Computer Science

CSC380: Principles of Data Science

Basic machine learning 3

Chicheng Zhang

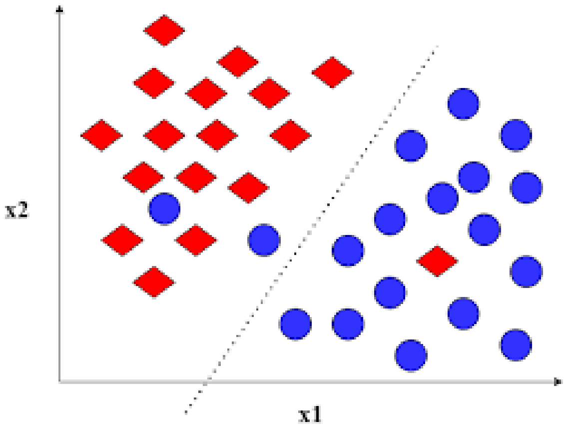
Outline

- Support Vector Machines
- Nonlinear models
  - Basis functions, kernels
  - Neural networks
- Unsupervised learning: clustering

Support vector machines

Classification

For this section (SVMs):



- We will focus on classification with binary labels

- We will use the convention that the labels of examples are in $\{-1, +1\}$

Linear classifier is a hyperplane

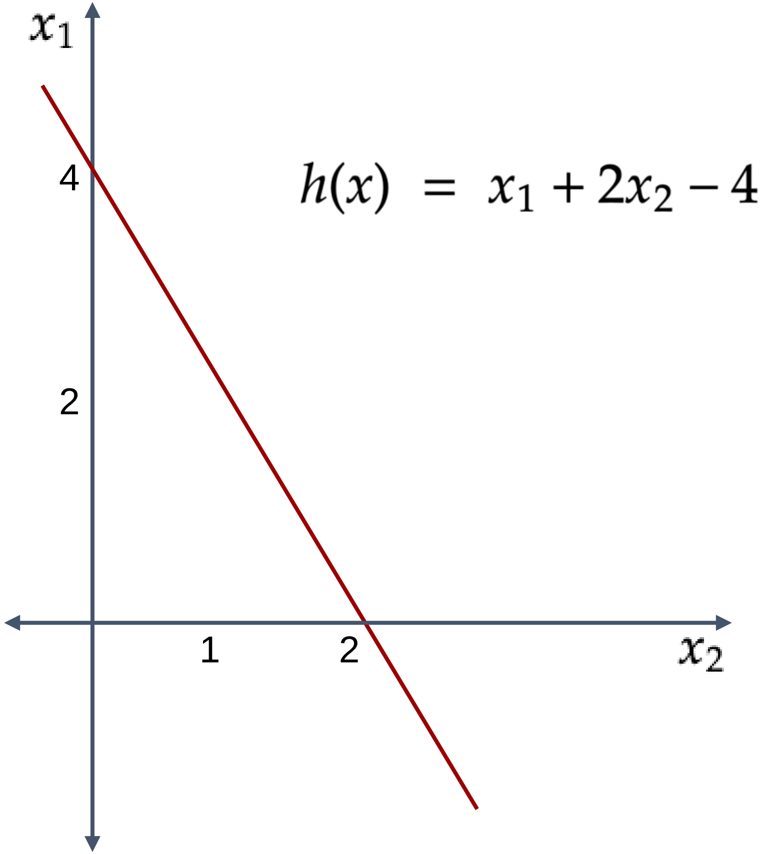
A linear classifier in d dimensions is given by a hyperplane, defined as follows:

Notation: inner product

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$
$$= w_1x_1 + w_2x_2 + \cdots + w_dx_d + b$$

For points that lie on the hyperplane, we have:

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = 0$$



Math Interlude: geometry of inner product

- Inner product (dot product):

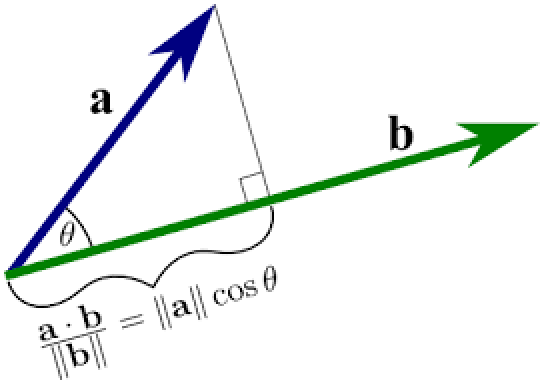
$$a \cdot b = \sum_{i=1}^{d} a_i \cdot b_i$$

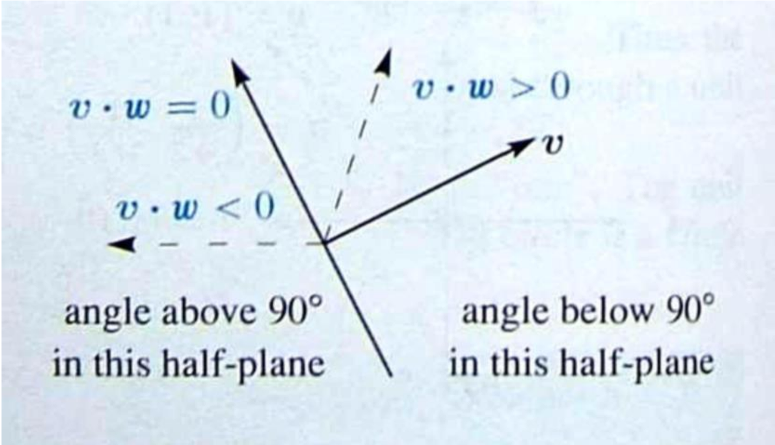
Same as $a^T b$

- Another way to find it:

$$\langle a, b \rangle = ||a||_2 \cdot ||b||_2 \cdot \cos(\theta)$$

where $\theta \in [0, \pi]$ is the angle between them

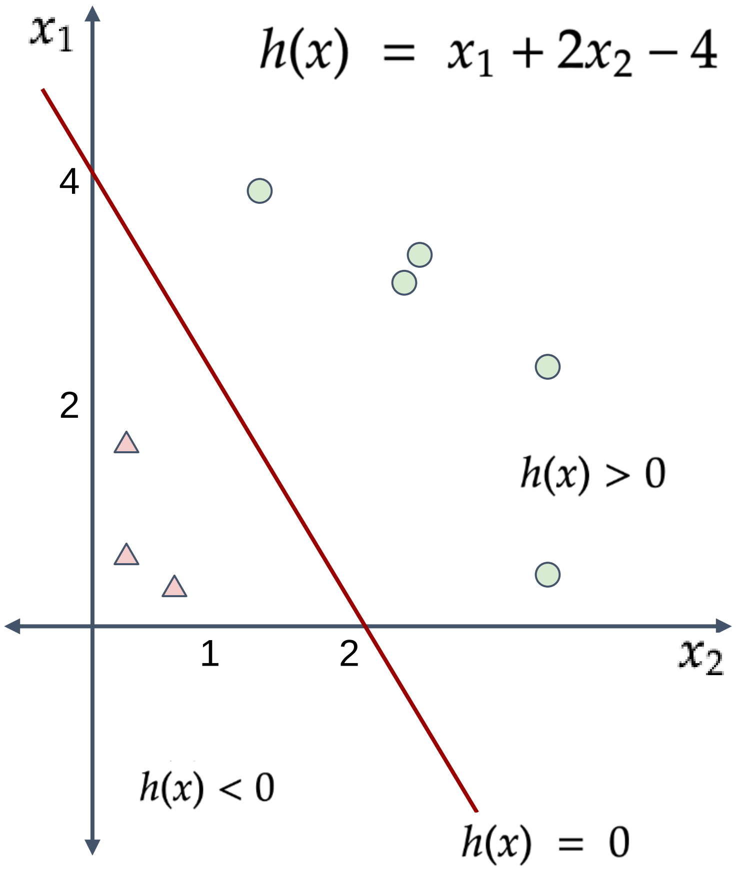




Separating Hyperplane

A hyperplane h(x) splits the original d-dimensional space into two half-spaces. If the input dataset is linearly separable:

$$y = \begin{cases} +1 & \text{if } h(\mathbf{x}) > 0 \\ -1 & \text{if } h(\mathbf{x}) < 0 \end{cases}$$



Separating Hyperplane: weight vector

Fact The weight vector **w** is orthogonal to the hyperplane.

w also known as the *normal vector*

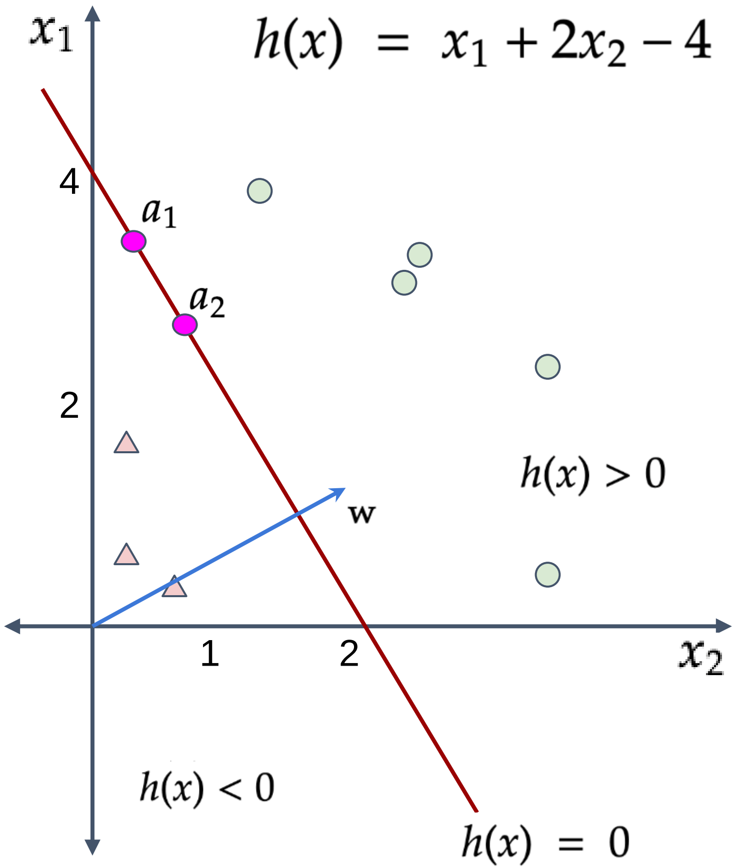
Let a_1 and a_2 be two arbitrary points that lie on the hyperplane, we have:

$$h(\mathbf{a}_1) = \mathbf{w}^T\mathbf{a}_1 + b = 0$$

$$h(\mathbf{a}_2) = \mathbf{w}^T\mathbf{a}_2 + b = 0$$

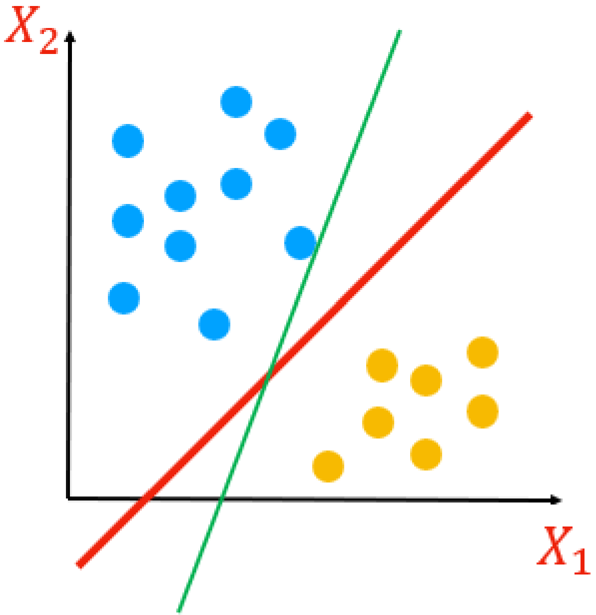
Subtracting one from the other:

$$\mathbf{w}^T(\mathbf{a}_1 - \mathbf{a}_2) = 0$$



Linear Decision Boundary

Any boundary that separates classes is equally good on training data

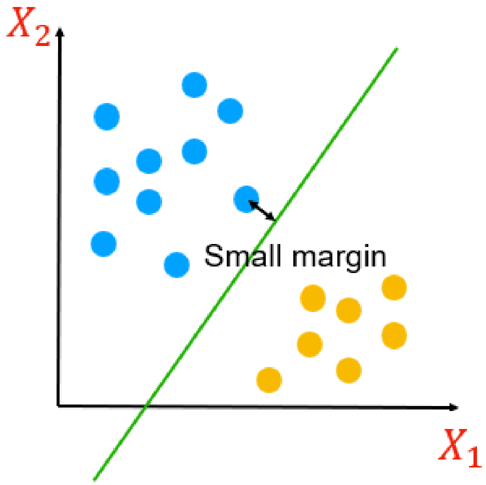


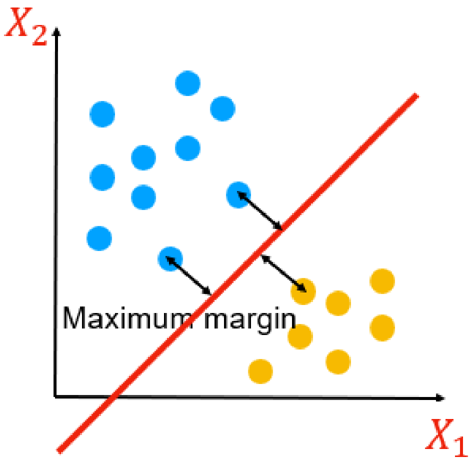
But are they equally good on unseen test data?

Which boundary is better, red or green?

[Source: http://www-bcf.usc.edu/~gareth/ISL/]

Classifier Margin



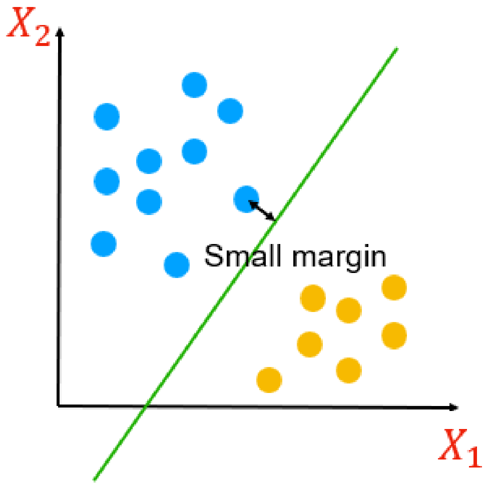


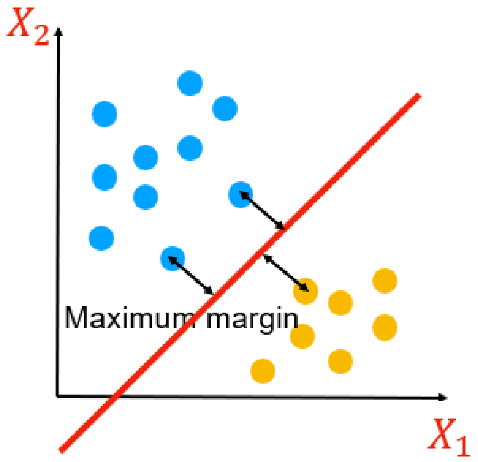
*The **margin** measures minimum distance between each class and the decision boundary*

Observation Decision boundaries with larger margins are more likely to generalize to unseen data

Idea Learn the classifier with the largest margin that still separates the data…

…we call this a *max-margin classifier*





Linear classification $f(x) = w \cdot x + b$

Predict + if $f(x) > 0$

gives decision boundaries that are straight

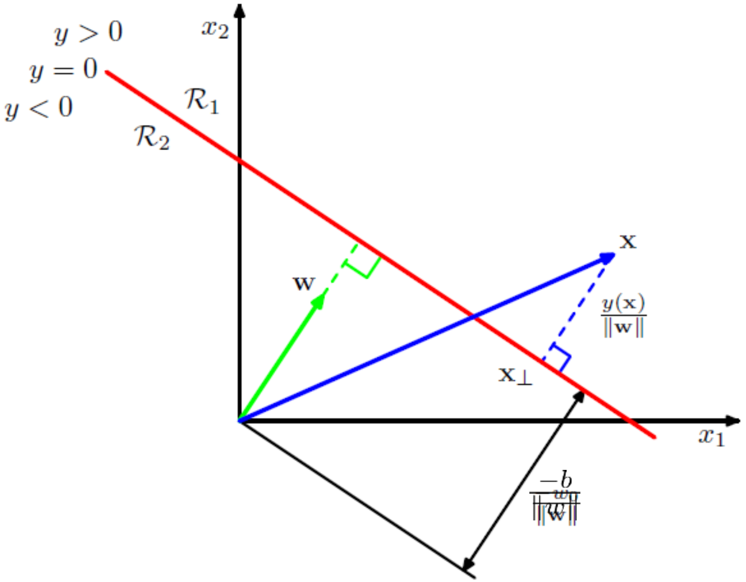
Observation Decision boundaries with larger margins are more likely to generalize to unseen data

Support vector machines (SVMs) find decision boundary with large margins

Background: distance of a point to decision boundary

A linear classifier is given by

$$f(x) = w^T x + b$$



Decision boundary is now at $f(x) = 0$ and distance of x to it is:

$$\frac{f(x)}{\|w\|}$$

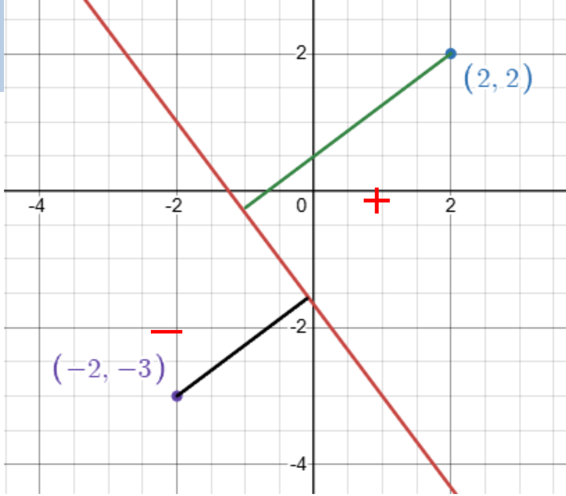
Where the norm of the weights is $\|w\| = \sqrt{w^T w} = \sqrt{\sum_i w_i^2}$

Known as the *distance from a point to a plane* equation: wiki/Distance_from_a_point_to_a_plane

Example

Linear classifier: $f(x) = 0.8x_1 + 0.6x_2 + 1$

Decision boundary: $0.8x_1 + 0.6x_2 + 1 = 0$



Distance of (2,2) to the boundary?

$$\frac{0.8 \times 2 + 0.6 \times 2 + 1}{\sqrt{0.8^2 + 0.6^2}} = 3.8$$

Distance of (-2,-3) to the boundary?

$$\frac{0.8 \times (-2) + 0.6 \times (-3) + 1}{\sqrt{0.8^2 + 0.6^2}} = -2.4$$

Here distances are *signed:*

sign represents which side the point is at

i.e, the predicted label

Classification margin

Given linear classifier $w \cdot x + b$, its *classification margin* on *labeled example* (x, y) is $\frac{y(w \cdot x + b)}{||w||_2}$

label x distance

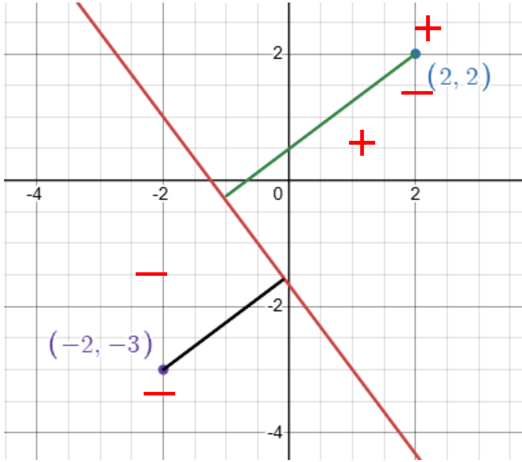
Example $f(x) = 0.8x_1 + 0.6x_2 + 1,$ $||w||_2 = 1$

x	y
$(2,2)$	$+$
$(-2,-3)$	$-$
$(2,2)$	$-$

margin = $+1 \times 3.8 = 3.8$

margin = $-(-2.4) = 2.4$

margin = $-1 \times 3.8 = -3.8$



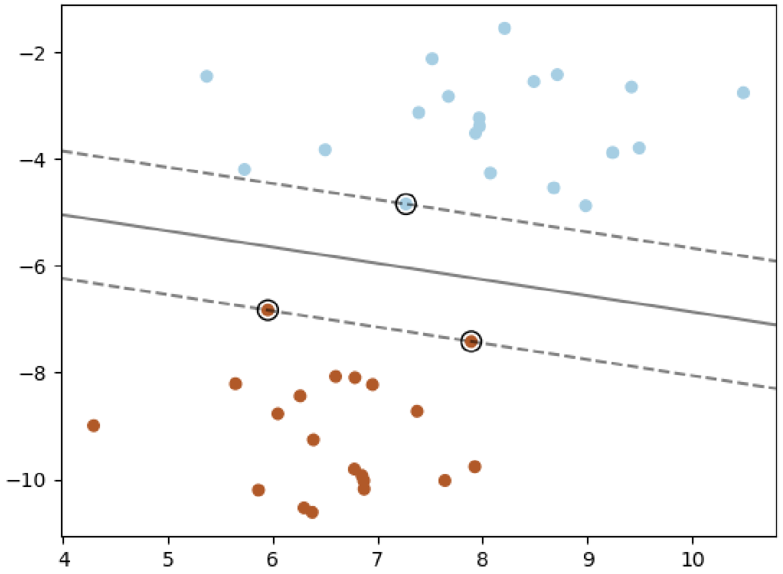
Margin > 0 ⇔ correct classification

Margin > 0 and larger margin: correct with higher confidence

Margin and Support Vectors

Over all n points, the ***margin*** of the linear classifier is the minimum distance of a point from the separating hyperplane:

$$\delta^* = \min_{\mathbf{x}_i} \left\{ \frac{y_i(\mathbf{w}^T\mathbf{x}_i + b)}{||\mathbf{w}||} \right\}$$



All the points that achieve this minimum distance are called ***support vectors***.

