Supervised Methods in Data Mining

Week 5

- INFO911 -

Supervised Machine Learning

in Data Mining -

Classification and Prediction

Presented by
Prof. Zhifeng Wang

Outline

- Part I
  - Machine Learning and Pattern Recognition
  - K-nearest neighbor (KNN)
  - Learning vector quantisation (LVQ)
- Part II
  - Artificial Neural Networks (ANN)
  - Multi-Layer Perceptron Networks (MLP)
- Part III
  - Advanced MLPs
- Part IV
  - Model evaluation

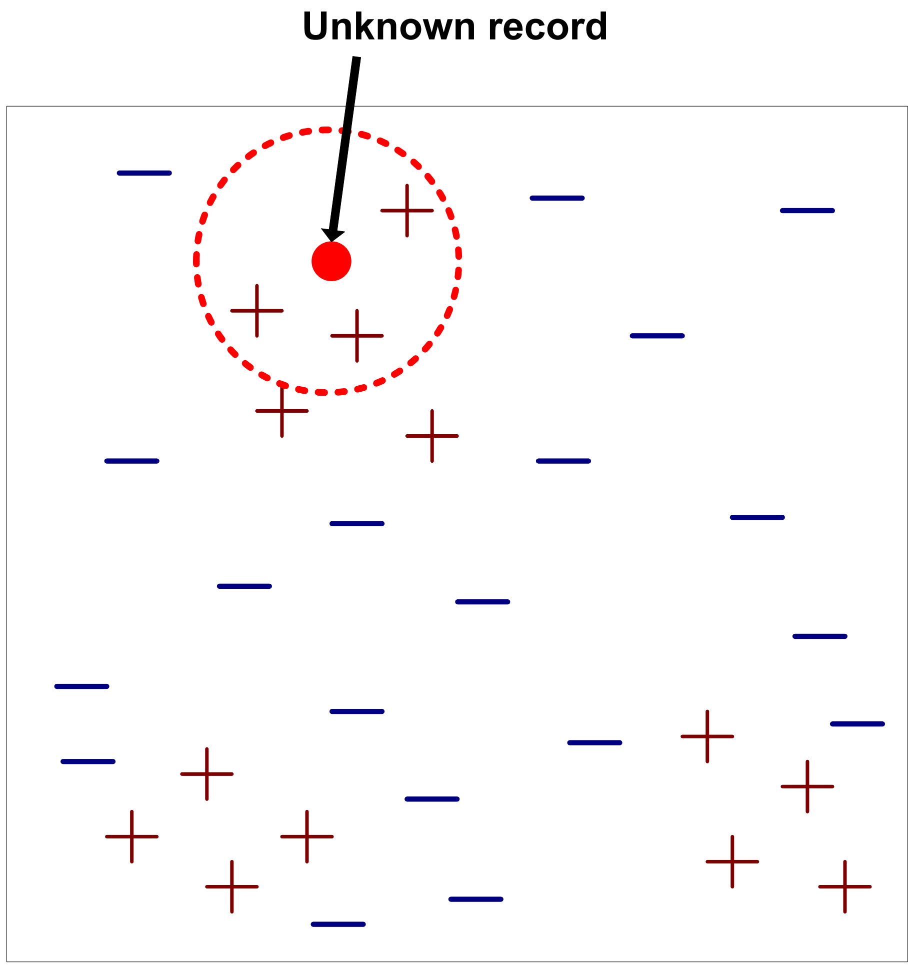
Classification via Machine Learning

- There are numerous classification methods. Methods based on Artificial Neural Networks (ANNs) are a focus in this lecture.
  - Later in the session we will also consider non-neural classifiers such as Decision Tree classifiers, Support Vector classifiers.
- There exist a large number of machine learning classifiers. Some of the more popular ones in Data Mining are:
  - Artificial Neural Networks (i.e. MLPs)
  - LVQ
  - KNN

Notes:

- KNN is not a machine “learning” algorithm.
- LVQ can be considered the supervised version of Kmeans.

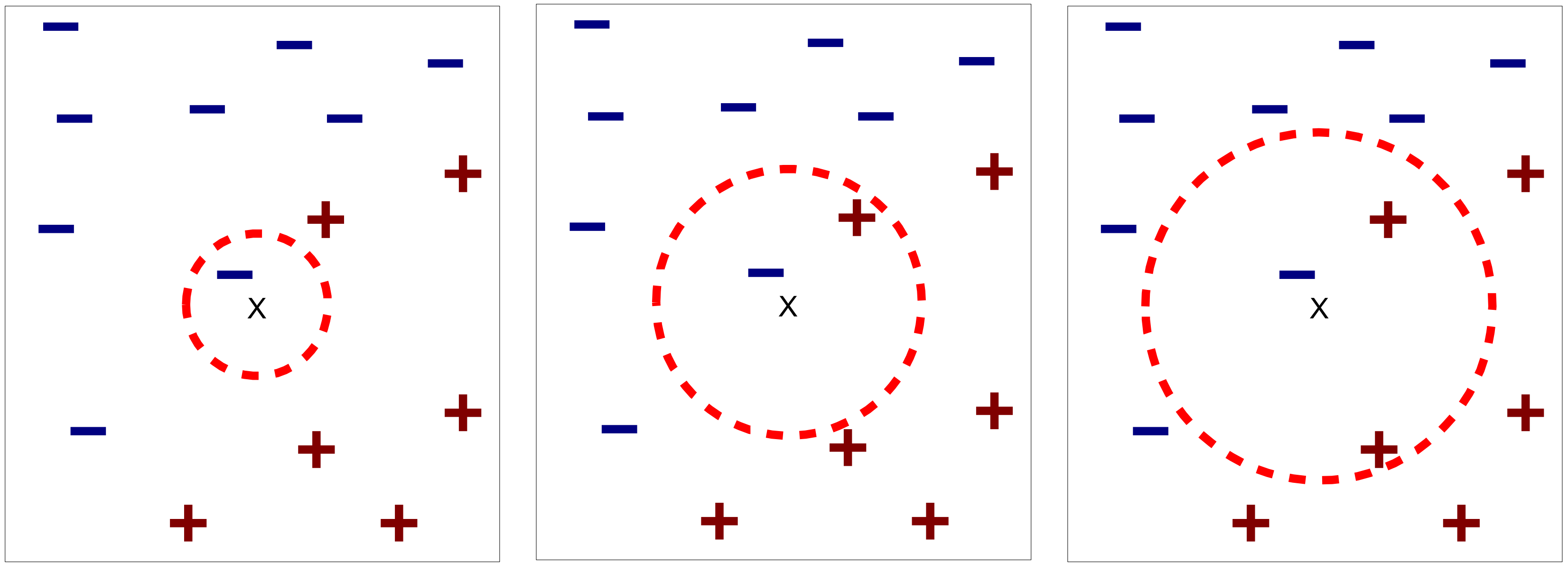
Nearest-Neighbor Classifiers



- Requires three things
  - The set of stored records
  - Distance Metric to compute distance between records
  - The value of *k*, the number of nearest neighbors to retrieve

- To classify an unknown record:
  - Compute distance to other training records
  - Identify *k* nearest neighbors
  - Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

Definition of Nearest Neighbor



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

Nearest Neighbor Classification

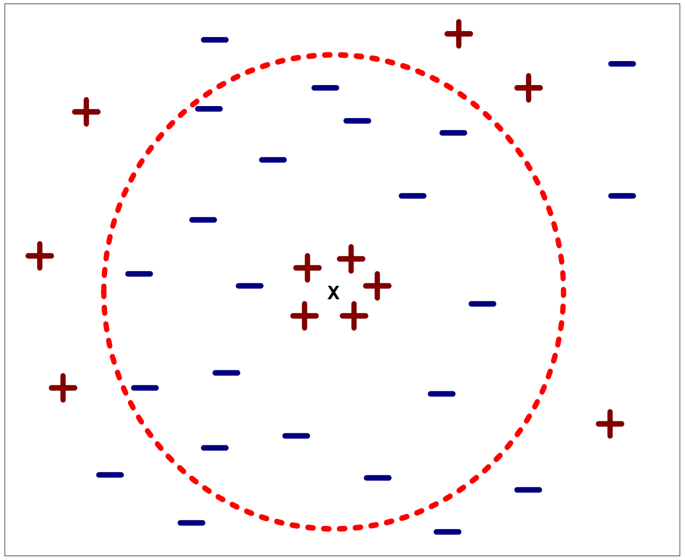
- Compute distance between two points:
  - Euclidean distance

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

- Determine the class from nearest neighbor list
  - take the majority vote of class labels among the k-nearest neighbors
  - Weigh the vote according to distance
    - weight factor, $w = 1/d^2$

Nearest Neighbor Classification…

- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes
  - Computational cost often increases when k increases



Nearest Neighbor Classification…

- The issue with scale
  - Attributes may have to be scaled to prevent distance measures from being dominated by some of the attributes
  - Example:
    - height of an adult may vary from 1.5m to 1.9m
    - weight of an adult may vary from 65kg to 130kg
    - income of a person may vary from $10K to $1M
- Possible solutions
  - Z-score normalization
  - Min-max normalization

Nearest Neighbor Classification…

- Problem with Euclidean measure:
  - High dimensional data
    - curse of dimensionality
  - Can produce counter-intuitive results

1 1 1 1 1 1 1 1 1 1 1 0		1 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1	vs	0 0 0 0 0 0 0 0 0 0 0 1
d = 1.4142		d = 1.4142

- Solution: Normalize the vectors to unit length

Nearest neighbor Classification…

- k-NN classifiers are lazy learners
  - It does not build models explicitly
  - Classifying unknown records are relatively expensive (weakness)

Learning Vector Quantization (LVQ)

LVQ addresses shortcomings of the k-nearest neighbor.

