UNIT II

Introduction to Decision Tree Learning:

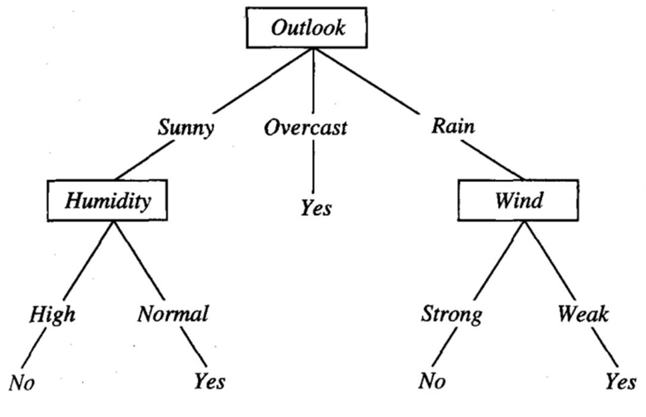
Decision tree learning is a method used to approximate discrete-valued target functions.

- The learned function is represented by a decision tree, which is a hierarchical structure consisting of nodes and branches.
- Decision trees can be learned using various algorithms, including ID3, C4.5, and CART.
- Learned trees can be re-represented as sets of if-then rules, which can improve human readability.
- Decision tree learning is a popular inductive inference algorithm that has been successfully applied to a broad range of tasks, such as medical diagnosis and credit risk assessment.
- One of the advantages of decision tree learning is that it can handle both continuous and categorical input variables.
- Decision tree learning can suffer from overfitting, which occurs when the learned tree is too complex and fits the training data too closely, resulting in poor generalization performance on new data.
- Pruning is a technique used to prevent overfitting by removing branches from a tree that do not improve its accuracy on a validation set.

Decision Tree Representation:

Decision trees are used for classification by sorting instances down the tree from the root to some leaf node, which provides the classification of the instance.

- Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute.
- An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example.
- Decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances.
- Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself to a disjunction of these conjunctions.
- "(Outlook = Sunny, Temperature = Hot, Humidity = High, Wind = Strong)" is an example of a set of attribute values for which a decision tree can provide a classification based on the tree structure.



Appropriate problems for decision tree learning:

Decision tree learning is best suited for problems where instances are represented by attribute-value pairs.

- The target function has discrete output values and decision tree methods can easily extend to learning functions with more than two possible output values.
- Decision trees naturally represent disjunctive expressions and may require disjunctive descriptions.
- Decision tree learning methods are robust to errors in classifications and attribute values of training examples.
- Decision tree methods can be used even when some training examples have unknown values, i.e., the training data may contain missing attribute values.

The basic decision tree learning algorithm:

Most algorithms for learning decision trees use a top-down, greedy search approach.

- ID3 and its successor C4.5 are commonly used algorithms for decision tree learning.
- The basic algorithm for decision tree learning is similar to ID3.
- The algorithm starts with selecting the attribute to test at the root node by evaluating each instance attribute using a statistical test.
- A descendant node is then created for each possible value of the selected attribute, and the process is repeated for each descendant node.
- This is a greedy search algorithm that does not backtrack to reconsider earlier choices.
- **ID3(Examples, Targetattribute, Attributes):**
    - Create a root node for the tree.
    - If all examples are positive, return a single-node tree with label "+".
    - If all examples are negative, return a single-node tree with label "-".
    - If no attributes remain to be considered, return a single-node tree with label equal to the most common target attribute value.
    - Select the attribute that best classifies the examples.
    - Assign the selected attribute as the decision attribute for the root node.
    - For each possible value of the selected attribute:
        - a. Create a new tree branch below the root node corresponding to the test of the selected attribute's value.
        - b. Let Examples_v be the subset of examples that have value v for the selected attribute.
        - c. If Examples_v is empty, add a leaf node below this new branch with the label equal to the most common target attribute value.
        - d. Else add a subtree below this new branch by recursively calling ID3(Examples_v, Targetattribute, Attributes - {selected attribute}).
    - Return the root node.
- The goal is to select the attribute that is most useful for classifying examples.
- A statistical property called information gain is defined to measure how well an attribute separates the training examples according to their target classification.
- ID3 uses the information gain measure to select among the candidate attributes at each step while growing the tree.
- The entropy of a collection S, containing positive and negative examples of some target concept, relative to this boolean classification is defined as:

$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

Where:
  - P+, is the proportion of positive examples in S.
  - P-, is the proportion of negative examples in S
- If the target attribute can take on c different values, then the entropy of S relative to this c-wise classification is defined as:

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i$$

Where:
  - pi is the proportion of S belonging to class i.
- the information gain, Gain(S, A) of an attribute A, relative to a collection of examples S, is defined as:

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

<u>Issues in decision tree learning:</u>

Avoid Overfitting the Data:

Decision tree algorithms aim to grow each branch of the tree just enough to perfectly classify the training examples.

