CS 760: Machine Learning
Midterm Review

Misha Khodak

University of Wisconsin-Madison

20 October 2025

Logistics

- Midterm: 75 min in-class October 22nd
  - covers material through October ~~15~~**8th**, focusing on everything **before** neural networks (but still includes MLPs)
  - mix of short answer and derivations
  - one double-sided 8.5x11 cheat sheet
  - no calculators
  - practice midterm posted in today's readings

- No office hours today (will still hold them tomorrow)

Outline

- **Instance-based learning:** k-NN, decision trees
- **Evaluation:** data splitting, metrics
- **Parametric modeling:** linear & logistic regression, regularization, MLE, MAP
- **Optimization:** gradient descent, SGD, convergence
- **Unsupervised learning:** centroid clustering, mixture models, PCA
- **Neural networks:** MLPs

Outline

- **Instance-based learning:** k-NN, decision trees
- **Evaluation:** data splitting, metrics
- **Parametric modeling:** linear & logistic regression, regularization, MLE, MAP
- **Optimization:** gradient descent, SGD, convergence
- **Unsupervised learning:** centroid clustering, mixture models, PCA
- **Neural networks:** MLPs

k-Nearest Neighbors

Training/learning: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Prediction: for x , find ***k*** most similar training points

Return plurality class

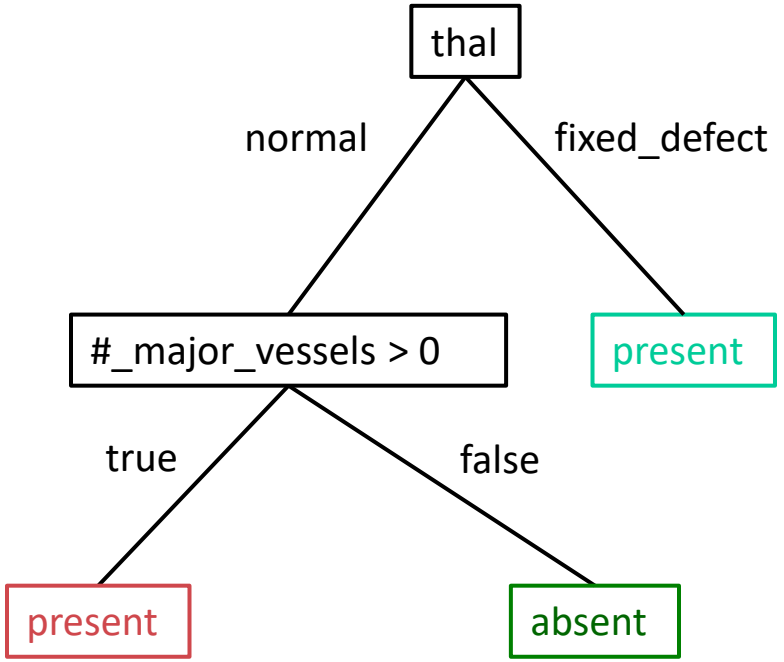
$$\hat{y} = \arg\max_{y \in \mathcal{Y}} \sum_{i=1}^{k} \mathbb{1}(y = y^{(i)})$$

- i.e. among the ***k*** points, output most popular class.

Decision trees

Model: assign a label to inputs $\mathbf{x}$ by traversing a tree starting from the root node

- if at a leaf node, output the label
- else go to the branch of the tree determined by the node feature index i and the input's corresponding feature $\mathbf{x}_{[i]}$
  - if categoric, determined by assigning $\mathbf{x}_{[i]}$ to one of the category groupings
  - if numeric, determined by whether $\mathbf{x}_{[i]}$ is greater or less than a threshold



Learning: groupings, thresholds, and leaf labels determined via greedy heuristics

Outline

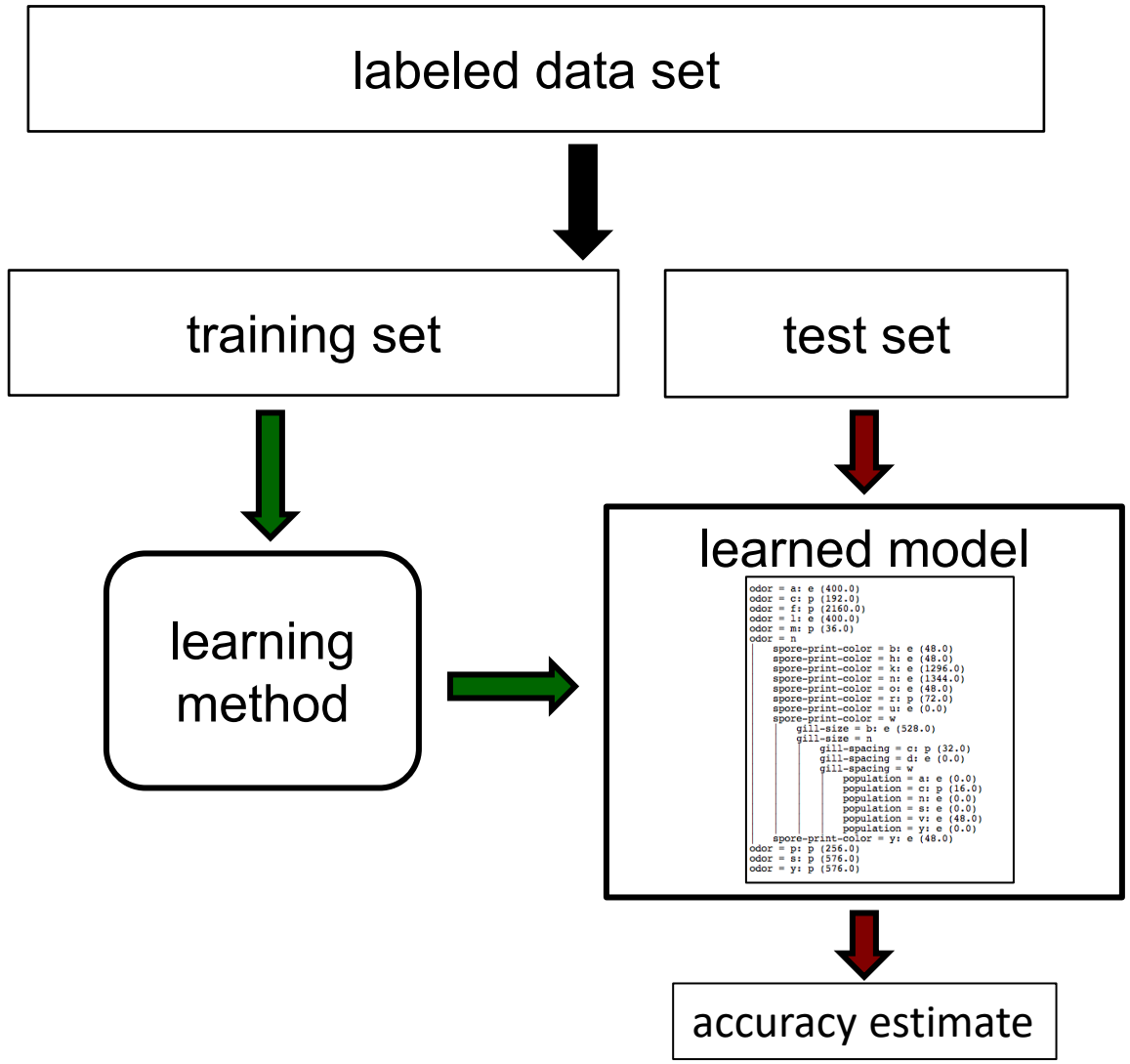
- **Instance-based learning:** k-NN, decision trees
- **Evaluation:** data splitting, metrics
- **Parametric modeling:** linear & logistic regression, regularization, MLE, MAP
- **Unsupervised learning:** centroid clustering, mixture models, PCA
- **Optimization:** gradient descent, SGD, convergence
- **Neural networks:** MLPs

Accuracy of a Model

How can we estimate the accuracy of a learned model?

- Typically: use a statistic $\hat{\theta}$ that is an **unbiased estimator** of θ computed over an **independent** test set

$$\mathbb{E}\left[\hat{\theta}\right] = \theta$$



Using a Test Set

- How can we estimate the accuracy of a learned model?
  - When learning a model, you should pretend that you don't have the test data yet
  - If the test-set labels influence the learned model in any way, accuracy estimates will not be correct, as you may have fitted to your test set.

- **Don't train on the test set!!!**

Single Train/Test Split: Limitations

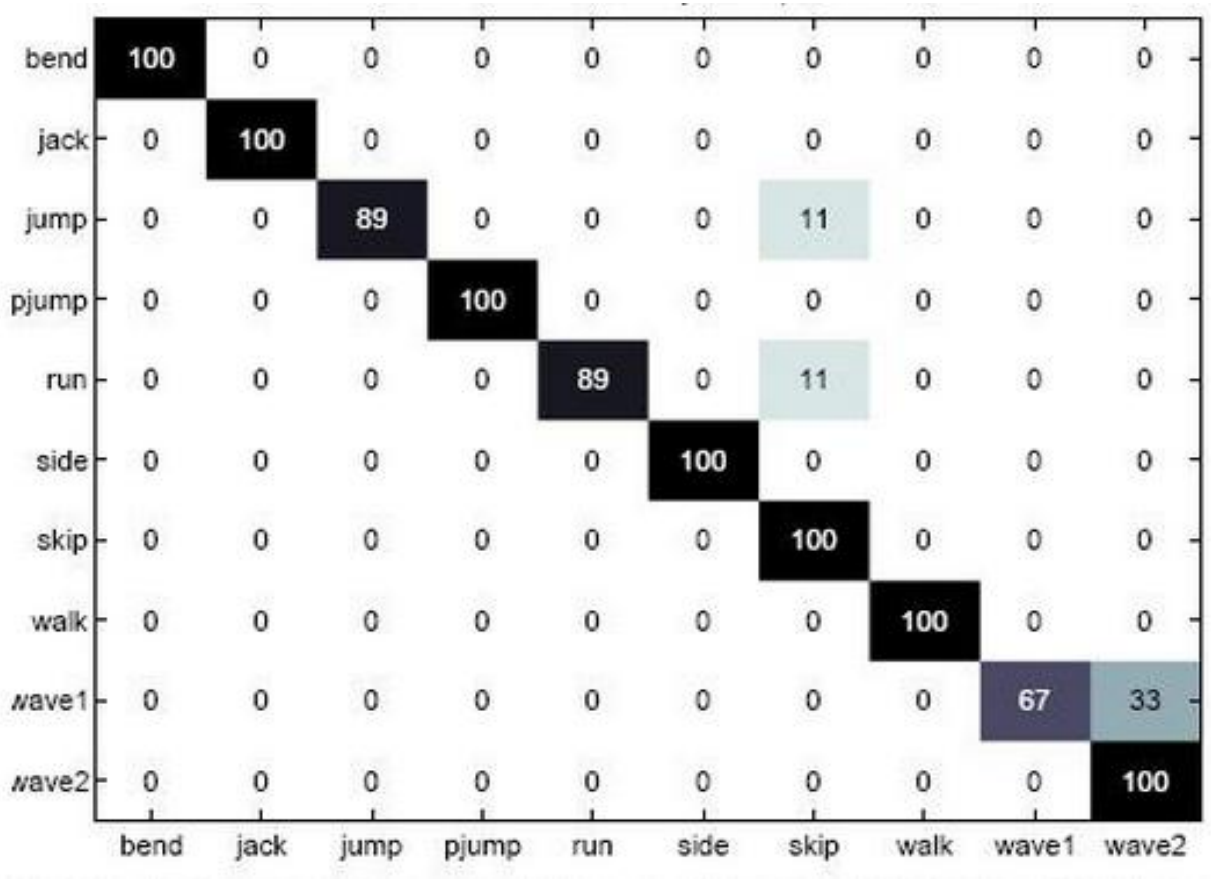
1. May not have enough data for sufficiently large training/test sets
   - A **larger test set** gives us more reliable estimate of accuracy (i.e. a lower variance estimate)
   - But… a **larger training set** will be more representative of how much data we actually have for the learning process

