CSC580: Principles of Machine Learning

Final Exam Review

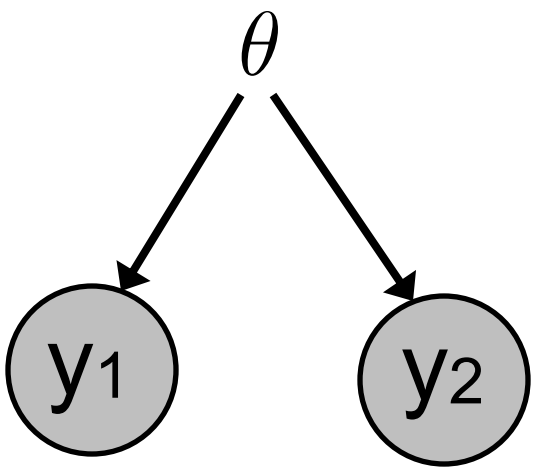
Jason Pacheco

Final Exam

- Similar format to Midterm but take-home
- Some students need to take it early so I will release at the start of next week
- 6+1 Questions
  - 1 of these is only for CSC580 students
  - No coding

Learning

Learning / Training



Model random data with hyperparameters θ :

$$y \sim p(y \mid \theta)$$

Sometimes we use: $p(y; \theta)$

Given training data:

$$\{y_i\}_{i=1}^{n} \overset{\text{i.i.d.}}{\sim} p(y \mid \theta)$$

Learn parameters, e.g. via *maximum likelihood estimation*:

$$\hat{\theta}^{\text{MLE}} = \arg\max_{\theta} \log p(y_1, \ldots, y_n \mid \theta)$$

We will talk more about MLE in coming weeks

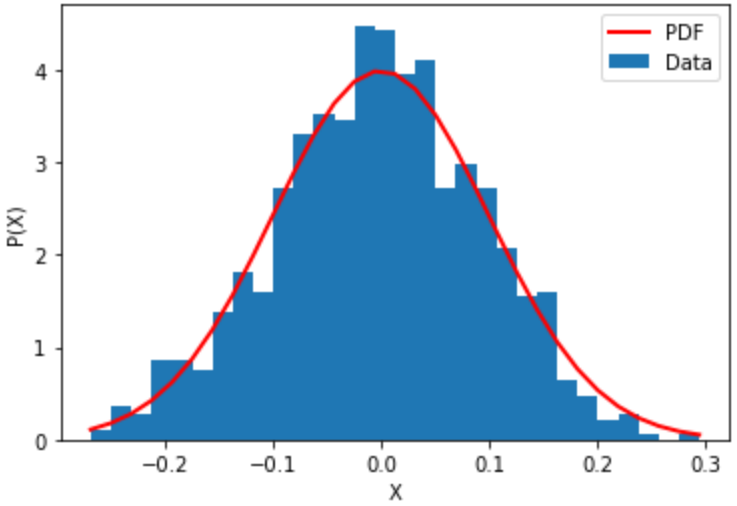
Other estimators are possible:

- *Maximum a posteriori* (MAP)
- *Minimum mean squared error (MMSE)*
- *Etc.*

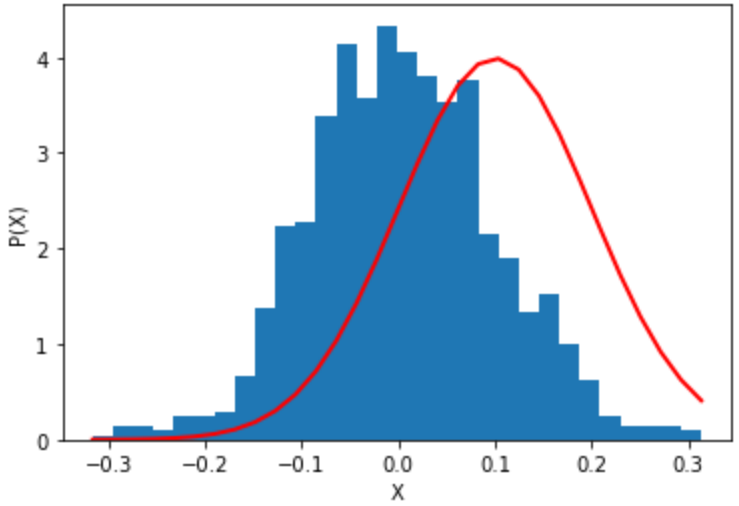
Likelihood (Intuitively)

Suppose we observe N data points from a Gaussian model and wish to estimate model parameters…

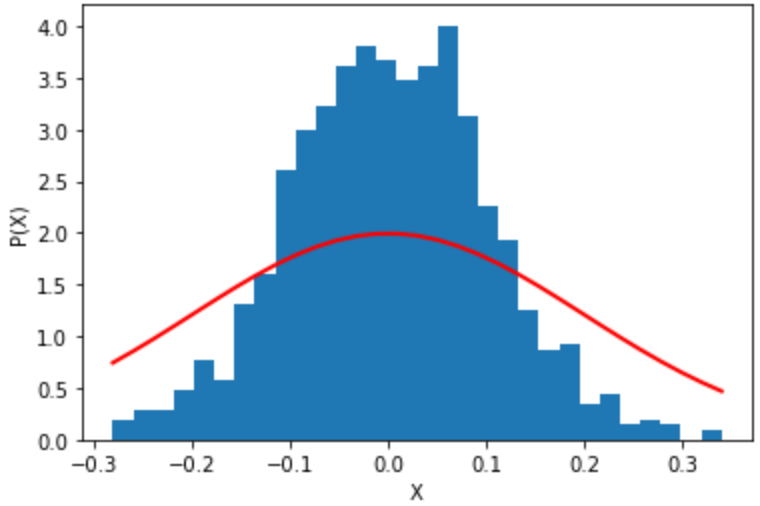
High Likelihood



Low Likelihood (mean)



Low Likelihood (variance)



Likelihood Principle *Given a statistical model, the likelihood function describes all evidence of a parameter that is contained in the data.*

Likelihood Function

Suppose $x_i \sim p(x;\theta)$, then what is the **joint probability** over N *independent identically distributed* (iid) observations $x_1, \ldots, x_N$?

$$p(x_1, \ldots, x_N; \theta) = \prod_{i=1}^{N} p(x_i; \theta)$$

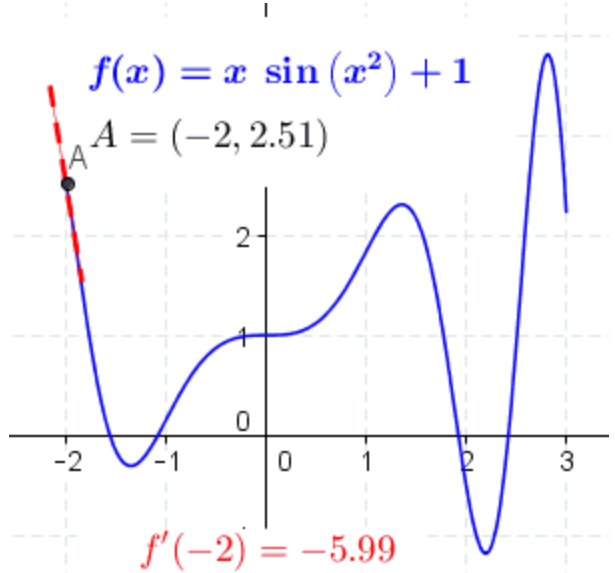
- We call this the **likelihood function**, often denoted $\mathcal{L}_N(\theta)$
- It is a function of the parameter θ, the data are fixed
- Measures how well parameter θ describes data (*goodness of fit*)

How could we use this to estimate a parameter θ?

Maximum Likelihood

Maximum Likelihood Estimator (MLE) as the name suggests, maximizes the likelihood function.

$$\hat{\theta}^{\mathrm{MLE}} = \arg\max_{\theta} \mathcal{L}_N(\theta) = \prod_{i=1}^{N} p(x_i; \theta)$$



Question How do we find the MLE?

Answer Remember calculus…

Is $\mathcal{L}_N(\theta)$ convex?

Yes → Unique, closed-form solution

No → Gradient-based optimization

Approach

- Compute derivative $\frac{d\mathcal{L}_N}{d\theta}$
- Set to zero and solve

$$\frac{d\mathcal{L}_N}{d\theta} = 0 \Rightarrow \hat{\theta}^{\mathrm{MLE}}$$

Still have to compute derivative…

Maximum Likelihood

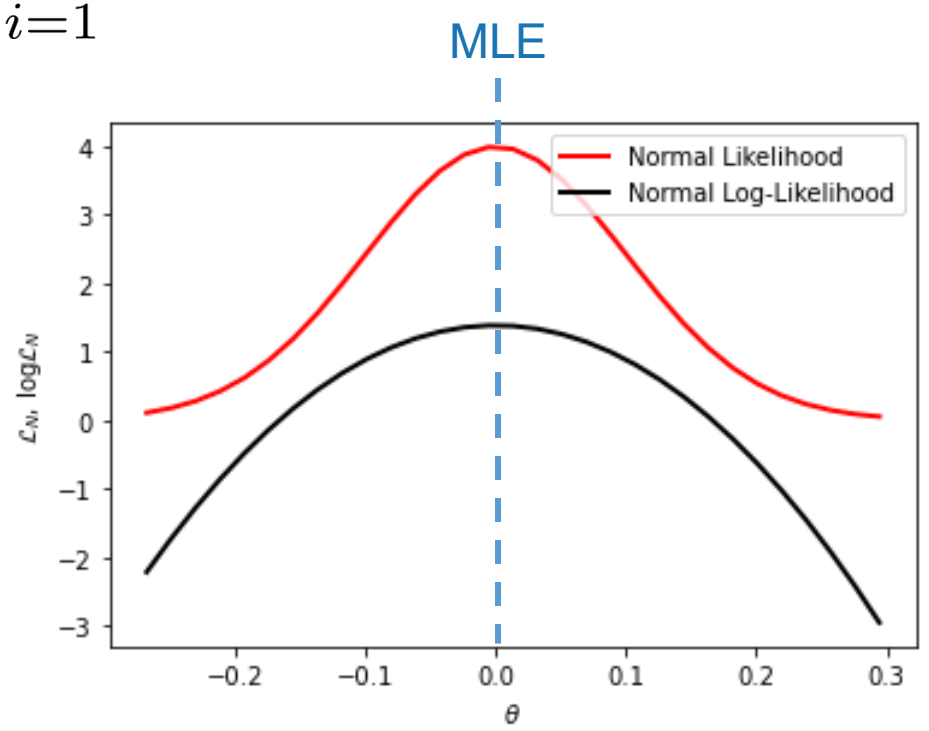
Maximizing log-likelihood makes the math easier (as we will see) and doesn't change the answer (logarithm is an increasing function)

$$\hat{\theta}^{\mathrm{MLE}} = \arg\max_{\theta} \log \mathcal{L}_N(\theta) = \sum_{i=1}^{N} \log p(x_i; \theta)$$

Derivative is a linear operator so,

$$\frac{d}{d\theta} \log \mathcal{L}_N(\theta) = \sum_{i=1}^{N} \frac{d}{d\theta} \log p(x_i; \theta)$$

One term per data point
Can be computed in parallel
(big data)

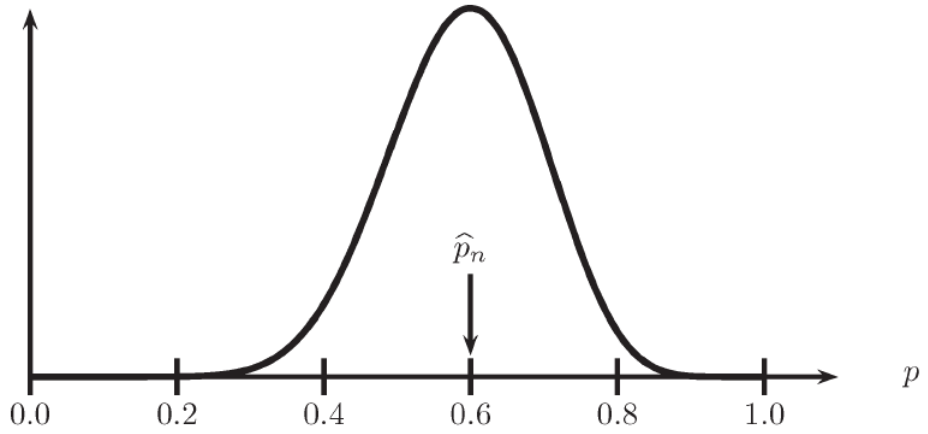


Maximum Likelihood

[Source: Wasserman, L. 2004]

Example Suppose we have N coin tosses with $X_1, \ldots, X_n \sim \text{Bernoulli}(p)$ but we don't know the coin bias p. The likelihood function is,

$$\mathcal{L}_n(p) = \prod_{i=1}^{n} p^{x_i}(1-p)^{1-x_i} = p^S(1-p)^{n-S}$$



Likelihood function for Bernoulli with n=20 and $\sum_i x_i = 12$ heads

where $S = \sum_i x_i$. The log-likelihood is,

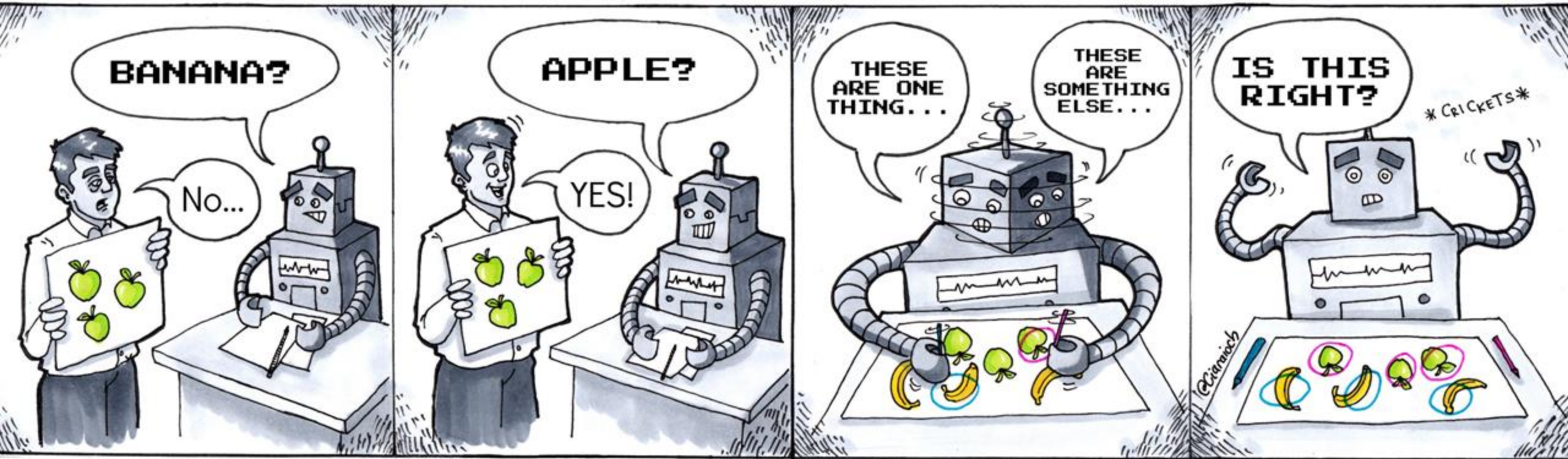
$$\log \mathcal{L}_n(p) = S \log p + (n - S) \log(1 - p)$$

Set the derivative of $\log \mathcal{L}_n(p)$ to zero and solve,

$$\hat{p}^{\text{MLE}} = S/n = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Maximum likelihood is equivalent to sample mean in Bernoulli

K-Means Clustering



Supervised Learning

Unsupervised Learning

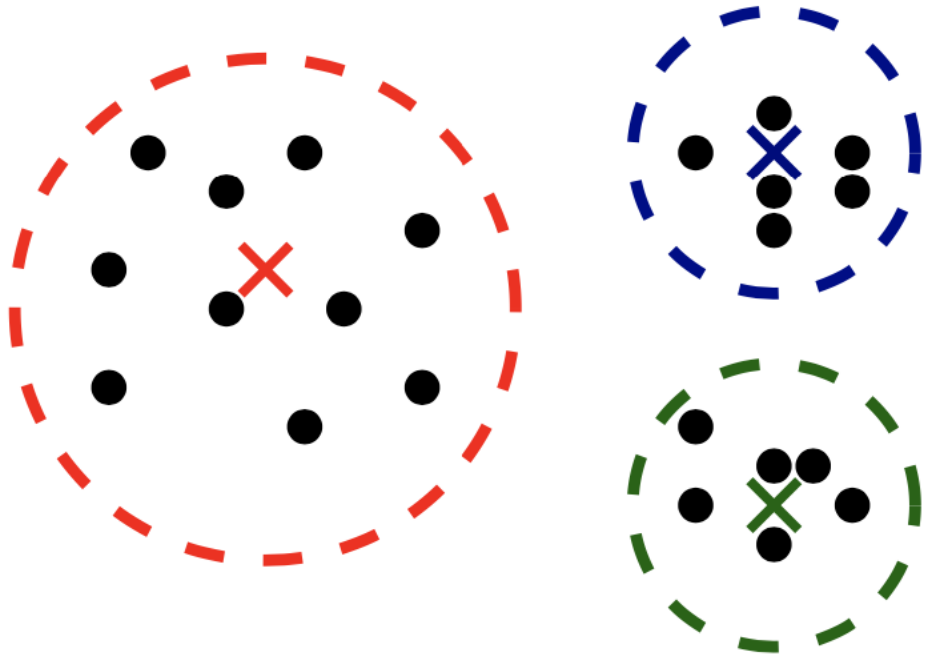
Clustering

- Input: k: the number of clusters (hyperparameter)

$$S = \{x_1, \dots, x_n\}$$

- Output
  - partition $\{G_i\}_{i=1}^{k}$ s.t. $S = \cup_i G_i$ (disjoint union).
  - often, we also obtain 'centroids'

- Q: what would be a reasonable definition of centroids?



