Classification Lecture Notes
cse352

Neural Networks

Professor Anita Wasilewska

Neural Networks Classification
Introduction

- **INPUT:** classification data, i.e. it contains an classification (class) attribute
- **WE** also say that the class label is known for all data.
- **DATA** is divided, as in any classification problem, into **TRAINING and TEST** data sets

Building a Neural Networks Classifier

– **ALL DATA must be normalized,** i.e. all values of attributes in the dataset has to be changed to contain values in the interval [0,1], or [-1,1]

TWO BASIC normalization techniques:

– Max- Min normalization and

– Decimal Scaling normalization.

Data Normalization

- **Max-Min Normalization**

Performs a linear transformation on the original data.

- Given an attribute A, we denote by

***min*A, *max*A** the minimum and maximum

values of the values of the attribute A

- **Max-Min Normalization** maps a value v

of A to v' in the range

- **[*new_min*A, *new_max*A]**

as follows.

Data Normalization

Max- Min normalization formula is as follows:

$$v' = \frac{v - \min A}{\max A - \min A}(new\_\max A - new\_\min A) + new\_\min A$$

Example: we want to normalize data to range of the interval [-1,1]

We put: **new_max A= 1**, **new_minA = -1**

In general, to normalize within interval [a,b] we put:

new_max A= b, **new_minA = a**

Example of Max-Min Normalization

Max- Min normalization formula

$$v' = \frac{v - \min A}{\max A - \min A}(new\_\max A - new\_\min A) + new\_\min A$$

Example: **We want to normalize data to range of the interval [0,1].**
We put: new_max A= 1, new_minA =0

Say, max A was 100 and min A was 20 (That means maximum and minimum values for the attribute A)

Now, if v = 40 (If for this particular pattern , attribute value is 40), v' will be calculated as
v' = (40-20) x (1-0) / (100-20) + 0
=> v' = 20 x 1/80
=> v' = 0.4

♪

Decimal Scaling Normalization

Normalization by decimal scaling normalizes by moving the decimal point of values of attribute A

A value v of A is normalized to v' by computing

$$\boxed{v' = \frac{v}{10^j}}$$

where j is the smallest integer such that max|v' |<1.

Example :

A – values range from -986 to 917   Max |v| = 986

v = -986 normalize to v' = -986/1000 = -0.986

Neural Network

- Neural Network is a set of connected INPUT/ OUTPUT UNITS, where each connection has a WEIGHT associated with it

- Neural Network learning is also called CONNECTIONIST **learning** due to the connections between units

- **Neural Network is always fully connected**
- It is a case of SUPERVISED, INDUCTIVE or CLASSIFICATION learning

Neural Network Learning

- **Neural Network learns** by adjusting the weights so as to be able to **correctly classify** the training data and hence, after testing phase, to classify unknown data

- **Neural Network** needs long time for training

- **Neural Network** has a high tolerance to noisy and incomplete data.

Classification by Backpropagation

- **Backpropagation:** a **neural network** learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- **A neural network:** a set of connected input/output units where each connection has a **weight** associated with it
- During the learning phase, the **network learns by adjusting the weights** so as to be able to predict the correct class label of the input tuples
- Also referred to as **connectionist learning** due to the connections between units

How A Multi-Layer Neural Network Works?

- The **inputs** to the network correspond to the attributes and their values for **each training** tuple
- **Inputs** are **fed simultaneously** into the **units** making up the **input layer**
- **Inputs** are then **weighted** and **fed simultaneously** to a **hidden layer**
- The number of **hidden layers** is arbitrary, although often only one or two
- The **weighted outputs** of the last hidden layer are **input** to units making up the **output layer**, which emits the network's prediction

How A Multi-Layer Neural Network Works?

- The network is **feed-forward** - it means that **none** of the **weights cycles back**

  to an **input unit** or to an **output unit** of

  a **previous layer**
- From a statistical point of view, networks perform **nonlinear regression**:
- Given enough hidden units and enough training samples, they can closely **approximate** any function

A Multilayer Feed-Forward (MLFF) Neural Network

Output vector;
Classes

Output nodes

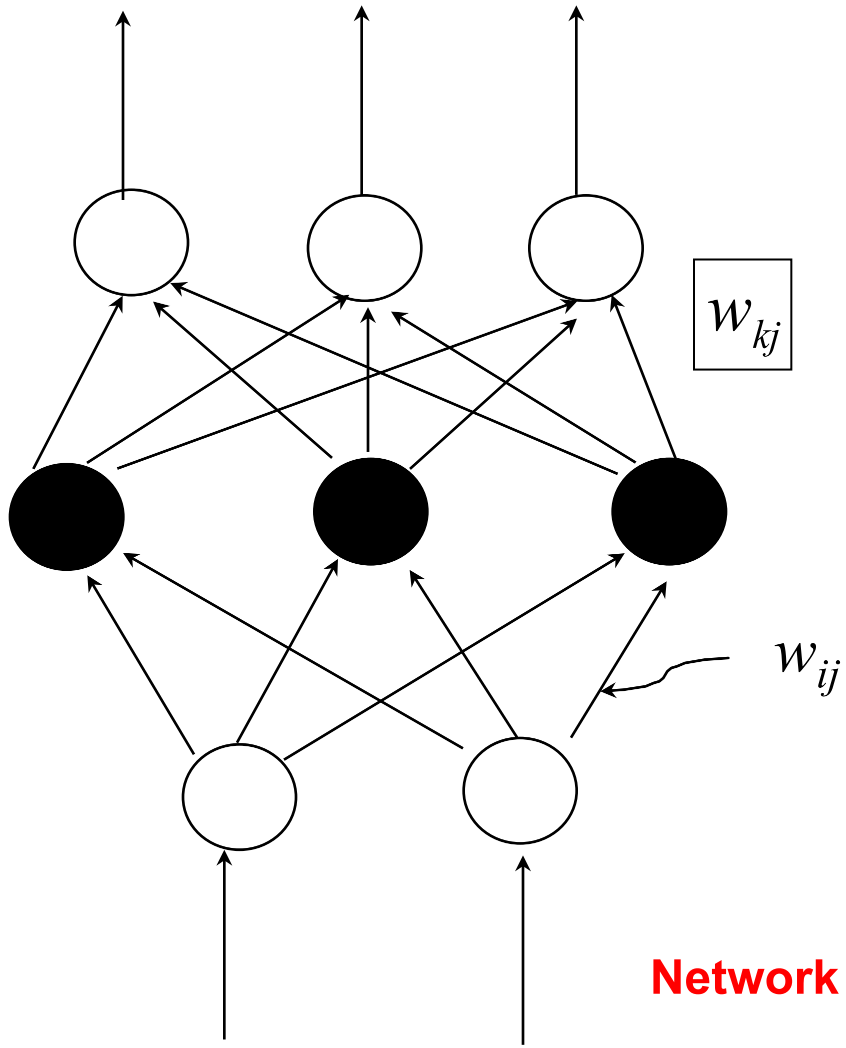
$\boxed{O_k}$

Hidden nodes

$\boxed{O_j}$

Input nodes

Input vector;
Record: $\mathbf{x}_i$



- weights

Network is fully connected

A Multilayer Feed-Forward (MLFF) Neural Network

- The units in the **hidden layers** and **output layer**
  are sometimes referred to as **neurones**
  due to their symbolic biological basis
  or just as **output units**
- A multilayer neural network shown on the previous slide has **two layers**
- The **input layer** is not counted because it serves only to **pass** the input values to **next layer**