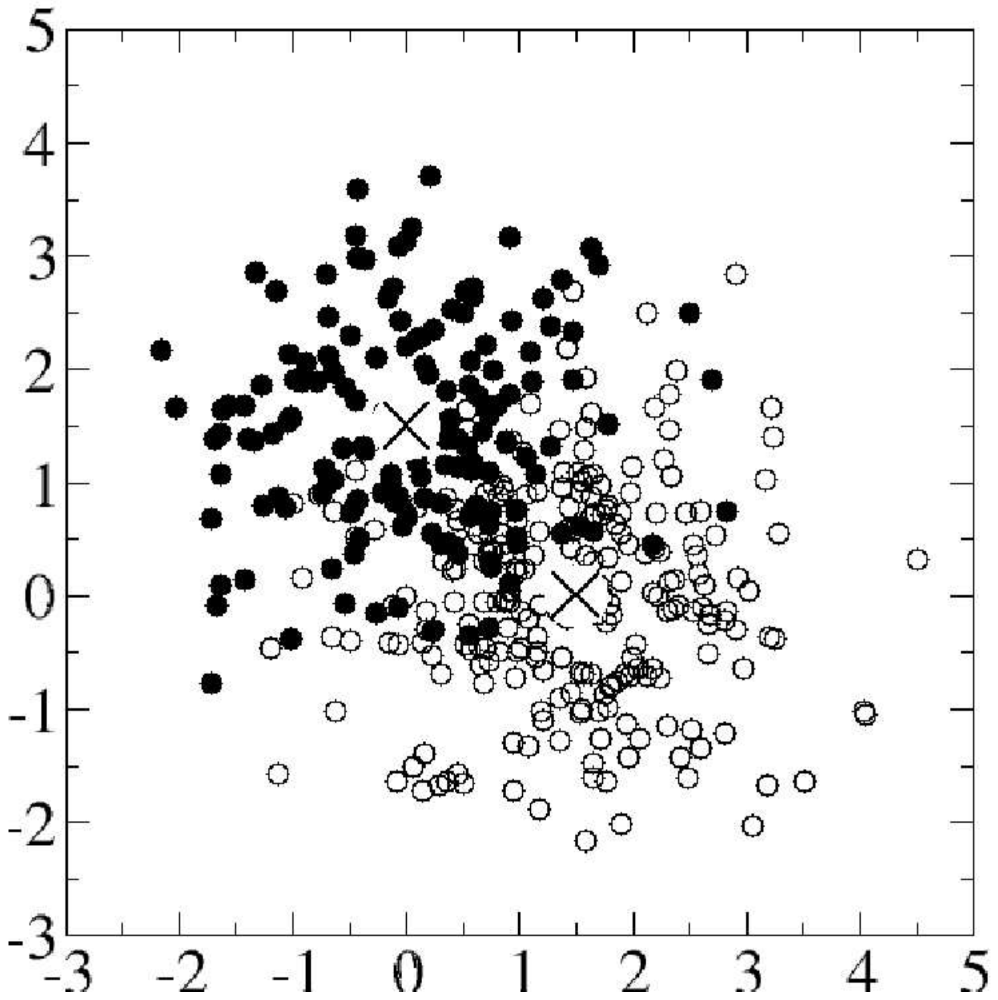
- Given n input/target pairs $\{\mathbf{x}_1, \mathbf{c}_1\}, \{\mathbf{x}_2, \mathbf{c}_2\}, \ldots, \{\mathbf{x}_n, \mathbf{c}_n\}$
- Initialize k-codebook vectors $w_1, \ldots, w_k$ (by randomly choosing k input/target samples)
- Repeat for a pre-set number of iterations:
  - Select an input/target pair $\{x_i, c_i\}$
  - Compute best matching codebook $s = \arg\min_k \|x_i - w_k\|$
  - Update the winning codebook

$$w_{s(t+1)} = \begin{cases} w_s(t) + \alpha(t)[x_i - w_s(t)] & \text{if class of } w_s = c_i \\ w_s(t) - \alpha(t)[x_i - w_s(t)] & \text{else} \end{cases}$$

Learning Vector Quantization (LVQ)

- Thus, LVQ trains a set of prototype vectors.
- These prototypes are then a representation of the various classes of a given dataset.
- This reduces the dimensionality of the problem when obtaining a class membership of a given test pattern.

Learning Vector Quantization (LVQ)



Example: Two prototype vectors (the crosses) in a two dimensional space defined by data belonging to two different classes.

To find the class membership of a new pattern, we simply compute the nearest prototype vector.

Learning Vector Quantization (LVQ)

- The presented algorithm is known as LVQ2.1. There is a refinement called LVQ3 which aim at enhancing the accuracy of the system.
  - In practise, LVQ2.1 is often the better choice.
  - Please consult the literature to understand LVQ3.
- LVQ is a simple **machine learning** approach to achieve **classification**. LVQ is said to engage a **competitive** , winner takes all, learning scheme.
  - Another competitive learning scheme is Kmeans and the Self-Organizing Maps though these are trained unsupervised.

Artificial Neural Networks (ANN)

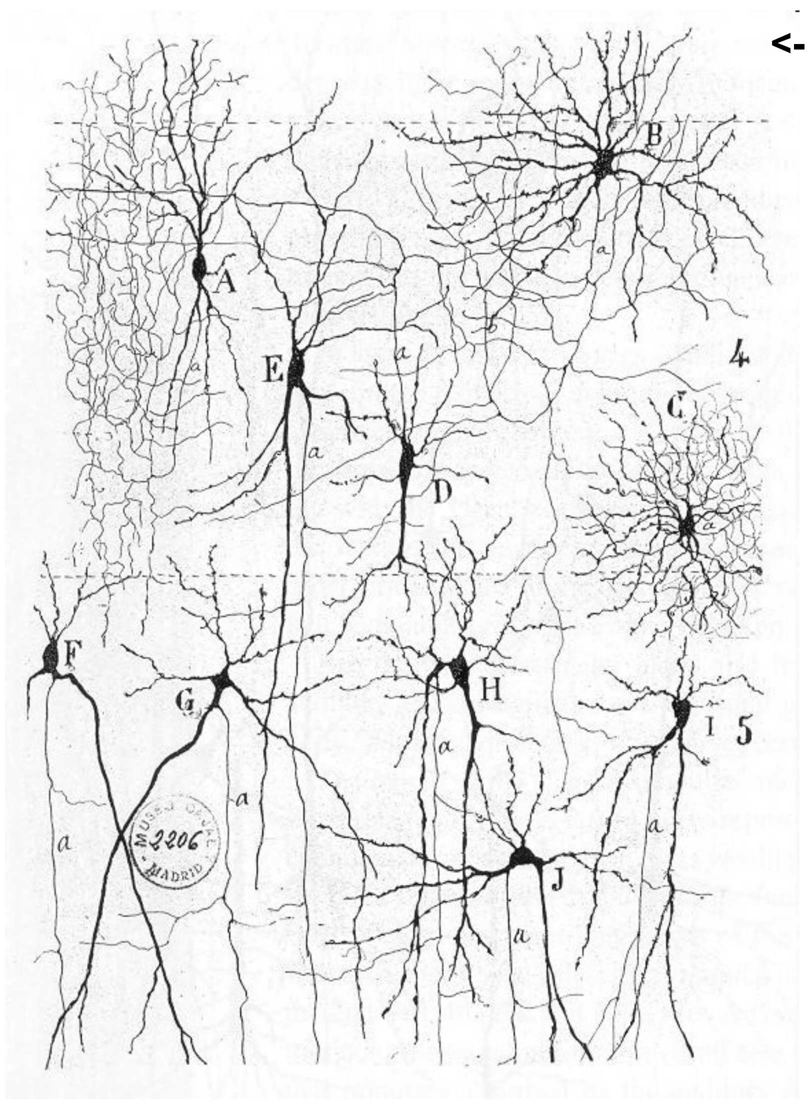
Artificial Neural Networks:

- Learn from data
- Simulate biological counterpart (the brain)
- Are massively parallel systems
- Tolerant to faults (within the system) and noise
- NN consist of Neurons and Axons, and Synapses
- ANN consist of simulated Neurons and Weights

Artificial Neural Networks

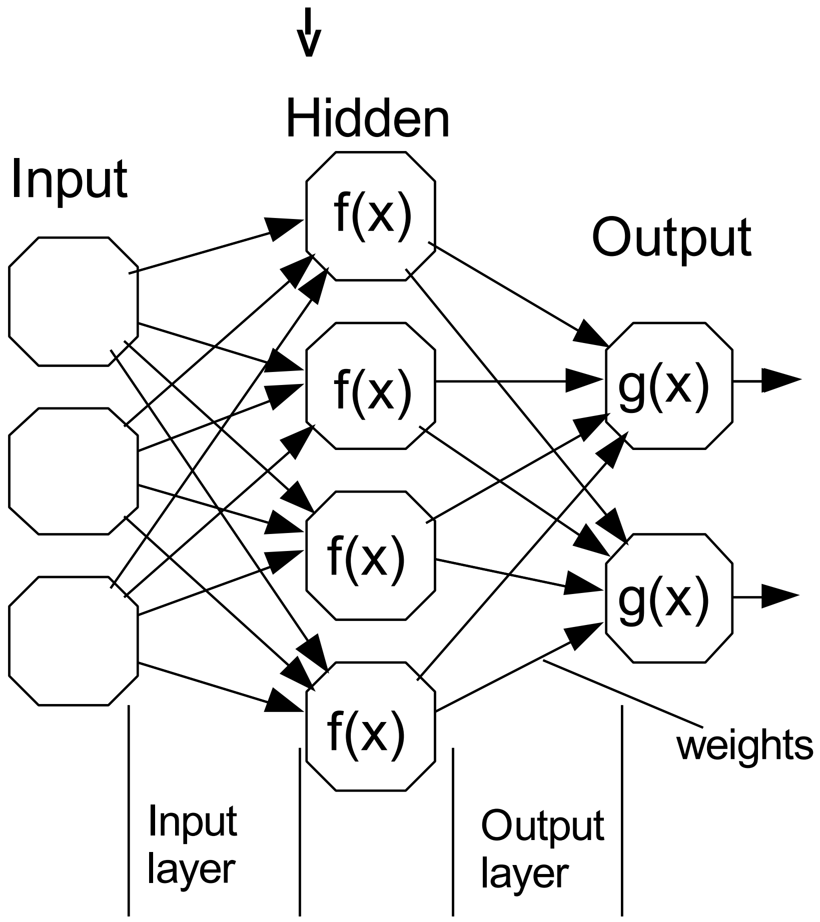
An example: The human brain consists of a large number of **neurons**. These neurons allow us to “learn”, “understand”, and “memorize”. But how do the neurons achieve this?