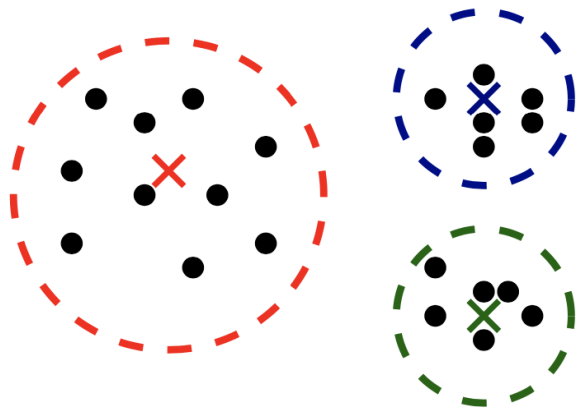
k-means clustering

- Idea: to partition the data, it would be great if someone gives us k reasonable centroids $c_1, \dots, c_k$, since then we can partition the data with them.

$$A(x) = \arg\min_{j \in [k]} \|x - c_j\|_2$$

- But we don't have those centroids => Let's find them with an optimization formulation.

$$\underset{c_1,\dots,c_k}{\text{minimize}}\ f(c_1, \dots, c_k), \text{ where } f(c_1, \dots, c_k) = \sum_{i=1}^{n} \min_{j \in [k]} \|x - c_j\|_2^2$$



Special case: k=1

- $\min_{c_1,\ldots,c_k} \sum_{i=1}^{n} \min_{j\in[k]} \|x_i - c_j\|_2^2$ => $\min_{c} \sum_{i=1}^{n} \|x_i - c\|_2^2$

- Let $F(c) = \sum_{i=1}^{n} \|x_i - c\|_2^2$ convex; minimizer c^* satisfies that $\nabla F(c^*) = 0$

  => $\sum_{i=1}^{n} (x_i - c^*) = 0$ => $c^* = \frac{1}{n} \sum_{i=1}^{n} x_i$

For $k \geq 2$

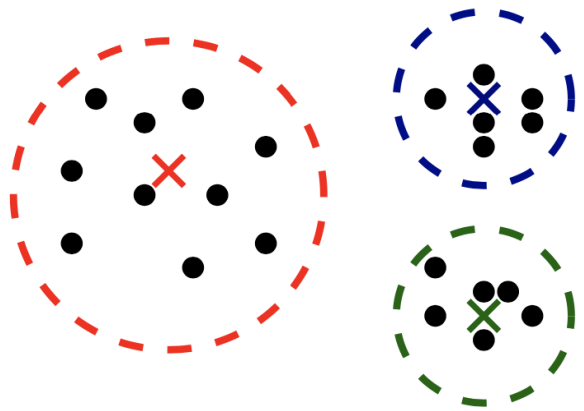
- $\underset{c_1,\ldots,c_k}{\text{minimize}}\, f(c_1, \ldots, c_k)$, where $f(c_1, \ldots, c_k) = \sum_{i=1}^{n} \min_{j \in [k]} \left\| x - c_j \right\|_2^2$ => NP-hard even when $d = 2$

- **K-means algorithm**: solve it approximately (heuristic)

(Also called Lloyd's algorithm)

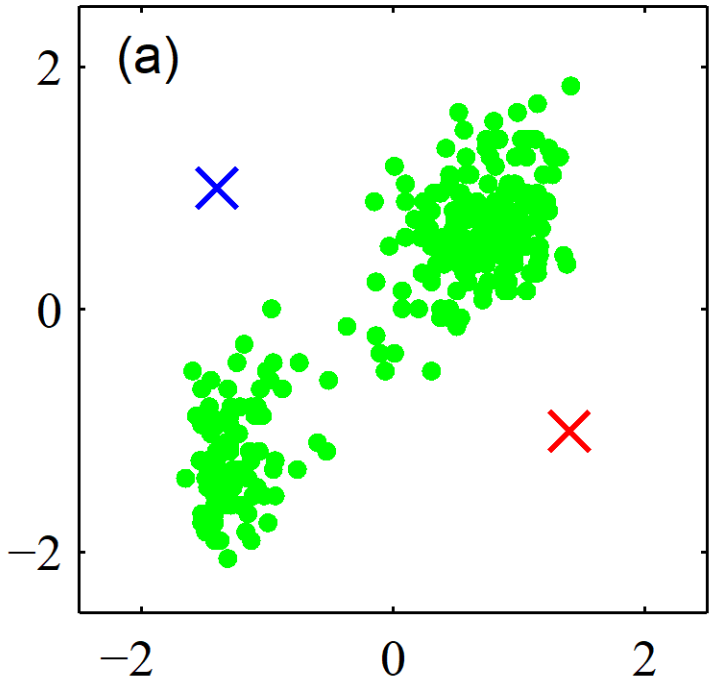
- Observation: The chicken-and-egg problem.
  - Cluster center location depends on the cluster assignment
  - Cluster assignment depends on cluster location

- Very common heuristic (that may or may not be the best thing to do)



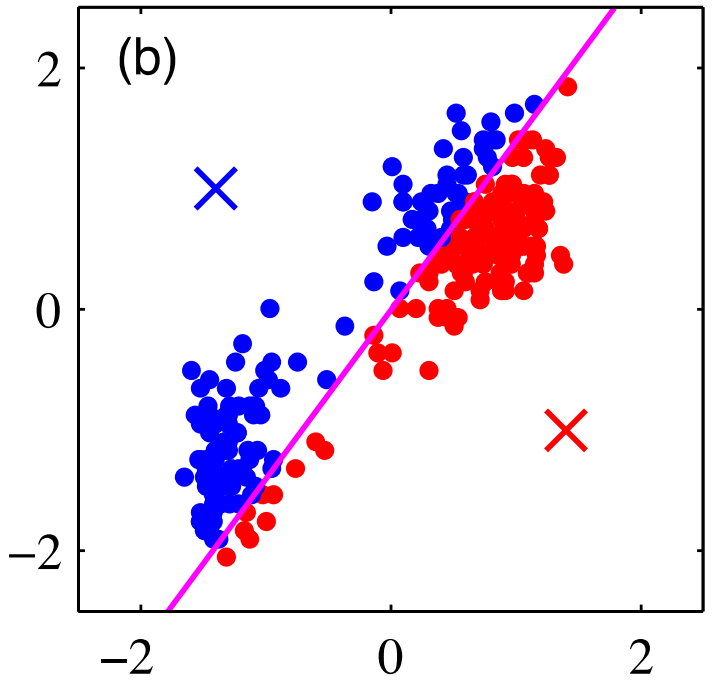
Andrea Vattani, "the hardness of k-means clustering in the plane"

Initialization

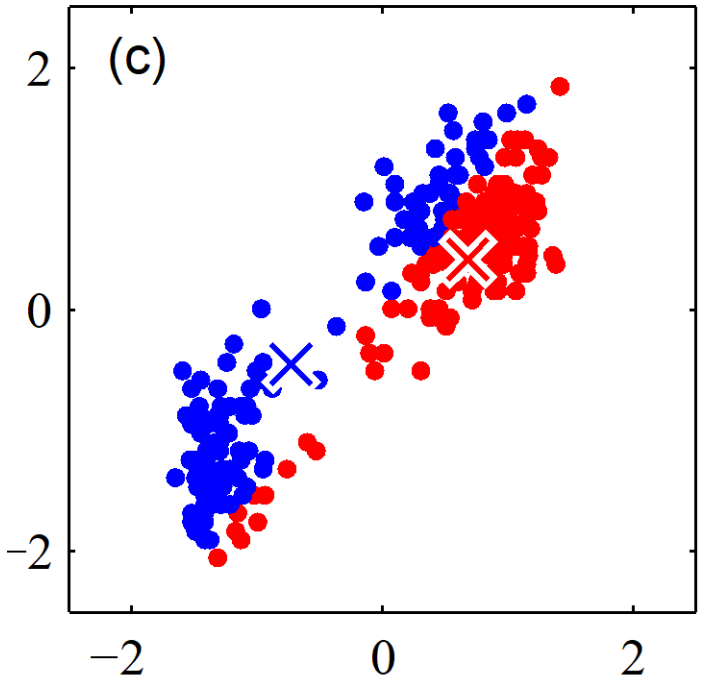


Arbitrary/random initialization of c_1 and c_2

Iteration 1

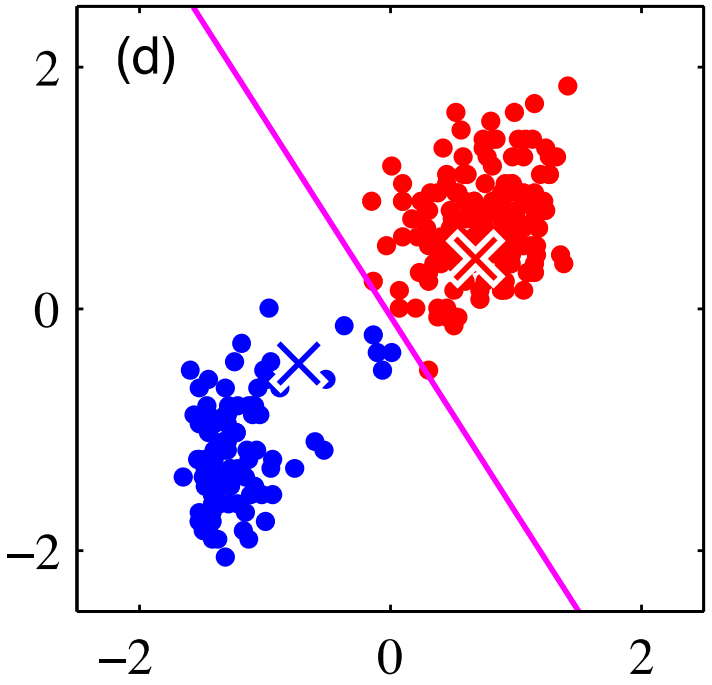


(A) update the cluster assignments.

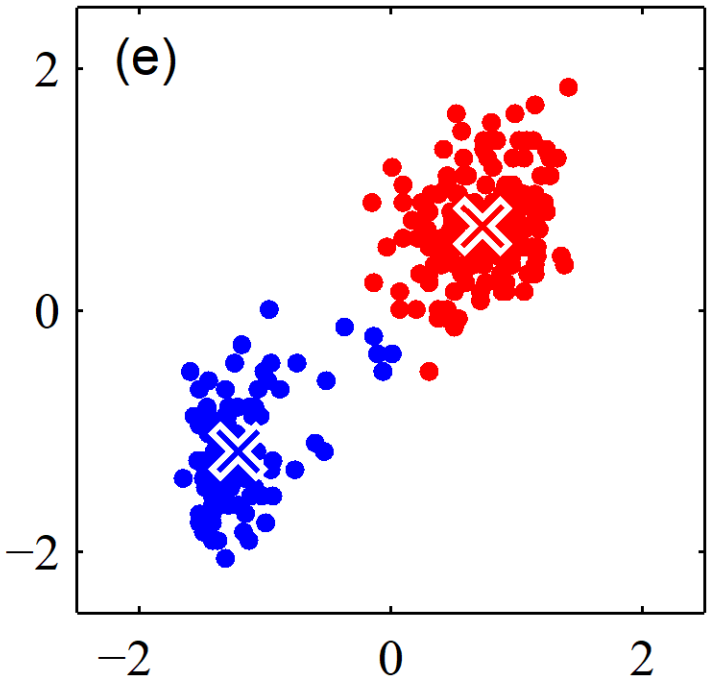


(B) Update the centroids $\{c_j\}$

Iteration 2

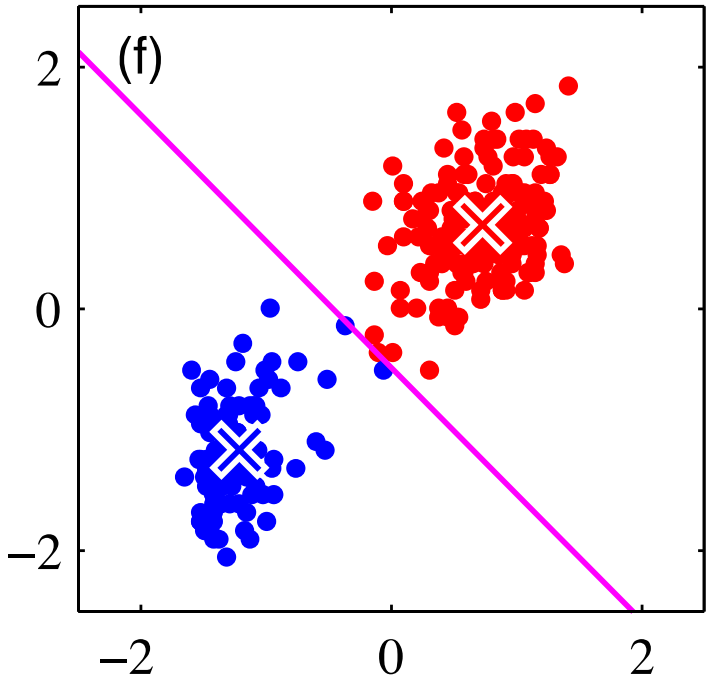


(A) update the cluster assignments.

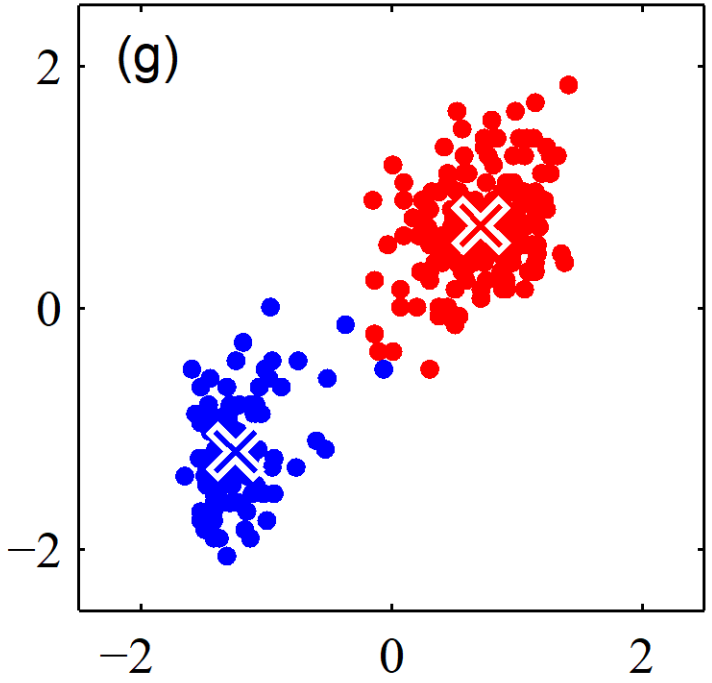


(B) Update the centroids $\{c_j\}$

Iteration 3

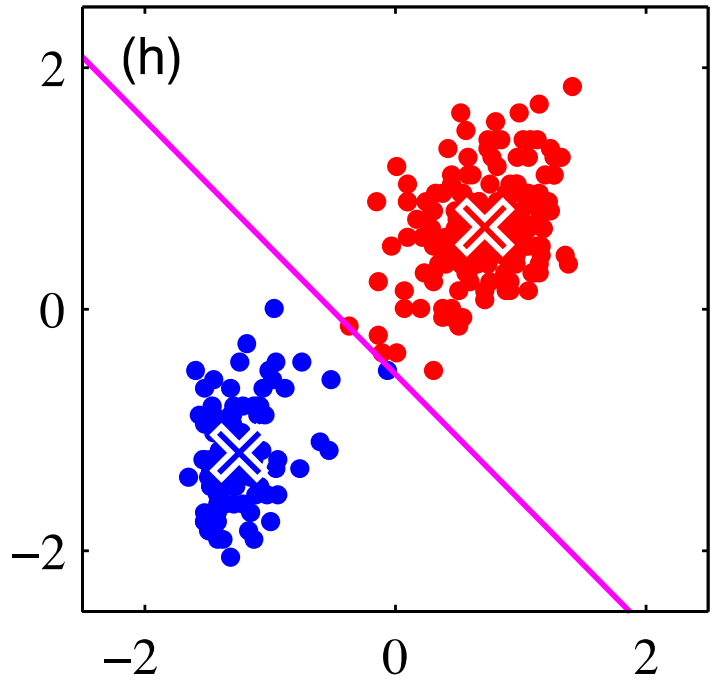


(A) update the cluster assignments.

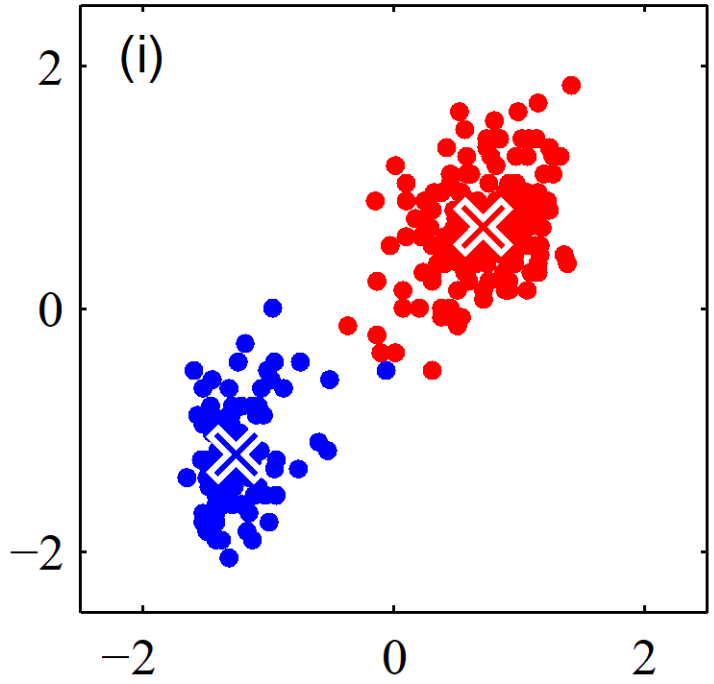


(B) Update the centroids $\{c_j\}$

Iteration 4



(A) update the cluster assignments.



(B) Update the centroids $\{c_j\}$

K-means clustering

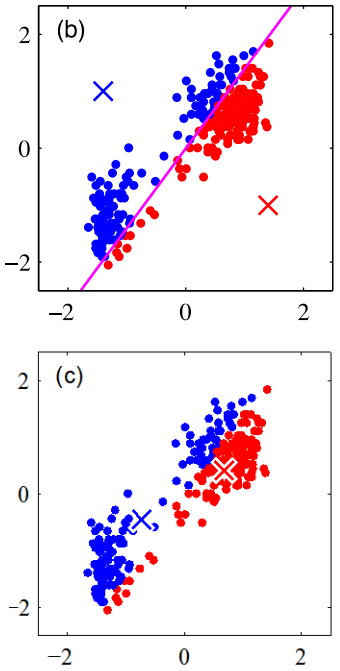
Input: k: num. of clusters, $S = \{x_1, \dots, x_n\}$

[Initialize] Pick $c_1, \dots, c_k$ as randomly selected points from S (see next slides for alternatives)

For t=1,2,…,max_iter

- ***[Assignments]*** $\forall x \in S, \quad a_t(x) = \arg\min_{j \in [k]} \|x - c_j\|_2^2$
- If t $\neq 1$ AND $a_t(x) = a_{t-1}(x), \forall x \in S$
  - break
- ***[Centroids]*** $\forall j \in [k], \quad c_j \leftarrow \text{average}(\{x \in S: a_t(x) = j\})$

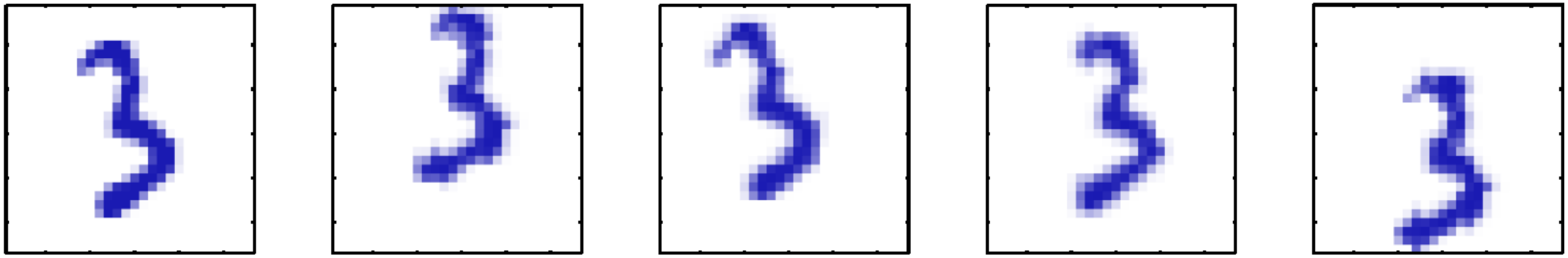
Output: $c_1, \dots, c_k$ and $\{a_t(x_i)\}_{i \in [n]}$



Dimensionality Reduction
and Principal Component Analysis (PCA)

Motivation

Data often have a lot of redundant information…



Example A dataset consisting of a hand-drawn 3 at random locations and rotations in a 100x100 pixel image.

