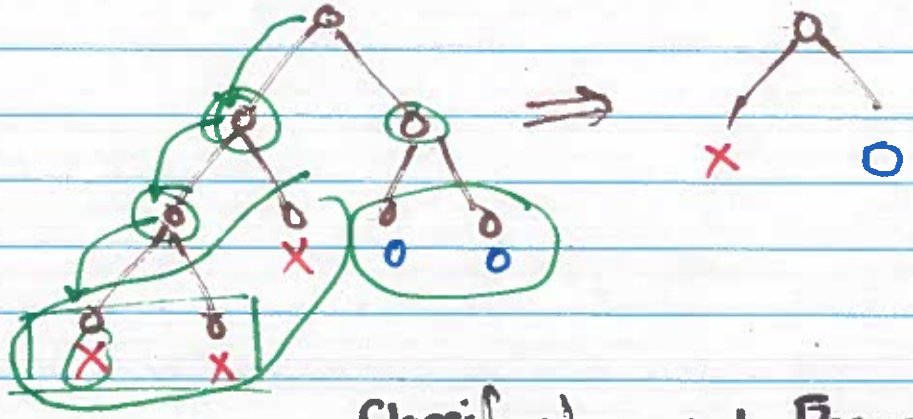
Approx K-NN: No backtrack



Classification and Regression Tree

CART

Goal: build a tree that's
(1) as small as possible, and
(2) only has 1 label per leaf node ("pure" leaves)

NP-Hard.

Idea: split recursively to minimize "impurity" at each node of tree.
heuristic that measures how "diverse" the labels are in the dataset.

Gini Impurity. (Classification)

Empirical distribution over labels

$$p_k = \frac{|\{(x,y) \in S \mid y = k\}|}{|S|} = \frac{\text{\# of examples of label } k}{\text{\# of examples.}}$$

$$\text{Gini} \Rightarrow G(S) = \sum_k p_k (1 - p_k)$$

\# of ways to draw 2 examples with unequal labels

$$\sum_k \sum_{\ell \neq k} |\{(x,y) \in S \mid y = k\}| \cdot |\{(x,y) \in S \mid y = \ell\}|$$

(\# with label k) · (\# with label ℓ)

$$= \sum_k (\text{\# with label } k) \cdot \sum_{\ell \neq k} (\text{\# with label } \ell)$$

$$\sum_{\ell \neq k} (\text{\# with label } \ell) = n - (\text{\# with label } k)$$

$$= \sum_k (\text{\# with label } k) \cdot (n - (\text{\# with label } k))$$

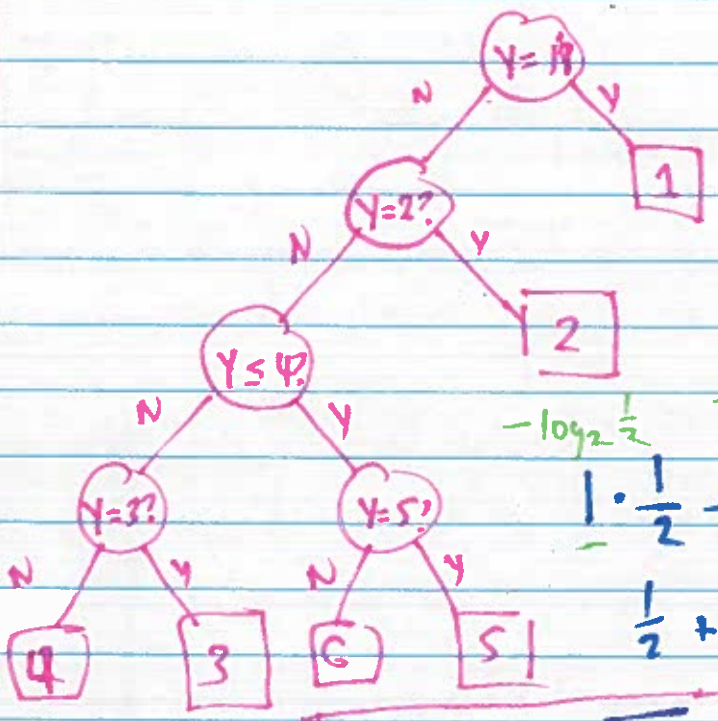
$$= \sum_k (n p_k) \cdot (n - (n \cdot p_k)) = n^2 \cdot \sum_k p_k (1 - p_k)$$

$$G^T(S) = \frac{|S_L|}{|S|} G^T(S_L) + \frac{|S_R|}{|S|} G^T(S_R)$$

S_L S_R

Entropy.