2. A single training set cannot reveal how sensitive accuracy is to specific training samples.

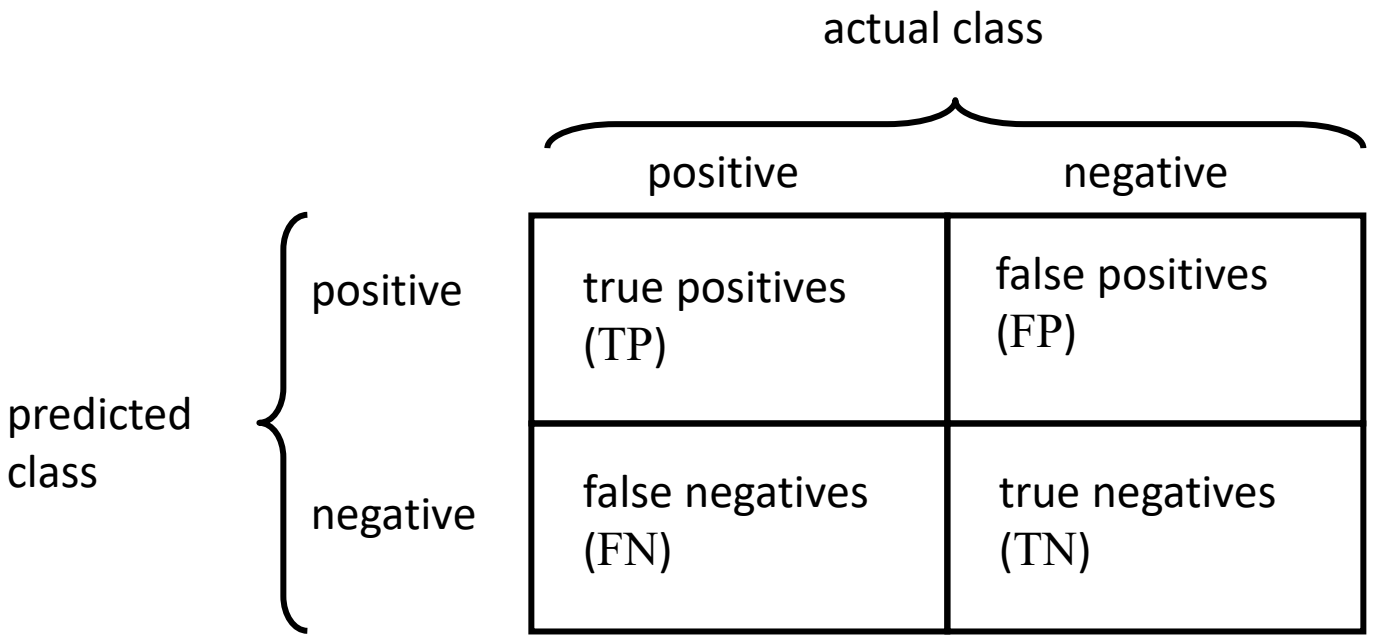
Beyond Accuracy: Confusion Matrices

- How can we understand what types of mistakes a learned model makes?

task: activity recognition from video



Confusion Matrices: 2-Class Version



$$\text{accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

$$\text{error} = 1 - \text{accuracy} = \frac{FP + FN}{TP + FP + FN + TN}$$

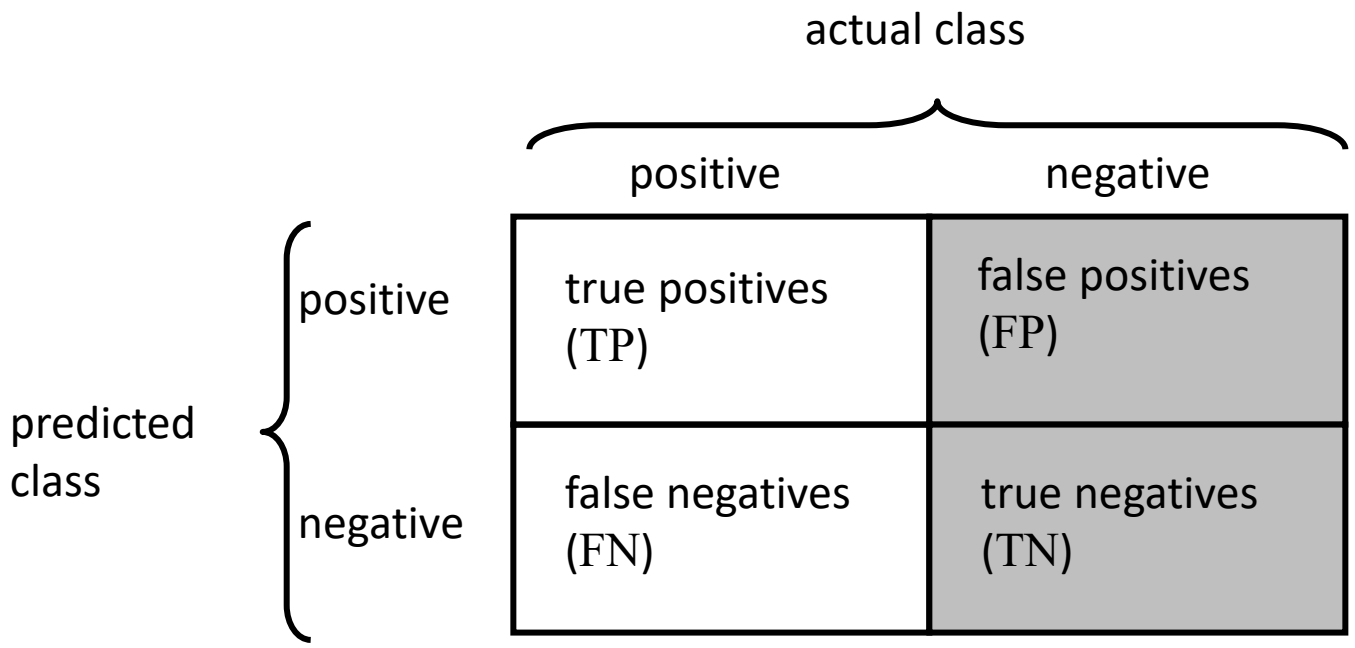
Accuracy: Sufficient?

Accuracy may not be useful measure in cases where

- There is a large class skew
  - Is 98% accuracy good when 97% of the instances are negative?

- There are differential misclassification costs – say, getting a positive wrong costs more than getting a negative wrong
  - Consider a medical domain in which a false positive results in an extraneous test but a false negative results in a failure to treat a disease

Other Metrics

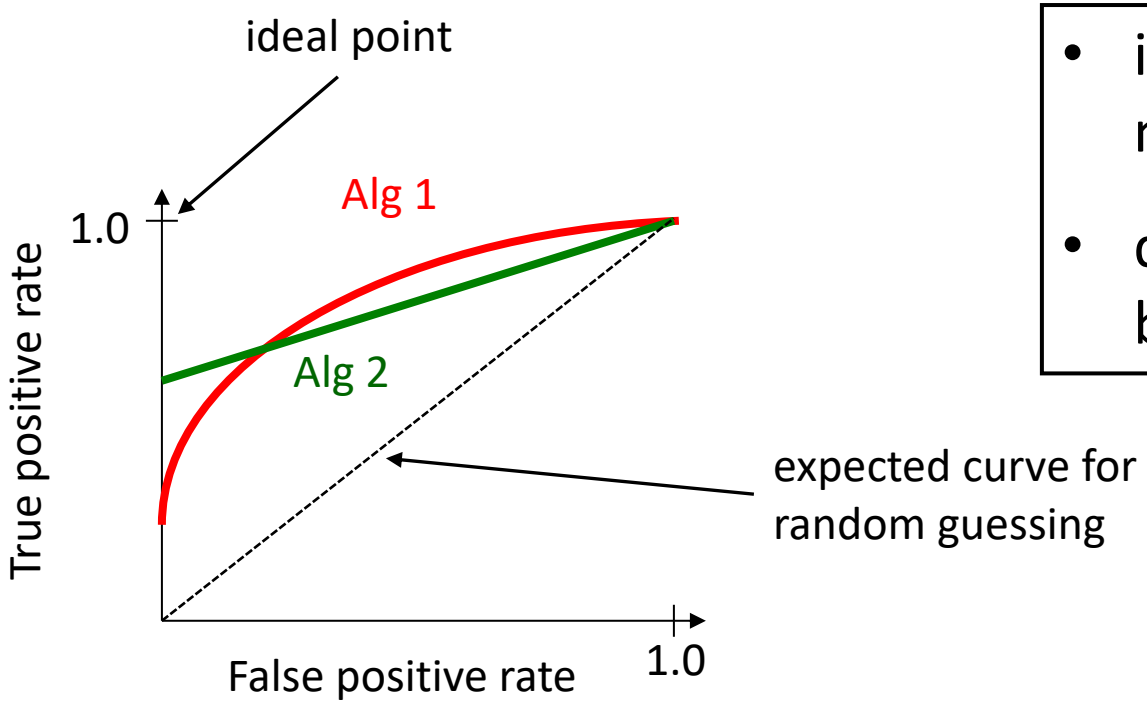


$$\text{true positive rate (recall)} = \frac{TP}{\text{actual pos}} = \frac{TP}{TP + FN}$$

$$\text{false positive rate} = \frac{FP}{\text{actual neg}} = \frac{FP}{TN + FP}$$

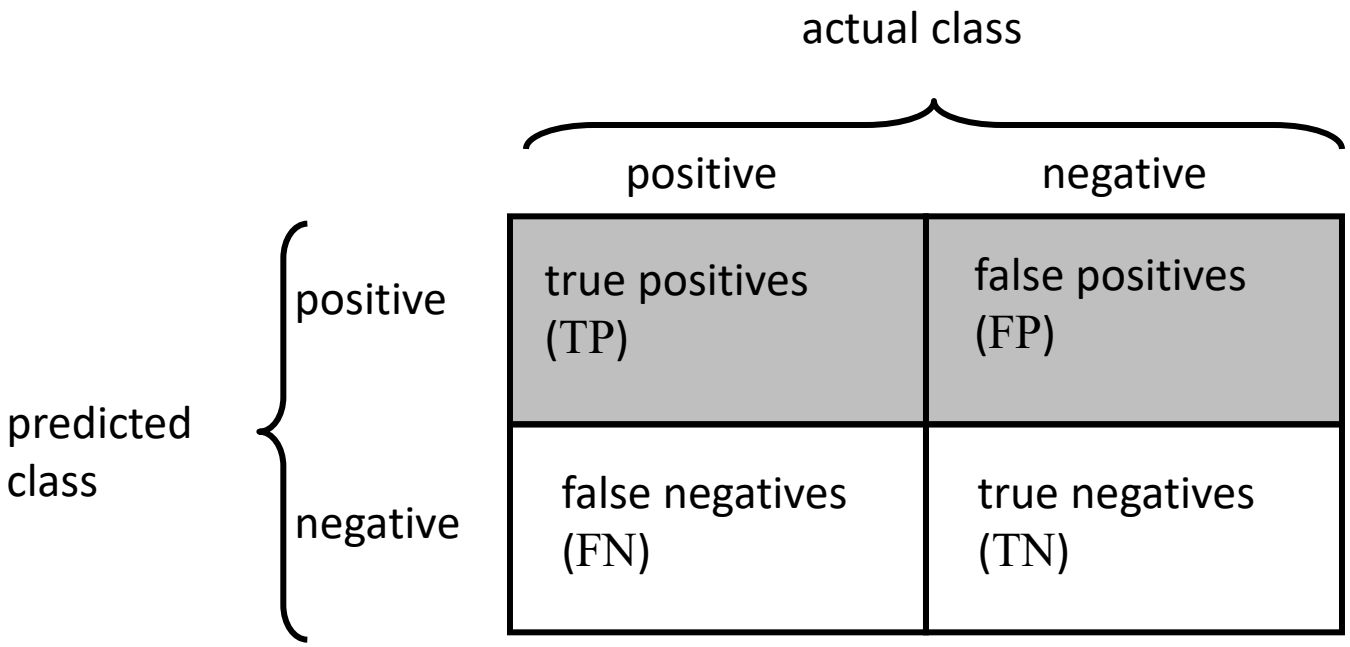
Other Metrics: ROC Curves

- A *Receiver Operating Characteristic* (*ROC*) curve plots the TP-rate vs. the FP-rate as a threshold on the confidence of an instance being positive is varied



- increasing the threshold c moves down along the curve
- different methods can work better at different points

Other Metrics: Precision



$$\text{recall (TP rate)} = \frac{\text{TP}}{\text{actual pos}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{precision (positive predictive value)} = \frac{\text{TP}}{\text{predicted pos}} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Outline

