Learning Agent

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

- How is knowledge represented?
- How is experience represented?
- What is the performance measure?
- Knowledge acquisition vs. skill acquisition
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," Mr. Gates responded, "that is worth 10 Microsofts." (Quoted in NY Times, Monday March 3, 2004)
Applications

- Medical diagnosis
- Autonomous control (planes, trains, automobiles, robotics)
- Perception (speech, language, images, video)
- Recommendations (Amazon, Netflix)
- Prediction (business, financial, environment, health, energy, security, ...)
- Fraud/intrusion detection
- ...
What Ever Happened to Voyager?

Star Trek The Motion Picture (1979)
Approaches

- Unsupervised Learning
  - Clustering

- Supervised Learning
  - Classification
  - Regression

- Reinforcement Learning
Bank Loan Example

- Debt vs Income graph
- Red triangles represent Default
- Yellow squares represent Good Status
Unsupervised Learning

Categories
1) Debt exceeds Income
2) High Debt, High Income
3) Low Debt

Artificial Intelligence
Supervised Learning

if Debt < a*Income + b then Loan else No Loan

\[\text{Debt} \leq a \times \text{Income} + b \Rightarrow \text{Loan}, \quad \text{otherwise No Loan}\]
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
  - $\pi$: State $\rightarrow$ Action

- Applications
  - Game-playing: Sequence of moves to win a game
  - Robotics: Sequence of actions to achieve a goal
Supervised Learning

- Given training set of N examples of $y = f(x)$
  - $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$
- 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 attributes: “engine power” and “price”
- Given several training examples of already-classified cars
- Output classifier that accurately classifies all cars
Definitions

- **Attribute** \( i \) of \( j^{th} \) example: \( x_{j,i} \)
  - A property of the object to be classified
  - Discrete or continuous
  - E.g., “engine power”, “price”

- **Instance** \( x_j = [x_{j,1}, x_{j,2}, \ldots, x_{j,n}] \)
  - The attribute values for a specific object
  - E.g., “engine power = 100”, “price = high”
Definitions

- **Class C**
  - Discrete set of different y’s (classes)
    - E.g., {“family car”, “other”}
  - Set of instances $x_j$ that all have the same $y_j$

- **Example** $(x_j,y_j)$
  - Instance along with its class membership $y_j = f(x_j)$
  - Positive example: member of class ($y_j = 1$)
  - Negative example: not a member of class ($y_j = 0$)
  - E.g., $(<150,medium>, “family car”)$

- **Training set** $X = \{(x_j,y_j)\}, 1 \leq j \leq N$
  - Set of N examples
Example: “Family Car” Class
Definitions

- **Target concept** \( (C) \)
  - Correct expression of class
  - E.g., \( (e_1 \leq \text{engine power} \leq e_2) \) and \( (p_1 \leq \text{price} \leq p_2) \)

- **Hypothesis** \( h(x) \rightarrow \{0,1\} \)
  - Approximation to target concept

- **Hypothesis class** \( H \)
  - Space of all possible hypotheses
  - E.g., axis-aligned rectangles
  - E.g., axis-aligned ellipses
Definitions

Learning goal
- Find hypothesis $h \in H$ that closely approximates target concept $C$
- $h$ is the output classifier
- Target concept may not be in $H$

Classification: $h(x) \in C$
Regression: $h(x) \in \mathbb{R}$
Example: Hypothesis Error

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

False positive
False negative
Definitions

- Empirical (sample) error
  - How well $h$ classifies training set $X$
    
    $$E(h \mid X) = \frac{1}{N} \sum_{j=1}^{N} 1(h(x_j) \neq y_j)$$

- Generalization error
  - How well $h$ classifies instances not in $X$

- True error
  - How well $h$ classifies all instances $I$
    
    $$E(h) = \frac{1}{|I|} \sum_{x \in I} 1(h(x) \neq C(x))$$
Noise

- Sources
  - Incorrect attribute values
  - Incorrect class labels
  - Hidden or latent attributes

- Impact
  - Overfitting: Trying too hard to fit $h$ to the noise
If A and B are noise, then $h_2$ overfits.