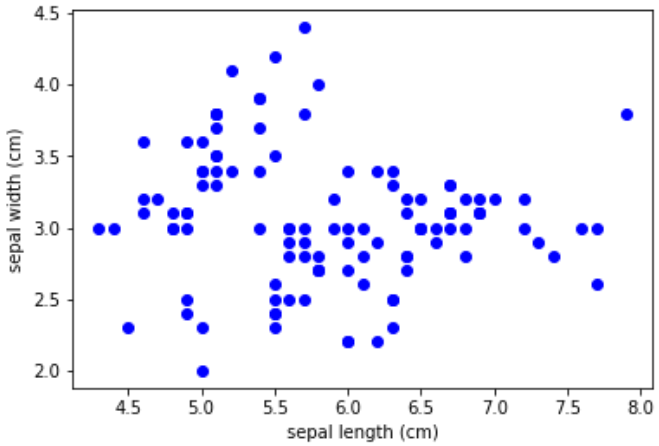
Data Dimension 100 x 100 = 10,000

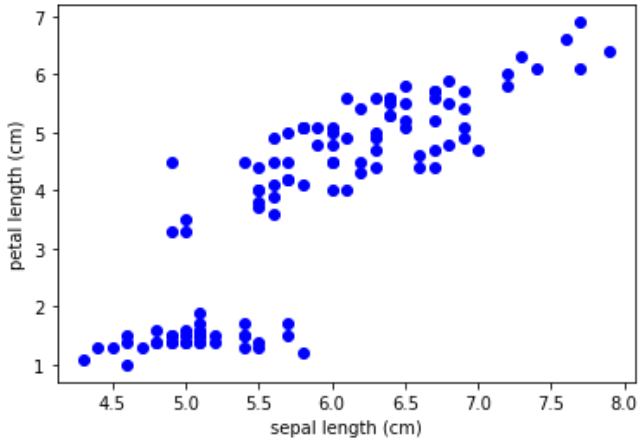
Intrinsic Dimension 3 (X-position, Y-position, Rotation)

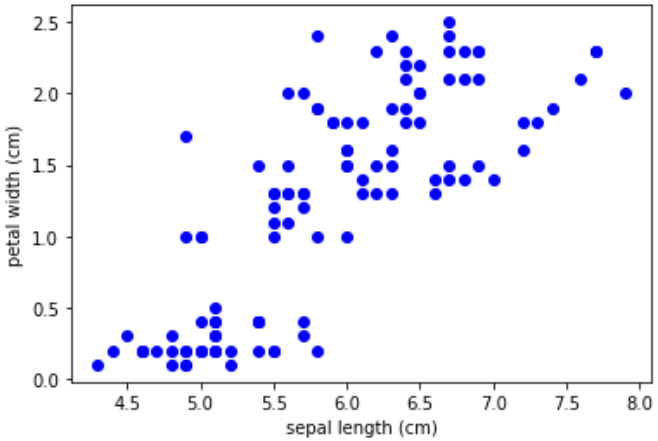
[Source: Bishop, C.]

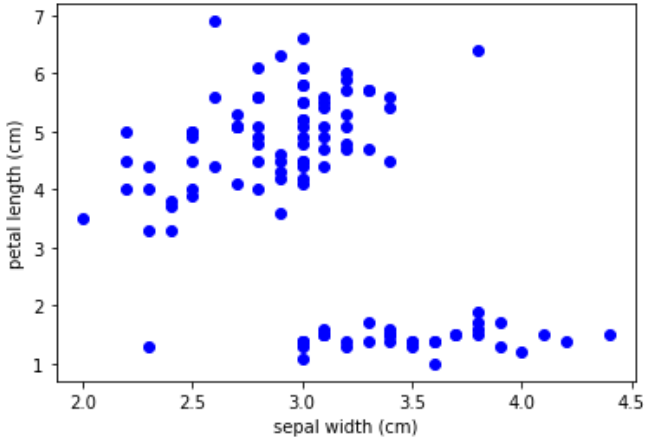
Example : Iris Dataset

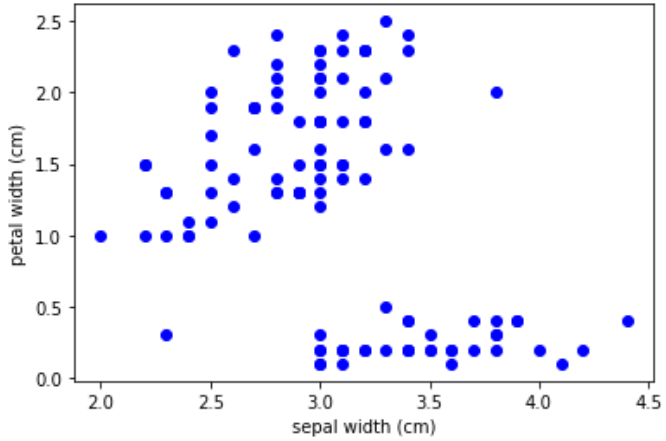
Recall that the Iris dataset has 4 features:
sepal length / width, petal length / width…

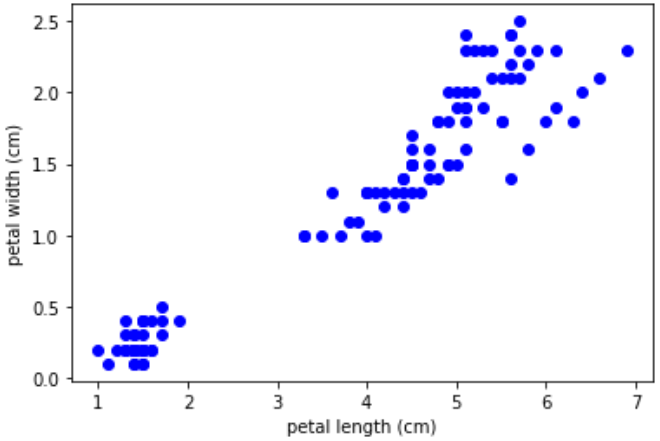




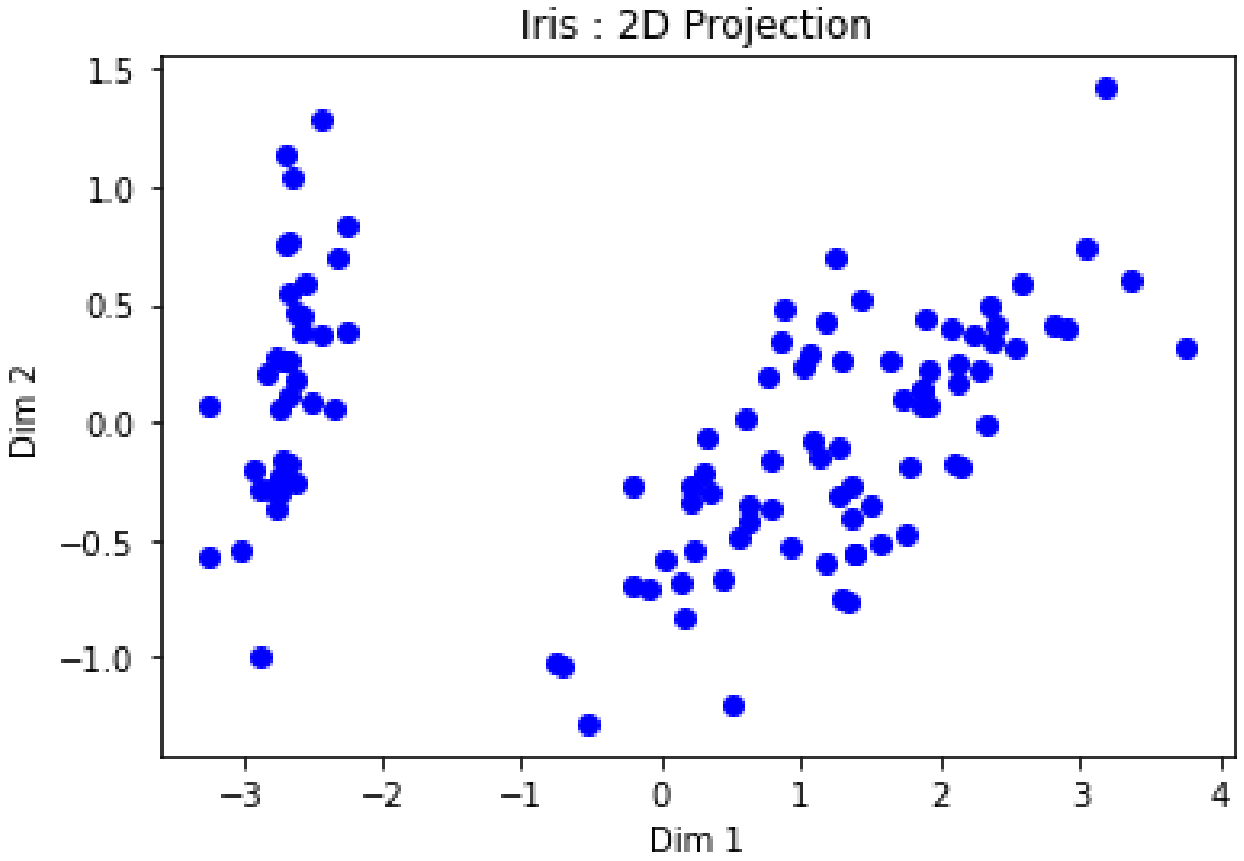








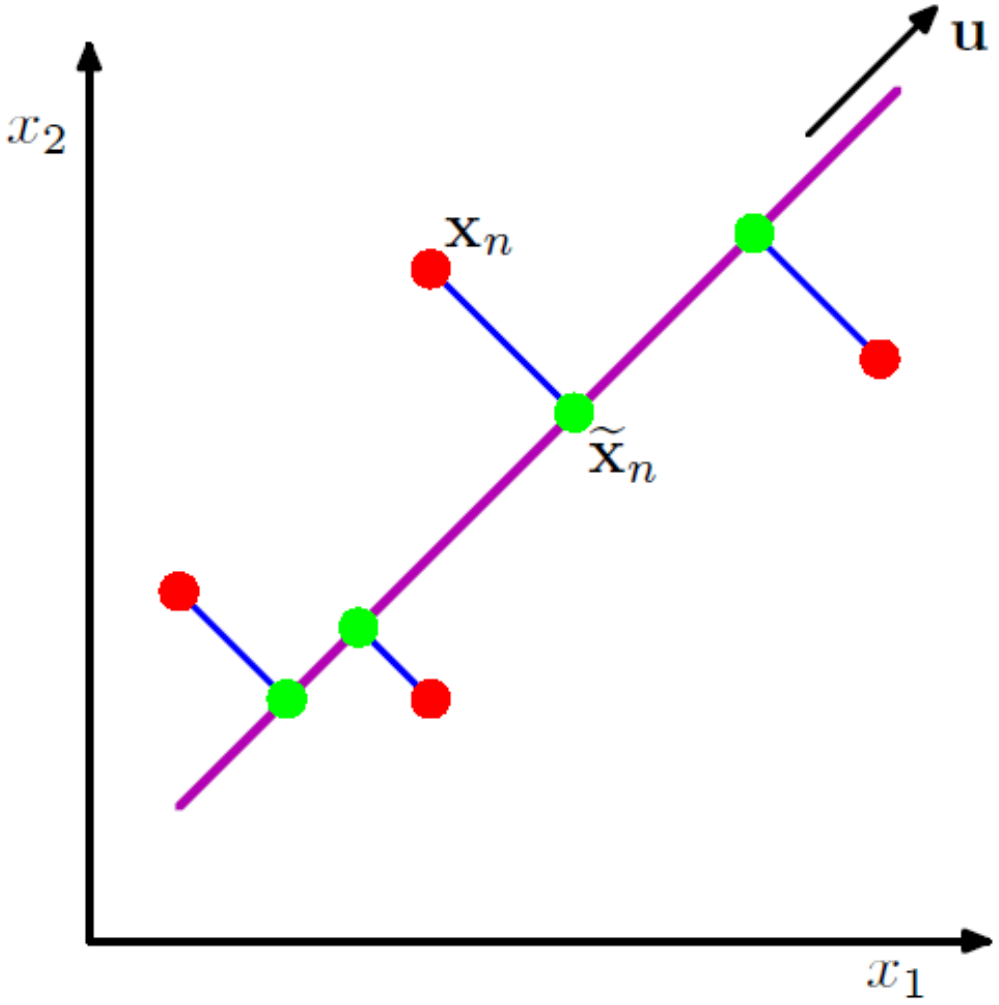
Example : Iris Dataset



Data still cluster in a two-dimensional subspace

We can fit model in 2D to reduce complexity, visualize results, etc.

Linear Dimensionality Reduction



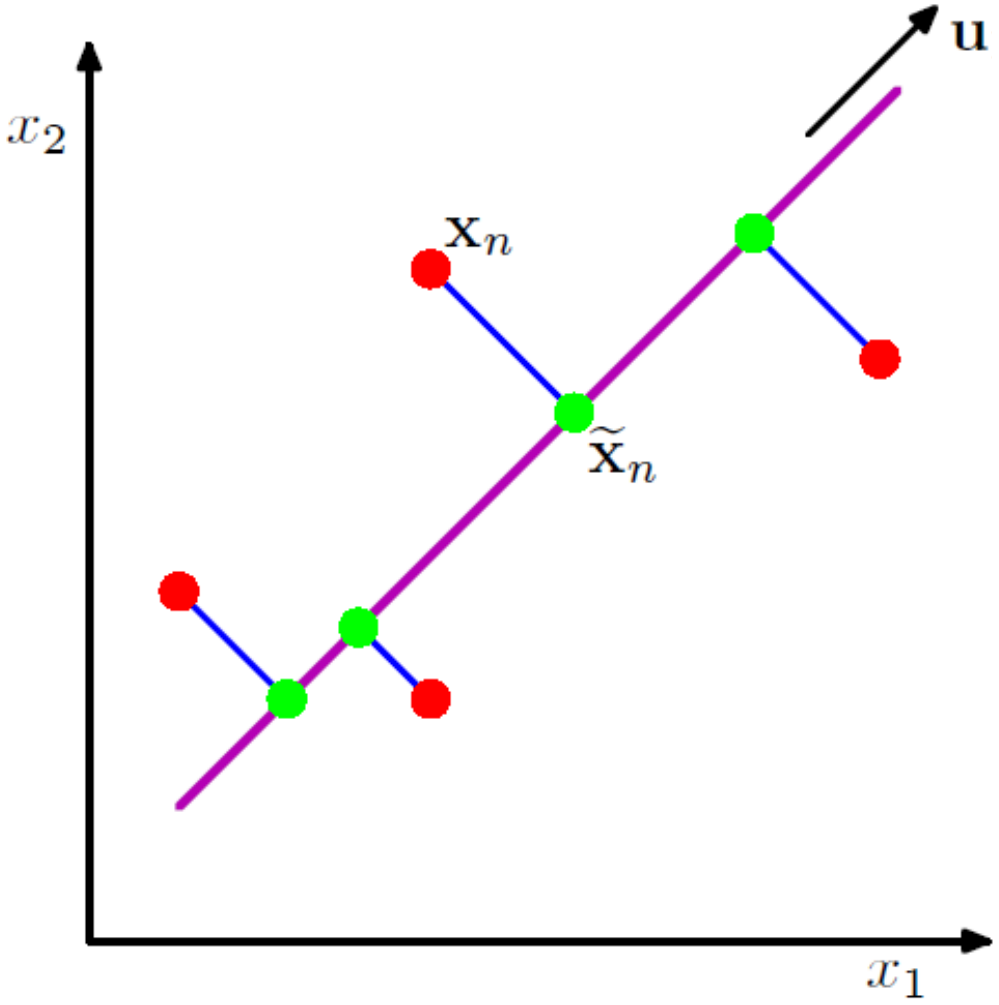
Project data onto a line or plane…

…one of the simplest dimensionality reduction approaches

First, let's review some linear algebra…

[Source: Bishop, C.]

Linear Dimensionality Reduction



Projecting data onto a vector is a simple inner product,

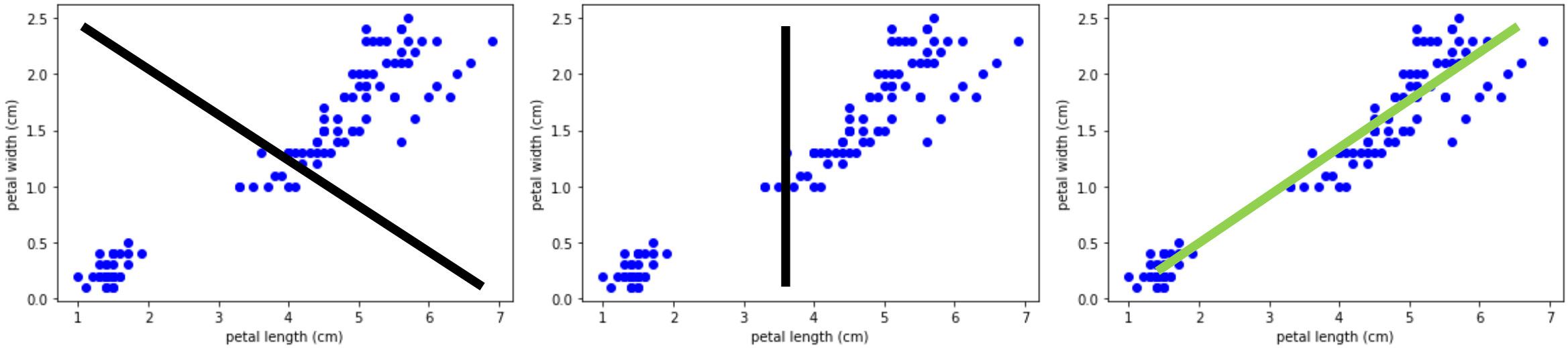
$$\widetilde{x}_n = u^T x_n$$

We call u the *linear subspace*

[Source: Bishop, C.]

Linear Dimensionality Reduction

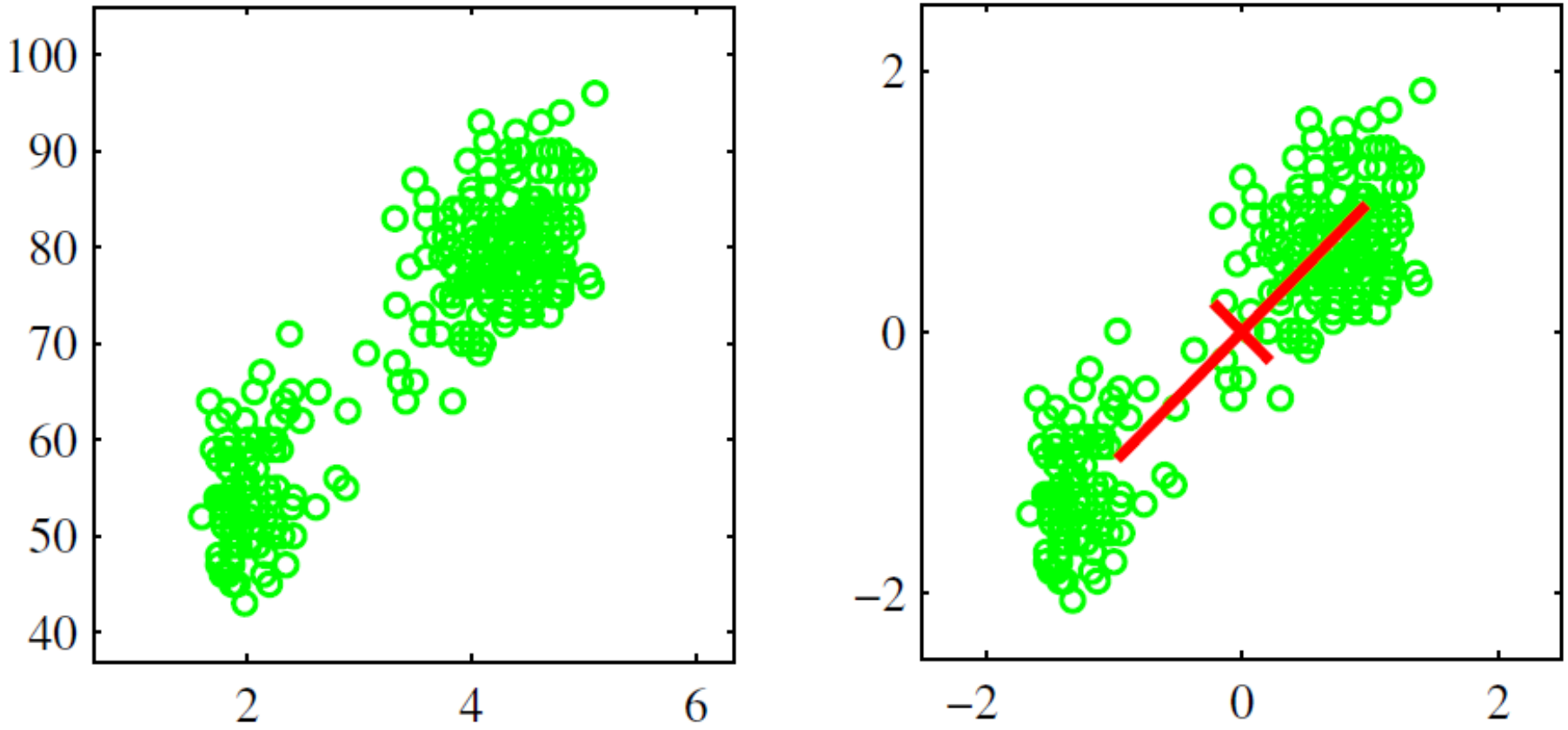
Which choice of subspace is best? And why?



