More Algorithms

Naive Bayes

X, y (supervised) dataset (I.I.D.)

Goal: Approximate $p_{data}(y \mid \vec{x}) = \dfrac{p_{data}(y, \vec{x})}{p_{data}(\vec{x})}$

Chain rule for conditional probabilities

$$\begin{aligned} p_{data}(y, \vec{x}) &= p(y, x_1, \dots, x_n) \\ &= p(x_1 \mid x_2, \dots, x_n, y) \cdot p(x_2 \mid x_3, \dots, x_n, y) \\ &\quad \vdots \\ &\quad \cdot p(x_{n-1} \mid x_n, y) \\ &\quad \cdot p(x_n \mid y) \\ &\quad \cdot p(y) \end{aligned}$$

Naive assumption: all features are independent given y.

$$\Longrightarrow p(y, \vec{x}) = p(y) \prod_{j=1}^{n} p(x_j \mid y)$$

$$\therefore \quad p(y|\vec{x}) = \frac{p(y)\prod_{j=1}^{n} p(x_j|y)}{p(\vec{x})}$$

Naive Bayes Prediction

$$\hat{y} = \underset{y}{\operatorname{argmax}}\ p(y|\vec{x})$$

$$\boxed{\hat{y} = \underset{y}{\operatorname{argmax}}\ p(y)\prod_{j=1}^{n} p(x_j|y)}$$

Maximum a posteriori (MAP) decision rule

Find maximum likelihood estimates for $p(y)$ and $p(x_j|y)$ $\quad 1 \le j \le n$

$p_{model}(y;\vec{\theta}) \qquad p_{model}(x_j|y;\vec{\theta}^{(j)})$

Goal: Find $\vec{\theta}_{ML}$, $\vec{\theta}^{(j)}_{ML}$

(models depend on the type of data)

E.g. (Classification) $X, \vec{y}$ $\qquad y_i \in \{1,\dots,S\}\ \forall i$

$X \in \mathbb{R}^{m\times n}$

X: continuous, $\vec{y}$: discrete

$p_{model}(y=i;\vec{\theta}) = \theta_i \qquad \sum_{i=1}^{S}\theta_i = 1$

$$P_{model}(x_j \mid y=i ; \mu_{i,j}, \sigma^2_{i,j}) = \mathcal{N}(x_j ; \mu_{i,j}, \sigma^2_{i,j})$$

Goal Calculate $\vec{\theta}_{ML}$ and $(\mu_{i,j}, \sigma^2_{i,j})_{ML}$

Claim: $(\vec{\theta}_{ML})_i = \dfrac{\#\{k \mid \vec{y}_k = i\}}{m}$

$$(\mu_{i,j})_{ML} = \frac{\sum_{k: y_k = i} X_{k,j}}{\#\{k \mid y_k = i\}} = \text{Mean}\left(\{X_{k,j} \mid y_k = i\}\right)$$

i = label
j = feature

$$(\sigma^2_{i,j})_{ML} = \text{Var}\left(\{X_{k,j} \mid y_k = i\}\right)$$

↑ use $\frac{1}{n-1}$ instead of $\frac{1}{n}$

"Bessel corrected variance"

Then we can predict for a new $\vec{x}$:

$$\hat{y} = \underset{i}{\text{argmax}}\ \theta_i \prod_{j=1}^{n} \mathcal{N}(x_j ; (\mu_{i,j})_{ML}, (\sigma^2_{i,j})_{ML})$$

Honorable Mentions

- Support Vector Machines (kernel methods)
- k-nearest neighbors
- Decision Trees / Random Forest

Unsupervised Algorithms (no target $\vec{y}$)

Have dataset X or $\{\vec{x}^{(1)}, \ldots, \vec{x}^{(m)}\}$

Some Common Tasks:

1. Estimate $p_{data}(\vec{x})$
2. Draw new samples from $p_{data}(\vec{x})$
3. ★ Denoise X
4. Find a manifold which lies near data
5. ★ Find a "simpler representation"
   - lower dimensional
   - sparse
   - independence of features (remove dependencies)

PCA revisited: $X \in \mathbb{R}^{m \times n}$

WLOG each column has mean 0 (if not, subtract the mean from each column)

(so $E[\vec{x}] = 0$)

Consider the unbiased covariance matrix:

$$\text{Var}[\vec{x}] = \frac{1}{m-1} X^T X$$

(i,j) entry measures how much feature i is linearly correlated with feature j. (i,i) entry measures the variance in feature i.

