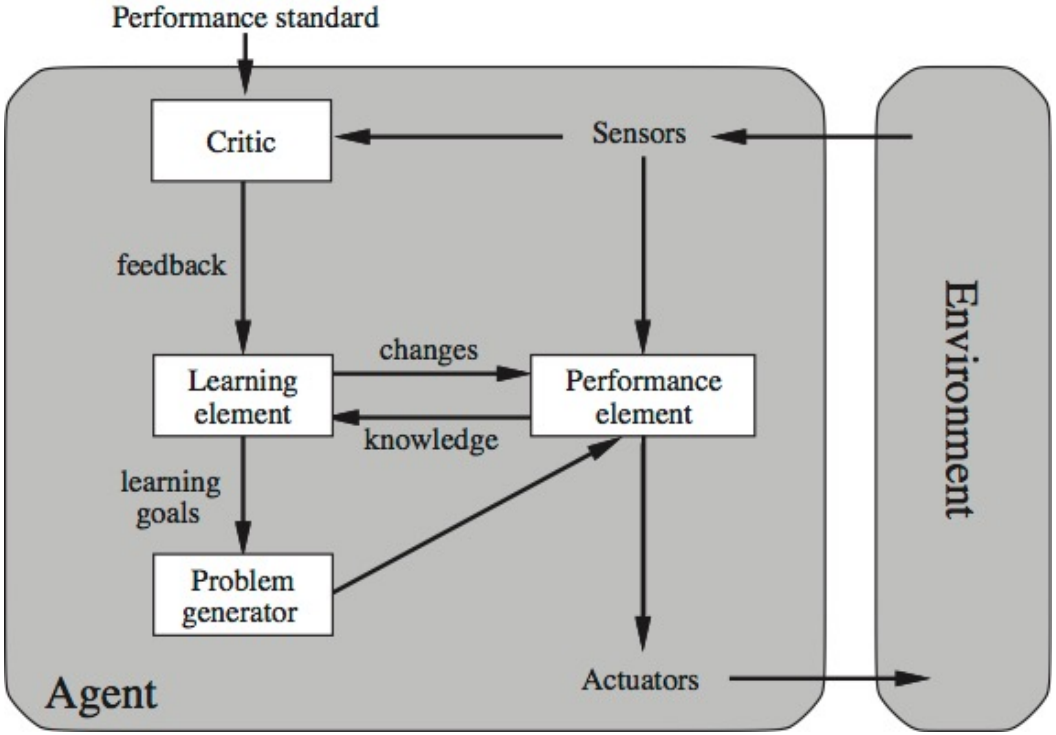
Learning

Larry Holder
School of EECS
Washington State University

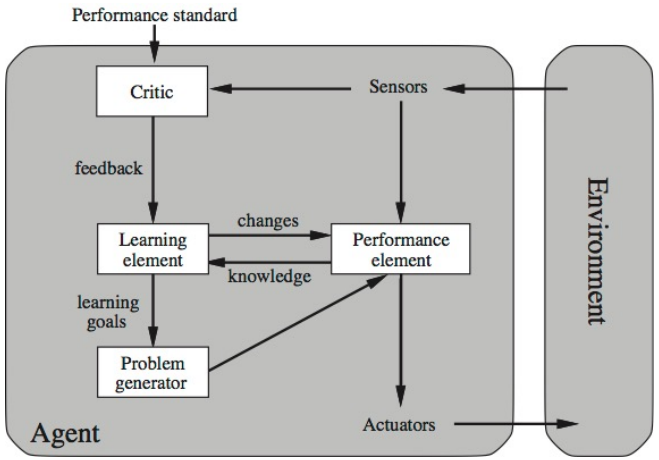
Learning Agent

- An agent that improves its performance at some task through experience



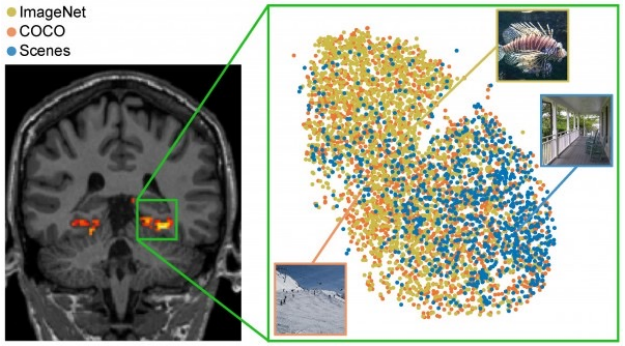
Details, details

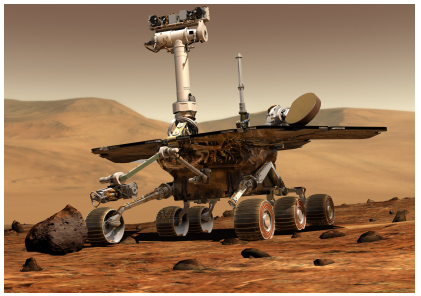
- How is knowledge represented?
- How is performance measured and critiqued?
- How is experience represented and obtained?



Why Do Machine Learning?

- Automated knowledge acquisition
- Discover new knowledge
- Understand human learning
- Agents need to adapt to unknown, dynamic environments

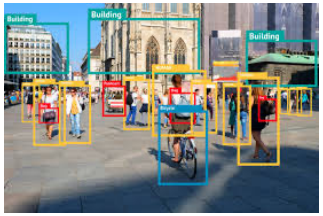


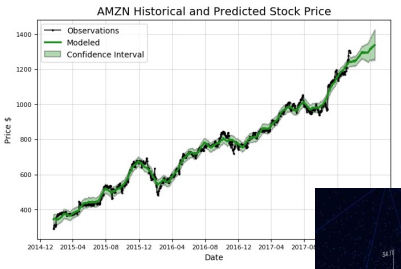


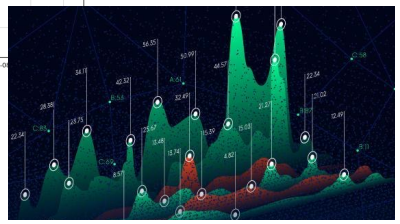
"If you invent a breakthrough in artificial intelligence, so machines can learn, that is worth ten Microsofts."
Bill Gates, 2004

Applications

- Medical diagnosis
- Autonomous control (planes, trains, automobiles, robots, …)
- Perception (speech, language, images, video)
- Recommendations (Amazon, Netflix)
- Prediction (business, financial, environment, health, energy, security, …)
- Fraud/intrusion detection
- …

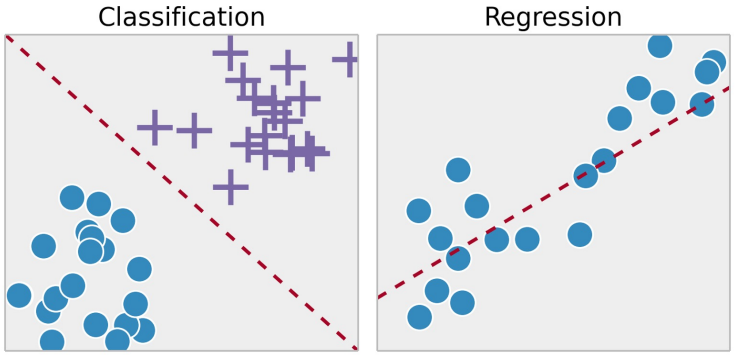


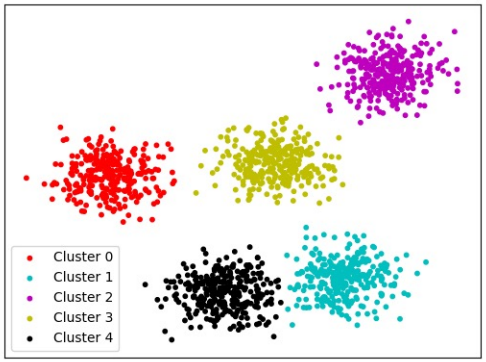


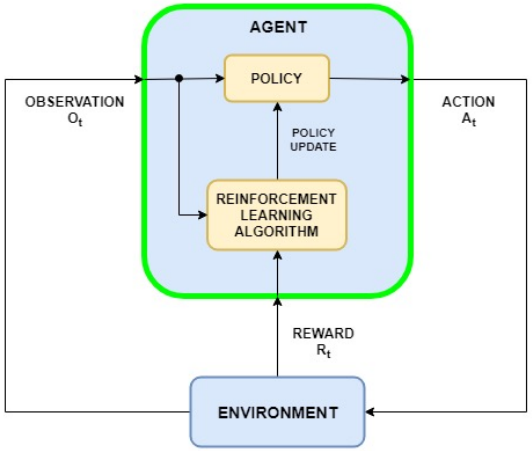


Approaches

- Supervised Learning
  - Classification
  - Regression
- Unsupervised Learning
  - Clustering
- Reinforcement Learning





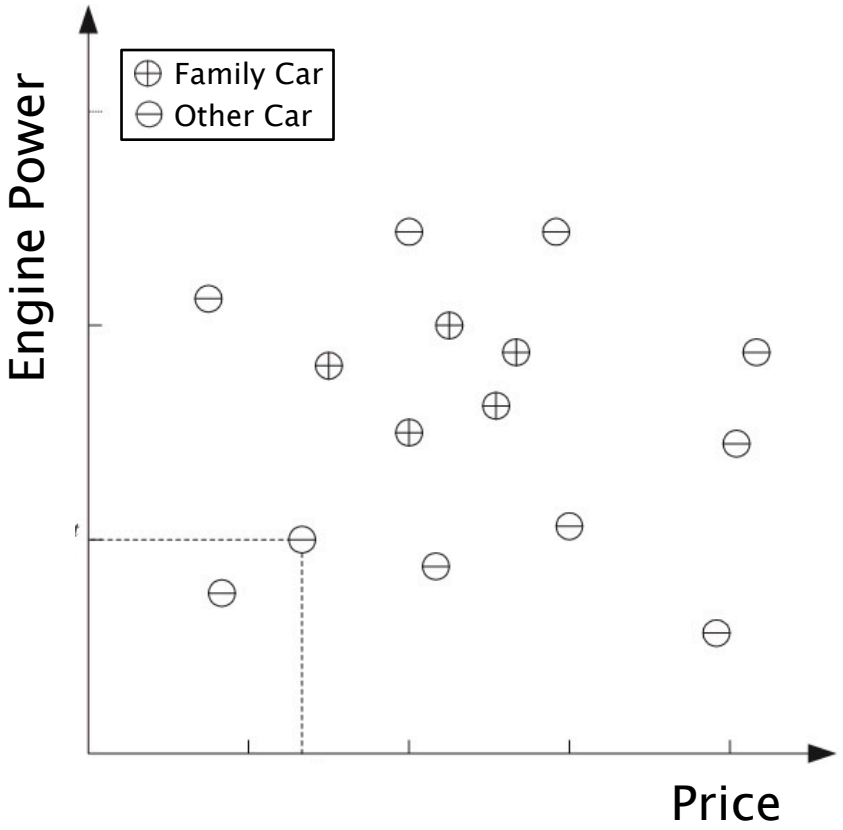


Supervised Learning

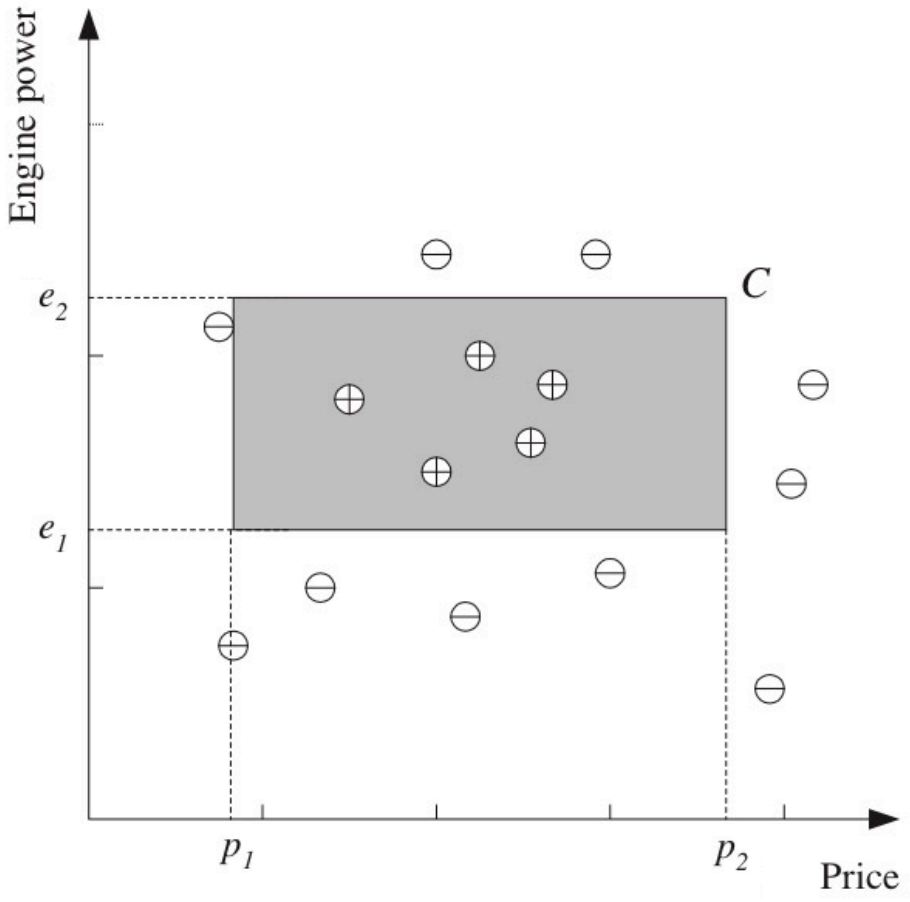
- Given training set of N examples of y=f(x)
  - (x_1,y_1), (x_2,y_2), …, (x_N,y_N)
- Find hypothesis h that approximates f

