs 2 matrix multiplications → $O(L)$ reduced to needed

→ Backprop was key step (seems obvious) in making layer NNs trainable!

→ also see gradients are products of many $A'(\cdot\cdot)$ & derivatives! wrt pos

→ gradients of earlier layer weights can be very suppressed or enhanced "vanishy/explody gradient problem"

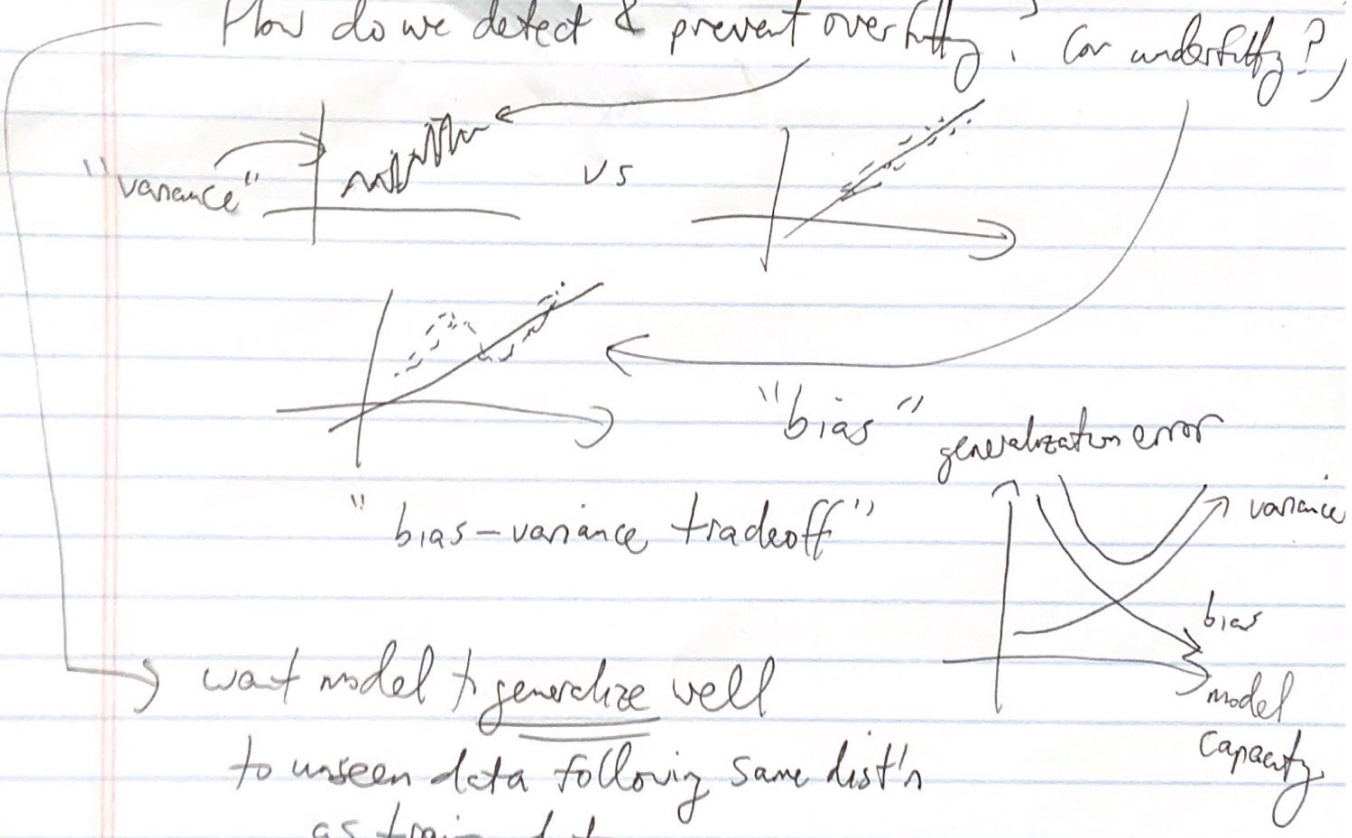
→ especially bad for sigmoid ameliorated by ReLU

Concept to introduce

One last ~~thing to discuss~~ before we can do a demo of NN training:

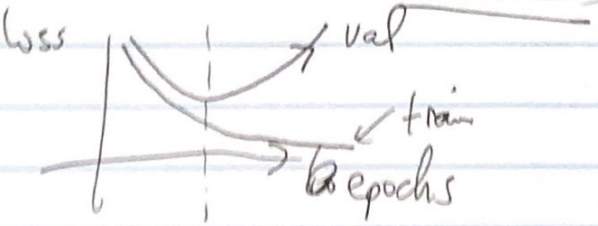
## Overfitting & train/val/test split

How do we detect & prevent overfitting? Can underfitting?




→ want model to generalize well to unseen data following same dist'n as training data

→ set aside validation set




stop training! "early stopping" or "model selection"

Then possibly biased on val set

→ final metrics on another held out dataset

"test set"