Simulation of Neural Networks



<- Biological Neural Network

Artificial Neural Network (weights omitted)



Simulation of Neural Networks

- The weights are adjustable, everything else is fixed.
- Adjusting the weights such that for any given input the network produces a desirable output is the task of a **training algorithm**.
- The training algorithm requires:
    1. A way to initialize the weights.
    2. A way to determine the output of each neuron (the activation function).
    3. A way to adjust the weights.
- Difficulties with the 3rd aspect hindered the development of ANN for many years.

Neural Networks for Mining Data

Time

- 2013 Deep Learning (Bengio et.al)
- 2012 Convolutional Neural Networks (LeCun)
- 2011 Supervised learning of graph-of-graphs (Scarselli et.al)
- 2009 Unsupervised learning of graphs (Hagenbuchner et.al.)
- 2008 Supervised learning of graphs (Hagenbuchner et.al.)
- 2002 Graph-matching editSOM (Bunke et.al.)
- 1999 Unsupervised learning of tree-data(Hagenbuchner et.al.)
- 1996 Supervised learning of tree-data (Frasconi et.al.)
- 1989 Supervised learning of sequences (Elman)
- 1986 Supervised learning of fixed vectors (PDP).
- 1986 Un-Supervised learning of fixed vectors (T.Kohonen)
- 1982 Un-Supervised learning of fixed vectors (Hopfield)
- The Minsky and Papert (1969) hole
- 1957 Supervised learning of fixed vectors (Rosenfeld)

Artificial Neural Networks

There are two forms of Neural Networks

- Supervised (target values are available for some data from the domain)
  - Classification
  - Regression
  - Projection
- Unsupervised (target values are not available)
  - Clustering
  - Projection (dimension reduction)

There is a special class of NN which is both, supervised and unsupervised: The Auto-associative memory.

Artificial Neural Networks

- ANNs cover the following ability of the human brain:
  - To learn with the help of a teacher.
  - To organize itself into dedicated regions.
  - To memorize.

- One of the easiest ANNs is the Hopfield model. It can **memorize** data in a fault tolerant fashion.
  - Not covered in this subject.

Multi-layer Perceptron Networks

- An MLP consist of neurons (called perceptrons) which are organized in layers.
- The layers are fully connected by weighted connections (the weights).
- MLP are trained in a supervised fashion in two phases:
  1. Forward phase
  2. Backward phase

Multi-layer Perceptron Networks

The neuron:

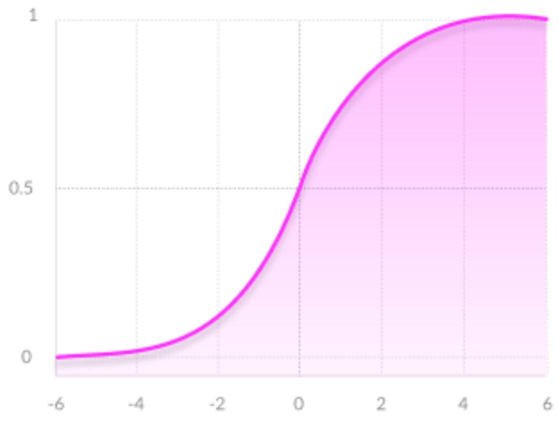
- Is connected to by other neurons. The connections are weighted.
  - The weights are initialized with small random values.
- Computes a weighted sum of all inputs, then produces a response.
- Uses an “activation” function to compute the response. Common activation functions are:

  Linear: $f(x):=x$ (great for neurons in the output layer)

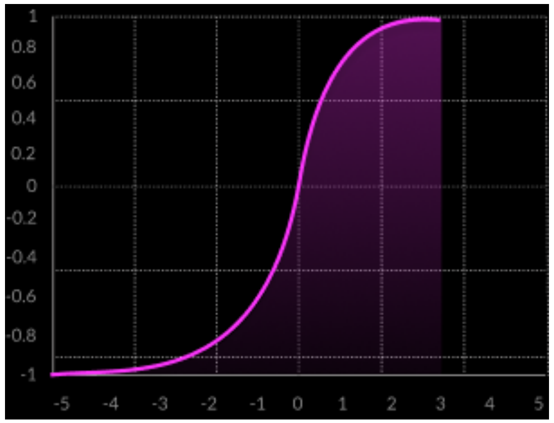
  $f(x):=x$ if $x>0$ else 0 (popular for networks with large number of hidden layers)

  Non-linear: $f(x):=\tanh(x)$ or $f(x):=1/(1+e^{-x})$ *(good for hidden neurons.)*

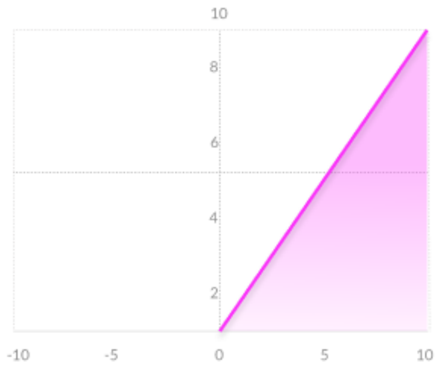
Commonly used activation functions



Sigmoid / Logistic



TanH / Hyperbolic Tangent



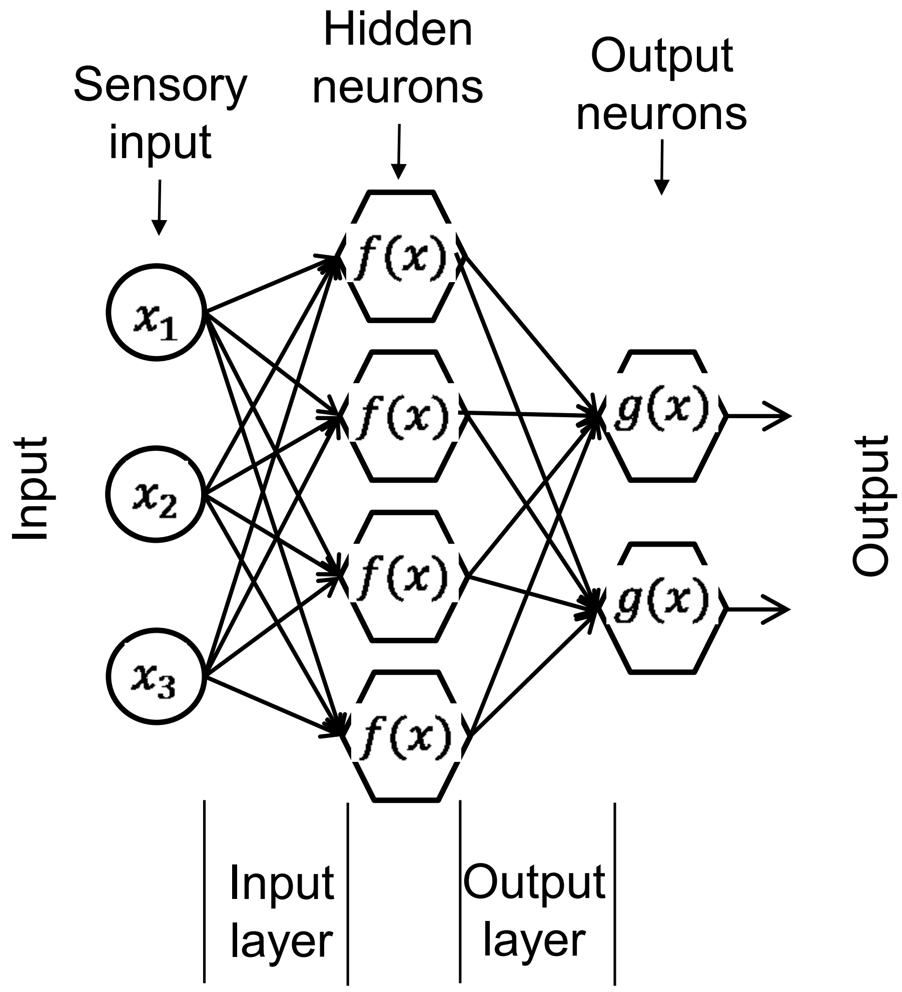
ReLU (Rectified Linear Unit)

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_k}} \; for\, j = 1, \ldots, k$$

Softmax

Example:

- A layered neural network with 6 neurons organized in two layers. The layers are fully connected.
- The interconnecting links are weighted.
  - Weights are not shown in this diagram.