Maximum margin classifier

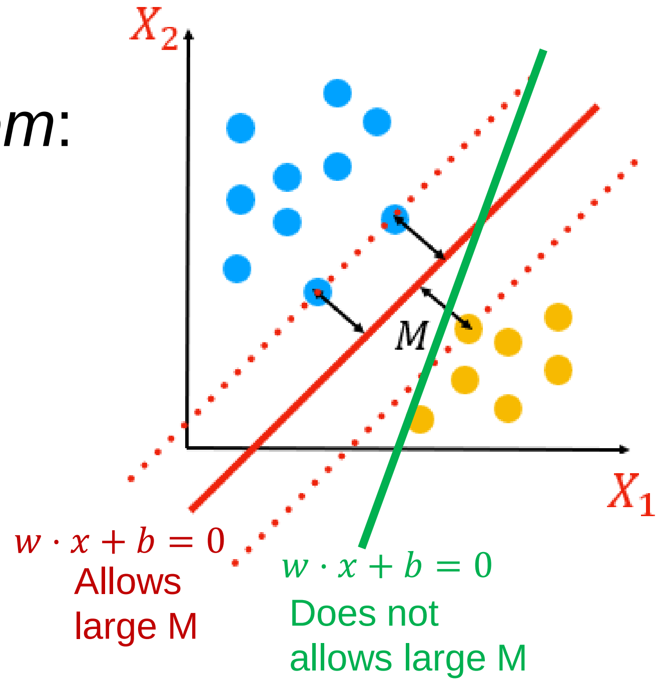
We can formulate finding a maximum margin classifier as an *optimization problem*:

Find $w, b, M \geq 0$ such that

$$\text{maximize } M$$

with the constraints that

$$\frac{y_i(w \cdot x_i + b)}{||w||_2} \geq M \quad \text{for all } i$$



Math Interlude: optimization problems

- The above falls to the general form of

$$\text{maximize } f(x)$$

subject to

$$g_i(x) \leq 0,\ i = 1, \ldots, m$$

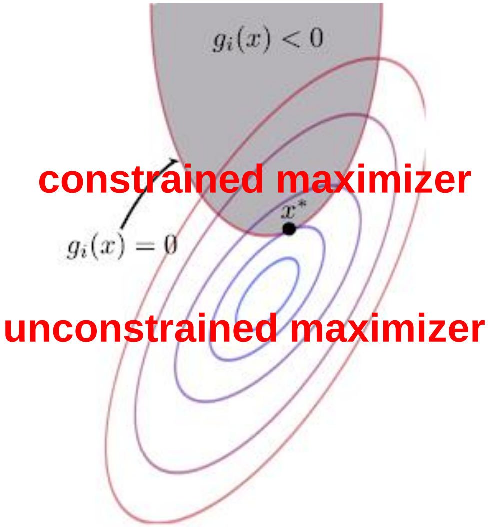
x: Optimization variables

constraints

- These are called *constrained* optimization problems

- Due to the constraints, finding the maximizer requires more care..

- Still, solvable by many standard packages



Math Interlude: optimization problems

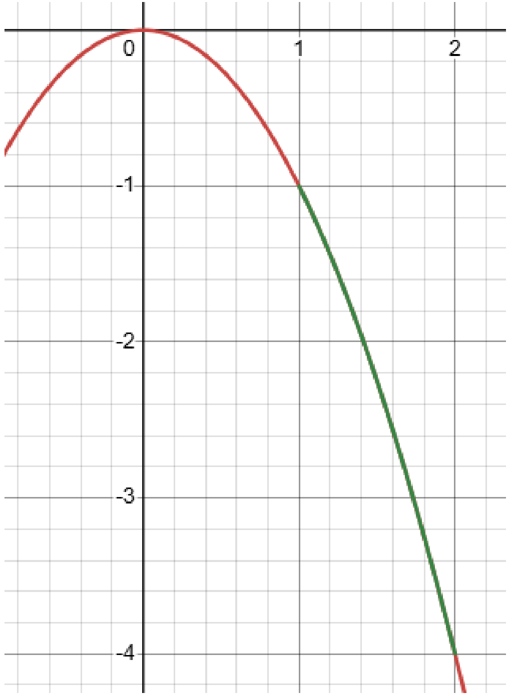
Example Find the solution of

$$\text{maximize } -x^2 \text{ subject to } x \geq 1 \text{ and } x \leq 2$$

Solution We can draw a picture..

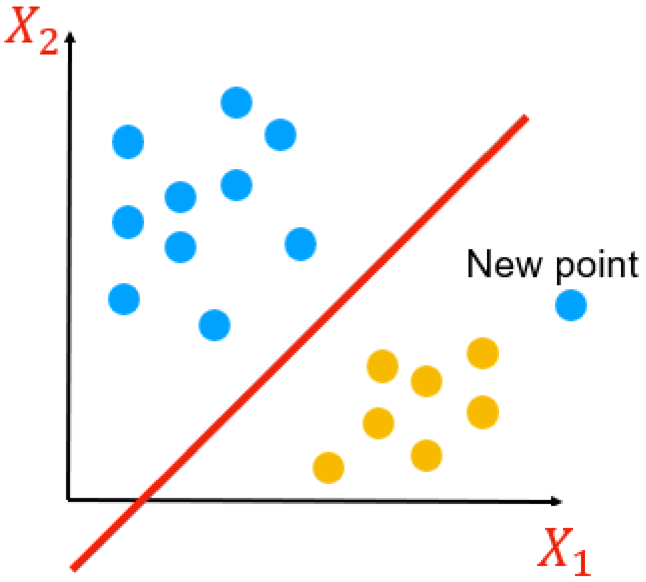
The objective is maximized at $x = 1$

Note: the constrained maximizer is **not** the vertex of the parabola (unconstrained maximizer)



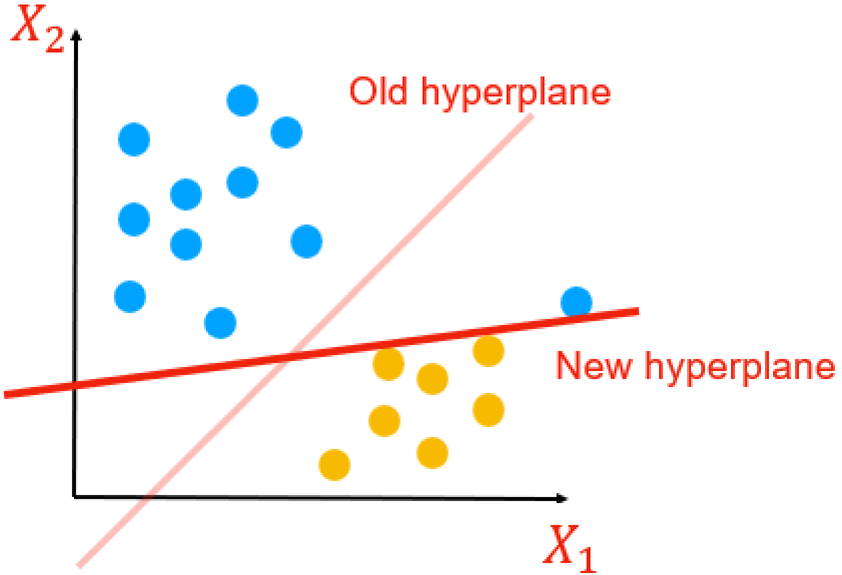
Support vector machine: extension

Problem 1: The maximum margin solution can be sensitive to outliers



Support vector machine: extension

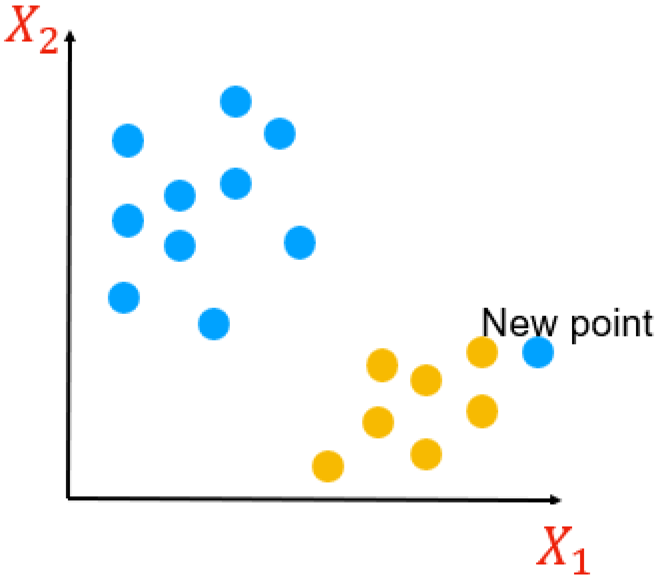
Problem 1: The maximum margin solution can be sensitive to outliers



Maybe prone to overfitting!

Support vector machine: extension

- Problem 2: The maximum margin solution may not even exist



No separating hyperplane (line in 2D)

Perhaps requiring the output classifier to predict every example correctly is too strict?

requirement of “hard margins”

Solution: soft margins – allow mistakes on some training examples

Soft margin support vector machines

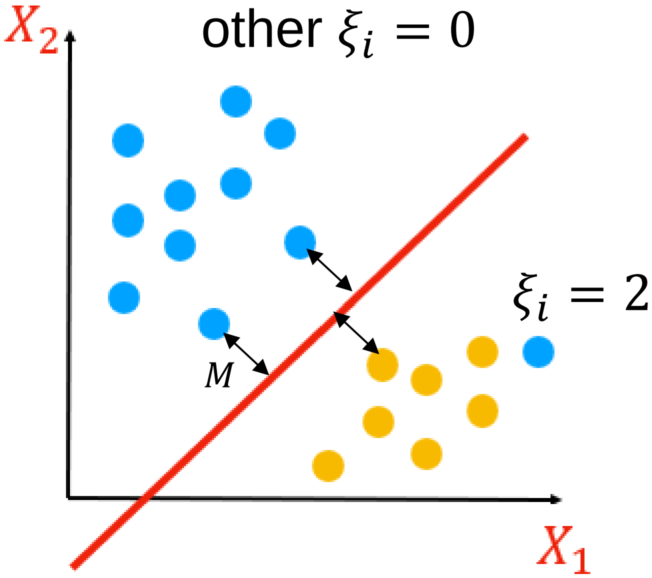
Find w, b, M, such that

$$\text{maximize } M$$

with the constraints that

$$\frac{y_i(w \cdot x_i + b)}{||w||_2} \geq M(1 - \xi_i) \text{ for all } i$$

and $\xi_i \geq 0$, $\sum_i \xi_i \leq C$



ξ_i: slack variables

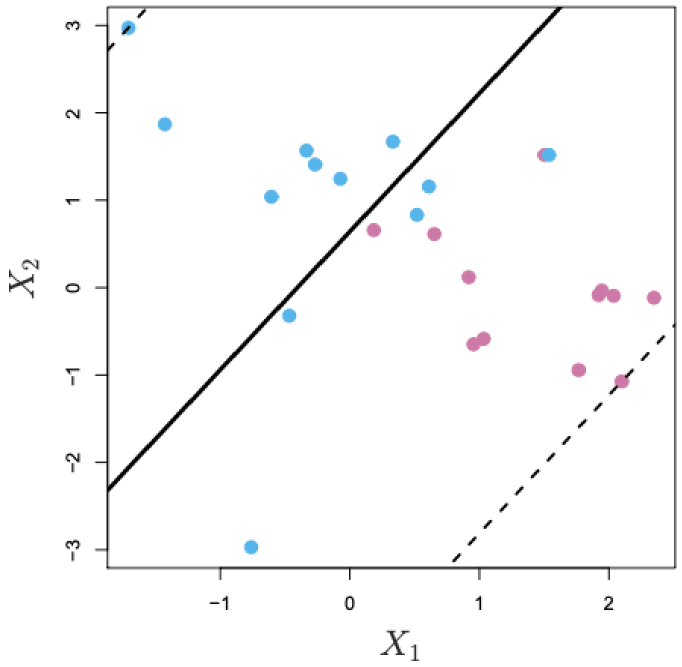
allows some examples to be on the wrong side ($\xi_i > 0$)

C: # in-margin examples allowed

Soft margin support vector machines

- Large C

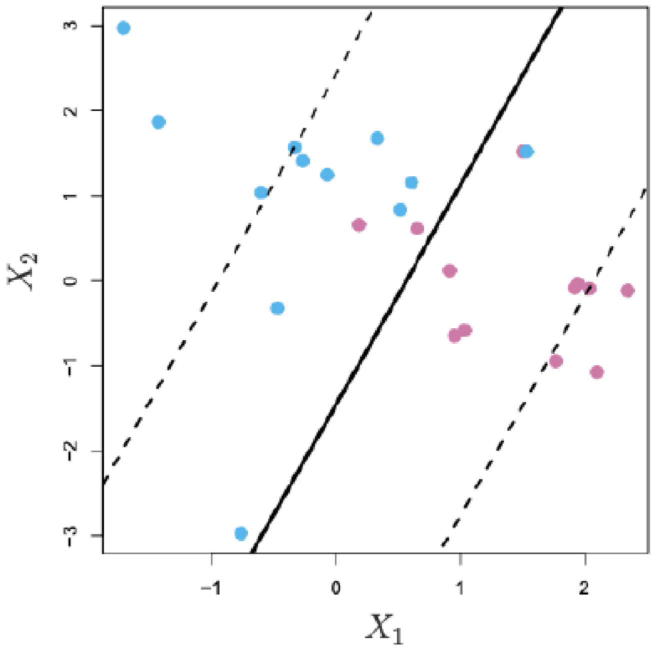
Many points inside the margin, many points on the wrong side of the line



Soft margin support vector machines

- Smaller C

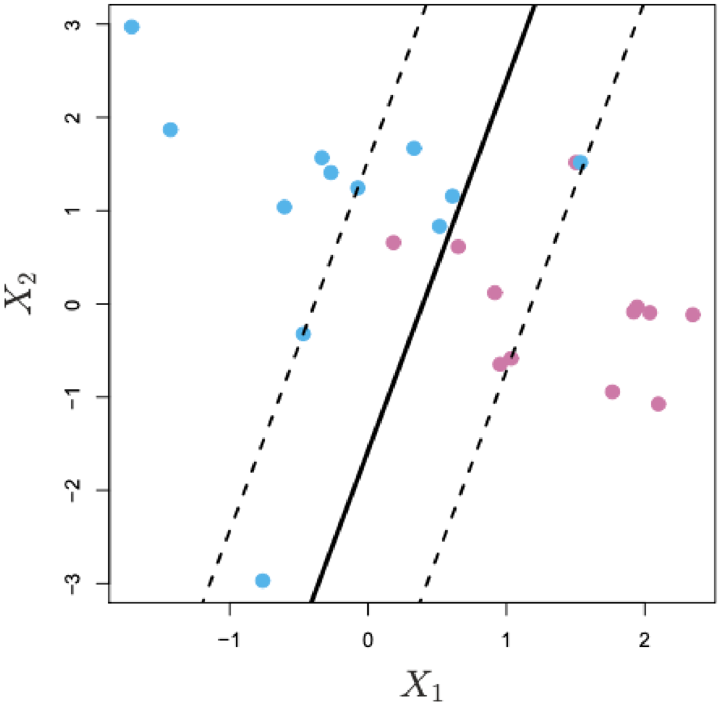
Fewer points inside the margin, Fewer points on the wrong side of the line



Soft margin support vector machines

- Even smaller C

Even fewer points inside the margin,
Very few points on the wrong side
of the line



Smaller C => More overfitting => Lower bias, higher complexity

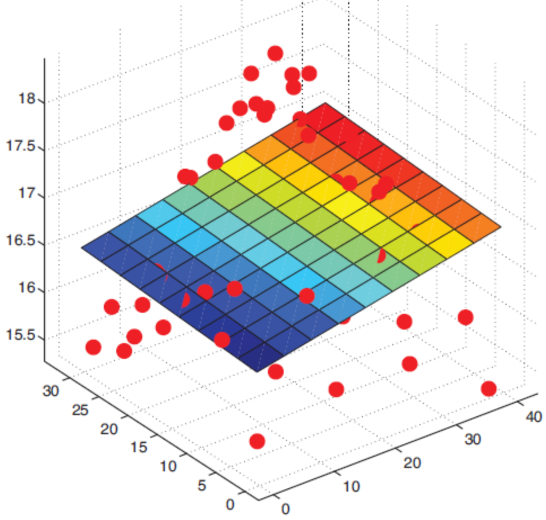
As usual, we can choose C by cross validation

Nonlinear prediction models

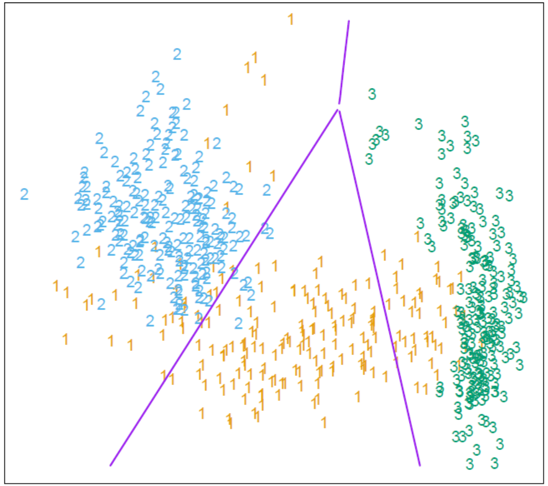
Nonlinear basis functions; kernels

Linear Models

[Image: Murphy, K. (2012)]



[Image: Hastie et al. (2001)]



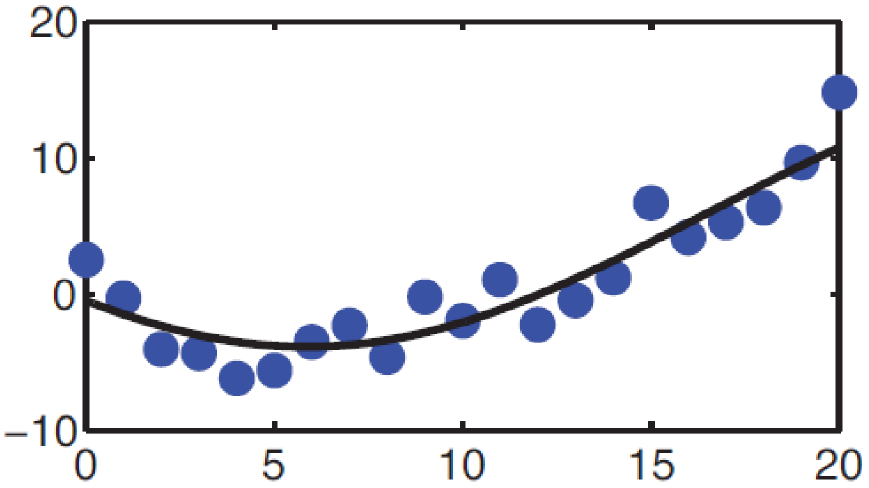
Linear Regression Fit a *linear function* to the data,

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

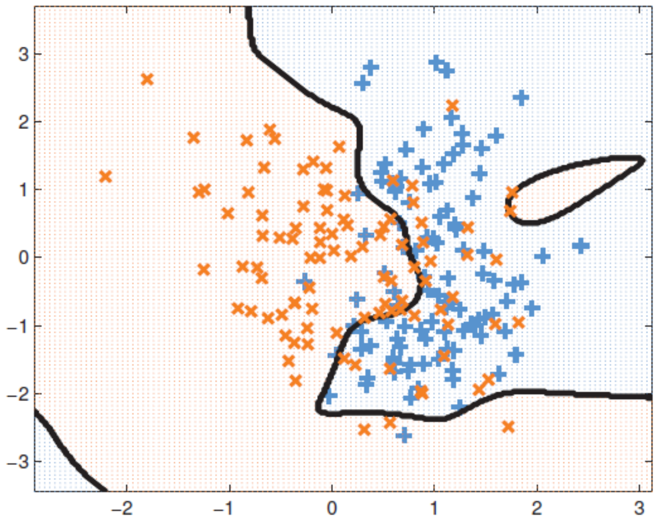
Logistic Regression Learn a decision boundary that is *linear in the data*,

$$P(y = 1 \mid w, x) = \sigma(w^T x)$$

Nonlinear Data



What if our data are *not* well-described by a linear function?