Consider the SVD $X = U \Sigma V^T$ and define $T: \mathbb{R}^n \to \mathbb{R}^n$ $T(\vec{x}) = V^T \vec{x}$. Consider the new representation of the data:

$$\vec{z} = T(\vec{x})$$

Then $\text{Var}[\vec{z}] = \frac{1}{m-1}\begin{bmatrix} -\ T(\vec{x}_1)\ - \\ \vdots \\ -\ T(\vec{x}_m)\ - \end{bmatrix}^T \begin{bmatrix} -\ T(\vec{x}_1)\ - \\ \vdots \\ -\ T(\vec{x}_m)\ - \end{bmatrix} = \frac{1}{m-1} Z^T Z$ where $Z = XV$

$$= \frac{1}{m-1}\begin{bmatrix} | & & | \\ V^T\vec{x}_1 & \cdots & V^T\vec{x}_m \\ | & & | \end{bmatrix}\begin{bmatrix} & & \\ & \text{"} & \\ & & \end{bmatrix}$$

$$= \frac{1}{m-1} V^T X^T X V$$

(V is an orthogonal eigenvector basis for X^TX with eigenvals $\sigma_1^2, \ldots, \sigma_n^2$)

$$= \frac{1}{m-1}\begin{bmatrix} \sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_n^2 \end{bmatrix}$$

$\therefore$ no features of $\vec{z}$ are linearly correlated.

Also, $\sigma_1^2 \geq \sigma_2^2 \geq \ldots \geq \sigma_n^2$ so

$$\text{Var}(z_1) \geq \text{Var}(z_2) \geq \ldots \geq \text{Var}(z_n)$$

so features of $\vec{z}$ are ordered in terms of importance

We also know $\forall\, k \leq n$

$$T_k : \mathbb{R}^n \to \mathbb{R}^k, \quad T_k(\vec{x}) = \underbrace{\begin{bmatrix} | & & | \\ v_1 & \cdots & v_k \\ | & & | \end{bmatrix}^T}_{V_k^T} \vec{x}$$

optimally preserves the data (as you proved in HW 1)