Multi-layer Perceptron Networks

Forward phase:

For each neuron, starting with the input layer, and working towards the output layer, compute its output:

$$x_j^n = f(\sum_{i=0}^{m} w_{ji} x_i^{n-1})$$

,where f(.) is the activation function), w_{ij} is the weight connecting the *i*-th neuron with the j-th neuron, and x_i^{n-1} is the output of the i-th neuron in layer n-1.

The output at the output layer $\boldsymbol{o} = \boldsymbol{x}^{n=max}$ is the output of the network.

Multi-layer Perceptron Networks

- **Backward phase:**
  - The output of the network is compared with a given target value.
  - The network is trained with the aim to minimize a "Cost" function. Common is:

$$\boldsymbol{E(w) = \frac{1}{2}\|O - C\|}$$

, where ***c*** is the target of the input that produced ***o*** as an output.

- This can be done by using a *gradient descent* method:
- For each weight in the network, compute the derivative of function E (with respect to this weight), then change the size of this weight into the negative direction of the gradient.

Multi-layer Perceptron Networks

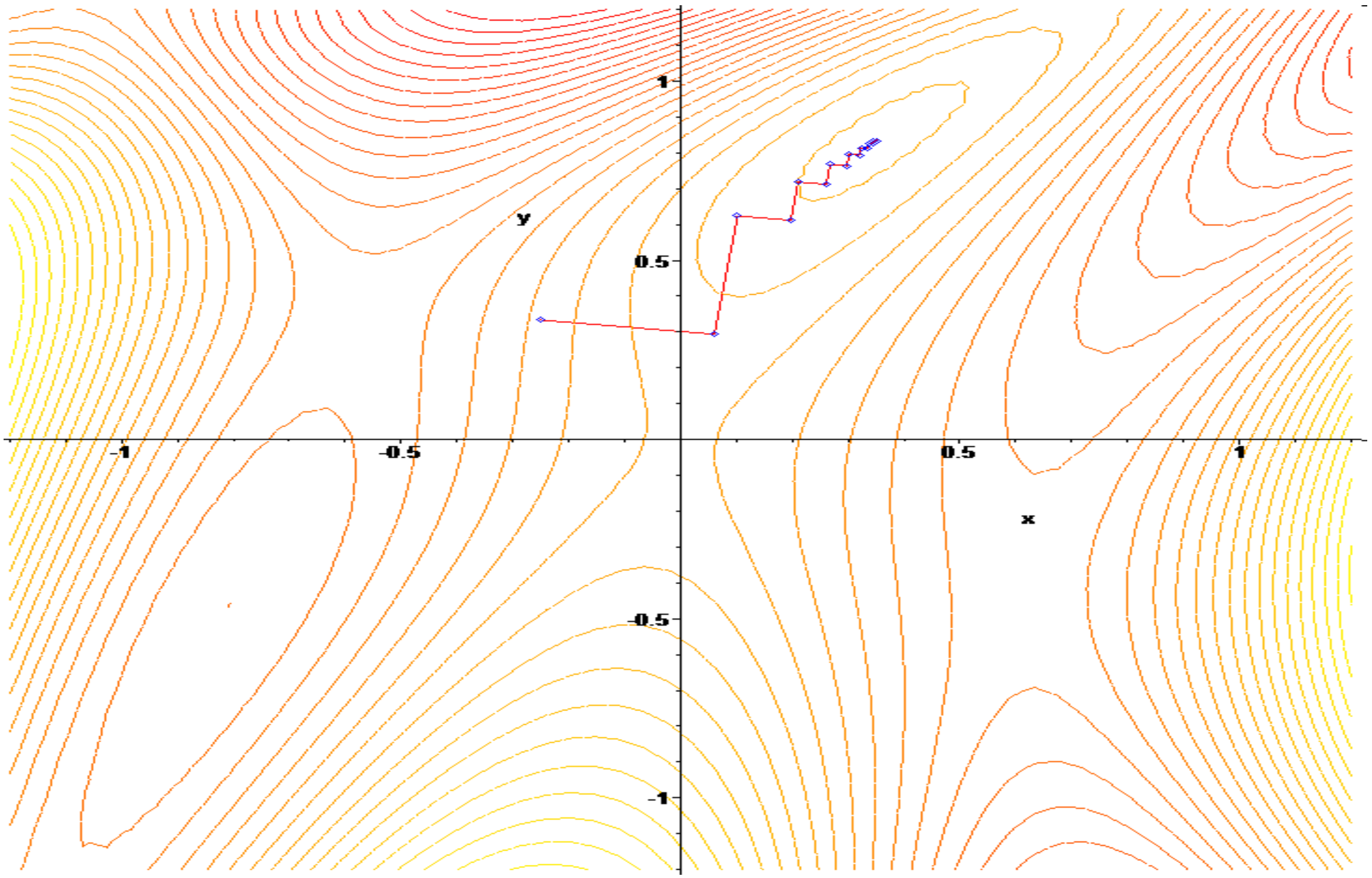
- The MLP is a parameterized model. The weight in this model can be adjusted.
- The purpose of the Error Back-propagation Algorithm is to minimize the error of a given cost function by adjusting the weights of the MLP.
- We do not know a-priori the amount by which each weight needs to be changed in order to minimize the error. But we can know the direction in which each weight needs to be changed so as to reduce the error.

Multi-layer Perceptron Networks

- The direction of a weight change can be computed by using the sign of the gradient of the cost function with respect to each of the weights.
- Change the weight into the negative direction of the gradient by a small amount.
- Repeating the forward and backward steps for many iterations will reduce the output error.
- The method propagates the error at the output layer back through to the weights in all of the network layers until the input layer is reached. Hence the name error-back propagation algorithm.

$$\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}},\ \text{where } 0 < \alpha \ll 1 \text{ is a learning rate.}$$

Gradient descent learning



Multi-layer Perceptron Networks

In practice:

- The output neurons often use softmax activation function.
- Hidden neurons always use a non-linear activation function. The most common one is ReLU

- An MLP can consist of many layers. Two- and three layer architectures are the most common in DM.
  - Evidence suggests that more layers are better.
  - But too many layers pose computational challenges.

Multi-layer Perceptron Networks

- **The training algorithm:**

```
Initialize the weights in the network with random values
Repeat
    For each example $\boldsymbol{x}$ in the training set
        $\boldsymbol{o}$ = neural-net-output(network, $\boldsymbol{x}$) ;    #forward pass
        $\boldsymbol{e}$ = Calculate error ($\boldsymbol{c}$ - $\boldsymbol{o}$).
        Compute $\Delta w$ for all weights in all layers by using the gradient
            descent method;
        Update all weights by their corresponding $\Delta w$
Until network error does no longer decrease.
```

Multi-layer Perceptron Networks

- A trained network is said to be a “model” which “encodes” the training data.
- Once trained the model can be applied to unseen data. The network will then produce an output in accordance to the input pattern.
  - This requires only one pass through the forward phase.
  - Linear computational complexity
- However, the MLP is a “black box”: We do not now (nor understand) how the model has encoded the information (there is no rule which we can extract).
  - Poor interpretability or explainability

Multi-layer Perceptron Networks

Limitations of MLP:

- Asymptotic algorithm; error reduction slows down with the approach to a minimum.
- Converges to a local minimum which is not necessarily the global minimum.
- Requires setting of network parameters, known as hyperparameters (number of hidden layers, number of neurons in each layer, initialization of weights)

These limitations can be addresses by:

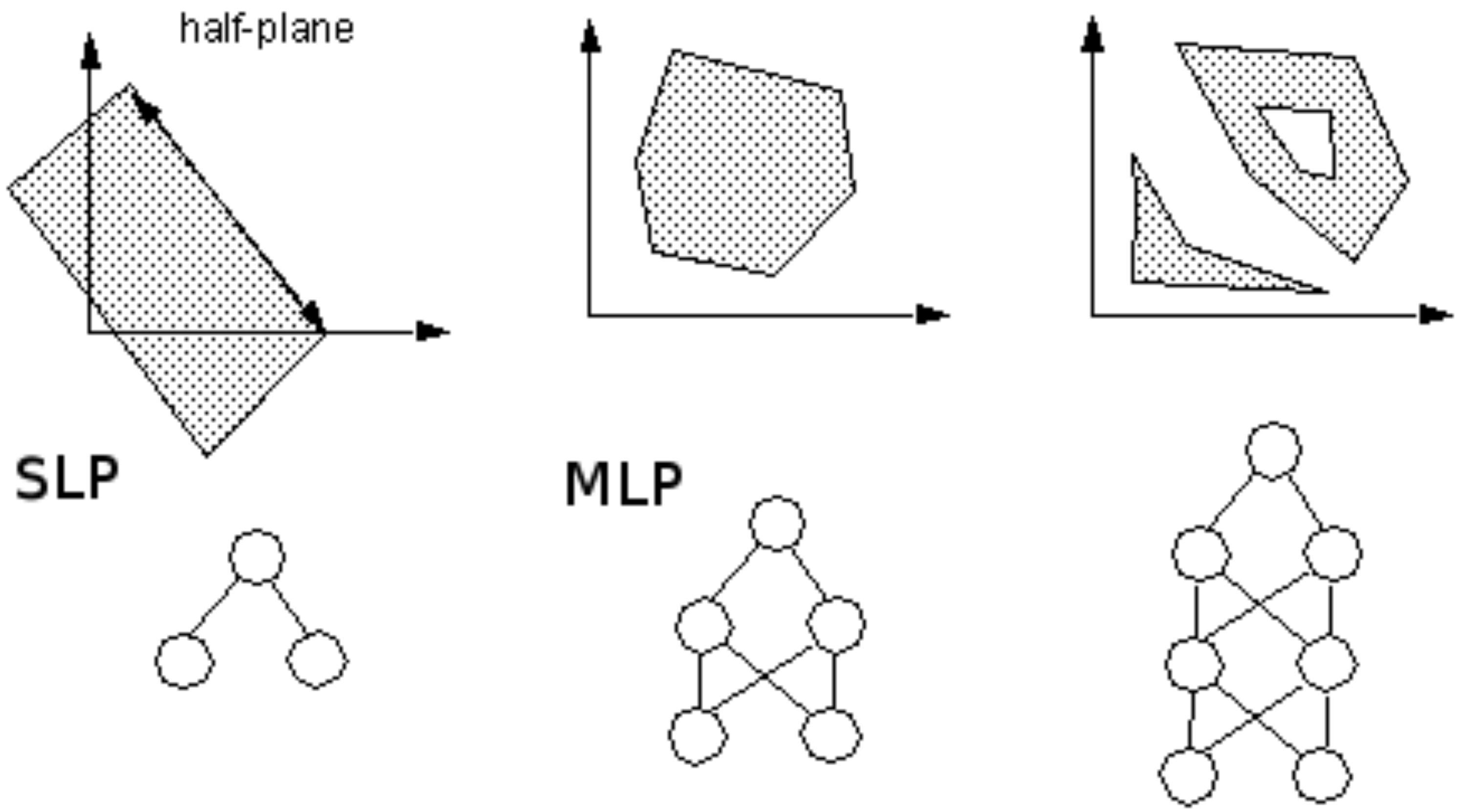
- Using a momentum term or adaptive learning rates.
- Careful network initialization (trial and error is OK)
- Use of Cascade correlation networks.