- Therefore, we say that it is **a two-layer** neural network

A Multilayer Feed-Forward (MLFF) Neural Network

- A network containing two hidden layers is called a three-layer neural network, and so on
- The network is **feed-forward** - it means that **none** of the **weights cycles back** to an input unit or to an output unit of a **previous layer**

MLFF Neural Network

Output vector;
3 classes here

Output nodes

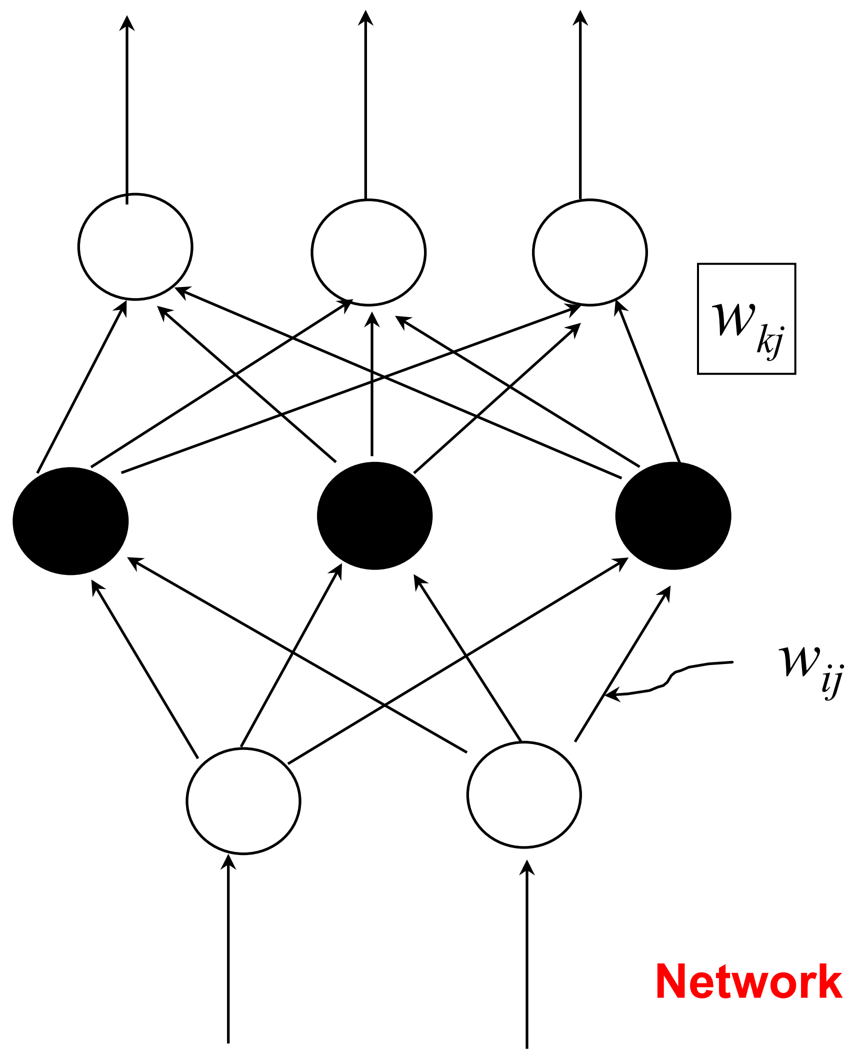
$\boxed{O_k}$

Hidden nodes

$\boxed{O_j}$

Input nodes

Input vector;
Record: x_i
2 attributes here



- weights

Network is fully connected

MLFF Network Input

- **INPUT:** records without class attribute and with normalized attributes values
- We call it an **input vector**
- **INPUT VECTOR:**

X = { x1, x2, …. xn}

where n is the number of (non class) attributes

Observe that {,} do not denote a SET symbol here!

NN network people like use that symbol for a vector;

Normal vector symbol is [x1, … xn]

MLFF Network Topology

- **Network topology:**
- We define the **network topology** by setting the following

1. number of units in the input layer

2. number of hidden layers

3. number of units in each hidden layer

4. number of units in the output layer

MLFF Network Topology

- **INPUT LAYER** – there are as many nodes as non-class attributes i.e. as the length of the input vector

- **HIDDEN LAYER** – the number of nodes in the hidden layer and the number of hidden layers **depends** on implementation

$$\boxed{O_j}$$

j=1, 2 ..#hidden nodes

MLFF Network Topology

- **OUTPUT LAYER** – corresponds to the **class attribute**
- There are as many nodes as classes (if classification has more than 2 classes)

$$\boxed{O_k}$$

k= 1, 2,.. #classes

- Network is **fully connected**, i.e. **each unit provides input** to **each unit** in the next forward layer

MLFF Network Topology

- Once a **network has been trained**
- and its predictive accuracy is **unacceptable**
- **repeat the training** process with a **different network topology**
- or a **different set** of **initial weights**

Classification by Backpropagation

- **Backpropagation** is a neural network learning algorithm
- **It learns** by iteratively processing a set of **training data**
- comparing the network's prediction for each record with the actual known **target value**
- **The target** value may be the known class label of the training tuple
- or a **continuous value** for prediction

Classification by Backpropagation

- For **each training sample**, the **weights** are first set **random** then they are **modified** as to minimize the **mean squared error** between the **network's classification** (prediction) and **actual classification**

- These **weights modifications** are propagated in "**backwards**" direction, that is,
- from the **output layer**, through **each hidden layer** down to the **first hidden layer**
- Hence the name **backpropagation**

