**Module-4**

**Artificial Neural Networks(ANN):**

ANN’s have gained a lot of traction in recent years as one of the most efficient and rapid learning approaches for solving complicated computer vision, speech recognition, and other problems.

It was designed to mimic the behavior of biological systems made up of “neurons.” ANNs are computer models based on the central nervous systems of animals. It has machine learning and pattern recognition capabilities. These were shown as interconnected “neurons” that could compute values based on inputs.

**Why ANN’s?**

* Artificial neural networks (ANNs) are a method for learning real-valued, discrete-valued, and vector-valued functions from examples that are both broad and practical.
* Learning has been effectively used to challenges such as understanding visual sceneries, voice recognition, and learning robot control techniques learning to recognize handwritten characters, spoken words, and faces learning to recognize handwritten characters, spoken words, and faces.
* Artificial neural networks are one of the most effective learning approaches now available for challenges like learning to comprehend complicated real-world sensor data.

**What is a Neural Network?**

A neural network is a machine learning algorithm that is based on a neuron model. There are millions of neurons in the human brain. It uses electrical and chemical impulses to deliver and process signals.

Synapses are a particular structure that connects these neurons. Synapses allow neurons to communicate with one another. Neural networks are formed from a huge number of simulated neurons.

We can use neural networks for more than just categorization. It may also be used for continuous target attribute regression.

In the field of data mining, neural networks have a lot of applications. For example, pattern recognition in economics, forensics, and other fields.

After thorough training, it may also be utilized for data categorization in vast amounts of data.

**Layers in Neural Network:**

The following three layers may be found in a neural network:

##### **Input layer**: The raw data that may be fed into the network is represented by the activity of the input units.

To identify the activity of each hidden unit, use the **hidden layer**. The weights on the links between the input and hidden units, as well as the actions of the input units. There might be one or more layers buried under the surface.

The activity of the hidden units and the weights between the hidden and output units determine the behavior of the output units in the **output layer.**  
An Artificial Neural Network is a type of data processing system. It functions in the same manner as the human brain does. ANN is made up of a large number of interconnected processing units that collaborate to process data. They also produce useful outcomes as a result of it.

**Neural Network Representation:**

ALVINN, a system that employs a trained ANN to guide an autonomous car operating at typical speeds on public highways, is an example of ANN learning.

A 30 by 32 grid of pixel intensities collected from a forward-pointing camera installed on the vehicle is sent into the neural network.

The network output determines the vehicle’s steering direction.

ALVINN has successfully driven at speeds of up to 70 miles per hour for distances of up to 90 miles on public roadways using its learned networks.

The neural network representation utilized in one version of the ALVINN system is shown in the above figure.

The input camera picture is exhibited below the network, which is illustrated on the left side of the diagram. The lines entering the node from below are its inputs, and each node (i.e., circle) in the network diagram corresponds to the output of a single network unit.

There are four units that get direct input from all of the image’s 30 × 32 pixels. Because their output is only available within the network and not as part of the global network output, these units are referred to as “hidden” units. Based on a weighted combination of its 960 inputs, each of these four hidden units produces a single real-valued output.

The outputs of these concealed units are then sent into the second layer of 30 “output” units.

Each output unit is associated with a certain steering direction, and the output values of these units decide which steering direction is most strongly advised.

The artificial neurons form the nodes of an artificial neural network, which is best described as a weighted directed graph.

The directed edges with weights represent the relationship between neuron outputs and neuron inputs. The Artificial Neural Network gets the input signal in the form of a pattern and a picture in the form of a vector from an external source.

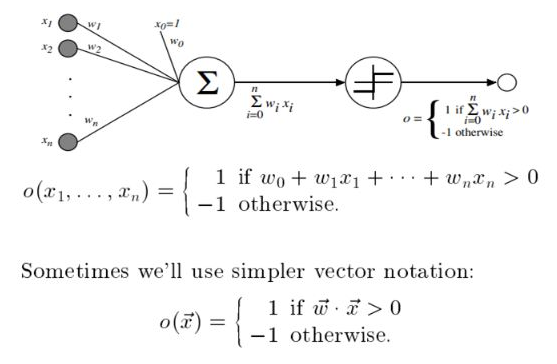
The notations x(n) are then used to mathematically allocate these inputs for each n number of inputs.

**PERCEPTRONS**

A neural network’s fundamental unit is a single-layer perceptron. Input values, weights, and a bias, as well as a weighted sum and activation function, make up a perceptron .

The basis of an ANN system is a unit known as a perceptron.