Example: "Family Car"

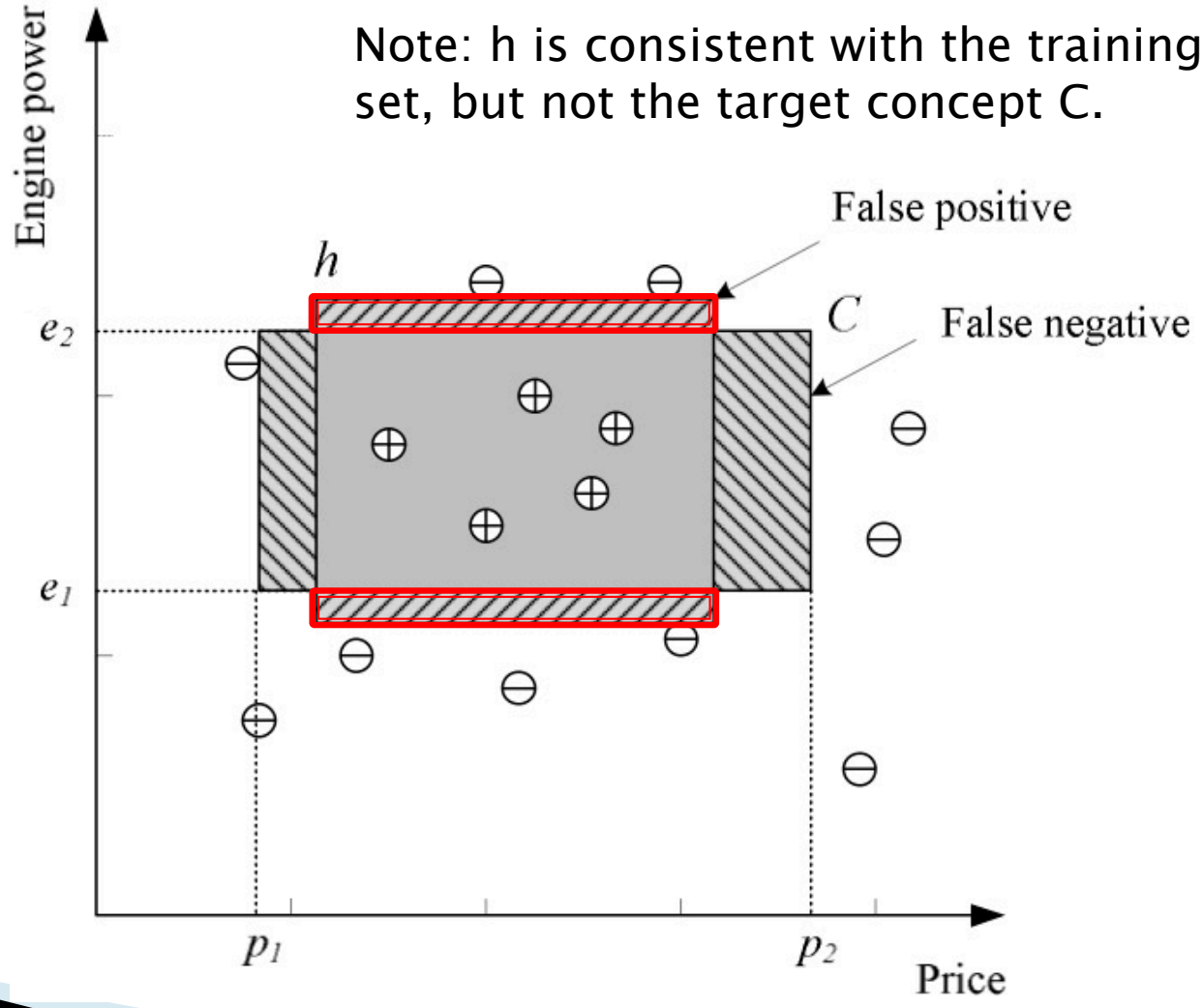
- Learn to classify cars into one of two classes: "family car" or "other"
- Each car is represented by two features: "engine power" and "price"
- Given several training examples of already-classified cars
- Output classifier that accurately classifies all cars



“Family Car” Target Concept

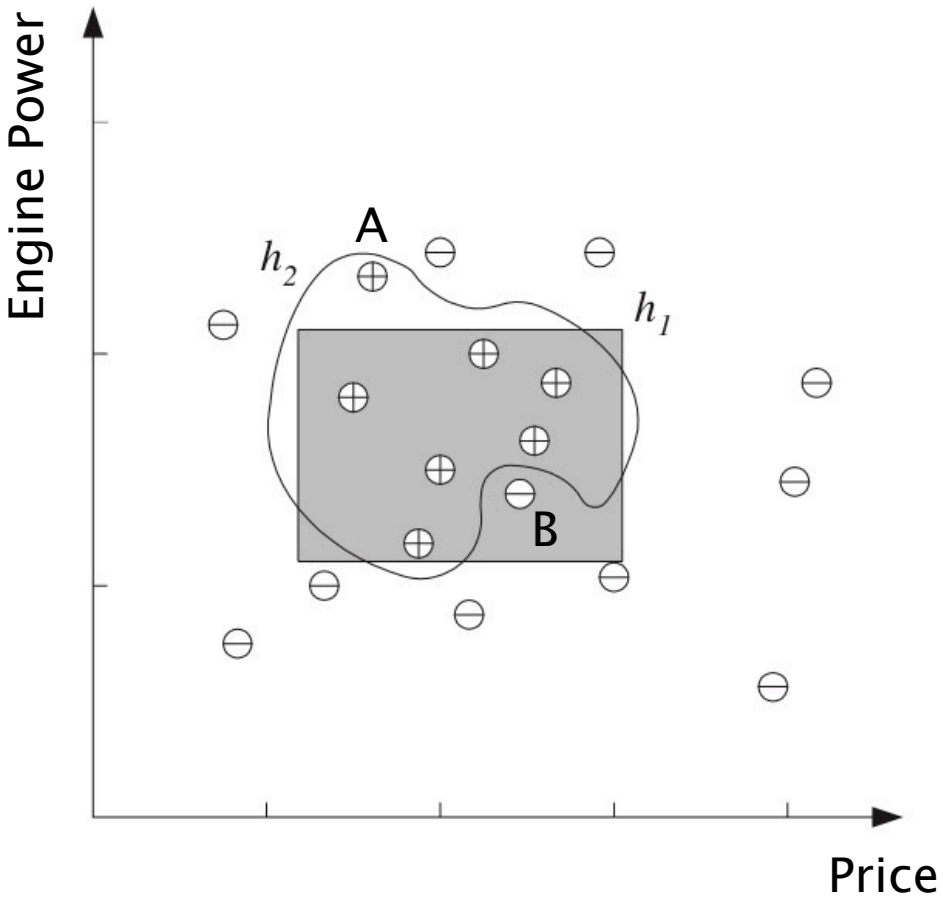


Hypothesis Error



Note: h is consistent with the training set, but not the target concept C.

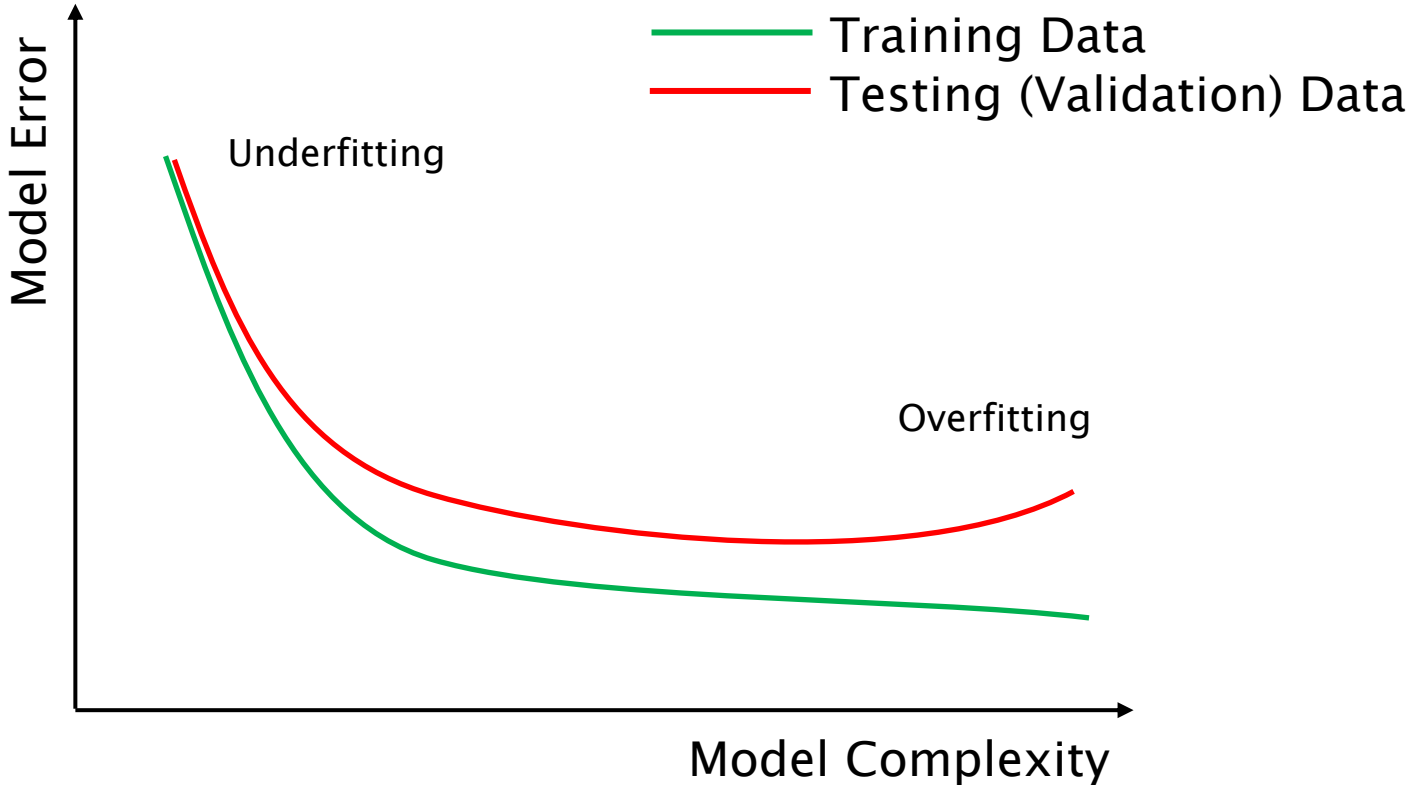
Underfitting vs. Overfitting



If A and B are noise, then h_2 overfits.

If A and B are *not* noise, then h_1 underfits.