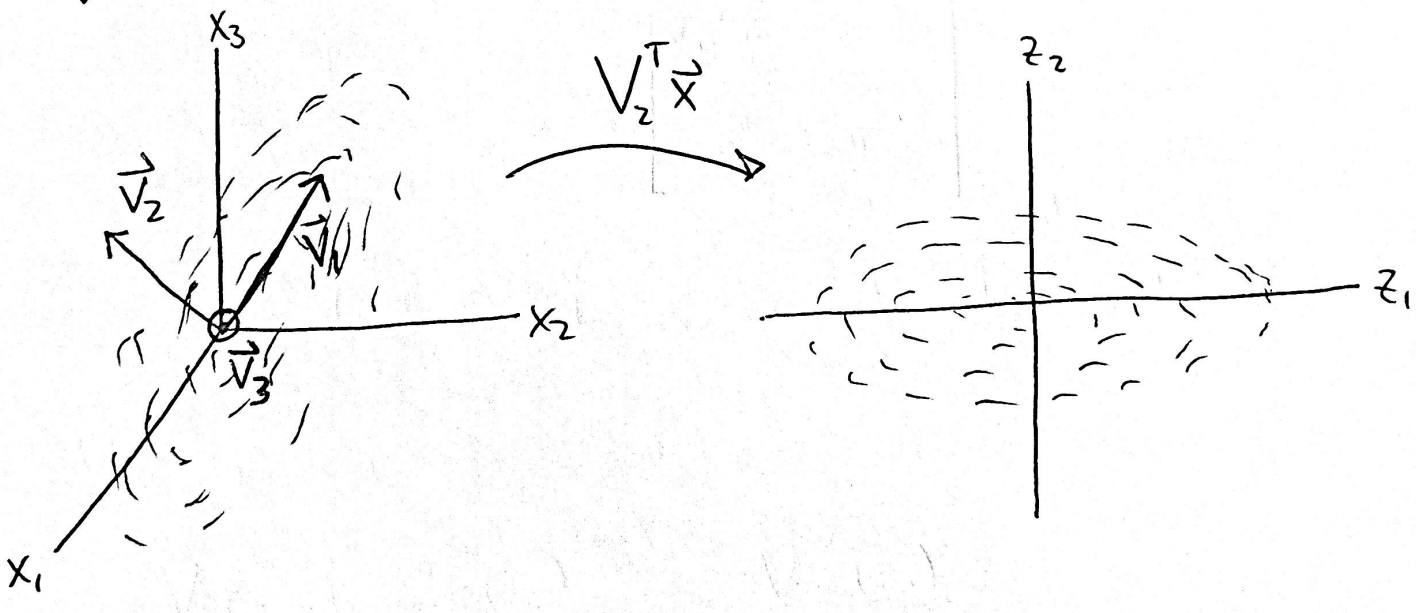
∴ representing the data via

$$Z = XV_k$$

accomplishes:

① Maximally preserves data
② Uncorrelates features
③ Orders features by importance

E.g.



"almost on a plane"

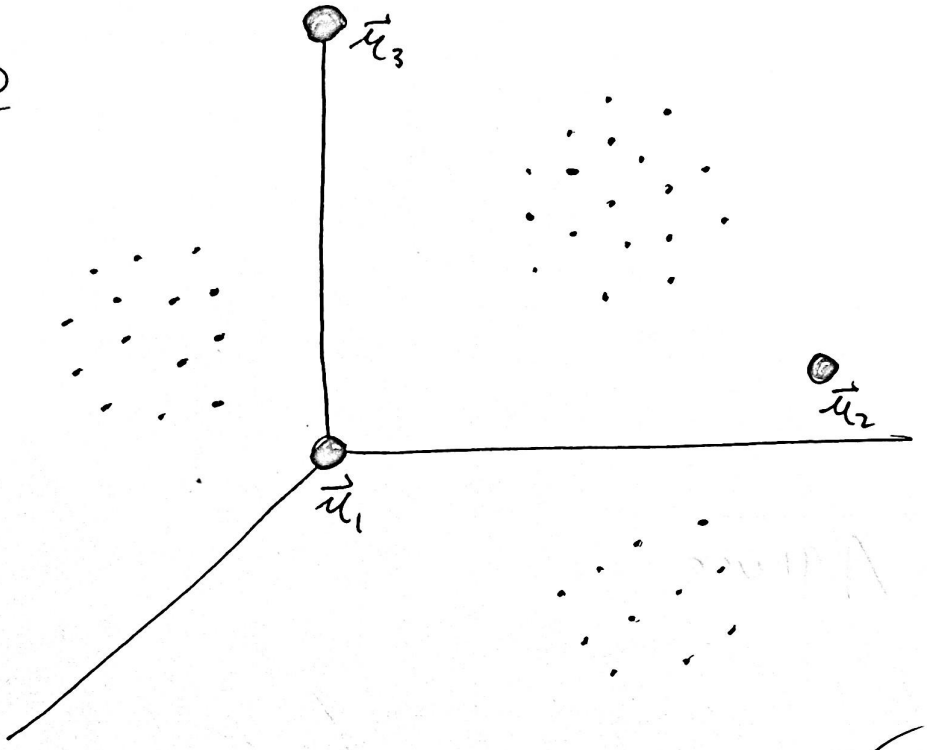
$X \in \mathbb{R}^{m \times 3}$

$V \in \mathbb{R}^{3 \times 3}$

k-means clustering (Unsupervised)

Dataset $X \in \mathbb{R}^{m \times n}$ datapoints: $X_{i,*} = \vec{x}^{(i)}$

step 0



stopping condition

Pick k (= # clusters). Pick $\varepsilon > 0$.

