



COMP3420: Advanced Databases and Data Mining

Classification and prediction:
Rule-based classification and
artificial neural networks

Lecture outline

- IF-THEN rule classification
 - Rule extraction from a decision tree
 - Rule extraction from training data
 - Sequential covering algorithm
- Classification: a mathematical mapping
- Linear classification
- Artificial neural networks
 - Classification through backpropagation
 - Neural network as a classifier
 - A multi-layer feed-forward neural network
 - Defining a network topology
 - Backpropagation and interpretability

Using IF-THEN rules for classification

- Represent the knowledge in the form of *IF-THEN* rules

- Rule R : IF **age="youth" AND student="yes"** THEN **buys_computer = "yes"**

rule antecedent/precondition

rule consequent

- Assessment of a rule: *coverage* and *accuracy*

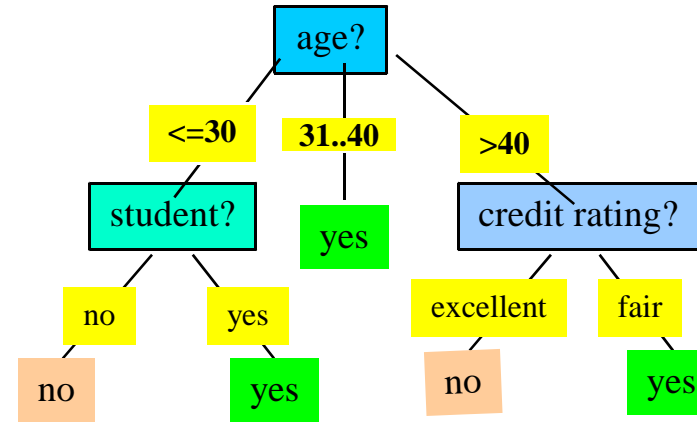
- n_{covers} = Number of tuples (records) covered by a rule R
 - $n_{correct}$ = Number of tuples correctly classified by a rule R
 - $coverage(R) = n_{covers} / |D|$ and $accuracy(R) = n_{correct} / n_{covers}$
(with D the training data set and $| \cdot |$ = number of records)

Using IF-THEN rules for classification

- If more than one rule is triggered, need *conflict resolution*
 - Size ordering: assign the highest priority to the triggering rule that has the “toughest” requirement (i.e., with the *most attribute tests*)
 - Rule ordering: prioritize the rule beforehand
 - Class-based ordering: decreasing order of *prevalence* or *misclassification cost per class*
 - Rule-based ordering (*decision list*): rules are organised into one long priority list, according to some measure of rule quality or by experts
- When there is no rule satisfied by a data record, how can we determine the class label
 - A fallback or *default rule*: may be the majority class of tuples that were not covered by any rule

Rule extraction from a decision tree

- Rules are easier to understand than large trees
- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction: the leaf holds the class prediction
- Rules are *mutually exclusive* and *exhaustive*



- Example: Rule extraction from our *buys_computer* decision tree

IF *age* <= 30 AND *student* = *no*

THEN *buys_computer* = *no*

IF *age* <= 30 AND *student* = *yes*

THEN *buys_computer* = *yes*

IF *age* = 31..40

THEN *buys_computer* = *yes*

IF *age* > 40 AND *credit_rating* = *excellent*

THEN *buys_computer* = *no*

IF *age* > 40 AND *credit_rating* = *fair*

THEN *buys_computer* = *yes*

Rule extraction from training data

- Sequential covering algorithm: Extracts rules directly from training data
 - Typical sequential covering algorithms: *FOIL*, *AQ*, *CN2*, *RIPPER*
- Rules are learned *sequentially*, each for a given class C_i will cover many tuples of C_i but none (or few) of the tuples of other classes
 - Rules are learned one at a time
 - Each time a rule is learned, the tuples covered by the rules are removed
 - The process repeats on the remaining tuples unless *termination condition*, for example, when no more training examples or when the quality of a rule returned is below a user-specified threshold
- Decision-tree induction: learning a set of rules *simultaneously*

Sequential covering algorithm

Algorithm: Sequential covering. Learn a set of IF-THEN rules for classification

Input:

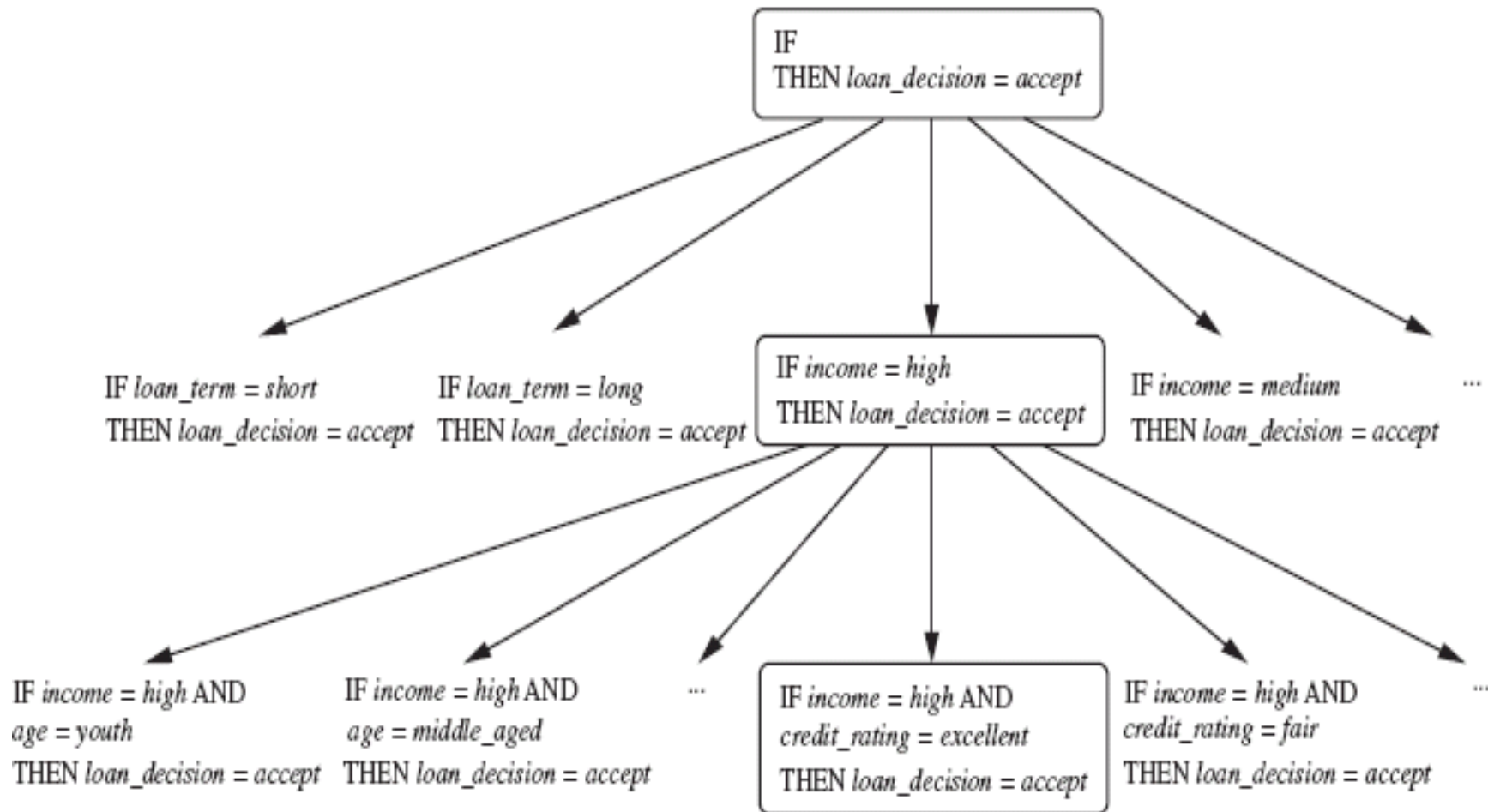
- D , a data set class-labeled tuples;
- $Att\text{-}vals$, the set of all attributes and their possible values.

Output: A set of IF-THEN rules.

Method:

- (1) $Rule_set = \{\}$; // initial set of rules learned is empty
- (2) for each class c do
- (3) repeat
- (4) $Rule = Learn_One_Rule(D, Att\text{-}vals, c)$;
- (5) remove tuples covered by $Rule$ from D ;
- (6) until terminating condition;
- (7) $Rule_set = Rule_set + Rule$ // add new rule to rule set
- (8) endfor
- (9) return $Rule_set$;

General to specific search through rule space



How to learn-one-rule?