Model (Hypothesis) Selection



Supervised Learning

Bayesian Learning
Neural Networks

Bayesian Learning

- Combines prior knowledge with evidence to make predictions
- Optimal classifier (not practical)
  - Need to know all true probabilities
- Naïve Bayes classifier (practical)
  - Assumes independence among features

Bayes Rule

$$P(C_i \mid \boldsymbol{x}) = \frac{P(\boldsymbol{x} \mid C_i)P(C_i)}{P(\boldsymbol{x})}$$

Thomas Bayes

- C_i is the class, $1 \le i \le K$
- $\boldsymbol{x}$ is the feature values of an instance
- $P(C_i \mid \boldsymbol{x})$ = probability that instance $\boldsymbol{x}$ belongs to class C_i (*posterior*)
- $P(\boldsymbol{x} \mid C_i)$ = probability that an instance drawn from class C_i would be $\boldsymbol{x}$ (*likelihood*)
- $P(C_i)$ = probability of class C_i (*prior*)
- $P(\boldsymbol{x})$ = probability of instance $\boldsymbol{x}$ (*evidence*)

Bayes Rule: Family Car

$$P(FamilyCar \mid EnginePower, Price) = \frac{P(EnginePower, Price \mid FamilyCar)P(FamilyCar)}{P(EnginePower, Price)}$$

Bayes Classifier

- Classify instance $\boldsymbol{x}$ as class C_i such that

$$i = \arg\max_{1 \le k \le K} P(C_k \mid \boldsymbol{x})$$

- Since only interested in maximum, can ignore denominator $p(\boldsymbol{x})$ (i.e., ignore normalization α)

$$i = \arg\max_{1 \le k \le K} P(\boldsymbol{x} \mid C_k) P(C_k)$$

- If prior probability distribution of classes is uniform, then can ignore $P(C_i)$

$$i = \arg\max_{1 \le k \le K} P(\boldsymbol{x} \mid C_k)$$

Bayes Classifier

- Practical issue #1
  - $P(\mathbf{x}|\, C_i)$ is a joint probability distribution
  - Need to know the probability of every possible instance given every possible class
  - Even for D boolean attributes and K classes, that's $K*2^D$ probabilities
- Solution
  - Assume attributes are independent of each other

$$p(x_1, x_2, \ldots, x_D \mid C_i) = \prod_{j=1}^{D} p(x_j \mid C_i)$$

Naïve Bayes Classifier

- Given training set X
- Estimate probabilities from X

$$P(C_i) = \frac{\left|\left\{(\boldsymbol{x},y) \in X \mid y = C_i\right\}\right|}{|X|}$$

$$P\left(x_j = v \,\middle|\, C_i\right) = \frac{\left|\left\{(\boldsymbol{x},y) \in X \mid x_j = v \wedge y = C_i\right\}\right|}{\left|\left\{(\boldsymbol{x},y) \in X \mid y = C_i\right\}\right|}$$

- Classify new instance $\boldsymbol{x}$ as class C_i such that

$$i = \arg\max_{1 \le k \le K} P(C_k) * \prod_{j=1}^{D} P(x_j \mid C_k)$$

Example

Price	EnginePower	FamilyCar
10000	100	yes
10000	200	yes
10000	300	no
20000	100	yes
20000	300	no
30000	100	no
30000	200	no
30000	300	no

- P(FamilyCar)
  - P(FamilyCar=yes) = 3/8
  - P(FamilyCar=no) =
- P(Price |FamilyCar)
  - P(Price=10000 | FamilyCar=yes) = 2/3
  - P(Price=20000 | FamilyCar=yes) =
  - P(Price=30000 | FamilyCar=yes) =
  - P(Price=10000 | FamilyCar=no) =
  - P(Price=20000 | FamilyCar=no) =
  - P(Price=30000 | FamilyCar=no) =
- P(EnginePower | FamilyCar)
  - P(EnginePower=100 | FamilyCar=yes) = 2/3
  - P(EnginePower =200 | FamilyCar=yes) =
  - P(EnginePower =300 | FamilyCar=yes) =
  - P(EnginePower =100 | FamilyCar=no) =
  - P(EnginePower =200 | FamilyCar=no) =
  - P(EnginePower =300 | FamilyCar=no) =
- P(FamilyCar=yes | Price=20000 ^ EnginePower=200) = ?

Example (cont.)

- P(FamilyCar=yes | Price=20000 ^ EnginePower=200) = ?

Naïve Bayes Classifier

- Practical issue #2
  - What if no examples in class C_i have $x_j = v$?

$$P(x_j = v \mid C_i) = 0$$

$$P(C_i) * \prod_{j=1}^{D} P(x_j \mid C_i) = 0$$

- Solution

$$p(x_j = v \mid C_i) = \frac{|\{(\boldsymbol{x}, y) \in X \mid x_j = v \, and \, y = C_i\}| + 1}{|\{(\boldsymbol{x}, y) \in X \mid y = C_i\}| + |values(x_j)|}$$

Another Example

- P(FamilyCar=yes | Price=30000 ^ EnginePower=200) = ?
  = α * P(Price=30000 | FamilyCar=yes) *
  P(EnginePower=200 | FamilyCar=yes) *
  P(FamilyCar=yes)
  = α (0/3)(1/3)(3/8) = ?

- values(Price) = {10000, 20000, 30000}
- | values(Price) | = 3

- P(Price=30000 | FamilyCar=yes) = ?

Naïve Bayes Classifier

- Practical issue #3
  - What if x_j is a continuous feature?
- Solution 1
  - Assume some parameterized distribution for x_j
    - E.g., normal
  - Learn parameters of distribution from data
    - E.g., mean and variance of x_j values
- Solution 2
  - Discretize feature
  - E.g., price $\in \Re$ to price $\in$ {low, medium, high}

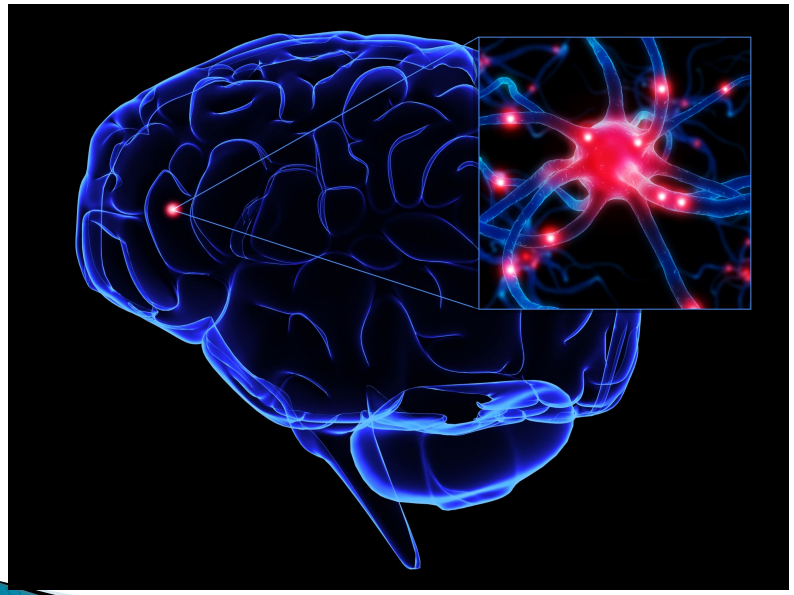
Bayesian Learning: Summary

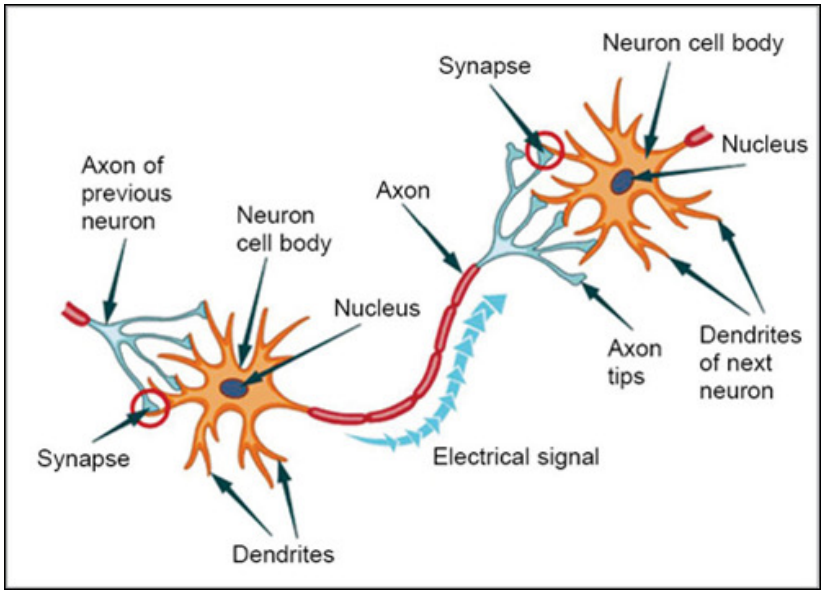
- Optimal learner, in theory
- Naïve Bayes learner, practical
  - Independence assumption rarely true
    - E.g., Is "price" independent of "engine power"?
  - Naïve Bayes learner still does surprisingly well
  - Simple, effective baseline for other learners

Neural Networks

Neural Networks

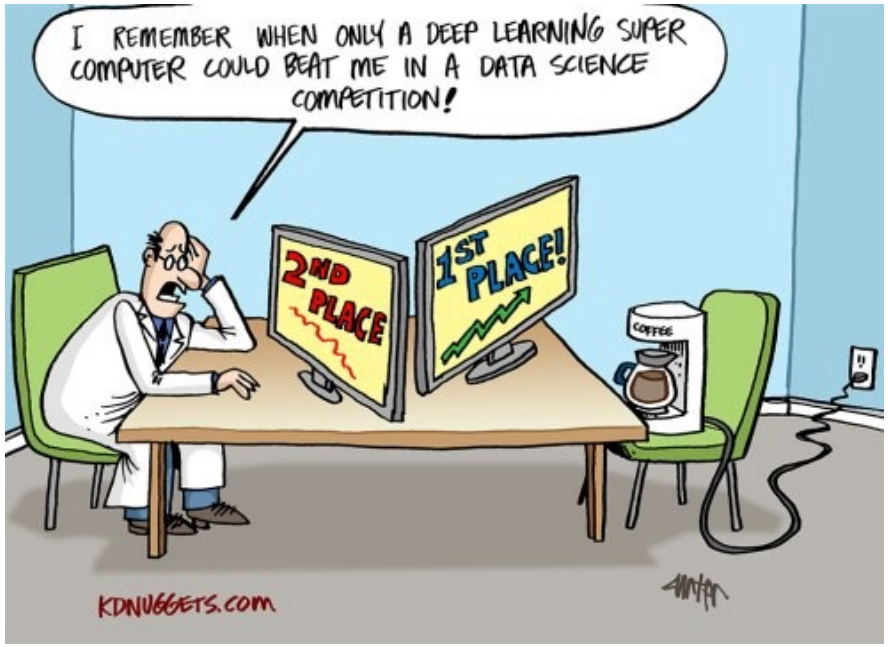
- Inspired from brain
  - Brain consists of interconnected neurons
  - Humans still outperform machines in most areas of intelligence



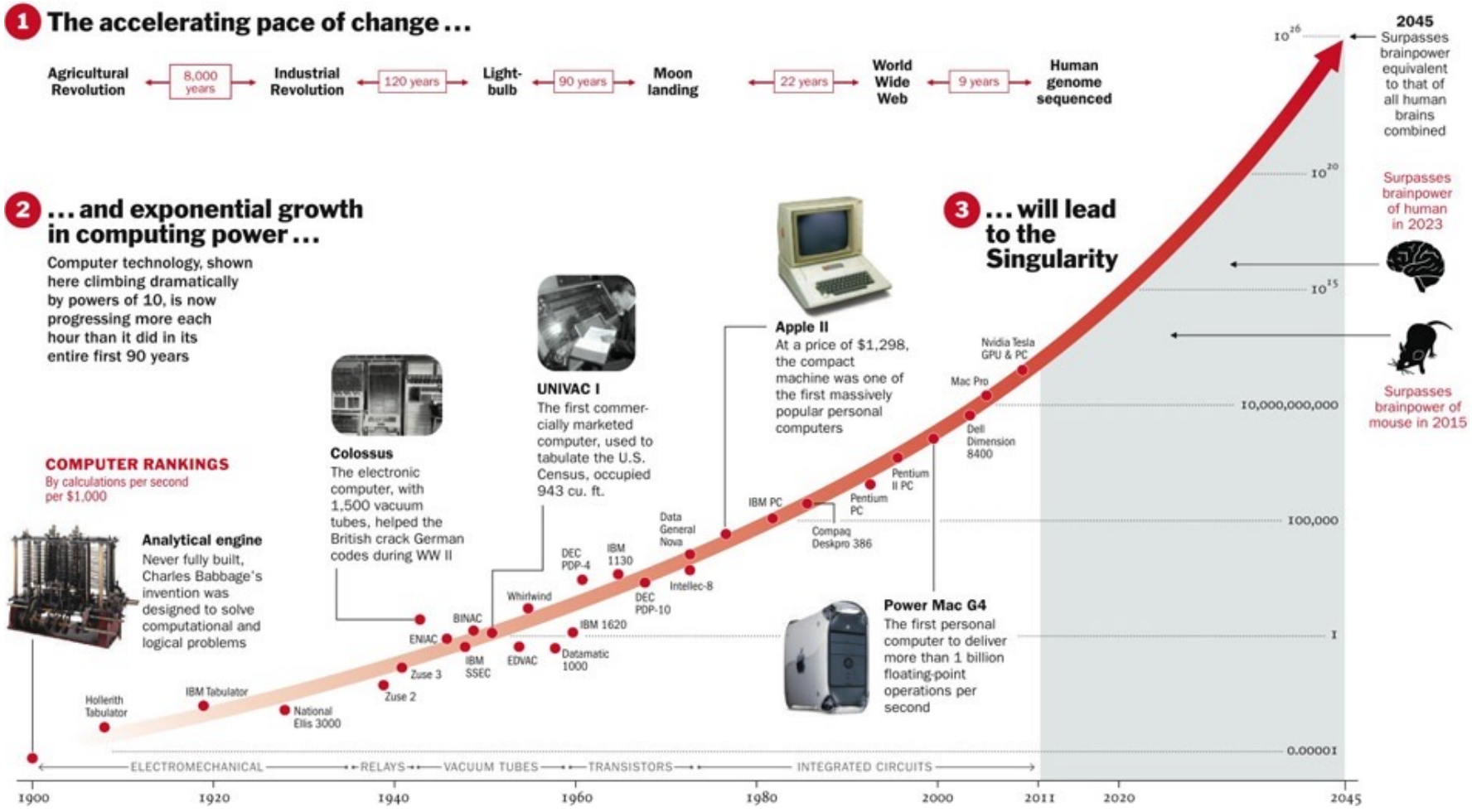


Human Brain vs. Computer

- Processors
  - Computer
    - CPUs: 20 cores (10^9 Hz)
    - GPUs: 10^4 cores (10^9 Hz)
  - Human brain
    - 10^{11} neurons (10^3 Hz)
- Parallelism
  - Computer
    - CPUs some, GPUs lots
    - Few interconnections
  - Human brain
    - 10^{15} synapses
    - Each neuron has ~10^4 connections to other neurons