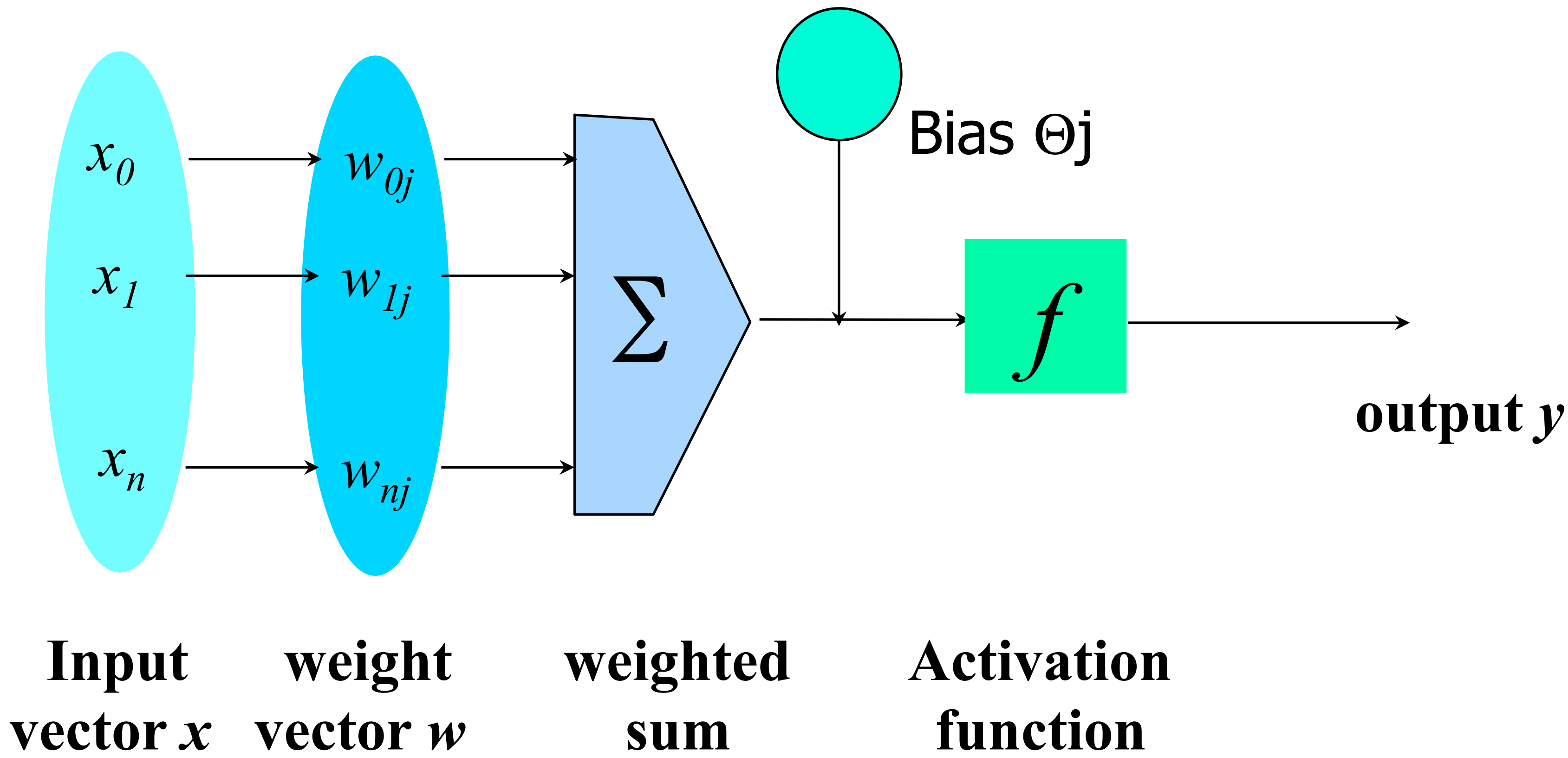
Steps in Backpropagation Algorithm

- **STEP ONE:** initialize the **weights and biases**
- **The weights** in the network are initialized to small random numbers ranging for example from -1.0 to 1.0, or -0.5 to 0.5.
- **Each unit** has a **BIAS** associated with it (see next slide).
- **The biases** are similarly initialized to small random numbers.
- **STEP TWO:** **feed** the training sample

Steps in Backpropagation Algorithm

- **STEP THREE:** propagate the inputs forward (by applying activation function)
- We compute the net input and output of each unit in the hidden and output layers
- **STEP FOUR: backpropagate** the error
- **STEP FIVE:** update weights and biases to reflect the propagated errors
- **STEP SIX: repeat** and apply **terminating conditions**

A Neuron; a Hidden, or Output Unit j



- The inputs to unit **j are outputs** from the **previous layer**. These are **multiplied** by their corresponding **weights** in order to form a **weighted sum**, which is **added** to the **bias** associated with **unit j**
- A **nonlinear activation function f** is applied to the net input

Step Three: propagate the inputs forward

- For **unit j** in the **input layer**, its **output** is equal to its **input**, that is,

$$\boxed{O_j = I_j}$$

The net input to each unit in the hidden and output layers is computed as follows.

•Given a **unit j** in a **hidden** or **output** layer, the **net input** is

$$\boxed{I_j = \sum_i w_{ij} O_i + \theta_j}$$

where wij is the weight of the connection from unit i in the previous layer to unit j; Oi is the output of unit i from the previous layer;

$\boxed{\theta_j}$ is **the bias** of the unit

Step 3: propagate the inputs forward

- Each **unit in the hidden and output layers** takes **its net input** and then applies an **activation function**.
- **The function** symbolizes the **activation** of the **neuron** represented by the unit
- It is also called **a logistic, sigmoid, or squashing function**.
- Given a **net input Ij** to unit j, then

$$Oj = f(Ij)$$

the **output** of unit j, is computed as

$$\boxed{O_j = \frac{1}{1+e^{-I_j}}}$$

Step 4: Back propagate the error

- When reaching the **output layer**, the error is computed and propagated backwards
- For a unit k in the output layer the **error** is computed by a formula:

$$\boxed{Err_k = O_k(1-O_k)(T_k - O_k)}$$

Where Ok is the **actual output** of unit k computed by **activation function**

$$\boxed{O_k = \frac{1}{1+e^{-I_k}}}$$

Tk is the **TRUE output** based of known class label of training sample

Observe: Ok(1-Ok) is a derivative (rate of change) of activation function

Step 4: Backpropagate the error

- **The error** is **propagated backwards** by updating weights and biases to reflect the error of the **network classification**
- For a unit j in the **hidden layer** the **error** is computed by a formula:

$$\boxed{Err_j = O_j(1 - O_j)\sum_k Err_k w_{jk}}$$

where **wjk** is the weight of the connection from unit j to unit k in the **next higher layer**, and Errk is the **error** of unit k