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

- Cross-validation
  - Measure generalization accuracy by testing on data unused during training (validation set)

- Regularization
  - Penalize complex models
  - $E' = \text{error on data} + \lambda \times \text{model complexity}$
Hypothesis Evaluation

- Empirical error too optimistic
- True error usually unobtainable
- General idea
  - Separate available examples into training set and test set
  - Learn hypothesis on training set
  - Evaluate hypothesis on test set
- Repeat above several times with different training/test sets and average results
The Matrix (1999)

Nearest Neighbor
k–Nearest Neighbor Classifier

- Given training examples X
- Given query instance $x_q$
- Find k examples in X that are the nearest neighbors to $x_q$
- Classification: $h(x_q) = \text{majority class among k neighbors}$
- Regression: $h(x_q) = \text{mean value of k neighbors}$
Voronoi Diagram of $k$-NN
k-NN Distance Metrics

- Euclidean distance for numeric features
  - Normalize feature values
- Hamming distance for discrete features
  - Distance = 1 if feature values differ, else 0

- Curse of dimensionality
  - Many attributes hurts performance
  - Training covers small portion of space
  - Noisy attributes equally influential
Overfitting in $k$-NN

(k = 1)  

(k = 5)
How to choose k?

- When k is small, single instances matter
  - High complexity $\rightarrow$ overfitting
- As k increases, we average over more instances
  - Low complexity $\rightarrow$ underfitting
- Cross-validation is used to tune k
Probabilistic Learning
Bayesian Learning

- Combines prior knowledge with evidence to make predictions
- Optimal (albeit impractical) classifier
- Naïve Bayes classifier (practical)
  - Assumes independence among attributes
Bayes Rule

\[ P(C_i \mid x) = \frac{P(x \mid C_i)P(C_i)}{P(x)} \]

- \( C_i \) is the class, \( 1 \leq i \leq K \)
- \( x \) is the attribute vector of an instance
- \( P(C_i \mid x) = \) probability that instance \( x \) belongs to class \( C_i \) (posterior)
- \( P(x \mid C_i) = \) probability that an instance drawn from class \( C_i \) would be \( x \) (likelihood)
- \( P(C_i) = \) probability of class \( C_i \) (prior)
- \( P(x) = \) probability of instance \( x \) (evidence)
Bayes Rule: Family Car

\[ P(\text{FamilyCar} \mid \text{EnginePower}, \text{Price}) = \]

\[ \frac{P(\text{EnginePower}, \text{Price} \mid \text{FamilyCar}) P(\text{FamilyCar})}{P(\text{EnginePower}, \text{Price})} \]
Classify instance $x$ as class $C_i$ such that

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

Since only interested in maximum, can ignore denominator $p(x)$ (i.e., ignore normalization $\alpha$)

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

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

$$i = \arg \max_{1 \leq k \leq K} P(x \mid C_k)$$
Practical issue
- $P(x \mid C_i)$ 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 $K \times 2^D$ 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 \( X \)
  
  \[
P(C_i) = \frac{\left| \{(x, r) \in X \mid r = C_i \} \right|}{|X|}
  \]
  
  \[
P(x_j = v \mid C_i) = \frac{\left| \{(x, r) \in X \mid x_j = v \text{ and } r = C_i \} \right|}{\left| \{(x, r) \in X \mid r = C_i \} \right|}
  \]
- Classify new instance \( x \) as class \( C_i \) such that

  \[
i = \arg \max_{1 \leq k \leq K} P(C_k) * \prod_{j=1}^{D} P(x_j \mid C_k)
  \]
Another practical issue
  ◦ 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 $\in \mathbb{R}$ to price $\in \{\text{low, medium, high}\}$
Yet another practical issue

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

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

- Independence assumption rarely true
  - E.g., Is “price” independent of “engine power”?
- Naïve Bayes classifier still does surprisingly well
- Simple, effective baseline for other learners
Decision Trees
Decision Tree Example
Decision Tree for Playing Tennis

Outlook

- Sunny
  - Humidity
    - High
      - No
    - Normal
      - Yes
- Overcast
  - Yes
- Rain
  - Wind
    - Strong
    - Weak
      - Yes
Each internal node tests one attribute
Each branch corresponds to a test result
  ◦ One for each value of discrete attribute
  ◦ Yes/no for numeric attributes

Leaf node
  ◦ Classification: Class value
  ◦ Regression: Real value
Learning Algorithm