What if classes are *not linearly-separable*?

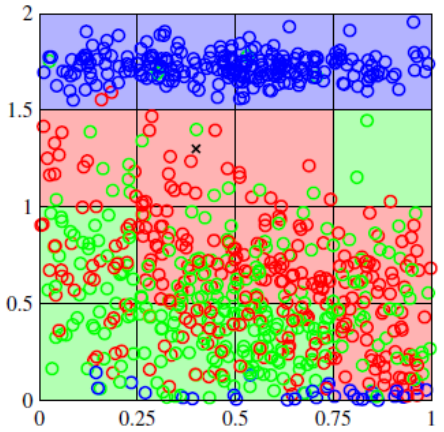
[Source: Murphy, K. (2012)]

Nonlinear prediction problems

- Nearest neighbor methods are OK, but they suffer from *the curse of dimensionality*

In high dimensions, all points are (kind-of) far from each other

For high-dimensional data, most cells are empty!



Alternative approach:

We can *reduce* learning nonlinear models

to learning linear models

Reducing nonlinear prediction to linear

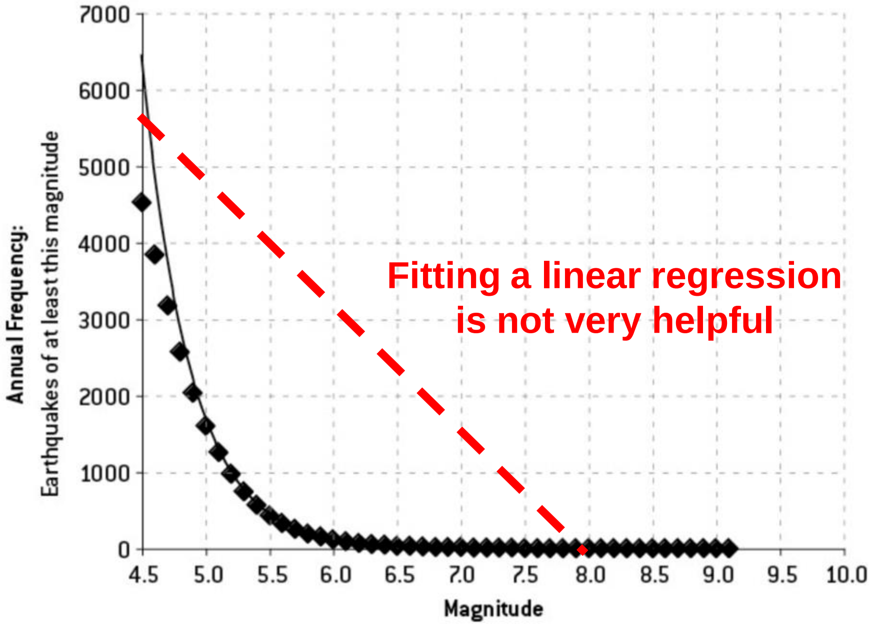
Two main approaches:

- Transforming the label
- Transforming the feature

Approach 1: Transforming the label

Suppose that we want to predict the number of earthquakes that occur of a certain magnitude. Our data are given by,

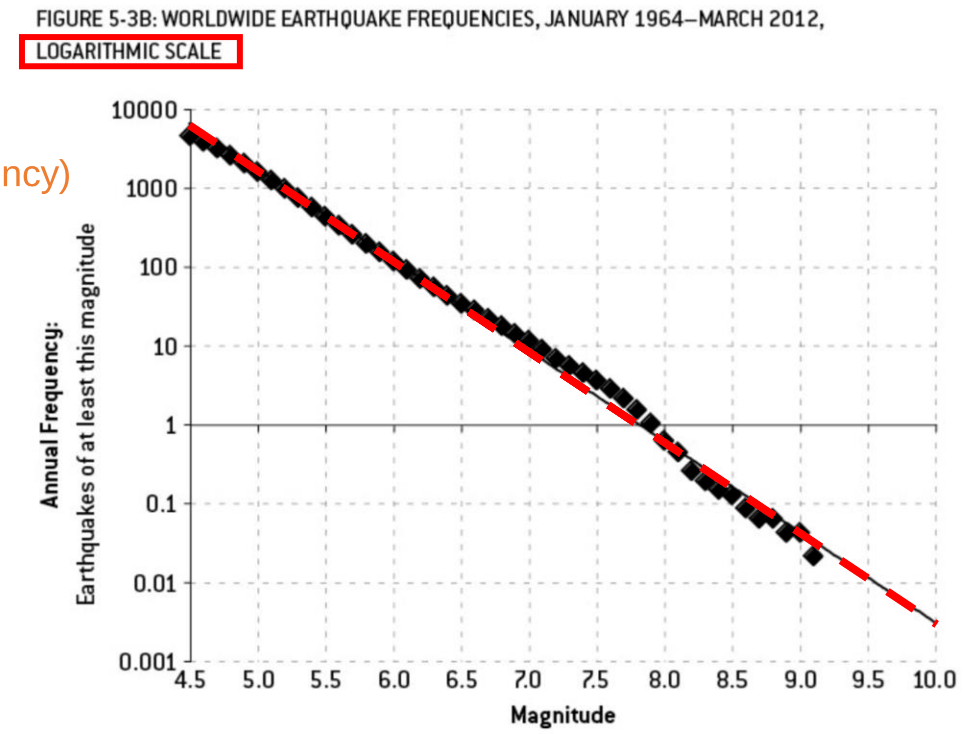
FIGURE 5-3A: WORLDWIDE EARTHQUAKE FREQUENCIES, JANUARY 1964–MARCH 2012



Approach 1: Transforming the label

Suppose that we want to predict the number of earthquakes that occur of a certain magnitude. Our data are given by,

(log frequency)

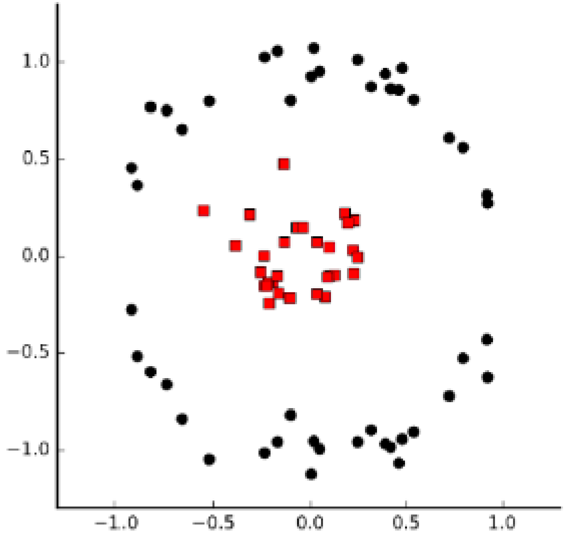


But plotting outputs on a logarithmic scale reveals a strong linear relationship…

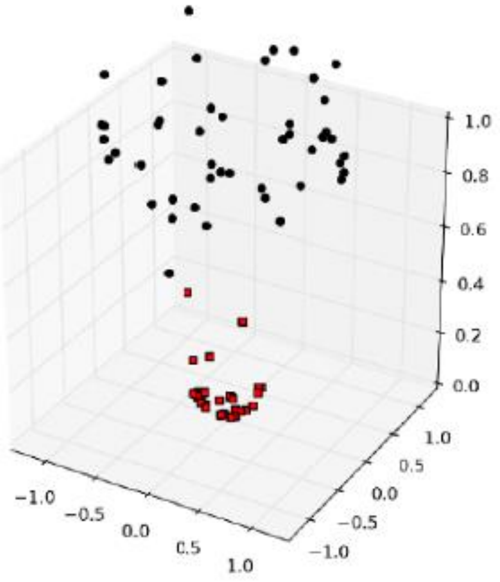
$$\log y = w \cdot x + b$$

We will do linear regression with new label $\log y$

Approach 2: transforming the features



$$\phi(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2)$$



Not Linearly separable

Linearly separable

Approach 2: transforming the features

- 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 model in terms of basis functions,

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

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

Common "All-Purpose" Basis Functions

- Linear basis functions recover the original linear model,

$$\phi_m(x) = x_m$$

Returns m[th] dimension of X

- Quadratic $\phi_m(x) = x_j^2$ or $\phi_m(x) = x_j x_k$ capture 2[nd] order interactions
- An order p polynomial $\phi \rightarrow x_d, x_d^2, \ldots, x_d^p$ captures higher-order nonlinearities (but requires O(d^p) parameters)
- Nonlinear transformation of single inputs,

$$\phi \rightarrow (\log(x_j), \sqrt{x_j}, \ldots)$$

- An indicator function specifies a region of the input,

$$\phi_m(x) = I(L_m \leq x_k < U_m)$$

I(A)=1 if A happens, =0 otherwise

sklearn.preprocessing.PolynomialFeatures

degree : *int or tuple (min_degree, max_degree), default=2*

If a single int is given, it specifies the maximal degree of the polynomial features. If a tuple `(min_degree, max_degree)` is passed, then `min_degree` is the minimum and `max_degree` is the maximum polynomial degree of the generated features. Note that `min_degree=0` and `min_degree=1` are equivalent as outputting the degree zero term is determined by `include_bias`.

interaction_only : *bool, default=False*

If `True`, only interaction features are produced: features that are products of at most `degree` *distinct* input features, i.e. terms with power of 2 or higher of the same input feature are excluded:

- included: `x[0]`, `x[1]`, `x[0] * x[1]`, etc.
- excluded: `x[0] ** 2`, `x[0] ** 2 * x[1]`, etc.

include_bias : *bool, default=True*

If `True` (default), then include a bias column, the feature in which all polynomial powers are zero (i.e. a column of ones - acts as an intercept term in a linear model).

order : *{'C', 'F'}, default='C'*

Order of output array in the dense case. `'F'` order is faster to compute, but may slow down subsequent estimators.

Example 1: Polynomial Basis Functions

Create three two-dimensional data points [0,1], [2,3], [4,5]:

```
>>> X = np.arange(6).reshape(3, 2)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5]])
```

Compute quadratic features $(1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)$,

```
>>> poly = PolynomialFeatures(degree=2)
>>> poly.fit_transform(X)
array([[ 1.,   0.,   1.,   0.,   0.,   1.],
       [ 1.,   2.,   3.,   4.,   6.,   9.],
       [ 1.,   4.,   5.,  16.,  20.,  25.]])
```

These are now our new data and ready to fit a model…

Example 2: Polynomial Regression

Create a 3rd order polynomial (cubic) regression data,

```
from sklearn.preprocessing import PolynomialFeatures
x = np.arange(5)
y = 3 - 2 * x + x ** 2 - x ** 3
y
```

```
array([  3,    1,   -5, -21, -53])
```

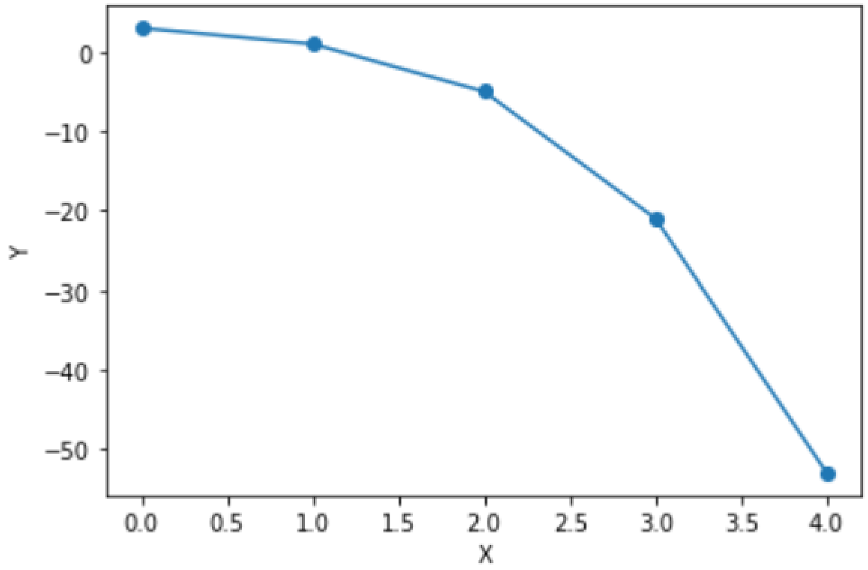
Create cubic features $(1, x, x^2, x^3)$,

```
from sklearn.linear_model import LinearRegression
poly = PolynomialFeatures(degree=3)
x_new = poly.fit_transform(x[:,np.newaxis])
x_new
```

```
array([[ 1.,   0.,   0.,   0.],
       [ 1.,   1.,   1.,   1.],
       [ 1.,   2.,   4.,   8.],
       [ 1.,   3.,   9.,  27.],
       [ 1.,   4.,  16.,  64.]])
```

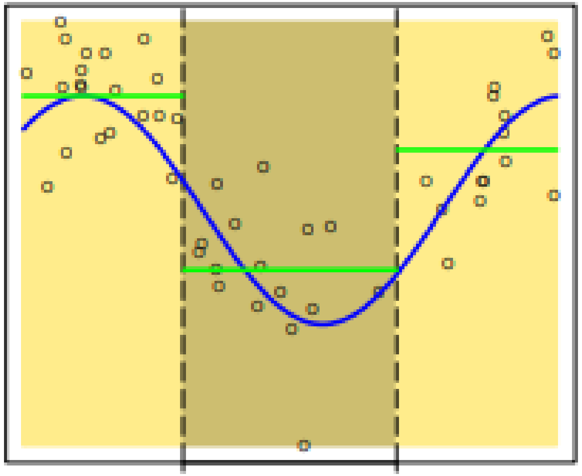
Example 2: Polynomial Regression

```
model = LinearRegression(fit_intercept=False).fit(x_new, y)
ypred = model.predict(x_new)
plt.scatter(x,y)
plt.plot(x,ypred,'-')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



Example: Piecewise Constant Regression

[Source: Hastie et al. (2001)]



Decompose the input space into 3 regions with indicator basis functions,

$$\phi_1(x) = I(x < \xi_1)$$
$$\phi_2(x) = I(\xi_1 \leq x < \xi_2)$$
$$\phi_3(x) = I(\xi_2 \leq x)$$

Fit linear regression model,

$$y = w_1\phi_1(x) + w_2\phi_2(x) + w_3\phi_3(x)$$

Effectively fits 3 constant functions to data in each region

Kernels

Training examples

Fact Many machine learning algorithms output linear models of the form $w = \sum_i \alpha_i x_i$ and thus makes prediction by

Sometimes called 'dual variables'

$$\sum_i \alpha_i x_i \cdot x + b$$

Examples: SVM, logistic regression

when learning with basis functions, the trained models make prediction by

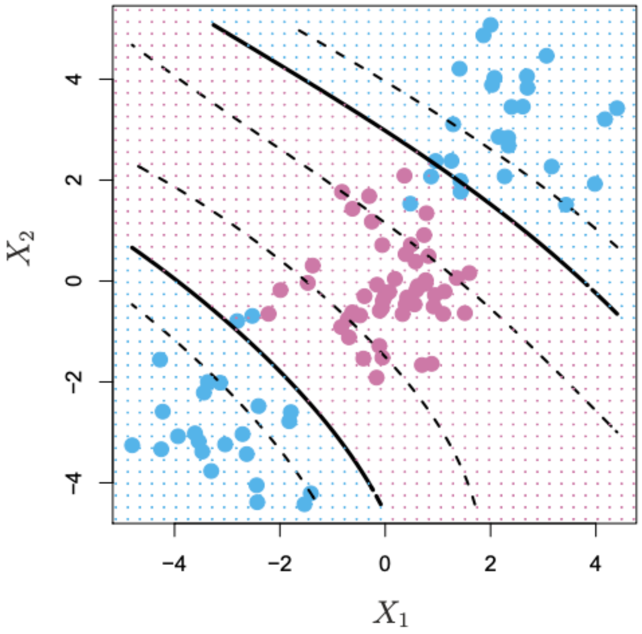
$$\sum_i \alpha_i \phi(x_i) \cdot \phi(x) + b$$

kernel: generalizes inner products; captures similarity between examples

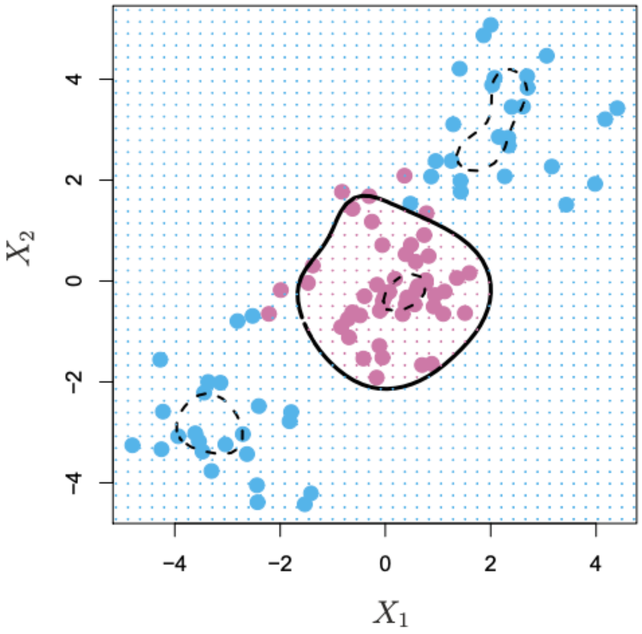
popular kernels: polynomial, radial

Kernel SVM

Applying kernel SVMs to nonlinear data obtains flexible nonlinear decision boundaries



polynomial (d=3) kernel



radial kernel

sklearn.svm.SVC

kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}, default='rbf'*

Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

gamma : *{'scale', 'auto'} or float, default='scale'*

Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

- if `gamma='scale'` (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features.

max_iter : *int, default=-1*

Hard limit on iterations within solver, or -1 for no limit.

verbose : *bool, default=False*

Enable verbose output. Note that this setting takes advantage of a per-process runtime setting in libsvm that, if enabled, may not work properly in a multithreaded context.

class_weight : *dict or 'balanced', default=None*

Set the parameter C of class i to class_weight[i]*C for SVC. If not given, all classes are supposed to have weight one. The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as `n_samples / (n_classes * np.bincount(y))`.

Example: Fisher's Iris Dataset

Train 8-degree polynomial kernel SVM classifier,

```
from sklearn.svm import SVC
svclassifier = SVC(kernel='poly', degree=8)
svclassifier.fit(X_train, y_train)
```

Generate predictions on held-out test data,

```
y_pred = svclassifier.predict(X_test)
```

Show confusion matrix and classification accuracy,

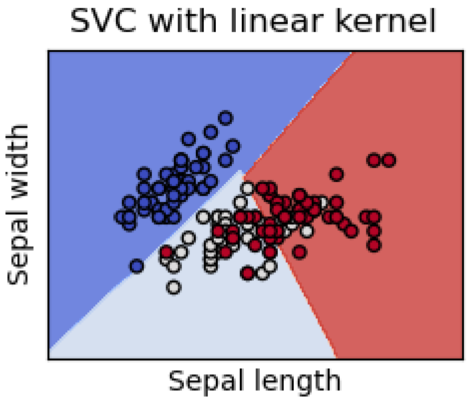
```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