- Start with the most general rule possible: *condition = empty*
- Adding new attributes by adopting a greedy depth-first strategy
 - Picks the one that most improves the rule quality
- Rule-quality measures: consider both *coverage* and *accuracy*

- $FOIL_{gain}$ (in *FOIL* and *RIPPER*): assesses *info_gain* by extending condition

(*pos'*, *neg'* positive and neg. tuples covered by rule *R'*)

$$FOIL_{Gain} = pos' \times \left(\log_2 \frac{pos'}{pos' + neg'} - \log_2 \frac{pos}{pos + neg} \right)$$

- It favors rules that have high accuracy and cover many positive tuples
- Rule pruning based on an independent set of test tuples

- *pos/neg* are number of positive/negative tuples covered by rule *R*

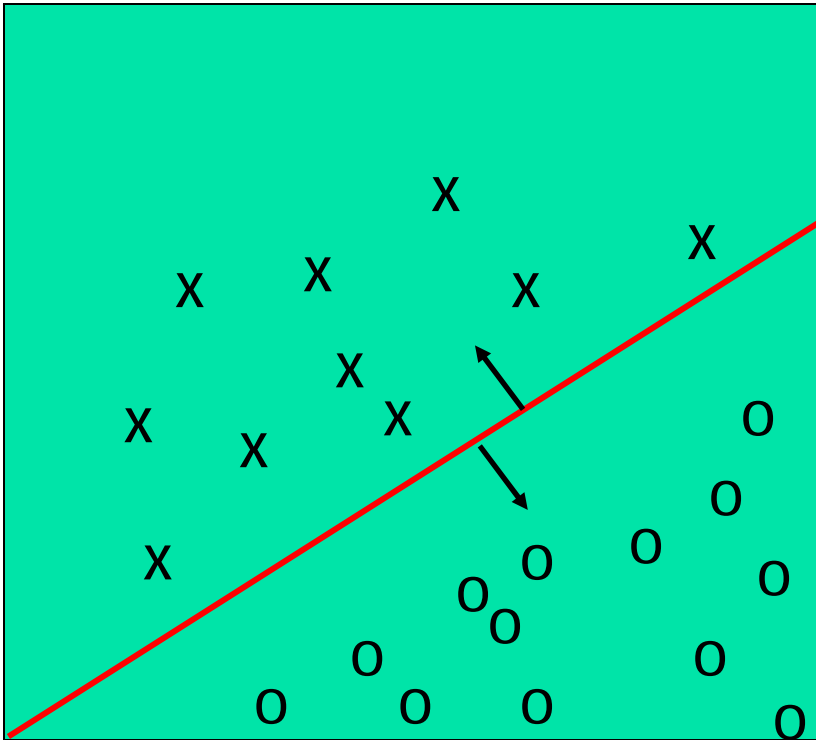
- If $FOIL_{Prune}$ is higher for the pruned version of *R*, prune *R*

$$FOIL_{Prune}(R) = \frac{pos - neg}{pos + neg}$$

Classification: A mathematical mapping

- Classification:
 - Predicts categorical class labels
- For example, personal homepage classification
 - $x_i = (x_1, x_2, x_3, \dots)$, $y_i = +1$ or -1
 - x_1 : number of occurrences of a word “homepage”
 - x_2 : number of occurrences of a word “welcome”
- Mathematically
 - $x \in X = \mathcal{R}^n$, $y \in Y = \{+1, -1\}$
 - We want a function $f: X \rightarrow Y$