Multi-layer Perceptron Networks

- MLPs are widely used for classification and regression problems.
- Are proven to solve **any** continuously differentiable function to **any** precision.
- Can be used as a convenient way to perform dimension reduction (by using an associative memory architecture)
- Have some limitations for “knowledge discovery” tasks (they are trained supervised, and hence, some a-priory knowledge is required).
- Unsupervised machine learning methods are more suitable for knowledge discovery tasks. For example: Self-Organizing Maps.

Role of the layers in an MLP

- The greater the number of layers and the more neurons in each layer the greater is the ability to model complex learning problems.



Computational power

- We have seen that the MLP learns a parameterized function, and that the number of parameters affect the capability of the MLP.
- In general, the more parameters we have, the better the (training) performance of the MLP at the cost of more computation time.
- However, the generalization ability can be impaired by having too many parameters -> Overfitting
- How best to evaluate the performance of a trained system (the model)?

Multi-layer Perceptron Networks

- MLPs can deal with vectorial information.
- What about more complex data structures such as sequences (i.e. time series information), trees (i.e. parse trees), and graphs (i.e. chemical molecules)?
- A classic approach for dealing with complex structures is to squash those into a vector form before feeding them to an MLP.

Advanced MLP networks

- **Elman Networks:** Encodes time series information (i.e., speech recognition learning problems)
- **RMLP:** Encodes tree structured information (i.e., XML document classification problems using parsing trees).
- **GNN:** Encodes general graph structured information (i.e., spam detection in the World Wide Web).

MLP, alternate notation

For the following, it is more convenient to use a matrix notation as follows:

$$\boldsymbol{o} = F_p(C\boldsymbol{y})$$
$$\boldsymbol{y} = F_n(A\boldsymbol{x}),$$

where $\boldsymbol{x}$ is an input vector, $\boldsymbol{y}$ is the output of a hidden layer, $\boldsymbol{o}$ is the output of the network, A is a matrix of weights connecting the input with the hidden layer, C is a matrix of weights connecting the hidden with the output layer, and F_n is an n-dimensional vector with all elements set to $f(\cdot)$.

Elman Network

- An MLP can only deal with fixed sized vectors. But there are many problem domains for which the input vectors are not fix in size. For example, time series information such as in speech processing, financial forecasting, text processing, and many more.

- An **Elman Network** is an extension of the MLP which is trained by using a shifting (fixed sized) window over the input sequence. This window serves as an input to the network. In addition, the output of the hidden layer is forwarded to the input layer when processing the next time window.

Elman Network

Formally, an Elman Network can be expressed as follows:

$$\boldsymbol{o} = F_p(C\boldsymbol{y})$$
$$\boldsymbol{y} = F_n(A\boldsymbol{x} + Bq^{-1}\boldsymbol{y}),$$

where q^{-1} is a shift operator such that $q^{-1}\mathbf{y}$ denotes the availability of the hidden layer output **y** from the previous iteration. In other words, it denotes the relationship of the current input with the previous input.

Elman Network

- The Elman Network processes an input sequence iteratively. The same gradient error descent method can be applied to train the system. A difference is that the gradient needs to be propagated back iteratively. The longer the sequence, the longer the path for which the gradient needs to be propagated back. Frasconi, Baldi proved that this can lead to **long term dependency** problems which can be addressed through deep learning strategies (not covered in this subject).
- The learning algorithm is gradient based as for MLP. But since the gradient needs to be passed back through the sequence, and hence, the algorithm is also known as BPTT (back-propagation through time).

Elman Network

$q^{-1}\mathbf{y}$ was the network's internal response to the processing of the previous time window. In other words, $q^{-1}\mathbf{y}$ refers to the state of the network from a previous time instance. Hence, we will refer to this simply as state.
The Elman Network can only deal with **directed sequences**.
There are problem domains which deal with more complex structures such as in document processing and document parsing, robot navigation, chinese character recognition, etc. In these cases, the data is represented as a tree data structure.

Observation: A tree data structure with a maximum out-degree of 1 is equivalent to a directed sequence. Hence, directed sequences are a special case of trees.

Recursive MLP (RMLP)

A subsequent extension of Elman's idea to tree structured data is possible if the maximum out-degree *c* is known. In this case

Formally, RMLP can be described as follows:

$$\boldsymbol{o} = F_p(C\boldsymbol{y})$$
$$\boldsymbol{y} = F_n(A\boldsymbol{x} + \boldsymbol{B}q^{-1}\boldsymbol{y}),$$

Where $\boldsymbol{x}$ refers to the numeric label attached to a node in a tree $\boldsymbol{B}q^{-1}\boldsymbol{y}$ is a shorthand notation of the following:

$$\boldsymbol{B}q^{-1}\boldsymbol{y} = [B_1, B_2, \cdots, B_c]\begin{bmatrix} q_1^{-1}\boldsymbol{y}_1 \\ q_2^{-1}\boldsymbol{y}_2 \\ \vdots \\ q_c^{-1}\boldsymbol{y}_c \end{bmatrix}$$

Frasconi et.al. proposed this approach. He coined its learning algorithm BPTS.

Back Propagation Through Structure

RMLP processes nodes in a tree, one at a time, and takes as input the networks' state for all of the nodes' children. It can be seen clearly that if $c = 1$ then RMLP will be an equivalent to an Elman Network.

Since the state of all children needs to be computed first when processing a given node, this dictates the processing of nodes in the inverse topological order (from leaf-to-root nodes). This is a recursive procedure, and hence, this class of networks is referred to as **Recursive Neural Networks**.

Training the RMLP

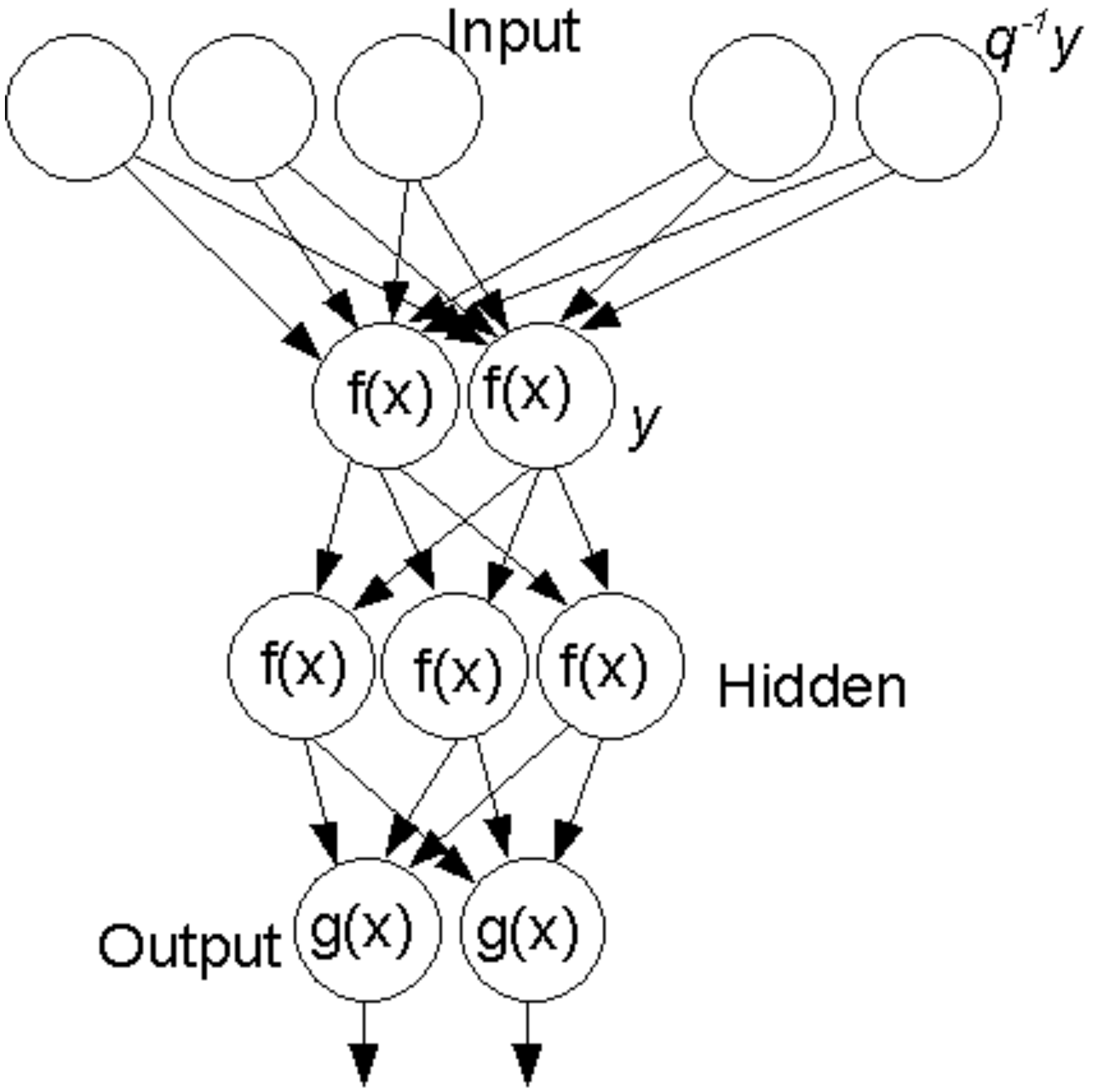
The principle of minimizing the cost function by following the gradient descent method remains unchanged. However, the gradient needs to be propagated back recursively through the same network over and over. This lead to the formation of the term **unfolding architecture**.