Idea Choose the subspace that captures the most variation in the original data

Principal Component Analysis (PCA)

Identify directions of *maximum variation* as subspaces…



…we call each direction a *principal component*

[Source: Bishop, C.]

Principal Component Analysis (PCA)

First, center the data by subtracting the sample mean,

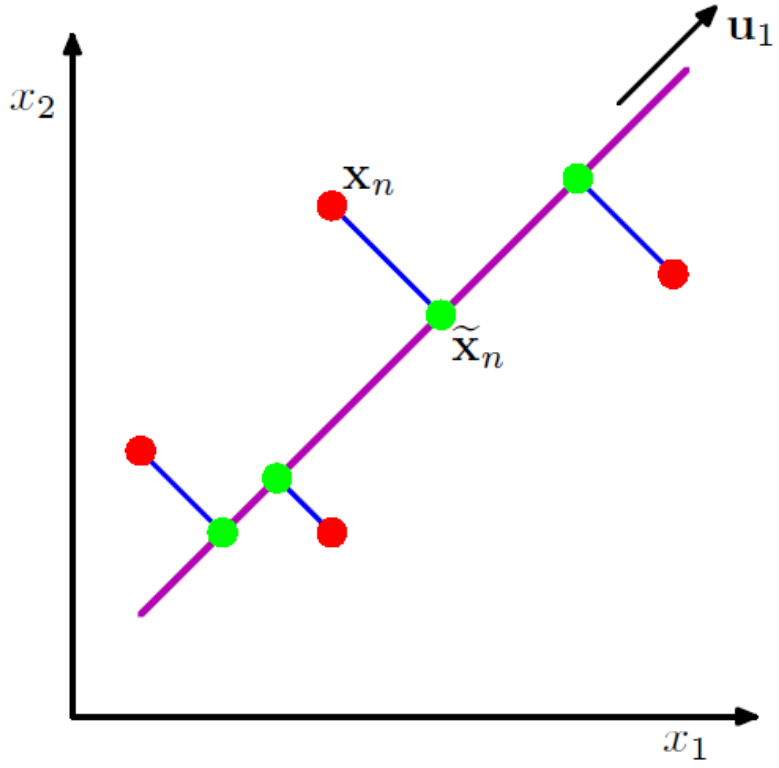
$$\bar{x} = \frac{1}{N}\sum_{n=1}^{N} x_n$$

Variance of projected subspace,

$$\frac{1}{N}\sum_{n=1}^{N}\left(u^T x_n - u^T \bar{x}\right)^2$$

Projection of nth data point

Projection of mean



Maximum Variance Formulation

A little algebra…

$$\frac{1}{N}\sum_{n=1}^{N}\left(u^T x_n - u^T \bar{x}\right)^2 = \frac{1}{N}\sum_{n=1}^{N}\left\{u^T(x_n - \bar{x})\right\}^2$$ Pull out u

Quadratic form

$$= \frac{1}{N}\sum_{n=1}^{N} u^T(x_n - \bar{x})(x_n - \bar{x})^T u$$

Define: $S = \frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{x})(x_n - \bar{x})^T$

Then: $$\frac{1}{N}\sum_{n=1}^{N}\left(u^T x_n - u^T \bar{x}\right)^2 = u^T S u$$

This is what we will optimize over u

Maximum Variance Formulation

Find u so that projected variance is maximal…

$$\max_u \; u^T S u$$

Don't want to *cheat* with large magnitude u, so we add penalty,

$$\max_u \; u^T S u - \lambda u^T u$$

Set the derivative (gradient) to zero and solve…

$$Su - \lambda u = 0$$

$$Su = \lambda u$$

What equation is this?

u is an *eigenvector* with *eigenvalue* λ

Recap of Concepts

- Learning a reduced *intrinsic dimension* is useful for a bunch of reasons
- The easiest approach is to find a *linear subspace*
- PCA defines the linear subspace as that which maximizes variance of the projected data
- The set of subspaces are defined by the *eigenvectors,*

$$\max_{u} u^T S u - \lambda u^T u$$

$$Su = \lambda u$$

Eigenstuff

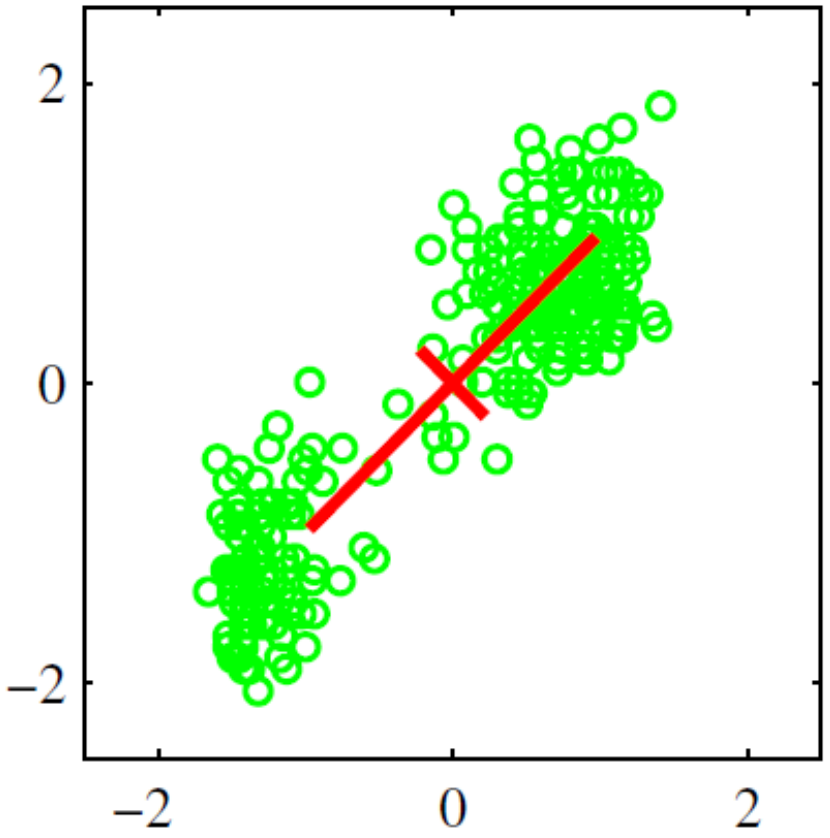
Eigenvectors / values of a matrix solve the equation

$$Su = \lambda u$$

- Matrix S may have *multiple* eigenvectors / values that solve the above equation
- For D-dimensional u can find all vectors in $O(D^3)$ time
- PCA finds M<D vectors with largest eigenvalues
- Can find M<D sorted eigenvectors in $O(MD^2)$ time
- Note that D can be large!

Principal Component Analysis (PCA)

How much variance is captured by just the first principal component (i.e. eigenvector with largest eigenvalue)?



Let u_1 be the first principal component, then variance of first PC is,

$$\frac{1}{N}\sum_n \left\{u_1^T(x_n - \bar{x})\right\}^2$$

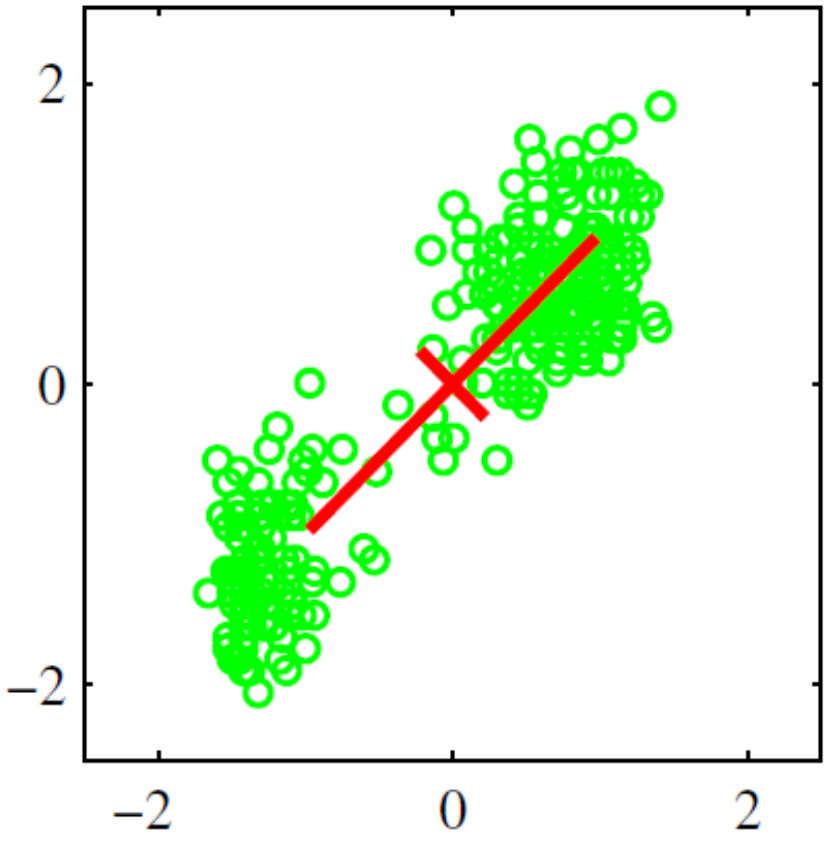
How much in the second PC?

$$\frac{1}{N}\sum_n \left\{u_2^T(x_n - \bar{x})\right\}^2$$

[Source: Bishop, C.]

Explained Variance

How much variance is captured in M < D principal components?



$$\frac{1}{N}\sum_{m=1}^{M}\sum_{n}\left\{u_m^T(x_n-\bar{x})\right\}^2$$

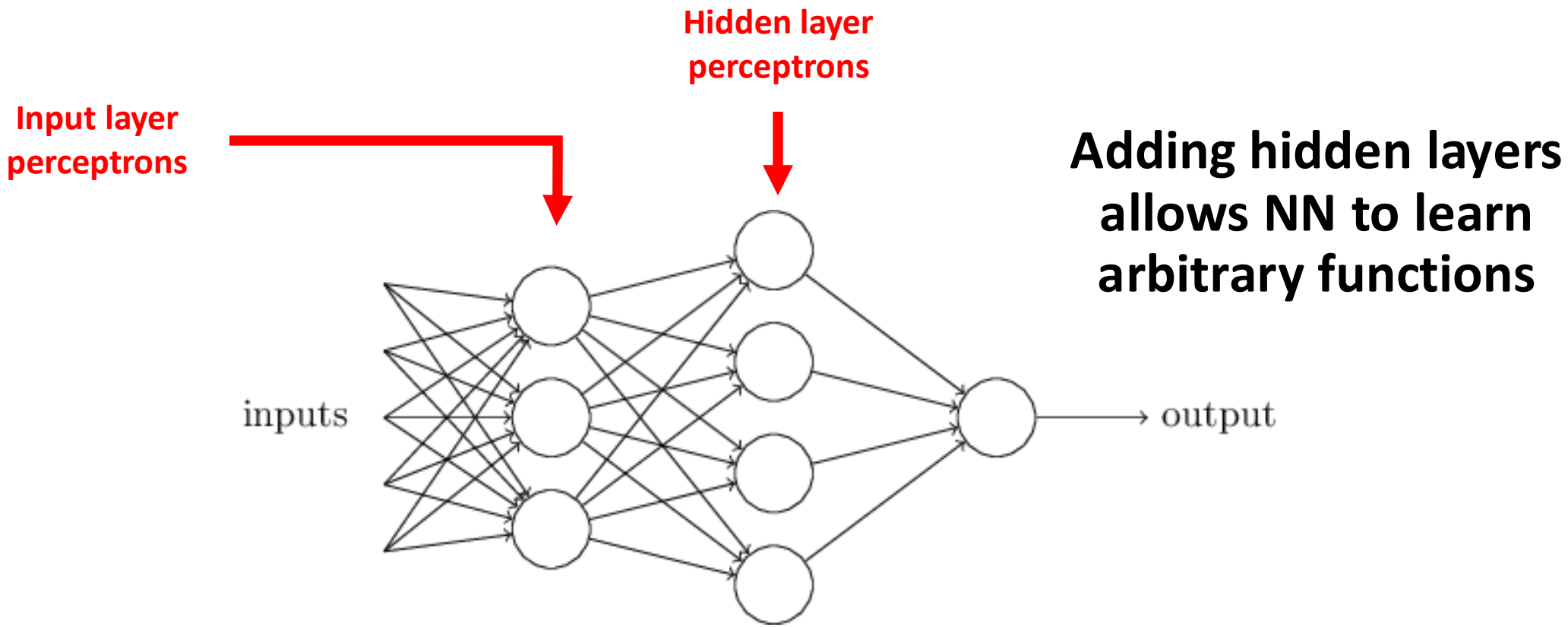
We call this the *explained variance* of the first M principal components

Divide by total variance to find percentage of the total variance explained by the subspace

[Source: Bishop, C.]

Neural Networks

Multilayer Perceptron



Adding hidden layers allows NN to learn arbitrary functions

This is the quintessential *Neural Network*…

…also called *Feed Forward Neural Net* or *Artificial Neural Net*

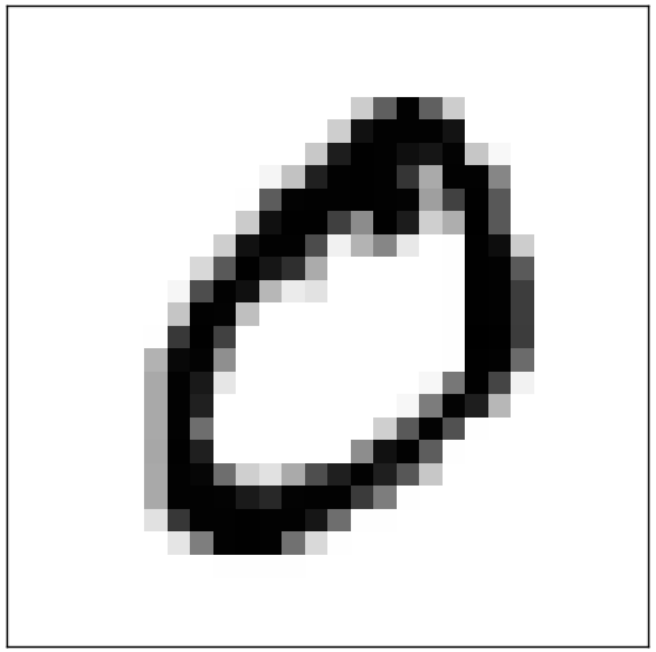
[Source: http://neuralnetworksanddeeplearning.com]

Handwritten Digit Classification

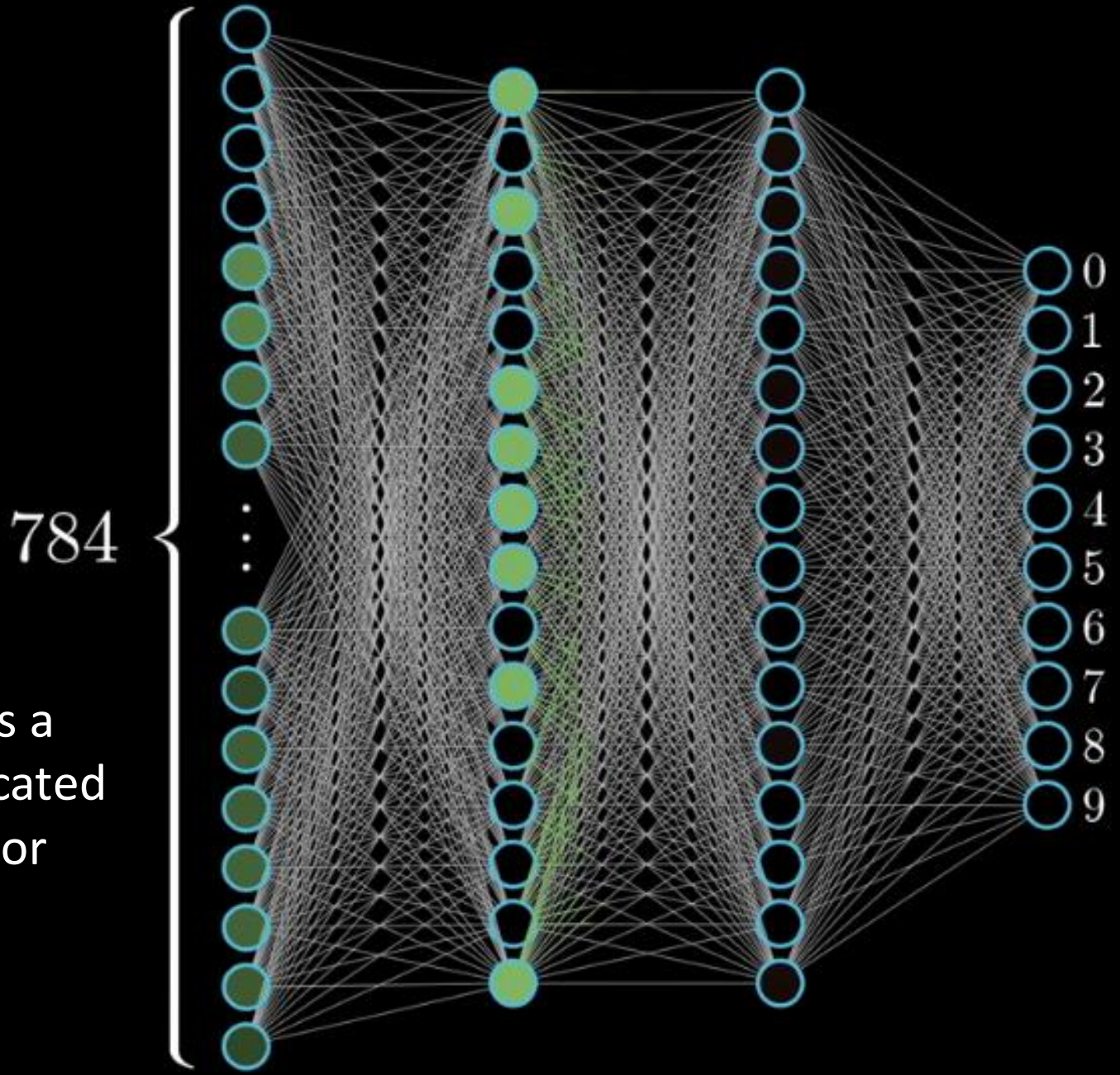
Classifying handwritten digits is the “Hello World” of NNs

Modified National Institute of Standards and Technology (MNIST) database contains 60k training and 10k test images

