CS6421: Deep Neural Networks

Gregory Provan

Spring 2020

Lecture 2: Classification and Learning

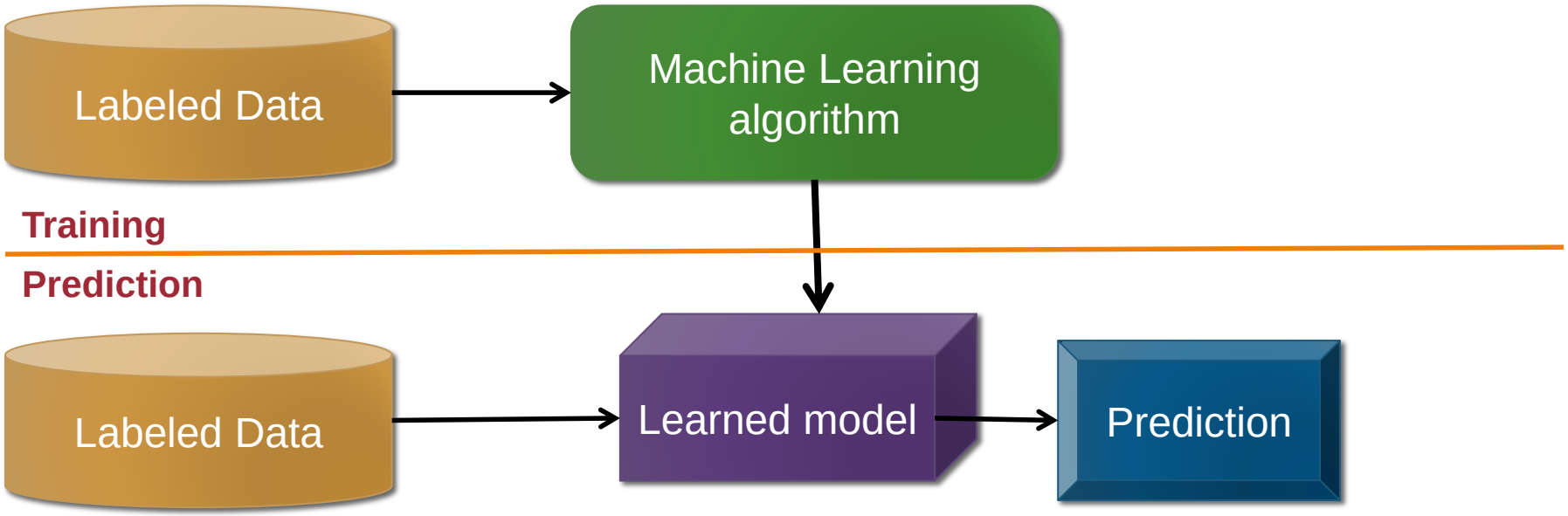
Based on notes from John Canny, Ismini Lourentzou

Overview

- Machine Learning Basics
- Key tasks of Deep Learning network
  - Classification
  - Learning
- Classification in DN
- Learning in DN
  - Backpropagation

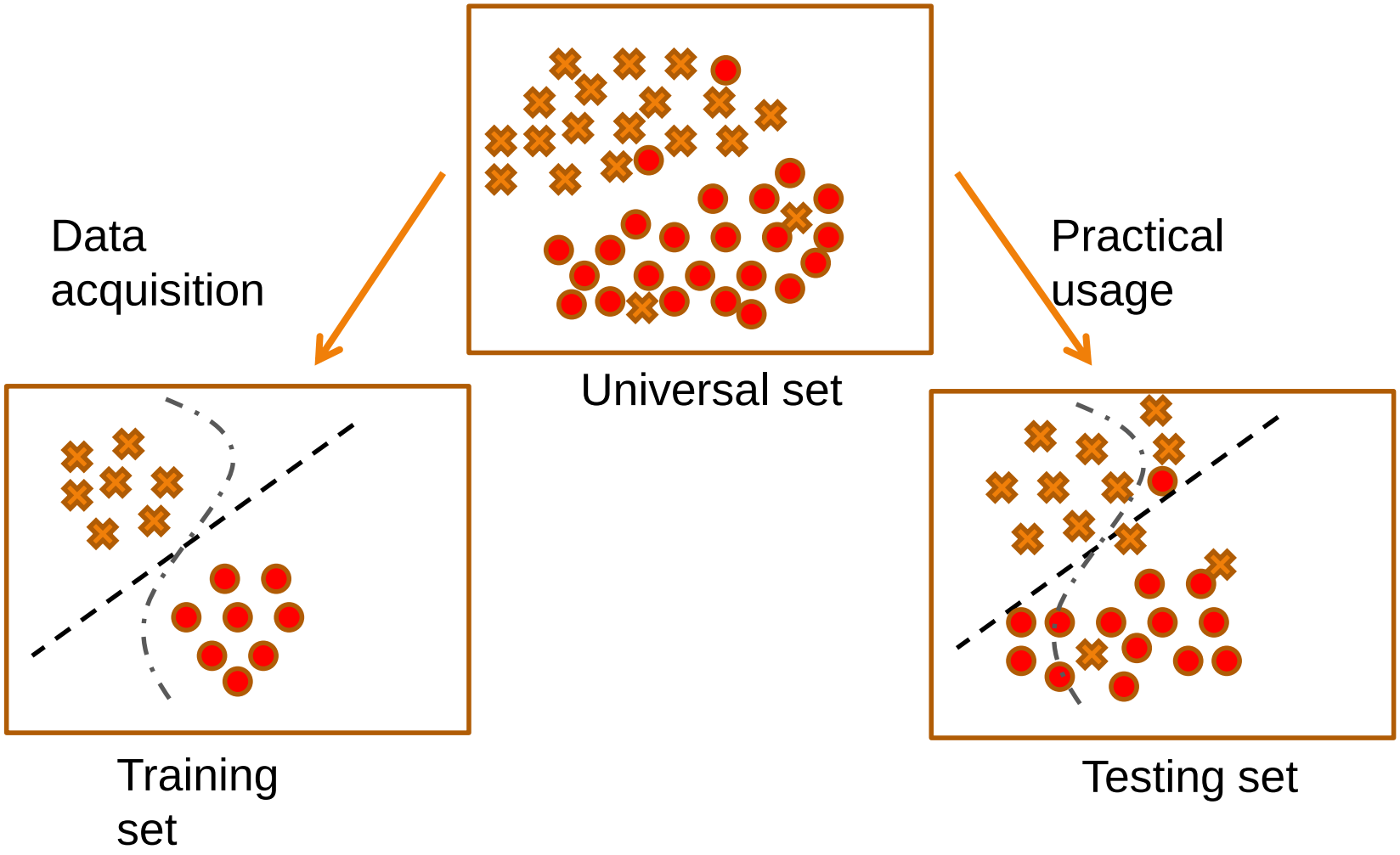
Machine Learning Basics

Machine learning is a field of computer science that gives computers the ability to **learn without being explicitly programmed**



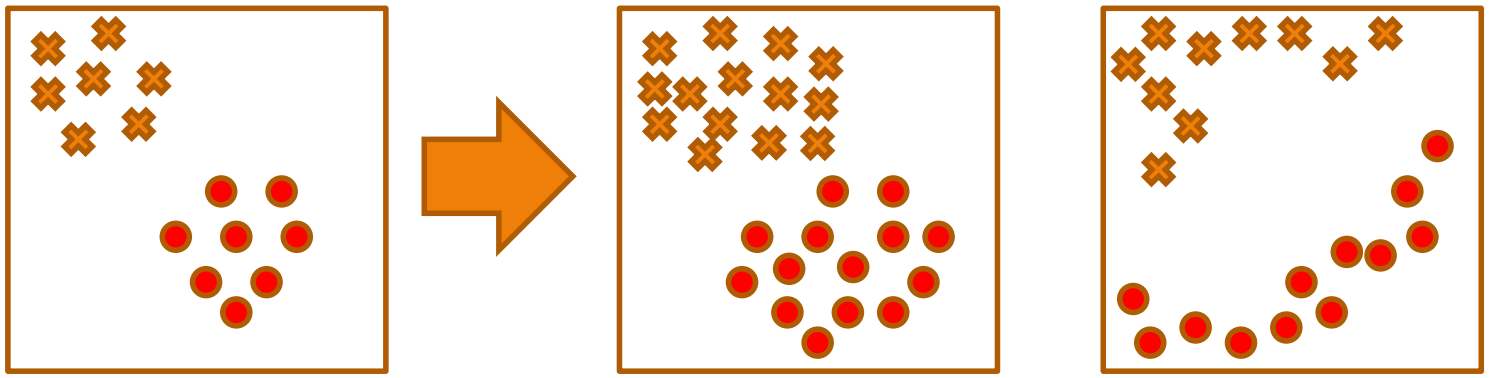
Methods that can learn from and make predictions on data

Training and testing: Supervised Case



Assumptions

- Training is the process of making the system able to learn.
- No free lunch rule:
  - Training set and testing set come from the same distribution
  - Adequacy of functional approximation



Performance

- There are several factors affecting the performance:
  - **Types of training** provided
  - The form and extent of any initial **background knowledge**
  - The **type of feedback** provided
  - The **learning algorithms** used
- Two important factors:
  - Modeling
  - Optimization

Algorithms

- The success of machine learning system also depends on the algorithms.
- The algorithms control the search to find and build the knowledge structures.
- The learning algorithms should extract useful information from training examples.

Types of Learning

Supervised: Learning labels ***y*** with a **labeled training** set ***x***
Example: email *classification* with already labeled emails

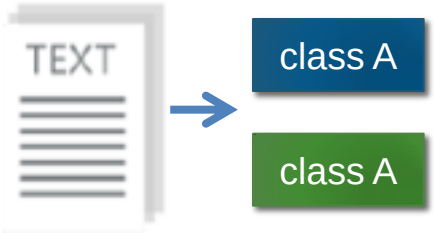
$y=f(x)$

Unsupervised: Discover **patterns** in **unlabeled** data ***x***
Example: *cluster* similar documents based on text

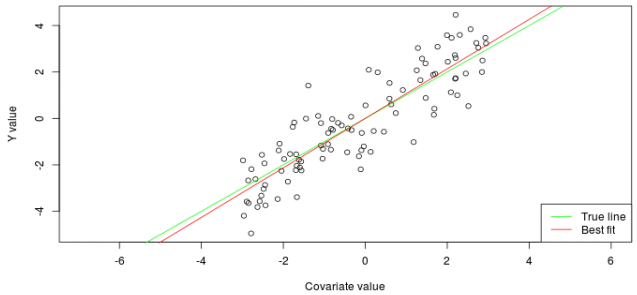
$f(x)$

Reinforcement learning: learn to **act** based on **feedback/reward *z***
Example: learn to play Go, reward: *win or lose*

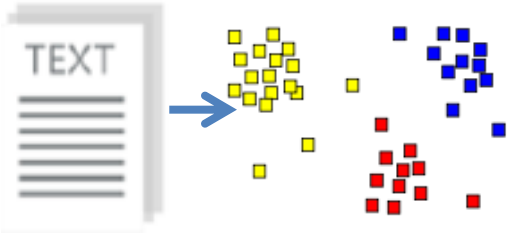
$y=f(x)$
z



Classification



Regression



Clustering

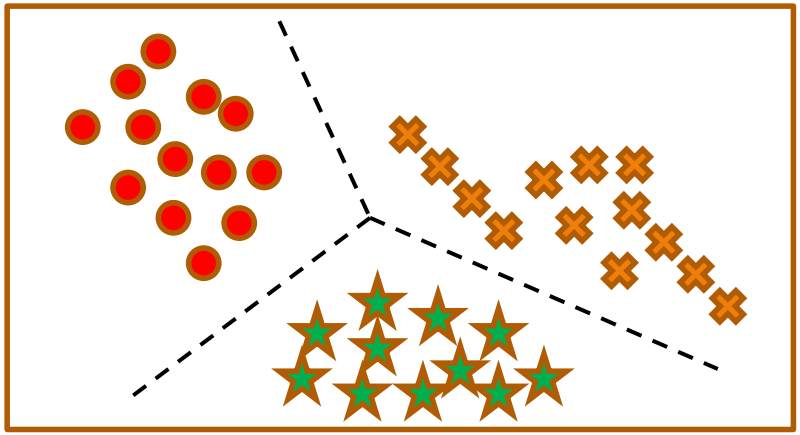
Anomaly Detection
Sequence labeling
…

http://mbjoseph.github.io/2013/11/27/measure.html

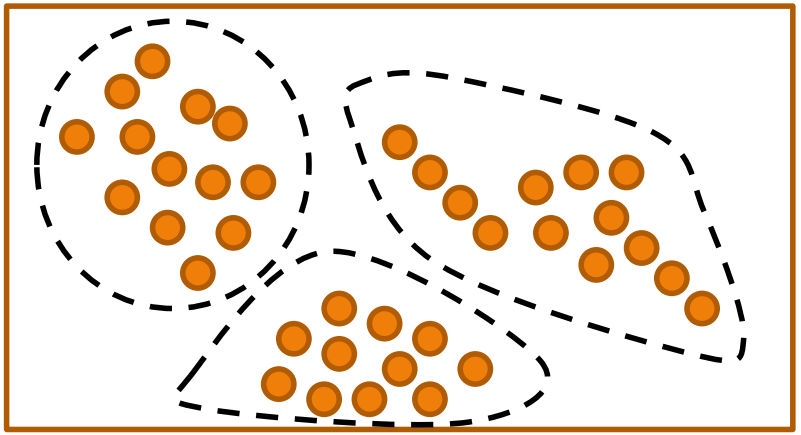
Tasks of Different Approaches

- **Supervised learning** ($\{x_n \in R^d, y_n \in R\}_{n=1}^N$)
  - Prediction
  - Classification (discrete labels), Regression (real values)
- **Unsupervised learning** ($\{x_n \in R^d\}_{n=1}^N$)
  - Clustering
  - Probability distribution estimation
  - Finding association (in features)
  - Dimension reduction
- **Semi-supervised learning**
- **Reinforcement learning**
  - Decision making (robot, chess machine)