Step 5: Update weights and biases

- **Weights** are **updated** by the following equations, where *l* is a constant between 0.0 and 1.0 reflecting
- **the learning rate -** this learning rate is fixed for implementation

$$\boxed{\Delta w_{ij} = (l) Err_j O_i}$$

$$\boxed{w_{ij} = w_{ij} + \Delta w_{ij}}$$

The rule of thumb is to set the learning rate to
l = 1/k where k is the number of iterations through the training set so far

Backpropagation Formulas

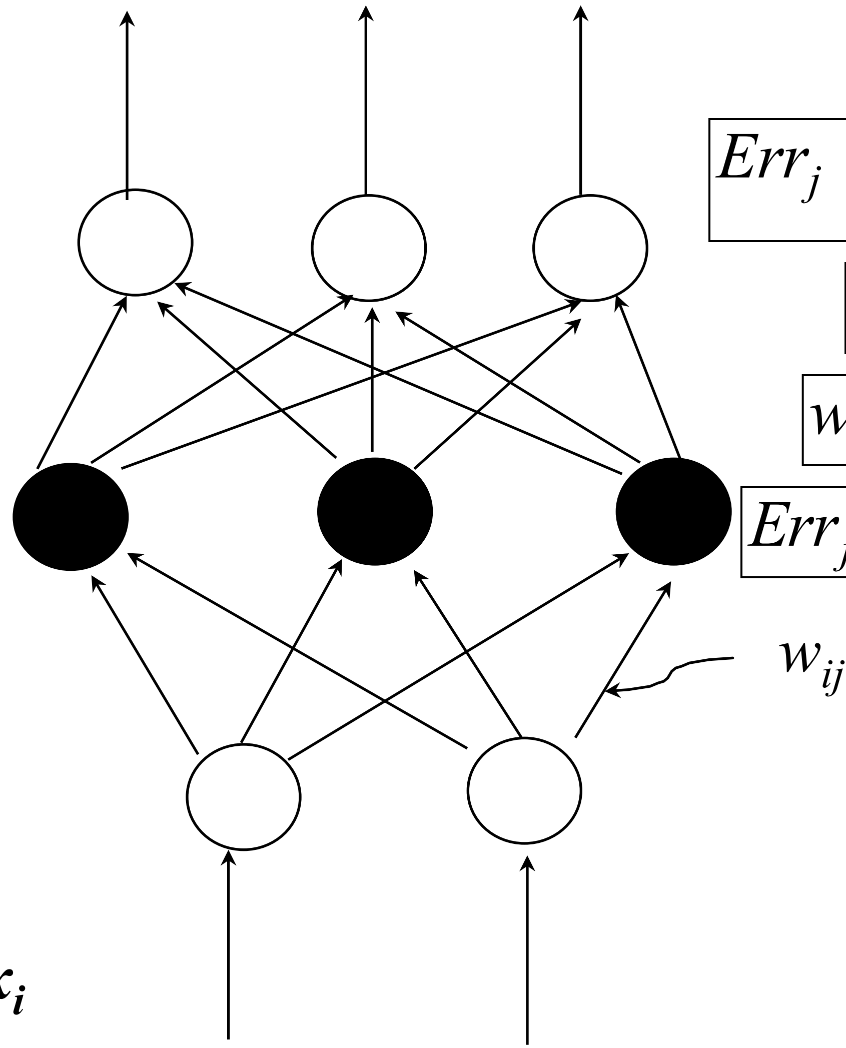
Output vector

Output nodes

Hidden nodes

Input nodes

Input vector: x_i



$$Err_j = O_j(1 - O_j)\sum_k Err_k w_{jk}$$

$$\theta_j = \theta_j + (l)Err_j$$

$$w_{ij} = w_{ij} + (l)Err_j O_i$$

$$Err_j = O_j(1 - O_j)(T_j - O_j)$$

$$O_j = \frac{1}{1 + e^{-I_j}}$$

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

Step 5: Update weights and biase

Learning Rate

- The **learning rate** helps avoid getting stuck at
- local mimimum (i.e. where the weights appear to converge, but are not optimum solution)
- The **learning rate** encourages finding the global minimum
- If the **learning rate** is **too small**, then learning will occur at a very **slow pace**
- If the **learning rate** is **too large**, then **oscillation** between inadequate solutions may occur.

Step 5: Update weights and biases

Bias update

Biases are **updated** by the following equations

$$\boxed{\Delta\theta_j = (l)Err_j}$$

$$\boxed{\theta_j = \theta_j + \Delta\theta_j}$$

Where $\boxed{\Delta\theta_j}$ is the change in the bias

Weights and Biases Updates

- **Case updating:** we are updating weights and biases after the presentation of **each sample**

Epoch: One iteration through the training set

- Epoch updating:
- The weight and bias increments are **accumulated** in variables and the weights and biases are updated after **all of the samples** of the training set have been presented

- **Case updating is more accurate**

Terminating Conditions

- Training **stops** when

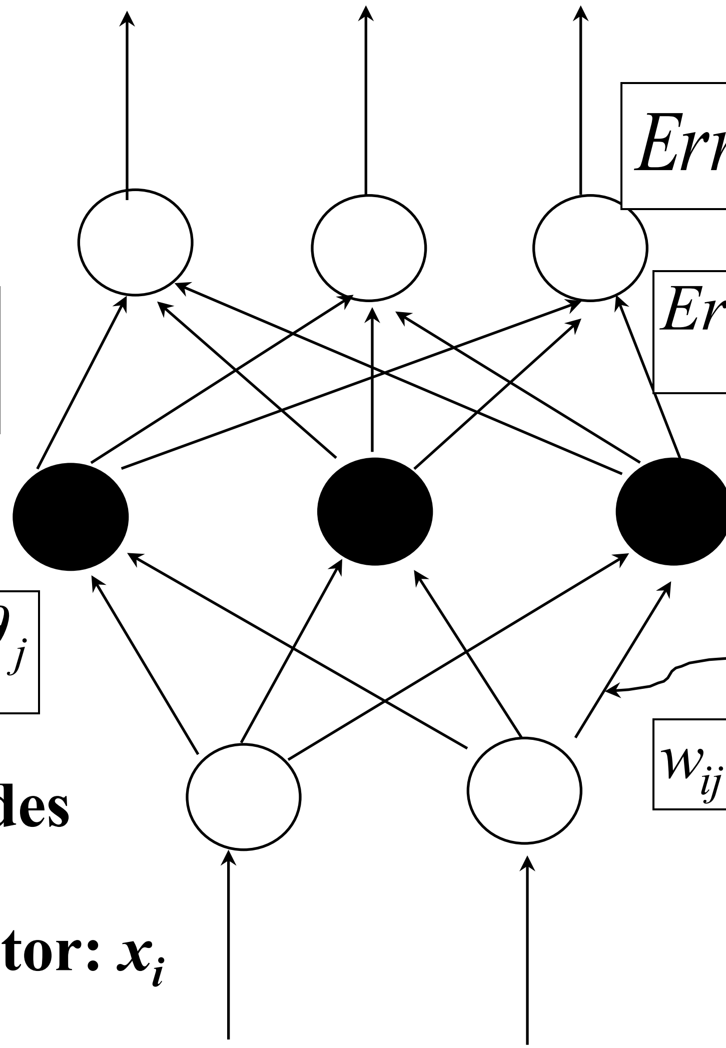
  - All $\boxed{\Delta w_{ij}}$ in the previous epoch are below some threshold, or