The Singularity

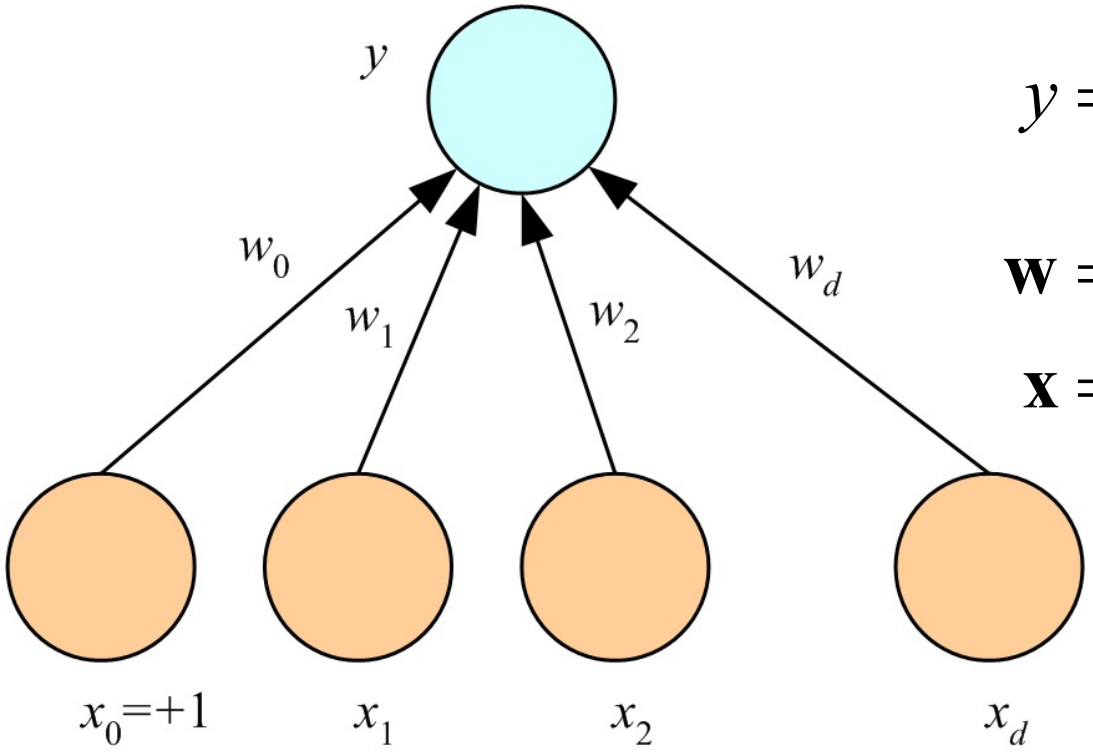


Time Magazine (Feb 2011)

What Happens Then…?

Terminator 2 Judgment Day (1991)

Perceptron (Rosenblatt, 1962)



$$y = \sum_{j=1}^{d} w_j x_j + w_0 = \mathbf{w} \cdot \mathbf{x}$$

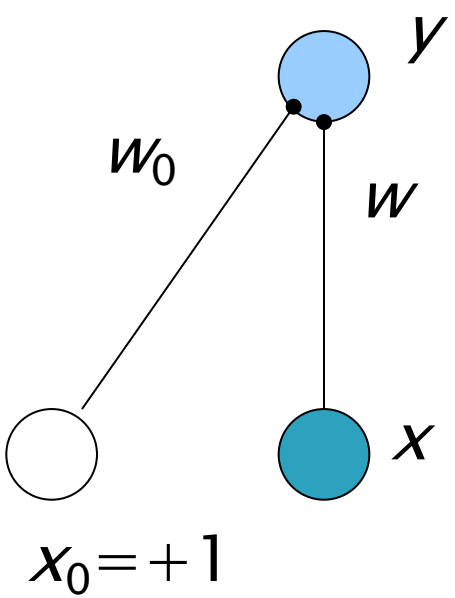
$$\mathbf{w} = \left[w_0, w_1, \ldots, w_d \right]$$

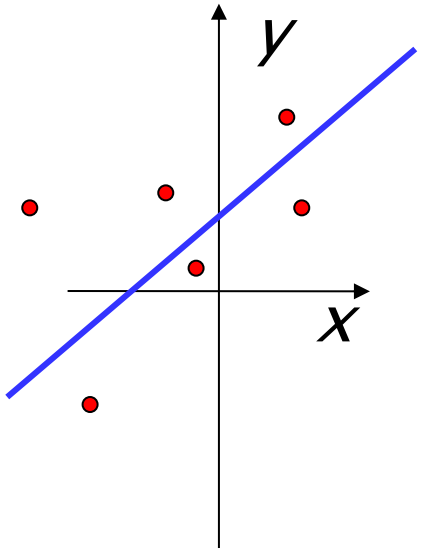
$$\mathbf{x} = \left[1, x_1, \ldots, x_d \right]$$

Learn function f
Modify weights so y = f(x)

Perceptron Regression

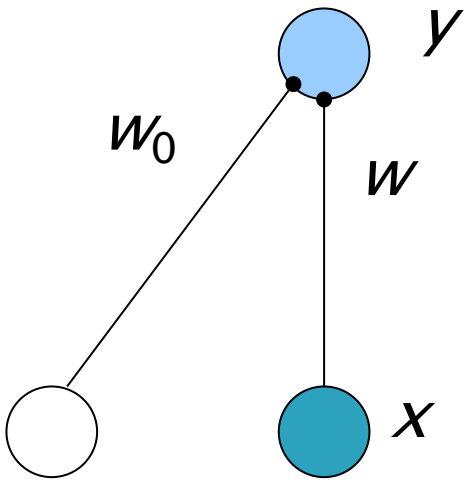
- $y = wx + w_0$

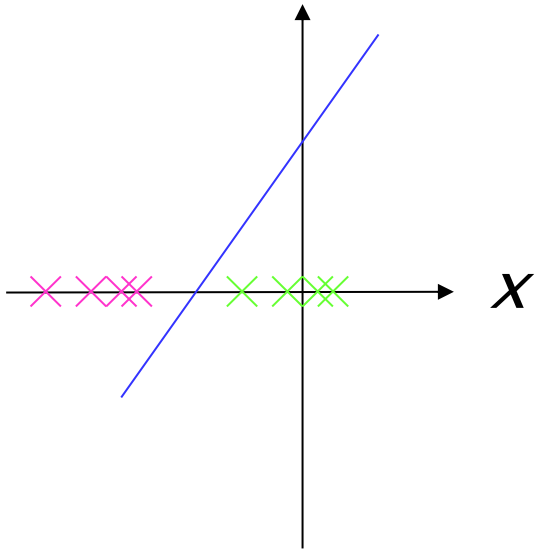




Perceptron Classification

- If ($wx + w_0 > 0$) Then y=1 Else y=0





Perceptron Training

- Change weights to reduce error
- Gradient descent

$$\Delta w_i = \eta(y_j - o_j)x_{ji}$$

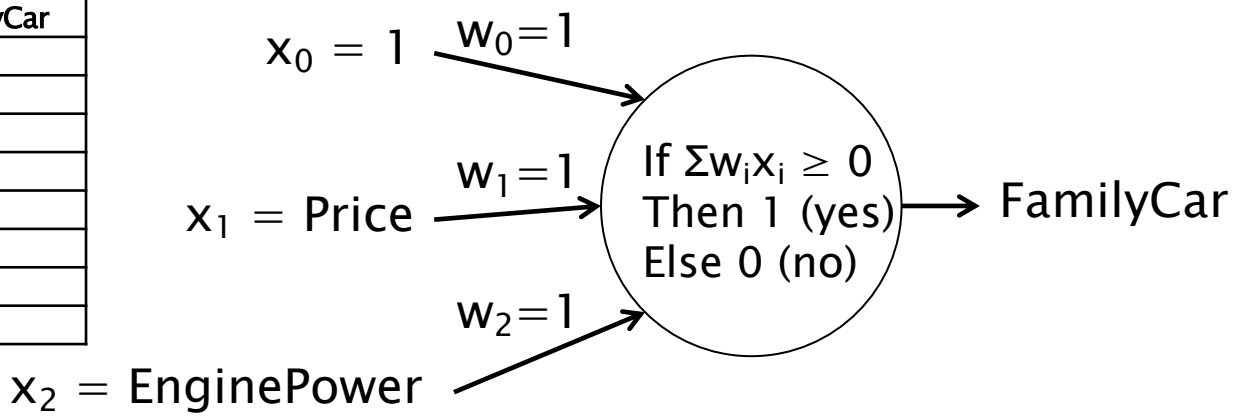
  - o_j = output of perceptron for example j
  - Learning rate η controls rate of descent

Example

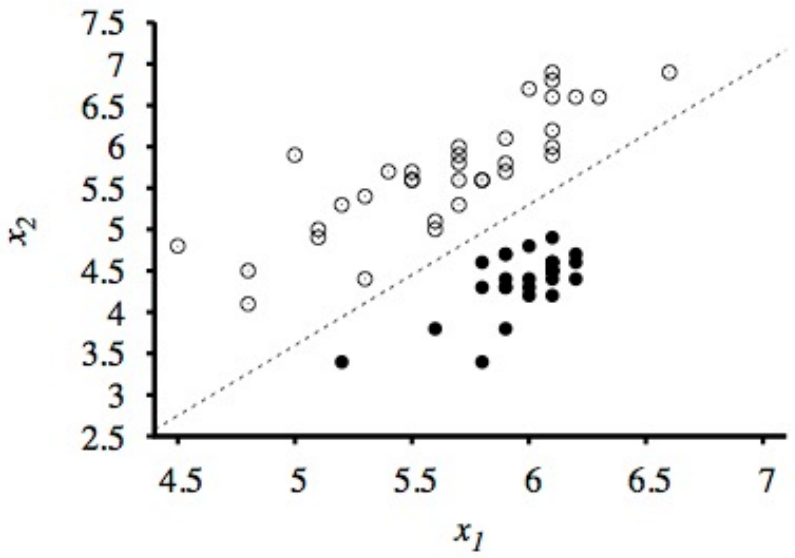
Price	EnginePower	FamilyCar
1	1	1
1	2	1
1	3	0
2	1	1
2	3	0
3	1	0
3	2	0
3	3	0