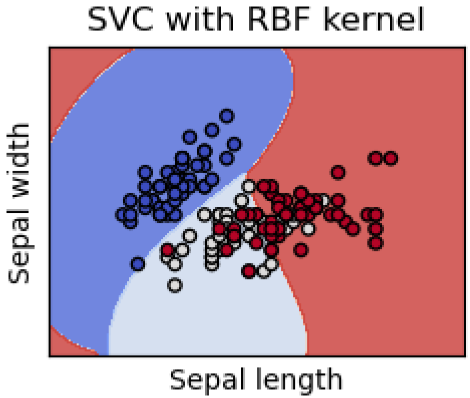
```
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]
```

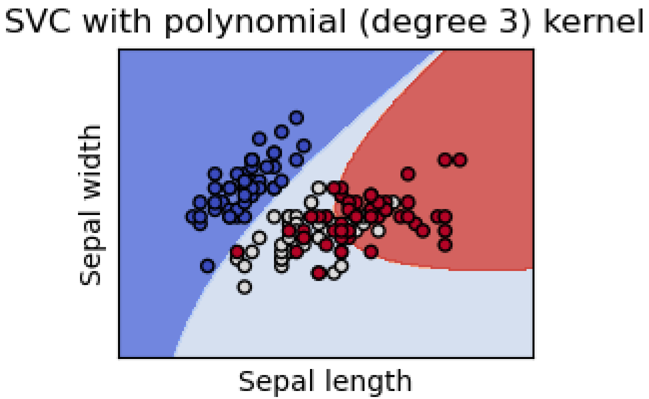
	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	1.00	0.92	0.96	13
Iris-virginica	0.86	1.00	0.92	6
avg / total	0.97	0.97	0.97	30

[Source: https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/]

Kernel SVM in Scikit Learn







- General kernel-based SVM lives in:

  [`sklearn.svm.svc(kernel='kernel_name')`]

Neural networks

Basis Functions

Basis functions transform linear models into nonlinear ones…

Linear Regression

$$y = w^T x$$

↓

$$y = w^T \phi(x)$$

Classification
(Logistic Regression)

$$y = \sigma(w^T x)$$

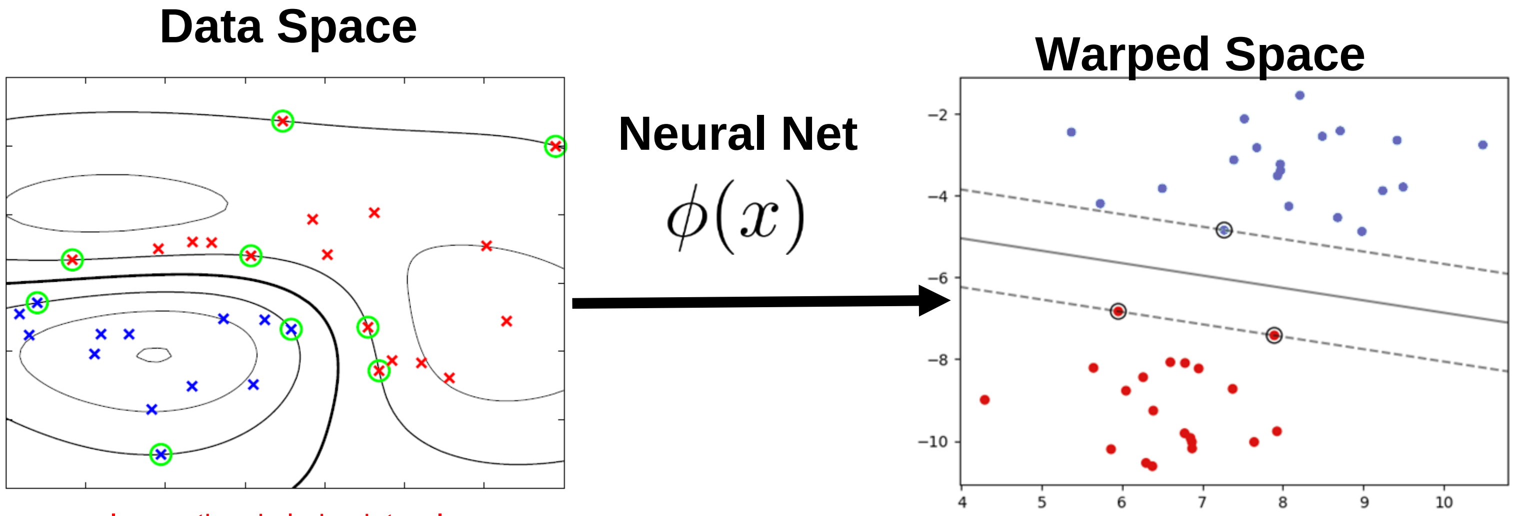
↓

$$y = \sigma(w^T \phi(x))$$

…but it may be difficult to find a good basis transformation

Learning Basis Functions

Wouldn't it be great if we could *learn* a basis function so that a simple linear model performs well…



Ignore the circled points…I
reused these from the SVM slides

This is called "representation learning"

Neural networks provides a flexible way to do this…

Neural Networks

Forms of NNs are used all over the place nowadays…

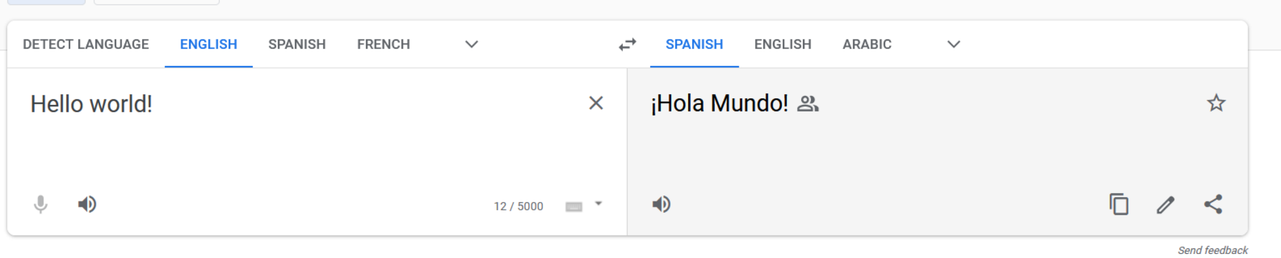
ChatGPT

AI Chat Bots

Self-Driving Cars

Creepy Robots

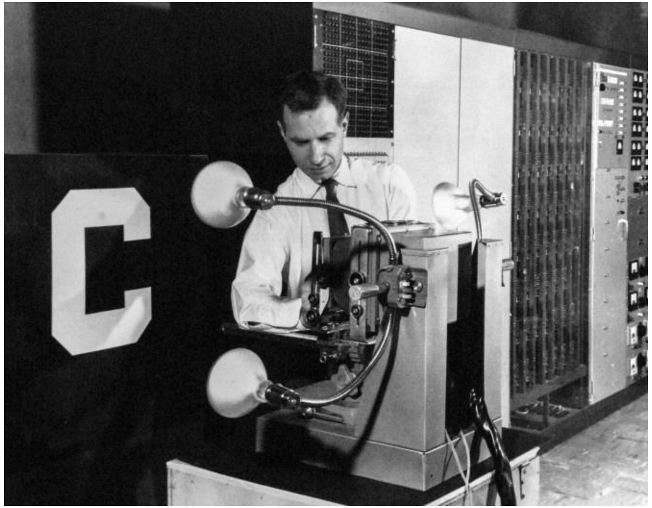
Machine Translation



Rosenblatt's Perceptron

Despite recent attention, neural networks are fairly old

In 1957 Frank Rosenblatt constructed the first (single layer) neural network known as a "perceptron"



He demonstrated that it is capable of recognizing characters projected onto a 20x20 "pixel" array of photosensors

Rosenblatt’s Perceptron

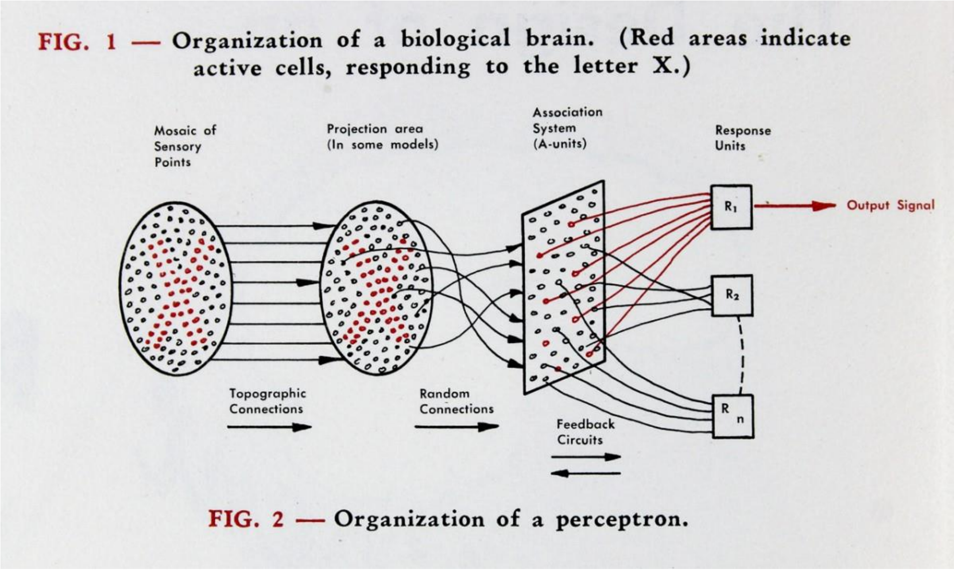
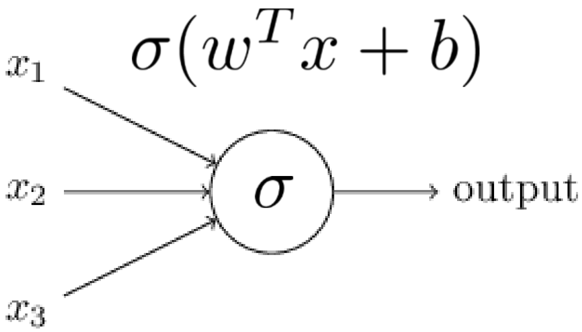


FIG. 1 — Organization of a biological brain. (Red areas indicate active cells, responding to the letter X.)

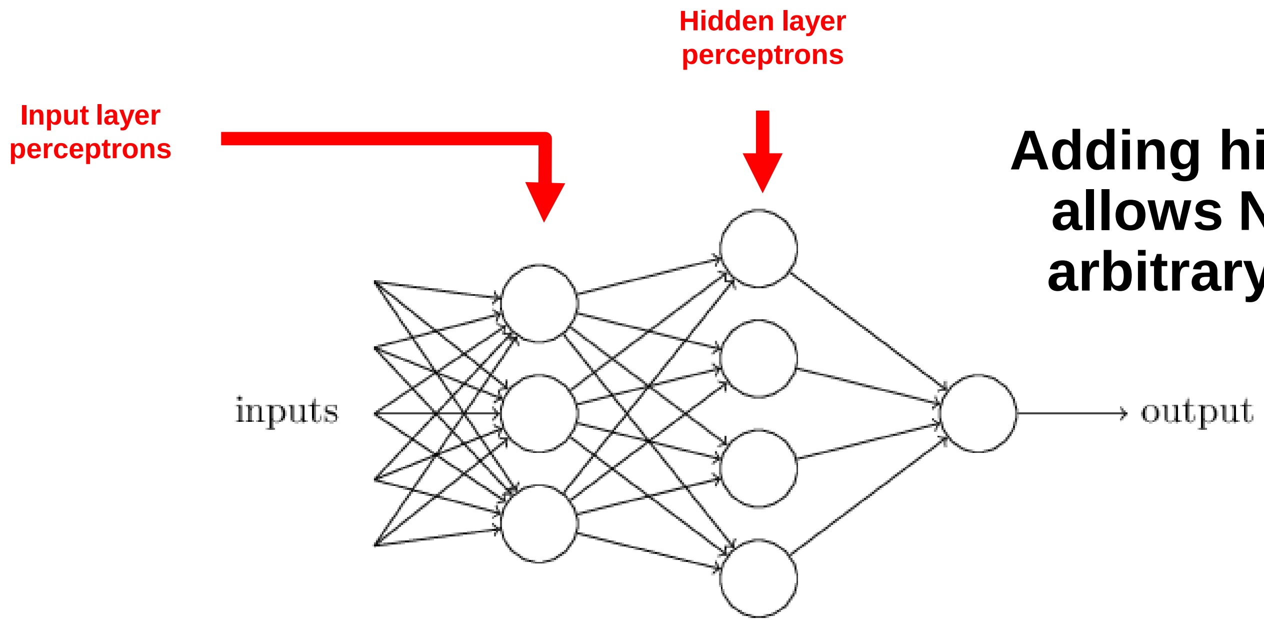
FIG. 2 — Organization of a perceptron.

Perceptron



- In Rosenblatt’s perceptron, the inputs are tied directly to output
- “Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanics” (1962)
- Criticized by Marvin Minsky in book “Perceptrons” since can only learn linearly-separable functions
- **The perceptron is just linear classification in disguise**

Multilayer Perceptron



Adding hidden layers allows NN to learn arbitrary functions

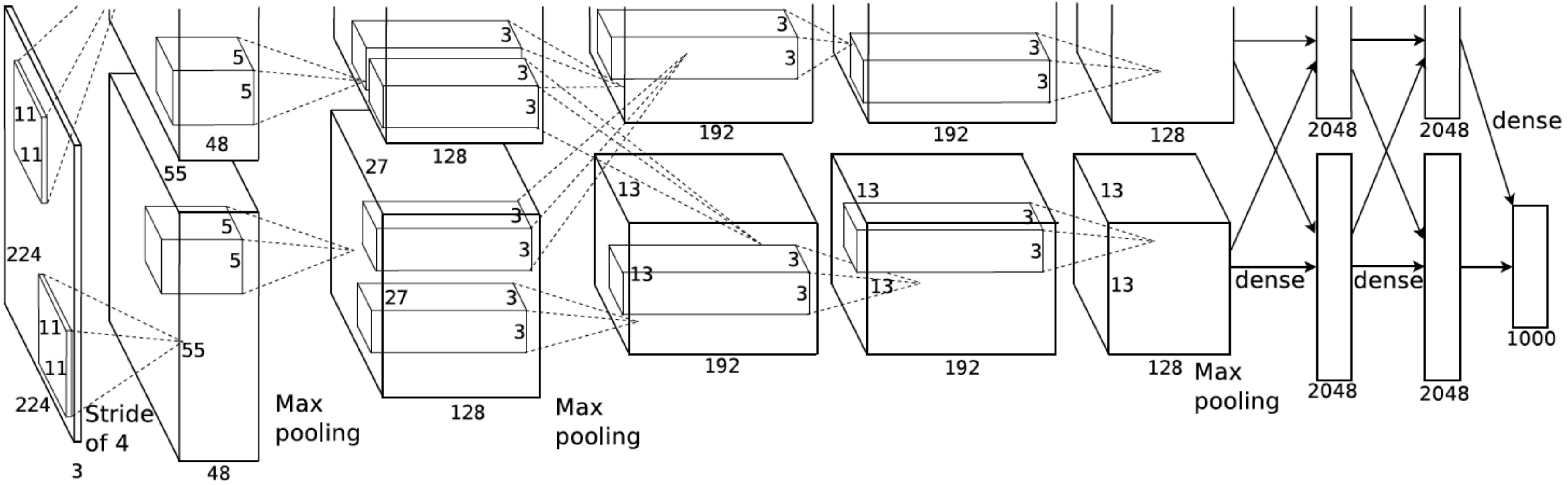
This is the quintessential *Neural Network…*

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

[Source: http://neuralnetworksanddeeplearning.com]

Modern Neural Networks: "deep learning"

Modern *Deep Neural networks* have many hidden layers



…and have millions - trillions of parameters to learn

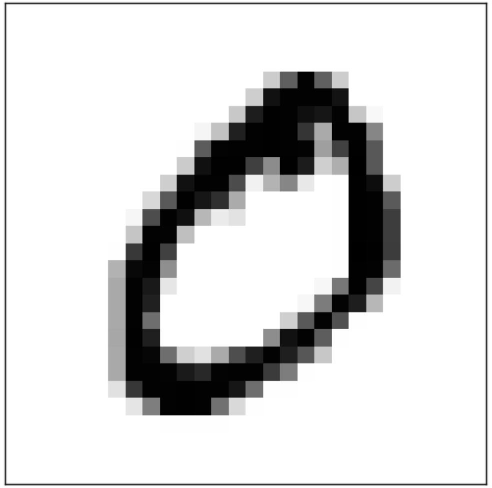
[Source: Krizhevsky et al. (NeurIPS 2012)]

Handwritten Digit Classification

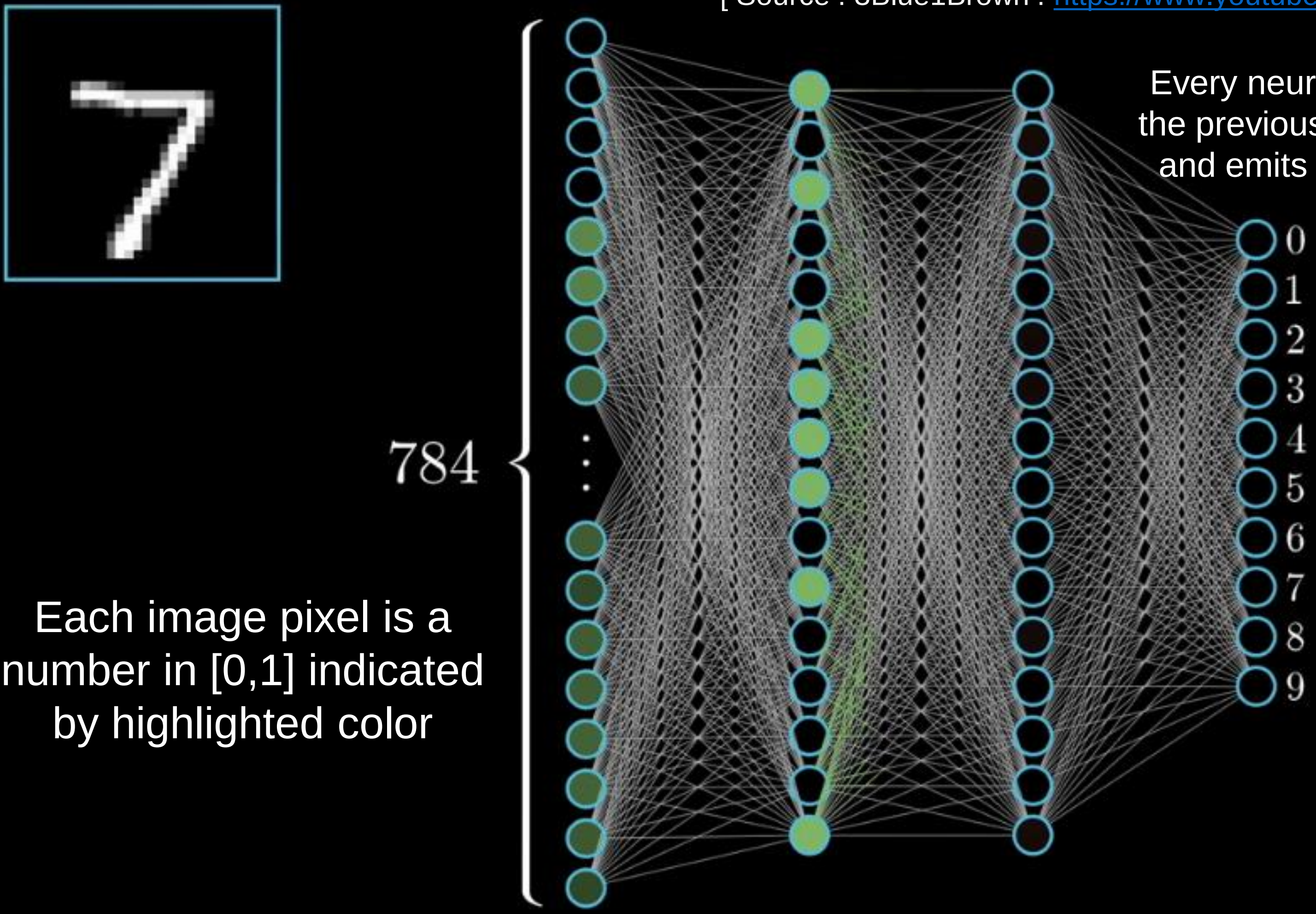
Classifying handwritten digits is the “Hello World” of NNs