$P_1 = \frac{1}{2}$ $P_2 = \frac{1}{4}$, $P_3 = P_4 = P_5 = P_6 = \frac{1}{16}$



$-\log_2 \frac{1}{2}$ $-\log_2 \frac{1}{4}$ $-\log_2 \frac{1}{16}$

$1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 4 \cdot \left(\frac{4}{16}\right)$

$\frac{1}{2} + \frac{1}{2} + 1 = 2.$

$$H(S) = \sum_k -P_k \log_2 P_k$$

$H(S)$ is a monotonic function of $D_{KL}(p \| \text{uniform})$

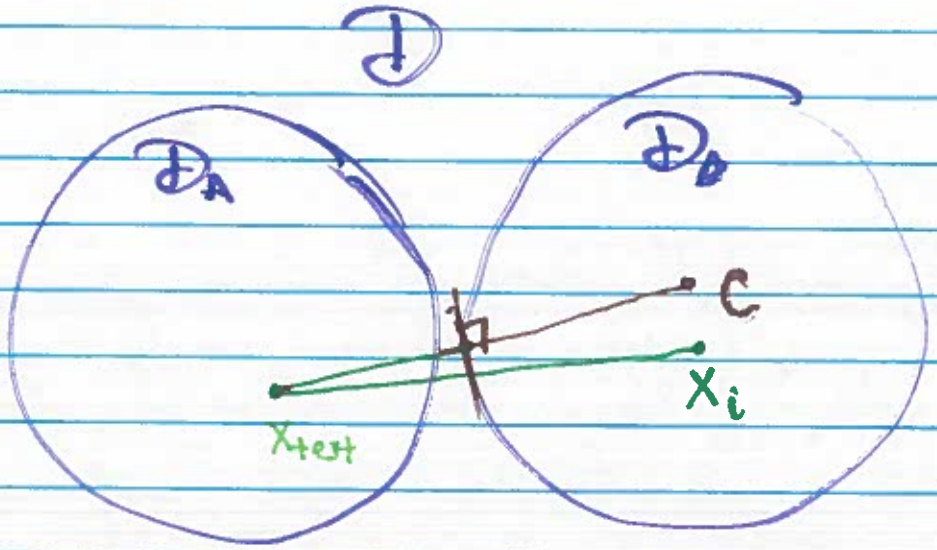
$D_{KL}(p\|q) =$

$$H^T(S) = \frac{|S_L|}{|S|} \cdot H^T(S_L) + \frac{|S_R|}{|S|} \cdot H^T(S_R)$$

$$D_{KL}(p\|q) = \sum_k p_k \log\left(\frac{q_k}{p_k}\right)$$ H

$$D_{KL}(p \| \text{uniform}) = \sum_k p_k \log\left(\frac{1/c}{p_k}\right) = \sum_k p_k \log \frac{1}{c} - \sum_k p_k \log p_k$$

Ball Trees



Impurity

- Gini impurity ≈ how often will random labels disagree
- Entropy ≈ information
- Least squares

ID3

Input: Dataset D

Search all features:

  search all splits for feature:

    evaluate impurity (entropy)

pick feature/split that minimizes impurity

construct node

recursively call ID3 on subsets

How to split?

- categorical features → 1 child per category split
- real-valued features → threshold

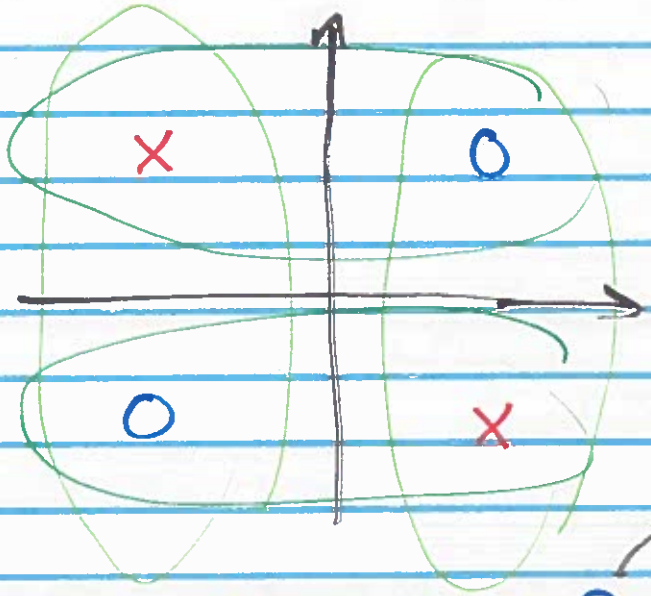
Base cases: if $|D| = 1$

classification → majority

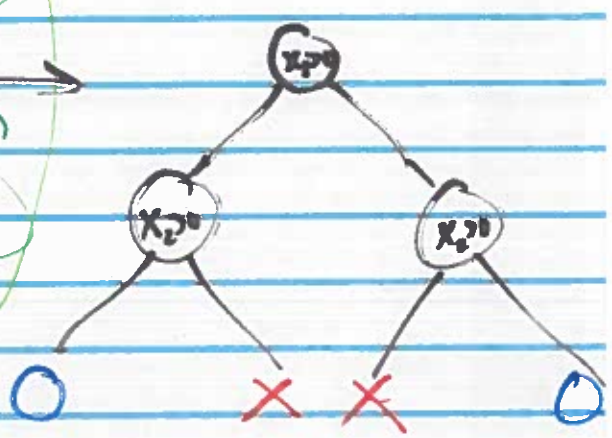
regression → average

1. ∃ exists some $\hat{x}$ s.t. $\forall (x,y) \in D$, $x = \hat{x}$
2. exists $\hat{y}$ s.t. $\forall (x,y) \in D$, $y = \hat{y} \Rightarrow$ leaf $\hat{y}$
3. $|D| = 0$, predict majority/average of "parent" dataset

Why not stop when impurity doesn't decrease.



$P_X = \frac{1}{2}$ $P_O = \frac{1}{2}$



Decision tree inference time is proportional to depth

⇓

inference very fast!