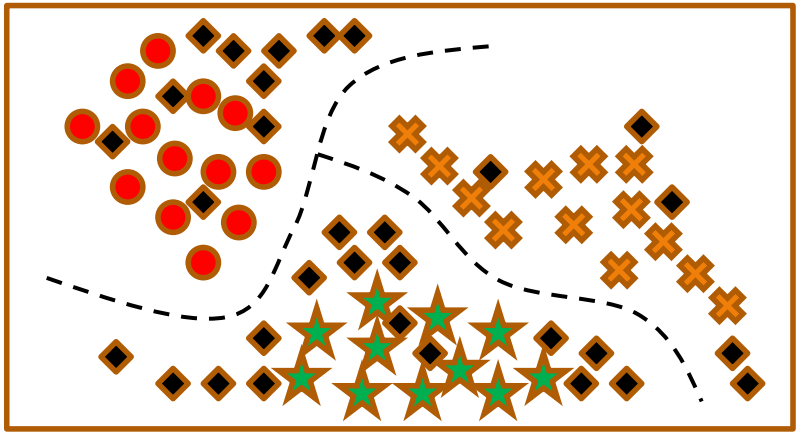
Visualisation of Approaches



Supervised learning



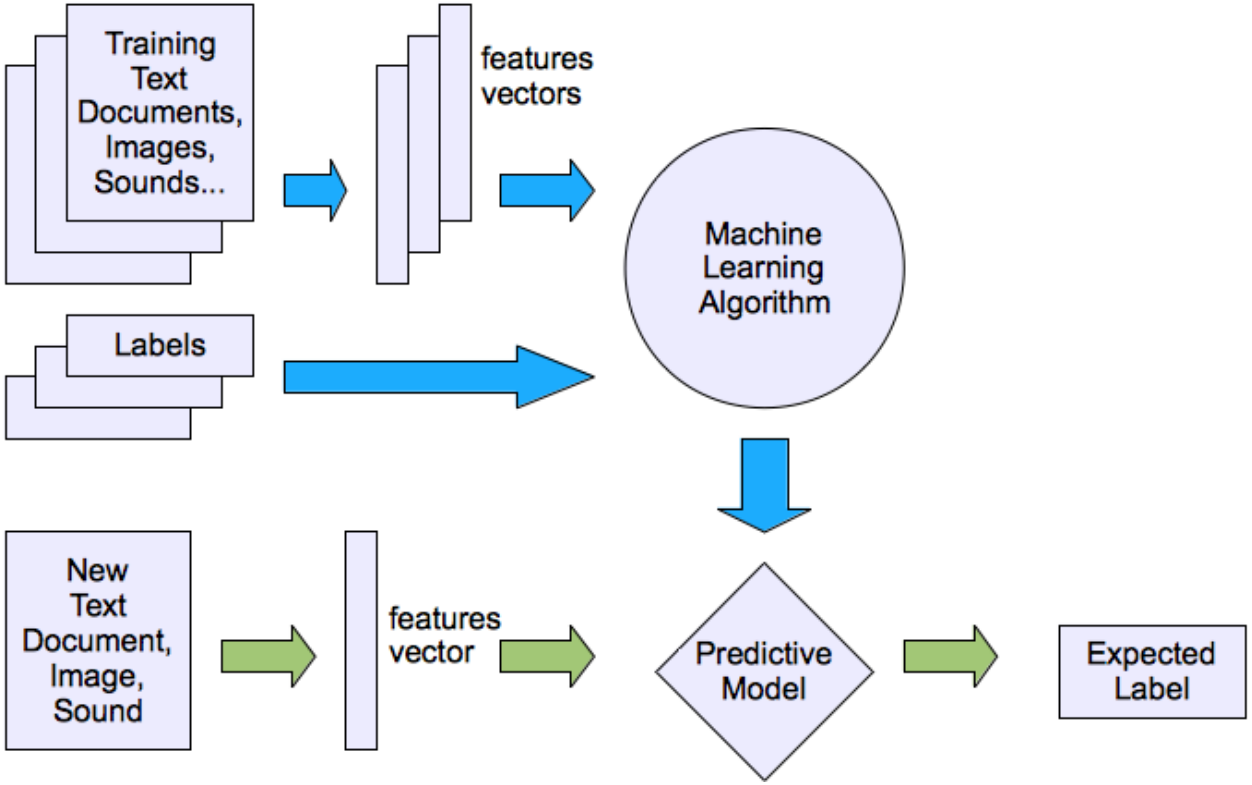
Unsupervised learning



Semi-supervised learning

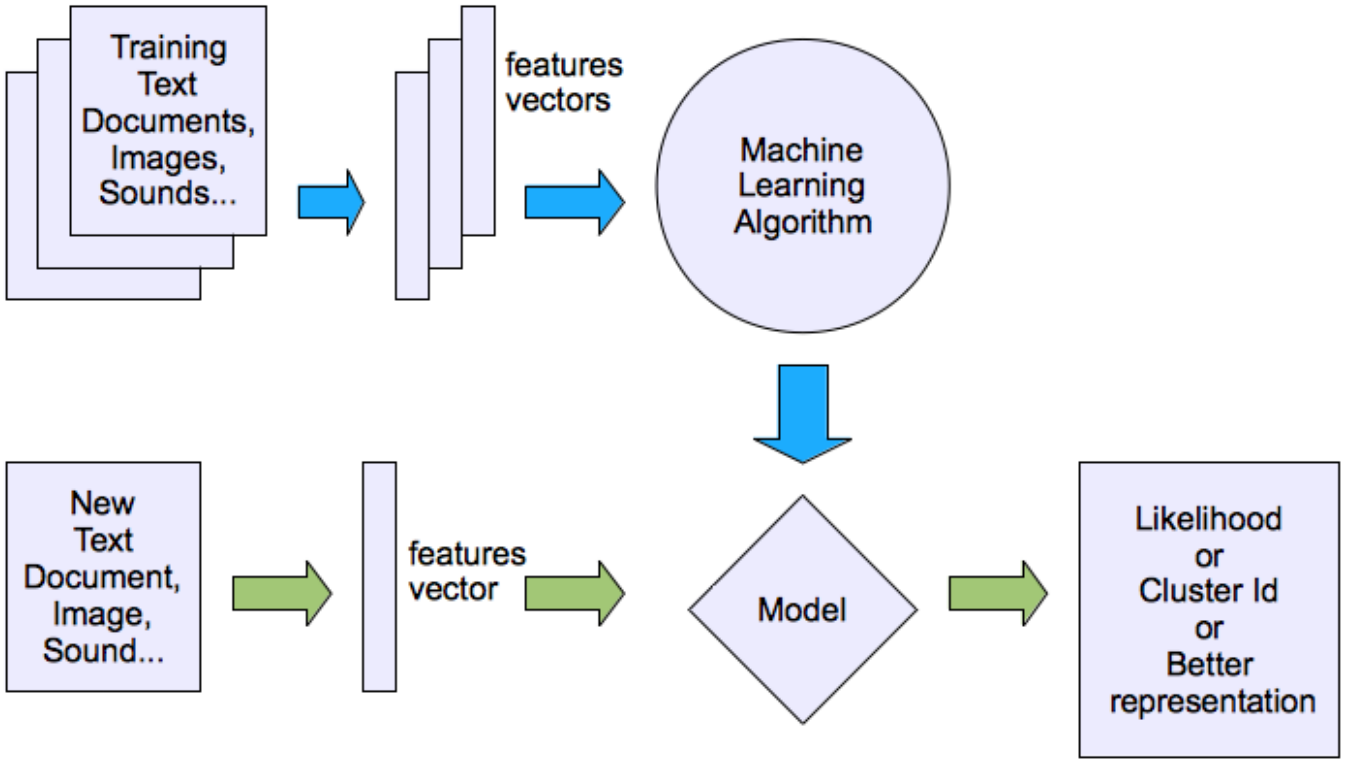
Machine Learning Methodology

- ***Supervised learning***



Machine Learning Methodology

- *Unsupervised learning*



Objectives (Examples)

- ***Supervised***: Low Eout or maximize probabilistic terms

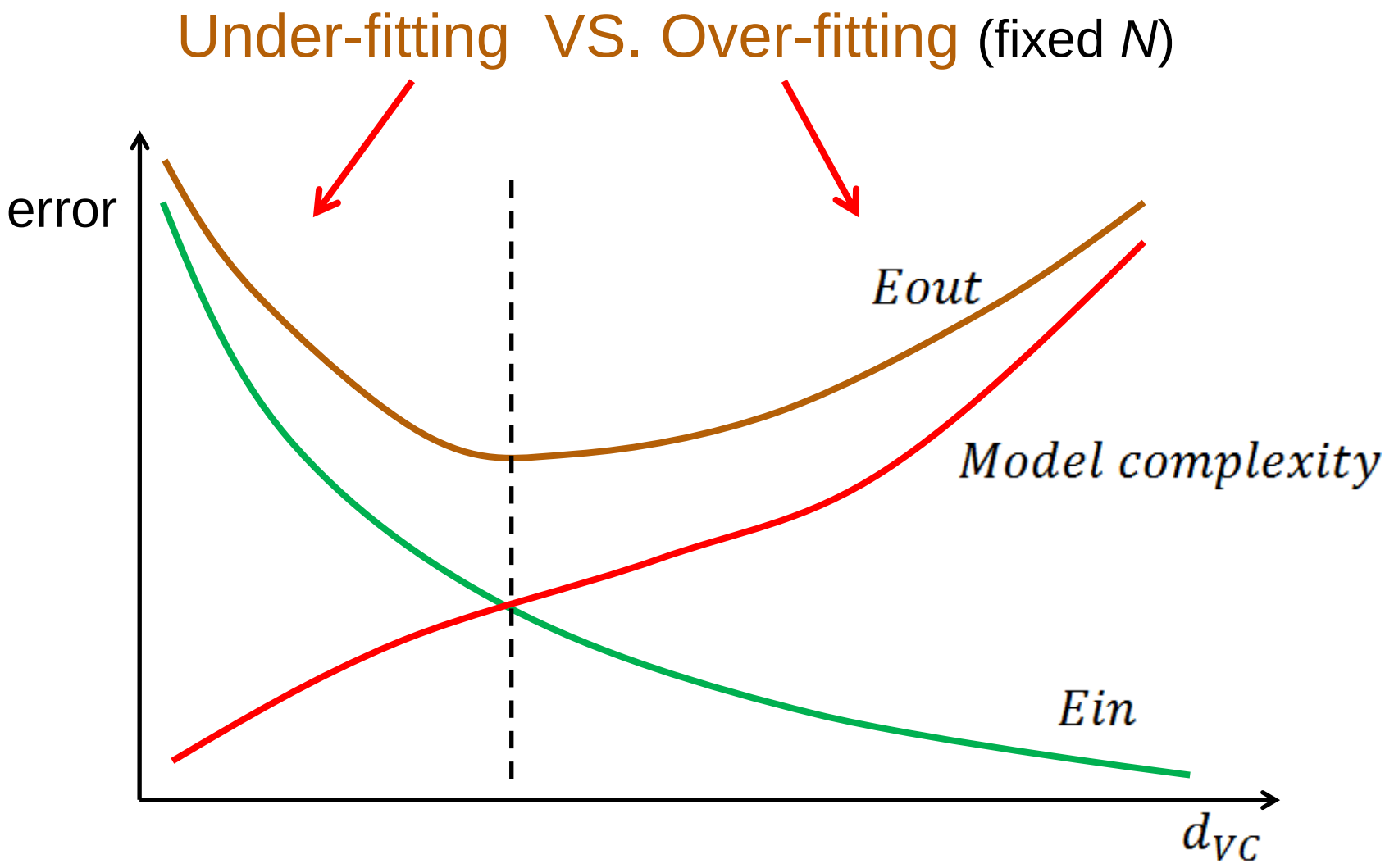
$$error = \frac{1}{N}\sum_{n=1}^{N}[y_n \neq g(x_n)]$$

Ein: for training set
Eout: for testing set

$$Eout(g) \leq Ein(g) \pm O\left(\sqrt{\frac{d_{VC}}{N} ln\, N}\right)$$

- ***Unsupervised***: Minimum quantization error, Minimum distance, MAP, MLE(maximum likelihood estimation)

What are we seeking?