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



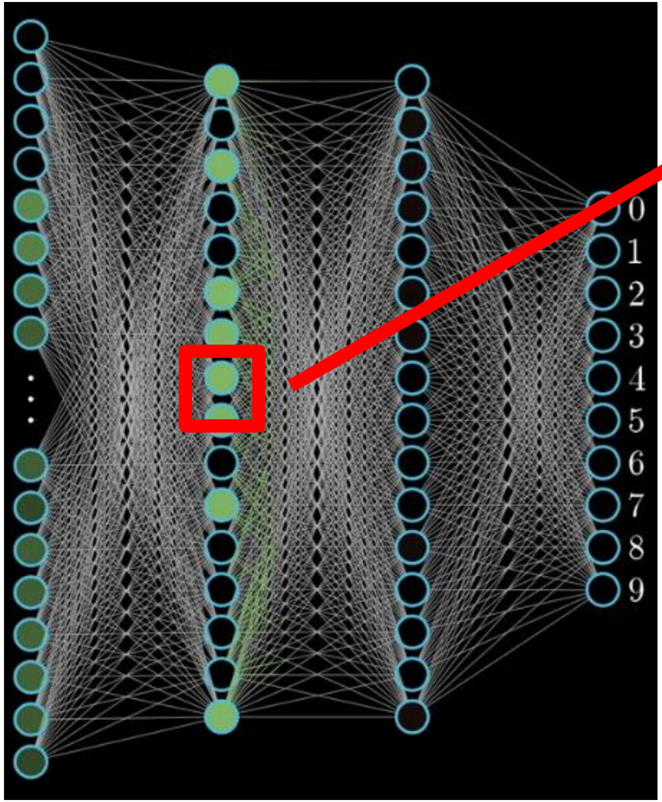
[Source : 3Blue1Brown : https://www.youtube.com/watch?v=aircAruvnKk]



Every neuron receives signal from the previous layer, processes them, and emits signal to the next layer

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…

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

$x_1, \ldots, x_n$: nodes at previous layer

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

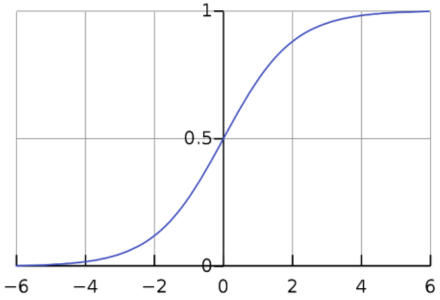
Nonlinear Activation functions

We call this an *activation function* and typically write it in vector form,

$$\sigma(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b) = \sigma(w^T x + b)$$

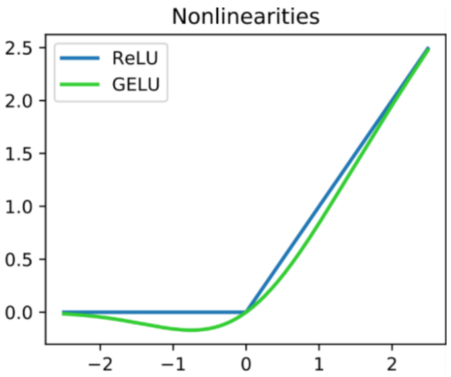
An early choice was the *logistic function*,

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

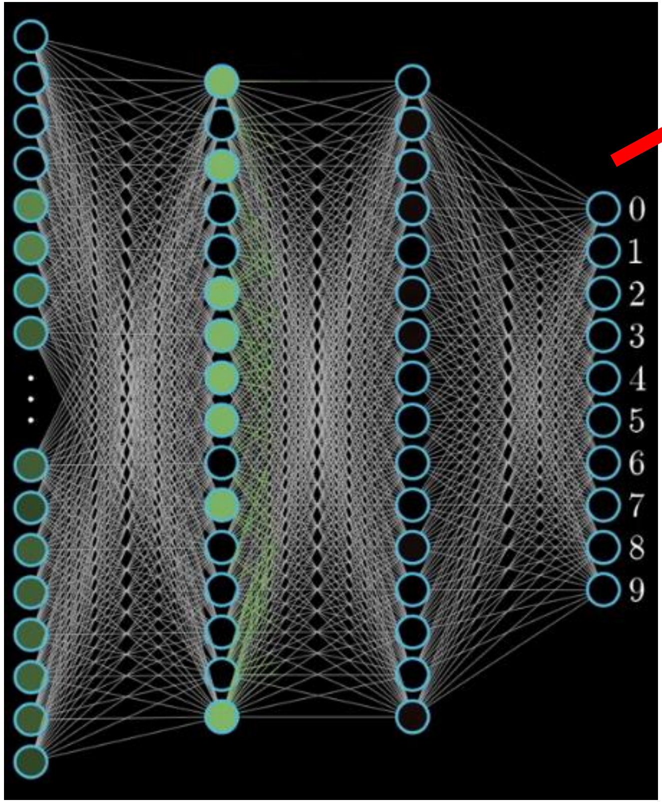


Later found to lead to slow learning and the *rectified linear unit (ReLU) become popular,*

$$\sigma(w^T x + b) = \max(0, w^T x + b)$$



Multilayer Perceptron



Final layer is typically a linear model… each output node is computed by

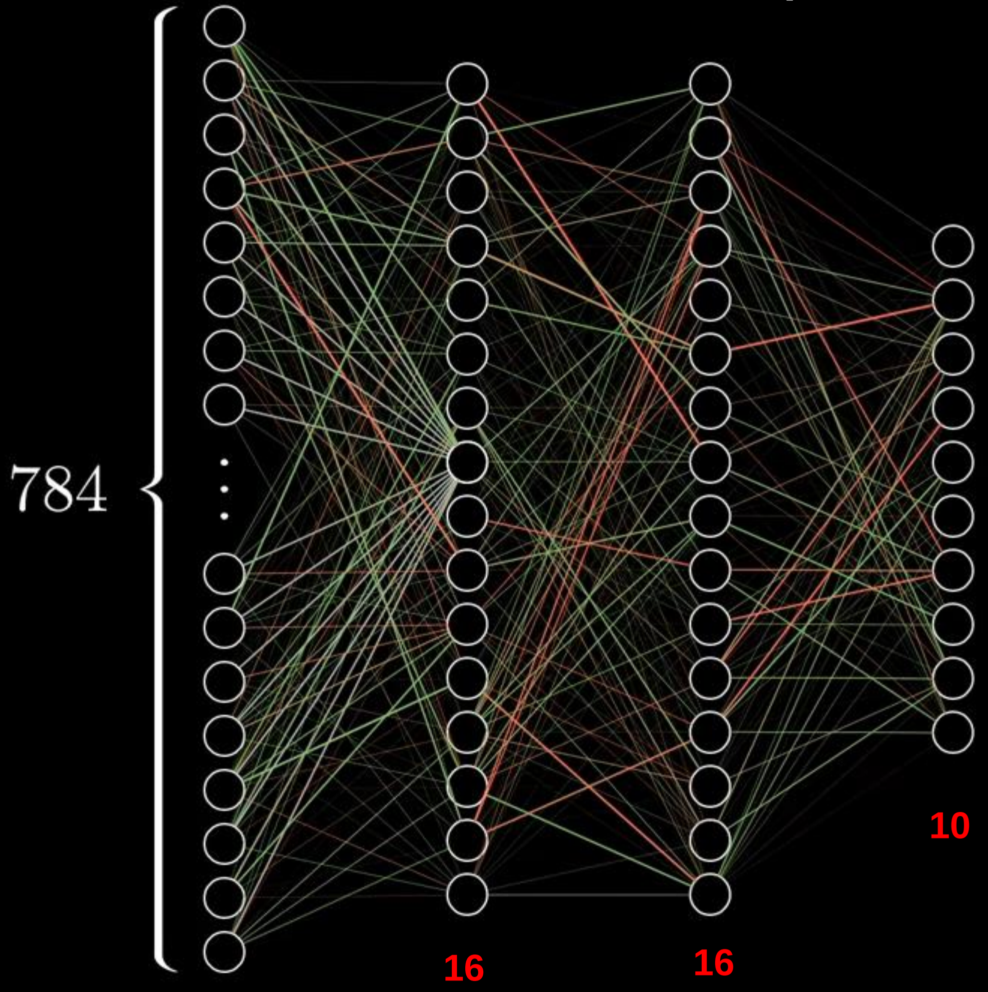
$$\sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$$

x: Vector of activations from previous layer

Recall that for binary logistic regression with 2 classes,

$$p(\text{Class} = 1 \mid x) \propto \sigma(w^T x + b)$$

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



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

weights

$$16 + 16 + 10$$

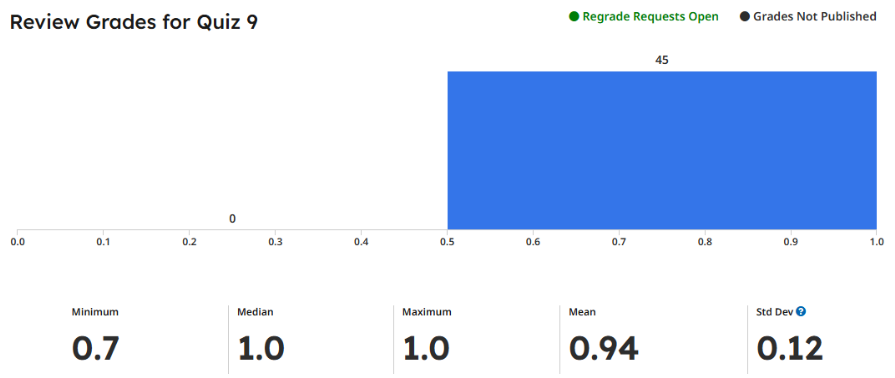
biases

13,002

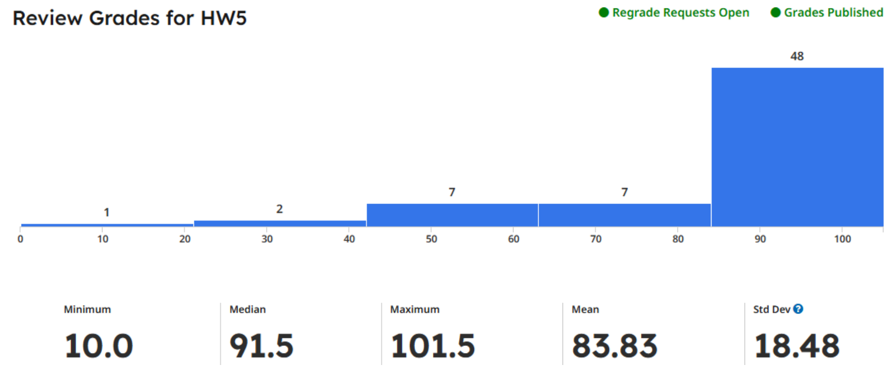
Each parameter has some impact on the output…need to tweak (learn) all parameters simultaneously to improve prediction accuracy

Announcements 4/16

- Quiz 9 graded



- HW5 graded



- Office hours participation instances can now be earned more than once a week

Quiz 10

Suppose we have two linear classifiers:

$$f(x) = 4x_1 + 3x_2 + 6, \qquad g(x) = 4x_1 + 3x_2$$

and a training set

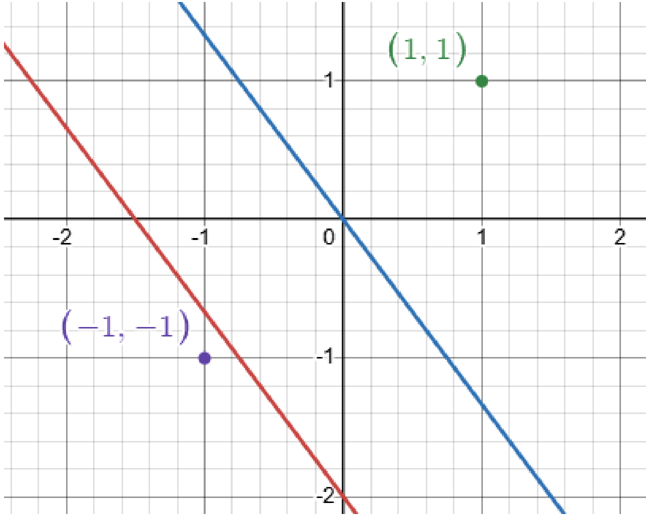
x	y
$(1,1)$	$+$
$(-1,-1)$	$-$

1. Visualize f, g and the training set in a 2D plane
2. What are the margins of f and g on these points?
3. Which of f, g has a smaller margin on the whole training set?

(Hint: $\sqrt{3^2 + 4^2} = 5$)

Quiz 10

- $f(x) = 4x_1 + 3x_2 + 6$,
- $g(x) = 4x_1 + 3x_2$



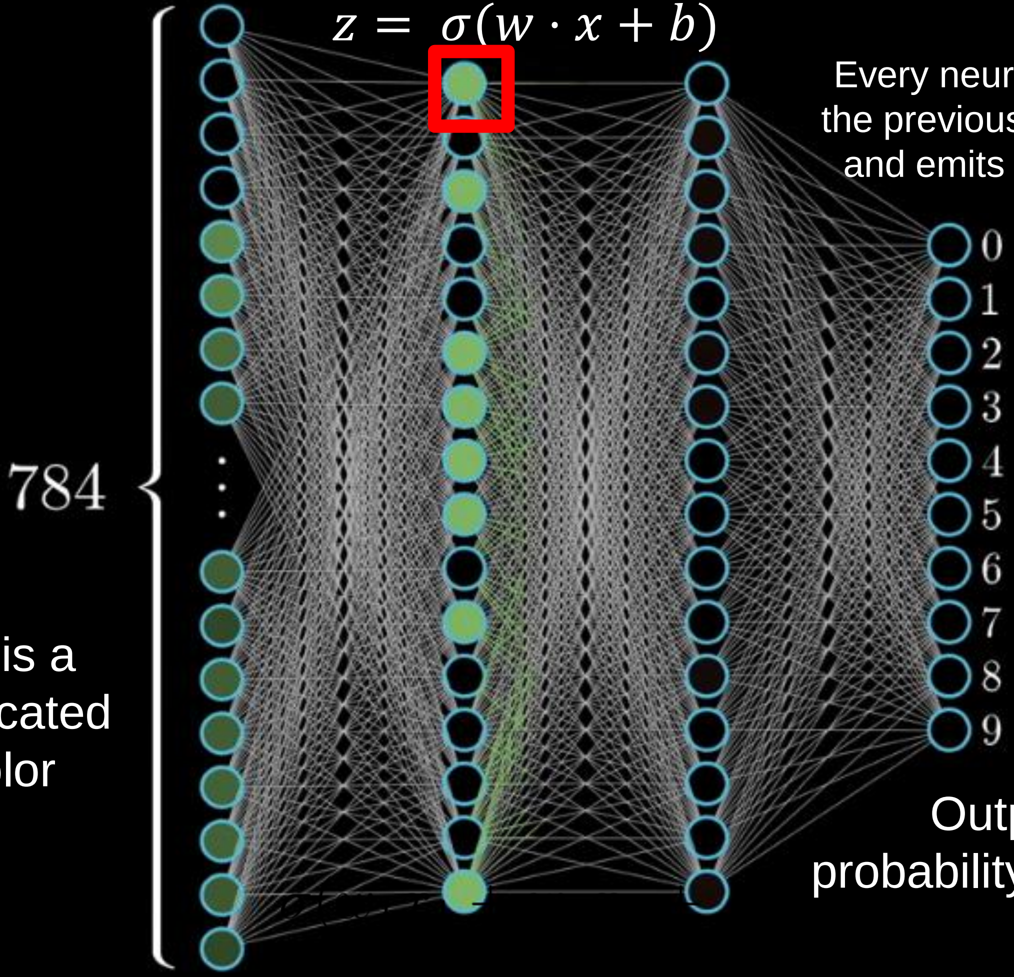
x	y
(1,1)	+
(−1,−1)	−

	f's margin	g's margin
	$+\frac{4+3+6}{5} = 2.6$	1.4
	$-\frac{-4-3+6}{5} = 0.2$	1.4

- f's margin on the dataset = min(2.6, 0.2) = 0.2 — Smaller
- g's margin on the dataset = min(1.4, 1.2) = 1.4 — Larger

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

$$z = \sigma(w \cdot x + b)$$



Every neuron receives signal from the previous layer, processes them, and emits signal to the next layer

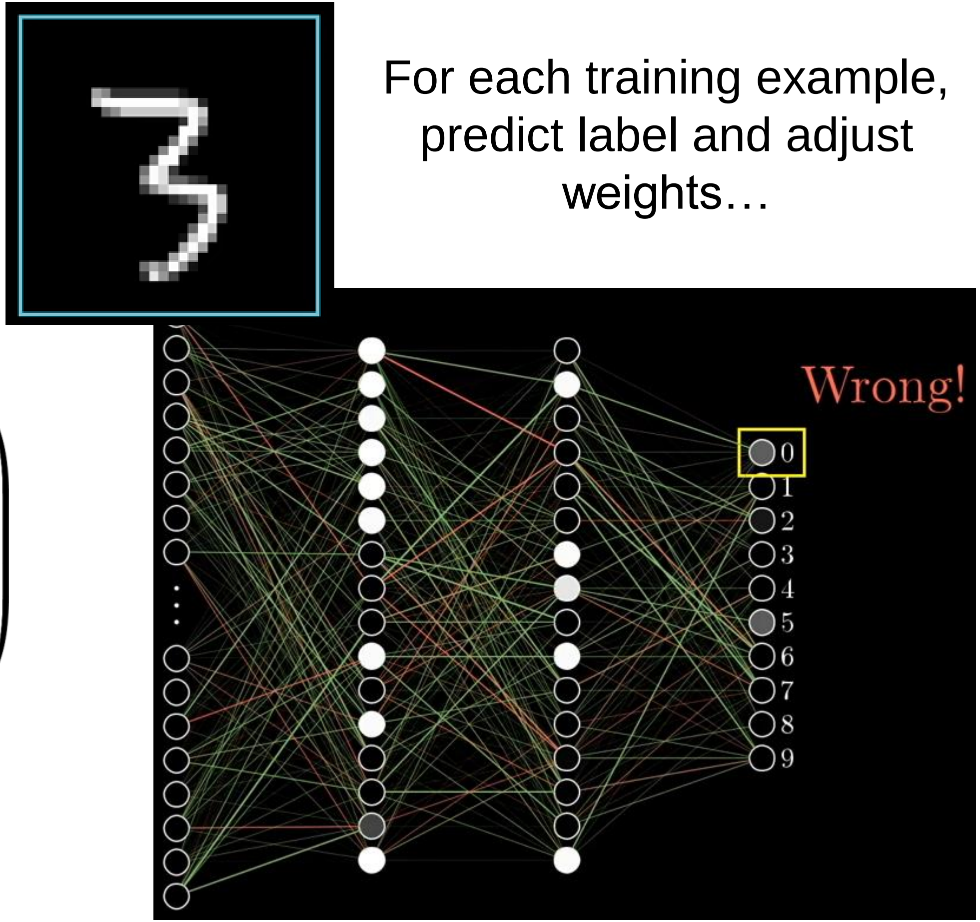
Each image pixel is a number in [0,1] indicated by highlighted color

Output layer:
probability of each class

Training Multilayer Perceptron

$$X^{\text{Train}} =$$

$$Y^{\text{Train}} = \begin{pmatrix} 0 & 4 & 1 & \dots & 3 \\ 5 & 3 & 6 & \dots & 4 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 7 & 4 & 6 & \dots & 5 \end{pmatrix}$$



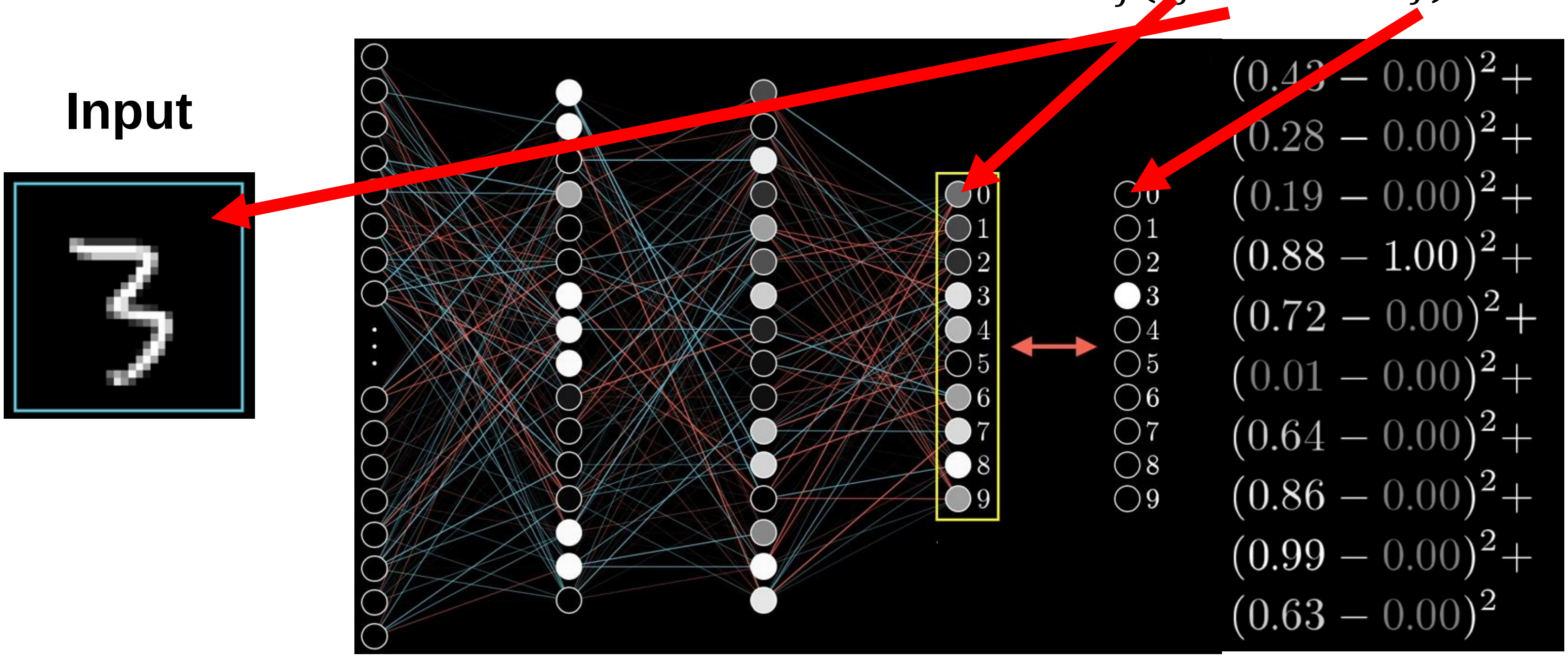
For each training example, predict label and adjust weights…

- How to score final layer output?
- How to adjust weights?

