Machine Learning
Introduction

Mohsen Afsharchi
IASBS

What is Learning?

- Herbert Simon: "Learning is any process by which a system improves performance from experience."
- What is the task?
  - Classification
  - Making Intelligent Decision (Problem solving / planning / control)

Classification

- Assign object/event to one of a given finite set of categories.
  - Medical diagnosis
  - Credit card applications or transactions
  - Fraud detection in e-commerce
  - Worm detection in network packets
  - Spam filtering in email
  - Recommended articles in a newspaper
  - Recommended books, movies, music, or jokes
  - Financial investments
  - DNA sequences
  - Spoken words
  - Handwritten letters
  - Astronomical images

Problem Solving / Planning / Control

- Performing actions in an environment in order to achieve a goal.
  - Solving calculus problems
  - Playing checkers, chess
  - Balancing a pole
  - Driving a car or a jeep
  - Flying a plane, helicopter, or rocket
  - Controlling an elevator
  - Controlling a character in a video game
  - Controlling a mobile robot

Measuring Performance

- Classification Accuracy
- Solution correctness
- Solution quality (length, efficiency)

Why Study Machine Learning? Engineering Better Computing Systems

- Develop systems that are too difficult/expensive to construct manually because they require specific detailed skills or knowledge tuned to a specific task (***knowledge engineering bottleneck***).
- Develop systems that can automatically adapt and customize themselves to individual users.
  - Personalized news or mail filter
  - Personalized tutoring
- Discover new knowledge from large databases (***data mining***).
  - Market basket analysis (e.g. diapers and beer)
  - Medical text mining (e.g. migraines to calcium channel blockers to magnesium)

Example 1

We are given categories for these images: What are these?

From ETH database of object categories, [Leibe & Schiele 2003]

- A *classification* problem: predict category y based on image $\mathbf{x}$.
- Little chance to "hand-craft" a solution, without learning.
- Applications: robotics, HCI, web search (a *real* image Google...)

Why Study Machine Learning? Cognitive Science

- Computational studies of learning may help us understand learning in humans and other biological organisms.
  - Hebbian neural learning
    - "Neurons that fire together, wire together."
  - Human's relative difficulty of learning disjunctive concepts vs. conjunctive ones.
  - Power law of practice (the logarithm of the reaction time for a particular task decreases linearly with the logarithm of the number of practice trials taken)

Why Study Machine Learning?
The Time is Ripe

- Many basic effective and efficient algorithms available.
- Large amounts of on-line data available.
- Large amounts of computational resources available.

Related Disciplines

- Artificial Intelligence
- Data Mining
- Probability and Statistics
- Information theory
- Numerical optimization
- Computational complexity theory
- Control theory (adaptive)
- Psychology (developmental, cognitive)
- Neurobiology
- Linguistics
- Philosophy

Supervised Learning

This is an example of *supervised learning*, which consists of the following basic steps:

- **Data collection** Start with *training data* for which we know the correct outcome provided by a *teacher* or *oracle*. In this case: images for which we know the object category.
- **Representation** Choose how to represent the data.
- **Modeling** Choose a *hypothesis class* - a set of possible explanations for the connection between images and categories. This is our *model* of the problem.
- **Estimation** Find best hypothesis you can in the chosen class.
- **Model selection** We may reconsider the class of hypotheses given the outcome.

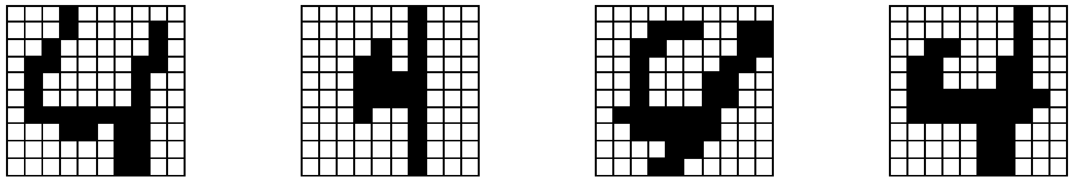
Each of these steps can make or break the learning outcome.