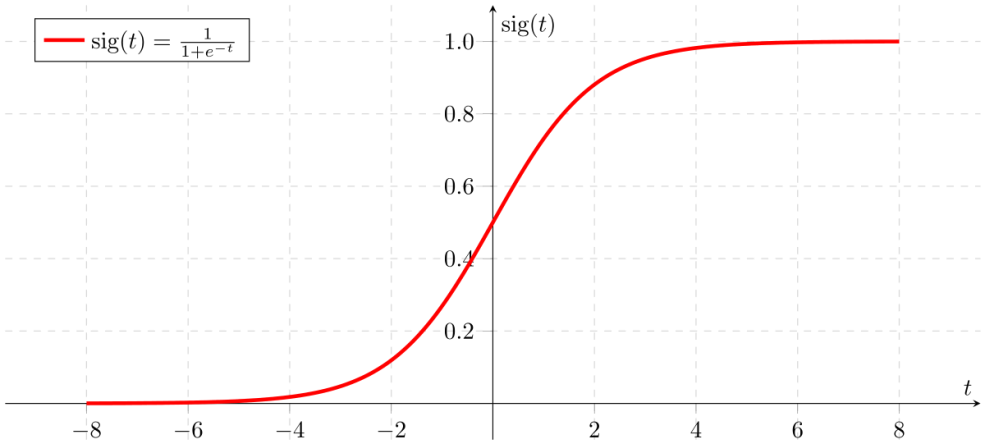
- **Instance-based learning:** k-NN, decision trees
- **Evaluation:** data splitting, metrics
- **Parametric modeling:** linear & logistic regression, regularization, MLE, MAP
- **Optimization:** gradient descent, SGD, convergence
- **Unsupervised learning:** centroid clustering, mixture models, PCA
- **Neural networks:** MLPs

Linear Classification

- Let's think probabilistically and learn $P_\theta(y|x)$

- How?
  - Specify the conditional distribution $P_\theta(y|x)$
  - Use **MLE** to derive a loss
  - Run gradient descent (or related optimization algorithm)

  - Leads to logistic regression



Likelihood Function

- Captures the probability of seeing some data as a function of model parameters:

$$\mathcal{L}(\theta; X) = P_\theta(X)$$

  - If data is iid, we have $\mathcal{L}(\theta; X) = \prod_j p_\theta(x_j)$

- Often more convenient to work with the log likelihood
  - Log is a monotonic + strictly increasing function

Maximum Likelihood

- For some set of data, find the parameters that maximize the likelihood / log-likelihood

$$\hat{\theta} = \arg\max_{\theta} \mathcal{L}(\theta; X)$$

- Example: suppose we have n samples from a Bernoulli distribution

$$P_\theta(X = x) = \begin{cases} \theta & x = 1 \\ 1 - \theta & x = 0 \end{cases}$$

Then,

$$\mathcal{L}(\theta; X) = \prod_{i=1}^{n} P(X = x_i) = \theta^k (1 - \theta)^{n-k}$$

Maximum Likelihood: Example

- Want to maximize likelihood w.r.t. Θ

$$\mathcal{L}(\theta; X) = \prod_{i=1}^{n} P(X = x_i) = \theta^k (1-\theta)^{n-k}$$

- Differentiate (use product rule) and set to 0. Get

$$\theta^{h-1}(1-\theta)^{n-h-1}(h - n\theta) = 0$$

- So: ML estimate is $\hat{\theta} = \frac{h}{n}$

ML: Conditional Likelihood

- Similar idea, but now using conditional probabilities:

$$\mathcal{L}(\theta; Y, X) = p_\theta(Y|X)$$

  - If data is iid, we have

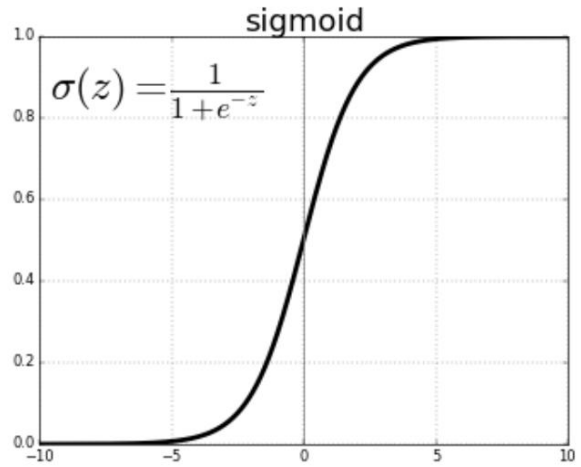
$$\mathcal{L}(\theta; Y, X) = \prod_j p_\theta(y_j|x_j)$$

  - Now we can apply this to linear classification: yields **logistic regression.**

Logistic Regression: Conditional Distribution

- Notation: $\sigma(z) = \dfrac{1}{1+\exp(-z)} = \dfrac{\exp(z)}{1+\exp(z)}$

Sigmoid



- **Conditional Distribution**:

$$P_\theta(y=1|x) = \sigma(\theta^T x) = \frac{1}{1+\exp(-\theta^T x)}$$

Logistic Regression: Loss

- Conditional MLE:

$$\log \text{ likelihood}(\theta|x^{(i)}, y^{(i)}) = \log P_\theta(y^{(i)}|x^{(i)})$$

- So:

$$\min_\theta \ell(f_\theta) = \min_\theta -\frac{1}{n}\sum_{i=1}^{n} \log P_\theta(y^{(i)}|x^{(i)})$$

Or,

$$\min_\theta -\frac{1}{n}\sum_{y^{(i)}=1} \log \sigma(\theta^T x^{(i)}) - \frac{1}{n}\sum_{y^{(i)}=0} \log(1-\sigma(\theta^T x^{(i)}))$$

Logistic regression: Summary

- **Logistic regression = sigmoid conditional distribution + MLE**

- More precisely:
  - Give training data iid from some distribution D,
  - **Train**:

$$\min_{\theta} \ell(f_\theta) = \min_{\theta} -\frac{1}{n} \sum_{i=1}^{n} \log P_\theta(y^{(i)}|x^{(i)})$$

  - **Test**: output label probabilities

$$P_\theta(y = 1|x) = \sigma(\theta^T x) = \frac{1}{1 + \exp(-\theta^T x)}$$

Logistic Regression: Beyond Binary

- Let's set, for y in 1,2,…,k

$$P_\theta(y = i|x) = \frac{\exp((\theta^i)^T x)}{\sum_{j=1}^{k} \exp((\theta^j)^T x)}$$

- Note: we have several weight vectors now (1 per class).
- To train, same as before (just more weight vectors).

$$\min_\theta -\frac{1}{n} \sum_{i=1}^{n} \log P_\theta(y^{(i)}|x^{(i)})$$

Linear Regression: Setup

- **Training/learning**: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

- Find $f_\theta(x) = \theta^T x = \sum_{i=0} \theta_i x_i$ that minimizes

Note: set $x_0 = 1$

Hypothesis Class

$$\ell(f_\theta) = \frac{1}{n} \sum_{j=1}^{n} \underbrace{(f_\theta(x^{(j)}) - y^{(j)})^2}$$

Loss function (how far are we)?

Linear Regression: Notation

- **Matrix notation**: set *X* to have *j*th row be $\left(x^{(j)}\right)^T$
  - And *y* to be the vector $[y^{(1)}, \ldots, y^{(n)}]^T$
- Can re-write the loss function as

$$\ell(f_\theta) = \frac{1}{n}\sum_{j=1}^{n}(f_\theta(x^{(j)}) - y^{(j)})^2 = \frac{1}{n}\|X\theta - y\|_2^2$$

Linear Regression: Fitting

- **Set gradient to 0 w.r.t. the weight,**

$$\nabla \ell(f_\theta) = \nabla \frac{1}{n} \|X\theta - y\|_2^2 = 0$$

$$\implies \nabla[(X\theta - y)^T (X\theta - y)] = 0$$

$$\implies \nabla[\theta^T X^T X\theta - 2\theta^T X^T y + y^T y] = 0$$

$$\implies 2X^T X\theta - 2X^T y = 0$$

$$\implies \theta = (X^T X)^{-1} X^T y$$

(assume $X^T X$ is invertible)

Evaluation: Metrics

- MSE/RMSE (mean-square error + root version)
- MAE (mean average error)
- R-squared

High-dimensional linear regression

Data matrix X is $n \times d$

- number of data points n
- number of features d

If $n > d$ and X has full column rank then $X^\top X$ is invertible

But what if $d \gg n$?

- e.g. a training set of n =1K documents, each represented as a **bag-of-words** vector ($X_{[j,i]} = \#\ times\ word\ i\ is\ in\ document\ j$) with vocabulary size d =10K
- now $X^\top X$ will not be invertible

Solution: **Regularization**

- Same setup, new loss (**Ridge regression**):

$$\ell(f_\theta) = \frac{1}{n}\sum_{j=1}^{n}(f_\theta(x^{(j)}) - y^{(j)})^2 + \lambda\|\theta\|_2^2$$

regularization parameter

- Conveniently, still has a closed form solution

$$\theta = (X^T X + \lambda n I)^{-1} X^T y$$

- **Goals:**
  - solves the problem of $X^\top X$ not being invertible
  - results in a θ with small norm, which is often less likely to overfit

Alternative regularization: **LASSO**

- Another type of regularization:

$$\ell(f_\theta) = \frac{1}{n}\sum_{j=1}^{n}(f_\theta(x^{(j)}) - y^{(j)})^2 + \lambda\|\theta\|_1$$

regularization parameter

  - unlike the ℓ_2-norm, regularizing by the ℓ_1-norm is known to encourage a sparse θ
    - theoretical understanding of this phenomenon exists under assumptions on X and y (**compressed sensing**)
    - useful for both regularization and **feature selection**

Other things you can do with regularization

- combine ℓ_1 and ℓ_2 regularization (Elastic Net)
- feature selection: determine which features of your model are important
- regularize classifiers like logistic regression (just add a norm penalty to the MLE objective)

Probabilistic interpretation

the ordinary least squares (OLS) estimator $\theta = (X^T X)^{-1} X^T y$ estimator is the MLE of a Gaussian probabilistic model:

- $y^{(i)} \sim N(\theta^\top x^{(i)}, \sigma^2)$
- assume variance σ^2 is known

Ridge regression and LASSO are **MAP estimators of the same probabilistic model** with different priors for θ

- Ridge regression: $\theta \sim N(0_d, \tau^2 I_d)$
- LASSO: $\theta \sim \text{Laplace}(0_d, \tau)$
- in both cases τ depends on σ^2 and λ

Another Approach: **Bayesian Inference**

- Let's consider a different approach
- Need a little bit of terminology

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

- *H* is the hypothesis
- *E* is the evidence

Bayesian Inference Definitions

- Terminology:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)} \longleftarrow \textbf{Prior}$$

- Prior: estimate of the probability **without** evidence

Bayesian Inference Definitions

- Terminology:

Likelihood

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

- Likelihood: probability of evidence **given a hypothesis**.
  - Compare to the way we defined the likelihood earlier

Bayesian Inference Definitions

- Terminology:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

↑

Posterior

- Posterior: probability of hypothesis **given evidence**.

MAP Definition

- Suppose we think of the parameters as random variables
  - There is a prior $P(\theta)$

- Then, can do learning as Bayesian inference
  - "Evidence" is the data

$$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)}$$

  - **Maximum a posteriori probability** (**MAP**) estimation

$$\theta^{\mathrm{MAP}} = \arg\max_{\theta} \prod_{i=1}^{n} p(x^{(i)}|\theta)p(\theta)$$

MAP vs ML

- What's the difference between ML and MAP?

$$\theta^{\mathrm{MLE}} = \arg\max_{\theta} \prod_{i=1}^{n} p(x^{(i)}|\theta)$$

$$\theta^{\mathrm{MAP}} = \arg\max_{\theta} \prod_{i=1}^{n} p(x^{(i)}|\theta)p(\theta)$$

- **the prior!**

Outline

- **Instance-based learning:** k-NN, decision trees
- **Evaluation:** data splitting, metrics
- **Parametric modeling:** linear & logistic regression, regularization, MLE, MAP
- **Optimization:** gradient descent, SGD, convergence
- **Unsupervised learning:** centroid clustering, mixture models, PCA
- **Neural networks:** MLPs

Optimization in ML

in supervised learning, we

- have a training dataset of $\left(x^{(i)}, y^{(i)}\right)$ pairs for $i = 1, \ldots, n$
- search a hypothesis space H for a function h that
  - predicts well, i.e. $h\left(x^{(i)}\right) = y^{(i)}$ on most of the training data
  - satisfies other constraints, e.g. simplicity so as not to overfit

Optimization in ML

often **searching the hypothesis** space is an **optimization problem**:

- decision trees

$$\min_{\substack{h(x^{(i)})=y^{(i)} \\ \forall i \in [n]}} depth(h)$$

predicts well **is simple**

- parametric models

$$\min_{\theta \in \mathbb{R}^d} \sum_{i=1}^{n} loss\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) + penalty(\theta)$$

Iterative Methods: Gradient Descent

- What if there's no closed-form solution?
- Use an iterative approach to gradually get closer to the solution.