- Overfitting can occur when there is noise in the data or when the number of training examples is too small.
- Overfitting is when a hypothesis h fits the training data better than h', but h' has smaller error than h over the entire distribution of instances.
- Overfitting can occur due to random errors or noise in the training examples, which can cause the algorithm to construct a more complex tree.
- Overfitting can also occur when coincidental regularities exist, leading to a risk of overfitting even with noise-free data.
- Overfitting is a significant practical difficulty for decision tree learning and many other learning methods.
- Overfitting can decrease the accuracy of learned decision trees by 10-25% on most problems, according to one experimental study of ID3.
- Overfitting in decision tree learning can be avoided through approaches that stop growing the tree earlier or post-prune the tree.
- Approaches for determining the correct final tree size include using a separate set of examples for validation, using statistical tests such as chi-square, or minimizing the encoding size of the training examples and decision tree.
- The training and validation set approach is the most common, where data is separated into two sets for training and validation.
- Reduced-error pruning is one approach to prevent overfitting using a validation set. It involves iteratively pruning decision nodes based on the accuracy over the validation set until further pruning decreases accuracy.
- Rule post pruning is a successful method for finding high accuracy hypotheses.
- The steps of rule post pruning are:
  - Infer the decision tree from the training set.
  - Convert the learned tree into an equivalent set of rules.
  - Prune (generalize) each rule by removing any preconditions that result in improving its estimated accuracy.

    - Sort the pruned rules by their estimated accuracy, and consider them in this sequence when classifying subsequent instances.
- In rule post pruning, one rule is generated for each leaf node in the tree.
- Each attribute test along the path from the root to the leaf becomes a rule antecedent (precondition), and the classification at the leaf node becomes the rule consequent (post condition).
- Each rule is pruned by removing any antecedent, whose removal does not worsen its estimated accuracy.
- Rule accuracy is estimated using a validation set of examples disjoint from the training set.
- Rule accuracy is pessimistically estimated by the observed accuracy over the training set, minus 1.96 times the estimated standard deviation.
- Converting the decision tree to rules before pruning allows distinguishing among the different contexts in which a decision node is used, improves readability, and avoids issues such as how to reorganize the tree if the root node is pruned while retaining part of the subtree below this test.

Incorporating Continuous-Valued Attributes:
ID3 algorithm is restricted to attributes that take on a discrete set of values.

- To incorporate continuous-valued decision attributes into the learned tree, new boolean attributes that partition the continuous attribute value into a discrete set of intervals can be dynamically defined.
- The threshold value for the continuous attribute can be selected by sorting the examples according to the continuous attribute and identifying adjacent examples that differ in their target classification to generate a set of candidate thresholds midway between the corresponding values of the continuous attribute.
- The value of c that maximizes information gain must always lie at a boundary between adjacent values of the continuous attribute.
- The information gain can be computed for each of the candidate attributes, and the best one can be selected to create the dynamically created boolean attribute.
- This dynamically created boolean attribute can then compete with other discrete-valued candidate attributes available for growing the decision tree.
- An extension to this approach is to split the continuous attribute into multiple intervals rather than just two intervals based on a single threshold.
- Other approaches define features by thresholding linear combinations of several continuous-valued attributes.

Alternative Measures for Selecting Attributes:
The information gain measure can favour attributes with many values over those with few values, leading to a biased selection of decision attributes.

- The attribute Date is an extreme example of this bias, as it has many possible values but is a poor predictor of the target function over unseen instances.
- One alternative measure that has been used successfully is the gain ratio, which uses split information to discourage the selection of attributes with many uniformly distributed values.

- The gain ratio measure is defined in terms of the earlier gain measure and split information.