Example2 : Document Classification

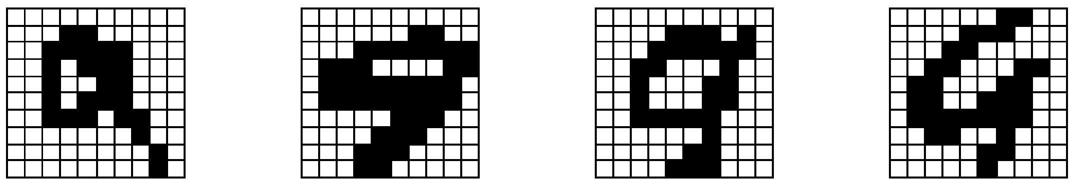
- A few labeled web pages with categories: faculty, student, department, course etc.
- Need to automatically classify previously unseen web pages.
- What would be good *features* to represent these data?
- *Feature selection* methods allow us to select from a large set of features those most helpful for the task.

Example 3: Binary Classification

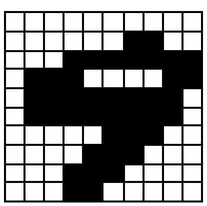
$y = +1$



$y = -1$



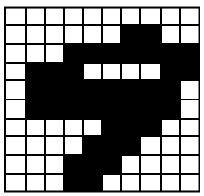
- Representation as a vector:



$\Rightarrow \quad [0000000000\ 0000001100\ 0001111111\ \ldots\ 0001100000]^T$

Modelling

- Examples are binary vectors of dimension $d = 100$:



$$= \mathbf{x} = [0000000000\ 0000001100\ 0001111111\ \ldots\ 0001100000]^T$$

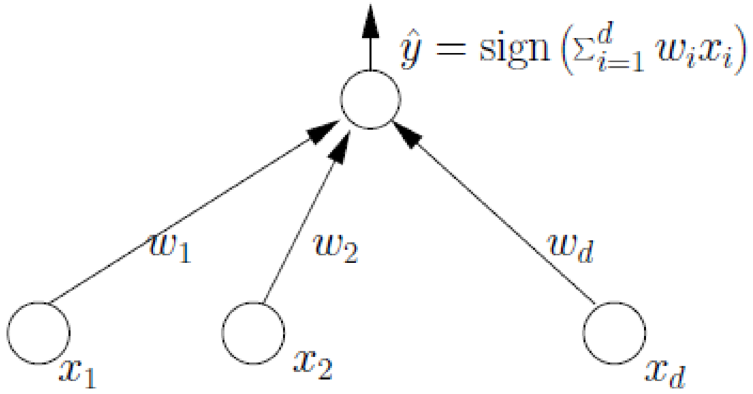
- Labels are binary as well, $y \in \{-1, +1\}$.

- We consider the following hypothesis class: $\hat{y} = \operatorname{sign}(\mathbf{w} \cdot \mathbf{x})$

The "hat" ^ means "estimated". Dot product: $\mathbf{w} \cdot \mathbf{x} = \mathbf{w}^T \mathbf{x} = \sum_{i=1}^{d} w_i x_i$

- This is a *linear classifier* (based on a linear combination of input components). It defines a mapping from the data to labels.