•The percentage of samples **misclassified** in the previous epoch is below some threshold, or

• a pre- specified number of epochs has **expired**

• In practice, several hundreds of thousands of epochs may be required before the **weights will converge**

Backpropagation Formulas



Output vector

$$Err_k = O_k(1 - O_k)(T_k - O_k)$$

Output nodes

$$O_j = \frac{1}{1 + e^{-I_j}}$$

$$Err_j = O_j(1 - O_j)\sum_k Err_k w_{jk}$$

Hidden nodes

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

w_{ij}

$$\theta_j = \theta_j + (l) Err_j$$

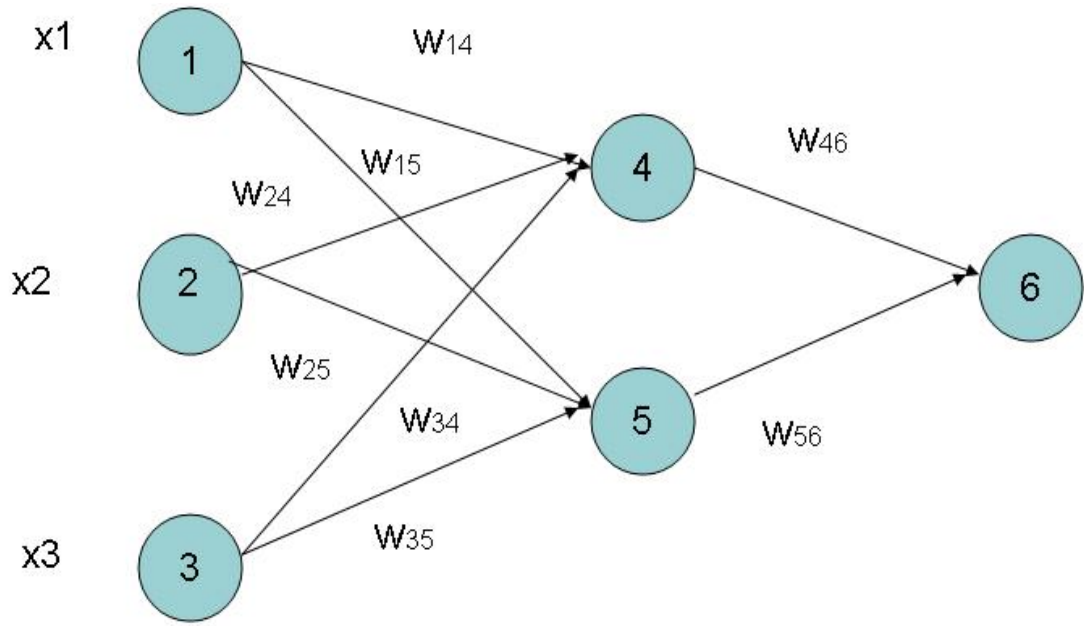
$$w_{ij} = w_{ij} + (l) Err_j O_i$$

Input nodes

Input vector: x_i

Example of Back Propagation

Input = 3, Hidden Neuron = 2 Output =1



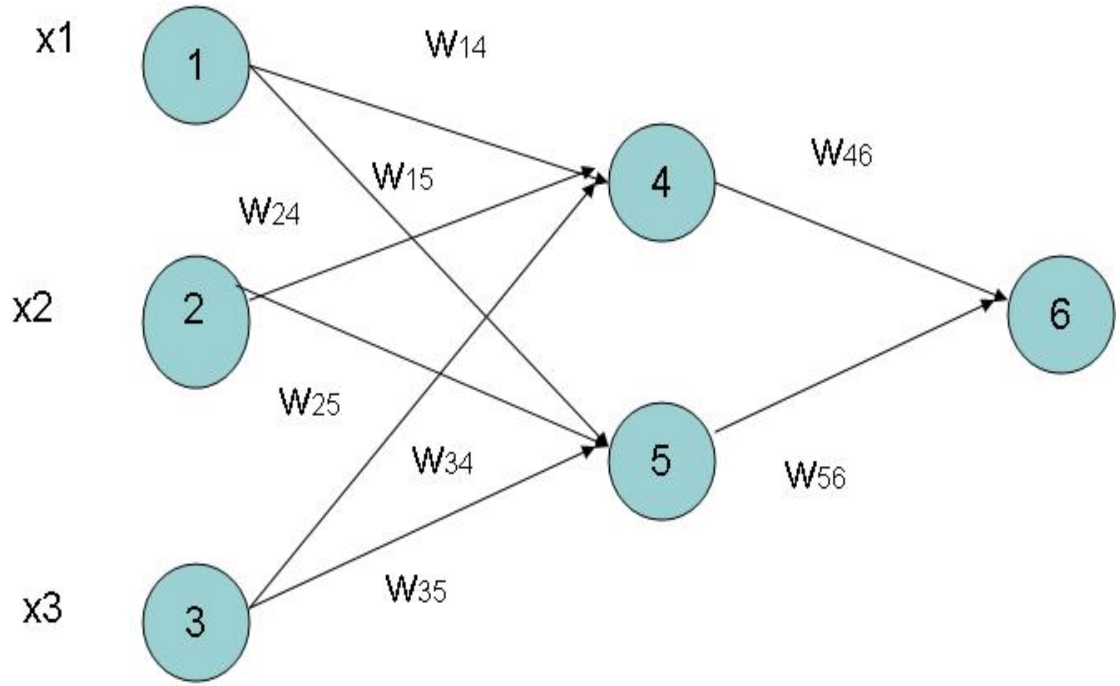
Initialize weights :

Random Numbers from -1.0 to 1.0

Initial Input and weight

x1	x2	x3	W_{14}	W_{15}	W_{24}	W_{25}	W_{34}	W_{35}	W_{46}	W_{56}
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2

Example of Back Propagation



- **Bias added to Hidden and output nodes**
- **Initialize Bias**
- **Bias: Random Values from**
- **-1.0 to 1.0**