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

- A practical issue with using the gain ratio measure is that the denominator can be zero or very small when |S| ≈ |S1| or one of the Si, which makes the gain ratio undefined or very large.
- An alternative to the gain ratio measure is a distance-based measure introduced by Lopez de Mantaras, which defines a distance metric between partitions of the data and evaluates each attribute based on the distance between the data partition it creates and the perfect partition.
- This distance measure is not biased toward attributes with large numbers of values and can produce smaller trees in the case of data sets whose attributes have very different numbers of values.

Handling Training Examples with Missing Attribute Values:
In some cases, data may be missing for certain attributes in a dataset used for decision tree learning.

- One strategy for handling missing attribute values is to assign the most common value among training examples at the decision node.
- Another strategy is to assign probabilities to each possible value of the missing attribute based on observed frequencies among examples at the decision node.
- Fractional examples are created for the purpose of computing information gain and can be further subdivided at subsequent branches of the tree if a second missing attribute value must be tested.
- Fractional examples can also be used for classifying new instances whose attribute values are unknown. The classification is based on the most probable classification, computed by summing the weights of the instance fragments classified in different ways at the leaf nodes of the tree.

Handling Attributes with Differing Costs:
In some learning tasks, attributes may have associated costs.

- Decision trees that use low-cost attributes where possible are preferred.
- ID3 can be modified to take into account attribute costs by introducing a cost term into the attribute selection measure.
- Cost-sensitive measures bias the search in favour of low-cost attributes.

$$\frac{Gain^2(S, A)}{Cost(A)}$$

- Nunez (1988) describes a related approach and its application to learning medical diagnosis rules.

$$\frac{2^{Gain(S,A)} - 1}{(Cost(A) + 1)^w}$$

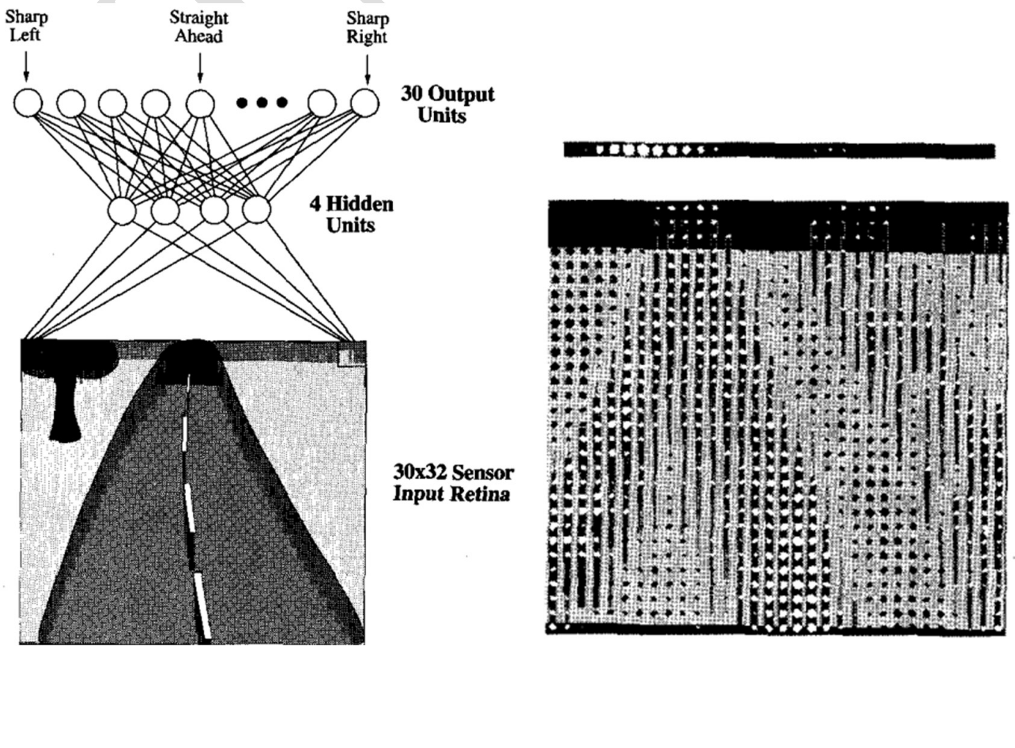
- The approach determines the relative importance of cost versus information gain.

- Decision tree construction becomes inefficient for large datasets that cannot fit in memory.
- More scalable approaches are required for handling bulk training data.
- Strategies to "save space" include discretizing continuous-valued attributes.