RMLP

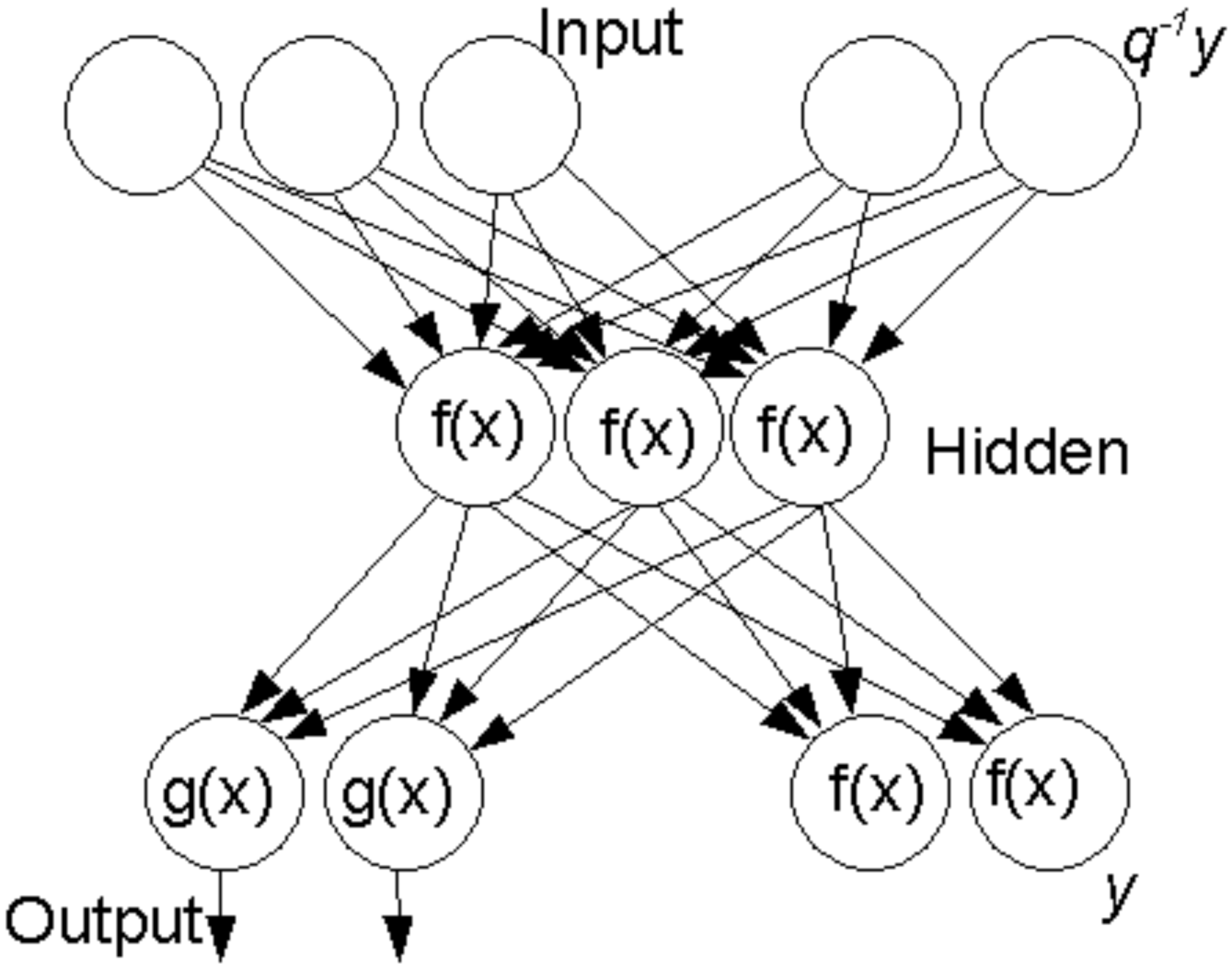
There are several architectural variantions of RMLP. For example:

- The output MLP architecture
- The state MLP architecture

The state MLP architecture



The output MLP architecture



Some Limitations of the BPTS

- Can only encode causal relationships.
- Can “only” deal with tree data structures. But what about problems involving chemical molecules, image processing applications, the world wide web, etc. are a few examples of problem domains which are represented by cyclic directed or undirected graphs.
- The BPTS can not deal with cyclic dependencies between nodes in a graph.
- Graph Neural Networks (GNNs) overcome these problems though the GNN is not covered in this subject.

Model Evaluation for supervised models

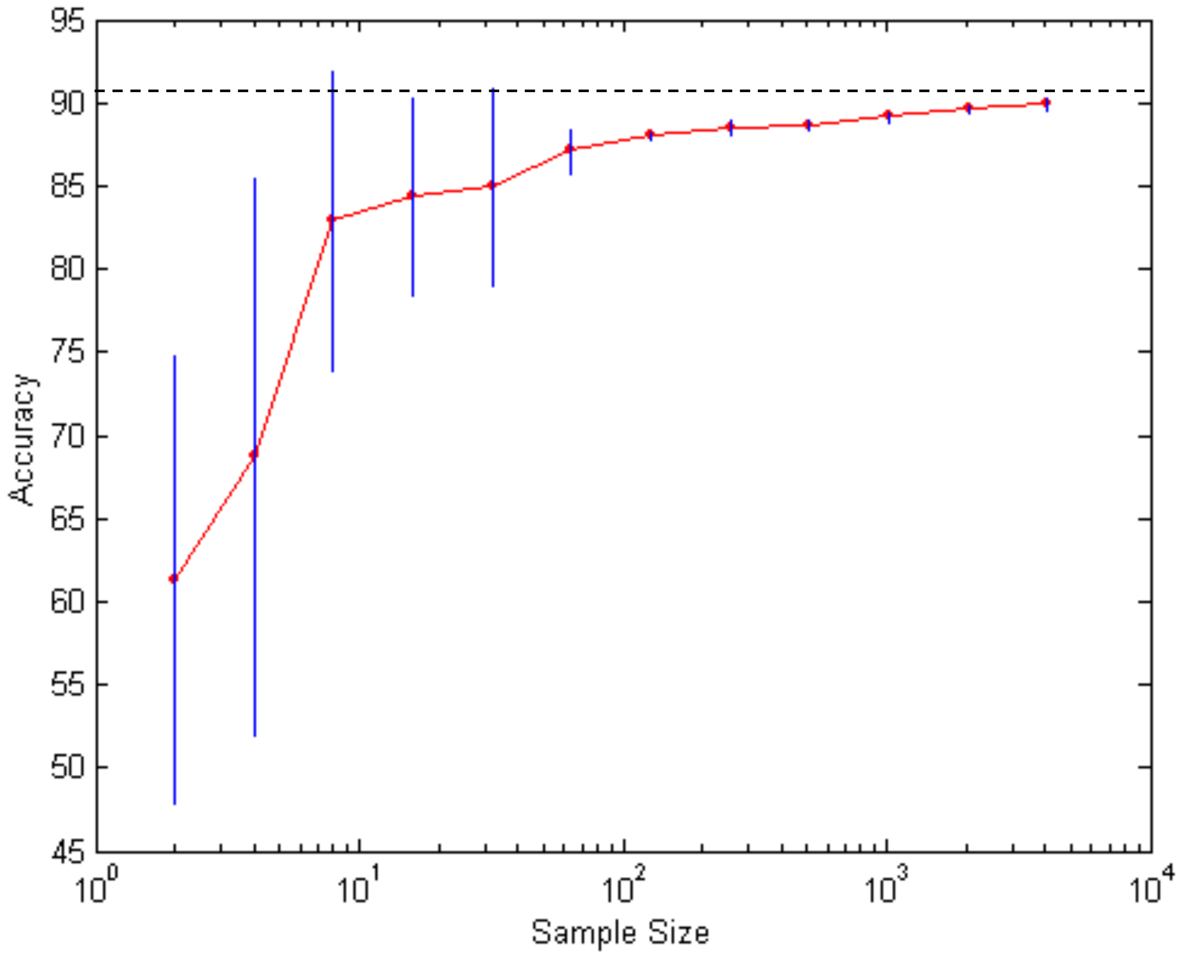
- Methods for Performance Evaluation
  - How to obtain reliable estimates?
- Metrics for Performance Evaluation
  - How to evaluate the performance of a supervised model?
- Methods for Model Comparison
  - How to compare the relative performance among competing models?