Randomly initialize k points $\vec{u}_1, \ldots, \vec{u}_k \in \mathbb{R}^n$

$\pi \in \mathbb{R}^m$ $\pi[i] = j$ if $\vec{x}^{(i)}$ is closest to $\vec{u}_j$

$J = \frac{1}{m} \sum_{i=1}^{m} \| \vec{x}^{(i)} - \vec{u}_{\pi[i]} \|_2$; costs = [J] ; $i = 0$

While True:

    if $i > 1$ and $|costs[i] - costs[i-1]| < \varepsilon$:

        break

for i in range(k):

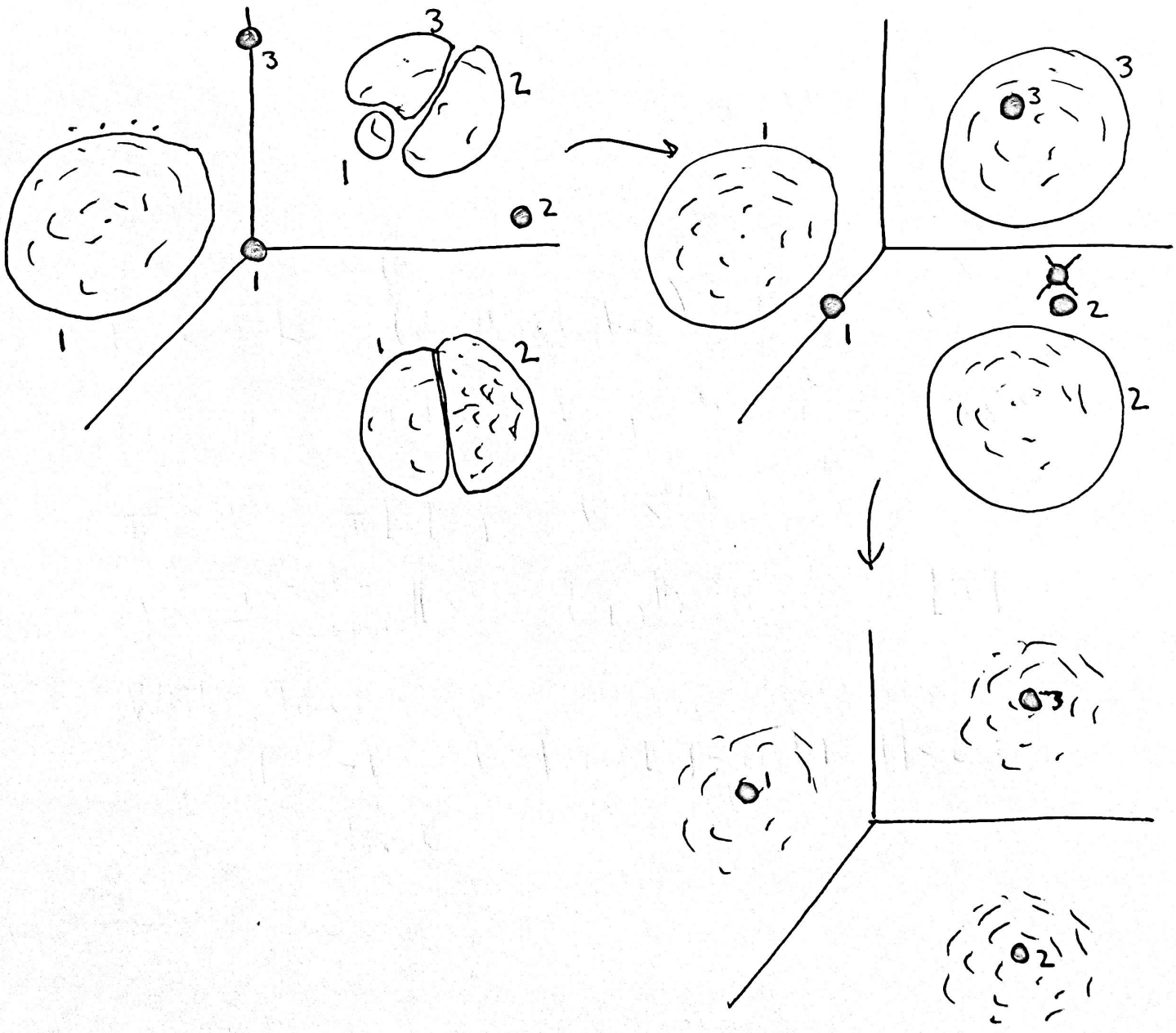
$$\vec{\mu}_i = \frac{1}{m_i} \sum_{j:\, \pi[j]=i} \vec{x}^{(j)} \qquad m_i = \#\{j \mid \pi[j] = i\}$$

recalculate π, J

i += 1

costs.append(J)