- Gradient descent:
  - Suppose we're computing $\min_{\theta} g(\theta)$
  - Start at some θ_0
  - Iteratively compute $\theta_{t+1} = \theta_t - \alpha \nabla g(\theta_t)$
  - Stop after some # of steps

learning rate/step size

Gradient Descent: Convergence

- Let's analyze it. We'll need some **assumptions**
  - convex and differentiable objective
  - has L-Lipschitz-continuous gradients

- Under these assumptions, we have the following guarantee:
  - if we run T steps of GD with fixed step size $\alpha \leq 1/L$ starting at x_0, then the Tth iterate x_T satisfies

$$f(x_T) - f(x^*) \leq \frac{||x_0 - x^*||_2^2}{2T\alpha}$$

minimizer

Gradient Descent: Drawbacks

- Why would we use anything but GD?
- Let's go back to ERM. $\arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(h(x^{(i)}, y^{(i)})$
- For GD, need to compute $\nabla \ell(h(x^{(i)}, y^{(i)})$
  - Each step: n gradient computations
  - ImageNet: 10^6 samples… so for 100 iterations, **10^8 gradients**

Solution: **Stochastic** Gradient Descent

- Simple modification to GD.
- Let's use some notation: ERM:

$$\arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ell(f(\theta; x^{(i)}), y^{(i)})$$

Note: this is what we're optimizing over!
x's are fixed samples.

- GD: $$\theta_{t+1} = \theta_t - \frac{\alpha}{n} \sum_{i=1}^{n} \nabla \ell(f(\theta_t; x^{(i)}), y^{(i)})$$

Solution: **Stochastic** Gradient Descent

- Simple modification to GD:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{n} \sum_{i=1}^{n} \nabla \ell(f(\theta_t; x^{(i)}), y^{(i)})$$

- SGD: $$\theta_{t+1} = \theta_t - \alpha \nabla \ell(f(\theta_t; x^{(a)}), y^{(a)})$$

  - Here a is selected uniformly from 1,…,n (“**stochastic**” bit)
  - Note: **no sum**!
  - In expectation, same gradient as GD.
  - In practice we often update using **minibatches** of data to take advantage of (GPU) parallelism

Outline

- **Instance-based learning:** k-NN, decision trees
- **Evaluation:** data splitting, metrics
- **Parametric modeling:** linear & logistic regression, regularization, MLE, MAP
- **Optimization:** gradient descent, SGD, convergence
- **Unsupervised learning:** centroid clustering, mixture models, PCA
- **Neural networks:** MLPs

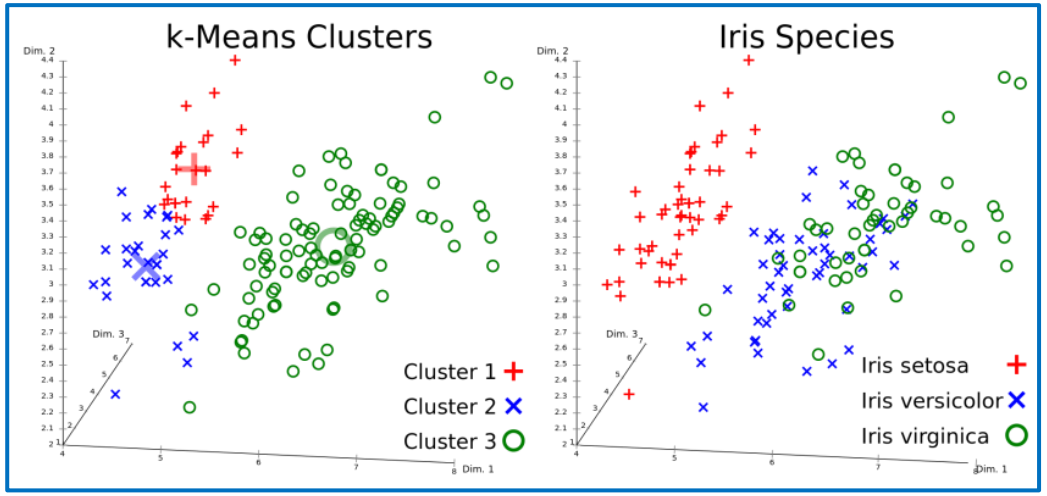
Clustering

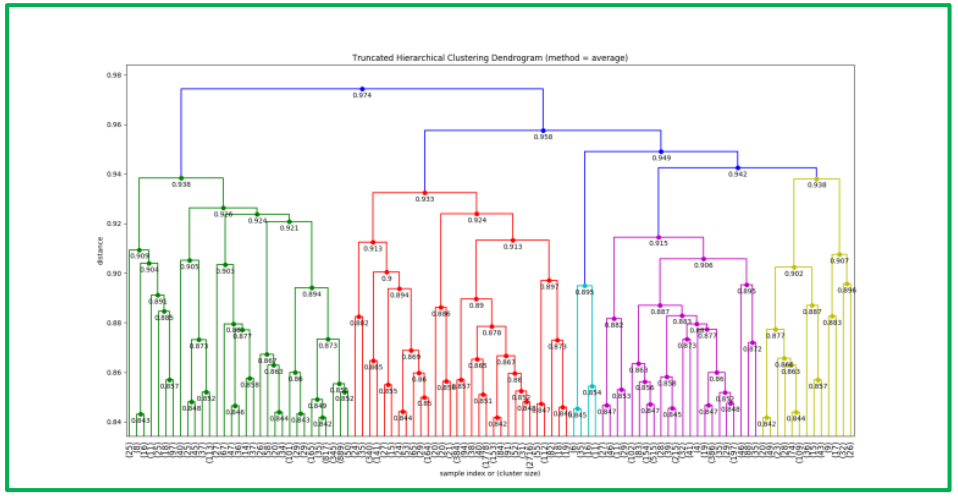
Several types:

Partitional
- Centroid
- Graph-theoretic
- Spectral

Hierarchical
- Agglomerative
- Divisive

Bayesian
- Decision-based
- Nonparametric



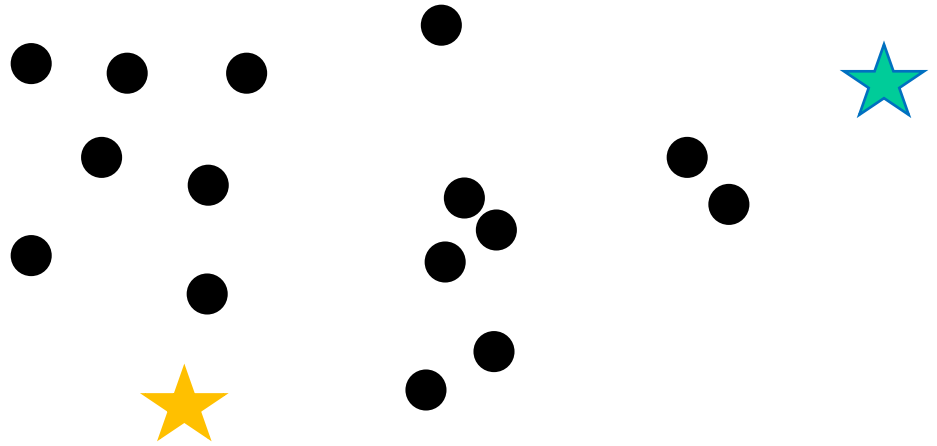


K-Means Clustering

k-means is a type of partitional **centroid-based clustering**

Algorithm:

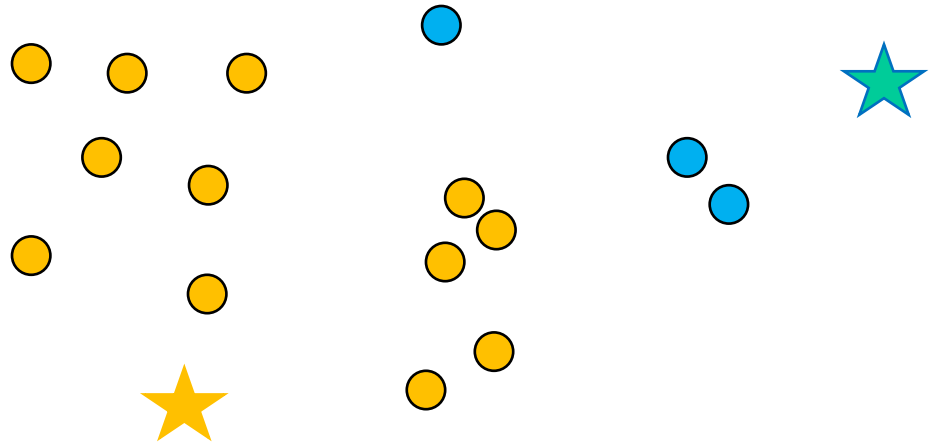
1. Randomly pick k cluster centers



K-Means Clustering: Algorithm

k-means clustering

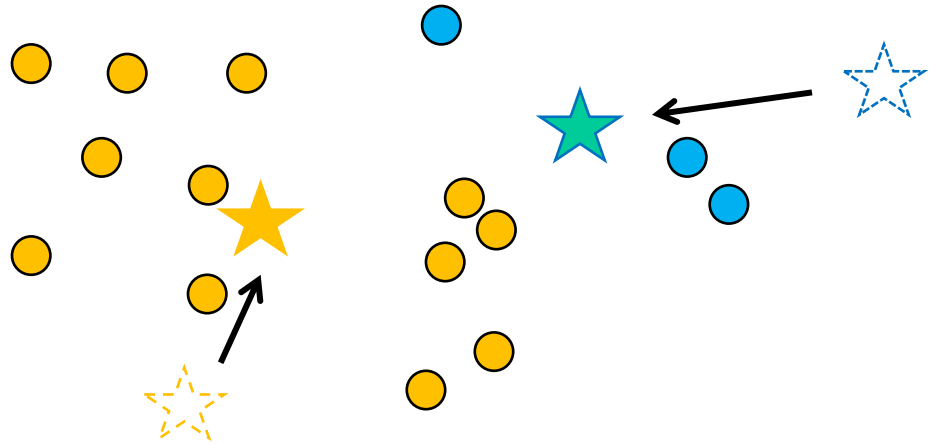
2. Find closest center for each point



K-Means Clustering: Algorithm

k-means clustering

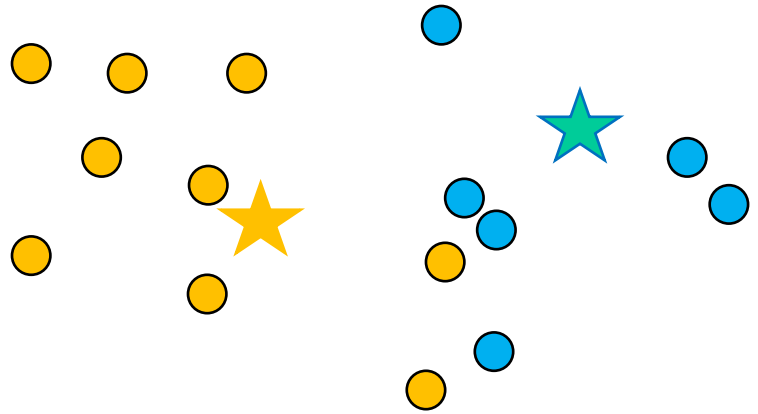
3. Update cluster centers by computing **centroids**



K-Means Clustering: Algorithm

k-means clustering

Repeat Steps 2 & 3 until convergence



K-means clustering (Lloyd's) algorithm

Input: # clusters k, points $x_1, \ldots, x_n$

Step 1: select k cluster centers $c_1, \ldots, c_k$

Step 2: for each point $x \in \{x_1, \ldots, x_n\}$ determine its nearest cluster center:

$$i_x = \text{argmin}_i \|x - c_i\|_2$$

Step 3: update cluster centers as the **centroids**:

$$c_i = \frac{\sum_{x:i_x=i} x}{|\{x \ : \ i_x = i\}|}$$

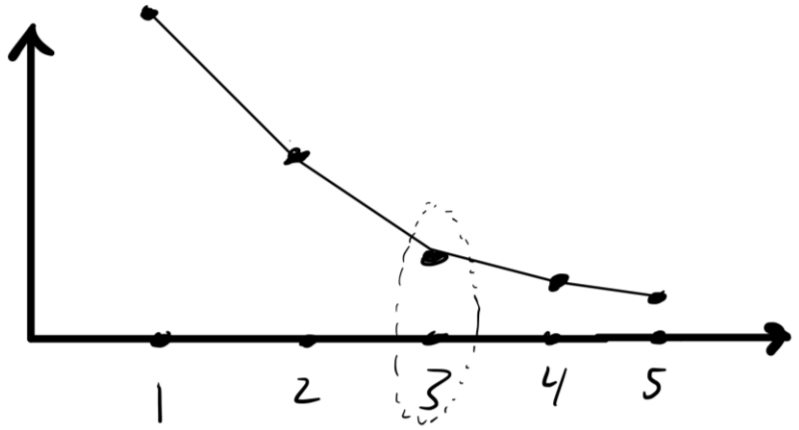
Repeat step 2 and 3 until the cluster centers no longer change

Questions on k-means

- What is k-means trying to optimize?

$$\sum_{x \in \{x_1, \ldots, x_n\}} \left\| x - c_{i_x} \right\|^2$$

- Will k-means stop (converge)?
- Will it find a global or local optimum?
- How many clusters should we use?
- How to pick starting cluster centers?