A real-valued vector input is taken as an input by the perceptron, and a linear combination of it is calculated. The perceptron either outputs a 1 if the result exceeds the threshold or -1.

The perceptron computes the output given the inputs through .

Sometimes we will use the simpler vector notation

Where is a weighted real-valued constant that specifies the contribution of input to the perceptron output.

The amount (-) is a threshold that must be exceeded by the weighted combination of inputs for the perceptron to output a 1.

Imagine an extra constant input = 1 to simplify notation, enabling us to write the above mentioned inequality as  or in vector, form as

For convenience, we will sometimes write the perceptron function as

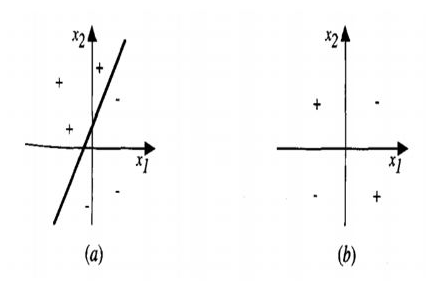
Where

Choosing values for the weights w0,…..wn is part of learning a perceptron. As a result, the set of all potential real-valued weight vectors is the space H of candidate hypotheses examined in perceptron learning.

**Representational Power of Perceptrons:**

* In the n-dimensional space of occurrences, the perceptron may be seen as a hyperplane decision surface (i.e points).
* For occurrences on one side of the hyperplane, the perceptron produces a 1; for instances on the other side, it emits a -1. This decision hyperplane’s equation is
* No hyperplane can distinguish some groups of positive and negative examples. Linearly separable collections of examples are those that can be separated.
* Many boolean functions can be represented by a single perceptron.
* If we assume boolean values for 1(true) and -1(false), one approach to construct the AND function with a two-input perceptron is to set the weights  = -0.8 and  =  = 0.5. By changing the threshold to = -0.3, this perceptron may now represent the OR function.
* All of the elementary boolean operations AND, OR, NAND (! AND), and NOR (!OR) may be represented by perceptrons.
* Unfortunately, some boolean functions, such as the XOR function, which returns 1 if and only if  != , cannot be represented by a single perceptron.

#### Decision Surface of a Perceptron:



A two-input perceptron represents the decision surface. (a) A collection of training examples and a perceptron’s decision surface that correctly classifies them. (b) A non-linearly separable collection of training instances (i.e, that cannot be correctly classified by any straight line). The perceptron inputs are  and . ‘+’ denotes positive instances, whereas ‘-‘ denotes negative examples.

* Every boolean function may be represented by a two-level network of perceptrons, with the inputs sent to several units and the outputs of these units being fed to a second, final stage.
* Because single units cannot represent a wide range of functions, we will be interested in learning multilayer networks of threshold units.

**The Perceptron Training Rule:**

* This learning issue may be solved using a variety of techniques. The perceptron rule and the delta rule are the two algorithms we’ll look at here (a variant of the LMS rule for learning evaluation functions).
* Under slightly different conditions, these two algorithms are guaranteed to converge to slightly different accepted hypotheses.
* They are crucial to ANNs because they serve as the foundation for large-scale learning networks.
* Starting with random weights, iteratively apply the perceptron rule to each training example, adjusting the perceptron weights anytime it misclassifies an example is one technique to discover an acceptable weight vector.
* This approach is repeated as needed, iterating through the training instances until the perceptron properly classifies all training examples.

At each step, the perceptron training rule is followed and the weights are changed which alters  and  of the input based on the rule.

The learning rate’s purpose is to regulate the amount of weight change at each stage.

A low value like 0.1 is usually set and can be assigned to decay as the number of weight-turning iterations grows.

where

Where:

O is perceptron

is small constant(e.g..1) called learning rate

For example,

If  =.8, = 0.1, t = 1, and o = – 1.

The weight update is= (t – o) = 0.1(1 – (-1))0.8 = 0.16.

Weights linked with positive  , on the other hand, will be lowered rather than raised if t = -1 and o = 1.

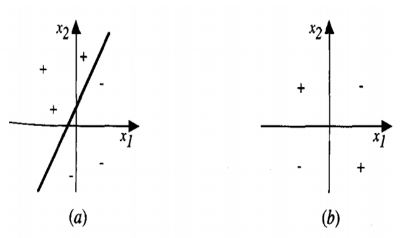
In reality, if the training instances are linearly separable and n is small enough, the following learning technique may be shown to converge to a weight vector that properly classifies all training examples in a limited number of perceptron training rule operations.

Convergence is not guaranteed if the data are not linearly separable.