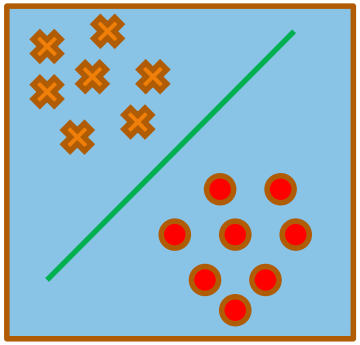


Learning Representations

- Supervised learning representations
  - **Linear classifier** (numerical functions)
  - **Parametric** (Probabilistic functions)
    - Naïve Bayes, Gaussian discriminant analysis (GDA), Hidden Markov models (HMM), Probabilistic graphical models
  - **Non-parametric** (Instance-based functions)
    - *K*-nearest neighbors, Kernel regression, Kernel density estimation, Local regression
  - **Non-metric** (Symbolic functions)
    - Classification and regression tree (CART), decision tree
  - **Aggregation**
    - Bagging (bootstrap + aggregation), Adaboost, Random forest

Linear Classifiers

- Linear classifier



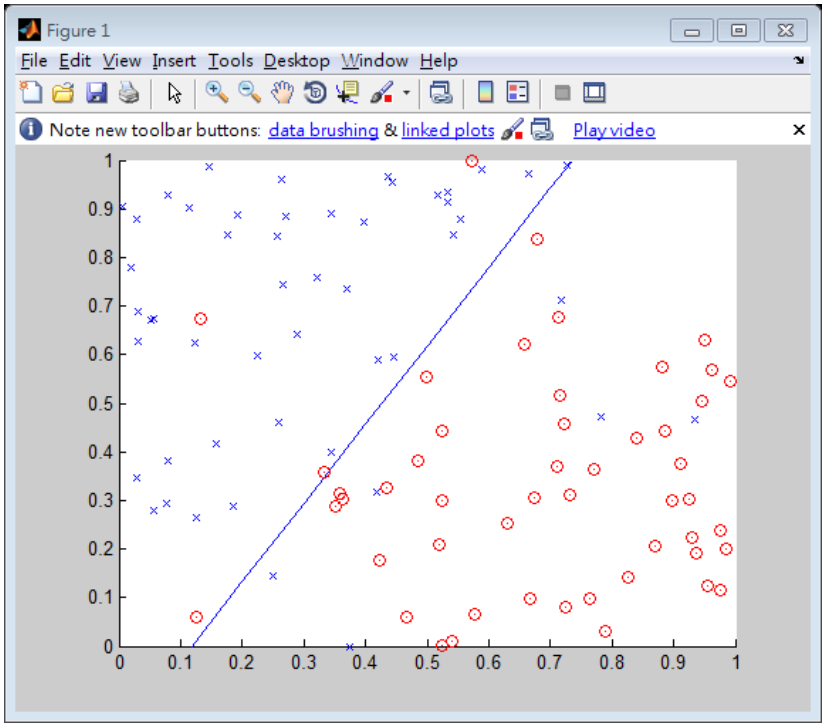
$$g(x_n) = sign(w^T x_n)$$

where *w* is an *d*-dim vector (learned)

- Techniques:
  - Perceptron
  - Logistic regression
  - Support vector machine (SVM)
  - Ada-line
  - Multi-layer perceptron (MLP)

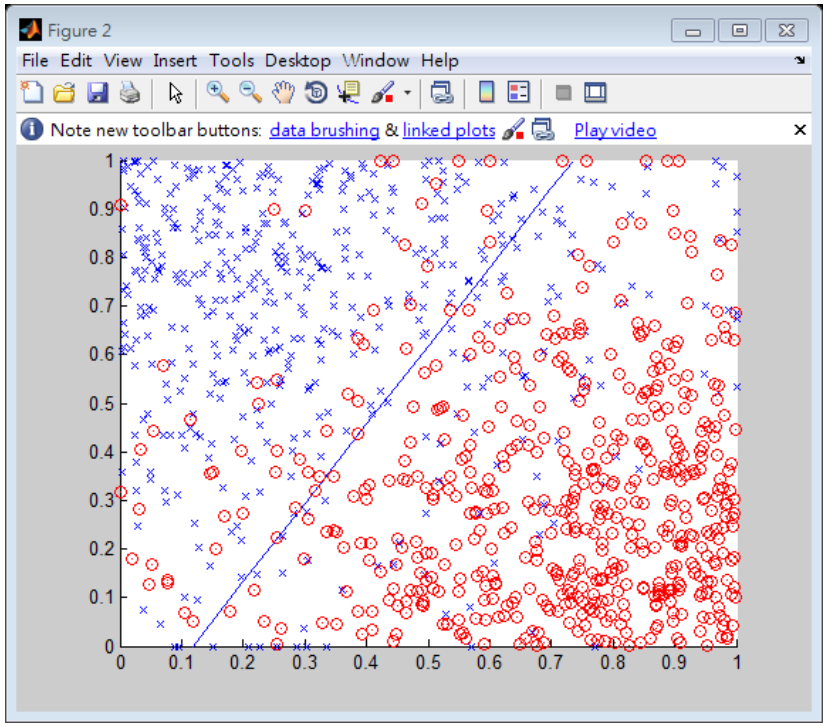
Linear Classification using NN

Using **perceptron learning algorithm**(PLA)



Training

Error rate: 0.10

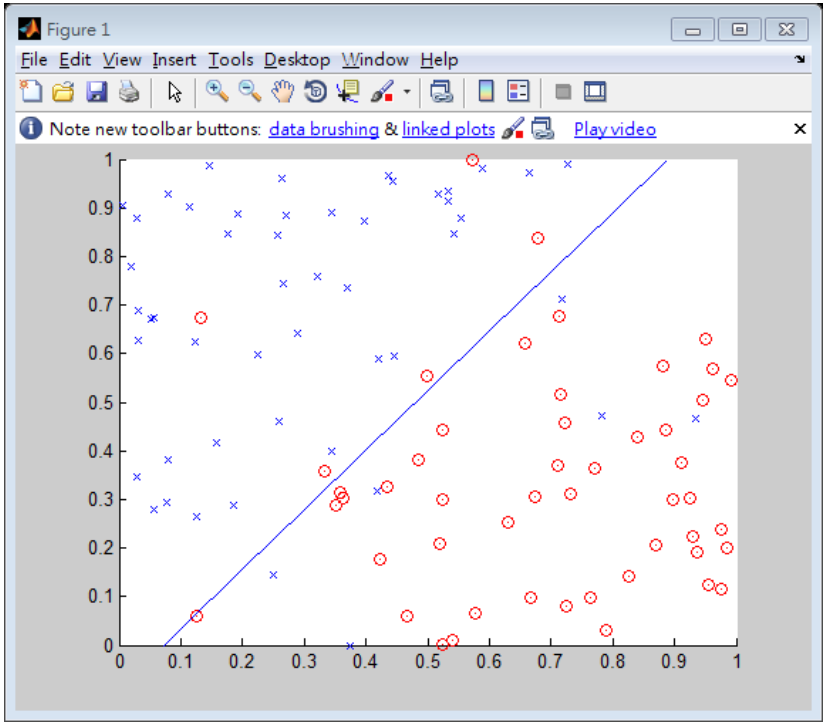


Testing

Error rate: 0.156

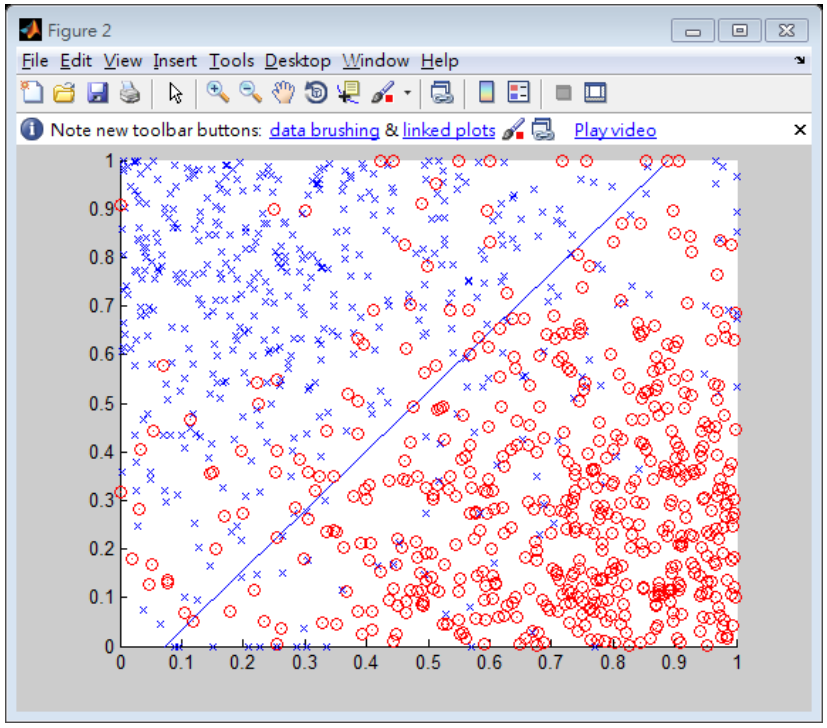
Linear Classification using Regression

Using **logistic regression**



Training

Error rate: 0.11

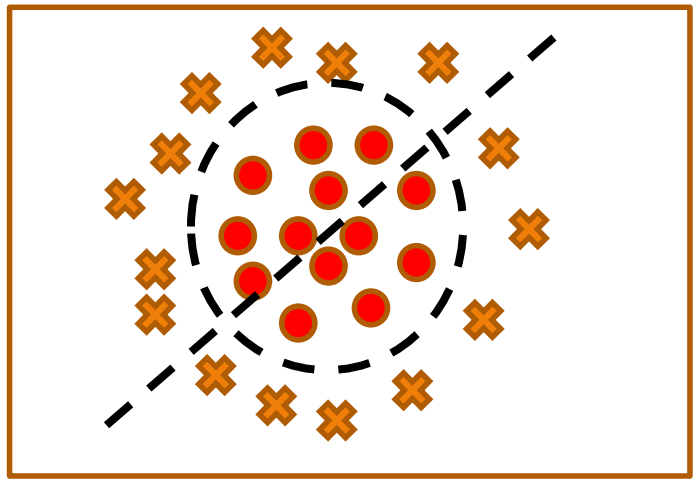


Testing

Error rate: 0.145

Non-Linear Classification

- Non-linear case



$$x_n = [x_{n1}, x_{n2}]$$

$$x_n = [x_{n1}, x_{n2}, x_{n1} * x_{n2}, {x_{n1}}^2, {x_{n2}}^2]$$

$$g(x_n) = sign(w^T x_n)$$

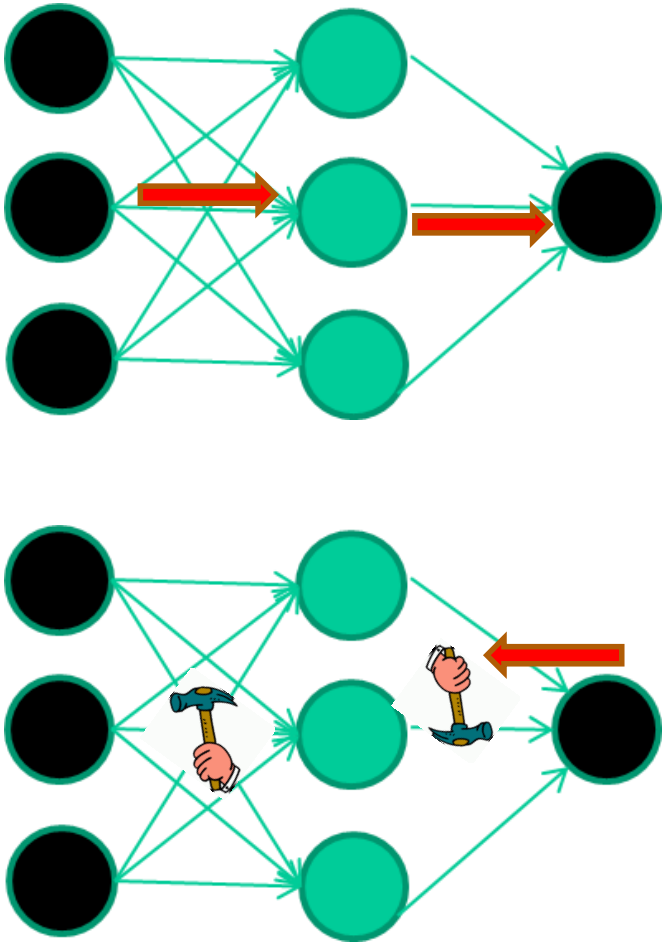
- Support vector machine (SVM):
  - Linear to nonlinear: **Feature transform** and **kernel function**