Estimation



- How can we set the parameter vector $\mathbf{w}$ using training data?
  - Start with a random set of weights
  - Iterate through the training data; for each $(\mathbf{x}, y)$, if the current classifier makes a mistake, set

$$w_i \leftarrow w_i + y x_i \text{ for all } i = 1, \ldots, d.$$

Estimation: mistake driven algorithm

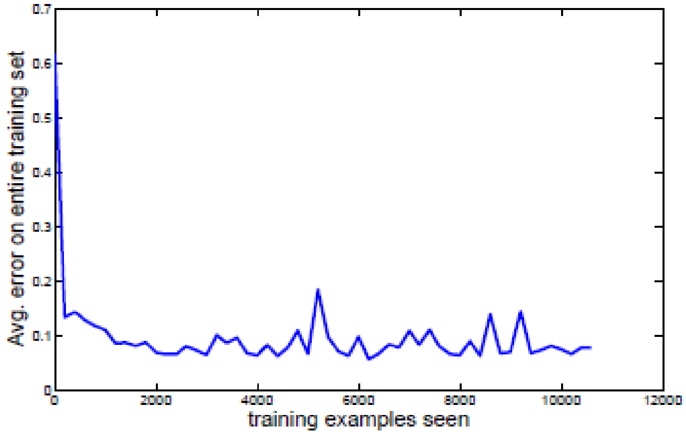
- Start with a random set of weights
- Iterate through the training data; for each $(\mathbf{x}, y)$, if the current classifier makes a mistake, set

$$w_i \leftarrow w_i + yx_i \text{ for all } i = 1, \ldots, d.$$

- Magnitude $|w_i|$ reflects importance (*weight*)of the i-th pixel.
  - Negative w_i means that i-th pixel being on suggests a 9;
  - positive w_i means it suggests a 4.
- When do we stop?..

Evaluation

- We can see how well we can predict the labels in the training set:

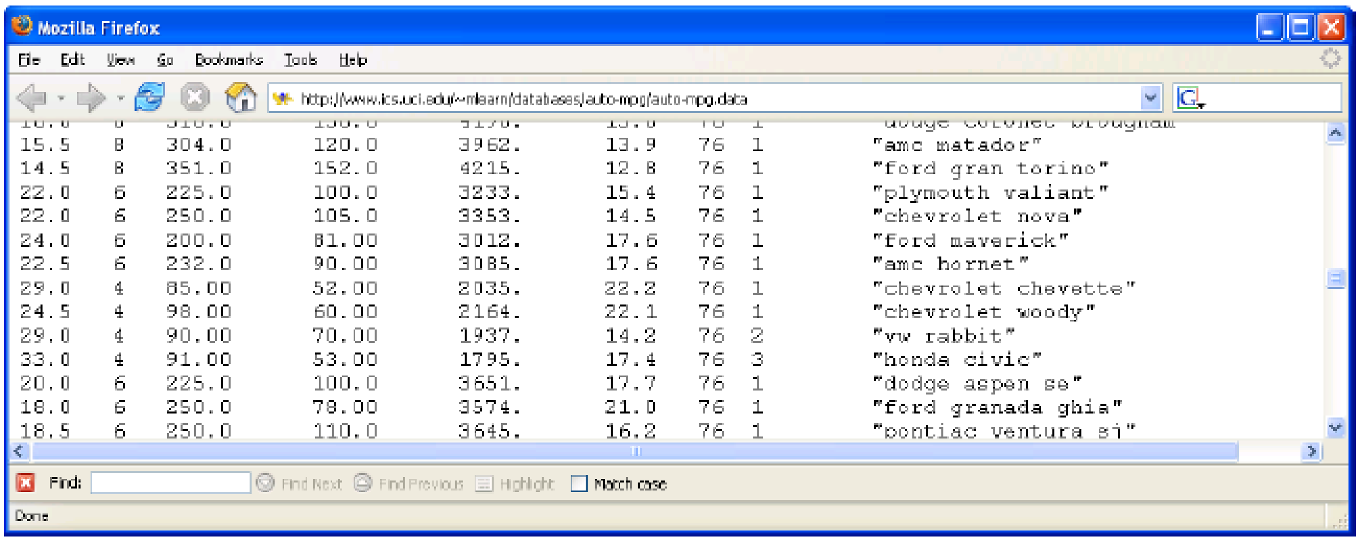


- Expect the average classification error to go down as we look at more examples. *Do we expect it to reach zero?*

Supervised learning beyond classification

- Often the goal is not to classify a data point but to predict some quantitative outcome. This is a *regression* problem.