- Create root node with all examples $X$
- Call GenerateTree (root, $X$)

\[
\text{GenerateTree (node, } X) \\
\text{ If node is “pure” enough} \\
\text{ Then assign class to node and return} \\
\text{ Else Choose “best” split attribute } A \\
\text{ Foreach value } v \text{ of } A \\
\text{ } X’ = \text{ examples in } X \text{ where } A = v \\
\text{ Create childNode with examples } X’ \\
\text{ GenerateTree (childNode, } X’) \
\]
Learning Algorithm

- When is a node “pure”?
  ◦ Most examples of same class

- Which attribute is “best”?
  ◦ One that leads to more pure nodes

- Entropy measures lack of purity
  ◦ I.e., uncertainty in classification
S = set of training examples
p∪ = proportion of positive examples in S
p∩ = proportion of negative examples in S
Entropy measures the impurity of S

\[ \text{Entropy}(S) \equiv - p∪ \log_2 p∪ - p∩ \log_2 p∩ \]
Information Gain

Gain(S, A) = expected reduction in entropy due to sorting \( S \) on attribute \( A \)

\[
Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)
\]

[Diagram of decision trees for A1=? and A2=? with leaf nodes labeled with data points]
## Training Examples: PlayTennis

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>PlayTennis</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D4</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D6</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D7</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D8</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D9</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D10</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D11</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D12</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D13</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
</tbody>
</table>
Selecting the Best Attribute

\[ S: [9+,5-] \]
\[ E = 0.940 \]

**Humidity**

- **High**
  - \([3+,4-]\)
  - \(E = 0.985\)

- **Normal**
  - \([6+,1-]\)
  - \(E = 0.592\)

\[ G_{\text{ain}} (S, \text{Humidity}) \]
\[ = 0.940 - \left( \frac{7}{14} \right) \cdot 0.985 - \left( \frac{7}{14} \right) \cdot 0.592 \]
\[ = 0.151 \]

**Wind**

- **Weak**
  - \([6+,2-]\)
  - \(E = 0.811\)

- **Strong**
  - \([3+,3-]\)
  - \(E = 1.00\)

\[ G_{\text{ain}} (S, \text{Wind}) \]
\[ = 0.940 - \left( \frac{8}{14} \right) \cdot 0.811 - \left( \frac{6}{14} \right) \cdot 1.0 \]
\[ = 0.048 \]
Selecting the Next Attribute

Which attribute should be tested here?

\[ S_{\text{Sunny}} = \{D_1, D_2, D_8, D_9, D_{11}\} \]

\[ \text{Gain} (S_{\text{Sunny}}, \text{Humidity}) = .970 - (3/5) 0.0 - (2/5) 0.0 = .970 \]

\[ \text{Gain} (S_{\text{Sunny}}, \text{Temperature}) = .970 - (2/5) 0.0 - (2/5) 1.0 - (1/5) 0.0 = .570 \]

\[ \text{Gain} (S_{\text{Sunny}}, \text{Wind}) = .970 - (2/5) 1.0 - (3/5) .918 = .019 \]
Consider adding noisy training example #15:
  ◦ (⟨Sunny, Hot, Normal, Strong⟩, PlayTennis = No)
What effect on earlier tree?
Overfitting in Decision Tree Learning

![Graph showing the relationship between tree size and error rate for validation and training sets. The graph illustrates how as the tree size increases, the error rate decreases for both sets, with the training set error generally being lower than the validation set error.](image)
Avoiding Overfitting

- How can we avoid overfitting?
  - Stop growing when error on validation set increases

- Minimum Description Length (MDL)
  - Minimize \[ \text{size(tree)} + \text{size(misclassifications(tree))} \]
Neural Networks
Neural Networks

- Inspired from human brain
  - Brain consists of interconnected neurons
  - Brain still outperforms machines on several tasks
  - E.g., vision, speech recognition, learning
- Classification and regression
- Trained using error backpropagation
- Also called multilayer perceptrons (MLP)
Human Neuron

- Dendrite
- Synapse
- Axon from another cell
- Axon
- Nucleus
- Cell Body or Soma
- Synapses
Human Brain vs. Computer

- Processors
  - Computer: Typically 2–4 (~10^9 Hz)
  - Brain: 10^{11} neurons (~ 10^3 Hz)

- Parallelism
  - Computer: Typically little
  - Brain: Massive parallelism
    - On average, each neuron is connected via synapses to 10,000 other neurons
The Singularity