Deep Learning Representation

- Functional approximation
  - NN is a universal function approximator
    - Can approximate an arbitrary non-linear function
- We focus on *supervised* learning

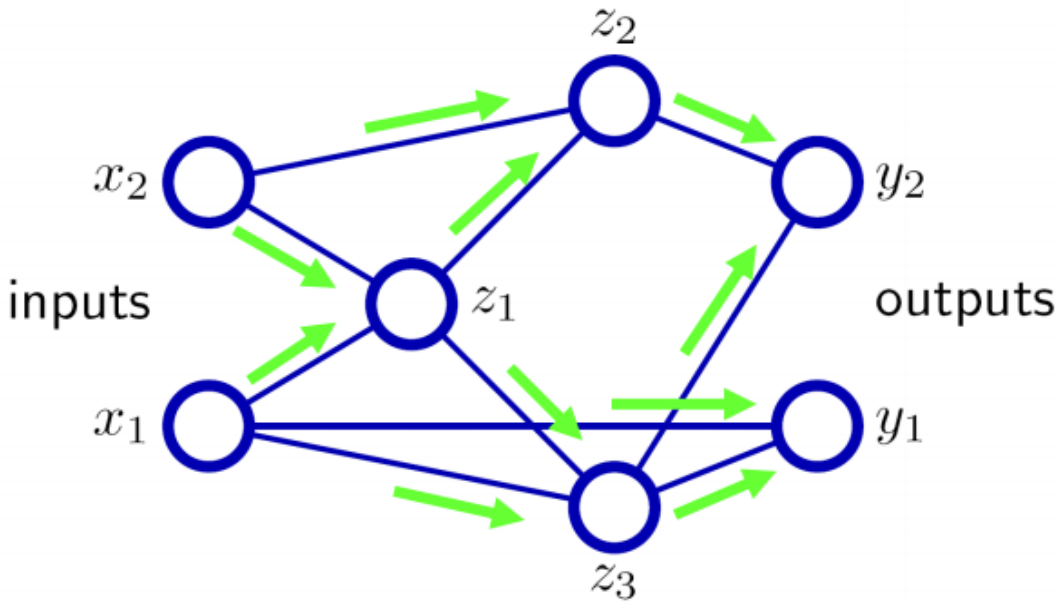
Neural Network Basics

- Two main operations
  - Forward propagation
  - Backward propagation (weight adjustment)



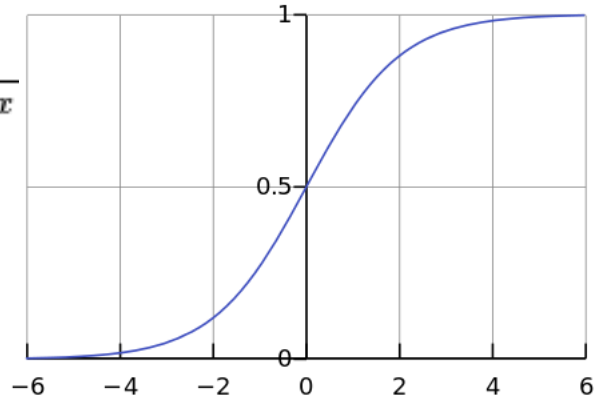
Forward Propagation

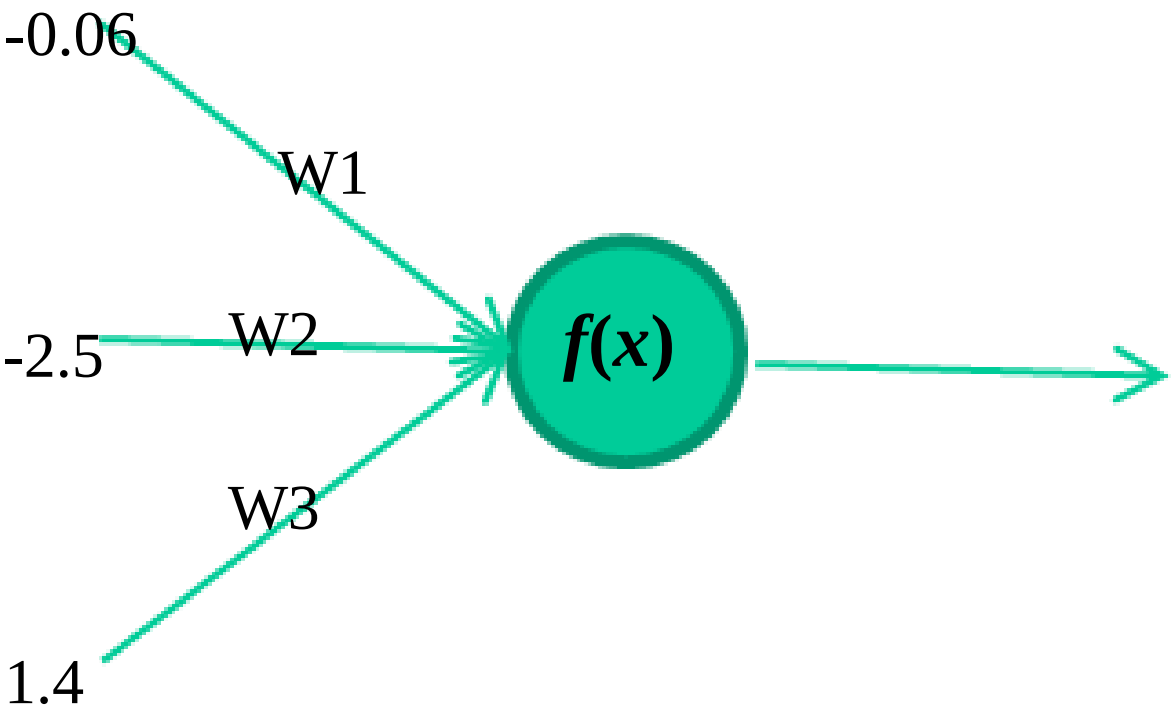
- Forward Propagation :
  - Sum inputs, produce activation, feed-forward