Chris De Sa

Mixture Models

- Let us get back to modeling probability densities, but **unconditionally**.
- Have dataset: $\{(x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$

- One type of model: **mixtures**
  - A function of a **latent variable** z
  - Model:

$$p(x^{(i)}|z^{(i)})p(z^{(i)})$$

Mixture Models: Gaussians

- Many different types of mixtures, but let us focus on Gaussians.
- What does this mean?
- Latent variable z has some multinomial distribution, $\sum_{i=1}^{k} \phi_i = 1$

$$z^{(i)} \sim \text{Multinomial}(\phi)$$

- Then, let us make x be Gaussian conditioned on z

$$x^{(i)}|(z^{(i)} = j) \sim \mathcal{N}(\mu_j, \Sigma_j)$$

Mean Covariance Matrix

Gaussian Mixture Models: Likelihood

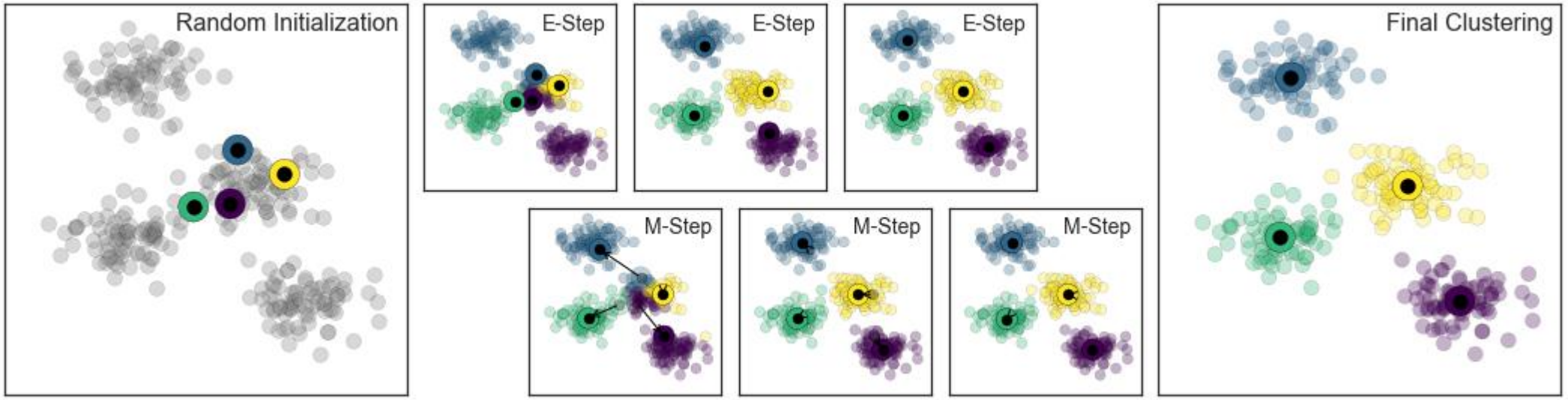
- How should we learn the parameters? ϕ, μ_j, Σ_j
- Could try our usual way: maximum likelihood
  - Log likelihood:

$$\ell(\phi, \mu, \Sigma) = \sum_{i=1}^{n} \log \sum_{z^{(i)}=1}^{k} p(x^{(i)}|z^{(i)}; \mu, \Sigma) p(z^{(i)}; \phi)$$

  - Turns out to be **hard** to solve… inner sum leads to problems!

GMMs: Expectation Maximization

- EM :an algorithm for dealing with latent variable problems
- Iterative, alternating between two steps:
  - **E**-step: estimate latent variable (probabilities) based on current model
  - **M**-step: update the parameters of $p(x|z)$
  - Note similarity to k-means clustering.



Jake VanderPlas

High-Dimensional Data

High-dimensions = lots of features

We've seen this repeatedly, but some examples:

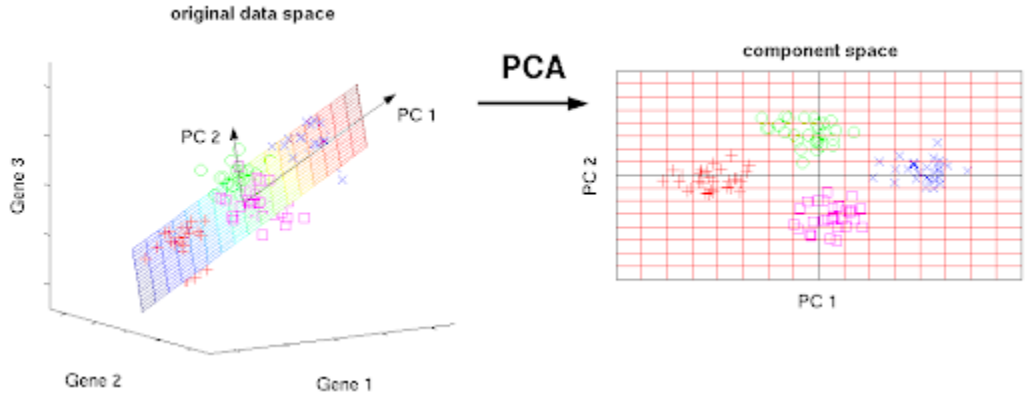
- Document classification
  - Features per document = thousands of words/unigrams, millions of bigrams, contextual information
- Surveys - Netflix

    480189 users x 17770 movies

	movie 1	movie 2	movie 3	movie 4	movie 5	movie 6
Tom	5	?	?	1	3	?
George	?	?	3	1	2	5
Susan	4	3	1	?	5	1
Beth	4	3	?	2	4	2

Dealing with Dimensionality

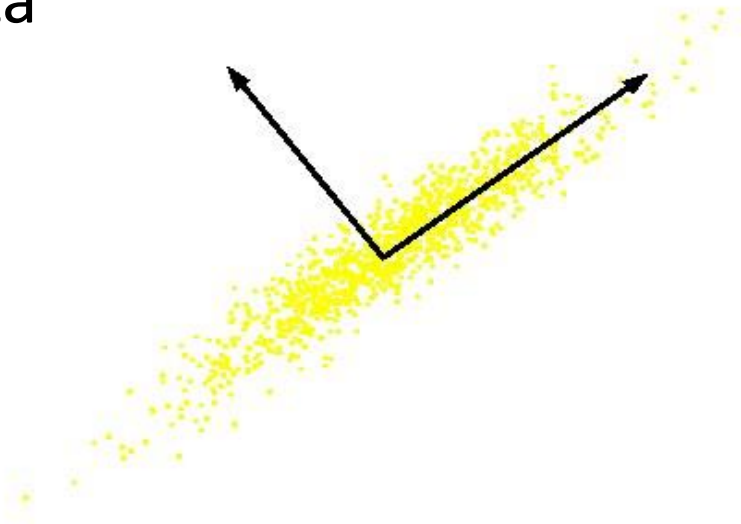
- **PCA, Kernel PCA, ICA**: Powerful unsupervised learning techniques for extracting hidden (potentially lower dimensional) structure from high dimensional datasets.
- Some uses:
  - Visualization
  - More efficient use of resources (e.g., time, memory, communication)
  - Noise removal (improving data quality)
  - Further processing by machine learning algorithms (representation transfer)



Principal Components Analysis

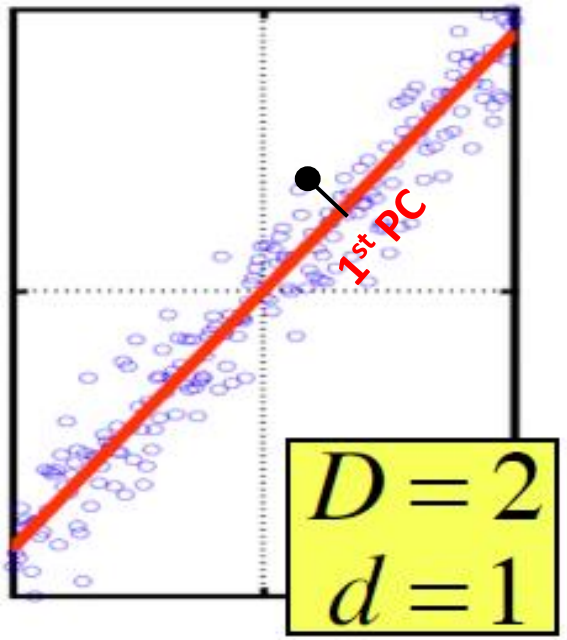
- Unsupervised technique for extracting variance structure from high dimensional datasets
  - also reduces dimensionality

- PCA: orthogonal projection / transformation of the data
  - Into a (possibly lower dimensional) subspace
  - Goal: maximize variance of the projected data



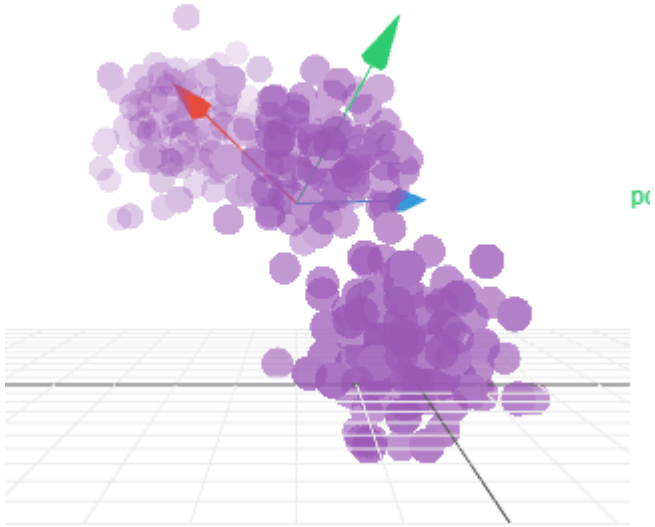
PCA: Principal Components

- **Principal Components (PCs)** are orthogonal directions that capture most of the variance in the data.
  - First PC – direction of greatest variability in data.
  - Projection of data points along first PC discriminates data most along any one direction



PCA: Principal Components and Projection

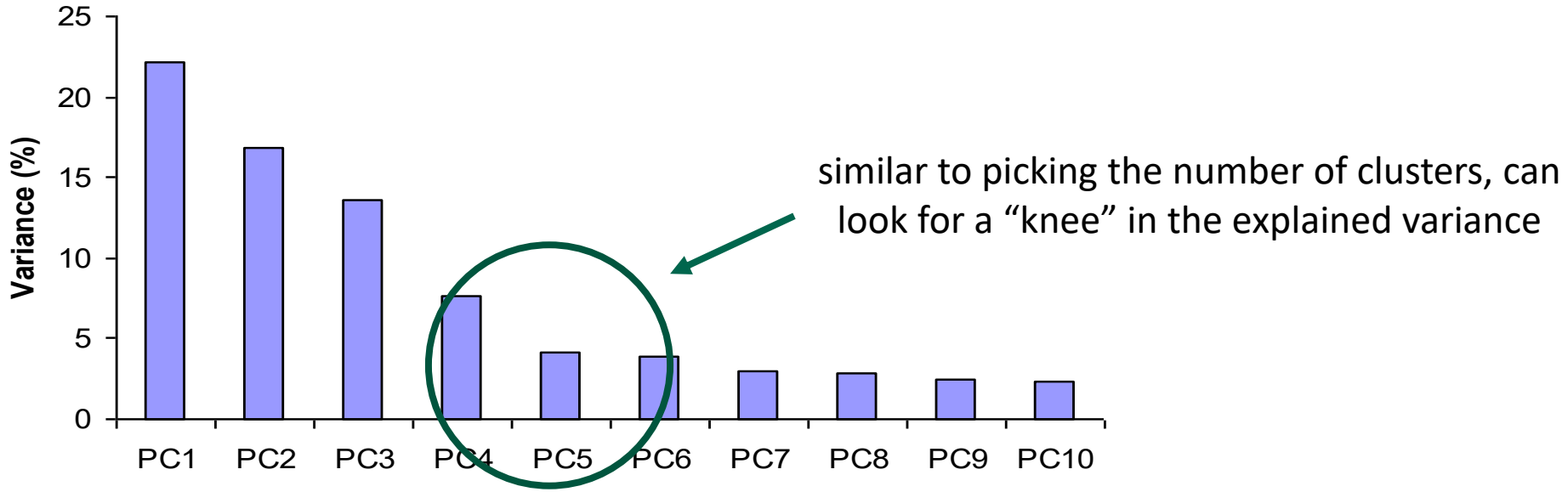
- How does dimensionality reduction work? From *d* dimensions to *r* dimensions:
  - Get orthogonal $v_1, v_2, \ldots, v_r \in \mathbb{R}^d$
- Maximizing variability
  - Equivalent to **minimizing reconstruction error**
- Then project data onto PCs → d-dimensional