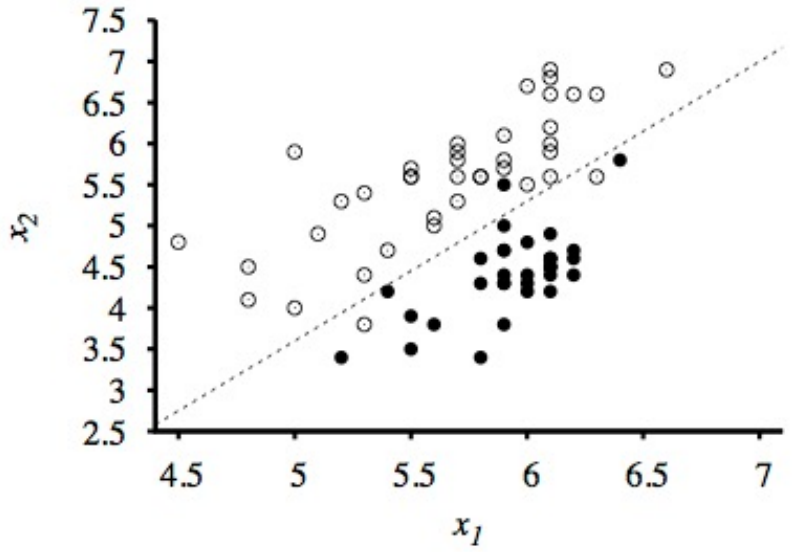
$$\Delta w_i = \eta(y_j - o_j)x_{ji}$$

Let $\eta = 0.5$



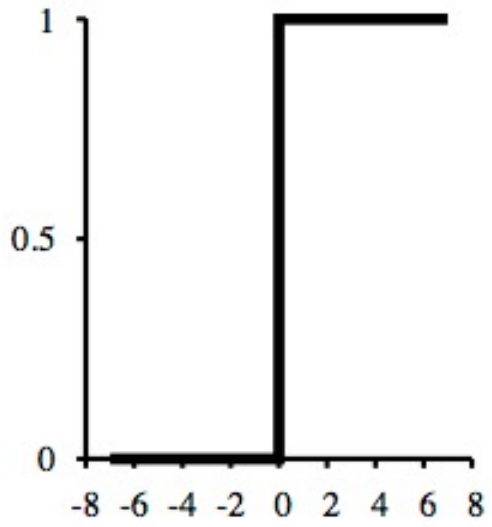
Linearly Separable

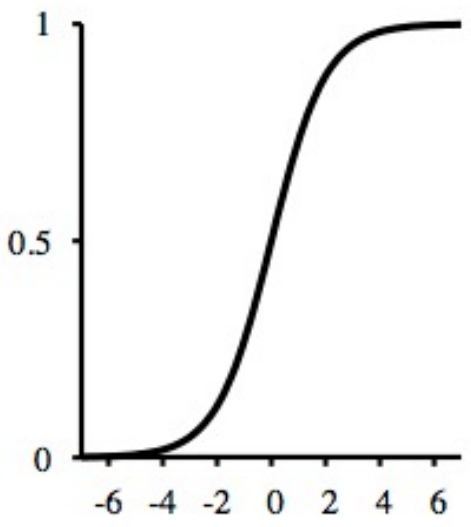




Logistic Function

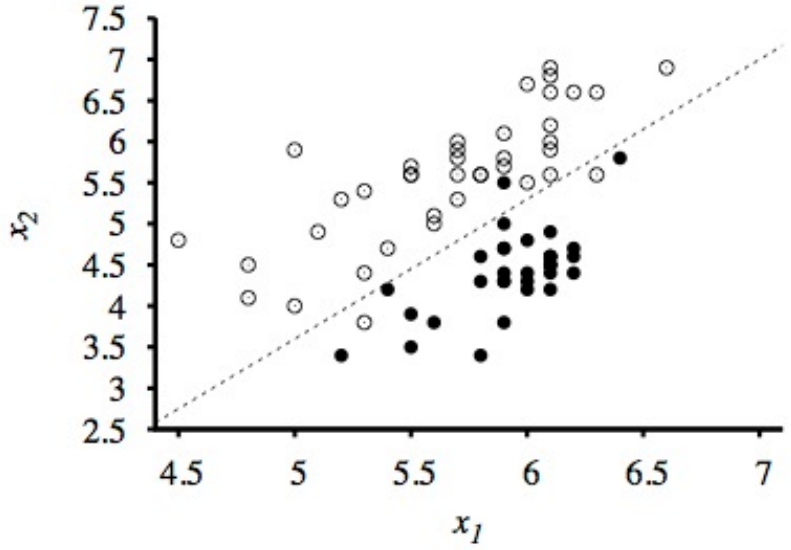
- Also called “sigmoid function”

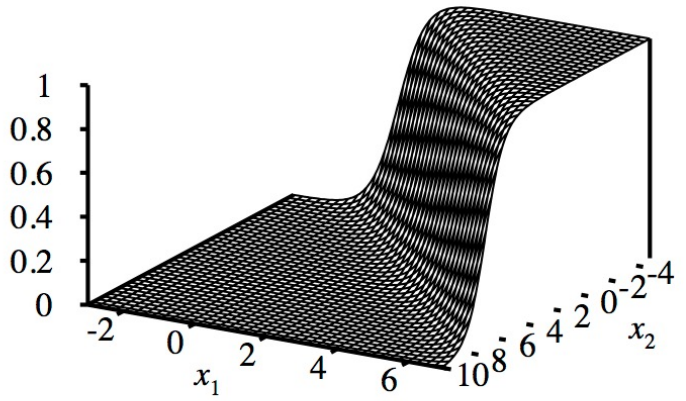




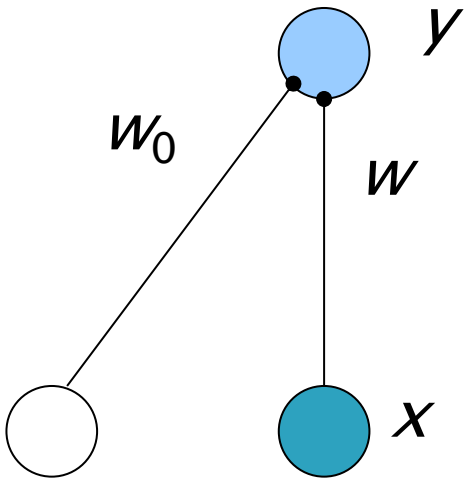
$$\text{logistic}(z) = \frac{1}{1 + e^{-z}}$$

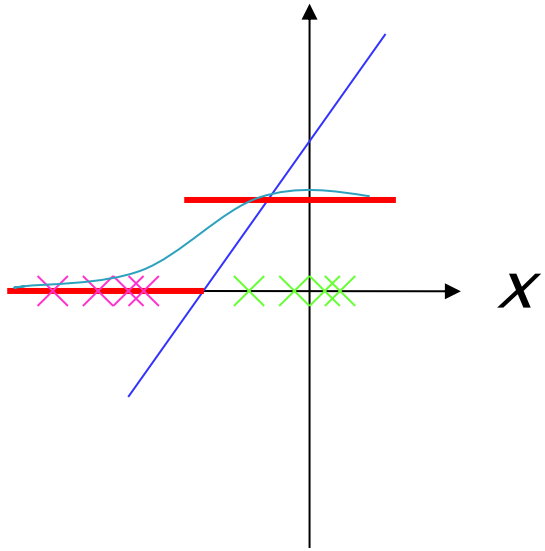
Logistic Function





Perceptron Classification

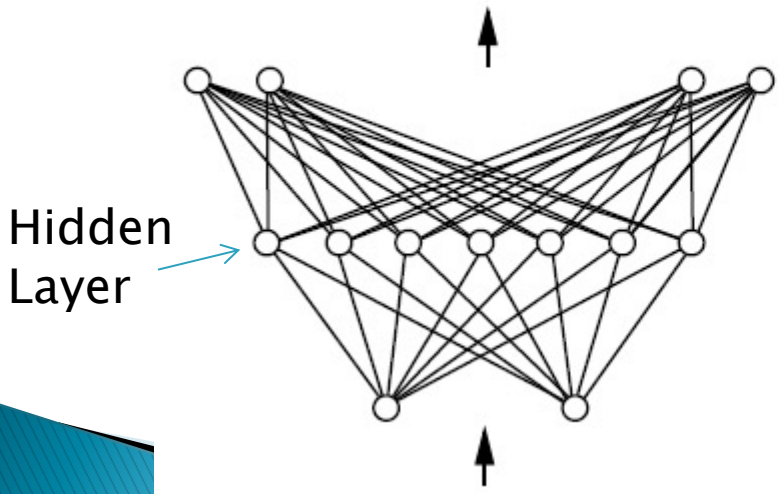


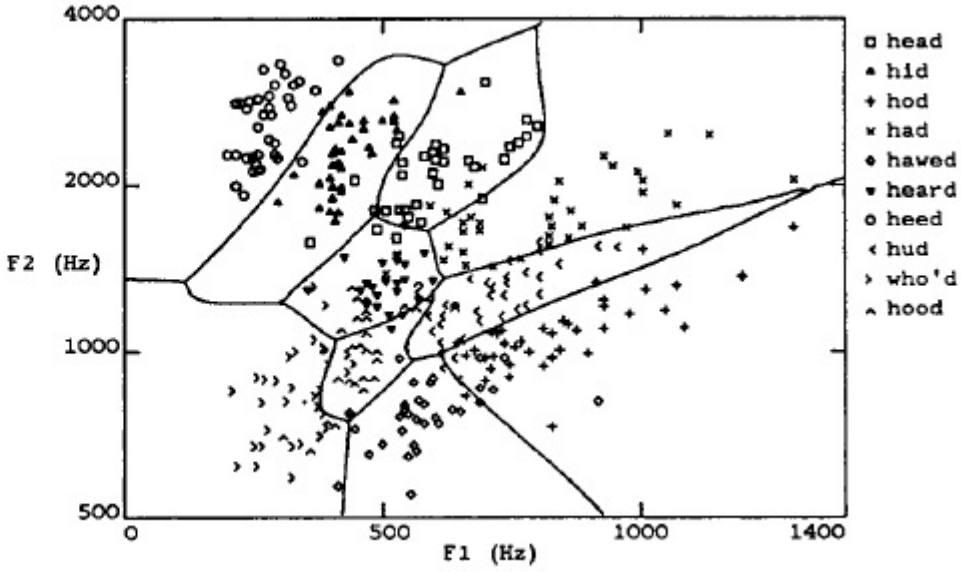


$$y = \text{logistic}\left(\mathbf{w} \cdot \mathbf{x}\right) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

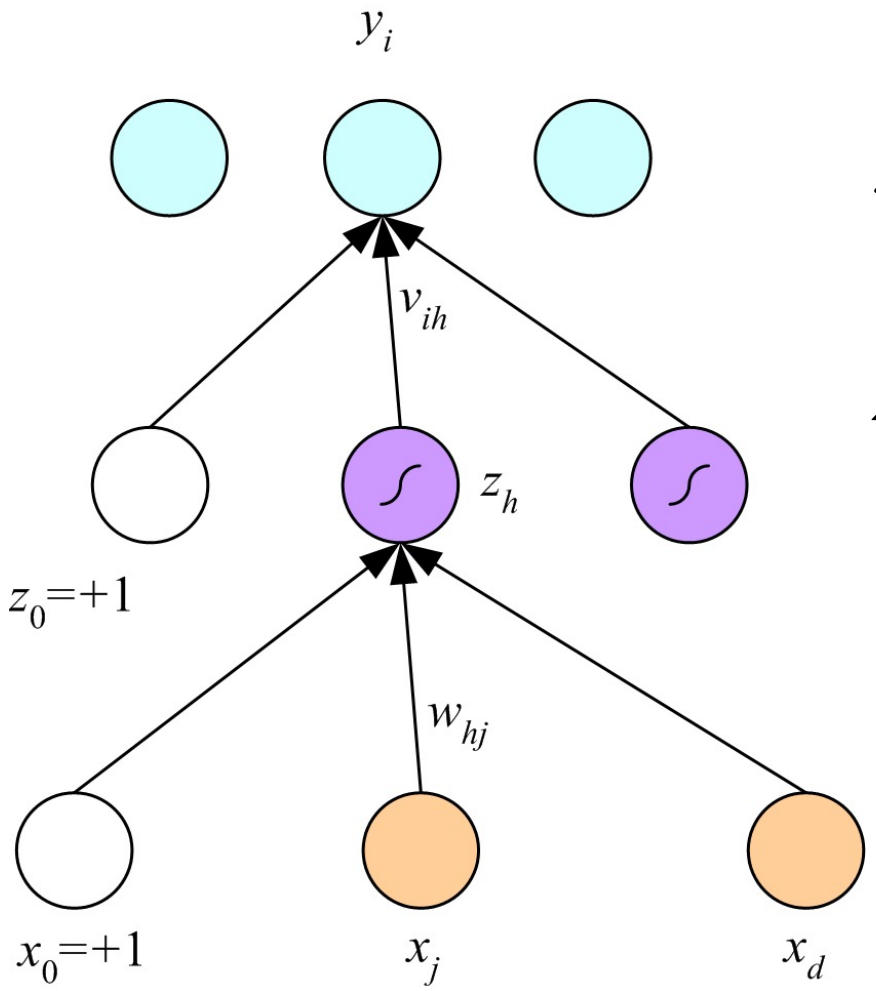
Learning Nonlinear Functions

- Perceptrons can only approximate linear functions
- But multiple layers of logistic perceptrons can approximate any nonlinear function