Overfitting:

⇑

$\text{bias}^2 + \text{variance} + \text{noise}$

Ensembling: average the prediction of some models

- draw m independent datasets $D_1, \dots D_m$
- for each dataset:
  - run ID3 $\rightarrow$ hypothesis h_i
- output $\hat{h}(x) = \frac{1}{m} \sum_{i=1}^{m} h_i(x)$

How does this effect: bias^2 variance noise?

same ↓ factor of m same

$$\text{Var}\left(\frac{1}{m}\sum_{i=1}^{m} h_i(x)\right) = \frac{1}{m^2}\text{Var}\left(\sum_{i=1}^{m} h_i(x)\right)$$

$$= \frac{1}{m^2}\sum_{i=1}^{m}\text{Var}(h_i(x))$$

$$= \frac{1}{m^2} \cdot m \cdot \text{Var}(h_i(x))$$

$$= \frac{1}{m}\text{Var}(h_i(x))$$

Bootstrap Aggregating

instead of drawing from source dist, we draw with replacement from $\mathcal{D}$

given $\mathcal{D}$ ~~subsampled~~
~~draw m init datasets~~
draw m datasets from $\mathcal{D}$ each of size N
for each, $\mathcal{F}$ train a decision tree (ID3)
average: $\hat{h}(x) = \frac{1}{m}\sum_{i=1}^{m} h_i(x)$.

still reduces variance!
- idea: individual examples are still distributed according to $\mathcal{P}$ (source dist)
- even though they're not independent, they're "independent enough" to reduce variance.

Random forest ⇒ full algorithm Bagging Trees

General Gradient Boosted Regression Trees

Input: loss l, α step size, $\mathcal{D}$, CART alg

Init: $H_0 = 0$; $H(x) = 0$

loop from $t = 1 : T$

$r_i = l'(H_{t-1}(x_i), y_i)$ for all $i \in \{1, \ldots n\}$

$h_t = \text{CART}(\{(x_1, -r_1), (x_2, -r_2), \ldots (x_n, -r_n)\})$

if $\sum_{i=1}^{n} r_i h(x_i) < 0$: → update ensemble by adding h

$H_T(x) = H_{T-1}(x) + \alpha h(x)$

else :

halt and return H_{t-1}.

end loop

return H_T.

could replace with $l(H_{T-1} + \alpha h) < l(H_{T-1})$

Random forest

Bagging + CART + Subsampling features

// each node searches all features & splits

= only search k features at random

increases "diversity" of trees

set $k \approx \sqrt{d}$

Hyper params: $k \Leftarrow$ # of features we search

$m \Leftarrow$ # of datasets/trees $\Rightarrow$ set as large as possible

sampled with replacement

1. Sample m "bagged" datasets from $\mathcal{D}$
   call $\mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_m$
2. for each bagged dataset $\mathcal{D}_i$
   run a variant of CART/ID3
   $\Rightarrow$ at each split choose k features at random and only consider splitting those
   outputting hypothesis h_i
3. Output random forest classifier/regressor

$$\hat{h}(x) = \frac{1}{m} \sum_{i=1}^{m} h_i(x).$$

Two Variants of Rf:

- build tree using one dataset and label the leaves with another independent dataset (prove consistency)

- build smaller trees (not to full depth) (pruning)

sequential parallel

Boosting vs Bagging

- sequentially call our tree-building algo, at each step building a tree that improves the ensemble

Goal: create an ensemble classifier/regressor

$$H_T(x) = \sum_{t=1}^{T} \alpha_t h_t(x) = H_{T-1}(x) + \alpha_T h_T(x)$$

H_T: ensemble after T steps; α_t: scalar weights; h_t: hypothesis learned at step t

Goal: minimize loss function

$$\ell(H) = \frac{1}{n} \sum_{i=1}^{n} \ell(H(x_i), y_i)$$

the hypothesis I pick at step t+1 is

$$h_{t+1} = \arg\min_{h \in \mathcal{H}} \ell(H_T + \alpha h)$$

for simplicity imagine fixed α

$$= \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(H(x_i) + \alpha h(x_i), y_i)$$

$$\approx \ell(H(x_i), y_i) + \langle \alpha h(x_i), \nabla \ell(H(x_i), y_i) \rangle$$

$$\approx \ell(H(x_i), y_i) + \alpha \langle h(x_i), \nabla \ell(H(x_i), y_i) \rangle$$

$$= \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \langle h(x_i), \nabla \ell(H(x_i), y_i) \rangle$$

$$= \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} h(x_i)\, \ell'(H(x_i), y_i)$$

$$= \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} h(x_i) \frac{\partial \ell(H(x_i), y_i)}{\partial [H(x_i)]}$$

let $r_i = \frac{\partial \ell(H(x_i), y_i)}{\partial [H(x_i)]} = \ell'(H(x_i); y_i)$

$$= \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} h(x_i) \cdot r_i$$

($h(x_i) \in \mathbb{R}$, $r_i \in \mathbb{R}$)

if this is negative, then $\ell(H_{T+1}) \leq \ell(H_T)$

Generic boosting (a.k.a. "AnyBoost")

1. Assume that hypothesis h has constant $\sum_{i=1}^{n} (h(x_i))^2$. e.g. if $h(x_i) \in \{+1, -1\}$.

$$= \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2}(h(x_i) + r_i)^2 - \frac{1}{2}(h(x_i))^2 - \frac{1}{2} r_i^2$$