Victor Powell

PCA Dimensionality Reduction

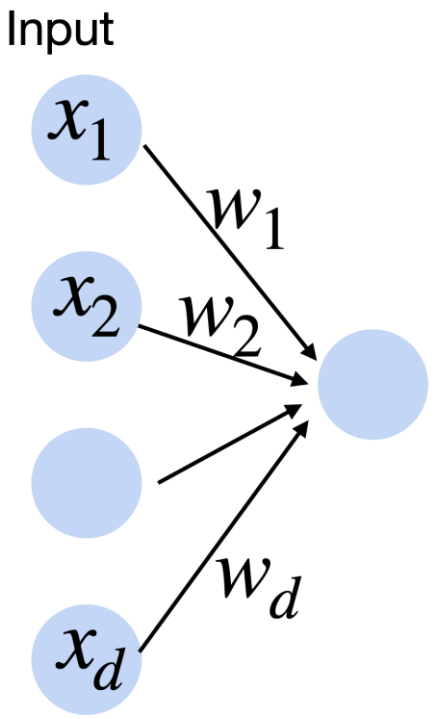
- In high-dimensional problems, data sometimes lies near a linear subspace, as noise introduces small variability
- Only keep data projections onto principal components with **large** eigenvalues
- Can *ignore* the components of smaller significance.



Outline

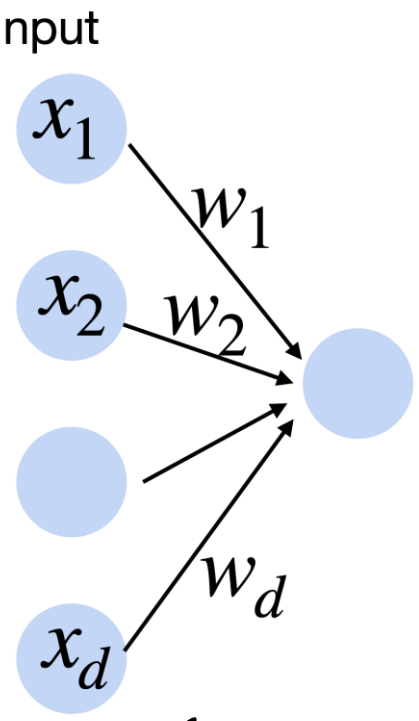
- **Instance-based learning:** k-NN, decision trees
- **Evaluation:** data splitting, metrics
- **Parametric modeling:** linear & logistic regression, regularization, MLE, MAP
- **Optimization:** gradient descent, SGD, convergence
- **Unsupervised learning:** centroid clustering, mixture models, PCA
- **Neural networks:** MLPs

Perceptron: Simple Network



$$\hat{y}(x) = \begin{cases} 1 & w^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Perceptron: Components



$$\hat{y}(x) = \begin{cases} 1 & w^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$w^T x$

$$\sigma(a) = \begin{cases} 1 & a \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{y}(x) = \sigma(w^T x)$$

Linear Transformation + **Activation Function**

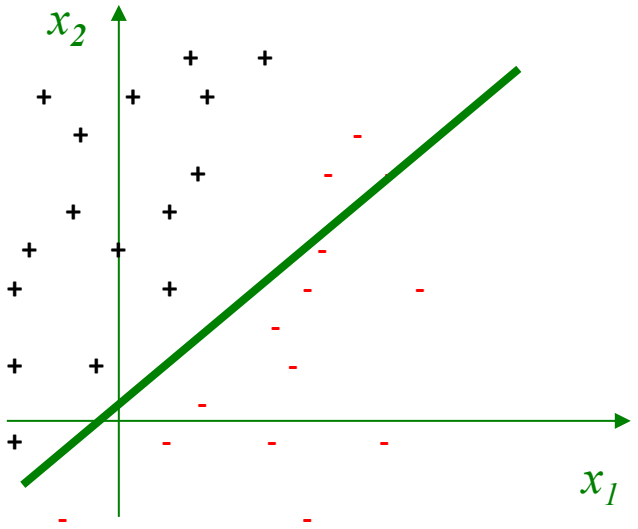
[McCulloch & Pitts, **1943**; Rosenblatt, **1959**; Widrow & Hoff, **1960**]

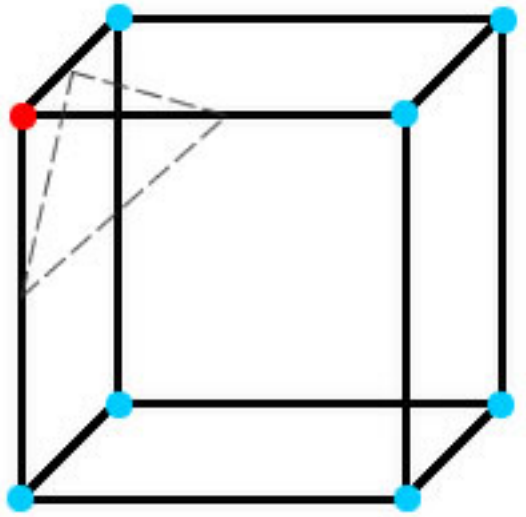
Perceptron: Representational Power

- Perceptrons can represent only *linearly separable* concepts

$$\hat{y}(x) = \begin{cases} 1 & w^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

- Decision boundary given by:

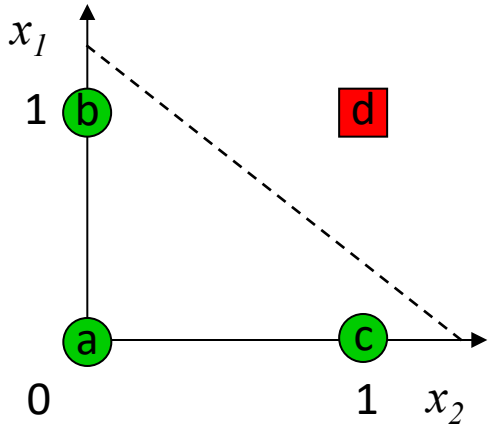




Which Functions are **Linearly Separable?**

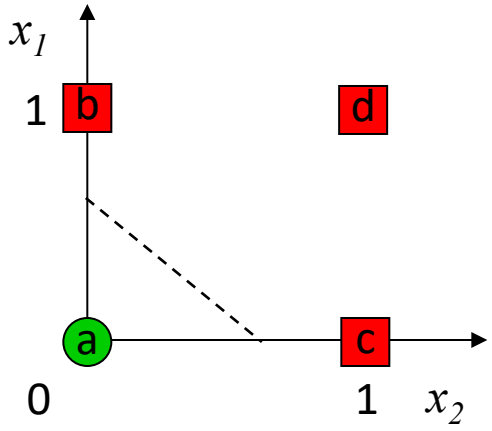
AND

	x_1	x_2	y
a	0	0	0
b	0	1	0
c	1	0	0
d	1	1	1



OR

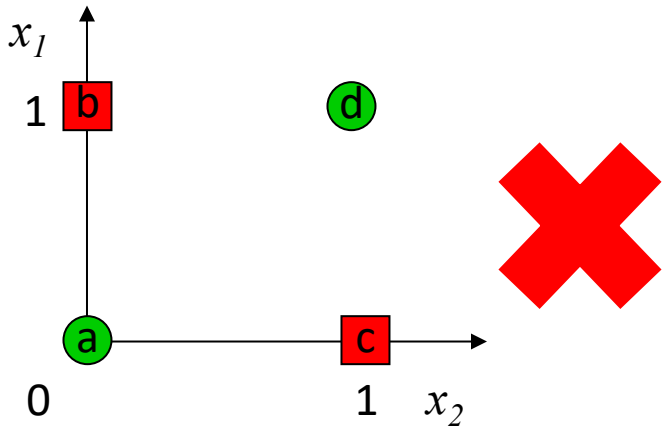
	x_1	x_2	y
a	0	0	0
b	0	1	1
c	1	0	1
d	1	1	1



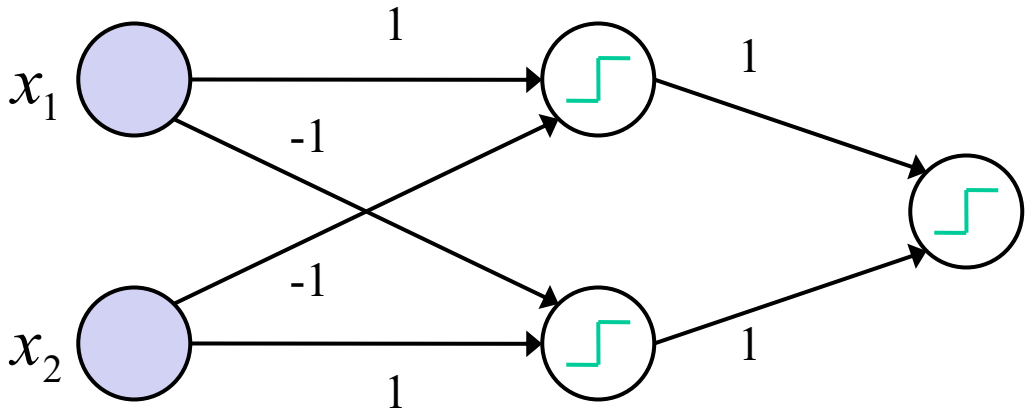
Which Functions are **Linearly Separable?**

XOR

	x_1	x_2	y
a	0	0	0
b	0	1	1
c	1	0	1
d	1	1	0



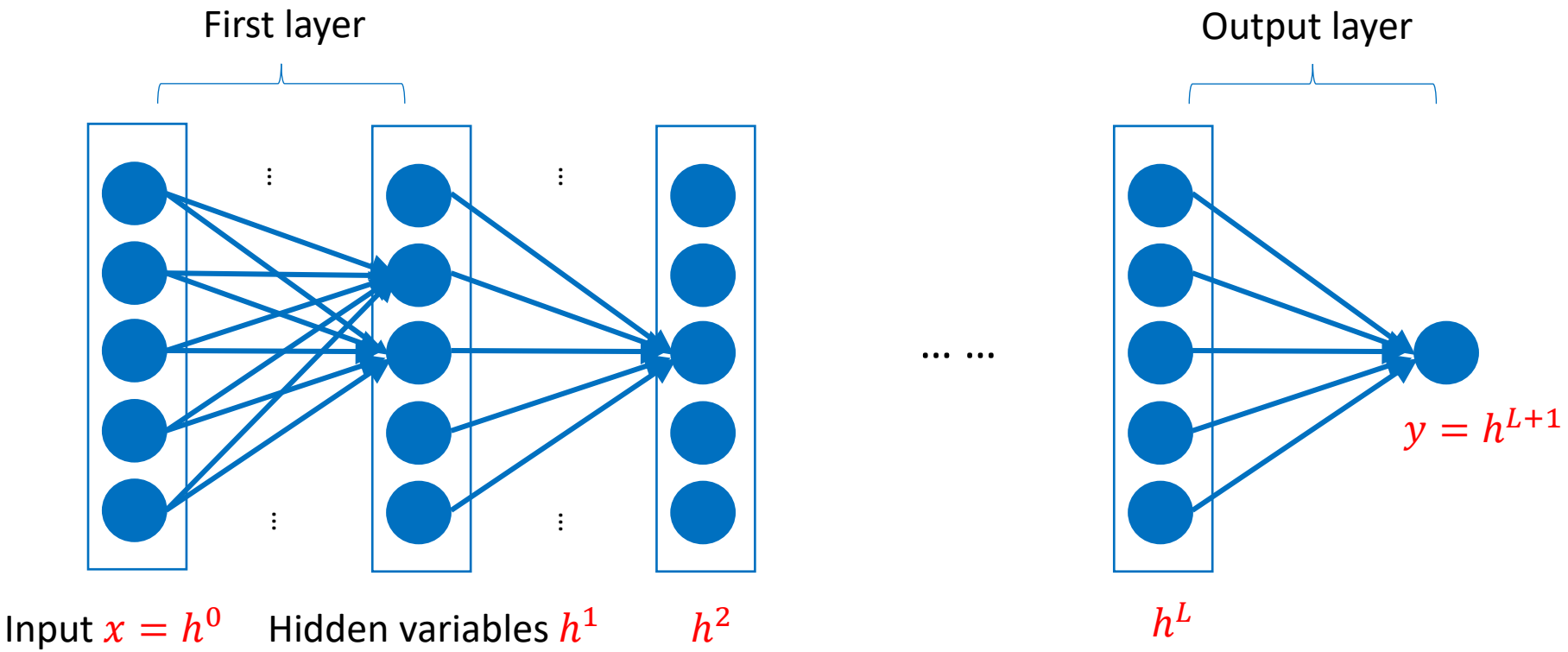
A multilayer perceptron can represent XOR!