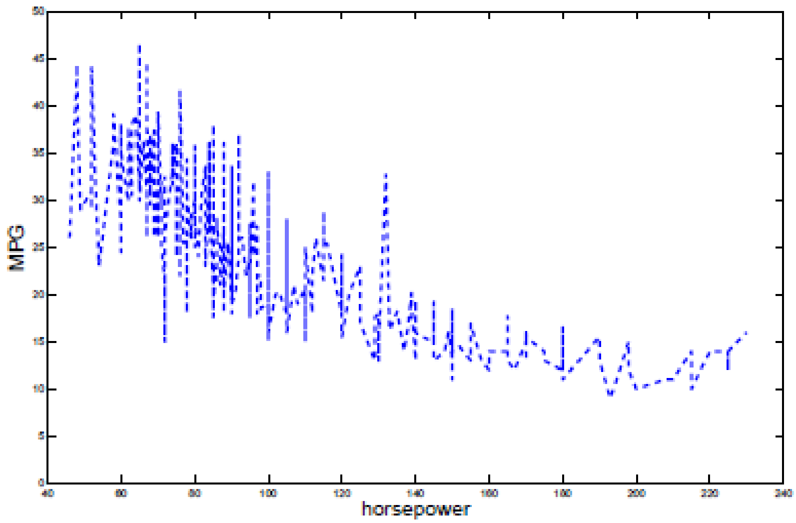
- Suppose we want to predict gas mileage of a car based on some characteristics: number of cylinders or doors, weight, horsepower, year etc.



15.5	8	304.0	120.0	3962.	13.9	76	1	"amc matador"
14.5	8	351.0	152.0	4215.	12.8	76	1	"ford gran torino"
22.0	6	225.0	100.0	3233.	15.4	76	1	"plymouth valiant"
22.0	6	250.0	105.0	3353.	14.5	76	1	"chevrolet nova"
24.0	6	200.0	81.00	3012.	17.6	76	1	"ford maverick"
22.5	6	232.0	90.00	3085.	17.6	76	1	"amc hornet"
29.0	4	85.00	52.00	2035.	22.2	76	1	"chevrolet chevette"
24.5	4	98.00	60.00	2164.	22.1	76	1	"chevrolet woody"
29.0	4	90.00	70.00	1937.	14.2	76	2	"vw rabbit"
33.0	4	91.00	53.00	1795.	17.4	76	3	"honda civic"
20.0	6	225.0	100.0	3651.	17.7	76	1	"dodge aspen se"
18.0	6	250.0	78.00	3574.	21.0	76	1	"ford granada ghia"
18.5	6	250.0	110.0	3645.	16.2	76	1	"pontiac ventura sj"

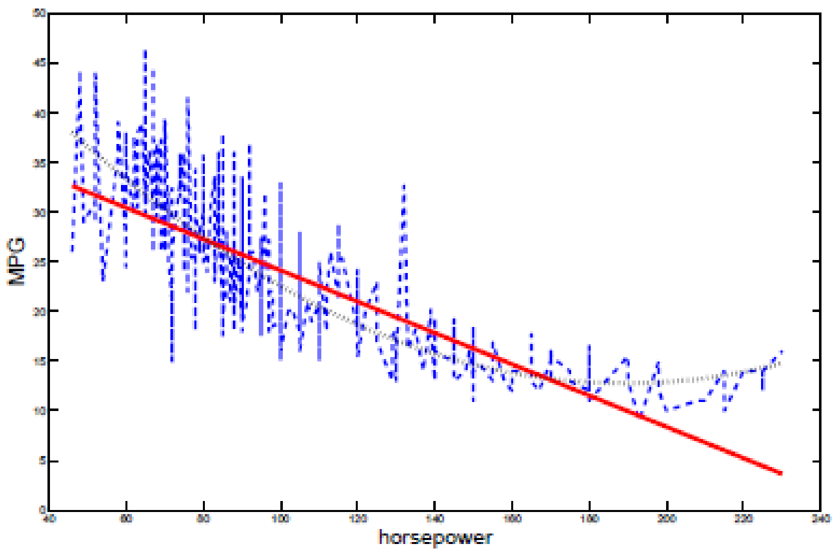
Regression

- Let us look at MPG (miles per gallon) as a function of horsepower only.
- We can fit a straight line to try and explain this behavior:



Regression

- We can try to fit a quadratic function: $\hat{y} = w_2 x^2 + w_1 x + w_0$.



Generalization

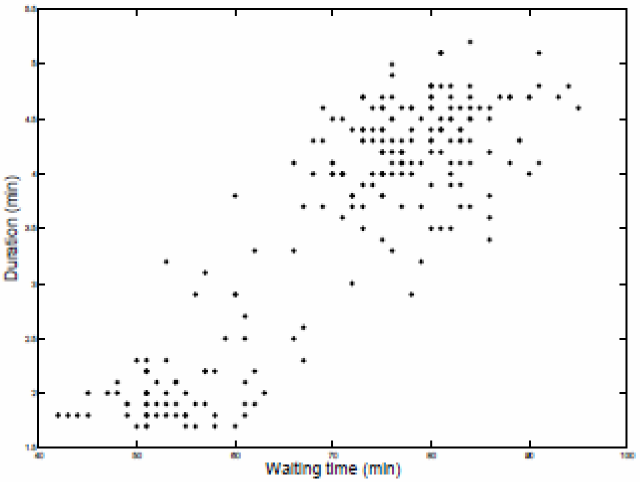
- The ultimate goal is to do as well as possible on new, unseen data (a *test set*).
- We only have access to labels ("ground truth") for the training set.
- There is a danger of *overfitting*: learning to predict training labels very well that does not generalize!
- What can we do about it?
  - The most naive approach: minimize training error and keep our fingers crossed.
  - A somewhat more clever approach: if we have enough training data, set some of it aside (*holdout*) and test on it once learning is done.
  - There are much more powerful, sophisticated and rigorous methods that we will study in this class.

Unsupervised Learning

- In *unsupervised learning* the goal is not to predict labels, but to learn some sort of structure in the data.
  - No labels involved!

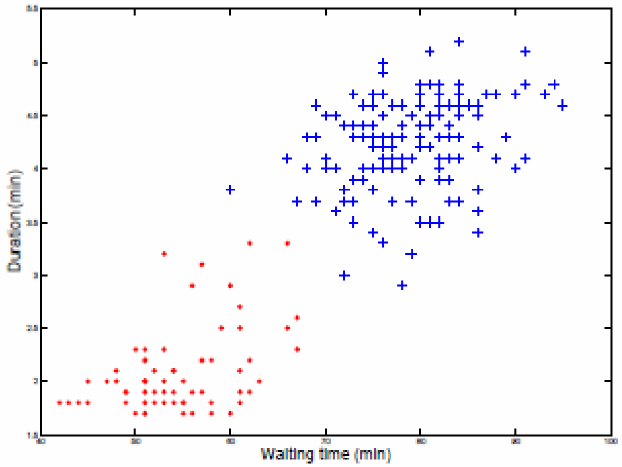
- Typical problem: *clustering*.

the Old Faithful data



Example

- Goal of clustering: discover coherent groups ("clumps") of data.



- Common applications: clustering documents, image segmentation (clustering pixels), activity discovery.

More unsupervised learning

- Other unsupervised tasks:
  - Compression and dimensionality reduction: finding a more parsimonious description for the data (e.g. coding).
  - Detection of outliers/anomalies.
  - Finding correlations between groups of variables.
- The objective is often more vague or subjective than in supervised learning. This is more of an exploratory/descriptive data analysis.