<u>Neural network representation in Artificial Neural Networks:</u>

Neural network learning methods are robust for approximating real-valued, discrete-valued, and vector-valued target functions and are particularly effective for learning to interpret complex real-world sensor data.

- ALVINN is a system that uses a learned ANN to steer an autonomous vehicle on public highways.
- The input to the neural network is a 30 x 32 grid of pixel intensities obtained from a forward-pointed camera mounted on the vehicle.
- The network output is the direction in which the vehicle is steered.
- The network has four hidden units that compute a single real-valued output based on a weighted combination of its 960 inputs.
- The hidden unit outputs are then used as inputs to a second layer of 30 "output" units.
- Each output unit corresponds to a particular steering direction, and the output values of these units determine which steering direction is recommended most strongly.
- The network structure of ALVINN is typical of many ANNs, with individual units interconnected in layers that form a directed acyclic graph.
- The most common and practical ANN approaches are based on the BACKPROPAGATION algorithm.
- The BACKPROPAGATION algorithm assumes the network is a fixed structure that corresponds to a directed graph, possibly containing cycles.
- The BACKPROPAGATION algorithm has been successful in various applications, such as recognizing handwritten characters, recognizing spoken words, and recognizing faces.



Appropriate problems for neural network learning:

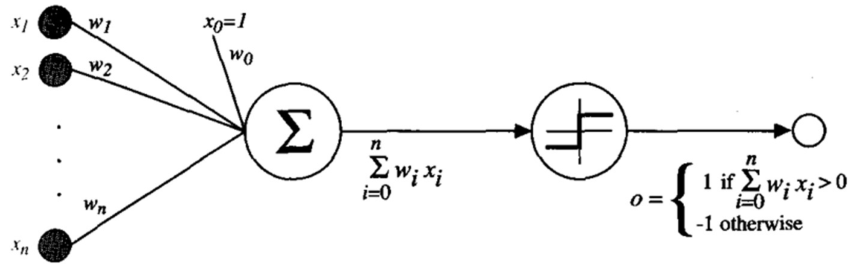
ANN learning is useful for problems with noisy and complex sensor data, as well as for problems with symbolic representations, such as decision tree learning tasks.

- The backpropagation algorithm is the most commonly used ANN learning technique.
- ANN learning is appropriate for problems with instances represented by many attribute-value pairs, with input values that can be any real values, and with discrete-valued or real-valued target function outputs.
- ANN learning methods are robust to noise in the training data.
- ANN training times can range from a few seconds to many hours, depending on various factors.
- Evaluating the learned target function with an ANN is typically very fast.
- The weights learned by neural networks are often difficult for humans to interpret, and learned neural networks are less easily communicated to humans than learned rules.

Perceptrons: Gradient descent and the Delta rule:

Perceptron:

Perceptron is a type of ANN system that takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs 1 if the result is greater than some threshold and -1 otherwise. The precise learning problem is to determine a weight vector that causes the perceptron to produce the correct ± 1 output for each of the given training examples.


x_1
w_1
x_2
w_2
$x_0=1$
w_0
w_n
x_n
Σ
$\sum_{i=0}^{n} w_i x_i$
$o = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$

- There are two algorithms for learning the weights for a single perceptron:
  - **(i) Perceptron rule**
    - The perceptron learning process begins with random weights and iteratively applies the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example.
    - The perceptron training rule revises the weight wi associated with input xi according to the rule: wi ← wi + n(t - o)xi, where t is the target output, o is the perceptron output, n is a positive constant called the learning rate, and xi is the input associated with weight wi.
    - The learning rate moderates the degree to which weights are changed at each step and is usually set to some small value (e.g., 0.1) and is sometimes made to decay as the number of weight-tuning iterations increases.

- The perceptron rule finds a successful weight vector when the training examples are linearly separable, but it can fail to converge if the examples are not linearly separable.

(ii) delta rule

- The delta rule is designed to overcome this difficulty by using gradient descent to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.
- The delta training rule is best understood by considering the task of training an unthresholded perceptron, which is a linear unit without the threshold.

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

- A measure for the training error of a hypothesis (weight vector) is defined, which is half the squared difference between the target output and the linear unit output, summed over all training examples.

$$E_d(\vec{w}) = \frac{1}{2}(t_d - o_d)^2$$

- This error measure is characterized as a function of the weight vector, and under certain conditions, the hypothesis that minimizes this error is also the most probable hypothesis given the training data according to Bayesian theory.