- **Bias (Random)**

θ_4	θ_5	θ_6
-0.4	0.2	0.1

Net Input and Output Calculation

Unit j	Net Input I_j	Output O_j
4	**0.2 + 0 - 0.5 -0.4 = -0.7**	$O_j = \frac{1}{1+e^{0.7}}$ = 0.332
5	**-0.3 + 0 + 0.2 + 0.2 =0.1**	$O_j = \frac{1}{1+e^{-0.1}}$ = 0.525
6	**(-0.3)0.332-(0.2)(0.525)+0.1= -0.105**	$O_j = \frac{1}{1+e^{0.105}}$ = 0.475

Calculation of Error at Each Node

Unit $_j$	Error $_j$
6	**0.475(1-0.475)(1-0.475) =0.1311** **We assume T_6 = 1**
5	**0.525 x (1- 0.525)x 0.1311x (-0.2) = 0.0065**
4	**0.332 x (1-0.332) x 0.1311 x (-0.3) = -0.0087**

Calculation of weights and Bias Updating

Learning Rate l =0.9

Weight	New Values
w_{46}	**-0.3 + 0.9(0.1311)(0.332) = -0.261**
w_{56}	**-0.2 + (0.9)(0.1311)(0.525) = -0.138**
w_{14}	**0.2 + 0.9(-0.0087)(1) = 0.192**
w_{15}	**-0.3 + (0.9)(-0.0065)(1) = -0.306**
……..similarly	**………similarly**
θ_6	**0.1 +(0.9)(0.1311)=0.218**
……..similarly	**………similarly**

Network Pruning and Rule Extraction

- Network pruning
  - Fully connected network is hard to articulate
  - N input nodes, h hidden nodes and m output nodes lead to $h(m+N)$ weights
  - Pruning: Remove some of the links without affecting classification accuracy of the network

Some Facts to be Remembered

- NNs perform well, generally better with larger number of hidden units
- More hidden units generally produce lower error
- Determining network topology is difficult
- Choosing single learning rate impossible
- Difficult to reduce training time by altering the network topology or learning parameters
- NN with Subsets (see next slides) learning often produce better results

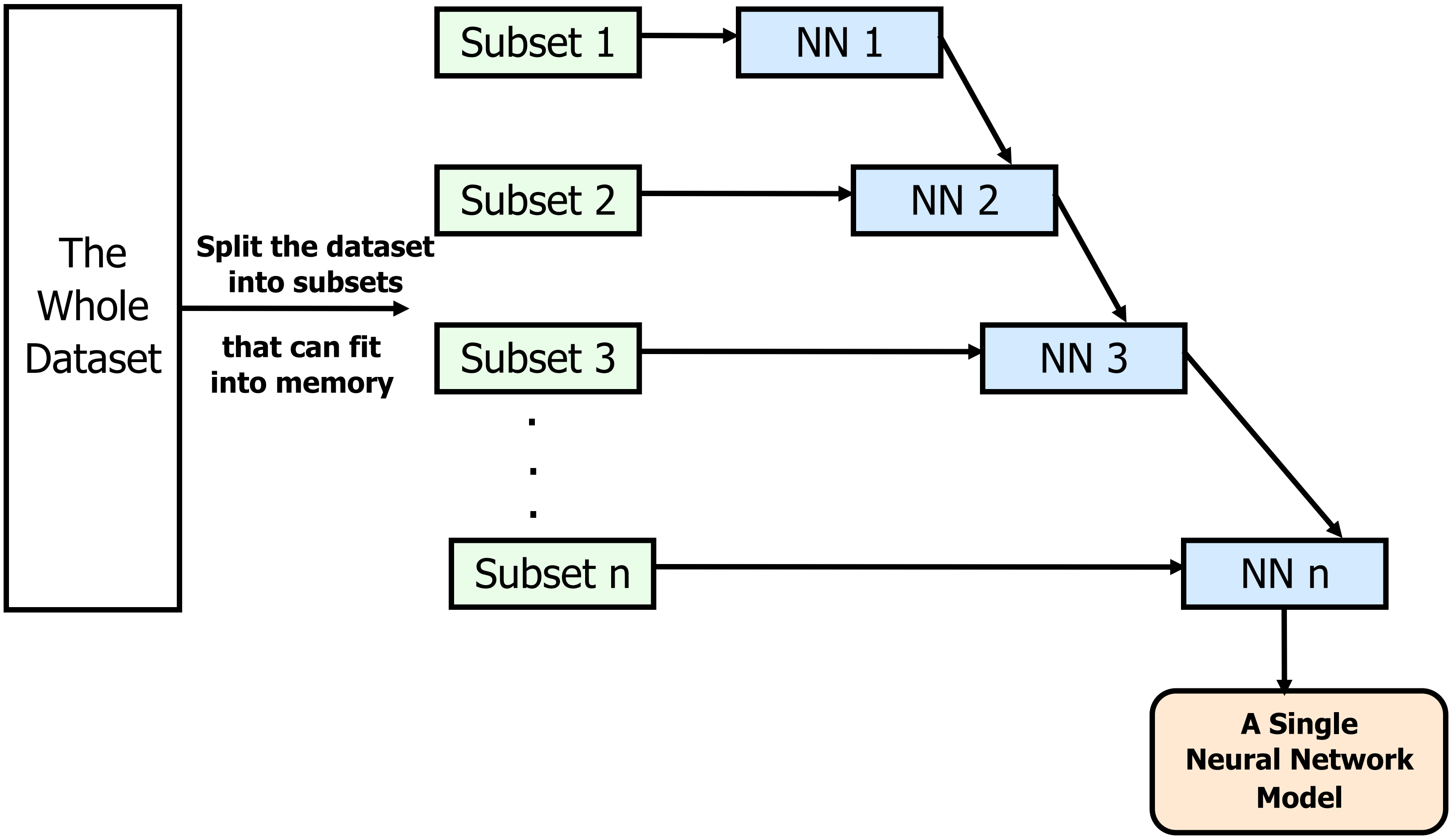
Some Facts to be Remembered

- **Rule extraction from networks:** network pruning
  - **Simplify** the network structure by removing weighted links that have the **least effect** on the trained network
  - **Then perform** link, unit, or activation value clustering
  - **The set** of input and activation values are studied to **derive rules** describing the **relationship** between the input and hidden unit layers
- **Sensitivity analysis:** assess the impact that a given input variable has on a network output.
- **The knowledge gained** from this analysis can be represented in **rules**

Advanced Features of Neural Network
(may be covered by students presentations)

- Training with Subsets
- Modular Neural Network
- Evolution of Neural Network

Training with subsets



Training with subsets

•**Break** the data into subsets, that can fit in memory

•**Train** one neural network on a series of the subsets

•**The result** is a single neural network model

•In this way, we attempt to overcome the difficulty making use of all the available data, without leaving anything

Training with Subsets

- Select subsets of data
- Build a **new classifier** on a subset
- **Aggregate** with previous **classifiers**
- Compare **error** after adding a classifier
- **Repeat** as long as error decreases

Modular Neural Network

- Modular Neural Network
  - Made up of a **combination** of several neural networks

  The idea is to reduce the load for each neural network as opposed to trying to solve the problem on a single neural network.