Linear classification

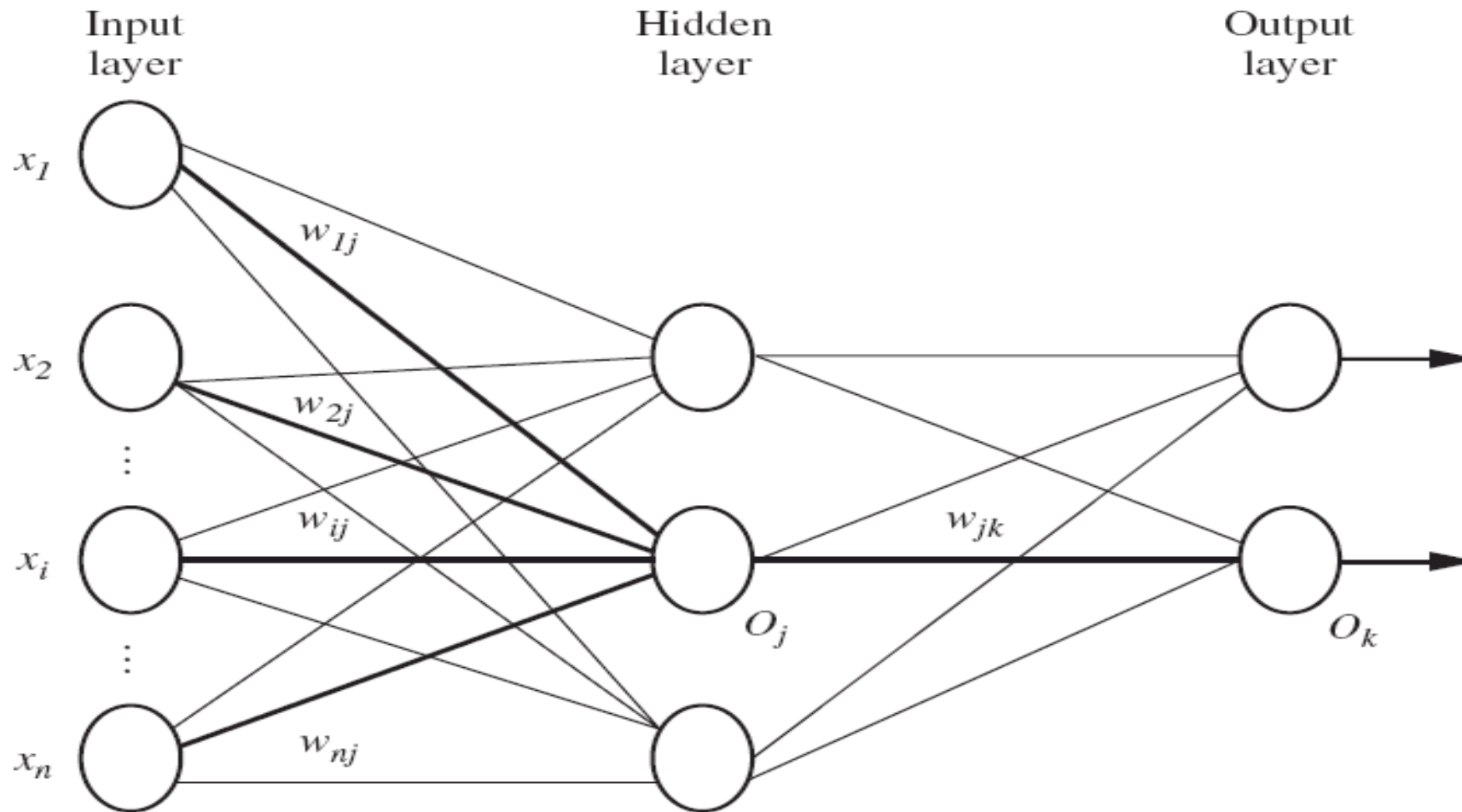


- Binary classification problem
- The data above the red line belongs to class 'x'
- The data below red line belongs to class 'o'
- Examples: Support vector machines (SVM), perceptron, probabilistic classifiers

Classification through backpropagation

- Backpropagation: An *artificial 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
 - A data structure that simulates the behaviour of neuron in a biological brain
- 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

A multi-layer feed-forward neural network



Neural network as a classifier

- Weakness

- Long training time
- Require a number of parameters typically best determined empirically, for example, the network topology or *structure*
- Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of *hidden units* in the network