Stochastic Approximation of Gradient Descent:

Gradient descent is a strategy for searching through a large or infinite hypothesis space in machine learning.

- It is applied whenever the hypothesis space contains continuously parameterized hypotheses, and the error can be differentiated with respect to these hypothesis parameters.
- The key practical difficulties in applying gradient descent are slow convergence to a local minimum and the possibility of getting stuck in a local minimum.
- Incremental or stochastic gradient descent is a common variation on gradient descent that updates weights incrementally, following the calculation of the error for each individual example.
- Stochastic gradient descent approximates the gradient descent search by updating weights upon examining each training example.
- In standard gradient descent, the error is summed over all examples before updating weights, whereas in stochastic gradient descent weights are updated upon examining each training example.

```
GRADIENT-DESCENT(training_examples, $\eta$)
    Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values,
    $t$ is the target output value. $\eta$ is the learning rate (e.g., .05).
• Initialize each $w_i$ to some small random value
• Until the termination condition is met, Do
    • Initialize each $\Delta w_i$ to zero.
    • For each $\langle \vec{x}, t \rangle$ in training_examples, Do
        • Input the instance $\vec{x}$ to the unit and compute the output $o$
        • For each linear unit weight $w_i$, Do
                $\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$
    • For each linear unit weight $w_i$, Do
                $w_i \leftarrow w_i + \Delta w_i$
```

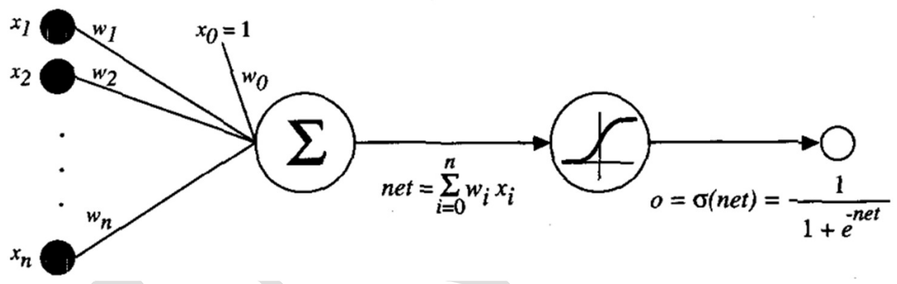
- Stochastic gradient descent can sometimes avoid falling into local minima because it uses various VEd(G) rather than VE(6) to guide its search.
- Both stochastic and standard gradient descent methods are commonly used in practice.

<u>Multilayer Networks and the Backpropagation Algorithm:</u>

Multilayer Networks:

Linear units produce only linear functions and are not suitable for highly nonlinear functions in multilayer networks.

- Perceptron units have a discontinuous threshold and are unsuitable for gradient descent.
- Sigmoid units are a possible solution as they produce a nonlinear function of their inputs, and their output is a differentiable function of its inputs.
- The output of a sigmoid unit is a continuous function of its input, which is computed using the sigmoid function.
- The sigmoid function has the property that its derivative is easily expressed in terms of its output, which is used in the gradient descent learning rule.
- Other differentiable functions with easily calculated derivatives are sometimes used in place of the sigmoid function.


x_1
w_1
$x_0 = 1$
w_0
x_2
w_2
w_n
x_n
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1+e^{-net}}$

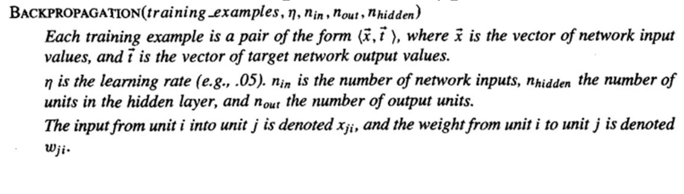
Backpropagation Algorithm:

The BACKPROPAGATION algorithm learns weights for a multilayer network to minimize the squared error between network output values and target values.

- The algorithm employs gradient descent to search a large hypothesis space defined by all possible weight values for all units in the network.
- The error is defined as the sum of errors over all network output units.

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

- The learning problem faced by BACKPROPAGATION can be visualized as an error surface, where the other dimensions of the space correspond to all of the weights associated with all units in the network.
- Gradient descent can be used to attempt to find a hypothesis to minimize the error.