Each character is centered in a 28x28=784 pixel grayscale image

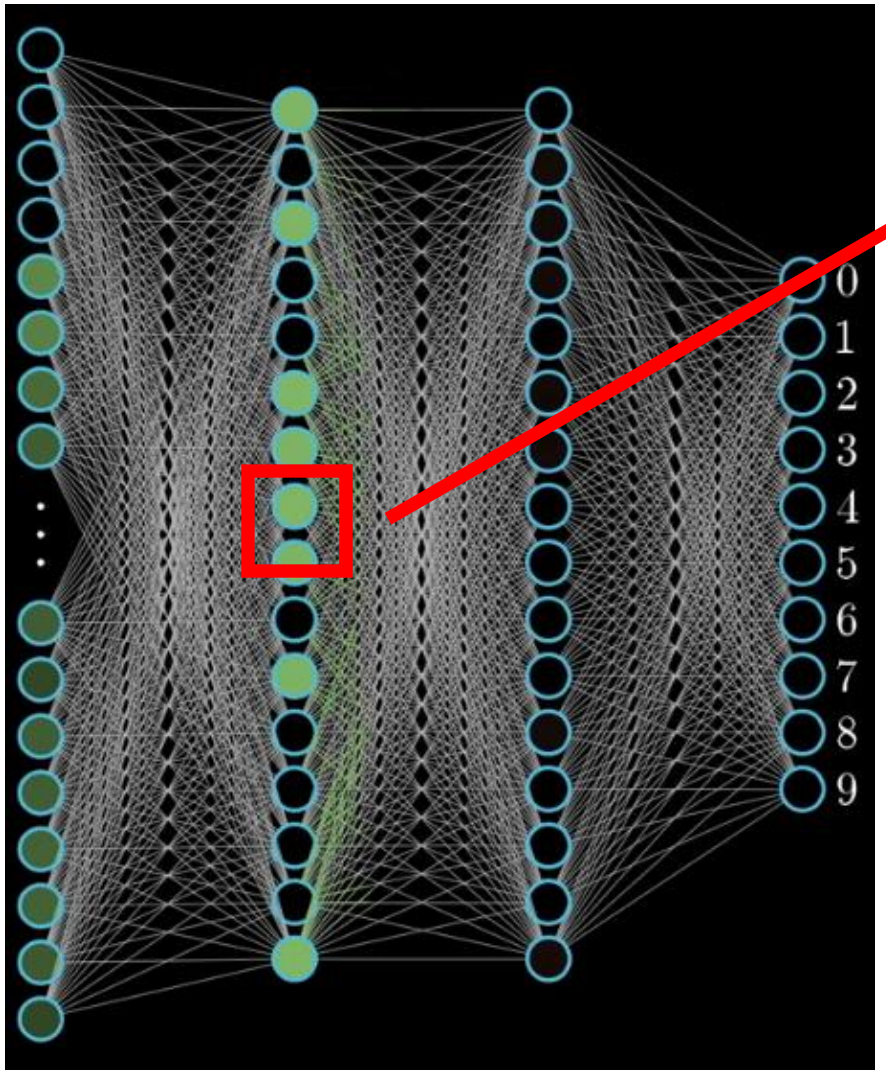


[Source : 3Blue1Brown : https://www.youtube.com/watch?v=aircAruvnKk]



Each image pixel is a number in [0,1] indicated by highlighted color

Feedforward Procedure



Each node computes a *weighted combination* of nodes at the previous layer…

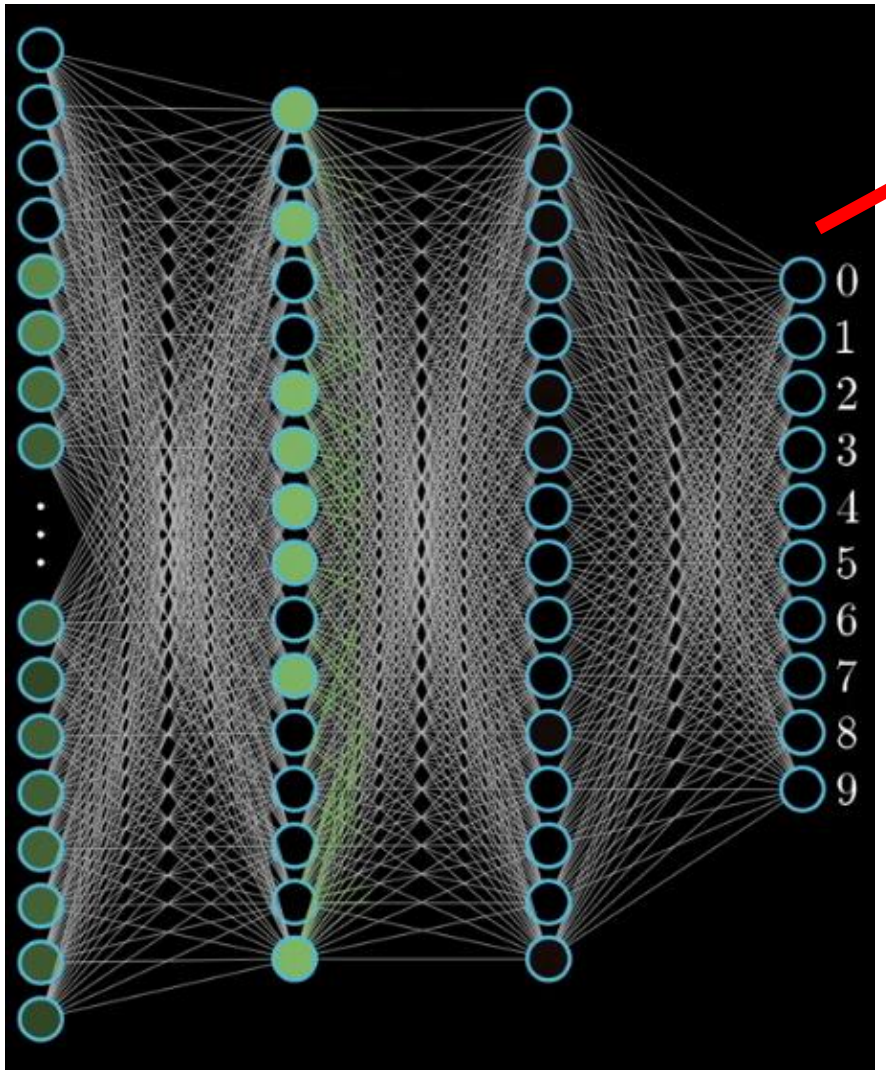
$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

Then applies a *nonlinear function* to the result

$$\sigma(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b)$$

Often, we also introduce a constant *bias* parameter

Multilayer Perceptron



Final layer is typically a linear model…for classification this is a Logistic Regression

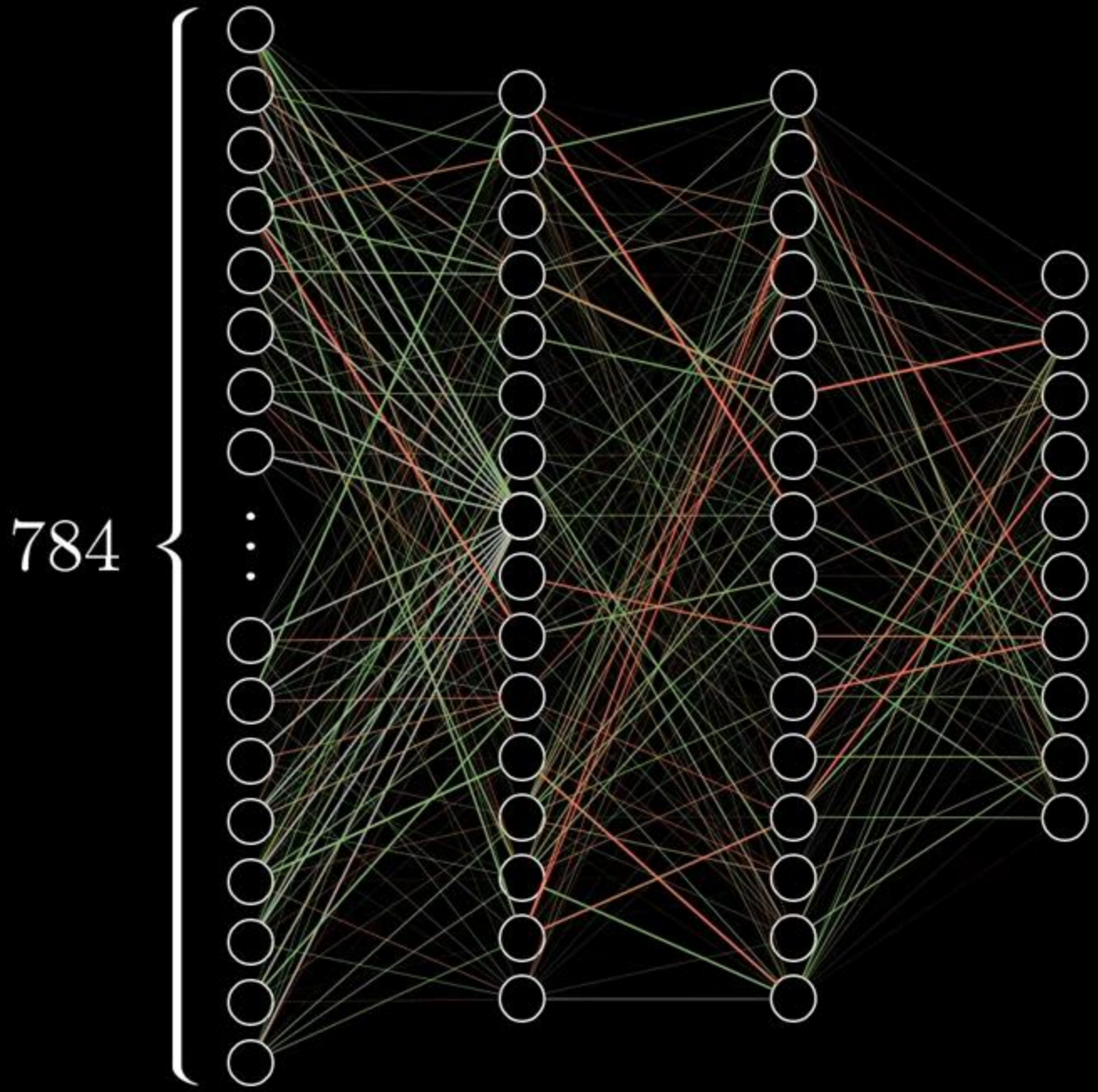
$$\sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$$

Vector of activations from previous layer

Recall that for multiclass logistic regression with K classes,

$$p(\text{Class} = k \mid x) \propto \sigma(w_k^T x + b_k)$$

[Source : 3Blue1Brown : https://www.youtube.com/watch?v=aircAruvnKk]



$$784 \times 16 + 16 \times 16 + 16 \times 10$$
weights

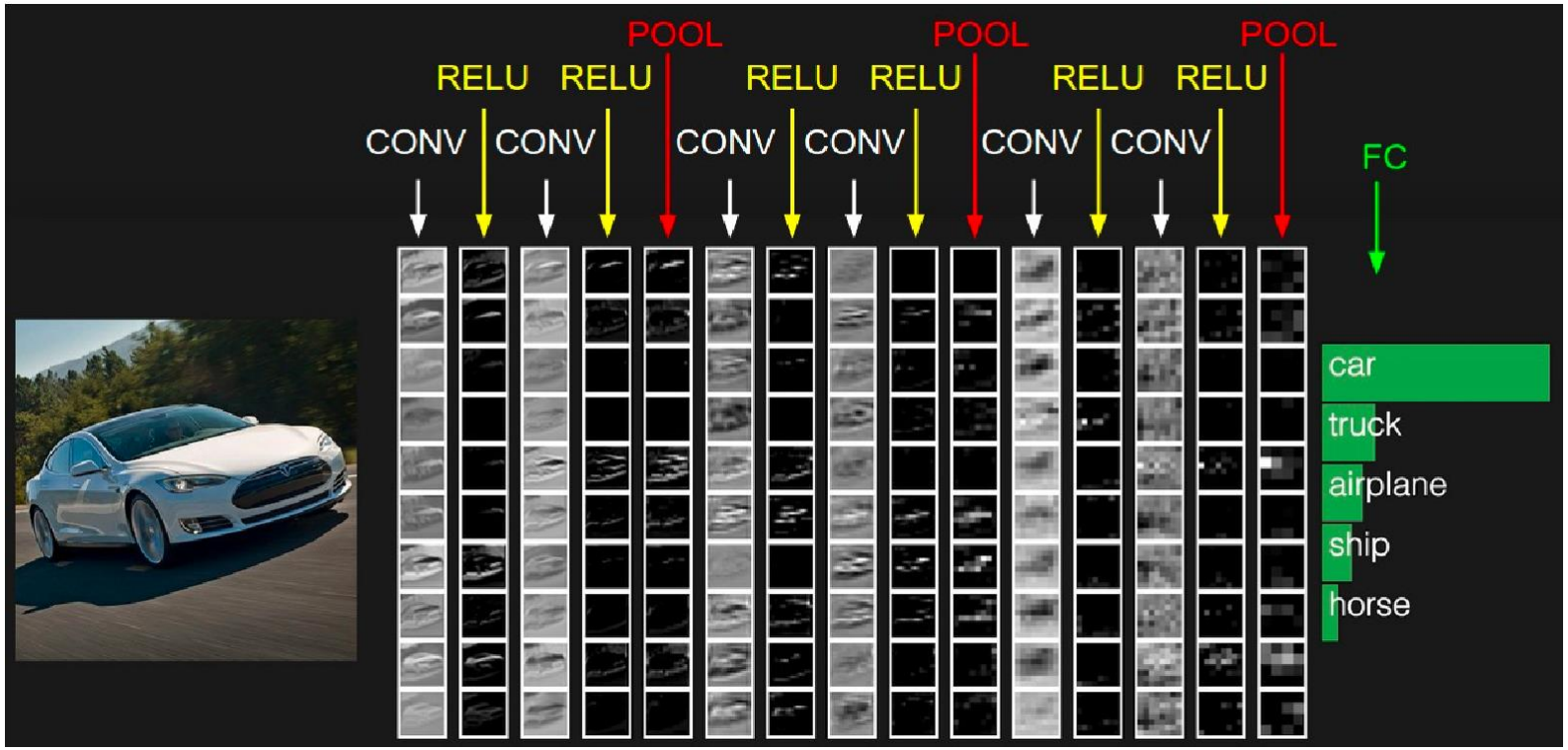
$$16 + 16 + 10$$
biases

$$13{,}002$$

Each parameter has some impact on the output…need to tweak (learn) all parameters simultaneously to improve prediction accuracy

Convolutional neural networks (CNN)

- A.K.A. ConvNet architecture
- A set of neural network architecture that consists of
  - convolutional layers
  - pooling layers
  - fully-connected (FC) layers

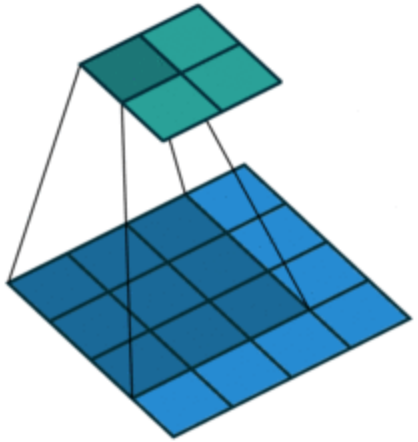


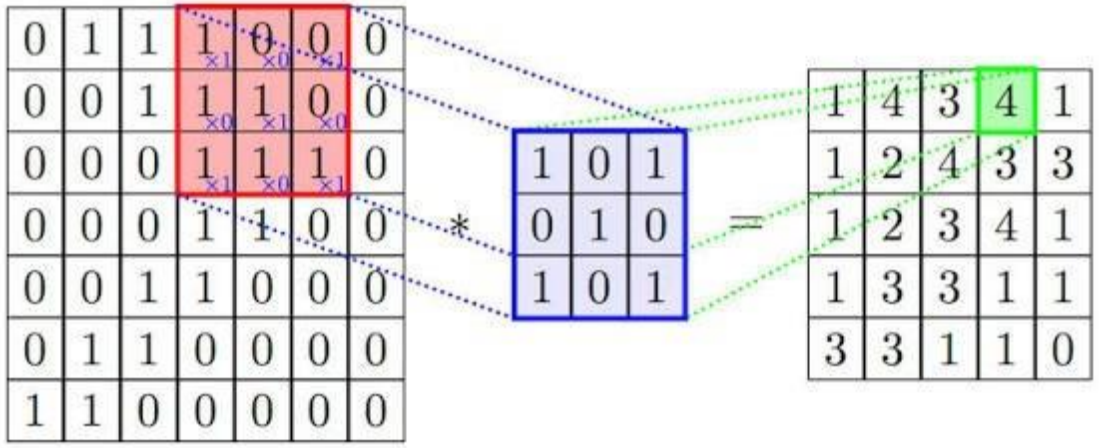
(Stanford CS231n)

Convolution for single-channel images

Consider one filter with weights $\{w_{i,j}\}$ with size F x F

- For every F x F region of the image, perform inner product (= element wise product, then sum them all)
- Q: given a w x h image, after convolution with a F x F filter, what is the size of the resulting image?
- Terminologies: filter size, receptive field size, kernel.



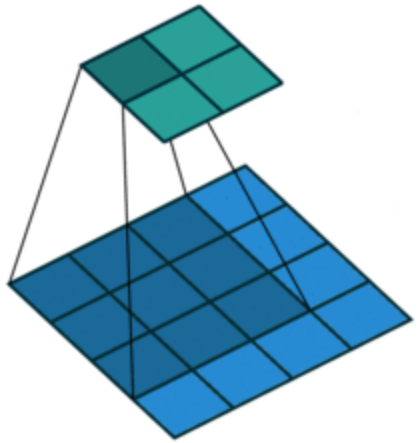


Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

Convolution: Some Intuition

Define the convolution of filter f on image I as:

$$(I * f)(x) = \sum_m \sum_n f(x - m, y - n) I(m, n)$$



Many ML libraries *actually* implement cross-correlation:

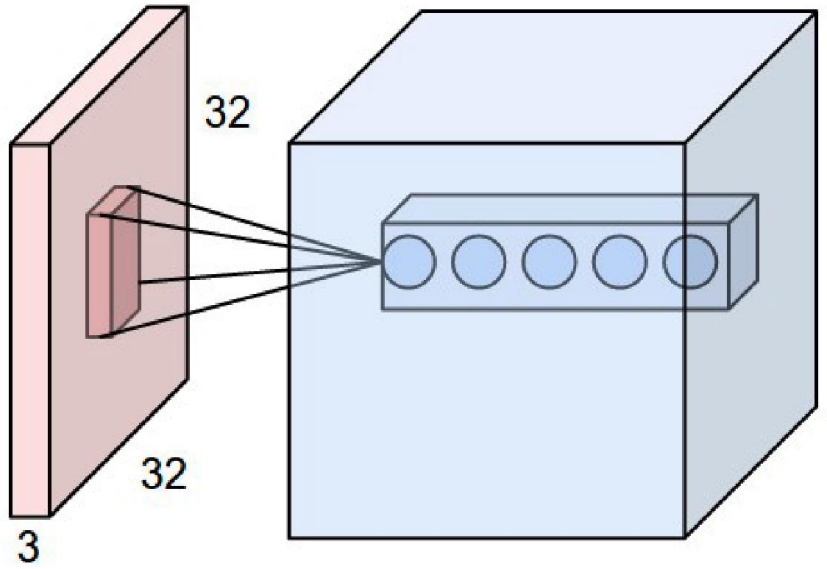
$$(f * I)(x) = \sum_m \sum_n f(x, y) I(x + m, y + n)$$

Learning finds good values for the convolution filter…

Convolutional layer for multi-channel images

Input: w (width) x h (height) x c (#channels)

- E.g. 32 x 32 x 3
- 3 channels: R, G, and B



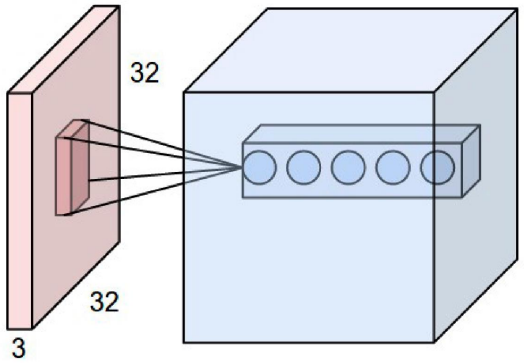
A convolutional filter on such image is of shape F x F x c

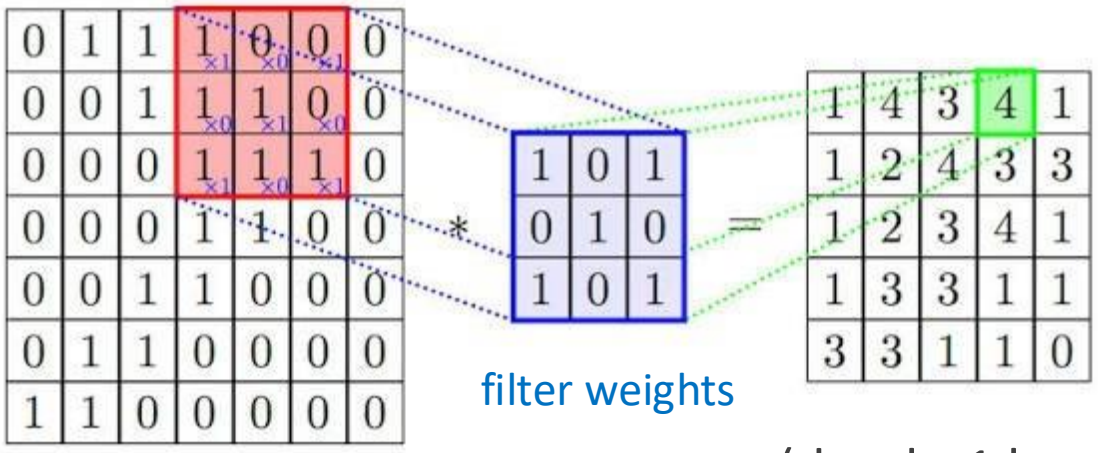
- Only spatial structure in the first two dimensions
- Denoted by $\{w_{i,j,k}\}$

image from Stanford CS231n

Convolutional layer: visual explanation

- Consider one filter with weights $\{w_{i,j,k}\}$ with 5 x 5 x 3
  - Imagine a sliding 3D window.
  - **Convolution:** For every 5 x 5 region of the image, perform inner product (= element wise product, then sum them all)
  - Then apply the activation function (e.g., ReLU)
- Results in 28 x 28 x 1 – called **activation map**.
- Now, we can do K of these filters but with different weights $\{w_{i,j,k}^{(\ell)}\}$ for $\ell \in [K]$ => output is 28 x 28 x K





(depth=1 here)

(image from https://www.quora.com/Why-do-we-use-convolutional-layers)

Convolutional Layer: More Details

Stride length S

- Skip input regions; Move the sliding window of a filter not by 1 but by S.
- E.g., S=2 means skipping every other 5 by 5 region.