Example

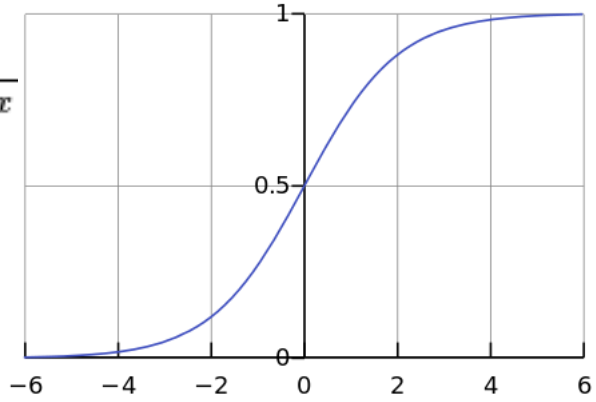
$$f(x) = \frac{1}{1 + e^{-x}}$$

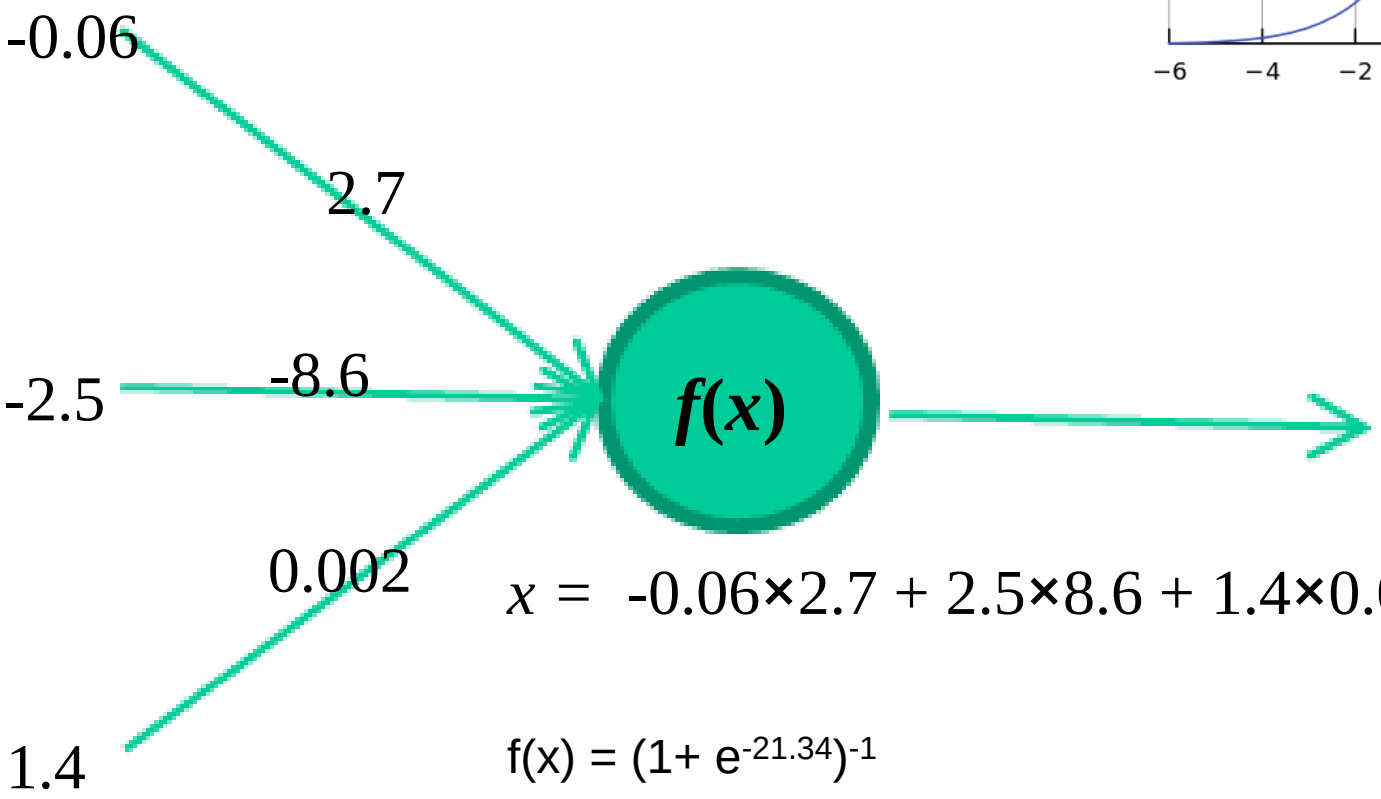




Example

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



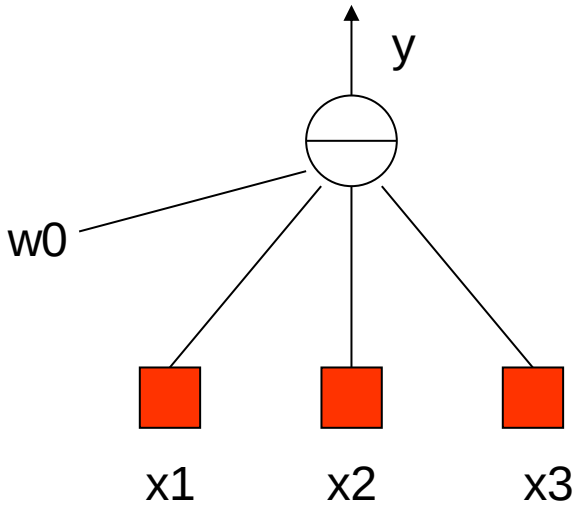


$x = -0.06 \times 2.7 + 2.5 \times 8.6 + 1.4 \times 0.002 = 21.34$

$f(x) = (1 + e^{-21.34})^{-1}$

General View: Forward Propagation

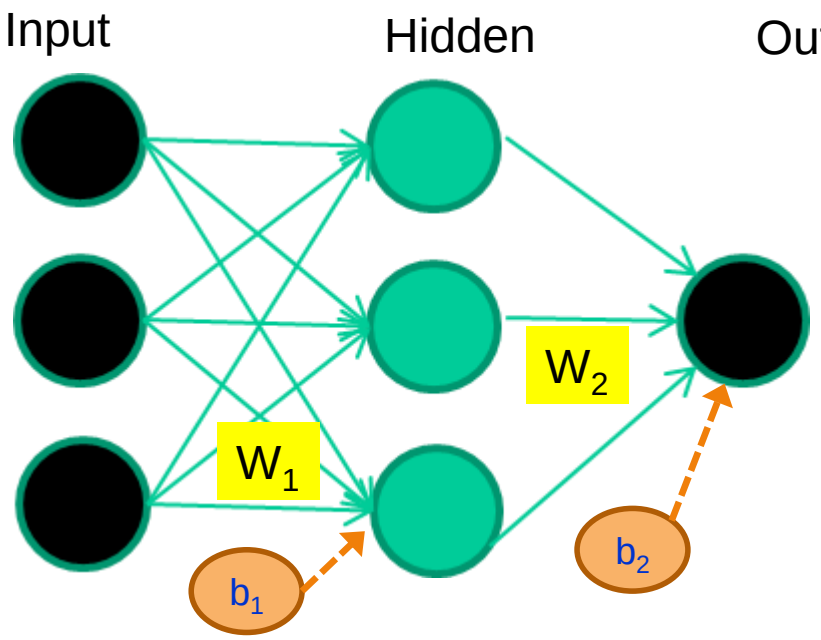
- An *Artificial Neuron* is a non-linear parameterized function with restricted output range



$$y = f\left(w_0 + \sum_{i=1}^{n-1} w_i x_i\right)$$

w_0 also called *a bias term (b_i)*

Multiple Layers: Artificial neuron

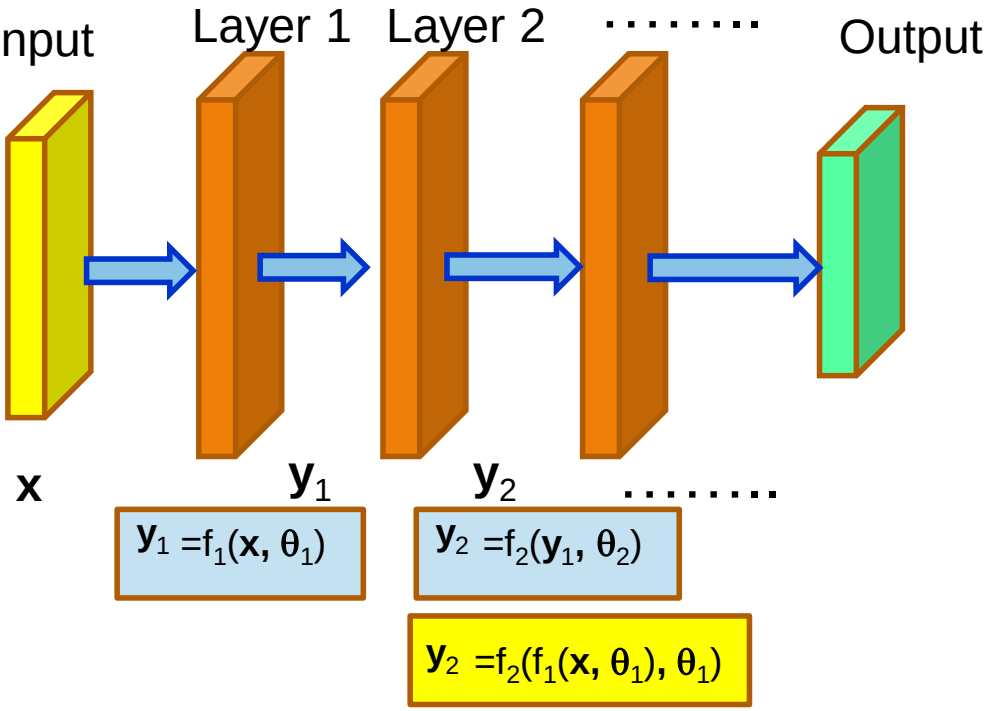


x $\quad$ **y_1** $\quad$ **y_2** $\quad$ VECTORS

$$\mathbf{y}_1 = f_1(W_1\,\mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{y}_2 = f_2(W_2\,\mathbf{y}_1 + \mathbf{b}_2)$$

Training: Deep Network

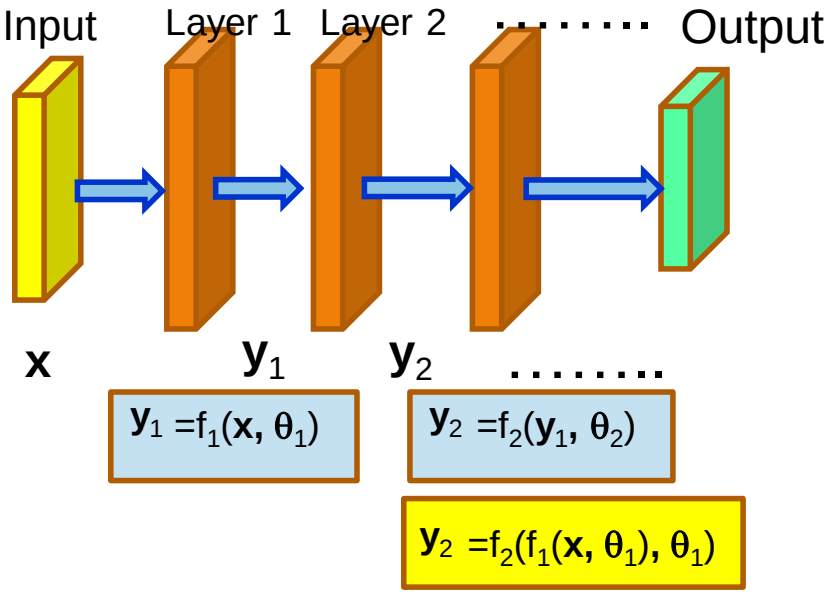


$$y_N(x, \theta_1, \theta_2, \ldots \theta_N) = f_N(f_{N-1}(\ldots(f_1(x, \theta_1), \theta_{N-1}), \theta_N)$$

Loss (cost) function

$$\theta^* = \arg\min_\theta \sum_{(x,y)\subseteq(X,Y)} J[y, fL(x, \theta_{1, \ldots L})]$$

Summary: Deep Network



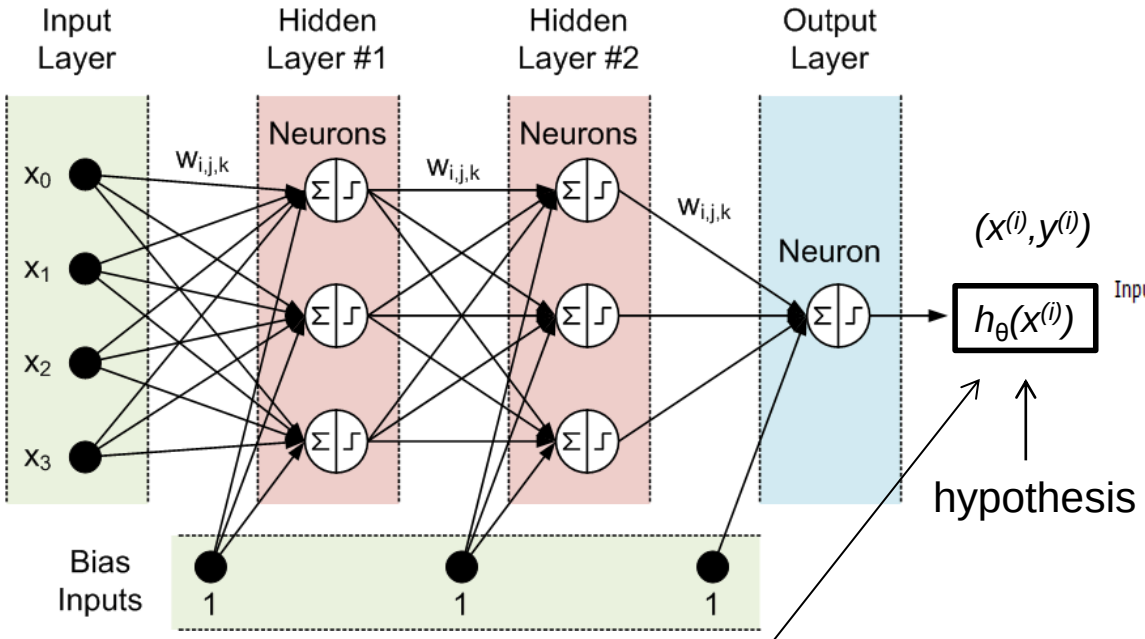
Deep Network: family of parametric, non-linear, hierarchical representations

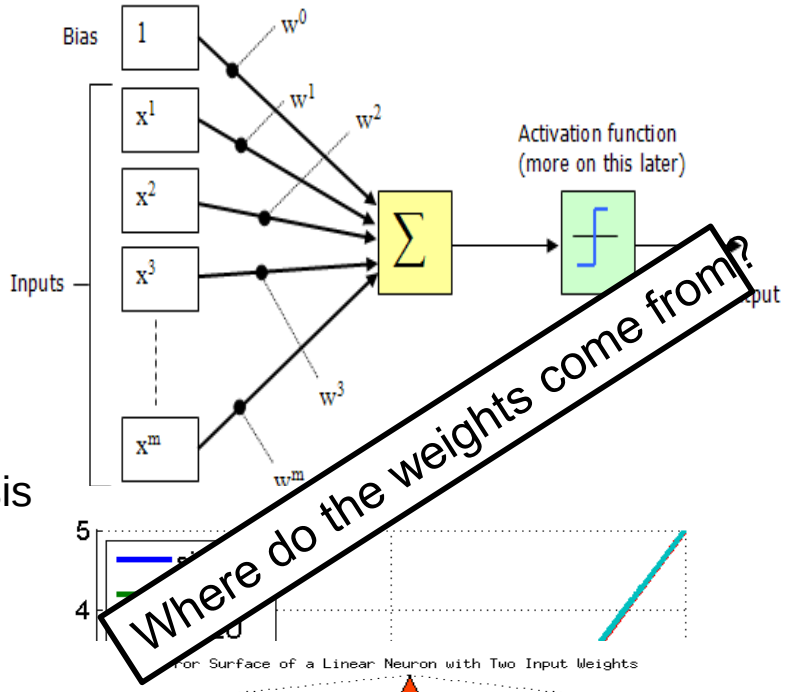
$$y_N(x, \theta_1, \theta_2, \ldots \theta_N) = f_N(f_{N-1}(\ldots(f_1(x, \theta_1), \theta_{N-1}), \theta_N)$$

Training: optimize network parameters to minimise loss over training set

$$\theta^* = \arg\min_\theta \sum_{(x,y)\subseteq(X,Y)} J[y, fL(x, \theta_{1, \ldots L})]$$

Deep Feed Forward Neural Nets (in 1 Slide (☺))

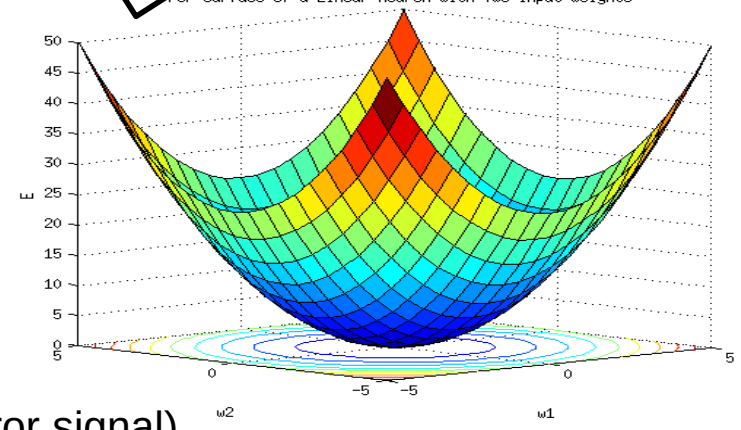




Forward Propagation

$$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

g?