Methods for Performance Evaluation

- Performance of a model may depend on other factors besides the learning algorithm:
  - Class distribution
  - Cost of misclassification
  - Size of training and test sets

- We often require a portion of the dataset to serve us for testing the model (the remainder of the dataset is used for creating the model)

Learning Curve



- Learning curve shows how accuracy changes with varying sample size
- Requires a sampling schedule for creating learning curve:
  - Arithmetic sampling (Langley, et al)
  - Geometric sampling (Provost et al)

Effect of small sample size:

- Bias in the estimate
- Variance of estimate

Methods of Estimation

- Holdout
  - I.e. reserve 2/3 for training and 1/3 for testing
- Random subsampling
  - Repeated holdout
- Cross validation
  - Partition data into k disjoint subsets
  - k-fold: train on k-1 partitions, test on the remaining one
  - Leave-one-out: k=n-1, test on the remaining one

Metrics for Performance Evaluation

- Focus on the predictive capability of a model
  - Rather than how fast it takes to classify or build models, scalability, etc.
- Confusion Matrix:

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	a	b
	Class=No	c	d

a: TP (true positive)

b: FN (false negative)

c: FP (false positive)

d: TN (true negative)

Metrics for Performance Evaluation…

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

- Most widely-used metric:

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

Limitation of Accuracy

- Consider a 2-class problem
  - Number of Class 0 examples = 9990
  - Number of Class 1 examples = 10

- If model predicts everything to be class 0, accuracy is 9990/10000 = 99.9 %
  - Accuracy is misleading because model does not detect any class 1 example

Cost Matrix

	PREDICTED CLASS		
ACTUAL CLASS	C(i|j)	**Class=Yes**	**Class=No**
	Class=Yes	C(Yes|Yes)	C(No|Yes)
	Class=No	C(Yes|No)	C(No|No)

C(i|j): Cost of misclassifying class j example as class i

Computing Cost of Classification

Cost Matrix	PREDICTED CLASS		
	C(i|j)	+	-
ACTUAL CLASS	+	-1	100
	-	1	0

Model M_1	PREDICTED CLASS		
		+	-
ACTUAL CLASS	+	150	40
	-	60	250

Model M_2	PREDICTED CLASS		
		+	-
ACTUAL CLASS	+	250	45
	-	5	200

Accuracy = 80%

Cost = 3910

Accuracy = 90%

Cost = 4255

Cost-Sensitive Measures

$$\text{Precision(p)} = \frac{a}{a+c}$$

$$\text{Recall(r)} = \frac{a}{a+b}$$

$$\text{F-measure(F)} = \frac{2a}{2a+b+c}$$

- Precision is biased towards C(Yes|Yes) & C(Yes|No)
- Recall is biased towards C(Yes|Yes) & C(No|Yes)
- F-measure is biased towards all except C(No|No)

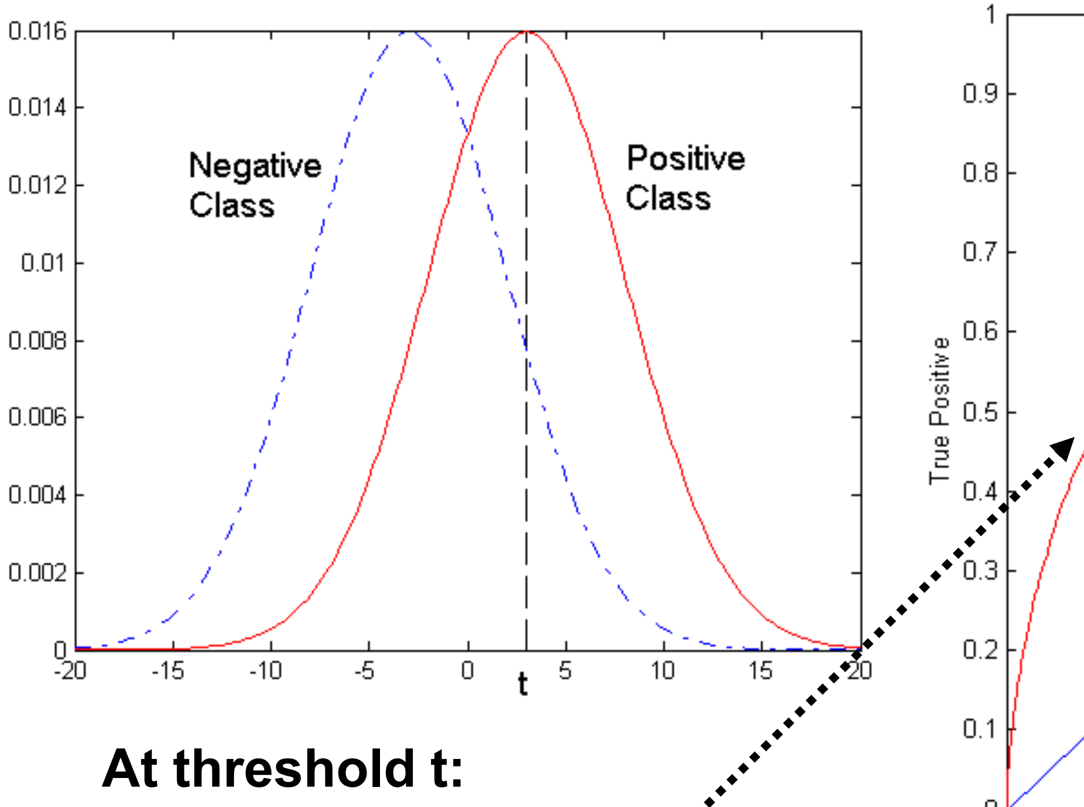
$$\text{Weighted Accuracy} = \frac{w_1 a + w_4 d}{w_1 a + w_2 b + w_3 c + w_4 d}$$

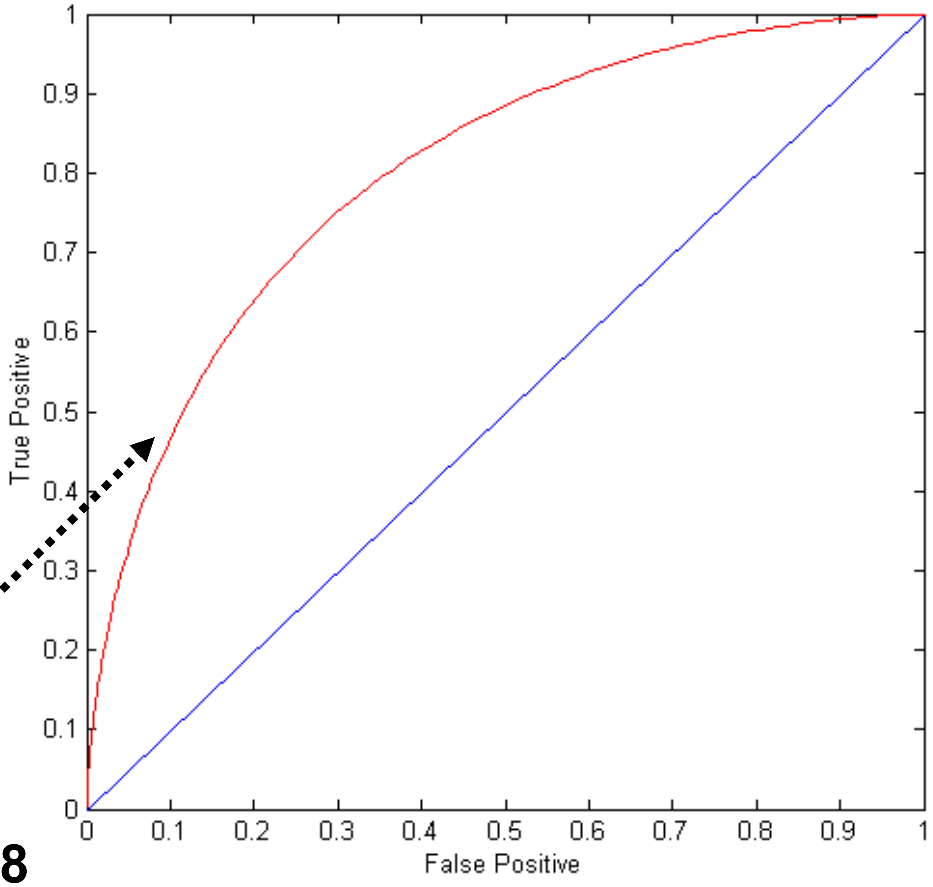
ROC (Receiver Operating Characteristic)

- Developed in 1950s for signal detection theory to analyze noisy signals
  - Characterize the trade-off between positive hits and false alarms
- ROC curve plots TP (on the y-axis) against FP (on the x-axis)
- Performance of each classifier represented as a point on the ROC curve
  - changing the threshold of algorithm, sample distribution or cost matrix changes the location of the point
- Has become very popular for evaluating ANNs.