1. The accelerating pace of change...
   - Agricultural Revolution: 8,000 years
   - Industrial Revolution: 120 years
   - Light-bulb: 90 years
   - Moon landing: 22 years
   - World Wide Web: 9 years
   - Human genome sequenced: 0 years

2. ...and exponential growth in computing power...
   Computer technology, shown here climbing dramatically by powers of 10, is now progressing more each hour than it did in its entire first 90 years

3. ...will lead to the Singularity
   - Apple II: At a price of $1,298, the compact machine was one of the first massively popular personal computers
   - Power Mac G4: The first personal computer to deliver more than 1 trillion floating-point operations per second

Time Magazine (Feb 2011)
What Happens Then...?

Perceptron (Rosenblatt, 1962)

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

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

\[ \mathbf{x} = [1, x_1, \ldots, x_d] \]
Perceptron Regression

- \( y = wx + w_0 \)
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 \( \eta \) controls rate of descent
Linearly Separable
Logistic Function

- Also called “sigmoid function”

\[
\text{logistic}(z) = \frac{1}{1 + e^{-z}}
\]
Logistic Function
Perceptron Classification

\[ y = \text{logistic} \left( w \cdot x \right) = \frac{1}{1 + e^{-w \cdot x}} \]
Learning Nonlinear Functions

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

\[ y_i = v_i \cdot z = \sum_{h=1}^{H} v_{ih} z_h + v_{i0} \]

\[ z_h = \text{logistic} \left( w_h \cdot x \right) \]

\[ = \frac{1}{1 + \exp \left[ - \left( \sum_{j=1}^{d} w_{hj} x_j + w_{h0} \right) \right]} \]

(Rumelhart et al., 1986)
MLP as Universal Approximator

- MLP can represent any Boolean function
- MLP with two hidden layers can approximate any function with continuous inputs and outputs
- MLP with one sufficiently large hidden layer can learn any nonlinear function
Learning the Weights

- Weights $v_{ih}$ feeding into output units learned using previous methods
- Weights $w_{hj}$ feeding into hidden units learned based on error propagated from output layer
- Error backpropagation (Rumelhart et al., 1986)

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

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

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}}$$
Note: $y$ is net output, $r$ = target value

Forward

$z_h = \text{logistic}(w_h \cdot x)$

$y^t = \sum_{h=1}^{H} v_h z_h^t + v_0$

Backward

$E(W, v | X) = \frac{1}{2} \sum_t (r^t - y^t)^2$

$\Delta v_h = \eta \sum_t (r^t - y^t) z_h^t$

$\Delta w_{hj} = -\eta \frac{\partial E}{\partial w_{hj}}$

$= -\eta \sum_t \frac{\partial E}{\partial y^t} \frac{\partial y^t}{\partial z_h^t} \frac{\partial z_h^t}{\partial w_{hj}}$

$= -\eta \sum_t -(r^t - y^t) v_h z_h^t (1 - z_h^t) x_j^t$

$= \eta \sum_t (r^t - y^t) v_h z_h^t (1 - z_h^t) x_j^t$
Example

- $f(x) = \sin(6x)$
- $x^t \sim U(-0.5,0.5)$
- $y^t = f(x^t) + N(0,0.1)$
- 2 hidden units
- After 100, 200 and 300 epochs

An epoch is one pass through all training examples.
Example (cont.)
Overfitting in MLPs

- Network with $d$ inputs, $K$ outputs and $H$ hidden units has $K(H+1)+H(d+1)$ weights.
- Choosing $H$ too high can lead to overfitting.
Overfitting in MLPs

Previous example:
\[ f(x) = \sin(6x) \]
Overfitting in MLPs

- Similar overfitting behavior if training continued too long
- More and more weights move from zero
- Overtraining
Cross validation can be used to choose a good $H$ and a good stopping condition.