Other learning scenarios

- *Semi-supervised* learning: lots of data available, but only small portion is labeled (e.g. since labeling is expensive).
  - Use unlabeled data to improve learning from the few labeled examples.

- *Reinforcement learning*: action-reward settings.
  - the goal is to find a sequence of actions that maximize expected reward.
  - Probably out of scope of this class...

Defining the Learning Task

Improve on task, T, with respect to
performance metric, P, based on experience, E.

T: Playing checkers
P: Percentage of games won against an arbitrary opponent
E: Playing practice games against itself

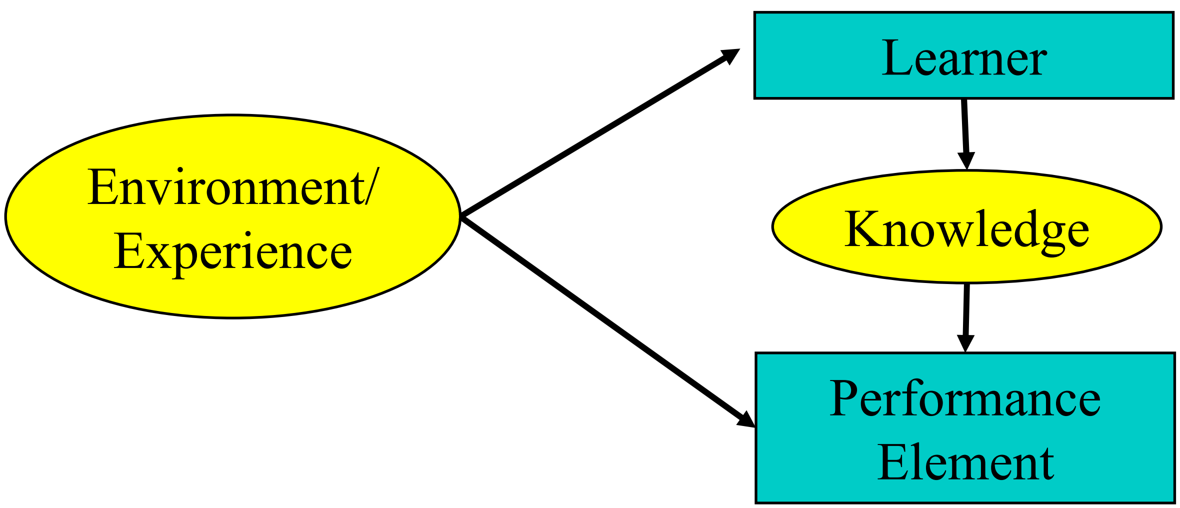
T: Recognizing hand-written words
P: Percentage of words correctly classified
E: Database of human-labeled images of handwritten words

T: Driving on four-lane highways using vision sensors
P: Average distance traveled before a human-judged error
E: A sequence of images and steering commands recorded while observing a human driver.

T: Categorize email messages as spam or legitimate.
P: Percentage of email messages correctly classified.
E: Database of emails, some with human-given labels

Designing a Learning System

- Choose the training experience
- Choose exactly what is too be learned, i.e. the ***target function***.
- Choose how to represent the target function.
- Choose a learning algorithm to infer the target function from the experience.



Sample Learning Problem

- Learn to play checkers from self-play
- We will develop an approach analogous to that used in the first machine learning system developed by Arthur Samuels at IBM in 1959.

Training Experience

- Direct experience: Given sample input and output pairs for a useful target function.
  - Checker boards labeled with the correct move, e.g. extracted from record of expert play
- Indirect experience: Given feedback which is ***not*** direct I/O pairs for a useful target function.
  - Potentially arbitrary sequences of game moves and their final game results.
- Credit/Blame Assignment Problem: How to assign credit blame to individual moves given only indirect feedback?