$ab = \frac{1}{2}(a+b)^2 - \frac{1}{2}a^2 - \frac{1}{2}b^2$

$t_i = -r_i = -\ell'(H(x_i), y_i)$

$$= \arg\min_{h \in \mathcal{H}} \frac{1}{2n} \sum_{i=1}^{n} (h(x_i) - (-r_i))^2$$

$(h(x_i) - t_i)^2$

$\approx$ CART$((x_1, t_1), (x_2, t_2), \ldots (x_n, t_n))$

AdaBoost

Assumption: binary classification ($y_i \in \{+1, -1\}$)

→ weak learners also binary ($h_i(x) \in \{+1, -1\}$)

→ "step size" α set by line search
set optimally to minimize loss

→ loss Exponential loss $\ell(H) = \sum_{i=1}^{n} \exp(-y_i H(x_i))$

single ex loss: $\ell(H(x); y) = \exp(-y\, H(x))$

$$\ell'(H(x); y) = -y \exp(-y\, H(x))$$

Boosting goal: $\arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} \frac{\ell(H(x_i) + \alpha h(x_i); y_i) - \ell(H(x_i))}{\ell'(H(x_i); y_i)} \cdot h(x_i)$

$$= \arg\min_{h} -\sum_{i=1}^{n} y_i \exp(-y_i H(x_i)) \cdot h(x_i)$$

let $w_i = \frac{1}{Z} \exp(-y_i H(x_i))$, $Z = \sum_{i=1}^{n} \exp(-y_i H(x_i))$

$$= \arg\min_{h} -\sum_{i=1}^{n} w_i \underbrace{y_i}_{\text{binary} \in \{-1,1\}} \underbrace{h(x_i)}_{h \in \{-1,1\}}$$

$$= \arg\min_{h} \sum_{i | y_i \neq h(x_i)} w_i - \sum_{i | y_i = h(x_i)} w_i$$

$$= \arg\min_{h} \sum_{i | y_i \neq h(x_i)} w_i - \left(1 - \sum_{i | y_i \neq h(x_i)} w_i\right)$$

$$= \arg\min_h \; 2 \sum_{i | y_i \neq h(x_i)} w_i - 1$$

$$= \arg\min_h \sum_{i | y_i \neq h(x_i)} w_i$$ ← weighted classification error of hypothesis h

Choosing α:

$$\text{pick } \alpha = \arg\min_\alpha L(H + \alpha h)$$

$$= \arg\min_\alpha \sum_{i=1}^{n} \exp\left(-y_i \left(H(x_i) + \alpha h(x_i)\right)\right)$$

$$= \arg\min_\alpha \sum_{i=1}^{n} \exp(-y_i H(x_i)) \exp(-y_i h(x_i) \alpha)$$

$$= \arg\min_\alpha \sum_{i=1}^{n} w_i \exp(-y_i h(x_i) \alpha)$$

$$= \arg\min_\alpha \underbrace{\sum_{i | y_i = h(x_i)} w_i \exp(-\alpha)}_{1-\varepsilon} + \underbrace{\sum_{i | y_i \neq h(x_i)} w_i \exp(+\alpha)}_{\varepsilon}$$

let $\varepsilon = \sum_{i | y_i \neq h(x_i)} w_i$ ← weighted class error of h

$$= \arg\min_\alpha \; (1-\varepsilon) \exp(-\alpha) + \varepsilon \exp(+\alpha)$$

$$= \alpha = \frac{1}{2} \log\left(\frac{1-\varepsilon}{\varepsilon}\right)$$

Finishing up AdaBoost:

- incrementally update w_i

$$w_i' = \frac{1}{Z'} \exp\left(-y_i\left(H_s(x_i) + \alpha h(x_i)\right)\right)$$

$$= \frac{Z}{Z'} w_i \cdot \exp\left(\underbrace{-y_i h(x_i)\alpha}_{+\alpha \quad -\alpha}\right) = \frac{w_i \exp(-\alpha y_i h(x_i))}{2\sqrt{\varepsilon(1-\varepsilon)}}$$

$$Z' = \sum_{i=1}^{n} \exp\left(-y_i\left(H(x_i) + \alpha h(x_i)\right)\right)$$

$$= \sum_{i=1}^{n} Z \cdot w_i \cdot \exp\left(-\alpha y_i h(x_i)\right)$$

$$= Z \cdot \left(\sum_{i \mid y_i \neq h(x_i)} w_i \exp(\alpha) + \sum_{i \mid y_i = h(x_i)} w_i \exp(-\alpha) \right)$$