Training Multilayer Perceptron

One way to score (square loss): based on difference between final layer and one-hot vector of true class… $\ell(\theta) = \sum_j \left(f_j(x;\theta) - y_j\right)^2$

Input



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

Training Multilayer Perceptron: for classification

For classification, it is more popular to use:

- A softmax layer as final output

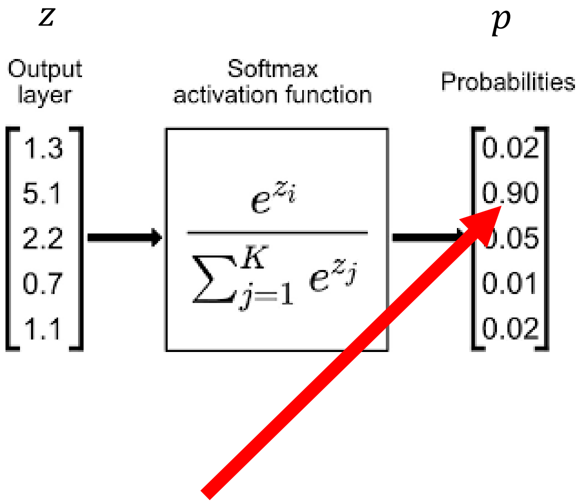
$$p_c = \frac{e^{z_c}}{\sum_{j=1}^{K} e^{z_j}}, c = 1, \ldots, K$$

gives probability estimate of each class given example $P(Y = c \mid X = x)$

- Cross-entropy (CE) loss for training

$$\ell(\vec{p}, y) = \log\left(\frac{1}{p_y}\right)$$

measures the neural network's "surprise" of seeing label y on this example



E.g. $y = 2$, $\ell(\vec{p}, y) = \log\frac{1}{0.90}$ -> small

E.g. $y = 4$, $\ell(\vec{p}, y) = \log\frac{1}{0.01}$ -> large

Training Multilayer Perceptron

Our loss function for i^th example is error in terms of weights / biases…

$$\ell_i(\theta), \qquad \theta := \underbrace{(w_1, \ldots, w_n, b_1, \ldots, b_n)}_{\text{13,002 Parameters in this network}}$$

13,002 Parameters
in this network

…minimize loss over all training data…

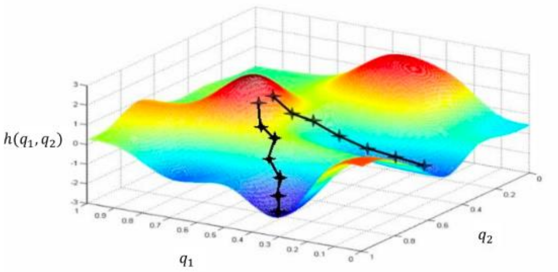
$$\min_{\theta} \mathcal{L}(\theta) = \frac{1}{m} \sum_{i=1}^{m} \ell_i(\theta)$$

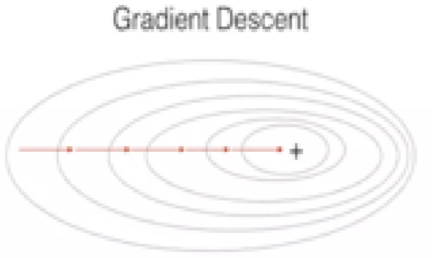
This is a super high-dimensional optimization (13,002 dimensions in this example)…how do we solve it?

Gradient descent: the go-to method for optimization

Gradient descent

- Gradient descent: Move in direction of greatest local improvement (greedily)

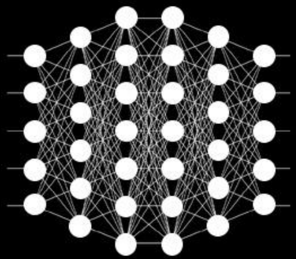




- “Knob turning”
  - ”knob” = weight of an edge
  - If a neuron increases the probability of an incorrect prediction, its knobs will be turned down.
  - If a neuron increases the probability of a correct prediction, its knobs will be turned up.

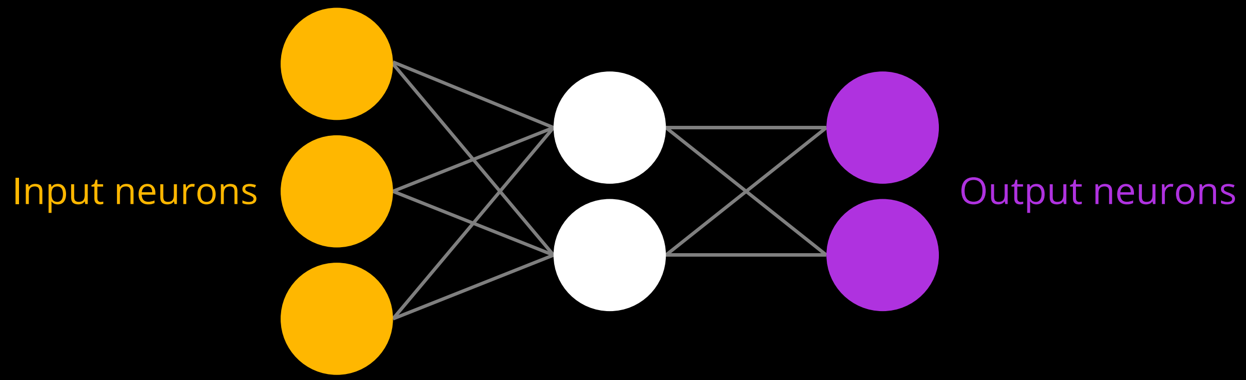
Deep learning, a field of machine learning

Learning algorithm
(backpropagation)

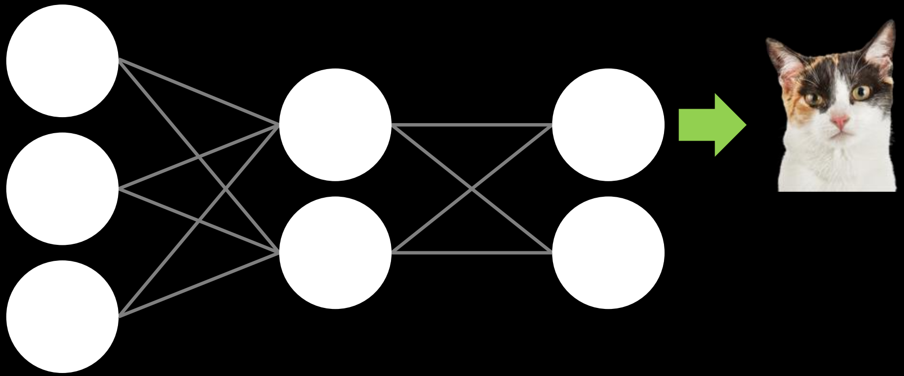


Dog 90%
Mop 10%

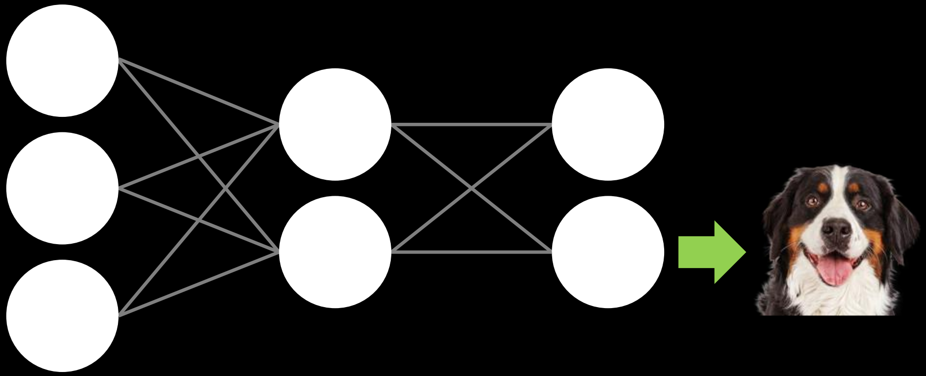
Deep learning with backpropagation



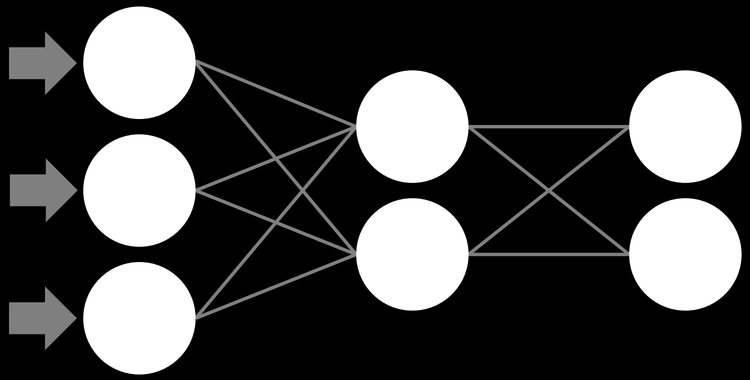
Deep learning with backpropagation



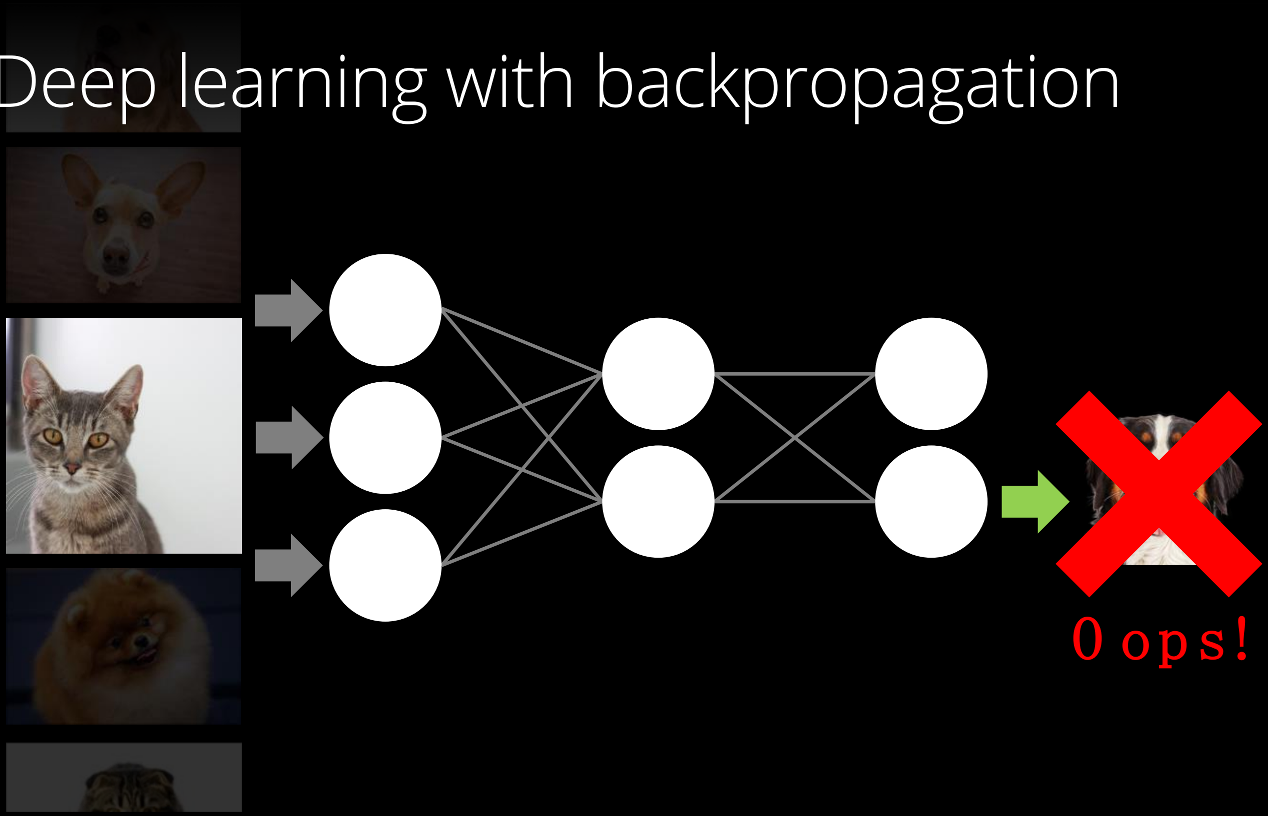
Deep learning with backpropagation



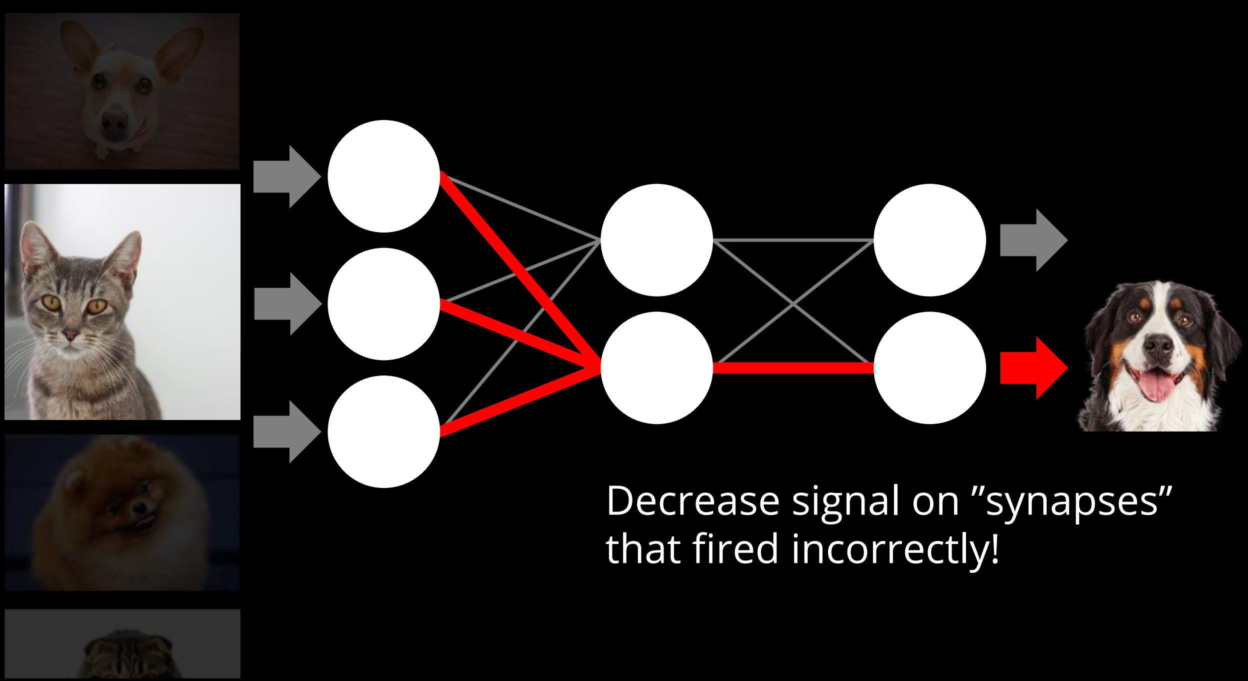
Deep learning with backpropagation



Deep learning with backpropagation

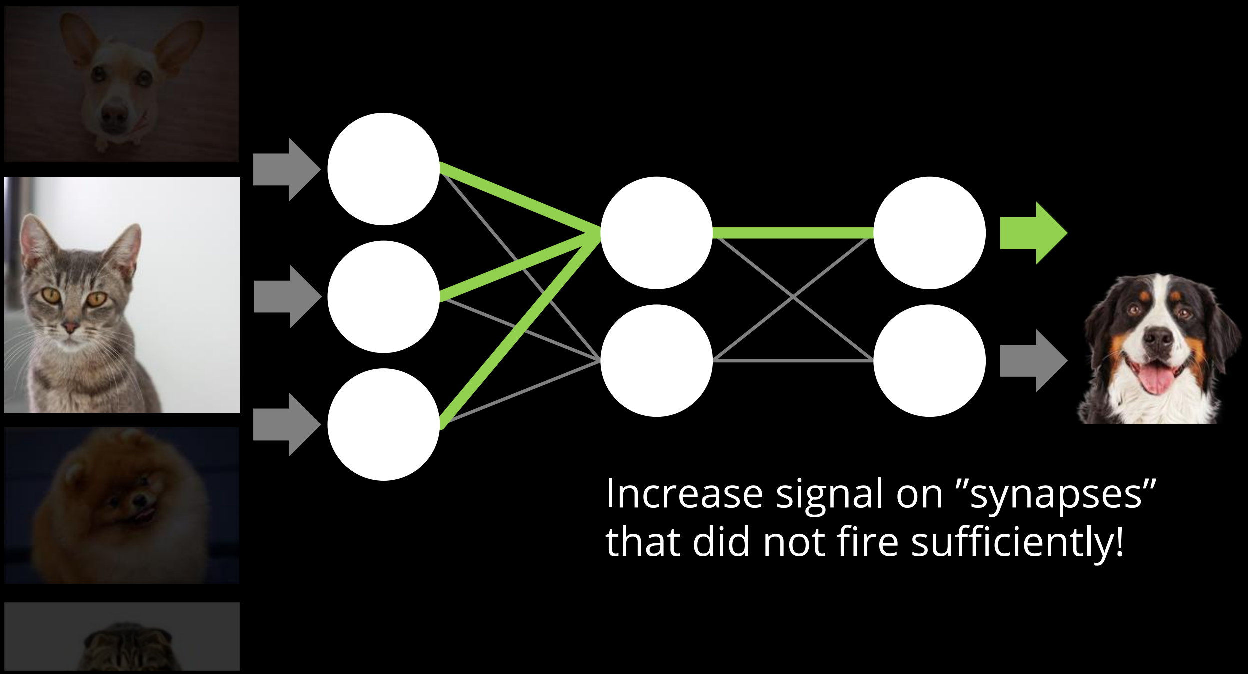


Deep learning with backpropagation



Decrease signal on "synapses" that fired incorrectly!