Learning is the adjusting of the weights $w_{i,j}$ such that the cost function $J(\boldsymbol{\theta})$ is minimized (a form of Hebbian learning).

Simple learning procedure: *Back Propagation* (of the error signal)

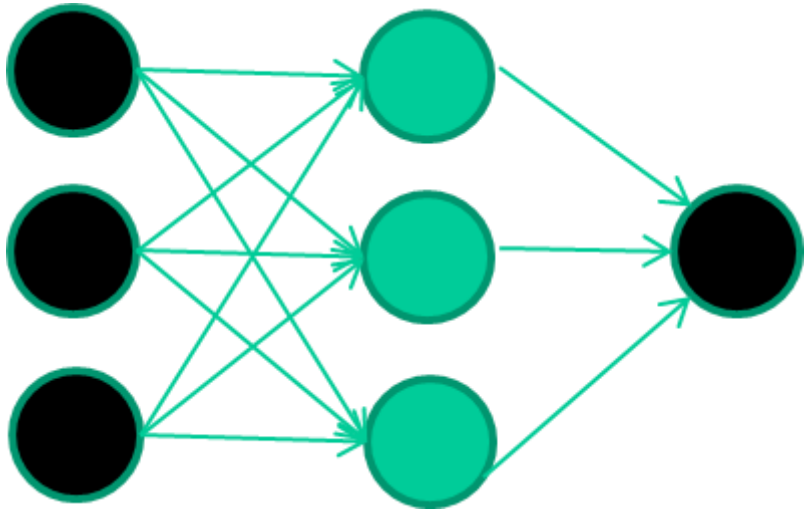
Supervised Training of Neural Networks

- Use set of training data D
- Adjust weights to optimize classification performance over D

Example

A dataset

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

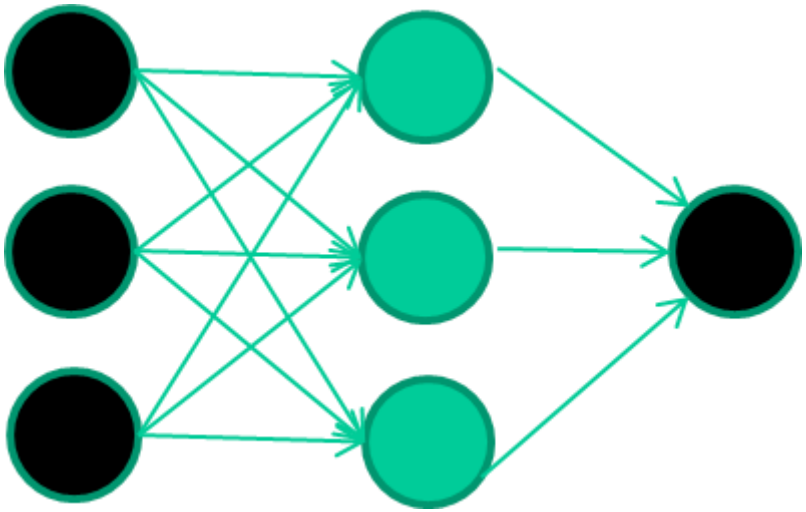


Example

Training the neural network

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

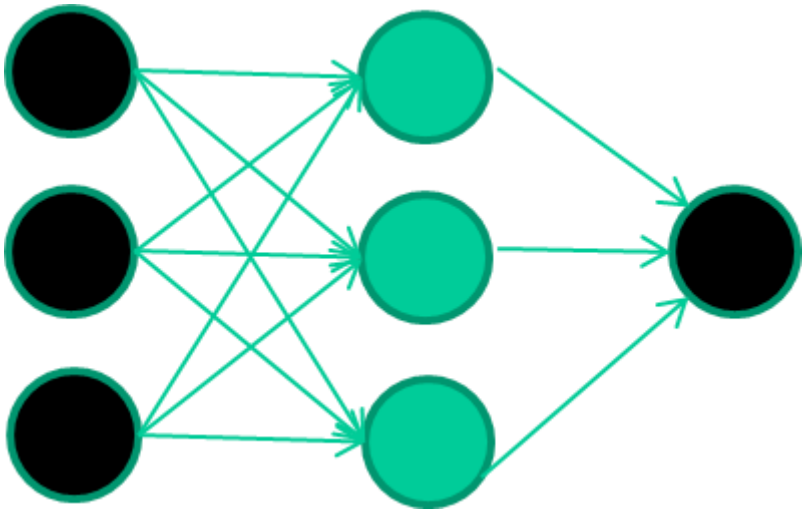


Example

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Initialise with random weights

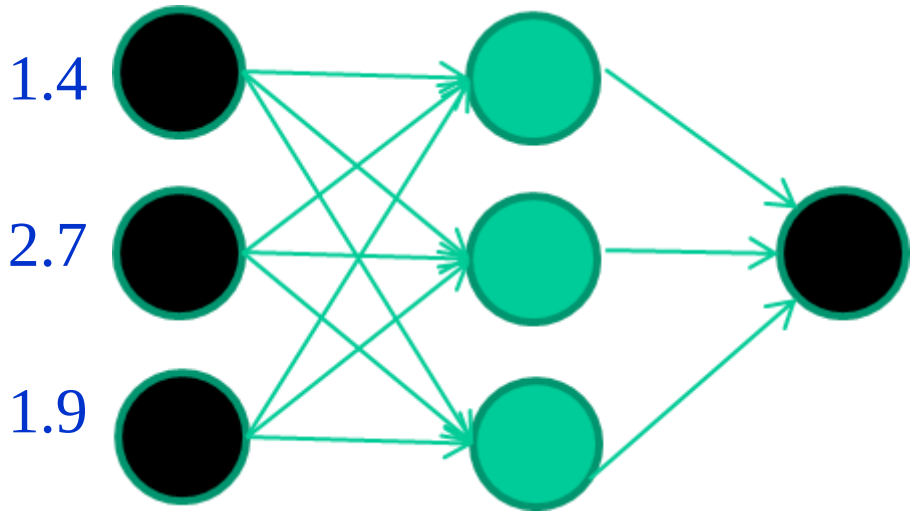


Example

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Present a training pattern

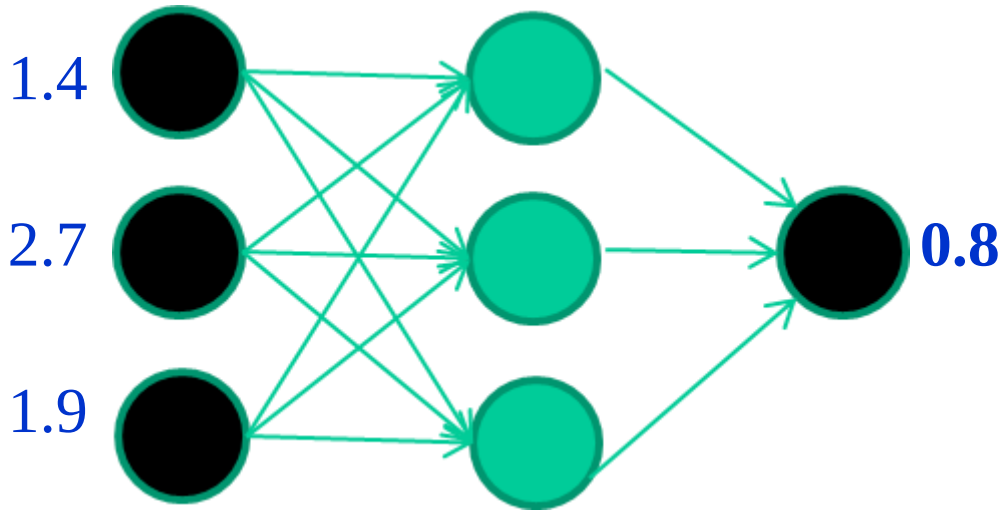


Example

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Feed it through to get output

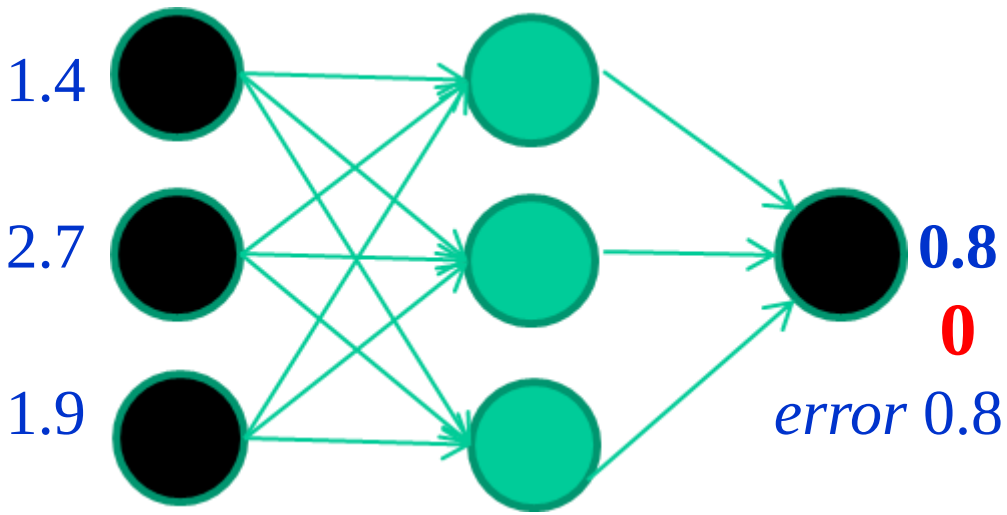


Example

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Compare with target output

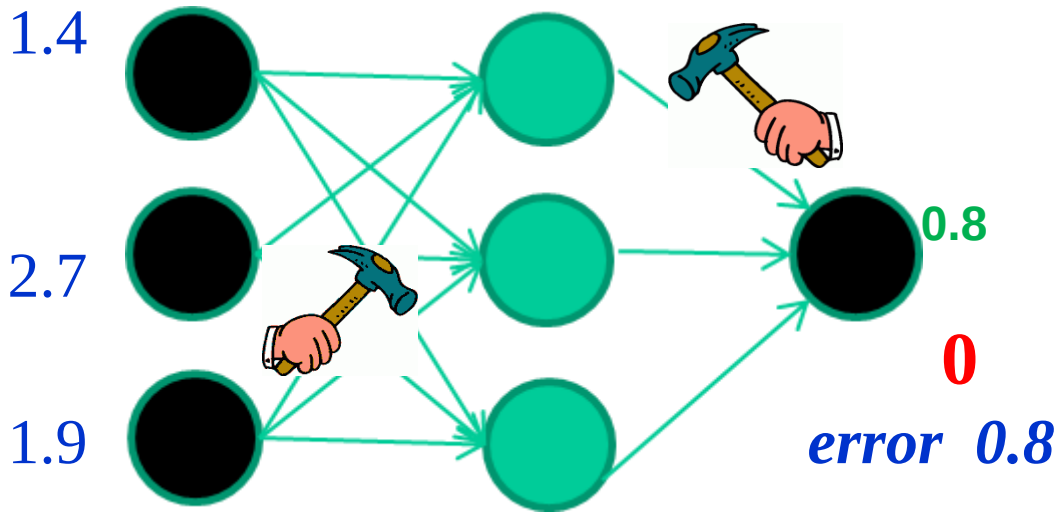


Example

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Adjust weights based on error