BACKPROPAGATION(*training_examples*, η, n_{in}, n_{out}, n_{hidden})

Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$, where $\vec{x}$ is the vector of network input values, and $\vec{t}$ is the vector of target network output values.

η is the learning rate (e.g., .05). n_{in} is the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.

The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}.

```
• Create a feed-forward network with $n_{in}$ inputs, $n_{hidden}$ hidden units, and $n_{out}$ output units.
• Initialize all network weights to small random numbers (e.g., between −.05 and .05).
• Until the termination condition is met, Do
    • For each $\langle \vec{x}, \vec{t} \rangle$ in training_examples, Do
        Propagate the input forward through the network:
        1. Input the instance $\vec{x}$ to the network and compute the output $o_u$ of every unit $u$ in
           the network.
        Propagate the errors backward through the network:
        2. For each network output unit $k$, calculate its error term $\delta_k$
                $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$                                (T4.3)
        3. For each hidden unit $h$, calculate its error term $\delta_h$
                $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh} \delta_k$      (T4.4)
        4. Update each network weight $w_{ji}$
                $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$
           where
                $\Delta w_{ji} = \eta \, \delta_j \, x_{ji}$                                   (T4.5)
```

- Many variations of Backpropagation have been developed.
- One common variation is to alter the weight-update rule in Equation (T4.5) in the algorithm.
- This variation makes the weight update on the nth iteration depend partially on the update that occurred during the (n-1)th iteration.
- The weight update is adjusted using the term αΔwji(n-1), where α is a constant between 0 and 1 called the momentum, and Δwji(n-1) is the weight update performed during the (n-1)th iteration.

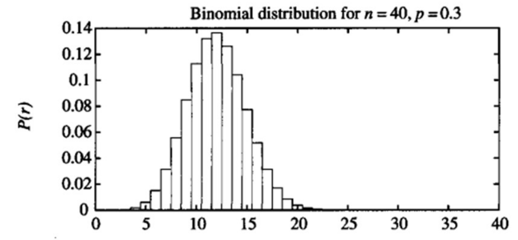
$$\Delta w_{ji}(n) = \eta \, \delta_j \, x_{ji} + \alpha \Delta w_{ji}(n-1)$$

- This momentum term helps to reduce oscillations in the weight updates during training.
- By adding a portion of the previous weight update to the current update, the algorithm can continue to make progress in the same direction, even if the gradient changes direction in subsequent iterations.
- The momentum term also helps the algorithm escape local minima more easily.

Basics of Sampling Theory:

- A *random variable* can be viewed as the name of an experiment with a probabilistic outcome. Its value is the outcome of the experiment.
- A *probability distribution* for a random variable Y specifies the probability $\Pr(Y = y_i)$ that Y will take on the value y_i, for each possible value y_i.
- The *expected value*, or *mean*, of a random variable Y is $E[Y] = \sum_i y_i \Pr(Y = y_i)$. The symbol μ_Y is commonly used to represent E[Y].
- The *variance* of a random variable is $Var(Y) = E[(Y - \mu_Y)^2]$. The variance characterizes the width or dispersion of the distribution about its mean.
- The *standard deviation* of Y is $\sqrt{Var(Y)}$. The symbol σ_Y is often used used to represent the standard deviation of Y.
- The *Binomial distribution* gives the probability of observing r heads in a series of n independent coin tosses, if the probability of heads in a single toss is p.
- The *Normal distribution* is a bell-shaped probability distribution that covers many natural phenomena.
- The *Central Limit Theorem* is a theorem stating that the sum of a large number of independent, identically distributed random variables approximately follows a Normal distribution.
- An *estimator* is a random variable Y used to estimate some parameter p of an underlying population.
- The *estimation bias* of Y as an estimator for p is the quantity $(E[Y] - p)$. An unbiased estimator is one for which the bias is zero.
- A *N% confidence interval* estimate for parameter p is an interval that includes p with probability $N\%$.

Error Estimation and Estimating Binomial Proportions:



A *Binomial distribution* gives the probability of observing r heads in a sample of n independent coin tosses, when the probability of heads on a single coin toss is p. It is defined by the probability function

$$P(r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r}$$

If the random variable X follows a Binomial distribution, then:

- The probability $\Pr(X = r)$ that X will take on the value r is given by $P(r)$
- The expected, or mean value of X, $E[X]$, is

$$E[X] = np$$

- The variance of X, $Var(X)$, is

$$Var(X) = np(1-p)$$

- The standard deviation of X, σ_X, is

$$\sigma_X = \sqrt{np(1-p)}$$

For sufficiently large values of n the Binomial distribution is closely approximated by a Normal distribution (see Table 5.4) with the same mean and variance. Most statisticians recommend using the Normal approximation only when $np(1-p) \geq 5$.