(assume activation is $\sigma(x) = 1_{\{x>0\}}$)

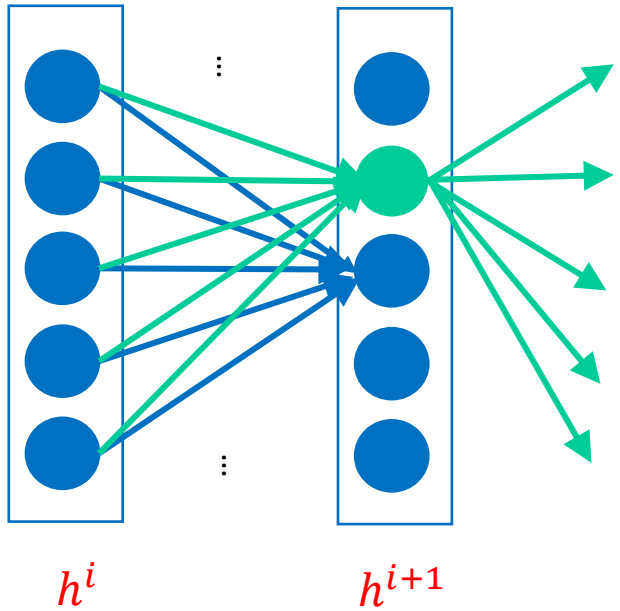
Neural Network Components

An $(L+1)$-layer network



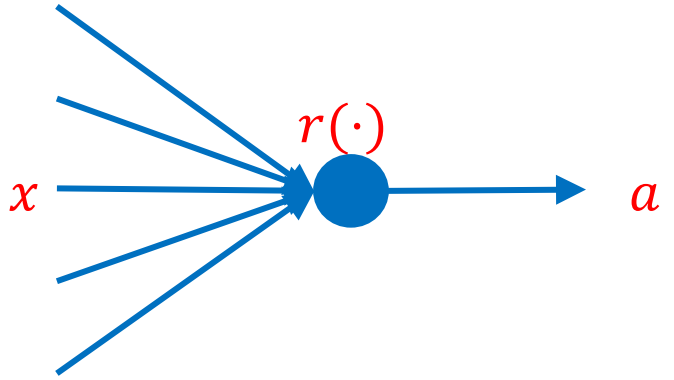
Hidden Layers

- Neuron takes weighted linear combination of the previous representation layer
  - Outputs one value for the next layer



Hidden Layers

- Outputs $a = r(w^T x + b)$

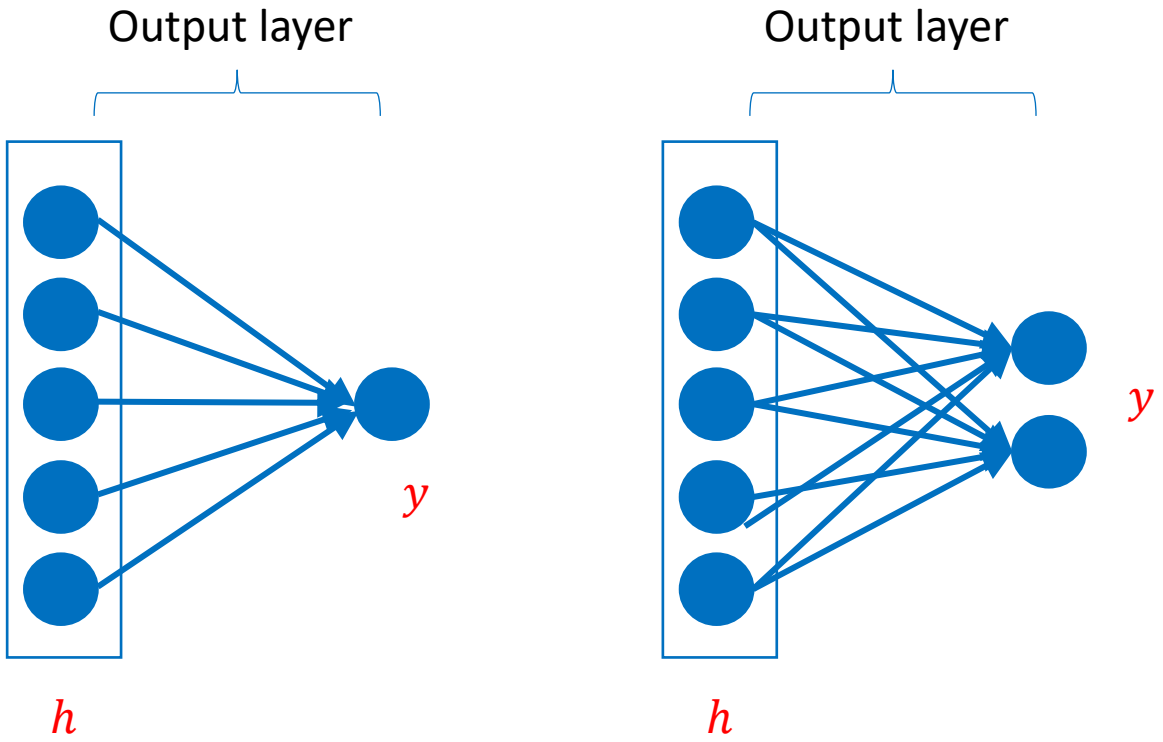


- Typical activation function r
  - threshold $\mathrm{h}(z) = 1_{\{z \geq 0\}}$
  - ReLU $\mathrm{ReLU}(z) = z \cdot t(z) = \max\{0, z\}$
  - sigmoid $\sigma(z) = 1/(1 + \exp(-z))$
  - hyperbolic tangent $\tanh(z) = 2\sigma(2z) - 1$

- Why not **linear activation** functions?
  - Model would be linear.

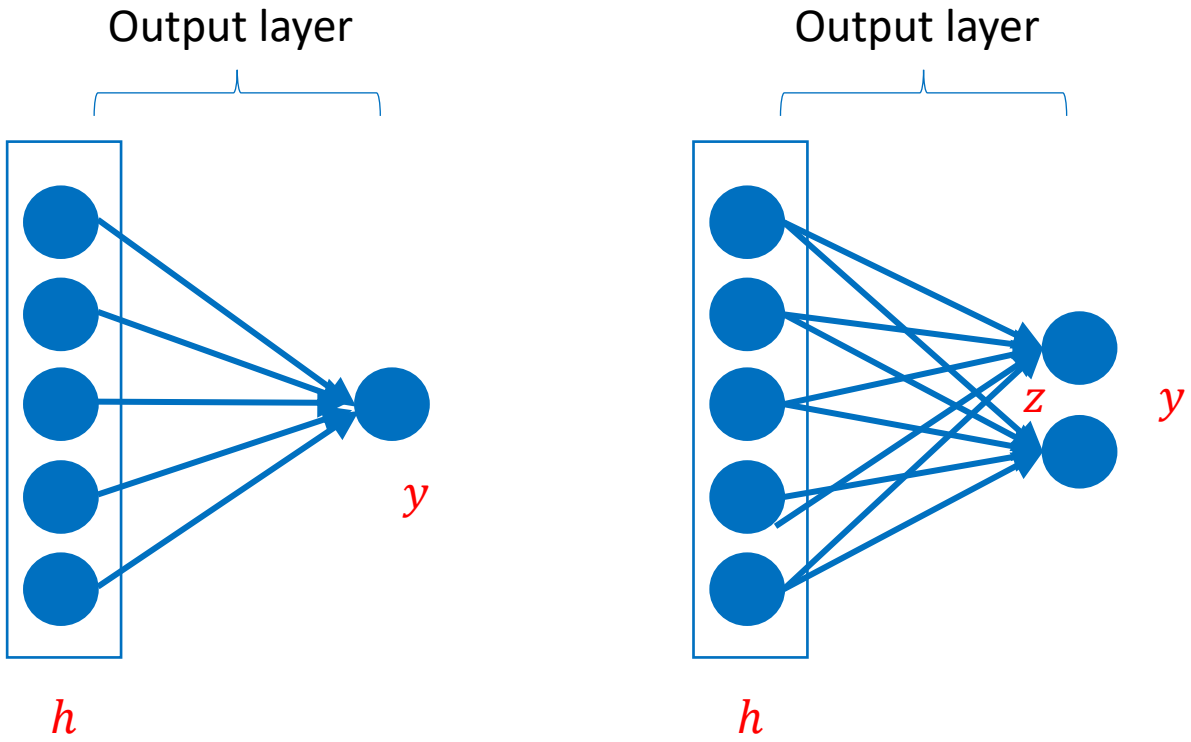
Output Layer: Examples

- Regression: $y = w^T h + b$
  - Linear units: no nonlinearity
- Multi-dimensional regression: $y = W^T h + b$
  - Linear units: no nonlinearity



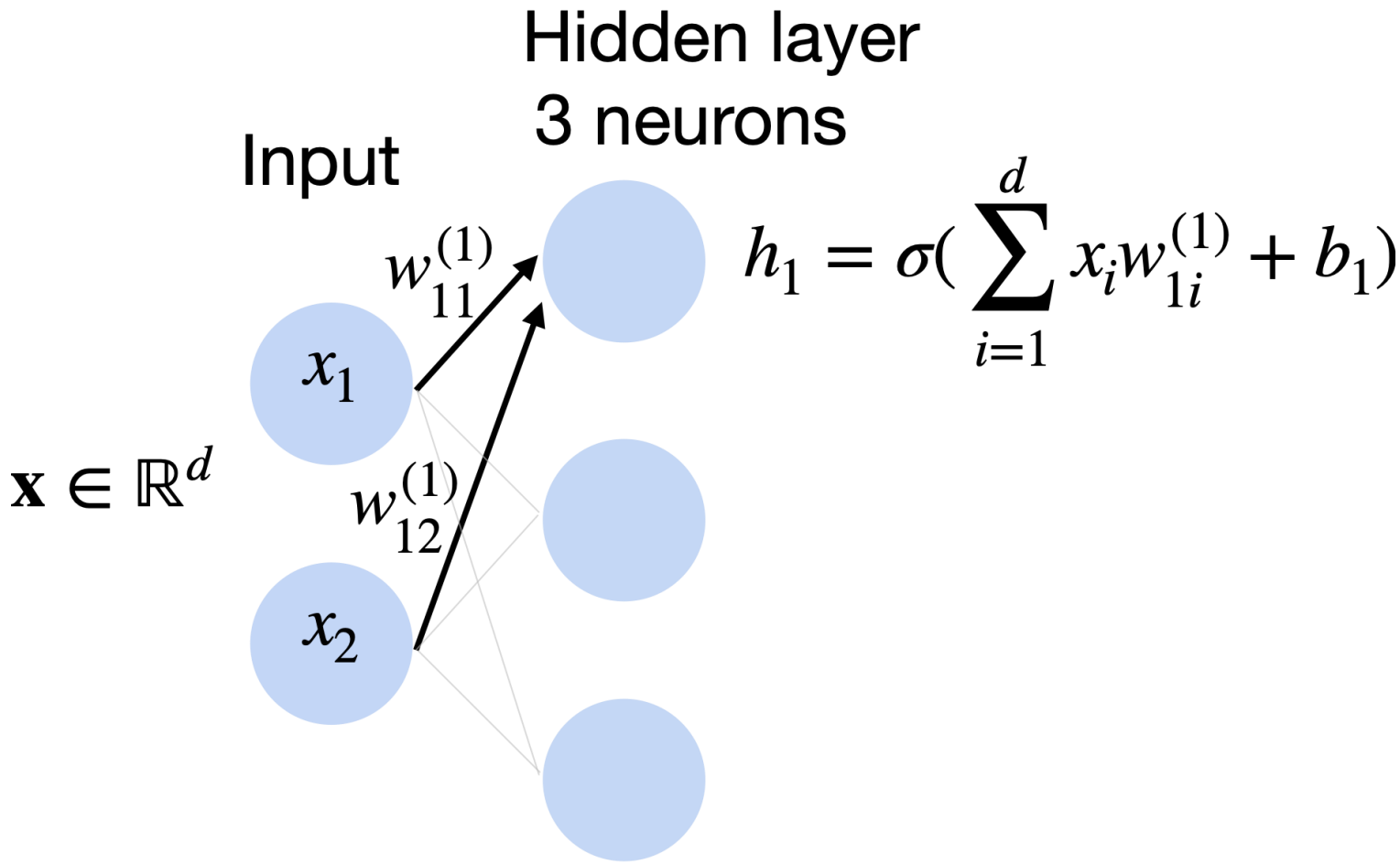
Output Layer: Examples

- Binary classification: $y = \sigma(w^T h + b)$
  - Corresponds to using logistic regression on h
- Multiclass classification:
  - $y = \text{softmax}(z)$ where $z = W^T h + b$