Source of Training Data

- Provided random examples outside of the learner's control.
  - Negative examples available or only positive?
- Good training examples selected by a "benevolent teacher."
  - "Near miss" examples
- Learner can query an oracle about class of an unlabeled example in the environment.
- Learner can construct an arbitrary example and query an oracle for its label.
- Learner can design and run experiments directly in the environment without any human guidance.

Training vs. Test Distribution

- Generally assume that the training and test examples are independently drawn from the same overall distribution of data.
  - IID: Independently and identically distributed
- If examples are not independent, requires ***collective classification***.
- If test distribution is different, requires ***transfer learning***.

Choosing a Target Function

- What function is to be learned and how will it be used by the performance system?
- For checkers, assume we are given a function for generating the legal moves for a given board position and want to decide the best move.
  - Could learn a function:

    ChooseMove(board, legal-moves) → best-move
  - Or could learn an ***evaluation function***, V(board) → R, that gives each board position a score for how favorable it is. V can be used to pick a move by applying each legal move, scoring the resulting board position, and choosing the move that results in the highest scoring board position.

Ideal Definition of $V(b)$

- If b is a final winning board, then $V(b) = 100$
- If b is a final losing board, then $V(b) = -100$
- If b is a final draw board, then $V(b) = 0$
- Otherwise, then $V(b) = V(b')$, where b' is the highest scoring final board position that is achieved starting from b and playing optimally until the end of the game (assuming the opponent plays optimally as well).
  – Can be computed using complete mini-max search of the finite game tree.

Approximating $V(b)$

- Computing $V(b)$ is intractable since it involves searching the complete exponential game tree.
- Therefore, this definition is said to be ***non-operational***.
- An ***operational*** definition can be computed in reasonable (polynomial) time.
- Need to learn an operational *approximation* to the ideal evaluation function.

Representing the Target Function

- Target function can be represented in many ways: lookup table, symbolic rules, numerical function, neural network.
- There is a trade-off between the expressiveness of a representation and the ease of learning.
- The more expressive a representation, the better it will be at approximating an arbitrary function; however, the more examples will be needed to learn an accurate function.

Linear Function for Representing $V(b)$

- In checkers, use a linear approximation of the evaluation function.

$$\widehat{V}(b) = w_0 + w_1 \cdot bp(b) + w_2 \cdot rp(b) + w_3 \cdot bk(b) + w_4 \cdot rk(b) + w_5 \cdot bt(b) + w_6 \cdot rt(b)$$

  - *bp*(*b*): number of black pieces on board b
  - *rp*(*b*): number of red pieces on board b
  - *bk*(*b*): number of black kings on board b
  - *rk*(*b*): number of red kings on board b
  - *bt*(*b*): number of black pieces threatened (i.e. which can be immediately taken by red on its next turn)
  - *rt*(*b*): number of red pieces threatened

Obtaining Training Values

- Direct supervision may be available for the target function.
  - < <*bp*=3,*rp*=0,*bk*=1,*rk*=0,*bt*=0,*rt*=0>, 100> (win for black)
- With indirect feedback, training values can be estimated using ***temporal difference learning*** (used in ***reinforcement learning*** where supervision is ***delayed reward***).

Temporal Difference Learning

- Estimate training values for intermediate (non-terminal) board positions by the estimated value of their successor in an actual game trace.

$$V_{train}(b) = \hat{V}(\text{successor}(b))$$

  where successor(b) is the next board position where it is the program's move in actual play.
- Values towards the end of the game are initially more accurate and continued training slowly "backs up" accurate values to earlier board positions.