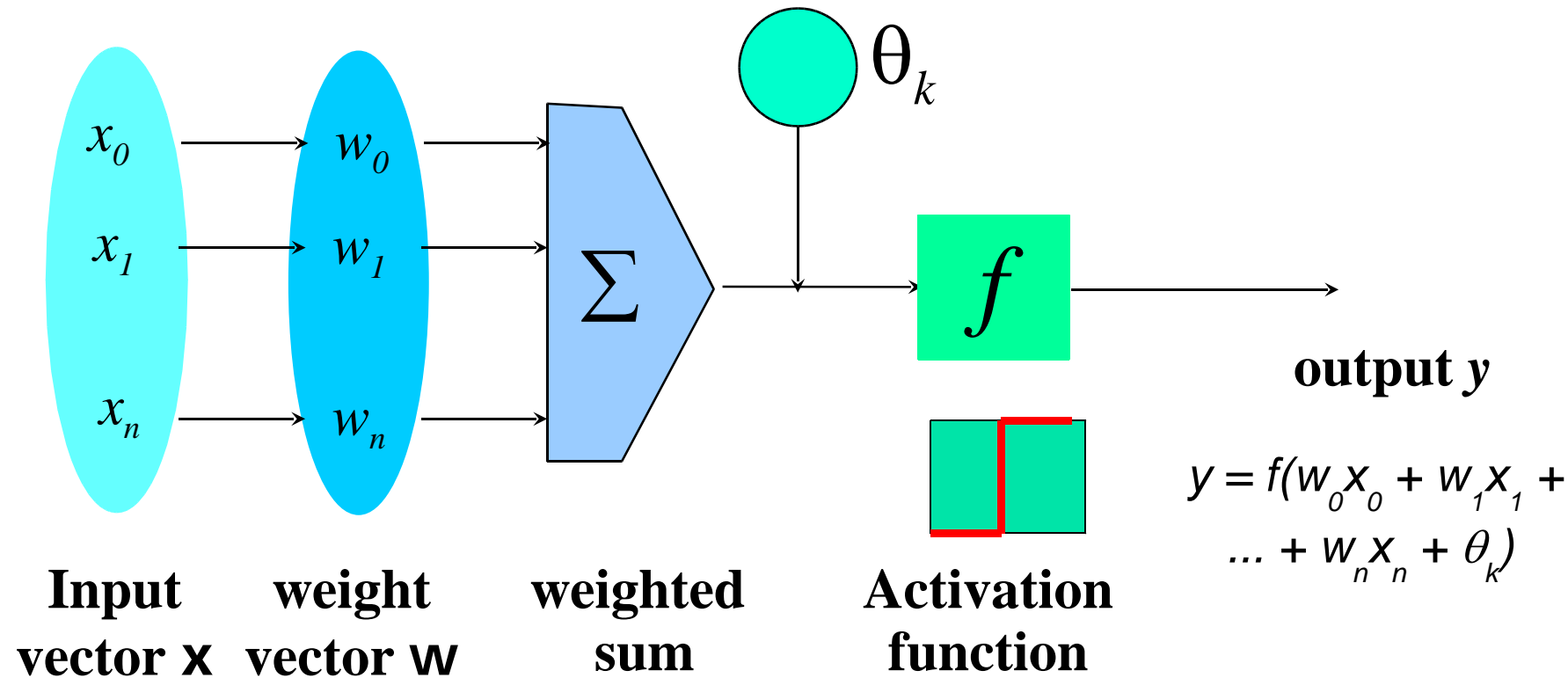
- Strength

- High tolerance to noisy data
- Ability to classify untrained patterns
- Well-suited for continuous-valued inputs and outputs
- Successful on a wide array of real-world data
- Algorithms are inherently parallel
- Techniques have recently been developed for the extraction of rules from trained neural networks

How a multi-layer neural network works

- The *inputs* to the network correspond to the attributes measured for each training tuple / record
- Inputs are fed simultaneously into the units making up the *input layer*
- They are then weighted and fed simultaneously to a *hidden layer*
- The number of hidden layers is arbitrary, although usually only one
- The weighted outputs of the last hidden layer are input to units making up the *output layer*, which emits the network's prediction
- The network is *feed-forward* in 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 neuron (= a perceptron)



- The n -dimensional input vector \mathbf{x} is mapped into variable y by means of the scalar product and a nonlinear function mapping

Defining a network topology

- First decide the *network topology*: number of units in the *input layer*, number of *hidden layers* (if > 1), number of units in *each hidden layer*, and number of units in the *output layer*
- Normalise the input values for each attribute measured in the training tuples to [0.0—1.0]
- For discrete (categorical) attributes: one *input* unit per value, each initialised to 0 (e.g. for three categories have three inputs)
- *Output*, if for classification and more than two classes, one output unit per class is used
- Once a network has been trained and its accuracy is *unacceptable*, repeat the training process with a *different network topology* or a *different set of initial weights*

Backpropagation

- Iteratively process a set of training tuples and compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to *minimise the mean squared error* between the network's prediction and the actual target value
- Modifications are made in the “*backwards*” direction: from the output layer, through each hidden layer down to the first hidden layer, hence “*backpropagation*”
- Steps
 - Initialise weights (to small random numbers) and biases in the network
 - Propagate the inputs forward (by applying activation function)
 - Backpropagate the error (by updating weights and biases)
 - Terminating condition (when error is very small, etc.)

Backpropagation and interpretability

- Efficiency of backpropagation: Each *epoch* (one iteration through the training set) takes $O(|D| * w)$, with $|D|$ tuples and w weights, but the number of epochs can be exponential to n , the number of input units, in the worst case
- 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