E.g. Continuing from step 0



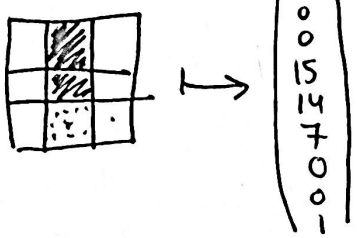
E.g. Classifying handwritten digits

Given dataset $X \in \mathbb{R}^{m \times 64}$ $X_{i,:}$ = straightened matrix of grayscale values of 8×8 image of digit (with values in 0-16)

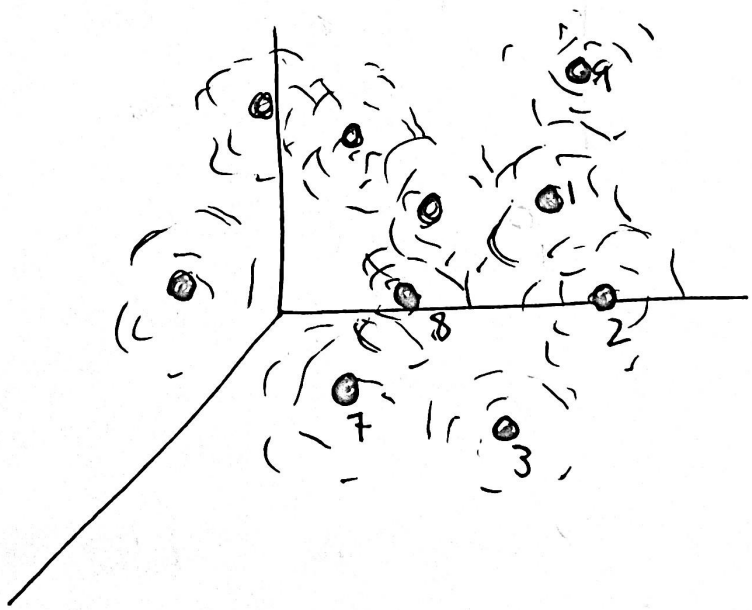
white ↖ ↗ black

E.g.

3×3 case



Run k-means



Label by hand. Make predictions on new digits by checking which is the closest cluster.

New Optimization Algorithm

Stochastic Gradient Descent (SGD)

Usually we have some cost function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{L(x^{(i)}, y^{(i)}; \theta)}_{\text{per example error "loss"}}$$

Then $\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(x^{(i)}, y^{(i)}; \theta)$

So $\nabla_\theta J(\theta)$ is an average

Idea: Estimate $\nabla_\theta J(\theta)$ by picking a subset of $\{1, \ldots, m\}$ of size $m' << m$ (maybe $m = 1$ billion, $m' = 100$)

This is called a mini-batch:

$$B = \{x^{(i_1)}, \ldots, x^{(i_{m'})}\} \subseteq \{x^{(1)}, \ldots, x^{(m)}\}$$

Compute $\nabla_\theta J(\theta) \approx \underbrace{\frac{1}{m'}\sum_{j=1}^{m'} \nabla_\theta L(x^{(i_j)}, y^{(i_j)}, \theta)}_{g}$

Then run similarly to the usual GD

$$\theta = \theta - \alpha g \quad (\alpha = \text{step size})$$

Suggestion Casually read the rest of Chp 5

Now Chp 6 Deep Feedforward Networks
or
Feedforward Neural Networks
or
Multilayer Perceptrons (MLPs)

(Supplement from "Deep Learning Architectures")

Goal Approximate a function $y = f^*(\vec{x})$ by finding parameters θ for a model $\hat{y} = f_{model}(\vec{x}; \vec{\theta})$ by running some optimization algorithm to find suitable θ