ε $\qquad$ $1-\varepsilon$

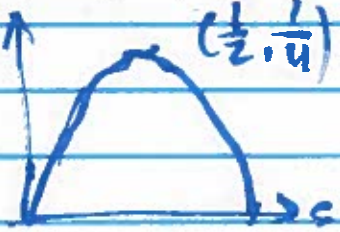
$$= Z \cdot \left(\varepsilon \exp(\alpha) + (1-\varepsilon)\exp(-\alpha)\right)$$

recall: $\alpha = \frac{1}{2}\log\left(\frac{1-\varepsilon}{\varepsilon}\right) \quad \exp(\alpha) = \sqrt{\frac{1-\varepsilon}{\varepsilon}}$

$$Z' = Z \cdot \left(\varepsilon \cdot \sqrt{\frac{1-\varepsilon}{\varepsilon}} + (1-\varepsilon)\cdot\sqrt{\frac{\varepsilon}{1-\varepsilon}}\right)$$

$$= Z \cdot \left(2\sqrt{\varepsilon \cdot (1-\varepsilon)}\right)$$

$$\ell(H + \alpha h) = \ell(H) \cdot \underbrace{\left(2\sqrt{\varepsilon(1-\varepsilon)}\right)}_{\leq 1}$$



if we halt whenever the classifier h has weighted test error ε too close to $\frac{1}{2}$, i.e. suppose/require that $\varepsilon \leq \frac{1}{2} - \gamma$

then: $2\sqrt{\varepsilon(1-\varepsilon)} \leq 2\sqrt{(\frac{1}{2}-\gamma)(\frac{1}{2}+\gamma)}$

$$= 2\left(\sqrt{\frac{1}{4} - \gamma^2}\right)$$

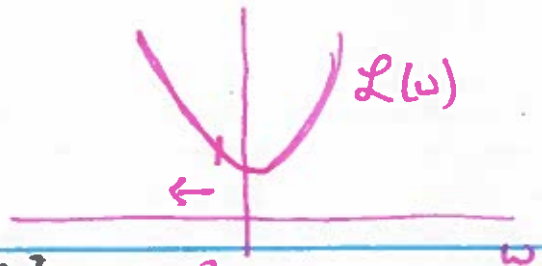
$$= \sqrt{1-4\gamma^2}$$

$$\ell(H + \alpha h) \leq \ell(H) \cdot \sqrt{1-4\gamma^2}$$

after we add k trees to the ensemble:

$$\ell(H) \leq \ell(0) \cdot (1-4\gamma^2)^{k/2} \leq n \cdot (1-4\gamma^2)^{k/2}.$$

if γ constant, need $O(\log n)$ trees to achieve small loss



e.g. $\mathcal{L}(w) = f_1(w) + f_2(w)$

$= (w-1)^2 + (w+1)^2 = 2w^2 + 2$

$w = \frac{1}{2}$

$\mathcal{L}'(w) = 4w$ $\quad f_1'(w) = 2(w-1)$

$\mathcal{L}'(w) f_1'(w) = 8w(w-1) \neq 0$

Why SGD: (x_i, y_i) = example sel'd at time t

$$\mathcal{L}(w_{t+1}) = \mathcal{L}(w_t) - \alpha \nabla \mathcal{L}(w_t)^T \nabla \ell(h(x_i; w_t); y_i) + O(\alpha$$

$$\mathbb{E}[\mathcal{L}(w_{t+1})] = \mathcal{L}(w_t) - \mathbb{E}[\alpha \nabla \mathcal{L}(w_t)^T \nabla \ell(h(x_i; w_t); y_i)] + O(\alpha$$

$$= \mathcal{L}(w_t) - \alpha \nabla \mathcal{L}(w_t)^T \mathbb{E}[\nabla \ell(h(x_i; w_t); y_i)] + O(\alpha^2)$$

$$\mathbb{E}[\nabla \ell(h(x_i; w_t); y_i)] = \sum_{i=1}^{n} \left(\frac{1}{n}\right) \nabla \ell(h(x_i; w_t); y_i)$$

$$= \nabla_w \left(\frac{1}{n} \sum_{i=1}^{n} \ell(h(x_i; w_t); y_i) \right)$$

$$= \mathcal{L}(w_t)$$

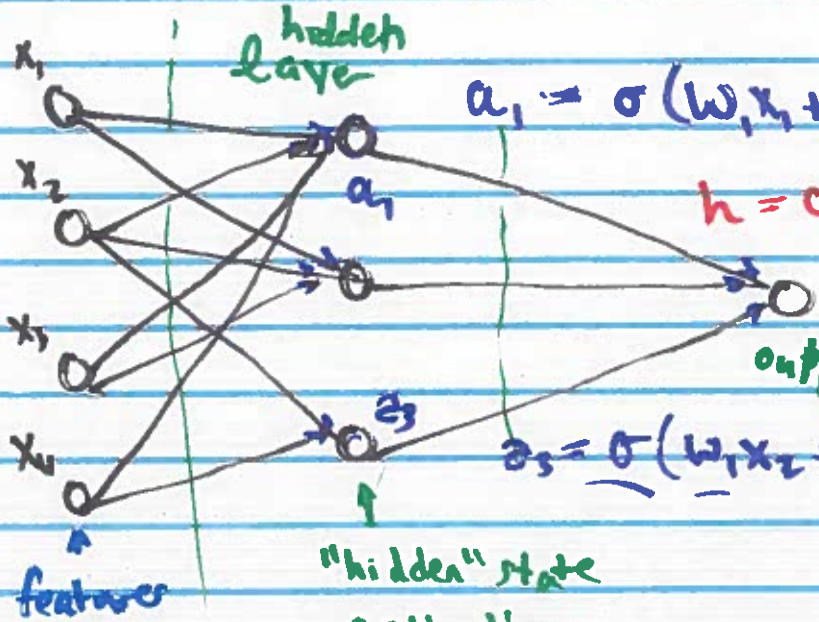
$$= \mathcal{L}(w_t) - \alpha \nabla \mathcal{L}(w_t)^T \nabla \mathcal{L}(w_t) + O(\alpha^2)$$

$$\nabla \mathcal{L}(w_t)^T \nabla \mathcal{L}(w_t) = \|\nabla \mathcal{L}(w_t)\|^2 \geq 0$$