Deep learning with backpropagation

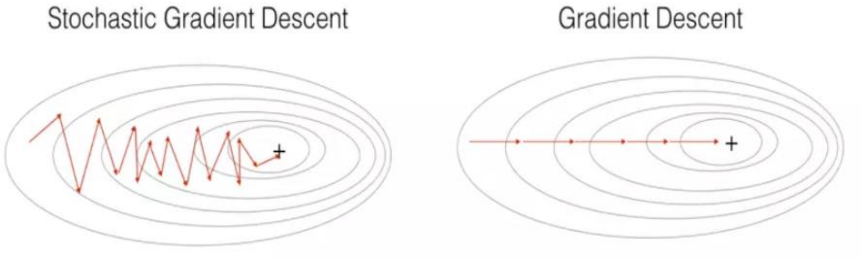


Increase signal on ”synapses” that did not fire sufficiently!

Algorithm 7: Stochastic gradient descent algorithm for the training of neural networks.

```
initialize parameters in Θ
while not converged do
    for each training example x_i in X do
        for each θ in Θ do
            θ = θ − α (d/dθ) C_i(Θ)
        end
    end
end
```

$$\theta = \theta - \alpha \frac{d}{d\theta} C_i(\Theta)$$



Algorithm 7: Stochastic gradient descent algorithm for the training of neural networks.

```
initialize parameters in Θ
while not converged do
    for each training example x_i in X do
        for each θ in Θ do
            θ = θ − α (d/dθ) C_i(Θ)
        end
    end
end
```

$$\theta = \theta - \alpha \frac{d}{d\theta} C_i(\Theta)$$

Collection of all weights and biases in the network

Algorithm 7: Stochastic gradient descent algorithm for the training of neural networks.

initialize parameters in Θ
while *not converged* **do**
  for *each training example* $\mathbf{x}_i$ **in X do**
    for *each* θ *in* Θ **do**
      $\theta = \theta - \alpha \frac{d}{d\theta} C_i(\Theta)$
    end
  end
end

One training example

Algorithm 7: Stochastic gradient descent algorithm for the training of neural networks.

```
initialize parameters in Θ
while not converged do
    for each training example x_i in X do
        for each θ in Θ do
            θ = θ − α (d/dθ) C_i(Θ)
        end
    end
end
```

$$\theta = \theta - \alpha \frac{d}{d\theta} C_i(\Theta)$$

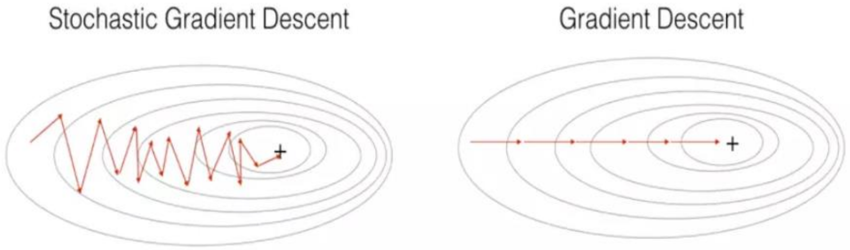
Partial derivative of the cost function C for each parameter (weight or bias) in the network

Algorithm 7: Stochastic gradient descent algorithm for the training of neural networks.

```
initialize parameters in Θ
while not converged do
    for each training example x_i in X do
        for each θ in Θ do
            θ = θ − α (d/θ) C_i(Θ)
        end
    end
end
```

$$\theta = \theta - \alpha \frac{d}{\theta} C_i(\Theta)$$

Learning rate, which is a hyper parameter



Neural network demo

- Tensorflow neural network playground

Visualizes:
- hidden neurons
- weights & biases
- learning curves (training & test losses vs number of iterations)

Let's try:
- editing the weights
- using no hidden layers
- using no hidden layers + basis functions
- using 1 hidden layer with 4 nodes
- playing with a harder dataset

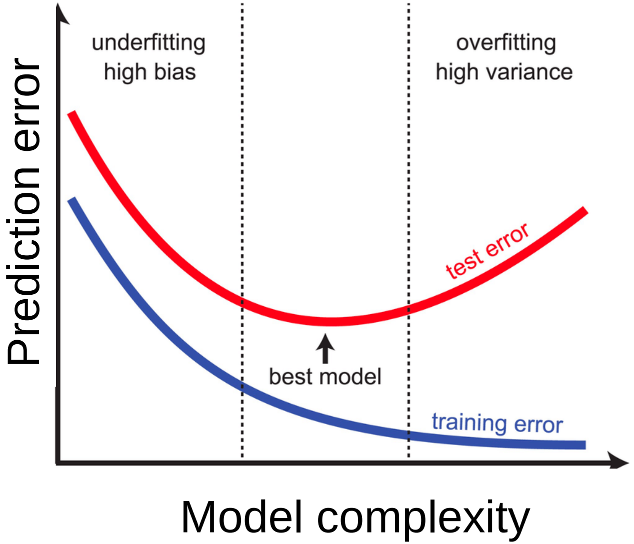
Regularization

With four parameters I can fit an elephant. With five I can make him wiggle his trunk. - John von Neumann

$$w = \arg\min_{w} \text{Cost}(w) + \alpha \cdot \text{Regularizer}(\text{Model})$$

Our example model has 13,002 parameters…that's a lot of elephants! Regularization is critical to avoid overfitting…

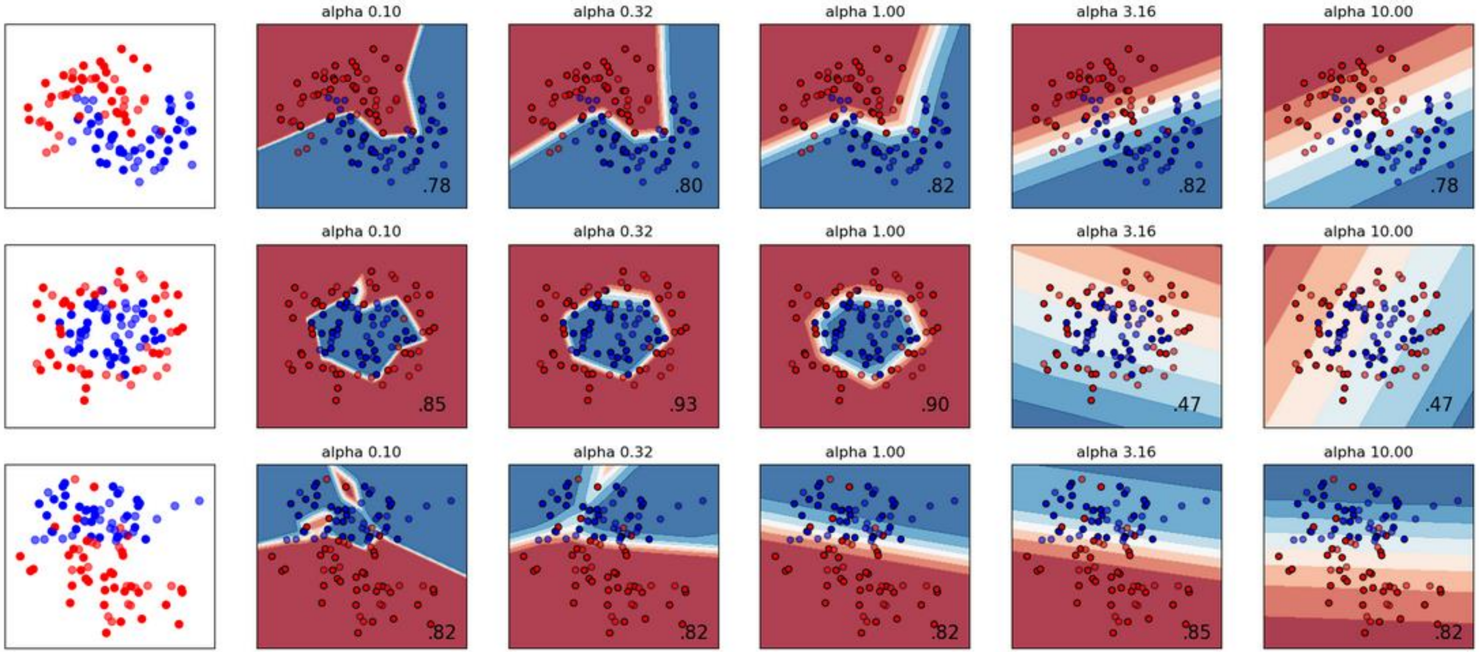
…numerous regularization schemes are used in training neural networks



Regularization: Weight Decay

In neural network terminology, adding an L2 penalty is called *weight decay*

$$w = \arg\min_{w} \text{Cost}(w) + \frac{\alpha}{2}\|w\|^2$$



`sklearn.neural_network`.MLPClassifier

hidden_layer_sizes : *tuple, length = n_layers - 2, default=(100,)*

The ith element represents the number of neurons in the ith hidden layer.

activation : *{'identity', 'logistic', 'tanh', 'relu'}, default='relu'*

Activation function for the hidden layer.

solver : *{'lbfgs', 'sgd', 'adam'}, default='adam'*

The solver for weight optimization.

alpha : *float, default=0.0001*

L2 penalty (regularization term) parameter.

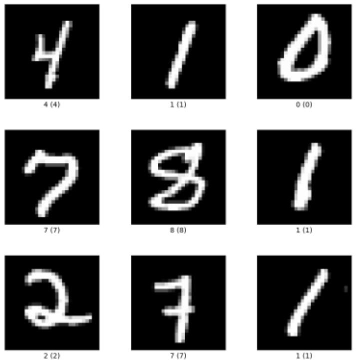
learning_rate : *{'constant', 'invscaling', 'adaptive'}, default='constant'*

Learning rate schedule for weight updates.

Scikit-Learn : Multilayer Perceptron

Fetch MNIST data from www.openml.org :

```
X, y = fetch_openml("mnist_784", version=1, return_X_y=True)
X = X / 255.0
```



Train test split (60k / 10k),

```
X_train, X_test = X[:60000], X[60000:]
y_train, y_test = y[:60000], y[60000:]
```

Create MLP classifier instance,

- Single hidden layer (50 nodes)
- Use stochastic gradient descent
- Maximum of 10 learning iterations
- Small L2 regularization alpha=1e-4

```
mlp = MLPClassifier(
    hidden_layer_sizes=(50,),
    max_iter=10,
    alpha=1e-4,
    solver="sgd",
    verbose=10,
    random_state=1,
    learning_rate_init=0.1,
)
```

Scikit-Learn : Multilayer Perceptron

Fit the MLP and print stuff…

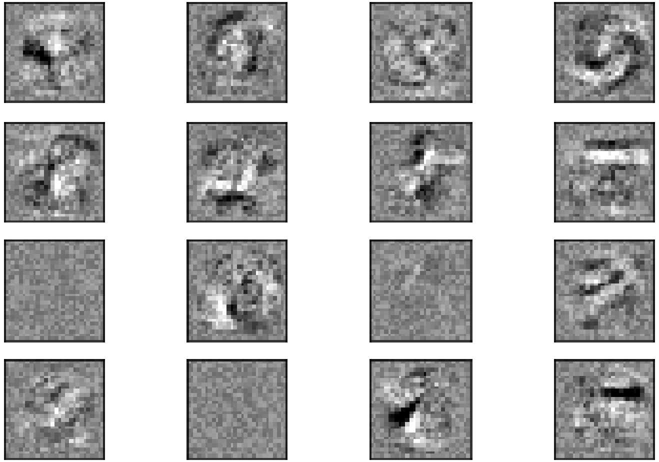
```
mlp.fit(X_train, y_train)
```

```
print("Training set score: %f" % mlp.score(X_train, y_train))
print("Test set score: %f" % mlp.score(X_test, y_test))
```

```
Iteration 1, loss = 0.32009978
Iteration 2, loss = 0.15347534
Iteration 3, loss = 0.11544755
Iteration 4, loss = 0.09279764
Iteration 5, loss = 0.07889367
Iteration 6, loss = 0.07170497
Iteration 7, loss = 0.06282111
Iteration 8, loss = 0.05530788
Iteration 9, loss = 0.04960484
Iteration 10, loss = 0.04645355
Training set score: 0.986800
Test set score: 0.970000
```

Visualize the weights for each node…

```
vmin, vmax = mlp.coefs_[0].min(), mlp.coefs_[0].max()
for coef, ax in zip(mlp.coefs_[0].T, axes.ravel()):
    ax.matshow(coef.reshape(28, 28), cmap=plt.cm.gray,
               vmin=0.5 * vmin, vmax=0.5 * vmax)
    ax.set_xticks(())
    ax.set_yticks(())
```



…magnitude of weights indicates which input features are important in prediction

More Advanced Topics

Many other NN architectures exist beyond MLP

- **Convolutional NN (CNN)** For image processing / computer vision.
- **Recurrent NN (RNN)** For sequence data (e.g. acoustic signals, video, etc.), long short-term memory (LSTM) is popular
- **Generative Adversarial Nets (GANs)** For generating creepy deepfakes
- **Transformers** For generating text (e.g. ChatGPT)

Many open areas being researched

- More reliable uncertainty estimates
- Robustness to input perturbations
- Interpretability
- Better scalability

Resources

There are **tons** of excellent resources for learning about neural networks online…here are two quick ones:

3Blue1Brown Youtube channel has a nice four-part intro:

https://www.youtube.com/watch?v=aircAruvnKk

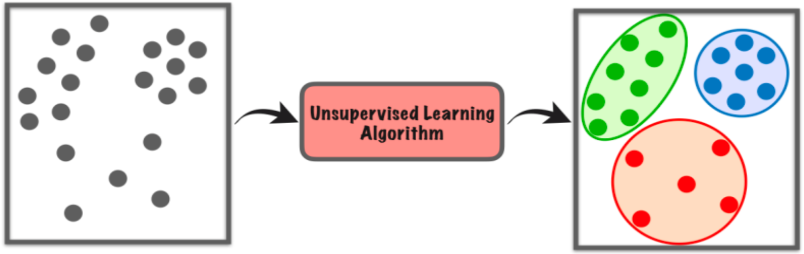
Free book by Michael Nielson uses MNIST example in Python:

http://neuralnetworksanddeeplearning.com/

Unsupervised learning: clustering

Unsupervised learning

Training data only contains inputs x, and does not have labels y



Goal: uncovering *structure* underlying the data

Understanding $p(x)$ (generative) instead of $p(y \mid x)$ (discriminative)

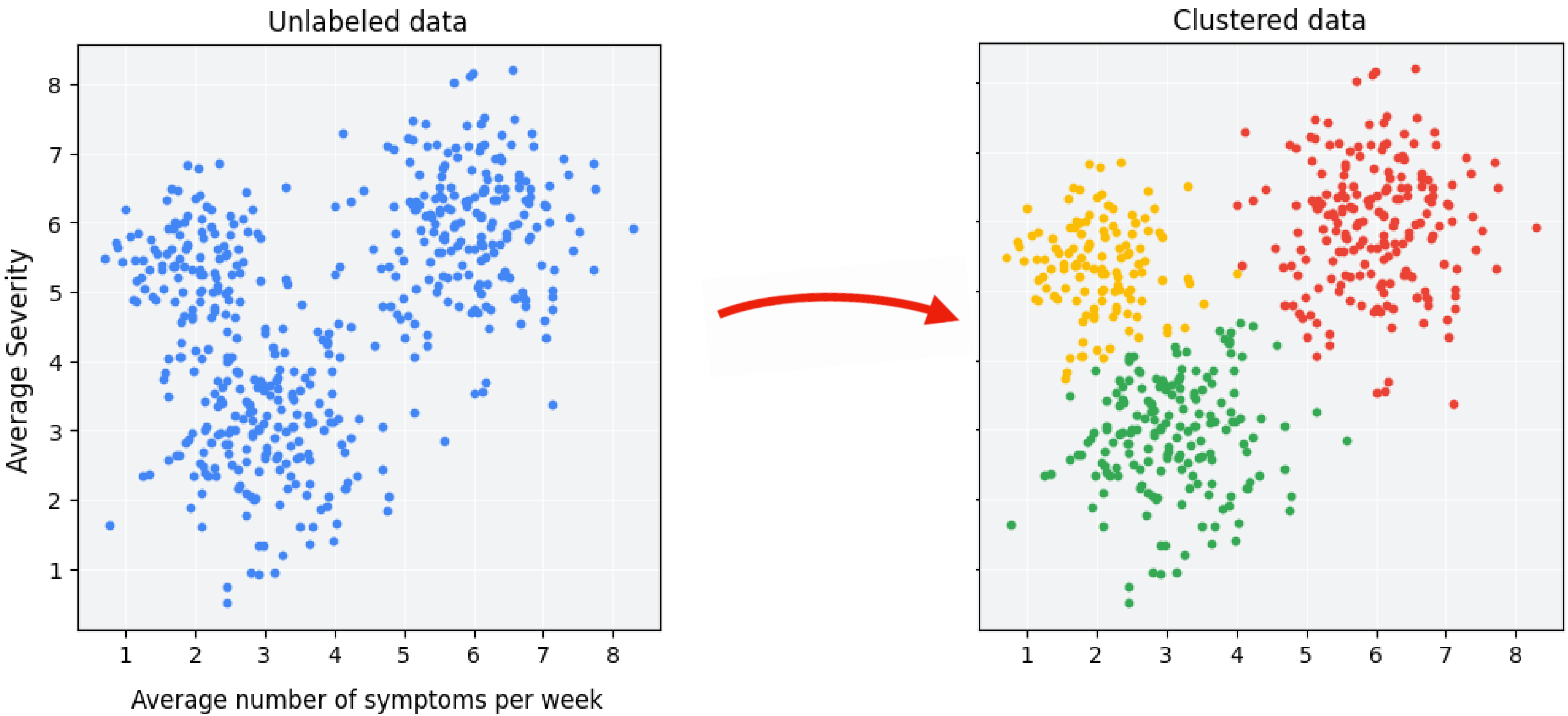
Two useful subproblems:

Clustering: uncovering hidden “classes” in data

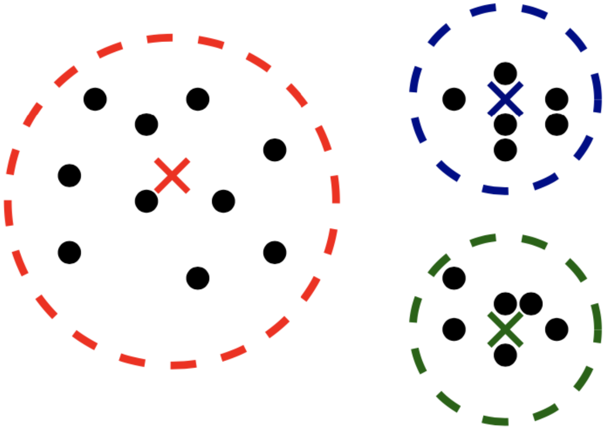
Component analysis: finding meaningful projections of data

Motivation of clustering: patient study

- Goal: assign customized treatments to patients



Clustering



Input: k: the number of clusters

dataset: $S = \{x_1, \ldots, x_n\}$

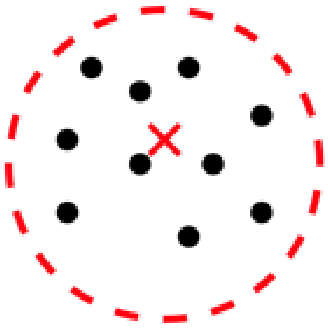
Output:

- clusters $\{G_i\}_{i=1}^{k}$ whose disjoint union is S
- we also often obtain 'centroids' – centers of each cluster

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

Centroid of a point set

A centroid c of point set $S = \{z_1, \ldots, z_n\}$ should be close to all points in that set

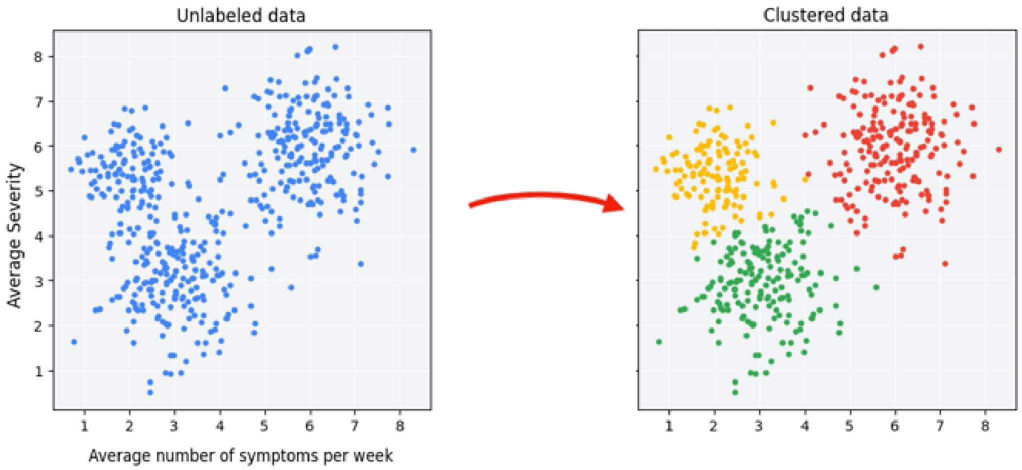


A reasonable definition: $c = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} \|z_i - w\|^2$