Multilayer Perceptrons



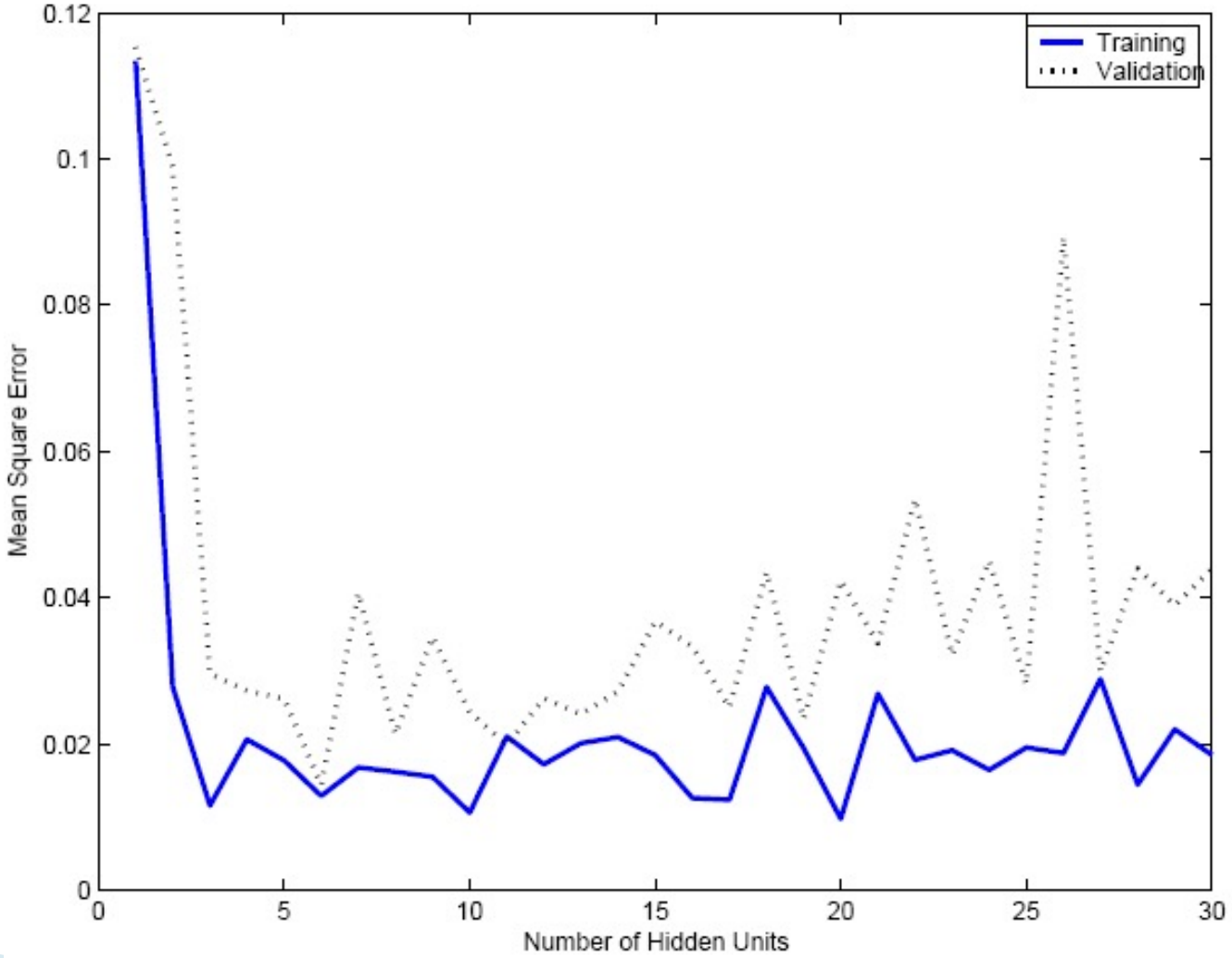
$$y_i = \mathbf{v}_i \cdot \mathbf{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

$$z_h = \text{logistic}\left(\mathbf{w}_h \cdot \mathbf{x}\right) = \frac{1}{1+\exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}$$

Learn weights by error backpropagation. (Rumelhart et al., 1986)

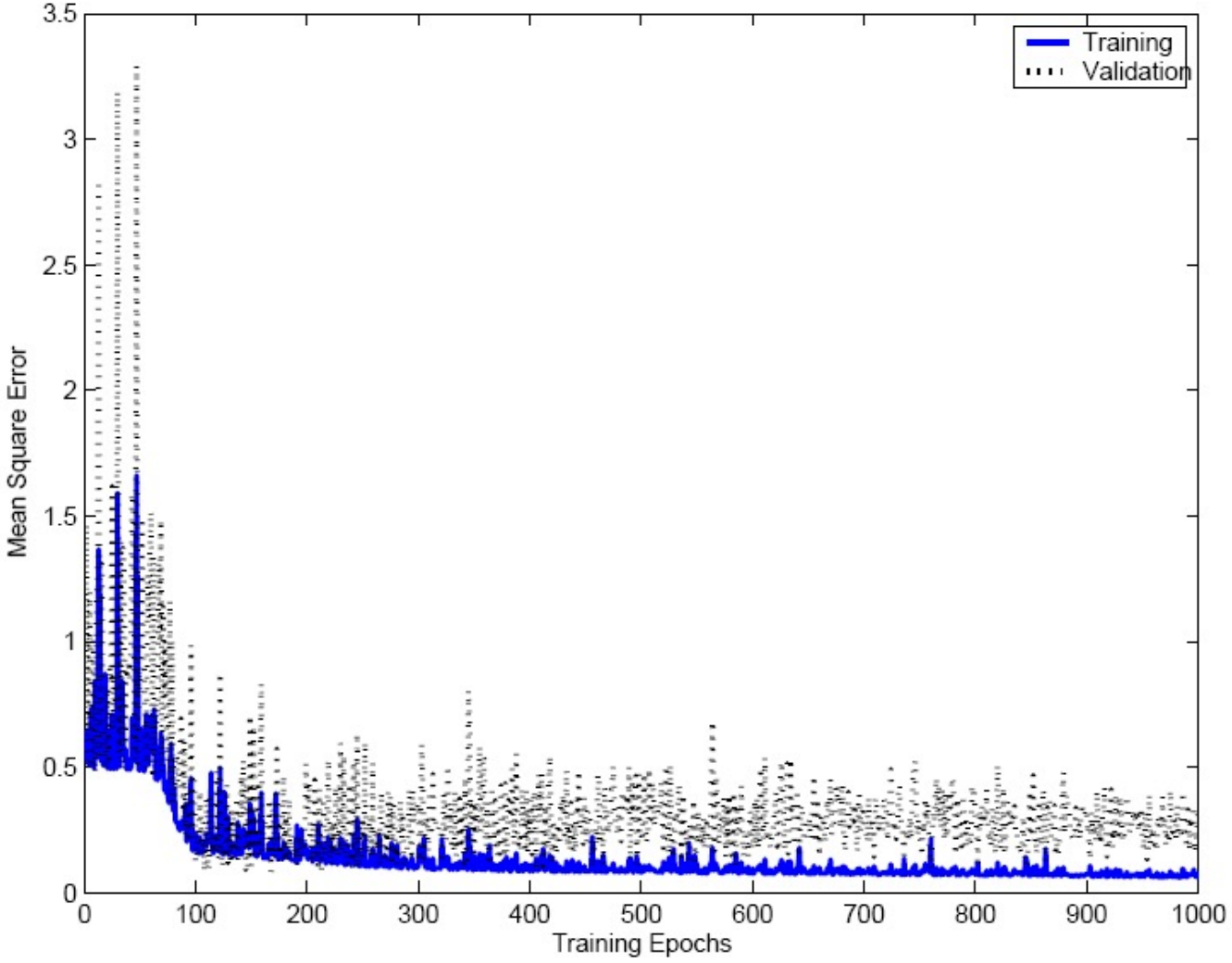
Overfitting in MLPs

f(x) = sin(6x)



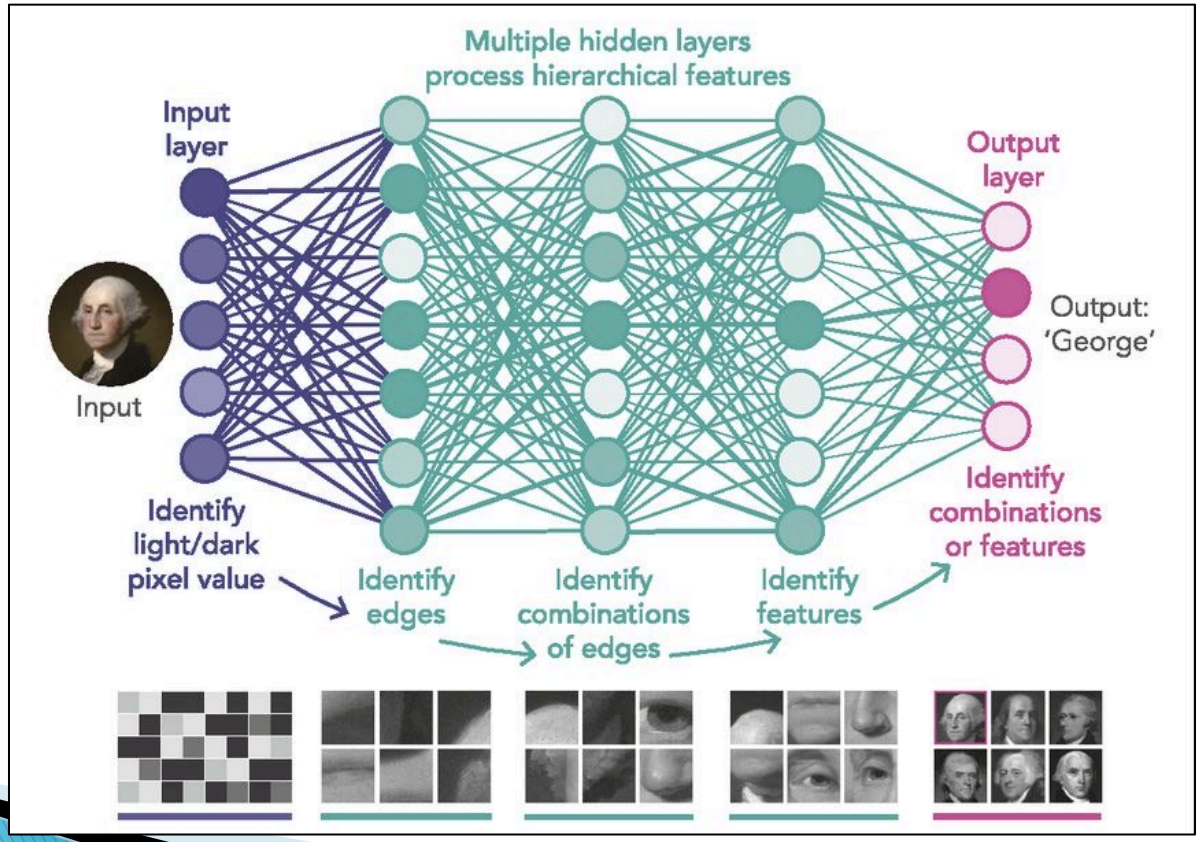
Overfitting in MLPs

- Similar overfitting behavior if training continued too long
- More and more weights move from zero
- Overtraining

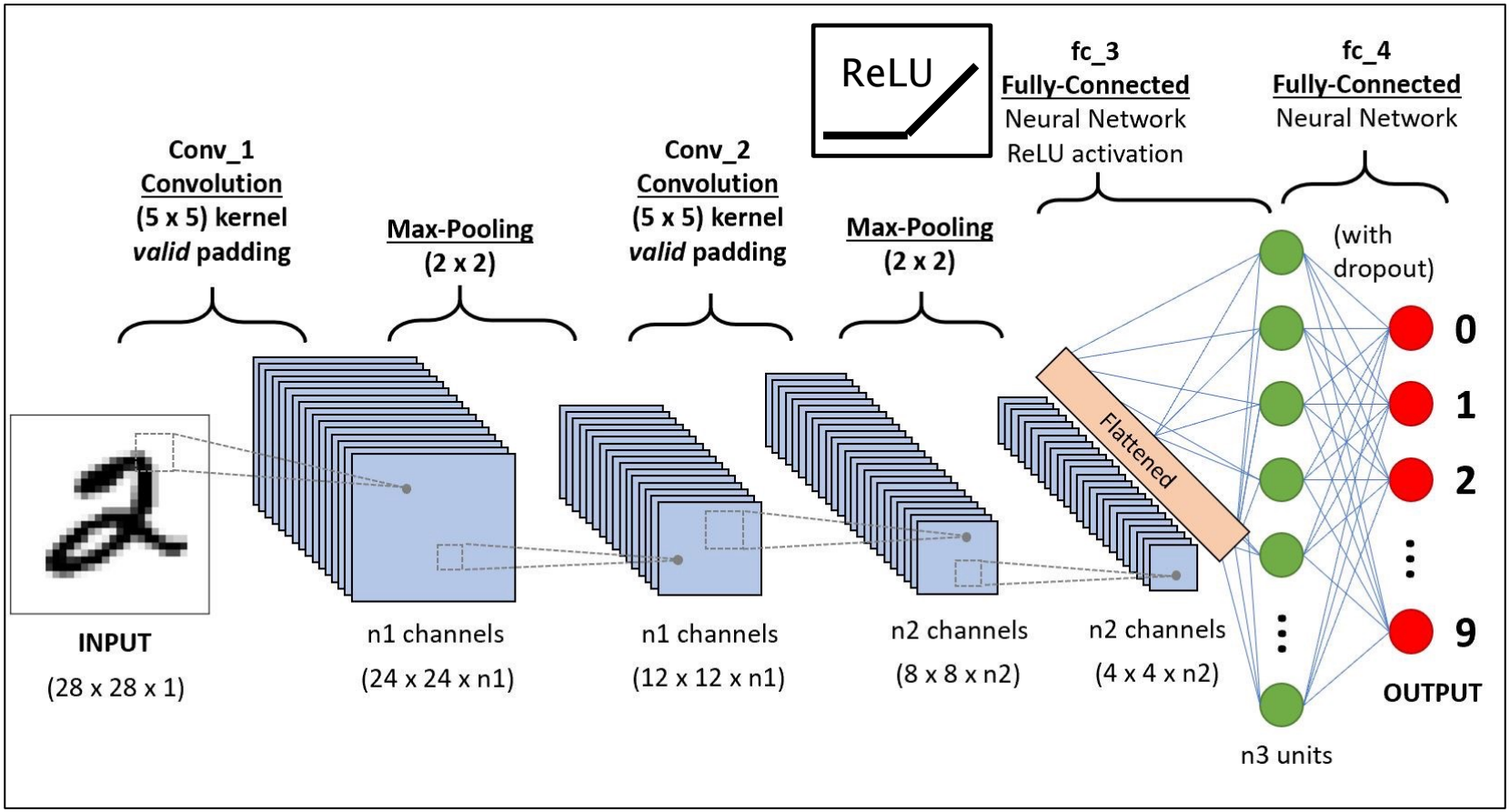


Deep Learning

- Using deep (many-layered) neural nets to learn complex abstractions (features)