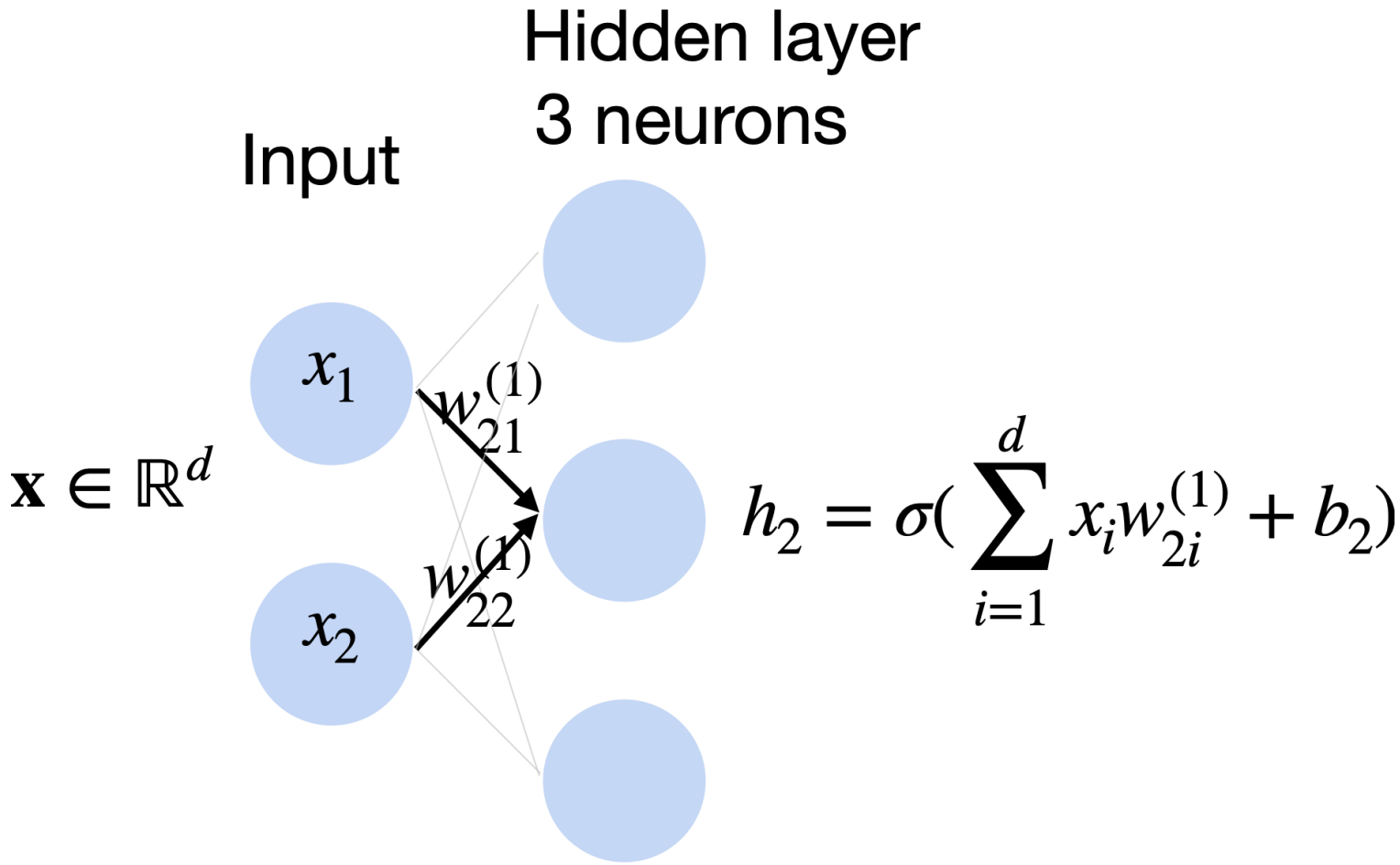
MLPs: Multilayer Perceptron

- **Ex**: 1 hidden layer, 1 output layer: depth 2



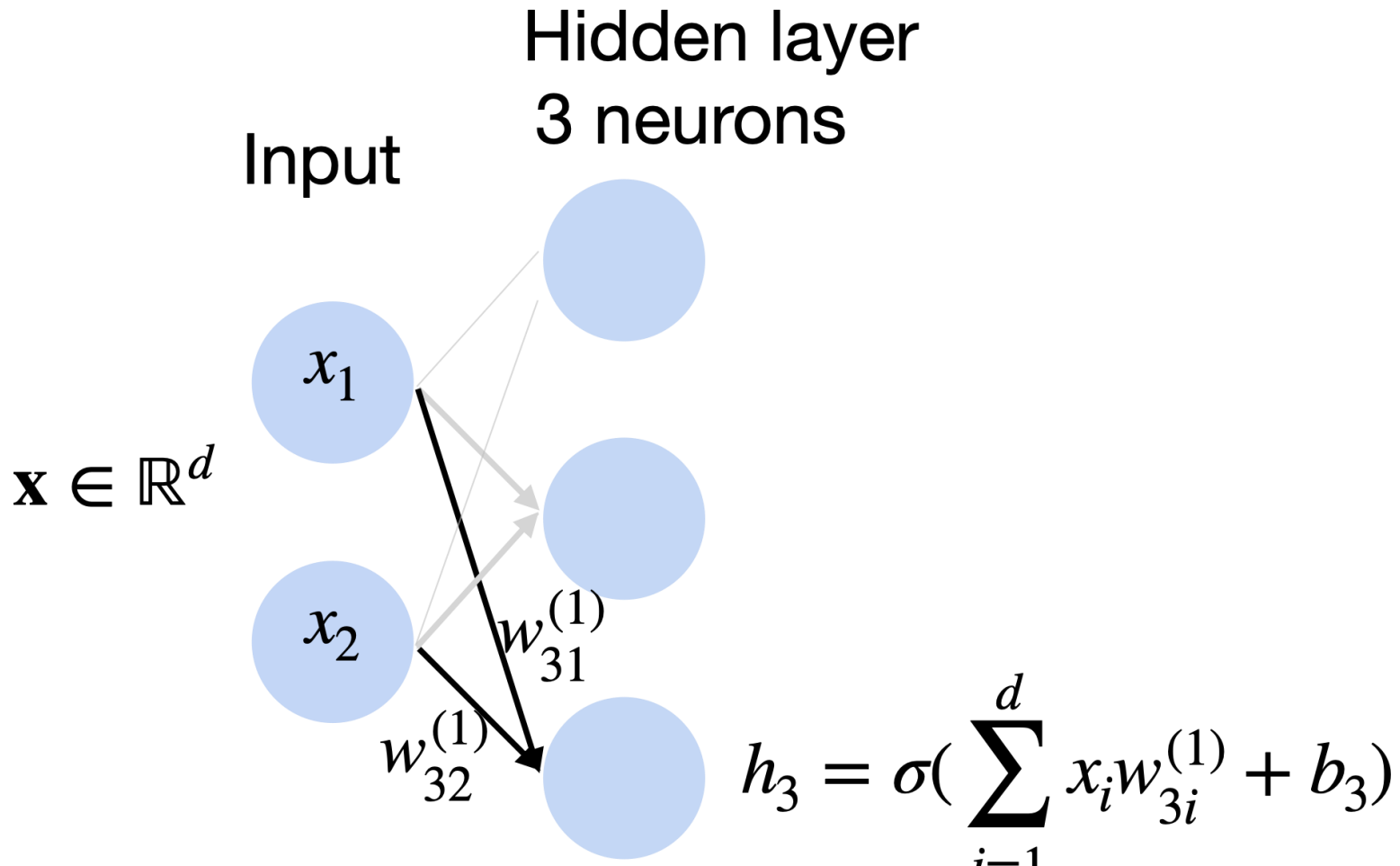
MLPs: Multilayer Perceptron

- **Ex**: 1 hidden layer, 1 output layer: depth 2



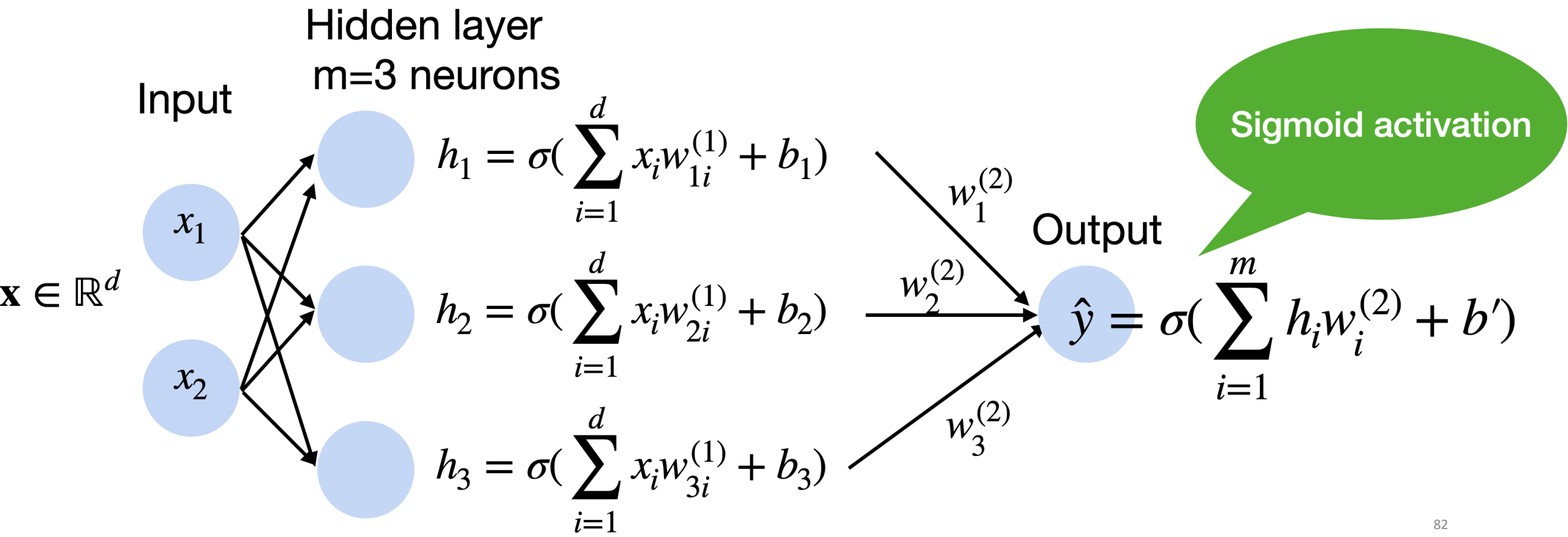
MLPs: Multilayer Perceptron

- **Ex**: 1 hidden layer, 1 output layer: depth 2



MLPs: Multilayer Perceptron

- **Ex**: 1 hidden layer, 1 output layer: depth 2



Thanks Everyone!

Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, Pedro Domingos, Jerry Zhu, Yingyu Liang, Volodymyr Kuleshov , Sharon Li, Chris Olah, Fred Sala, Tengyang Xie, Josiah Hanna, Kirthi Kandasamy