Evolving Network Architectures

- **Small networks without a hidden layer** can't solve problems such as XOR, that are not linearly separable.

  Large networks can easily overfit a problem to match the training data, limiting their ability to generalize a problem set

Constructive vs Destructive Algorithm

- **Constructive** algorithms take a minimal network and build up **new layers nodes** and connections **during training**

- **Destructive** algorithms take a maximal network and prunes **unnecessary layers** nodes and connections **during training**

Faster Convergence

- **Back propagation** requires many epochs to **converge**

An epoch is one presentation of all the training examples in the dataset

- Some ideas to overcome this are:
  - ***Stochastic learning:***
  - **updates weights** after each example,
    instead of updating them after **one epoch**

Faster Convergence

- ***Momentum:***
- This **optimization** is due to the fact that it speeds up the learning when the **weight** are moving in a single direction continuously by increasing the size of steps

- **The closer** this value is to one,

the more each weight change will not only include the current error,

but also the weight change from **previous examples**

(which often leads to faster convergence)

Discriminative Classifiers

- Advantages
  - prediction accuracy is generally high
    - As compared to Bayesian methods – in general
  - robust, works when training examples contain errors
  - fast evaluation of the learned target function
    - Bayesian networks are normally slow
- Criticism
  - long training time
  - difficult to understand the learned function (weights)
    - Bayesian networks can be used easily for pattern discovery
  - not easy to incorporate domain knowledge
    - Easy in the form of priors on the data or distributions

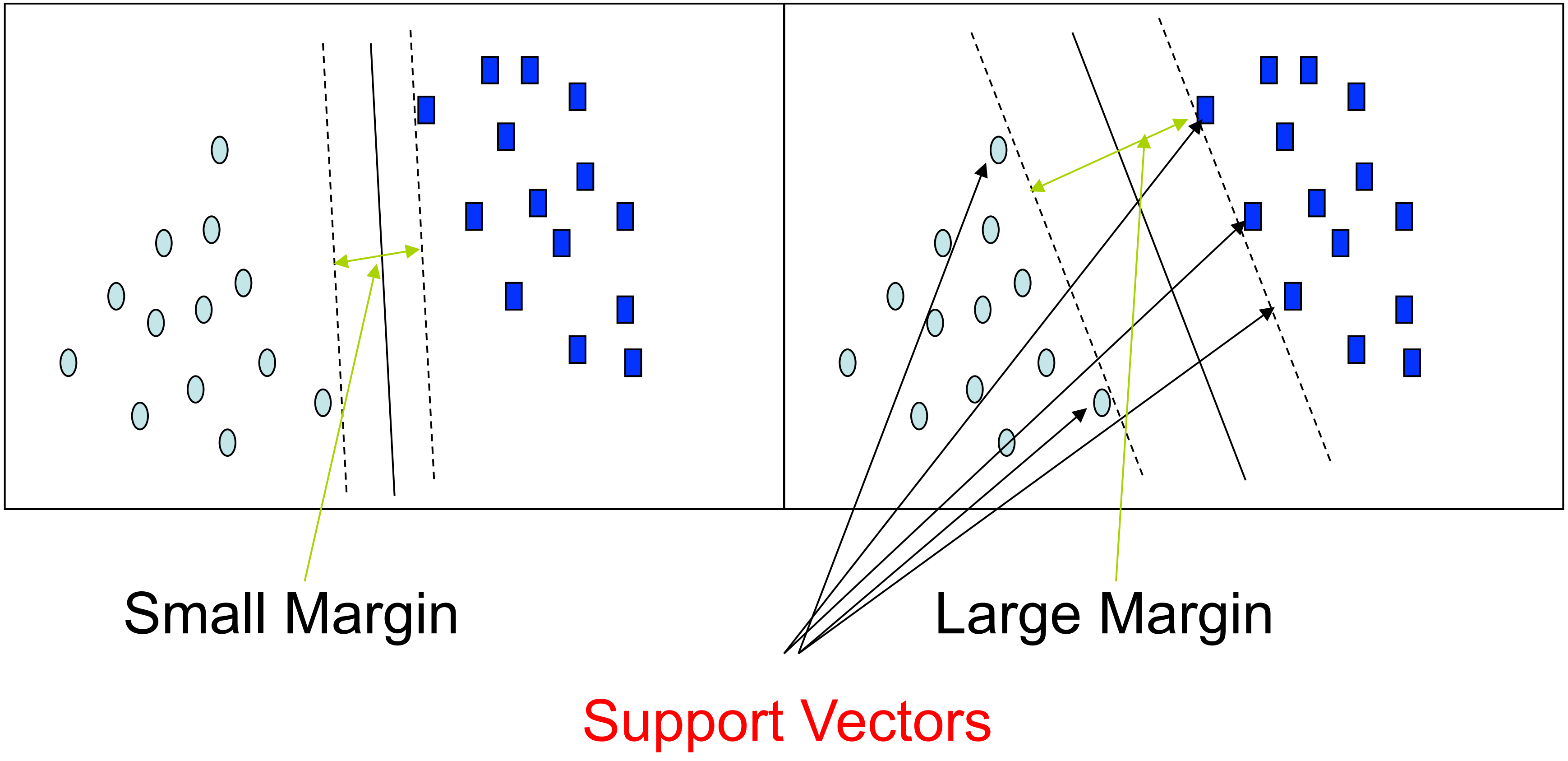
SVM—Support Vector Machines

- A new classification method for both linear and nonlinear data
- It uses a nonlinear mapping to transform the original training data into a higher dimension
- With the new dimension, it **searches** for the linear optimal separating hyper plane (i.e., "decision boundary")
- With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be **separated** by a hyper plane
- SVM finds this hyper plane using **support vectors** ("essential" training tuples) and **margins** (defined by the support vectors)

SVM—History and Applications

- **Vapnik** and colleagues (1992)—groundwork from Vapnik & Chervonenkis' statistical learning theory in 1960s
- **Features:** training can be slow but accuracy is high owing to their ability to model complex nonlinear decision boundaries (margin maximization)
- Used both for classification and prediction
- **Applications:**
  - handwritten digit recognition, object recognition, speaker identification, benchmarking time-series prediction tests

SVM—General Philosophy



Why Is SVM Effective on High Dimensional Data?