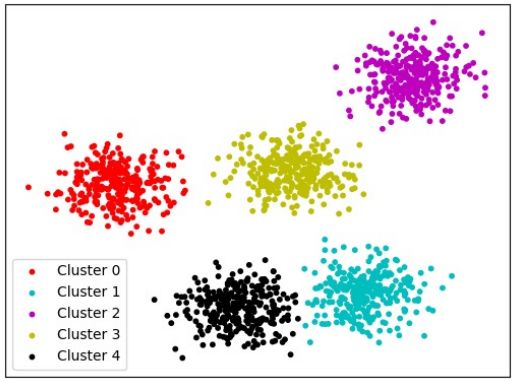
Deep Convolutional Neural Net (CNN) for Image Classification



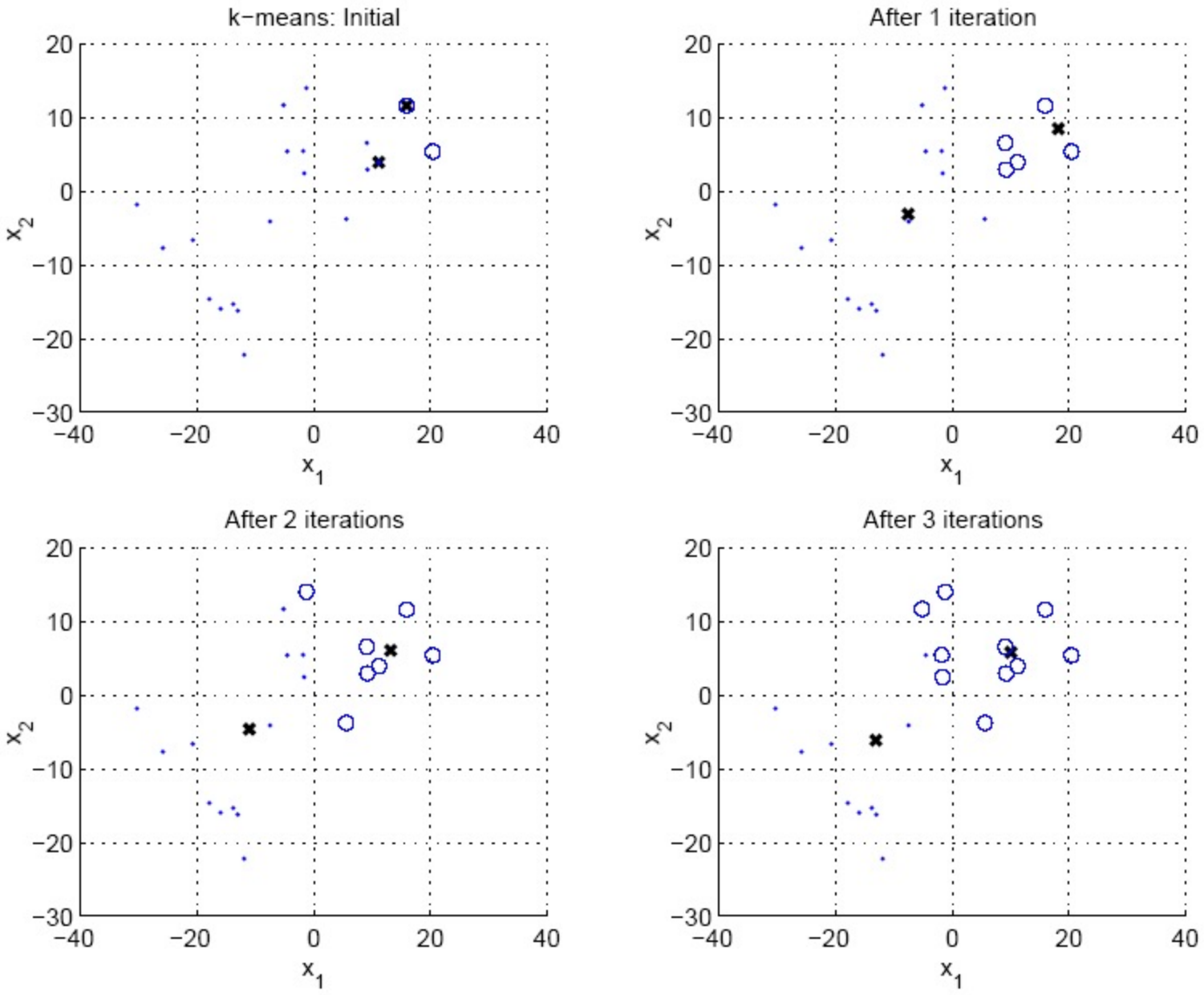
Unsupervised Learning

k-Means Clustering

k-means Clustering

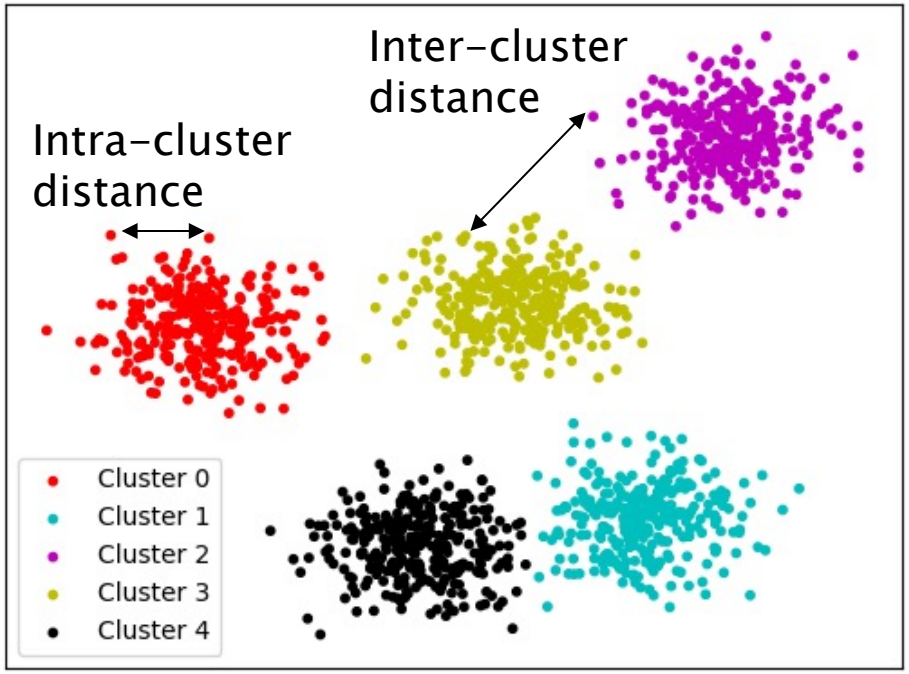


- Unsupervised learning
  - Just feature values, no classes
- Partition instances into *k* disjoint sets
- Each set has a representative instance m_i
- Place instance *x* into set *i* such that *distance*(x, m_i) is minimal
- Choose new central m_i for each set
- Repeat until m_i converge


k−means: Initial
After 1 iteration
After 2 iterations
After 3 iterations
x1
x2

k-means Clustering

- Choosing k ?
  - Try several ($2 \leq k \leq N$)
  - Choose k minimizing intra-cluster distance and maximizing inter-cluster distance

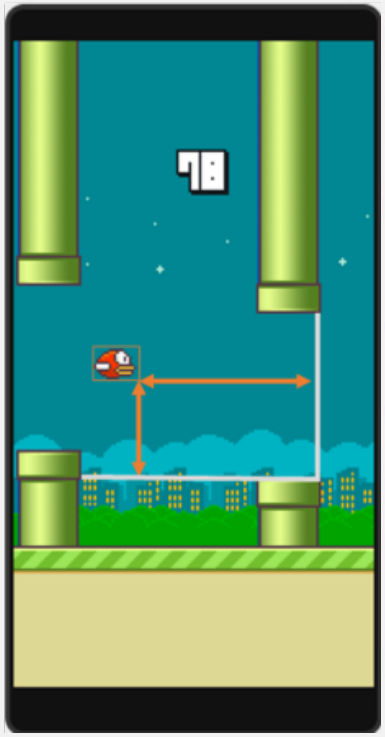


ML Software Tools

- Waikato Environment for Knowledge Analysis (WEKA)
  - Java based
  - http://www.cs.waikato.ac.nz/ml/weka
- SciKit Learn
  - Python based
  - http://scikit-learn.org
- Deep Learning (mostly Python)
  - TensorFlow (https://www.tensorflow.org)
  - Keras (https://keras.io)

Reinforcement Learning

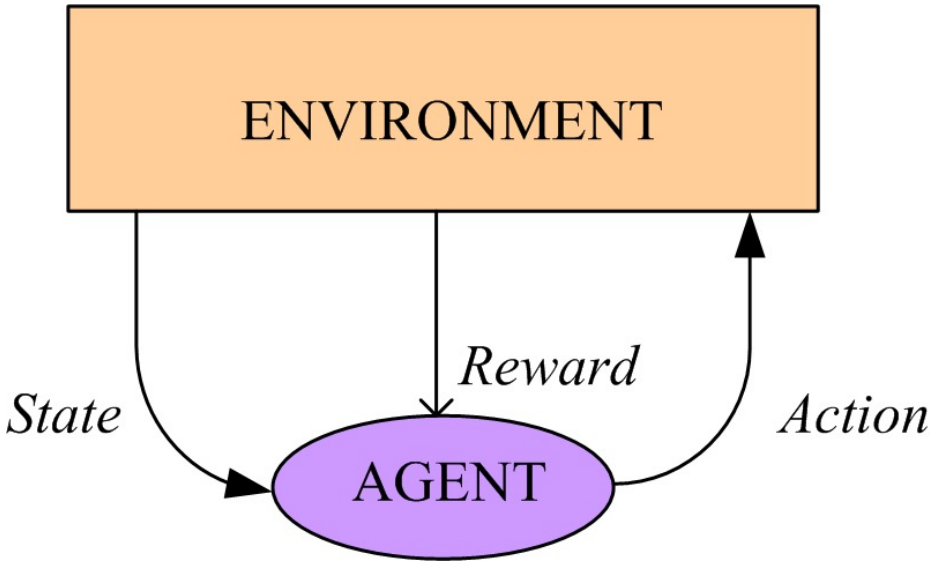
Reinforcement Learning



sarvagyavaish.github.io/FlappyBirdRL

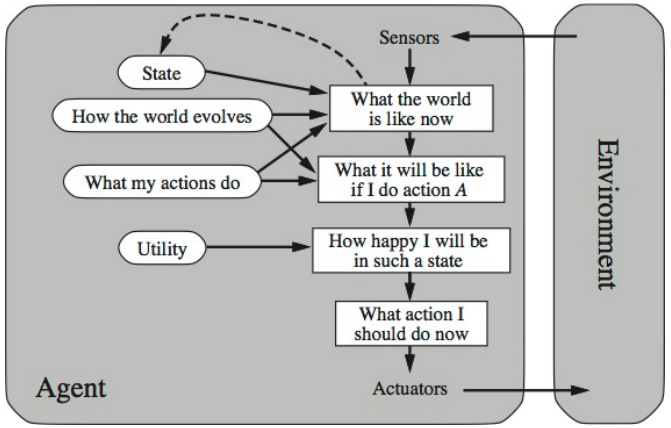
Reinforcement Learning

- Agent in some state in the environment, takes an action and sometimes receives reward, and the state changes
- Delayed reward
- Credit-assignment
- Learn a policy
  - π: State → Action
- Applications
  - Game-playing
  - Robot control