SGD vs GD:

* SGD is faster per iteration
* SGD generalizes better!
* SGD helps avoid local minimal

Neural Networks a.k.a "Multilayer Perceptron"



σ acts elementwise

$a \in \mathbb{R}^3$ $\quad a = \sigma(Wx + b)$

$\mathbb{R}^{3\times 4}$ $\quad \mathbb{R}^3$

$a_1 = \sigma(w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + w_{14}x_4 + b_1)$

$a_2 = \sigma(\cdots)$

$a_3 = \sigma(\cdots)$

$h = \sigma(u_1 a_1 + u_2 a_2 + u_3 a_3 + c)$

$h(x) = \sigma(u^T \sigma(Wx + b) + c)$

$\mathbb{R}^3 \quad \mathbb{R}^{3\times 4} \quad \mathbb{R}^3 \quad \mathbb{R}$

$h(x) = u^T \phi(x) + c$, where $\phi(x) = \sigma(Wx + b)$

What is σ?

- ReLU: Rectified linear unit.

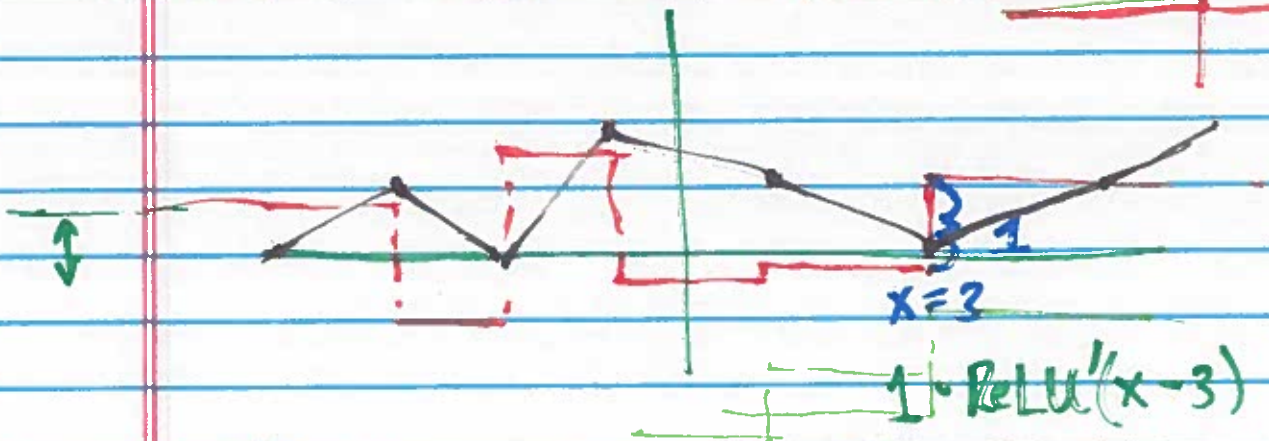
$$\text{ReLU}(x) = \sigma(x) = \max(0, x)$$

$$= \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

ReLU networks (nets where all nonlinearities are ReLU) are piecewise linear.

& all continuous piecewise linear fns on bounded domain can be expressed as a ReLU network.

Why does this hold? $\text{ReLU}'(x) =$



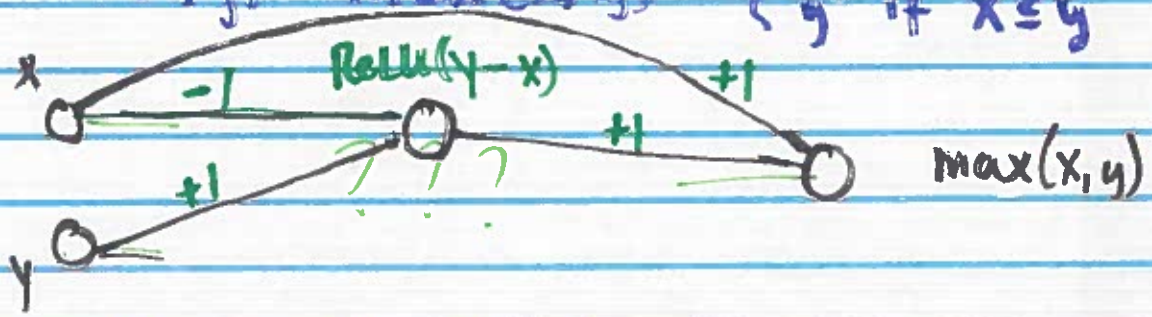
$$f'(x) = c_0 + a_1 \text{ReLU}'(x - b_1) + a_2 \cdot \text{ReLU}'(x - b_2) \cdots$$

$$f(x) = c_0 x + c_1 + a_1 \text{ReLU}(x - b_1) + a_2 \text{ReLU}(x - b_2) \cdots$$