Learning Algorithm

- Uses training values for the target function to induce a hypothesized definition that fits these examples and hopefully generalizes to unseen examples.
- In statistics, learning to approximate a continuous function is called ***regression***.
- Attempts to minimize some measure of error (***loss function***) such as ***mean squared error***:

$$E = \frac{\sum_{b \in B} [V_{train}(b) - \widehat{V}(b)]^2}{|B|}$$

Least Mean Squares (LMS) Algorithm

- A gradient descent algorithm that incrementally updates the weights of a linear function in an attempt to minimize the mean squared error

Until weights converge :

For each training example b do :

1) Compute the absolute error :

$$error(b) = V_{train}(b) - \widehat{V}(b)$$

2) For each board feature, f_i, update its weight, w_i :

$$w_i = w_i + c \cdot f_i \cdot error(b)$$

for some small constant (learning rate) c

LMS Discussion

- Intuitively, LMS executes the following rules:
  - If the output for an example is correct, make no change.
  - If the output is too high, lower the weights proportional to the values of their corresponding features, so the overall output decreases
  - If the output is too low, increase the weights proportional to the values of their corresponding features, so the overall output increases.
- Under the proper weak assumptions, LMS can be proven to eventetually converge to a set of weights that minimizes the mean squared error.

Lessons Learned about Learning

- Learning can be viewed as using direct or indirect experience to approximate a chosen target function.
- Function approximation can be viewed as a search through a space of hypotheses (representations of functions) for one that best fits a set of training data.
- Different learning methods assume different hypothesis spaces (representation languages) and/or employ different search techniques.

Various Function Representations

- Numerical functions
  - Linear regression
  - Neural networks
  - Support vector machines
- Symbolic functions
  - Decision trees
  - Rules in propositional logic
  - Rules in first-order predicate logic
- Instance-based functions
  - Nearest-neighbor
  - Case-based
- Probabilistic Graphical Models
  - Naïve Bayes
  - Bayesian networks
  - Hidden-Markov Models (HMMs)
  - Markov networks

Various Search Algorithms

- Gradient descent
  - Perceptron
  - Backpropagation
- Dynamic Programming
  - HMM Learning
- Divide and Conquer
  - Decision tree induction
  - Rule learning
- Evolutionary Computation
  - Genetic Algorithms (GAs)
  - Genetic Programming (GP)

Evaluation of Learning Systems

- Experimental
  - Conduct controlled cross-validation experiments to compare various methods on a variety of benchmark datasets.
  - Gather data on their performance, e.g. test accuracy, training-time, testing-time.
  - Analyze differences for statistical significance.
- Theoretical
  - Analyze algorithms mathematically and prove theorems about their:
    - Computational complexity
    - Ability to fit training data
    - Sample complexity (number of training examples needed to learn an accurate function)

History of Machine Learning

- 1950s
  - Samuel's checker player
  - Selfridge's Pandemonium
- 1960s:
  - Neural networks: Perceptron
  - Pattern recognition
  - Learning in the limit theory
  - Minsky and Papert prove limitations of Perceptron
- 1970s:
  - Symbolic concept induction
  - Winston's arch learner
  - Expert systems and the knowledge acquisition bottleneck
  - Quinlan's ID3
  - Michalski's AQ and soybean diagnosis
  - Scientific discovery with BACON
  - Mathematical discovery with AM

History of Machine Learning (cont.)

- 1980s:
  - Advanced decision tree and rule learning
  - Explanation-based Learning (EBL)
  - Learning and planning and problem solving
  - Utility problem
  - Analogy
  - Cognitive architectures
  - Resurgence of neural networks (connectionism, backpropagation)
  - Valiant’s PAC Learning Theory
  - Focus on experimental methodology
- 1990s
  - Data mining
  - Adaptive software agents and web applications
  - Text learning
  - Reinforcement learning (RL)
  - Inductive Logic Programming (ILP)
  - Ensembles: Bagging, Boosting, and Stacking
  - Bayes Net learning

History of Machine Learning (cont.)

- 2000s
  - Support vector machines
  - Kernel methods
  - Graphical models
  - Statistical relational learning
  - Transfer learning
  - Sequence labeling
  - Collective classification and structured outputs
  - Computer Systems Applications
    - Compilers
    - Debugging
    - Graphics
    - Security (intrusion, virus, and worm detection)
  - Email management
  - Personalized assistants that learn
  - Learning in robotics and vision