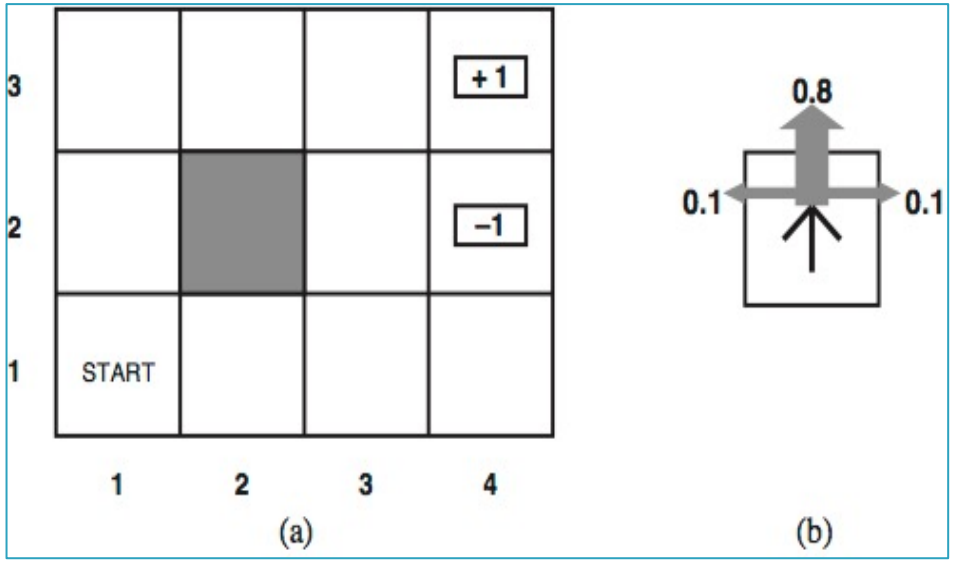
Reinforcement Learning Agents

- Utility-based agent
  - Learn utility function on states
  - Choose actions to maximize expected utility
  - Requires action model
    - Action: state → state
- Q-learning agent
  - Learns Q function: value of taking action A in state S
  - No action model, but learning slower
- Passive learning of utility or Q functions
  - Policy given
- Active learning: Also learn policy
  - Exploration



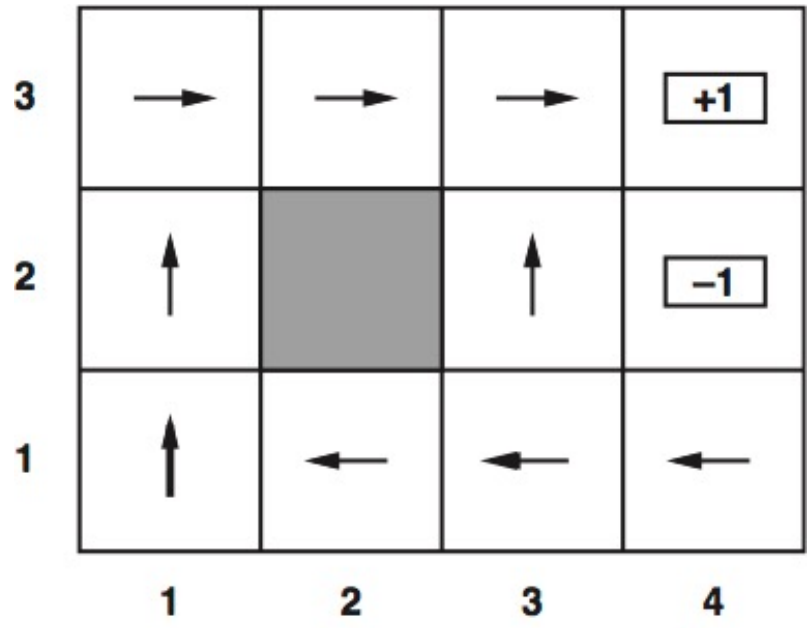
Simple Environment

- Actions: Up, Down, Left, Right
- 80% chance intended move executed
- 20% chance agent moves 90 degrees from intended direction
- Agent bumps into walls and obstacle
- Reward
  - +1 for reaching (4,3)
  - -1 for reaching (4,2)
  - -0.04 for other moves
- Fully observable



Passive Reinforcement Learning

- Given policy $\pi(s)$
- Learn utility function $U^{\pi}(s)$



Policy

3	0.812	0.868	0.918	+1
2	0.762		0.660	−1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Utilities

Utility Estimation

- Utility of state s is the reward in state s plus the expected utility of its successor states

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) U^{\pi}(s')$$

- γ is the discount factor ($0 < \gamma \leq 1$)

Utility Estimation: Example

- Let $\gamma=1$
- Using previous example and policy
  - Initially all $U(s)=0$
  - Except $U([4,3])=1$ and $U([4,2])=-1$
- $U([3,3]) = ?$, policy says go Right

- Eventually, $U([3,3])$
  - $= (-0.04) + (0.8)*(1) + (0.1)*(0.918) + (0.1)*(0.660)$
  - $= 0.918$

Temporal Difference Learning

- Observe transition from s to s'
- Adjust utility of state s to be closer to utility of state s'

$$U^{\pi}(s) = U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

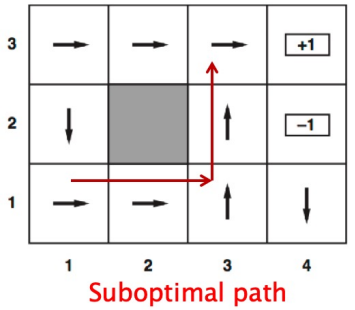
- Use learning rate α to control rate of adjustment
- If α decreases with the number of times state s is visited, then TD will converge to correct $U^{\pi}(s)$

Active Reinforcement Learning

- Learn policy $\pi(s)$
- Agent in state s takes action maximizing expected utility

$$\pi(s) = arg\max_{a} \sum_{s'} P(s'|s,a) * U(s')$$

- Initial actions random
- Estimate P(s' | s, a) and U(s) over time
- Can result in suboptimal policy



Exploration

- Force agent to explore unvisited states to avoid suboptimal policies
- Exploration function

$$f(u,n) = \begin{cases} R^+ & if\ n < N_e \\ u & otherwise \end{cases}$$

  - u is the utility estimate so far
  - n is the number of times state s has been visited
  - R^+ is estimate of best possible reward
  - N_e is the number of times an action-state must be tried before relying on utility estimate

Q-Learning

- Active temporal-difference learning agent
- Define Q(s,a) as the value of taking action a in state s

$$U(s) = \max_a Q(s,a)$$

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a')$$

- Q function combines π(s), P(s'|s,a) and U(s)

Q-Learning

- Use TD approach to eliminate P(s'|s,a)

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

- Calculated when observe transition from s to s' using action a

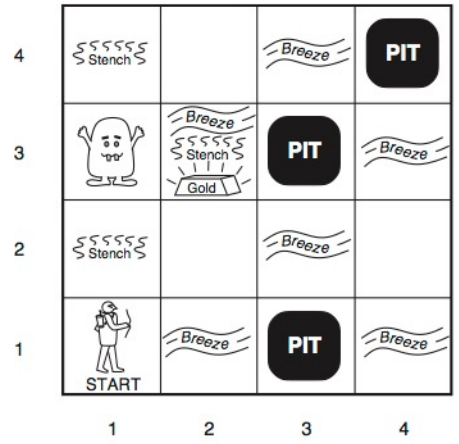
Q-Learning Agent

function Q-LEARNING-AGENT (*percept*) **returns** an action
  inputs: *percept*, a percept indicating current state s' and reward signal r'
  persistent: Q, table of action values indexed by state and action, initially zero
    N_{sa}, table of frequencies for state-action pairs, initially zero
    s, a, r, previous state, action and reward, initially null
  if TERMINAL?(s) **then**
    foreach $a' \in$ ACTIONS(s)
      $Q[s,a'] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s,a]$
    $Q[s,a] \leftarrow Q[s,a] + \alpha(N_{sa}[s,a])(r + \gamma \max_{a'} Q[s',a'] - Q[s,a])$
  $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s',a'], N_{sa}[s',a']), r'$
  return a

Exploration function f(u,n)

Learning rate (decreasing with n): $\alpha(n) = C / (C + n - 1)$

Wumpus World

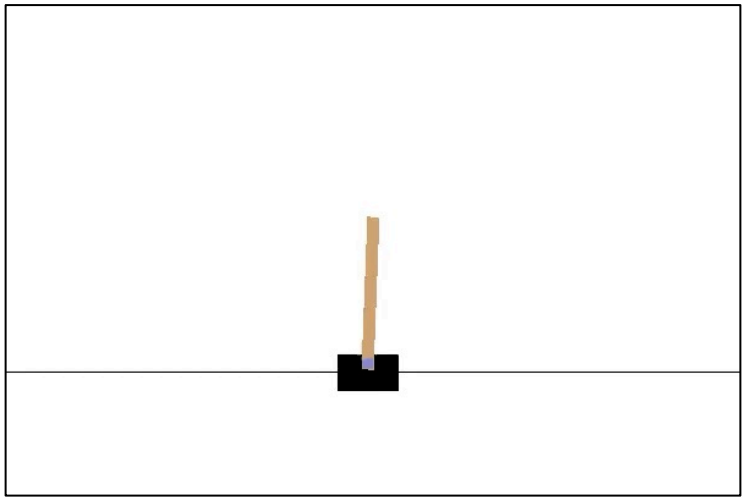


- Q-learning agent
- Limited to 4x4 worlds
- 512 states
  - 16 locations, 4 orientations, hasGold, hasArrow, inCave
- 6 actions
- Parameters
  - Best possible reward $R^+ = 1000$
  - Minimum state-action occurrences $N_e = 5$
  - Discount factor $\gamma = 0.9$
  - Learning rate $\alpha(n) = 100 / (99 + n)$

RL Software Tools

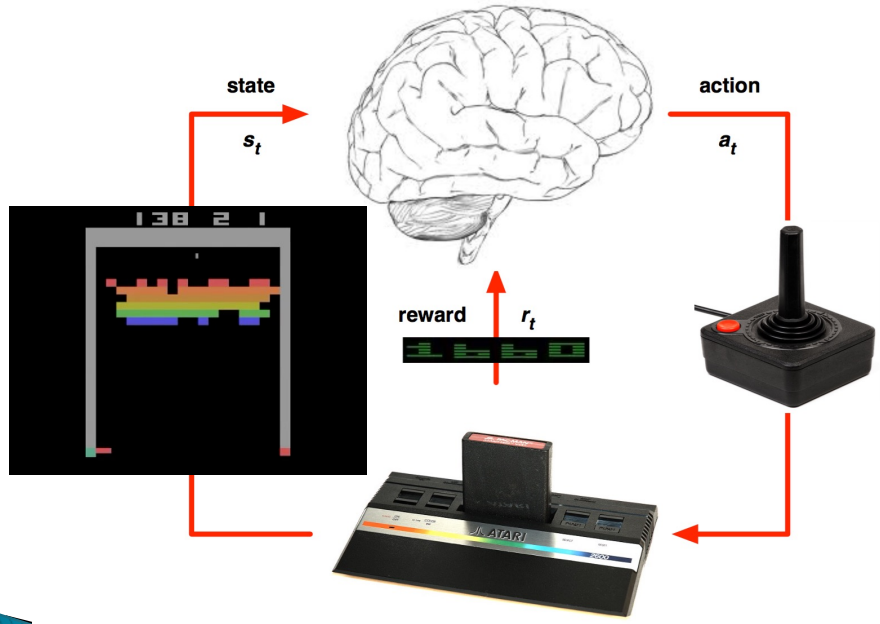
- OpenAI Gym
  - RL environments
  - gym.openai.com

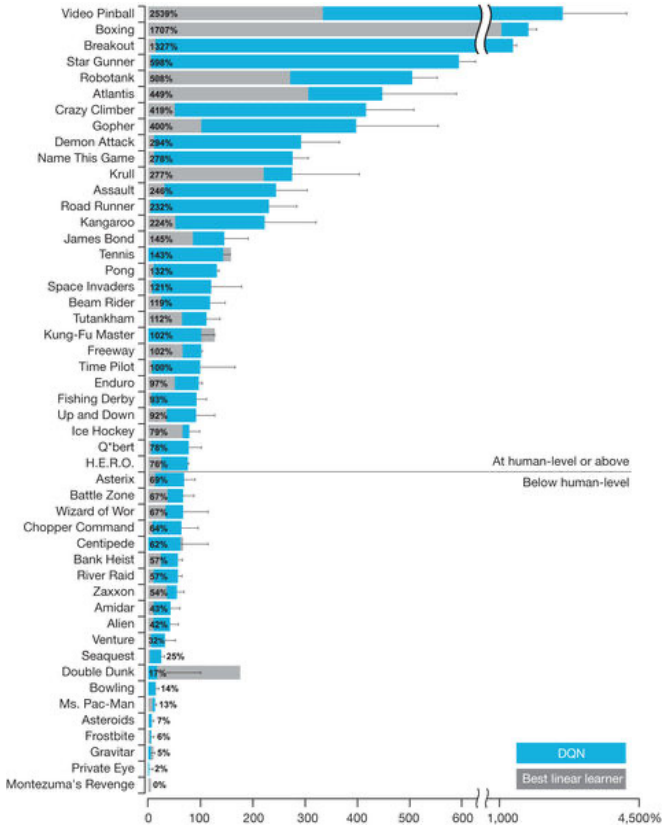
Cart Pole



Deep Reinforcement Learning

- Deep Q-Learning Network (DQN)
  - Use Deep Learning Network to learn Q function
  - Agent “replays” past games





Summary: Learning

- Improving performance at some task through experience
- Supervised learning methods
  - Naïve Bayes
  - Neural network
- How to choose the right model?
  - Overfitting
- Unsupervised learning
  - Clustering
- Reinforcement learning