- When $d = 1$: $c = \bar{z} = \frac{1}{n}\sum_{i=1}^{n} z_i$ (*)
- Fact: (*) is still true for general d

Recap 4/21

- Clustering:
  finding hidden classes using unlabeled data

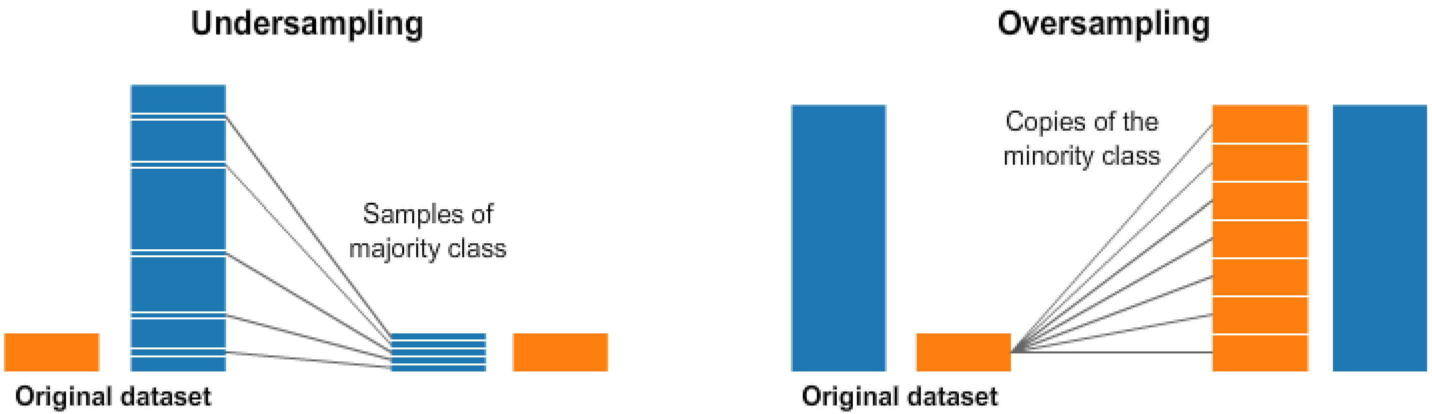


We will likely have a quiz next Monday (4/28)

Planning to release HW7 today

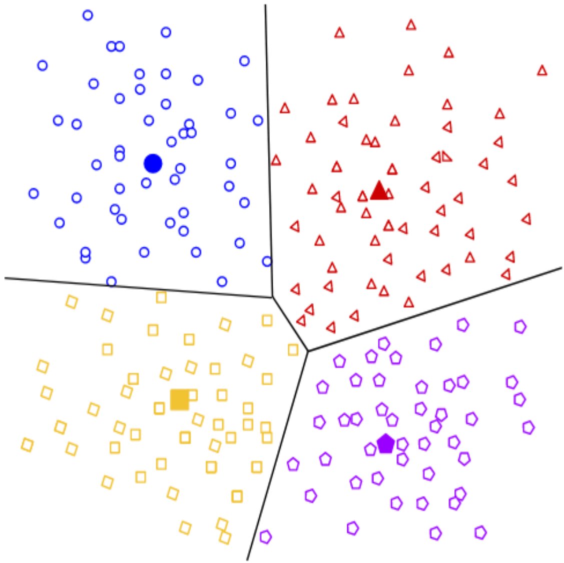
Imbalanced classification

- In imbalanced classification, training using original data may result in blind classifier that always predict majority class
- Ways to mitigate: re-balancing the datasets



- See Piazza more additional notes

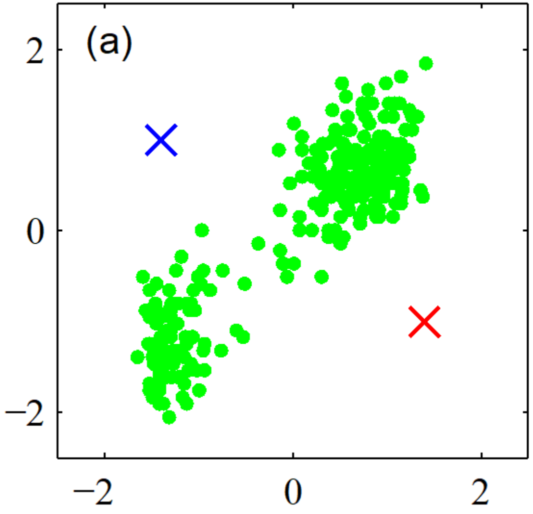
K-means clustering algorithm [Lloyd'82]



- Initialize Cluster Centroids

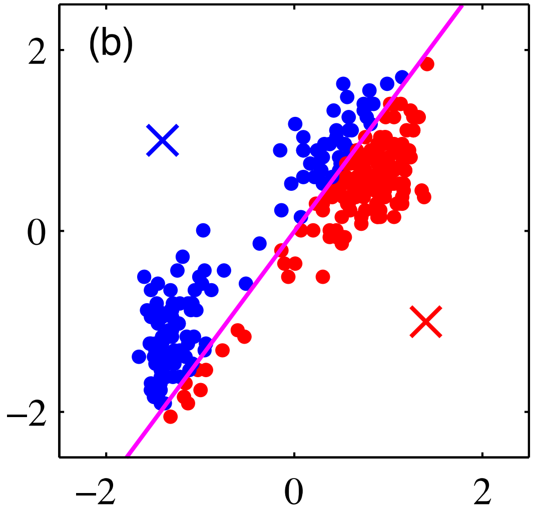
- Until Convergence:
  - **Cluster Assignment:** for each point, cluster with the nearest centroid
  - **Recompute Centroid:** for each cluster, recompute its centroid to be the cluster mean

Initialization

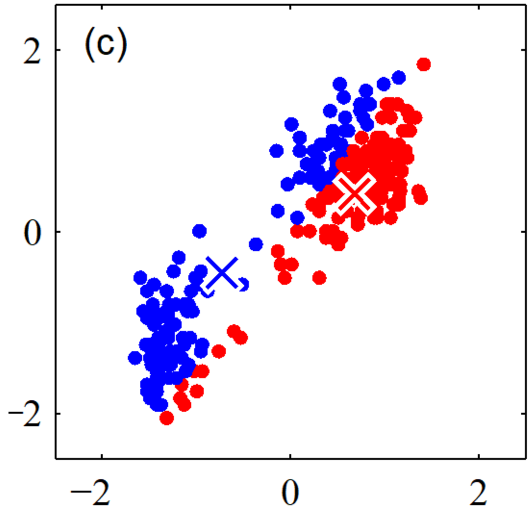


Arbitrary/random initialization of c_1 and c_2

Iteration 1

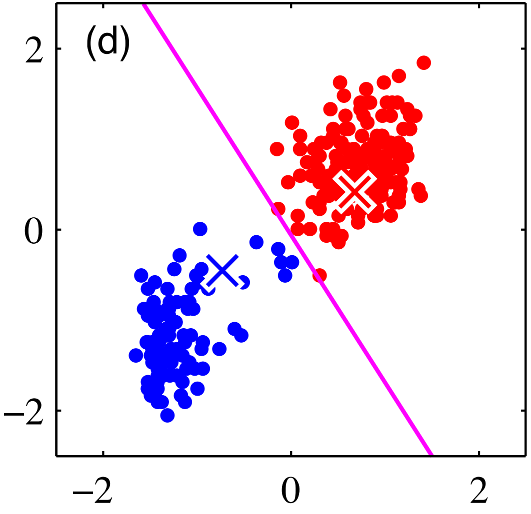


(A) update the cluster assignments

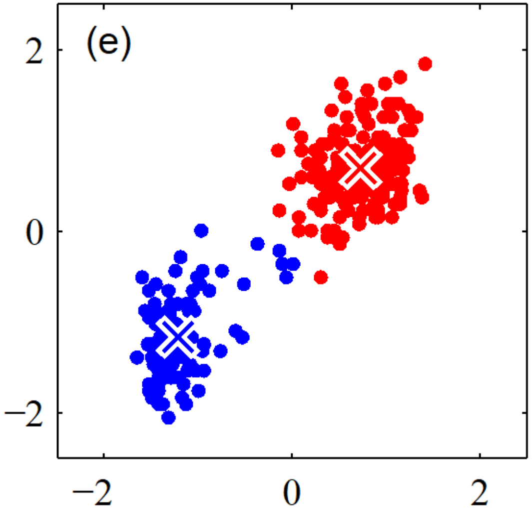


(B) Update the centroids c_1, c_2

Iteration 2

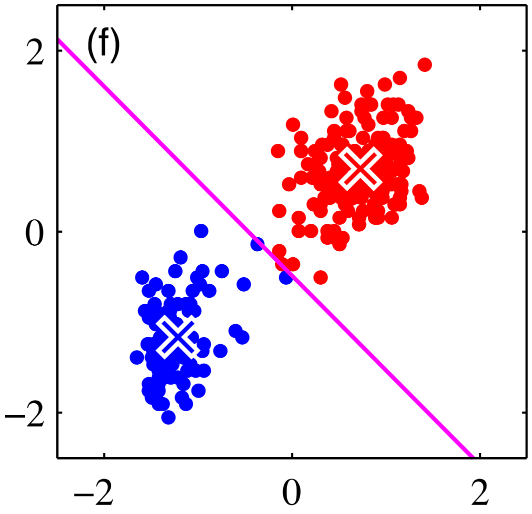


(A) update the cluster assignments

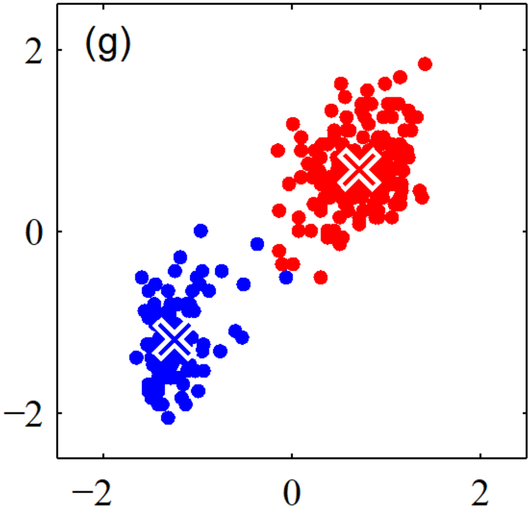


(B) Update the centroids c_1, c_2

Iteration 3

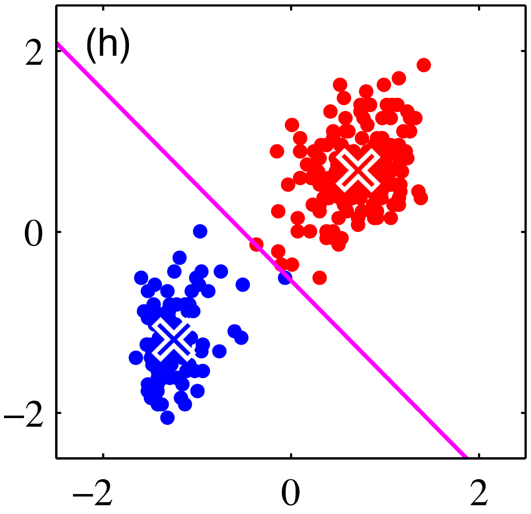


(A) update the cluster assignments

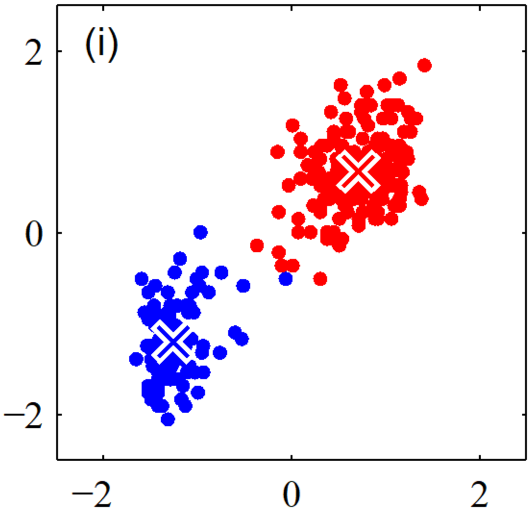


(B) Update the centroids c_1, c_2

Iteration 4

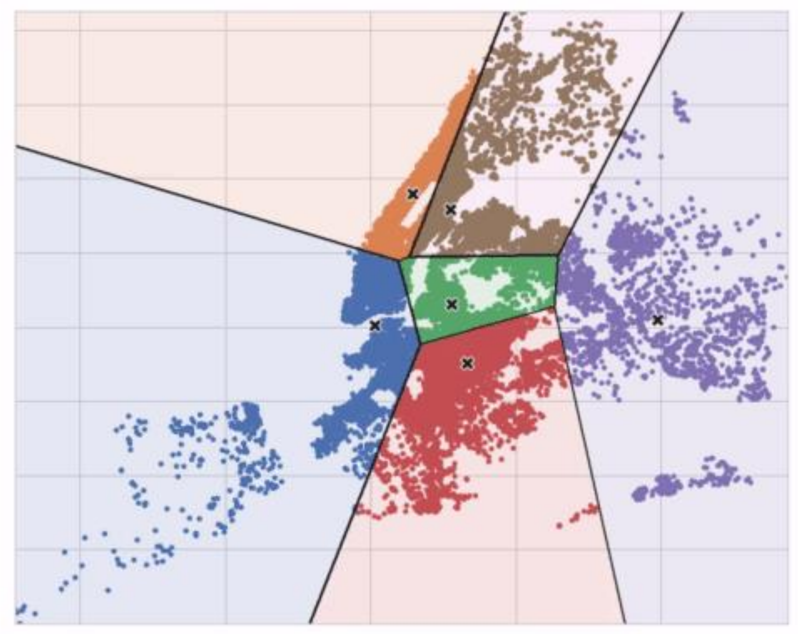


(A) update the cluster assignments



(B) Update the centroids c_1, c_2

Iterating until Convergence



Animation from Kaggle

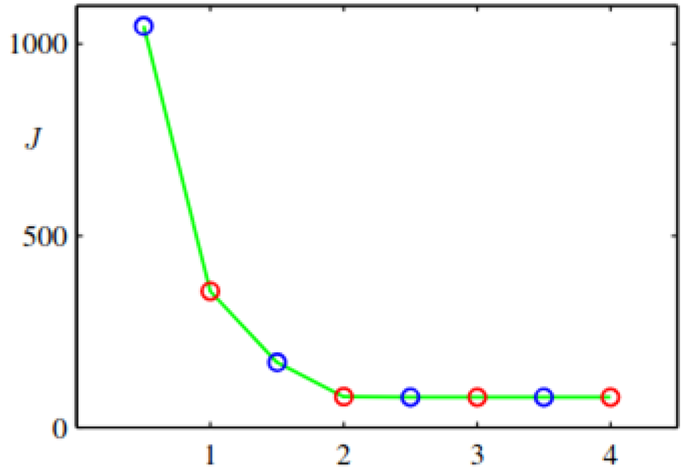
Promise of Convergence

Plot of the cost function J after each cluster assignment step and recompute centroid step

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

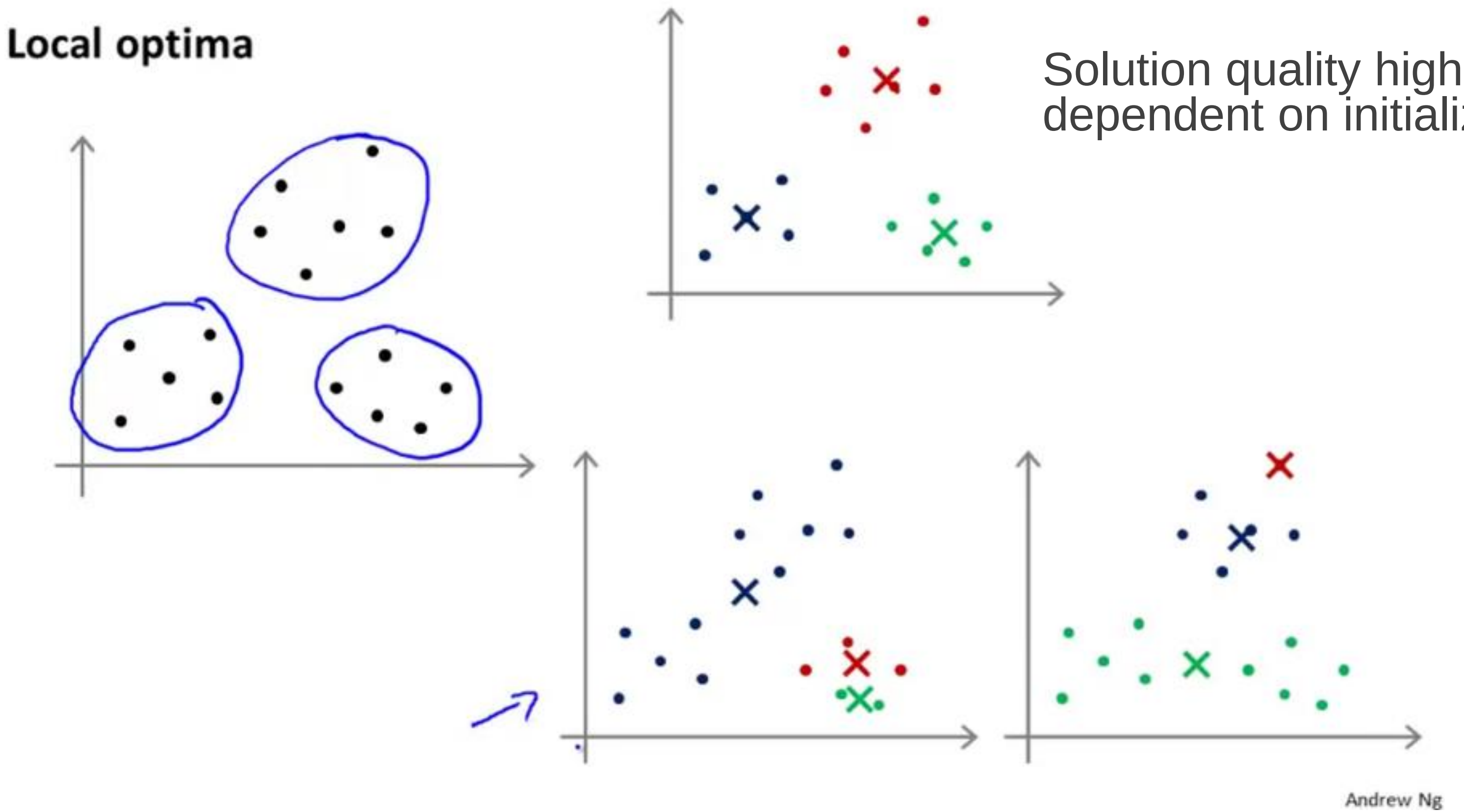
=1 if x_n is assigned to cluster k
=0 otherwise

Location of centroid k



But may converge to a local rather than global minimum of J.

Convergence to local optima

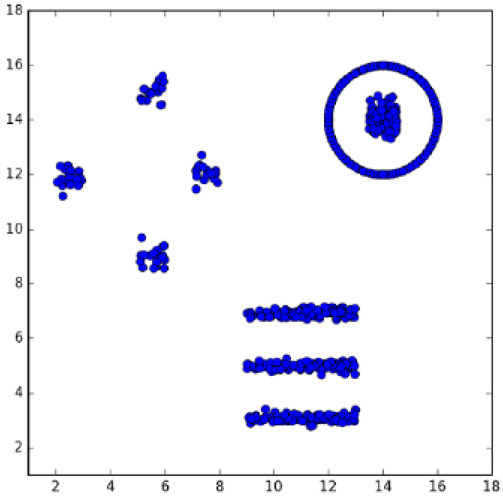


Solution quality highly dependent on initialization!

Image from Andrew NG Coursera Machine Learning Course

Clustering: concluding remarks

Definition of clusters may be subjective and application-dependent



How many clusters are there?

Hierarchical clustering

- multiresolution data analysis