Training multiple networks, each with different random initial weights, can address local minima in the error.
Neural Networks

- Based loosely on structure of human brain
- Multilayer perceptron
- Universal approximator
- Error backpropagation
- Overall, a powerful general-purpose learner, but requires considerable tuning
Using deep (many-layered) neural nets to learn complex abstractions (features)
Unsupervised Learning

k-Means Clustering
$k$-means Clustering

- Unsupervised learning
  - Just instances, no classes
- Partition instances into $k$ disjoint sets
- Each set has a representative instance $m_i$
- Place instance $x$ into set $i$ such that $\text{distance}(x, m_i)$ is minimal
- Choose new central $m_i$ for each set
- Repeat until $m_i$ converge
$k$-means Clustering

- Pre-processor for supervised learning
  - Cluster instances into $k$ sets
  - Learn local classifier for each set

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

- Waikato Environment for Knowledge Analysis (WEKA)
  - Java-based
- Orange
  - Python-based
  - [http://orange.biolab.si/](http://orange.biolab.si/)
- R, Python scikit-learn, ...
  - [http://scikit-learn.org](http://scikit-learn.org)
Reinforcement Learning
Reinforcement Learning

“Imagine playing a new game whose rules you don’t know; after a hundred or so moves, your opponent announces, ‘You lose.’ This is reinforcement learning in a nutshell.” [Russell & Norvig]
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
  - $\pi$: State $\rightarrow$ 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 $U^\pi(s)$
Utility Estimation

- Utility of state $s$ is the expected sum of the discounted rewards returned by following policy $\pi$ for $t$ times as $t \to \infty$

$$U^\pi(s) = E \left[ \sum_{t=0}^{\infty} \gamma^t R(S_t) \right]$$

- $S_t$ is the state reached at time $t$ according to policy $\pi$
- $\gamma$ is the discount factor ($0 < \gamma \leq 1$)
Utility Estimation

- Previous approach does not exploit relationships between states
- Utility of state $s$ is the reward in state $s$ plus the expected utility of its successor states

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

- Run many trials to learn $P(s' \mid s, \pi(s))$
- Solve above (Bellman) equations using dynamic programming
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
  - $U([3,3]) = R([3,3]) + (0.8)*U([4,3]) + (0.1)*U([3,3]) + (0.1)*U([3,2])$
  - $U([3,3]) = (-0.04) + (0.8)*(1) + (0.1)*(0) + (0.1)*(0)$
  - $U([3,3]) = 0.76$

- Eventually, $U([3,3])$
  - $U([3,3]) = (-0.04) + (0.8)*(1) + (0.1)*(0.918) + (0.1)*(0.66)$
  - $U([3,3]) = 0.918$
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 $\alpha$ to control rate of adjustment
- If $\alpha$ decreases with the number of times state $s$ is visited, then TD will converge to correct $U^\pi(s)$
Active Reinforcement Learning

- Passive RL assumes policy is given
- Active RL learns both utilities and policy
- If agent finds a suboptimal path to reward, may lead to suboptimal policy

![Chart showing RMS error and policy loss over number of trials]

![Diagram illustrating a grid with arrows and a suboptimal path highlighted]
Global Thermonuclear War

- How to overcome a suboptimal policy?

WarGames (1983)
Exploration

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

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

\(u\) 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
Exploration

- Previous example with exploration
  - $R^+ = 2, N_e = 5$
  - Finds optimal policy

![Graphs showing RMS error, policy loss, and utility estimates over number of trials.](image)
Q-Learning

- Active temporal-difference learning agent
- Passive TD agent required a policy \( \pi \) or an action model \( P(s'|s,a) \)
- Define \( Q(s,a) \) as the value of taking action \( a \) in state \( s \)

\[
U(s) = \max_a Q(s,a)
\]

\[
Q(s, a) = R(s) + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q(s', a')
\]
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
               Nsa, 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’ ∈ ACTIONS(s)
        Q[s,a ’] ← r’
if s is not null then
    increment Nsa[s,a]
    Q[s,a] ← Q[s,a] + α(Nsa[s,a])(r + γ maxa’ Q[s’,a’] - Q[s,a])
s, a, r ← s’, argmaxa’ f(Q[s’,a’], Nsa[s’,a’]), r’
return a

Exploration function f(u,n)

Learning rate (decreasing with n): α(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 $\alpha(n) = 100 / (99 + n)$
Reinforcement Learning Systems

- Maja Machine Learning Framework (MMLF)
  - Python library
  - mmlf.sourceforge.net
- OpenAI Gym
  - RL environments
  - gym.openai.com

heli.stanford.edu

sarvagyavaish.github.io/FlappyBirdRL
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
  - Nearest neighbor
  - Naïve Bayes
  - Decision tree
  - Neural network
- How to choose the right model?
  - Overfitting
- Unsupervised learning (clustering)
- Reinforcement learning