Zero-padding P: add P number of artificial pixels with value 0 around the input image on both sides

- To ensure the spatial dimension is maintained (otherwise, patterns at the corners are not detected well)
- If we use P=1, then the activation map will be 30 x 30, not 28 x 28 in our example!

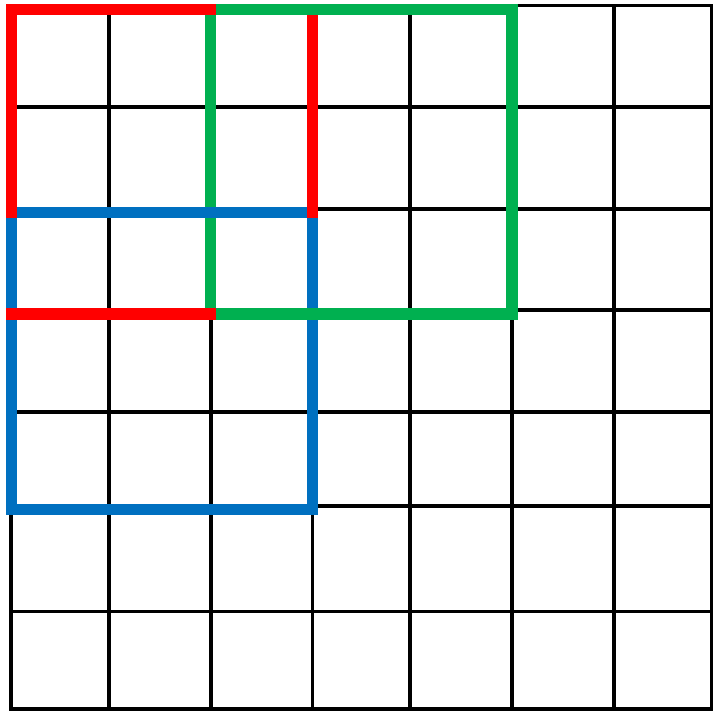


image from https://medium.com/@ayeshmanthaperera/what-is-padding-in-cnns-71b21fb0dd7

Example

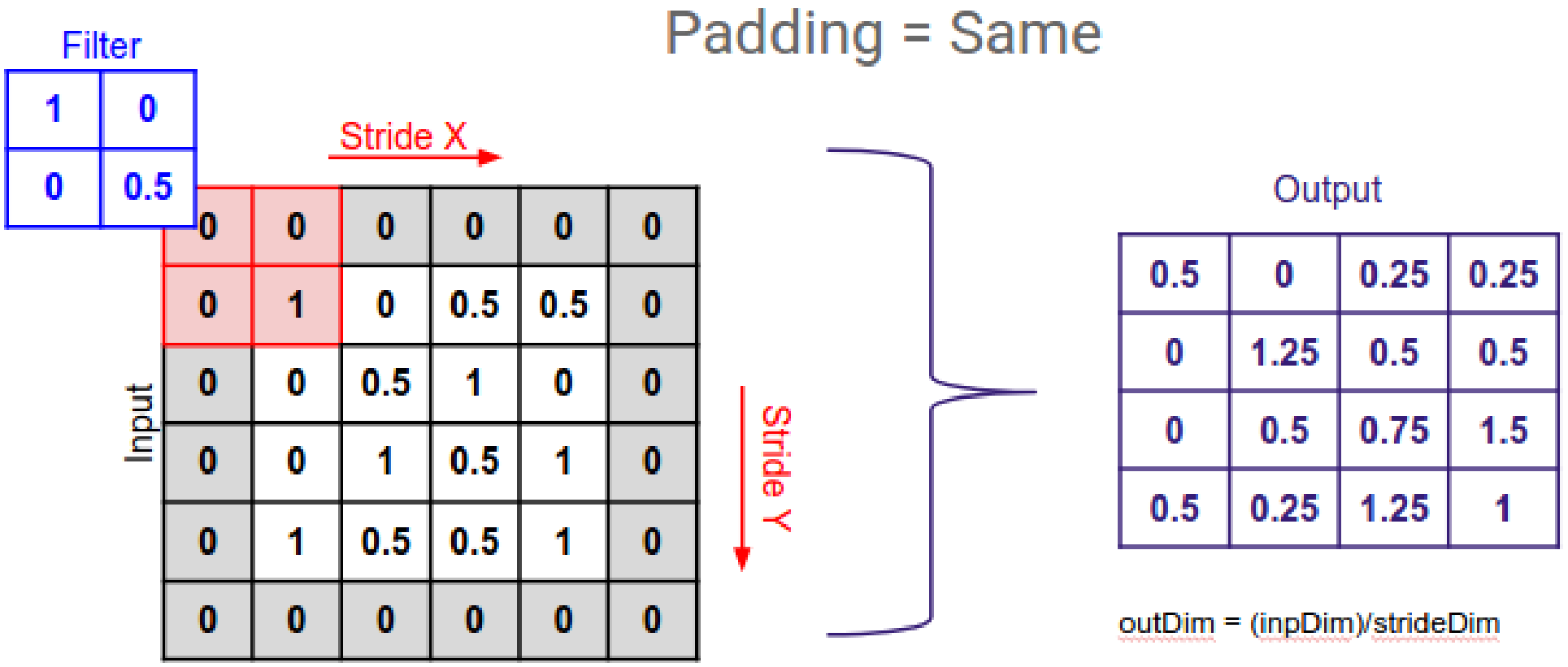
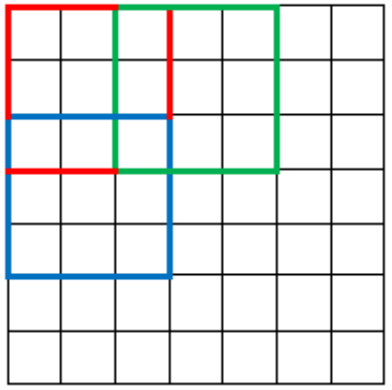


image from https://medium.com/@ayeshmanthaperera/what-is-padding-in-cnns-71b21fb0dd7

Convolutional Layer: More Details

Stride length S



- Skip input regions; Move the sliding window of a filter not by 1 but by S.
- E.g., S=2 means skipping every other 5 by 5 region.

Zero-padding P: add P number of artificial pixels with value 0 arour image.

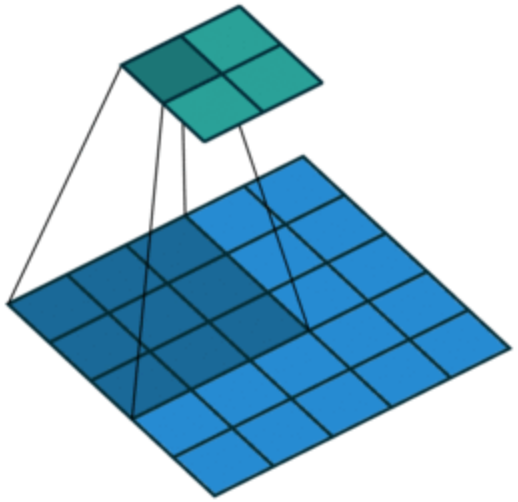
- To ensure the spatial dimension is maintained (otherwise, patterns at the corners are not detected well)
- If we use P=2, then the activation map will be 32 by 32 not 28 by 28 in our example!

Rules (same goes for height)

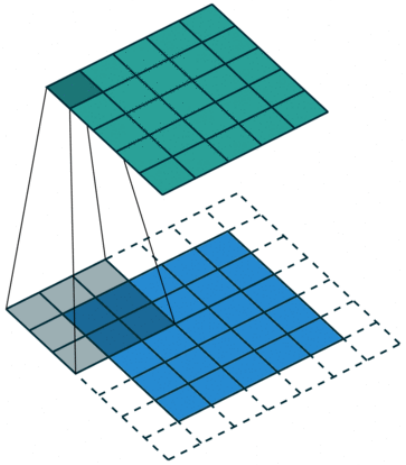
- W: input volume width, F: filter width (usually, the filter has the same width and height)
- The output width K = floor((W – F + 2P)/S) + 1
- E.g., W=32, F=5, P=0, S=1 => K = 28
- E.g., W=32, F=5, P=2, S=1 => K = 32

Strides and padding: animations

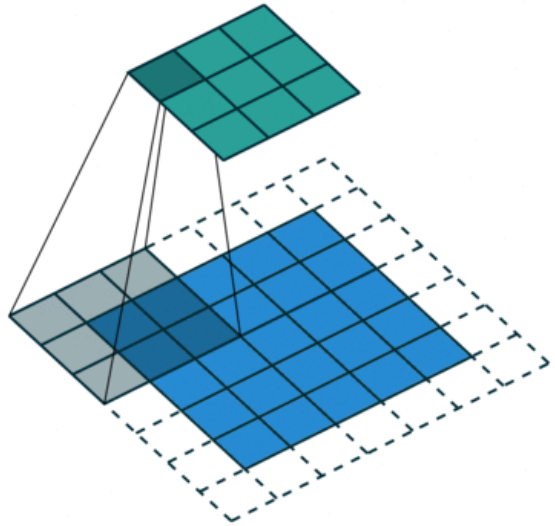
Strides only



Padding only

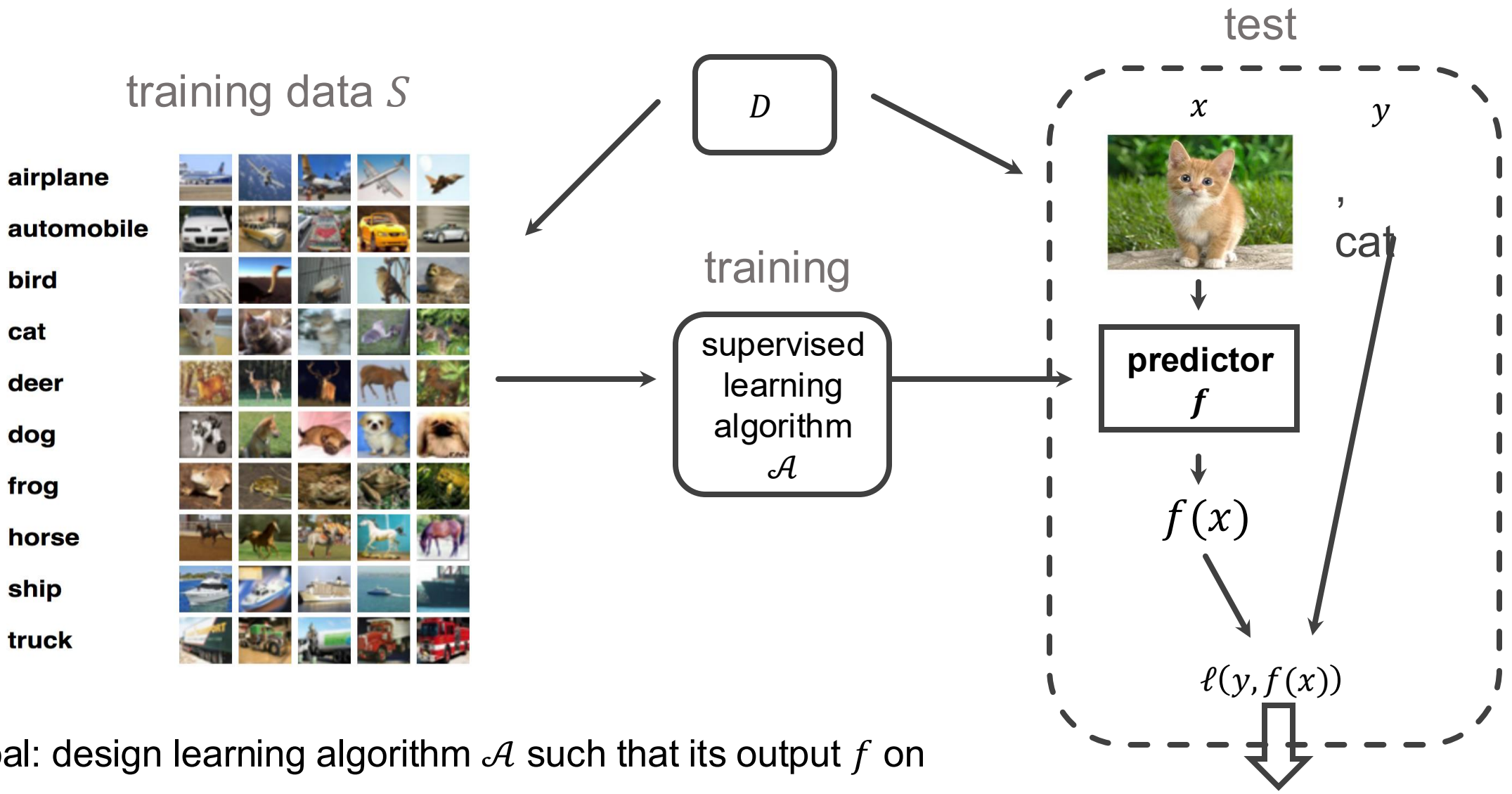


Strides + Padding



Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

Supervised learning setup: putting it together



- Goal: design learning algorithm $\mathcal{A}$ such that its output f on iid training data S has low generalization error

Generalization error: $L_D(f) = \mathrm{E}_{(x,y)\sim D}\, \ell(y, f(x))$

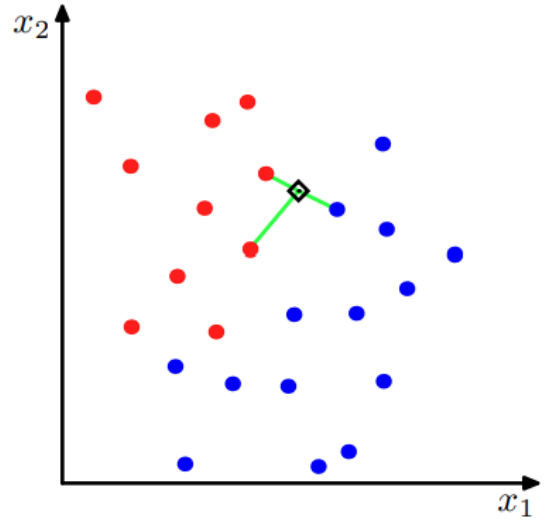
k-nearest neighbors (k-NN): main concept

Training set: $S = \{ (x_1, y_1), \ldots, (x_m, y_m)\}$

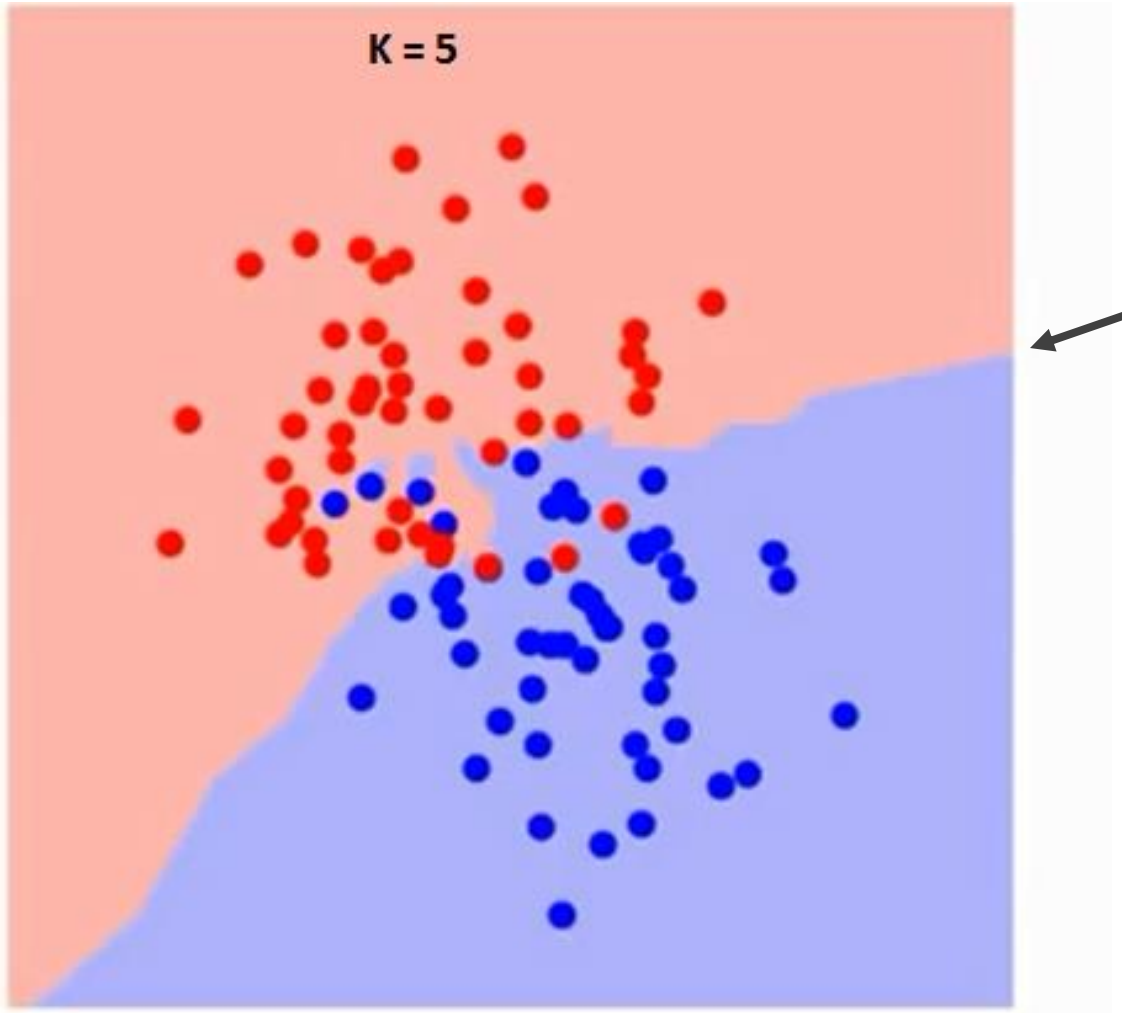
Inductive bias: given test example x, its label should resemble the labels of **nearby points**