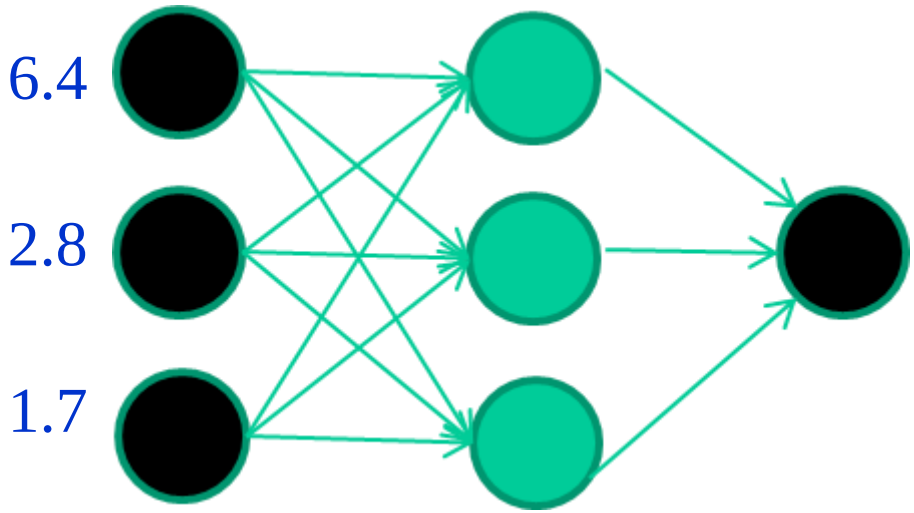
Example

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Present a training pattern

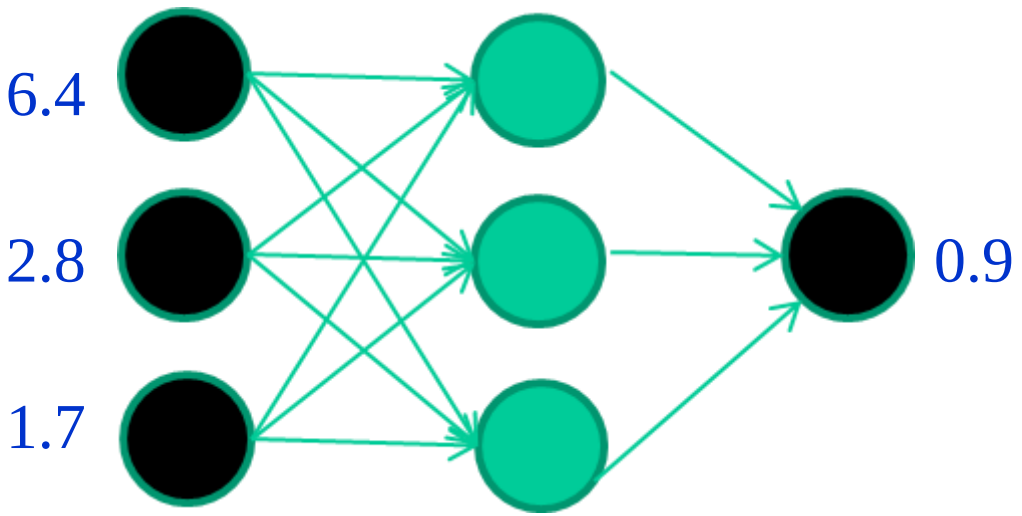


Example

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Feed it through to get output



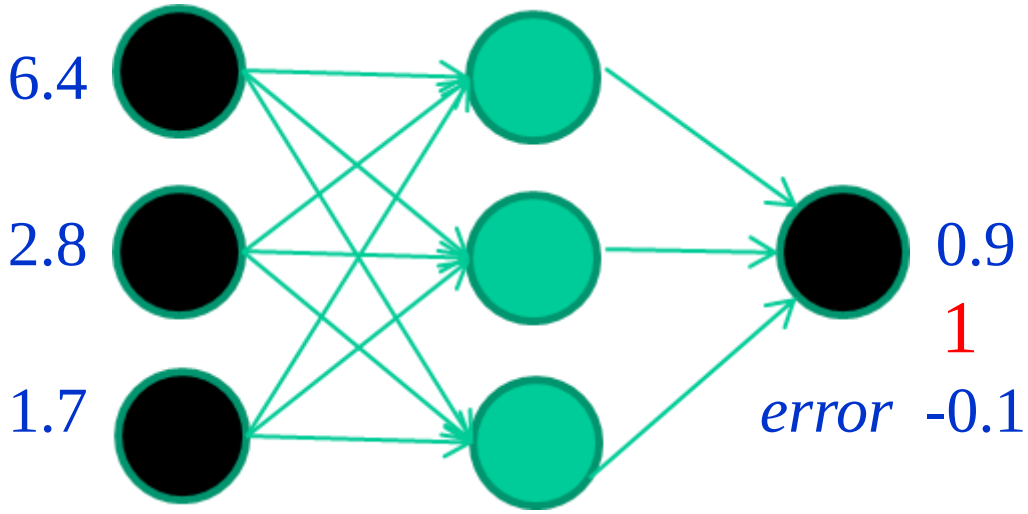
Example

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Compare with target output

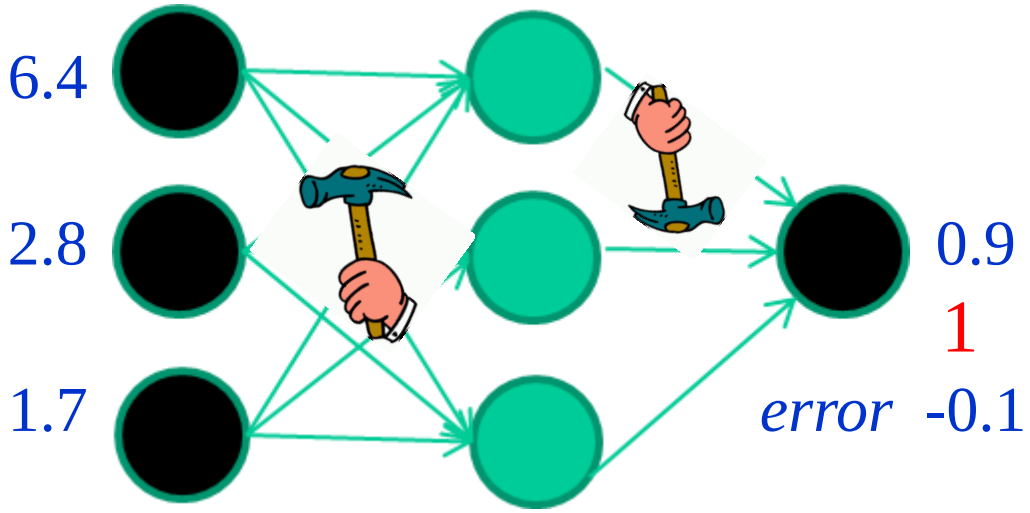


Example

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Adjust weights based on error

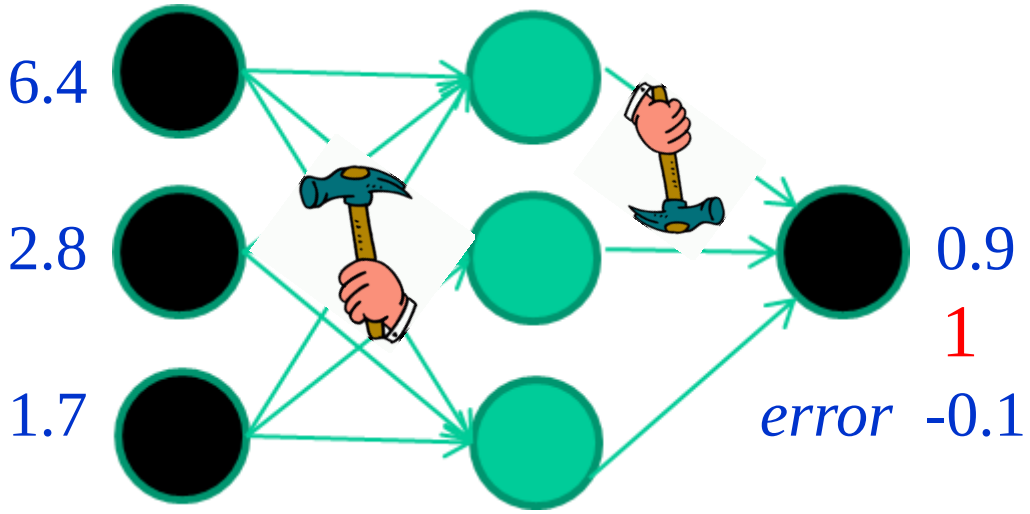


Example

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

And so on ….

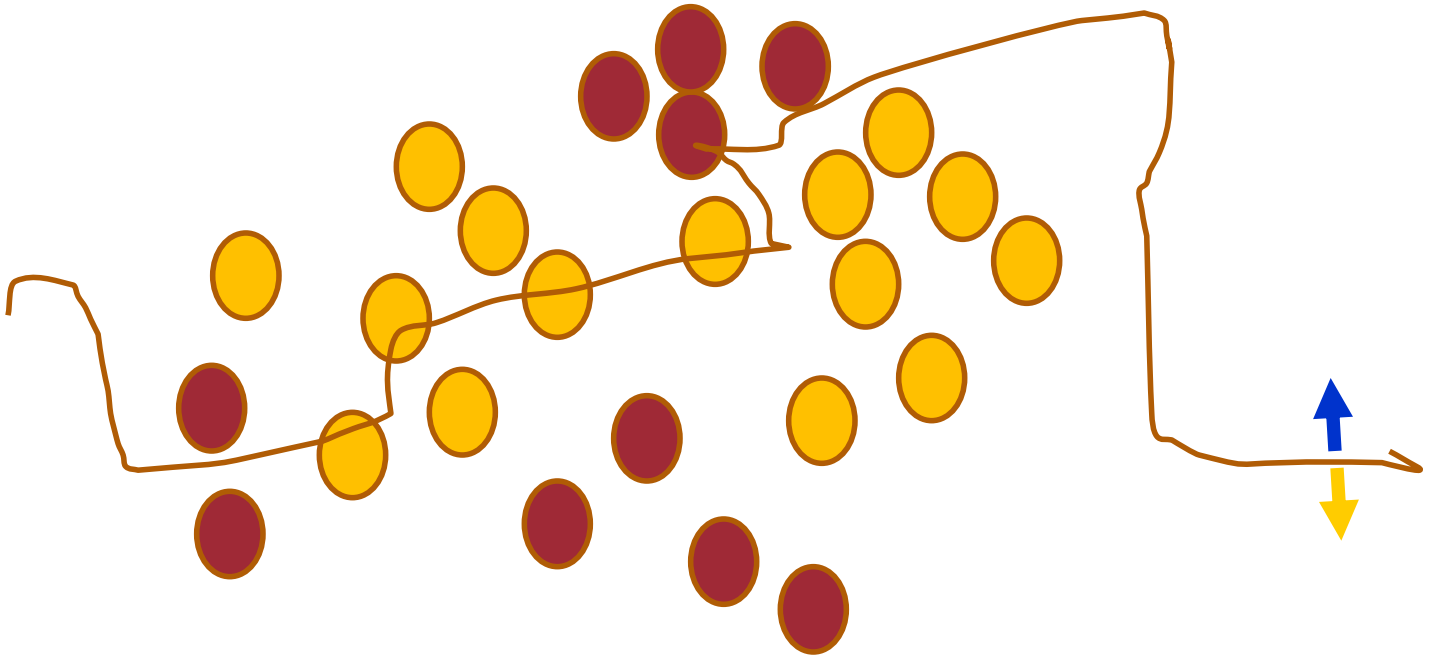


Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments

Algorithms for weight adjustment are designed to make changes that will reduce the error