- The complexity of trained classifier is characterized by the # of support vectors rather than the dimensionality of the data
- The support vectors are the essential or critical training examples — they lie closest to the decision boundary (MMH)
- If all other training examples are removed and the training is repeated, the same separating hyperplane would be found
- The number of support vectors found can be used to compute an (upper) bound on the expected error rate of the SVM classifier, which is independent of the data dimensionality
- Thus, an SVM with a small number of support vectors can have good generalization, even when the dimensionality of the data is high

SVM vs. Neural Network

- SVM
  - Relatively new concept
  - Deterministic algorithm
  - Nice Generalization properties
  - Hard to learn – learned in batch mode using quadratic programming techniques
  - Using kernels can learn very complex functions
- Neural Network
  - Relatively old
  - Nondeterministic algorithm
  - Generalizes well but doesn't have strong mathematical foundation
  - Can easily be learned in incremental fashion
  - To learn complex functions—use multilayer perceptron (not that trivial)

SVM Related Links

- SVM Website
  - http://www.kernel-machines.org/
- Representative implementations
  - LIBSVM: an efficient implementation of SVM, multi-class classifications, nu-SVM, one-class SVM, including also various interfaces with java, python, etc.
  - SVM-light: simpler but performance is not better than LIBSVM, support only binary classification and only C language
  - SVM-torch: another recent implementation also written in C.

SVM—Introduction Literature

- "Statistical Learning Theory" by Vapnik: extremely hard to understand, containing many errors too.
- C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Knowledge Discovery and Data Mining*, 2(2), 1998.
  - Better than the Vapnik's book, but still written too hard for introduction, and the examples are so not-intuitive
- The book "An Introduction to Support Vector Machines" by N. Cristianini and J. Shawe-Taylor
  - Also written hard for introduction, but the explanation about the mercer's theorem is better than above literatures
- The neural network book by Haykins
  - Contains one nice chapter of SVM introduction

Lazy vs. Eager Learning

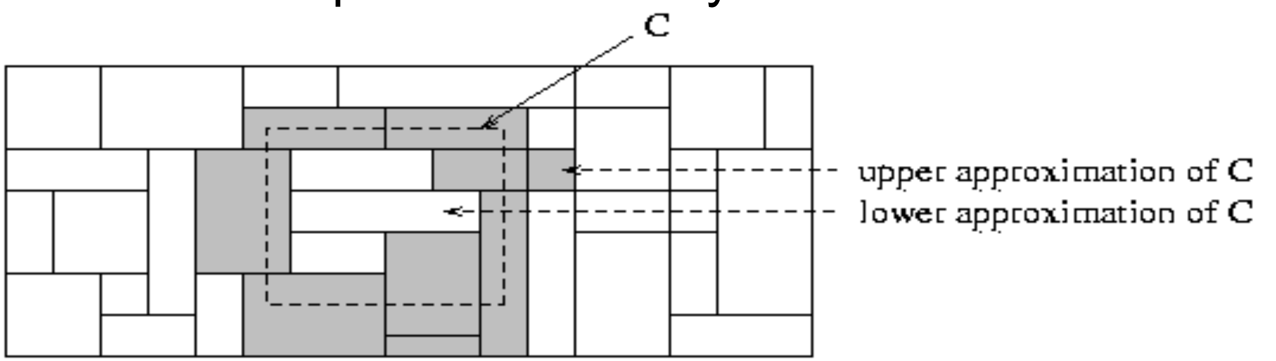
- <u>Lazy vs. eager learning</u>
  - Lazy learning (e.g., instance-based learning): Simply stores training data (or only minor processing) and waits until it is given a test tuple
  - Eager learning (the above discussed methods): Given a set of training set, constructs a classification model before receiving new (e.g., test) data to classify
- Lazy: less time in training but more time in predicting
- Accuracy
  - Lazy method effectively uses a richer hypothesis space since it uses many local linear functions to form its implicit global approximation to the target function
  - Eager: must commit to a single hypothesis that covers the entire instance space

Lazy Learner: Instance-Based Methods

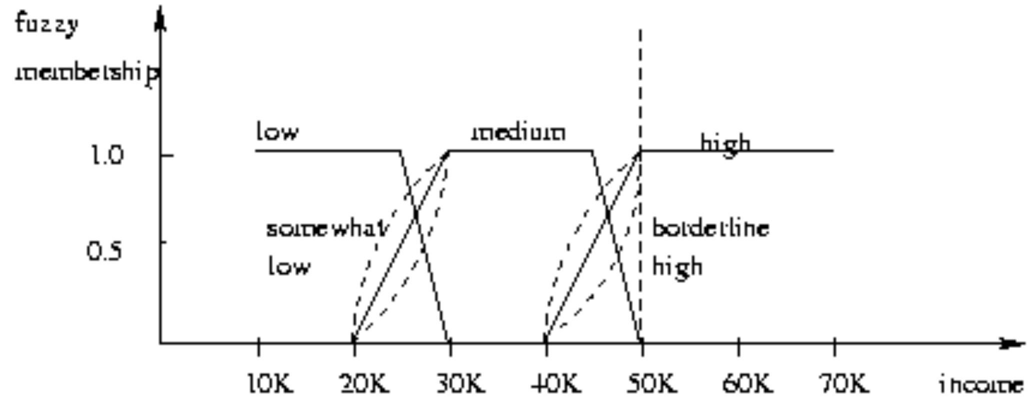
- Instance-based learning:
  - Store training examples and delay the processing ("lazy evaluation") until a new instance must be classified
- Typical approaches
  - *k*-nearest neighbor approach
    - Instances represented as points in a Euclidean space.
  - Locally weighted regression
    - Constructs local approximation
  - Case-based reasoning
    - Uses symbolic representations and knowledge-based inference

Rough Set Approach

- Rough sets are used to **approximately or “roughly” define equivalent classes**
- A rough set for a given class C is approximated by two sets: a lower approximation (certain to be in C) and an upper approximation (cannot be described as not belonging to C)
- Finding the minimal subsets (**reducts**) of attributes for feature reduction is NP-hard but a **discernibility matrix** (which stores the differences between attribute values for each pair of data tuples) is used to reduce the computation intensity



Fuzzy Set Approaches



- Fuzzy logic uses truth values between 0.0 and 1.0 to represent the degree of membership (such as using fuzzy membership graph)
- Attribute values are converted to fuzzy values
  - e.g., income is mapped into the discrete categories {low, medium, high} with fuzzy values calculated
- For a given new sample, more than one fuzzy value may apply
- Each applicable rule contributes a vote for membership in the categories
- Typically, the truth values for each predicted category are summed, and these sums are combined