Function

- input: x
- find the k nearest points to x from S; call their indices $N(x)$
- output: the majority vote of $\{y_i : i \in N(x)\}$
  - For regression, the average.



k-NN classification example



decision boundary

k-NN classification: pseudocode

- Training is trivial: store the training set
- Test:

Algorithm 3 KNN-PREDICT($\mathbf{D}$, K, $\hat{x}$)

```
1:  S ← [ ]                                                  (list)
2:  for n = 1 to N do
3:      S ← S ⊕ ⟨d(x_n, x̂), n⟩        // store distance to training example n   (append to list)
4:  end for
5:  S ← SORT(S)                        // put lowest-distance objects first     (sort in first coordinate)
6:  ŷ ← 0
7:  for k = 1 to K do
8:      ⟨dist,n⟩ ← S_k                 // n this is the kth closest data point
9:      ŷ ← ŷ + y_n                    // vote according to the label for the nth training point
10: end for
11: return SIGN(ŷ)                     // return +1 if ŷ > 0 and −1 if ŷ < 0    (Majority vote of {y_i : i ∈ N(x)})
```

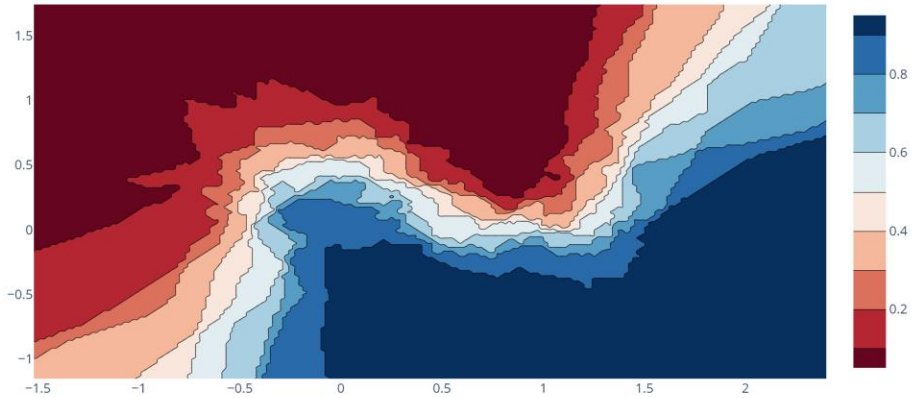
- Time complexity (assuming distance calculation takes $O(d)$ time)
  - $O(m\,d\ + m\log m\ + k\ ) = O\big(m(d\ + \log m)\big)$
- Faster nearest neighbor search: k-d trees, locality sensitive hashing

Variations

- Classification
  - Recall the majority vote rule: $\hat{y} = \arg\max_{y \in \{1,\ldots,C\}} \sum_{i \in N(x)} 1\{y_i = y\}$
  - Soft weighting nearest neighbors: $\hat{y} = \arg\max_{y \in \{1,\ldots,C\}} \sum_{i=1}^{m} w_i \, 1\{y_i = y\}$,

    where $w_i \propto \exp(-\beta \, d(x, x_i))$, or $\propto \frac{1}{1+d(x,x_i)^{\beta}}$

- Class probability estimates
  - $\hat{P}(Y = y \mid x) = \frac{1}{k} \sum_{i \in N(x)} 1\{y_i = y\}$
  - Useful for “classification with rejection”



Inductive Bias

Training

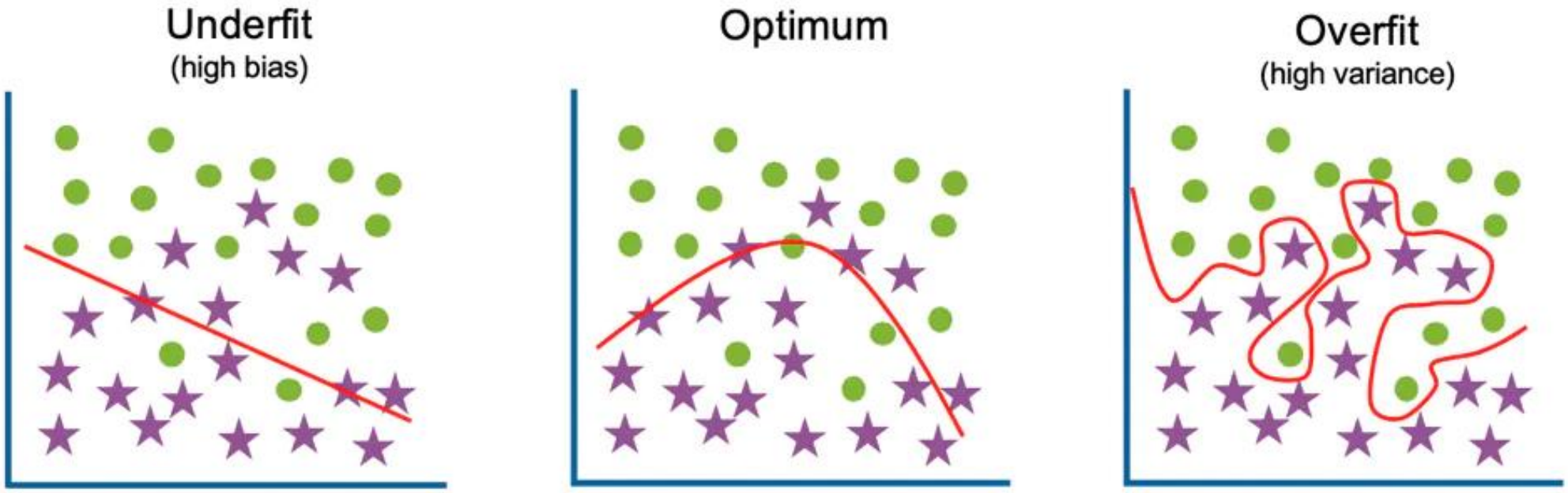
class A

class B

Test

How would you label the test examples?

Overfitting vs Underfitting



Source: ibm.com

Bayes optimal classifier

Theorem f_{BO} achieves the smallest 0-1 error among all classifiers.

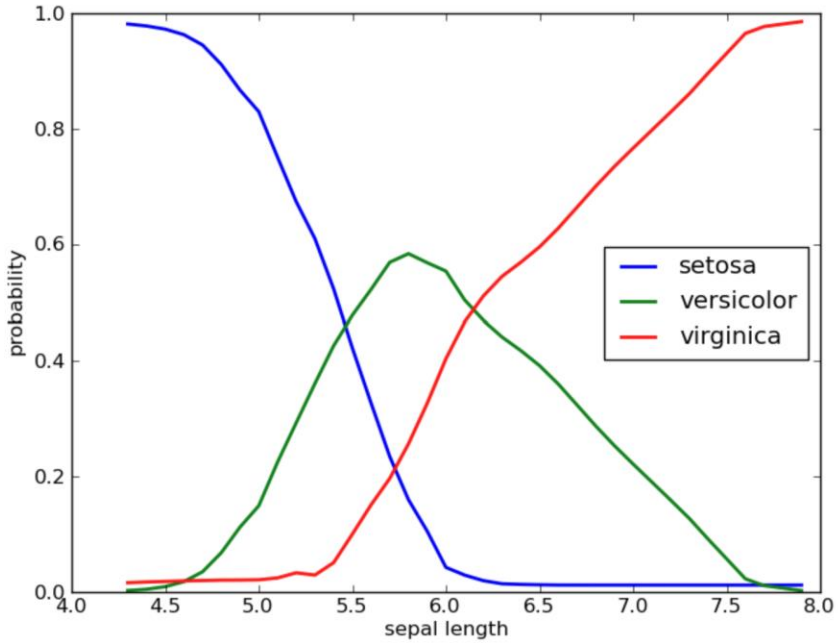
$$f_{BO}(x) = \arg\max_{y \in \mathcal{Y}} P_D(X = x, Y = y) = \arg\max_{y \in \mathcal{Y}} P_D(Y = y \,|X = x), \forall x \in \mathcal{X}$$

Example Iris dataset classification:

Iris Setosa

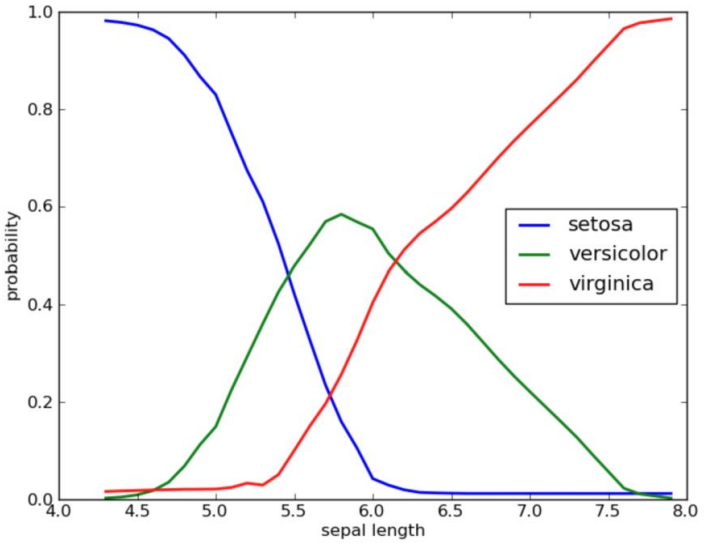
Iris Versicolor

Iris Virginica

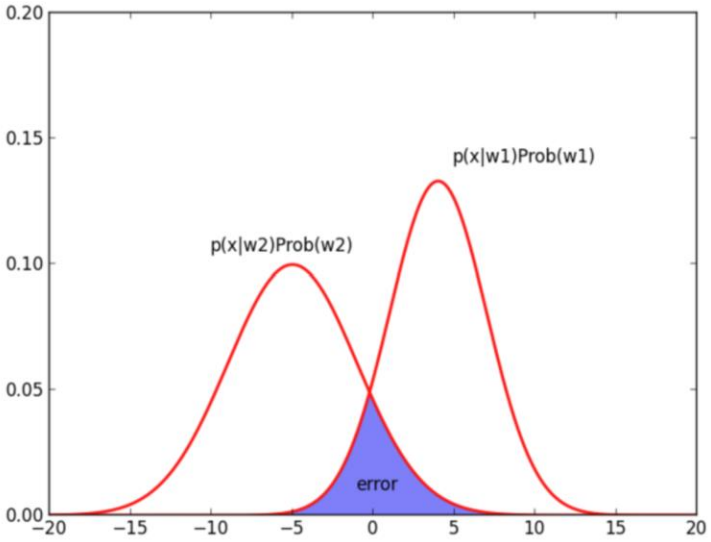


Bayes error rate: alternative form

$$
\begin{aligned}
L_D(f_{BO}) &= P_D\big(Y \neq f_{BO}(X)\big) \\
&= \textstyle\sum_x P_D(Y \neq f_{BO}(x) \mid X = x)\, P_D(X = x) \\
&= \textstyle\sum_x (1 - P_D(Y = f_{BO}(x) \mid X = x))\, P_D(X = x) \\
&= \textstyle\sum_x \left(1 - \max_y P_D(Y = y \mid X = x)\right) P_D(X = x) \\
&= \mathrm{E}\left[1 - \max_y P_D(Y = y \mid X)\right]
\end{aligned}
$$

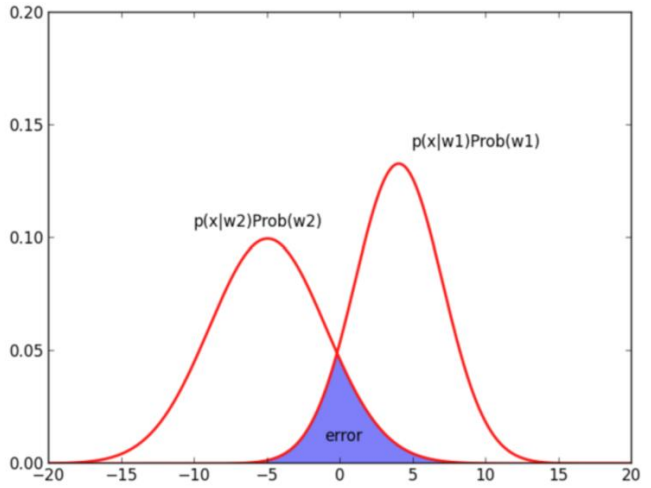


- Special case: binary classification
  - $L_D(f_{BO}) = \sum_x P_D(Y \neq f_{BO}(x), X = x)$
    $= \sum_x \min(\, P_D(Y = +1, X = x)\,, P_D(Y = -1, X = x))$

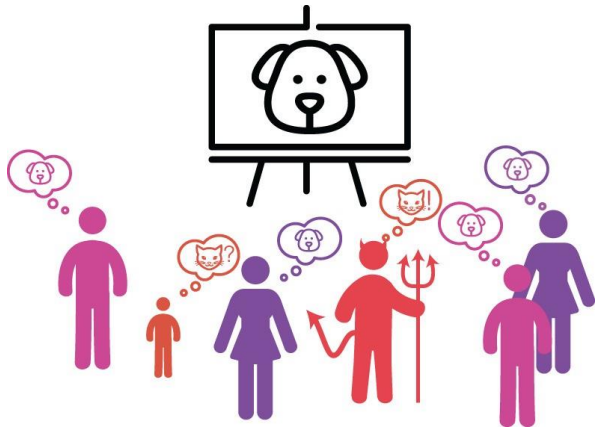


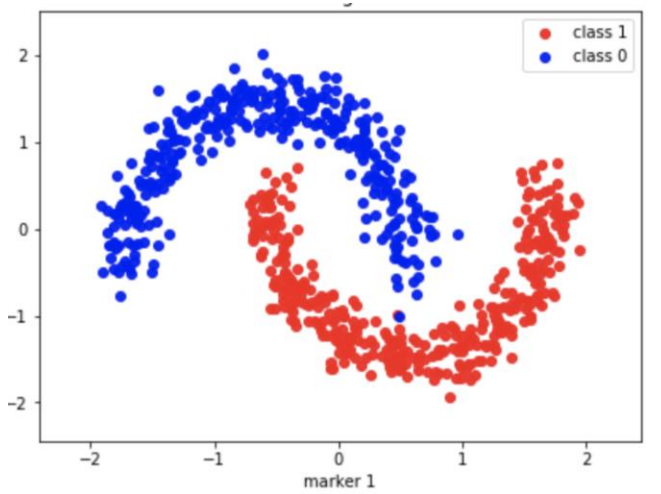
When is the Bayes error rate nonzero?

$$L_D(f_{BO}) = \sum_x \min(\, P_D(Y = +1, X = x)\,, P_D(Y = -1, X = x))$$



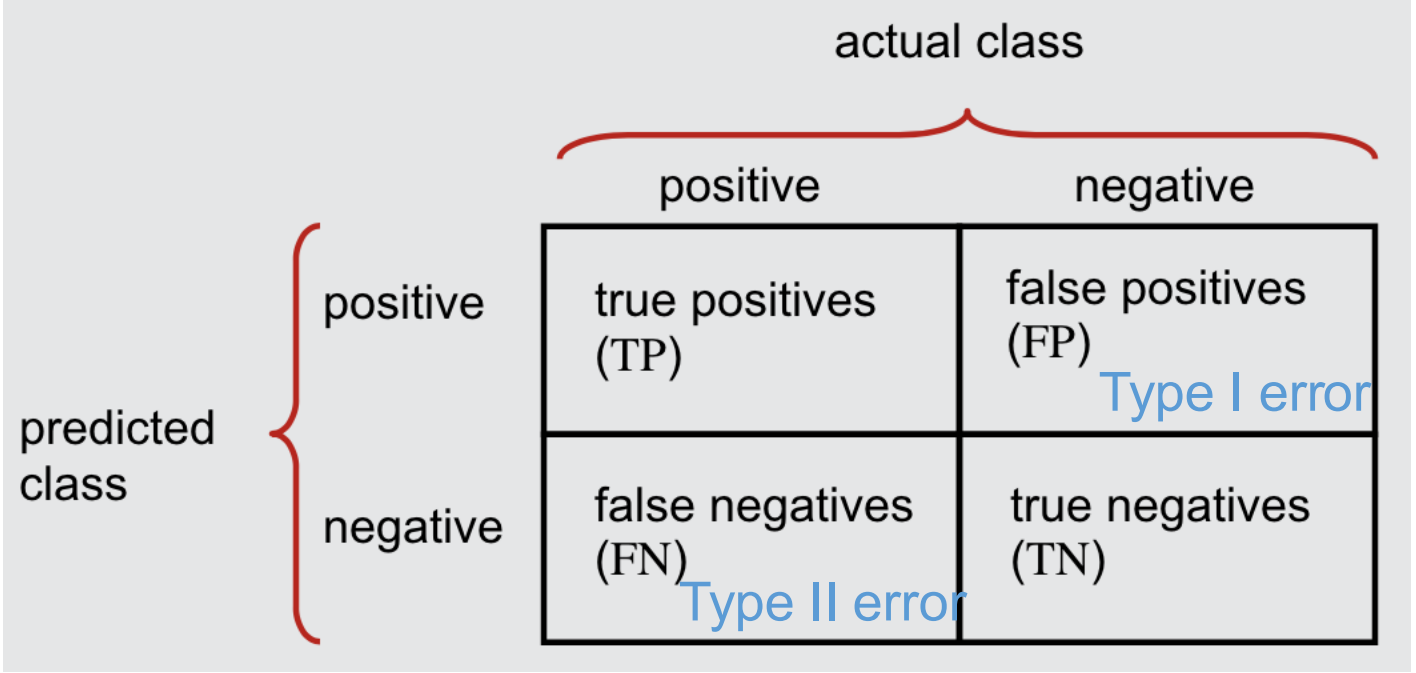
- Limited feature representation
- Noise in the training data
  - Feature noise
  - Label noise
  - Sensor failure
  - Typo in reviews for sentiment classification
- May not be a single “correct” answer
- Inductive bias of the model / learning algorithm