Abstract Neurons

Definition An abstract neuron (or just neuron) is a quadruple $(\vec{x}, \vec{w}, \varphi, y)$

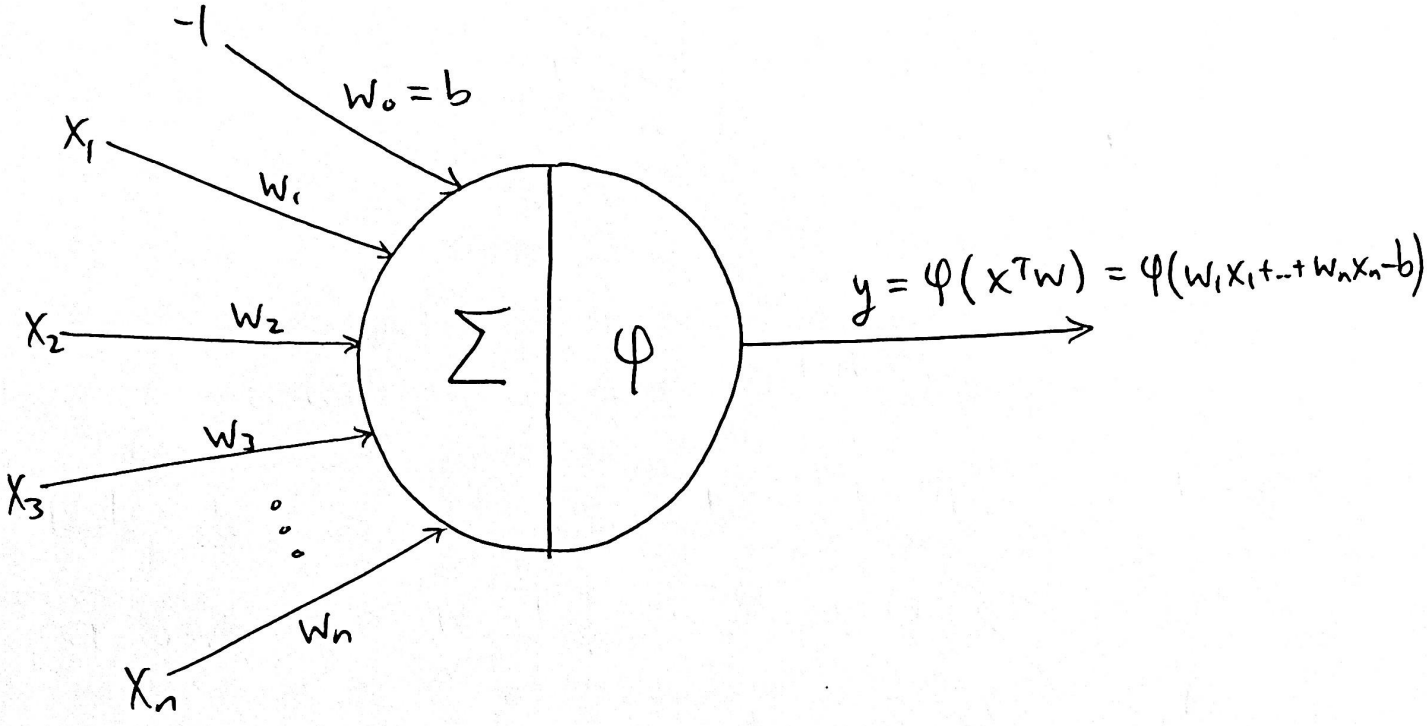
$\vec{x}^T = (x_0, \dots, x_n)$ "input vector" $x_0 = -1$

$\vec{w}^T = (w_0, \dots, w_n)$ "weight vector" $b = w_0$ = bias

$\varphi : \mathbb{R} \to \mathbb{R}$ "activation function"

$y = \varphi(x^T w)$ "output"

Idea: A neuron tries to "learn" a target z by tuning $\vec{w}$ so that $\varphi(x^T w) \approx z$



Def: A perceptron is a neuron with inputs $\vec{x} \in \{0,1\}^n$ and with activation function

$$\varphi(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$ (Heaviside function)