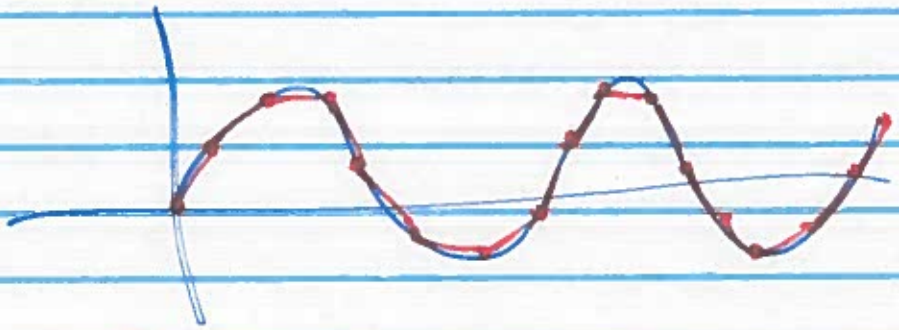
Let $f(x) = |x| = \begin{cases} -x & \text{if } x \le 0 \\ +x & \text{if } x \ge 0 \end{cases}$

$f(x) = -x + 2\,\text{ReLU}(x)$

Let $f(x,y) = \max(x,y) = \begin{cases} x & \text{if } x \ge y \\ y & \text{if } x \le y \end{cases}$



$f(x,y) = \text{ReLU}(y-x) + x$



any function ≈ some piecewise linear function ≡ ANN + ReLU

"universality" → approx any function!

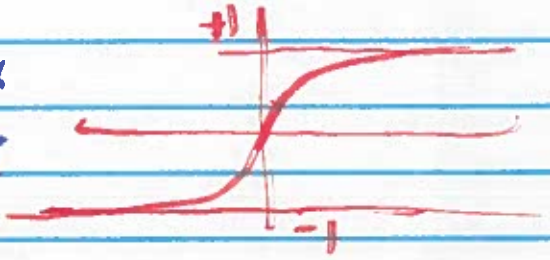
choose "architecture" + learn "weights/parameters"

$w_1, w_2, \ldots b_1 \ldots$

Common Nonlinearities

- ReLU

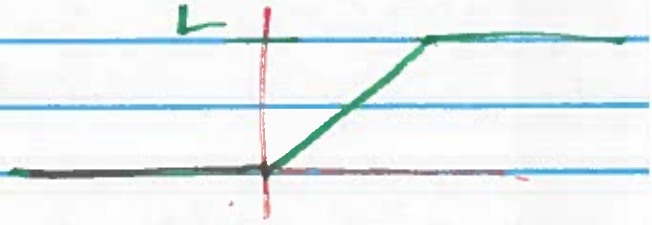
- $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



- $\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$

- "smoothed" ReLU

- "saturating" ReLU



$$\sigma(x) = \max(0, \min(x, L))$$

How to learn a DNN:

STOCHASTIC Gradient descent!) using a subsample of $\mathcal{D}$ to compute gradients at each step

$$\min_w \sum_{i=1}^{n} \ell(h(x_i; w); y)$$

+ automatic differentiation

* not numerical differentiation

$$f'(x) \approx \frac{f(x+0.01) - f(x)}{0.01}$$

* not symbolic diff

* about the same time as computing f to compute ∇f

→ BACKPROPAGATION

GD: $w \leftarrow w - \alpha \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(h(x_i; w), y_i)$

SGD: pick i from $\{1 \dots n\}$ unif at random

$w \leftarrow w - \alpha \nabla \ell(h(x_i; w); y_i)$

$$\mathcal{L}(\underbrace{w - \alpha \nabla \mathcal{L}(w)}_{\text{next } w}) = \mathcal{L}(w) - \alpha \nabla \mathcal{L}(w)^T \nabla \mathcal{L}(w) + O(\alpha^2)$$

$\nabla \mathcal{L}(w)^T \nabla \mathcal{L}(w) = \|\nabla \mathcal{L}(w)\|^2 \geq 0$

$$\mathcal{L}(w - \nabla \ell(h(x_i; w), y_i)) = \mathcal{L}(w) - \alpha \nabla \mathcal{L}(w)^T \nabla \ell(h(x_i; w), y_i) + O(\alpha^2)$$

Convolutional Neural Network.

Image is what? Multidim array $(3 \times W \times H)$

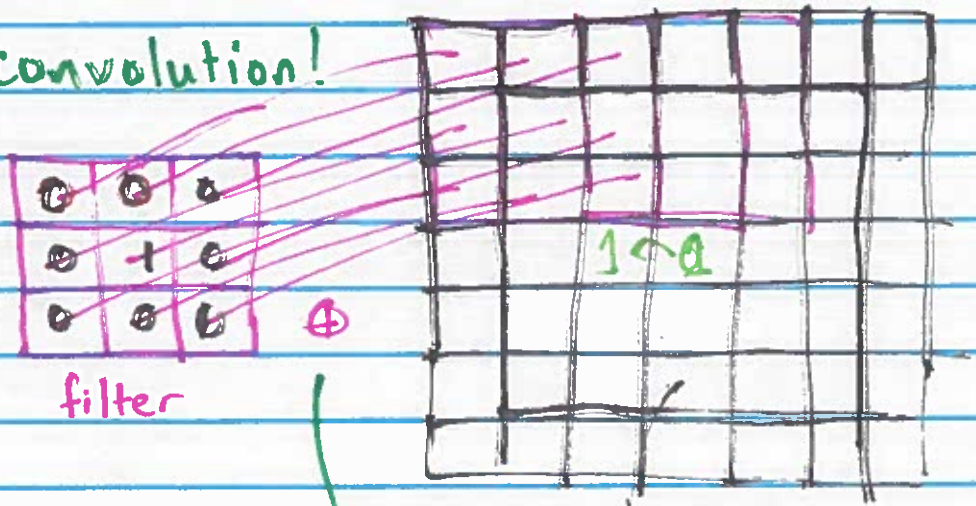
e.g. a 16×16 coloring: $(3 \times 16 \times 16)$ array

$$h(x) = U \cdot \text{ReLU}(\underline{Wx} + b)$$

$$= U \cdot \text{ReLU}(W \cdot \underbrace{\text{vec}(x)}_{\mathbb{R}^{3 \cdot 16 \cdot 16}} + b)$$