ROC Curve

- 1-dimensional data set containing 2 classes (positive and negative)
- any points located at x > t is classified as positive





At threshold t:

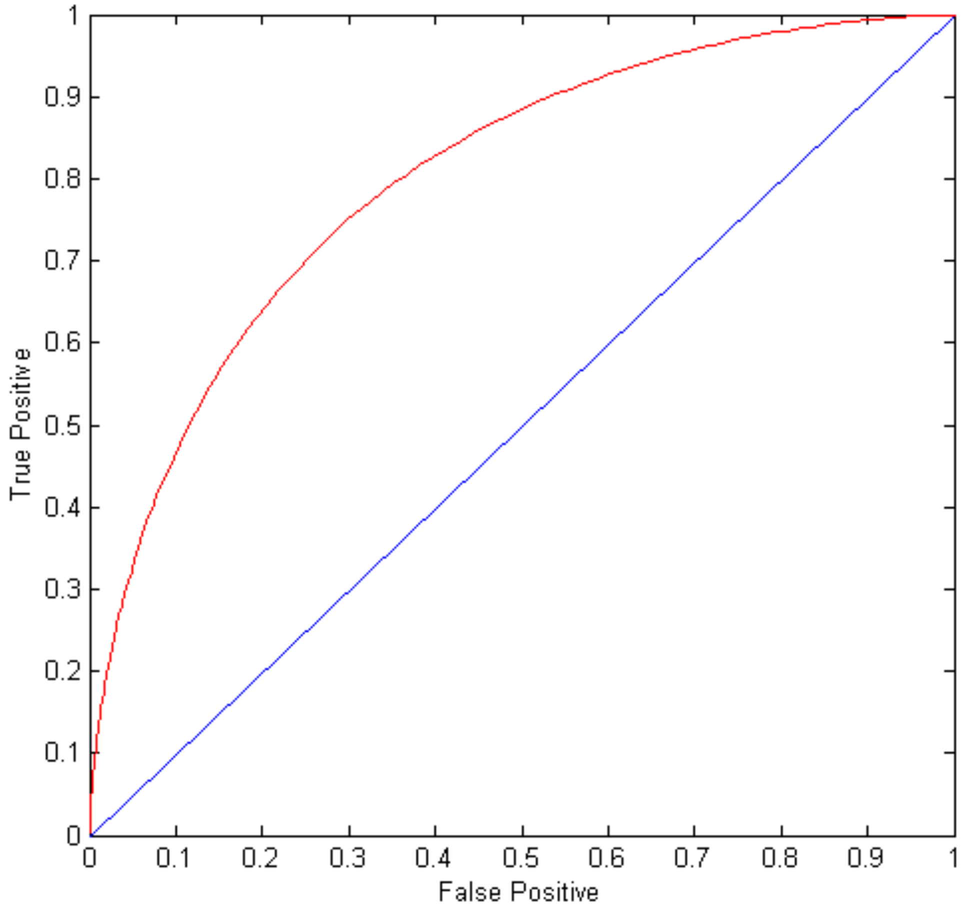
TP=0.5, FN=0.5, FP=0.12, FN=0.88

ROC Curve

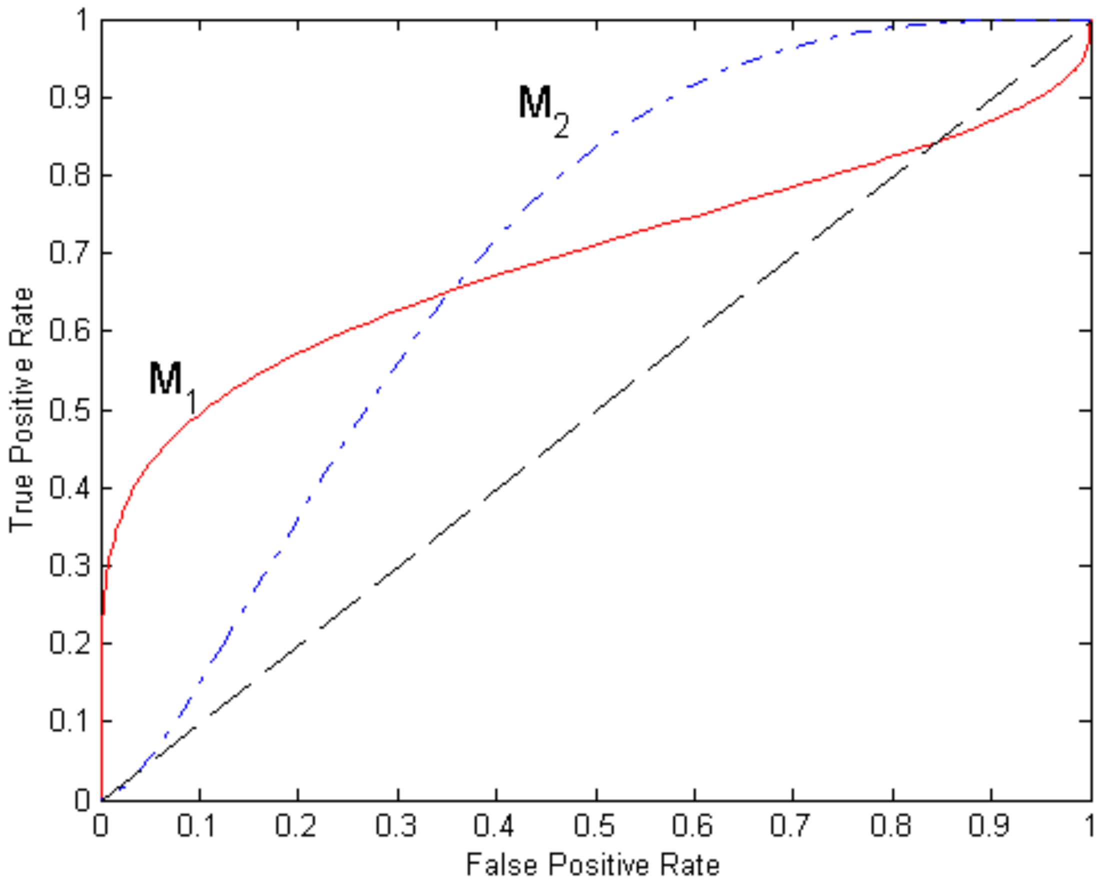
(TP,FP):

- (0,0): declare everything
  to be negative class
- (1,1): declare everything
  to be positive class
- (1,0): ideal

- Diagonal line:
  - Random guessing
  - Below diagonal line:
    - prediction is opposite of the true class



Using ROC for Model Comparison



- No model consistently outperform the other
  - M_1 is better for small FP rate
  - M_2 is better for large FP rate (FPR)
- Area Under the ROC curve
  - Ideal:
    - Area = 1
  - Random guess:
    - Area = 0.5

How to Construct an ROC curve

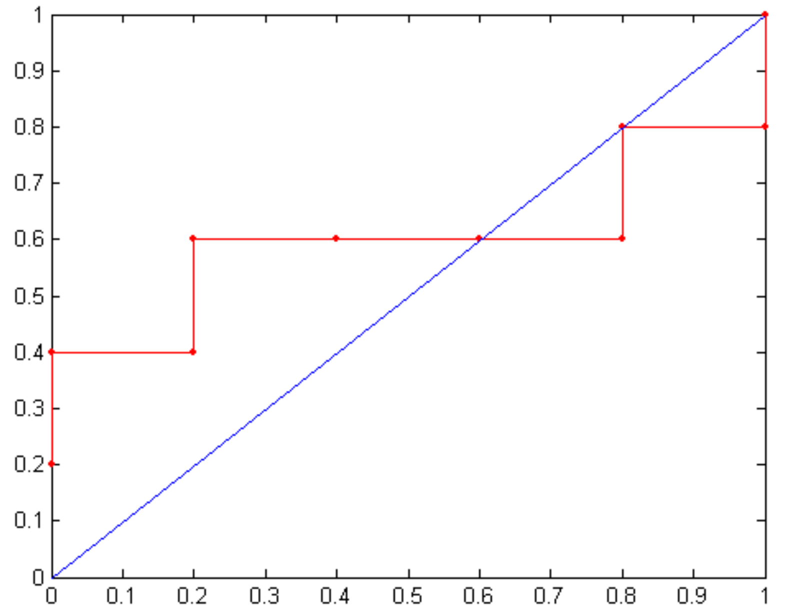
Instance	P(+|A)	True Class
1	0.95	+
2	0.93	+
3	0.87	-
4	0.85	-
5	0.85	-
6	0.85	+
7	0.76	-
8	0.53	+
9	0.43	-
10	0.25	+

- Use classifier that produces posterior probability for each test instance P(+|A)
- Sort the instances according to P(+|A) in decreasing order
- Apply threshold at each unique value of P(+|A)
- Count the number of TP, FP, TN, FN at each threshold
- TP rate, TPR = TP/(TP+FN)
- FP rate, FPR = FP/(FP + TN)

How to construct an ROC curve

Class	+	-	+	-	-	-	+	-	+	+	
Threshold >=	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
TP	5	4	4	3	3	3	3	2	2	1	0
FP	5	5	4	4	3	2	1	1	0	0	0
TN	0	0	1	1	2	3	4	4	5	5	5
FN	0	1	1	2	2	2	2	3	3	4	5
→ TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4	0.4	0.2	0
→ FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2	0	0	0

ROC Curve:



Test of Significance

- Given two models:
  - Model M1: accuracy = 85%, tested on 30 instances
  - Model M2: accuracy = 75%, tested on 5000 instances
- Can we say M1 is better than M2?
  - How much confidence can we place on accuracy of M1 and M2?
  - Can the difference in performance measure be explained as a result of random fluctuations in the test set?

Summary

Supervised learning can be used for Data Mining tasks

- If the problem requires the **classification** of data
- If the problem can be described by a function.
- If some **ground truth** information is **available**
- If an extraction of rules is not required

A main advantage of machine learning is that expert knowledge is not strictly required.

MLP provides a generic (one-size-fits-all) algorithm to classification.

Note that an MLP is hardware which is almost always simulated by software realizations.