0 = False
1 = True

E.g. $f^*: \{0,1\} \times \{0,1\} \to \{0,1\}$

$f^*(x,y) = x$ and y

Use a perceptron with $w = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$

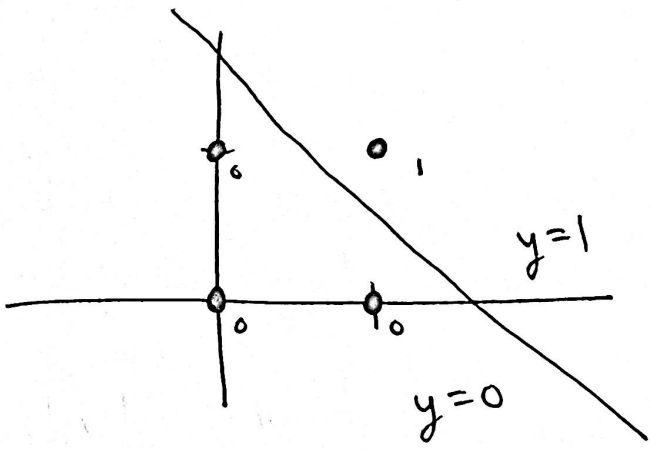
We want $\varphi(w^T x) = f^*(x) \quad \forall\, x \in \{0,1\}^2$

$$\Longrightarrow \begin{cases} \varphi\left(w^T \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}\right) = 0 \\ \varphi\left(w^T \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}\right) = 0 \\ \varphi\left(w^T \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}\right) = 0 \\ \varphi\left(w^T \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}\right) = 1 \end{cases} \Longleftrightarrow \begin{cases} w^T \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix} < 0 \\ w^T \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} < 0 \\ w^T \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} < 0 \\ w^T \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} \geq 0 \end{cases}$$

A solution: $w = \begin{pmatrix} 1.5 \\ 1 \\ 1 \end{pmatrix}$

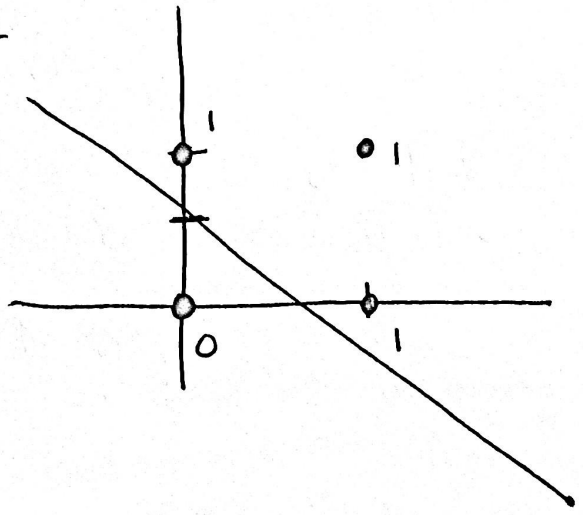
$$w^T x = -1.5 + x_1 + x_2 = 0$$

$$x_2 = 1.5 - x_1$$

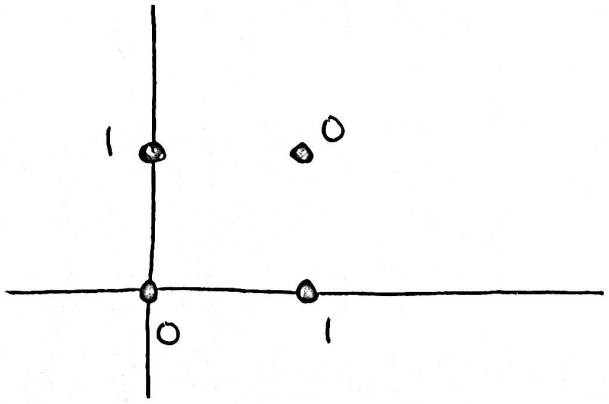


E.g. $f^*(x_1, x_2) = x_1$ or x_2

$$w = \begin{pmatrix} 0.5 \\ 1 \\ 1 \end{pmatrix}$$



E.g. $f^*(x_1, x_2) = x_1$ XOR x_2 (one or the other but not both)



Perceptron cannot fit this function

(no line in $\mathbb{R}^2$ separates 1's from 0's)