The Binomial Distribution:

The Binomial distribution is used to describe the probability of observing a specific number of successes (heads) in a series of n independent trials with a constant probability of success (p).

- The example of estimating the probability of heads in a worn and bent coin by tossing it n times and recording the number of times it turns up heads (r) is used to explain the Binomial distribution.
- Estimating the probability of heads from a random sample of coin tosses is equivalent to estimating errorv(h) from testing h on a random sample of instances.
- The Binomial distribution applies to any base experiment with an outcome described by a random variable, Y, that can take on two possible values and a constant probability of success, p.

$$R \equiv \sum_{i=1}^{n} Y_i$$

- The number of successes in a series of n independent trials is denoted by R and the probability of observing a specific number of successes is given by the Binomial distribution formula.

$$\Pr(R = r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r}$$

Mean and Variance:

Consider a random variable Y that takes on the possible values yl, . . . yn. The expected value of Y, E[Y], is

$$E[Y] \equiv \sum_{i=1}^{n} y_i \Pr(Y = y_i)$$

The variance, captures the "width or "spread" of the probability distribution i.e., it captures how far the random variable is expected to vary from its mean value.

The **variance** of a random variable Y, $Var[Y]$, is

$$Var[Y] \equiv E[(Y - E[Y])^2]$$

The square root of the variance is called the standard deviation.

The **standard deviation** of a random variable Y, σ_Y, is

$$\sigma_Y \equiv \sqrt{E[(Y - E[Y])^2]}$$

Estimators, Bias, and Variance:

$$error_S(h) = \frac{r}{n}$$

$$error_{\mathcal{D}}(h) = p$$

where n is the number of instances in the sample S, r is the number of instances from S misclassified by h, and p is the probability of misclassifying a single instance drawn from $\mathcal{D}$.

The **estimation bias** of an estimator Y for an arbitrary parameter p is

$$E[Y] - p$$

the standard deviation for $error_S(h)$ is given by

$$\sigma_{error_S(h)} = \frac{\sigma_r}{n} = \sqrt{\frac{p(1-p)}{n}}$$

$$\sigma_{error_S(h)} \approx \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

Confidence Intervals:

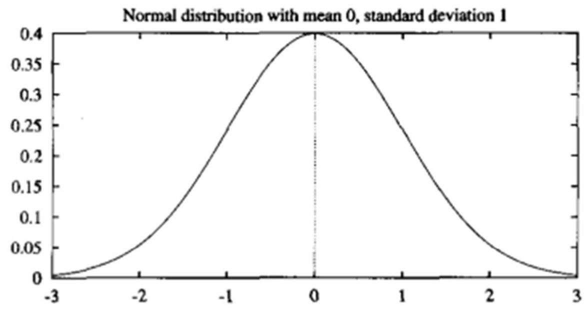
Confidence interval estimates provide a way to express the uncertainty of an estimate by giving a range of values within which the true value is likely to lie, along with the probability that it falls within that range.

- An N% confidence interval for some parameter p is an interval that is expected with probability N% to contain p.

$$\mu \pm z_N \sigma$$

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

- The Normal distribution is is a bell-shaped distribution fully specified by its mean μ and standard deviation σ
- For large n, any Binomial distribution is very closely approximated by a Normal distribution with the same mean and variance.
- The Central Limit Theorem states that the sum of a large number of independent, identically distributed random variables follows a distribution that is approximately Normal.



A Normal distribution (also called a Gaussian distribution) is a bell-shaped distribution defined by the probability density function

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

A Normal distribution is fully determined by two parameters in the above formula: μ and σ.

If the random variable X follows a normal distribution, then:

- The probability that X will fall into the interval (a, b) is given by

$$\int_a^b p(x)dx$$

- The expected, or mean value of X, $E[X]$, is

$$E[X] = \mu$$

- The variance of X, $Var(X)$, is

$$Var(X) = \sigma^2$$

- The standard deviation of X, σ_X, is

$$\sigma_X = \sigma$$