New measures of classification performance

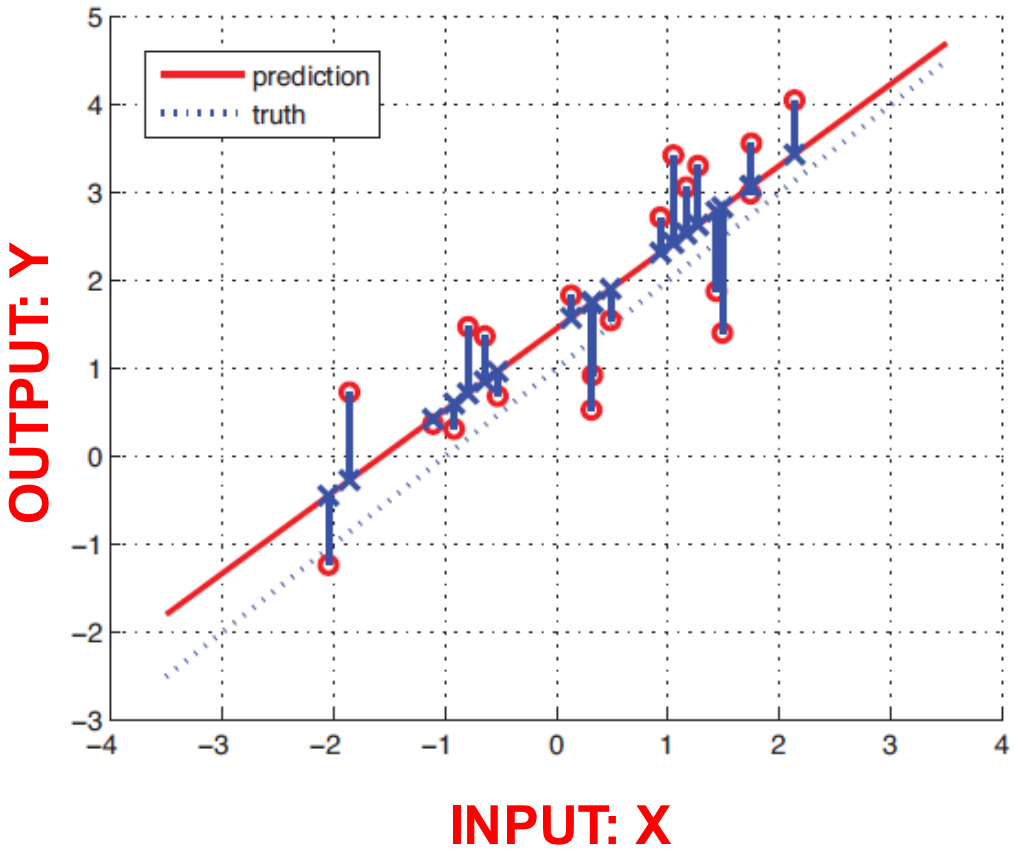
- True positive rate (TPR) $= \frac{\mathrm{TP}}{\mathrm{P}} = \frac{P(\hat{y}=+1, y=+1)}{P(y=+1)}$
  (aka recall, sensitivity)
- True negative rate (TNR) = $\frac{\mathrm{TN}}{\mathrm{N}}$
  (specificity)
- False positive rate (FPR) = $\frac{\mathrm{FP}}{\mathrm{N}}$
- False negative rate (FNR) = $\frac{\mathrm{FN}}{\mathrm{P}}$



P = TP + FN N = FP + TN

- Precision = $\frac{\mathrm{TP}}{\mathrm{P}-called} = \frac{P(\hat{y}=+1, y=+1)}{P(\hat{y}=+1)}$, $\mathrm{P} - called = \mathrm{TP} + \mathrm{FP}$

Linear Regression



Regression Learn a function that predicts outputs from inputs,

$$y = f(x)$$

Outputs y are real-valued

Linear Regression As the name suggests, uses a *linear function*:

$$y = w^T x + b$$

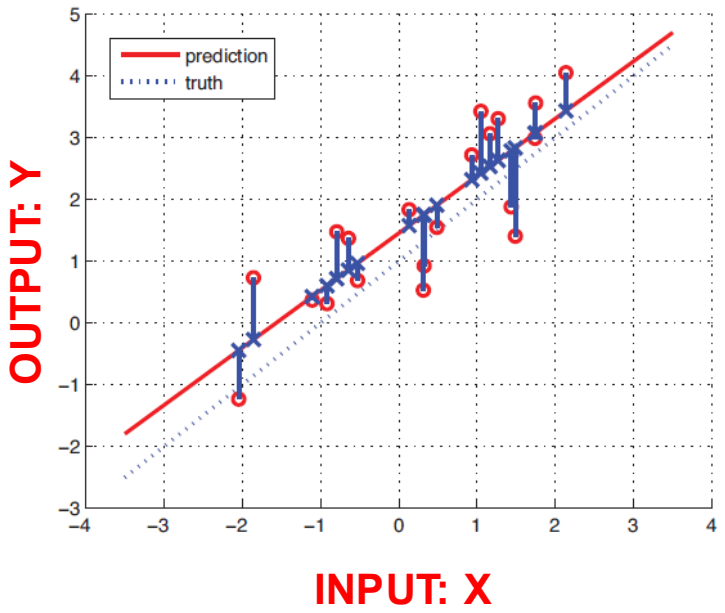
We will add noise later…

Linear Regression

Input-output mapping is not exact, so we will add zero-mean Gaussian noise,

Multivariate Normal (uncorrelated)

$$y = w^T x + \epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$



This is equivalent to the likelihood function,

$$p(y \mid w, x) = \mathcal{N}(y \mid w^T x, \sigma^2)$$

Because Adding a constant to a Normal RV is still a Normal RV,

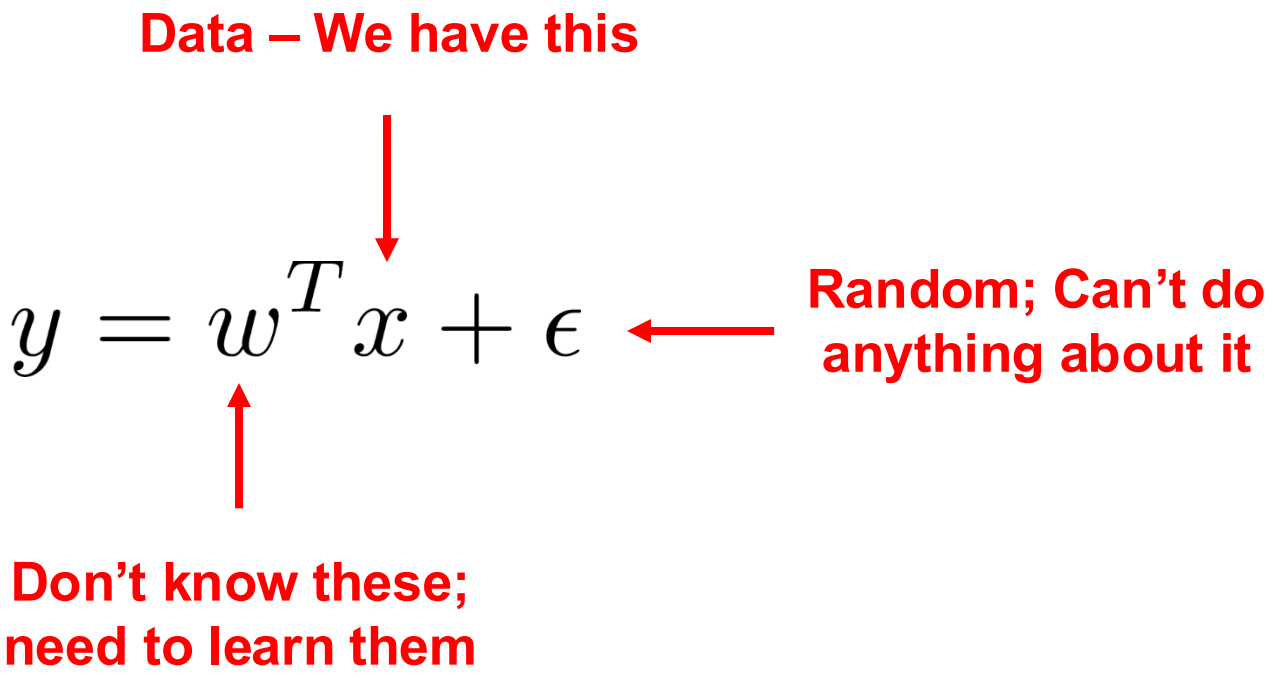
$$z \sim \mathcal{N}(m, P) \qquad z + c \sim \mathcal{N}(m + c, P)$$

In the case of linear regression $z \to \epsilon$ and $c \to w^T x$

Great, we're done right?

We need to fit it to data by learning the regression weights

How to do this? What makes *good* weights?



Learning Linear Regression Models

There are several ways to think about fitting regression:

- **Intuitive** Find a plane/line that is close to data
- **Functional** Find a line that minimizes the *least squares* loss
- **Estimation** Find maximum likelihood estimate of parameters

They are all the same thing…

Learning Linear Regression Models

There are several ways to think about fitting regression:

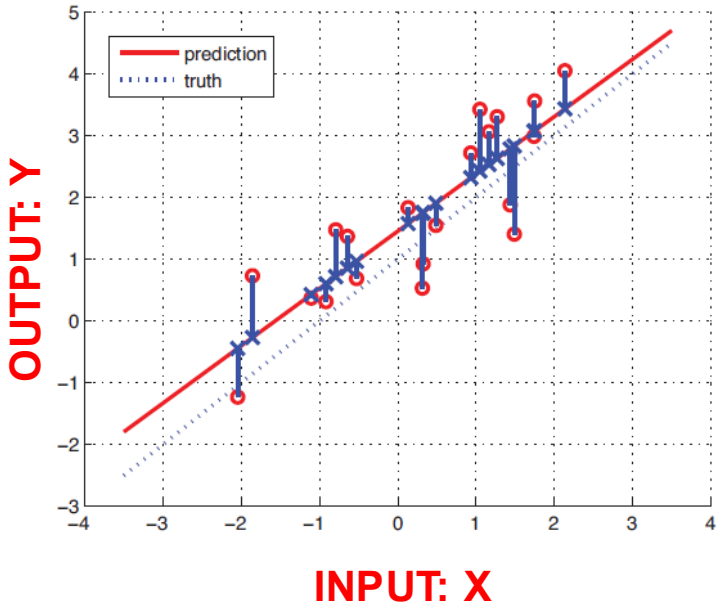
- **Intuitive** Find a plane/line that is close to data
- **Functional** Find a line that minimizes the *least squares* loss
- **Estimation** Find maximum likelihood estimate of parameters

They are all the same thing…

MLE for Linear Regression

Given training data $\{(x_i, y_i)\}_{i=1}^N$ likelihood function is given by,

$$\log \prod_{i=1}^{N} p(y_i \mid x_i, w) = \sum_{i=1}^{N} \log p(y_i \mid x_i, w)$$



Recall that the likelihood is Gaussian:

$$p(y \mid w, x) = \mathcal{N}(y \mid w^T x, \sigma^2)$$

So MLE maximizes the log-likelihood over the whole data as,

$$w^{\text{MLE}} = \arg\max_w \sum_{i=1}^{N} \log \mathcal{N}(y_i \mid w^T x_i, \sigma^2)$$

MLE of Gaussian Mean

Assume data are i.i.d. univariate Gaussian,

$$p(\mathcal{Y} \mid \mu) = \prod_{i=1}^{N} \mathcal{N}(y_i \mid \mu, \sigma^2)$$

Variance is known

Log-likelihood function:

$$\mathcal{L}(\mu) = \sum_{i=1}^{N} \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left(-\frac{1}{2}(y_i - \mu)^2 \sigma^{-2} \right) \right)$$

Constant doesn't depend on mean

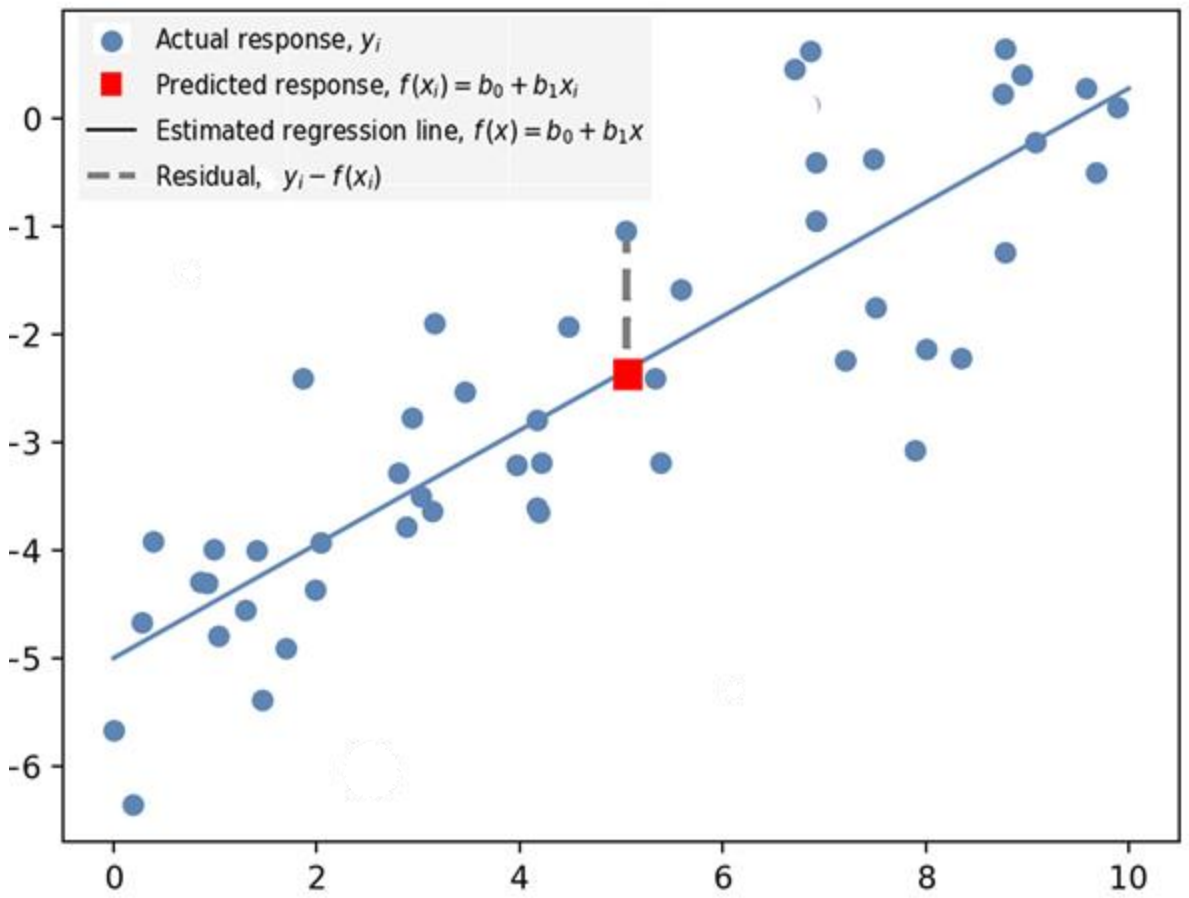
$$= \text{const.} - \frac{1}{2} \sum_{i=1}^{N} \left((y_i - \mu)^2 \sigma^{-2} \right)$$

MLE doesn't change when we:
1) Drop constant terms (in μ)
2) Minimize negative log-likelihood

MLE estimate is *least squares estimator*:

$$\mu^{\text{MLE}} = -\frac{1}{2\sigma^2} \arg\max_{\mu} \sum_{i=1}^{N} (y_i - \mu)^2 = \arg\min_{\mu} \sum_{i=1}^{N} (y_i - \mu)^2$$

MLE of Linear Regression



Substitute linear regression prediction into MLE solution and we have,

$$\min_w \sum_{i=1}^{N} (y_i - wx_i)^2$$

So for Linear Regression, MLE = Least Squares Estimation

https://www.activestate.com/resources/quick-reads/how-to-run-linear-regressions-in-python-scikit-learn/

MLE of Linear Regression

Using previous results, MLE is equivalent to minimizing squared residuals,

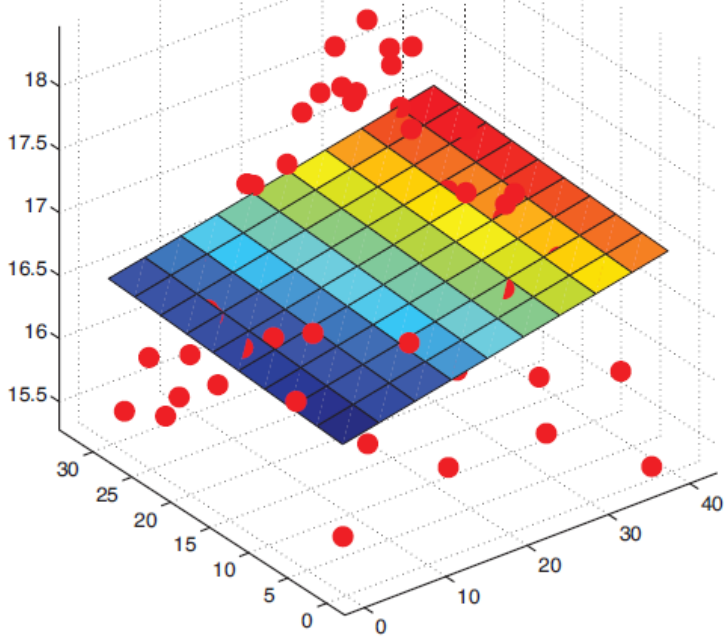
$$\min_w \sum_{i=1}^{N} (y_i - w^T x_i)^2 = \|\mathbf{y} - w^T \mathbf{X}\|^2$$

Some slightly more advanced linear algebra gives us a solution,

$$w = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Ordinary Least Squares (OLS) solution

[Image: Murphy, K. (2012)]



Derivation a bit involved for lecture but…

- We know it has a closed-form and why
- We can evaluate it
- Generally know where it comes from

Basis Functions

- A **basis function** can be any function of the input features X
- Define a set of m basis functions $\phi_1(x), \ldots, \phi_m(x)$
- Fit a linear regression model in terms of basis functions,

$$y = \sum_{i=1}^{m} w_i \phi_i(x) = w^T \phi(x)$$

- Regression model is *linear in the basis transformations*
- Model is *nonlinear in the data X*

Kernel Functions

*A **kernel function** is an inner-product of some basis function computed on two inputs*

$$k(x, x') = \boldsymbol{\phi}(x)^{\mathrm{T}} \boldsymbol{\phi}(x') = \sum_{i=1}^{M} \phi_i(x)\phi_i(x')$$

A consequence is that kernel functions are non-negative real-valued functions over a pair of inputs,

$$\kappa(x, x') \in \mathbb{R} \qquad \kappa(x, x') \geq 0$$

Kernel functions can be interpreted as a measure of distance between two inputs

Kernel Functions

Example The *linear basis* $\phi(x) = x$ produces the kernel,

$$\kappa(x, x') = \phi(x)^T \phi(x') = x^T x'$$

It is often easier to directly specify the kernel rather than the basis function…

Example Gaussian kernel models similarity according to an unnormalized Gaussian distribution,

$$\kappa(x, x') = \exp\left(-\frac{1}{2\sigma^2}(x - x')^2\right)$$

Note Despite the name, this is **not** a Gaussian probability density.

Also called a *radial basis function* (RBF)

Kernel Functions

Given *any* set of data $\{x_i\}_{i=1}^n$ a necessary and sufficient condition of a valid kernel function is that the nxn **gram matrix**,

$$\mathbf{K} = \begin{pmatrix} \kappa(x_1, x_1) & \kappa(x_1, x_2) & \dots & \kappa(x_1, x_n) \\ \kappa(x_2, x_1) & \kappa(x_2, x_2) & \dots & \kappa(x_2, x_n) \\ \vdots & \vdots & \vdots & \vdots \\ \kappa(x_n, x_1) & \kappa(x_n, x_2) & \dots & \kappa(x_n, x_n) \end{pmatrix}$$

Is a *symmetric positive semidefinite matrix.*

Kernel Ridge Regression

Kernel representation requires inversion of NxN matrix

Primal

$$\mathbf{\Phi} = \begin{pmatrix} 1 & \phi_1(x_1) & \dots & \phi_M(x_1) \\ 1 & \phi_1(x_2) & \dots & \phi_M(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \phi_1(x_N) & \dots & \phi_M(x_N) \end{pmatrix}$$

$$w = (\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I)^{-1}\mathbf{\Phi}^T\mathbf{y}$$

MxM Matrix Inversion
$O(M^3)$

Dual

$$\mathbf{K} = \begin{pmatrix} \kappa(x_1, x_1) & \kappa(x_1, x_2) & \dots & \kappa(x_1, x_n) \\ \kappa(x_2, x_1) & \kappa(x_2, x_2) & \dots & \kappa(x_2, x_n) \\ \vdots & \vdots & \vdots & \vdots \\ \kappa(x_n, x_1) & \kappa(x_n, x_2) & \dots & \kappa(x_n, x_n) \end{pmatrix}$$

$$y(x) = \mathbf{k}(\mathbf{x})^T(\mathbf{K} + \lambda I)^{-1}\mathbf{y}$$

NxN Matrix Inversion
$O(N^3)$

Number of training data N greater than basis functions M