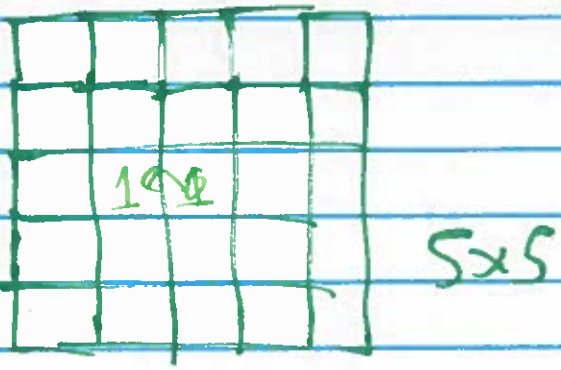
Want: something shift-invariant

Convolution!



Out = conv(in, filter)

$(d-k+1) \times (d-k+1)$ $d \times d$ $k \times k$



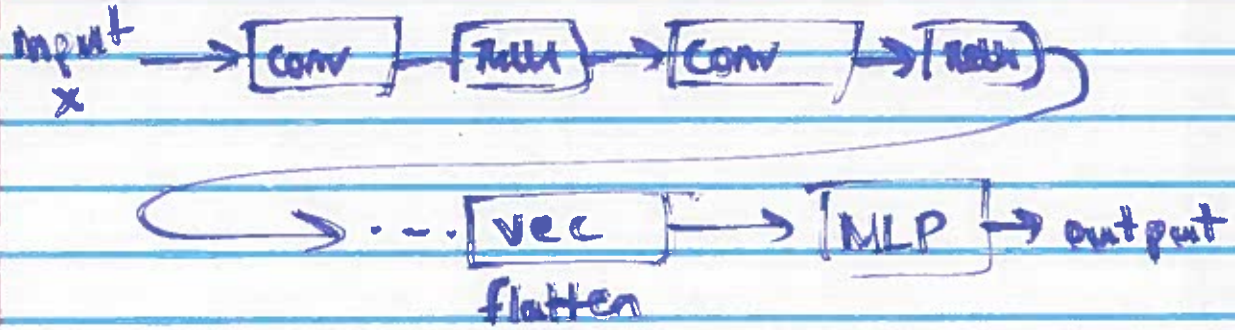
Conv layers:

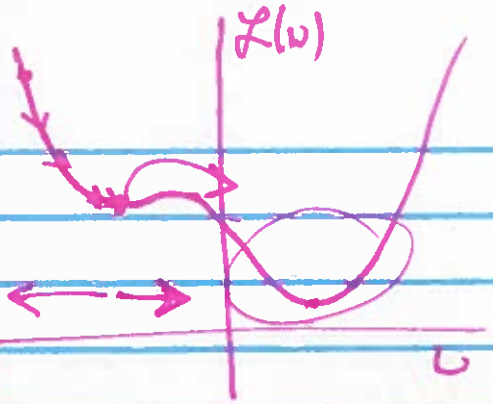
$\in \mathbb{R}^{C_{out} \times C_{in} \times k \times k}$

W parameters

$$(out)_j = \sum_i conv((in)_i, W_{j,i})$$

CNN





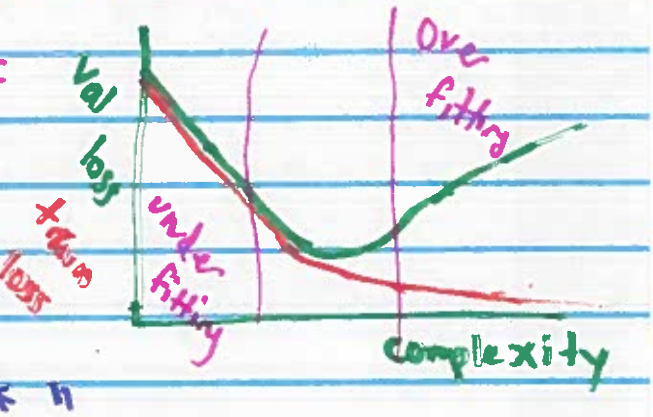
DNN and Overfitting

a model is overparameterized when

$d \gg n$

d: number of parameters

n: number of training examples



"DNNs don't overfit*"

as much as other models when trained using standard methods

Counter-overfitting methods:

$w - 2\alpha\gamma w - \alpha\nabla$

$2\gamma w$

$(1-2\alpha\gamma)w$

$-\alpha\nabla$

* l_2 regularization $+\gamma\|w\|^2$
  "weight decay"

* dropout: While training, for each step of SGD, "remove" each hidden node from the network w.p. $p > 0$, (e.g. $p = \frac{1}{2}$)

* batch normalization