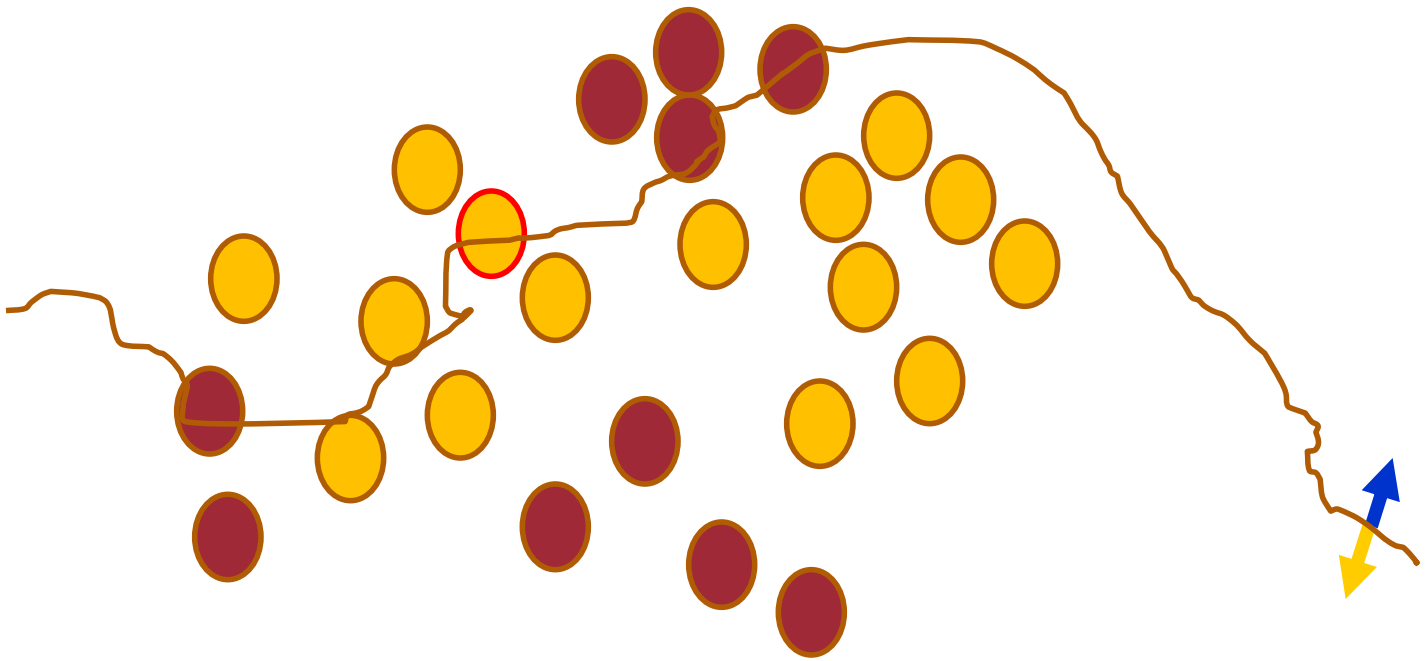
The decision boundary perspective…

Initial random weights



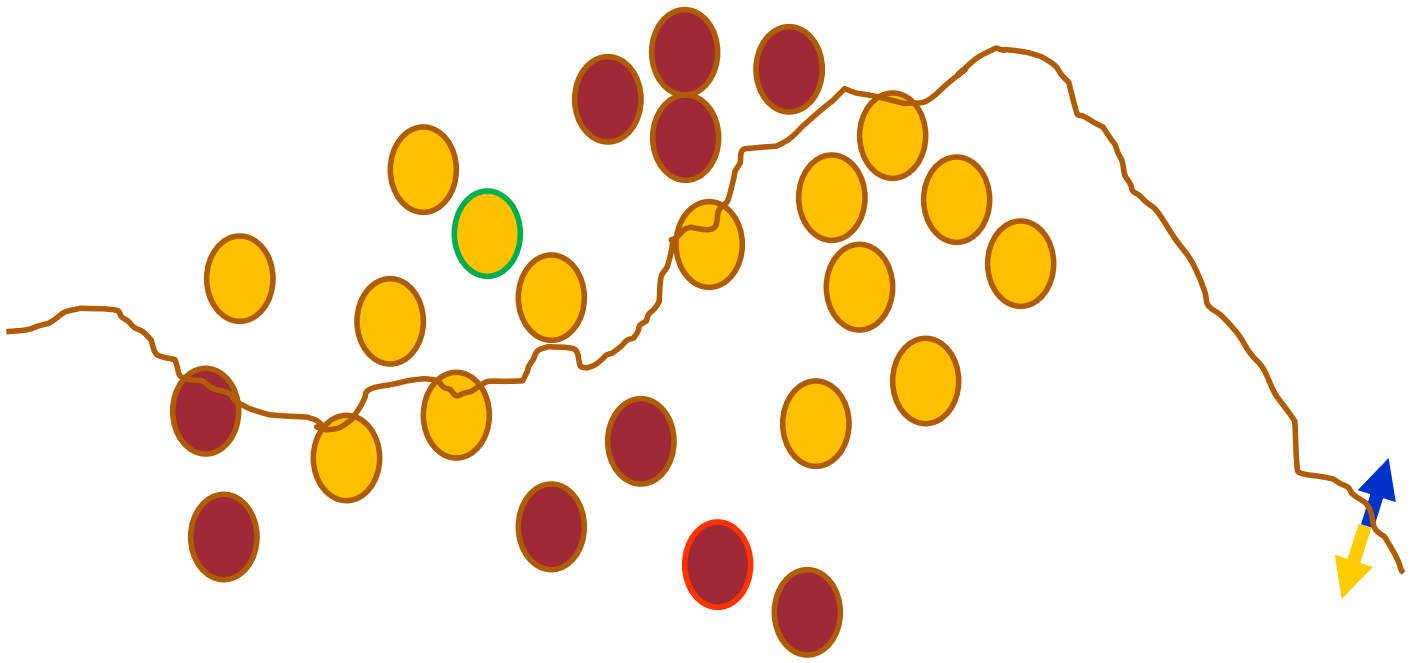
The decision boundary perspective…

Present a training instance / adjust the weights



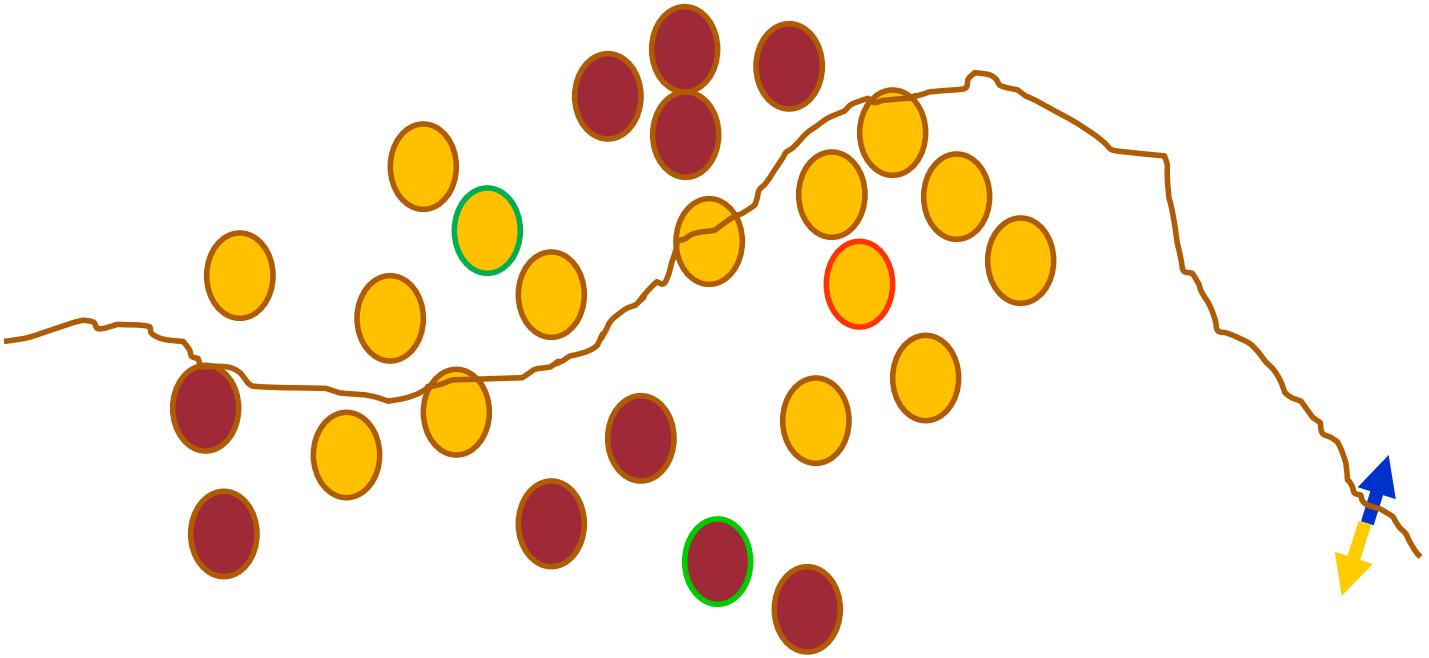
The decision boundary perspective…

Present a training instance / adjust the weights



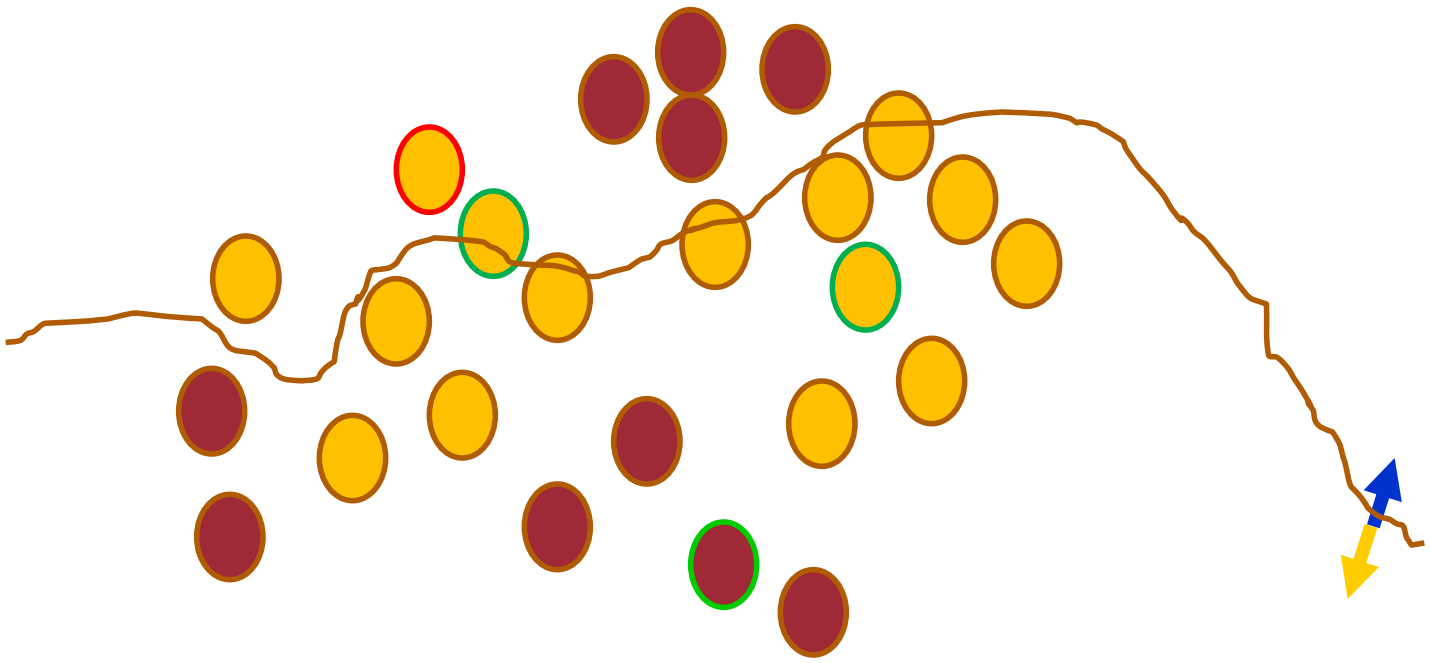
The decision boundary perspective…

Present a training instance / adjust the weights



The decision boundary perspective…

Present a training instance / adjust the weights

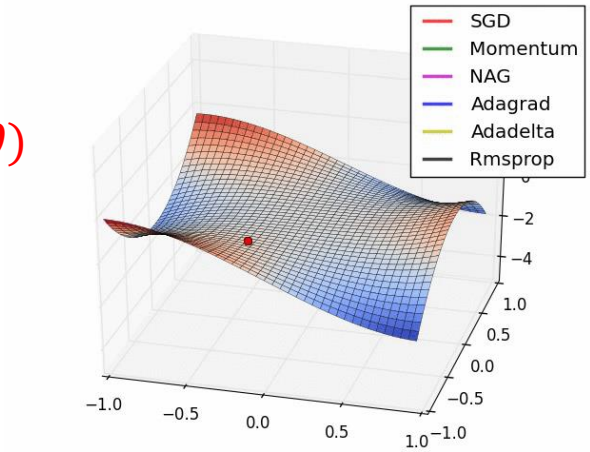


The decision boundary perspective…

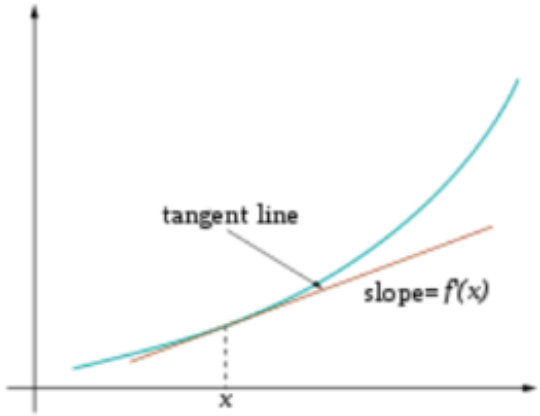
Eventually ….

Training: Summary

Sample labeled data (**batch**) → **Forward** it through the network, get predictions → **Back-propagate** the errors → **Update** the network weights



Optimize (min. or max.) **objective/cost function** $e(\theta)$
Generate **error signal** that measures difference between predictions and target values



Use error signal to change the **weights** and get more accurate predictions
Subtracting a fraction of the **gradient** moves you towards the **(local) minimum of the cost function**

https://medium.com/@ramrajchandradevan/the-evolution-of-gradient-descend-optimization-algorithm-4106a6702d39