To summarize, A perceptron operates by accepting numerical inputs and combining them with weights and a bias. It then multiplies these inputs by the weights assigned to them (this is known as the weighted sum). These items, together with the bias, are then combined. The weighted sum and bias are inputs to the activation function, which returns a final output.

**Gradient descent and Delta rule**

If a set of data points can be separated into two groups using a straight line, the data is said to be linearly separable. Non-linearly separable data is defined as data points that cannot be split into two groups using a straight line.



**Figure (a) -> Training Set is Linearly Separable**

**Figure (b) -> Training Set is non-linearly Separable**

When the training instances are linearly separable, the perceptron algorithm finds a successful weight vector; however, if the examples are not linearly separable, they may fail to converge.

The delta rule, a second training rule, is meant to address this challenge. In this blog, we’ll have a brief look at Gradient Descent and Delta Rule.

The delta rule converges toward a best-fit approximation to the target concept if the training instances are not linearly separable.

**Delta Rule’s Main Idea:**

The**Delta rule’s** main idea is to explore the hypothesis space of potential weight vectors using gradient descent to discover the weights that best suit the training instances.

This criterion is significant because the backpropagation algorithm, which can train networks with many linked units, is based on gradient descent.

**DERIVATION OF DELTA RULE:**

Consider the job of training a threshold perceptron, which is a linear unit whose output o is given by,

to better understand the delta training algorithm.

Let’s start by providing a measure for the training error of a hypothesis(weight vector) relative to the training instances in order to build a weight learning algorithm for linear units. Although there are a variety of methods to describe this inaccuracy, one frequent metric that will prove to be particularly useful is

Where D is the set of training examples., represents the target output for training example d, and represents the linear unit’s output for training example d.

According to this definition, is half the squared difference between the target output and the linear output , summed by all training samples.

**Derivation of Gradient Descent Rule**

To compute the steepest descent direction along the error surface following steps are done.

Computing the derivative of E with respect to each component of the vector w yields the steepest direction. The gradient of E with respect to w, denoted as is the vector derivative of this vector.

.......................(1)

The gradient determines the direction of the sharpest rise in E, hence the gradient descent training rule is

Where

......................(2)

The learning rate, or n, is a positive constant that controls the step size in the gradient descent search.

Because we wish to shift the weight vector in the direction of decreasing E, the negative sign is present

This training rule may also be expressed as a collection of components.

Where = .....................(3)

we know that ..............(4)

Differentiating equation 4 to obtain

Substitute in the above equation

...........(5)

For training example d,  signifies the single input component

We now have an equation that expresses the training examples’ inputs, outputs, and target values td in terms of linear unit inputs , outputs and target values

The weight update rule for gradient descent is obtained by substituting Equation (5) into Equation (3).

=

= ...... (6)

**Gradient-descent algorithm for training a linear unit:**

***GADIENT-DESCENT* (*training\_examples*, )**

*Each training example is a pair of the form* **(, *t),*** *where*  *is the vector of input values, and*  *t is the target output value. is the learning rate (e.g., .05).* .

Initialize each *,* to some small random value

Until the termination condition is met, Do

* Initialize each to zero.
* For each **(, *t )***in *training\_examples,* Do
* Input the instance x' to the unit and compute the output
* For each linear unit weight *,* Do

............(7)

For each linear unit weight *,* Do

.........(8)

To implement the stochastic approximation to gradient descent, Equation (8) is deleted, and Equation (7) is replaced by

To apply gradient descent algorithm for training linear units: Choose a random weight vector as your starting point.

After applying the linear unit to all training samples, determine each weight using Equation(6).

By adding to each weight , you may update it and then repeat the procedure.

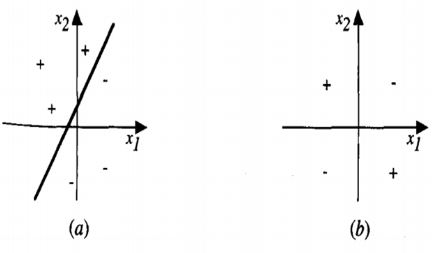
The algorithm is specified above. Because the error surface only has one global minimum, this approach will converge to a weight vector with the lowest error, regardless of whether the training instances are linearly separable, if the learning rate is small enough.

If is too big, the gradient descent search may end up overstepping the error surface’s minimum instead of settling into it.

When a result, as the number of gradient descent steps rises, one typical tweak to the method is to progressively lower the value of .

**Multilayer Neural Networks**

A single-layer neural network will work only for linearly separable data and not for non-linearly separable data. Hence there is a need for Multilayer Neural Networks, to be able to work with non-linearly separable data.



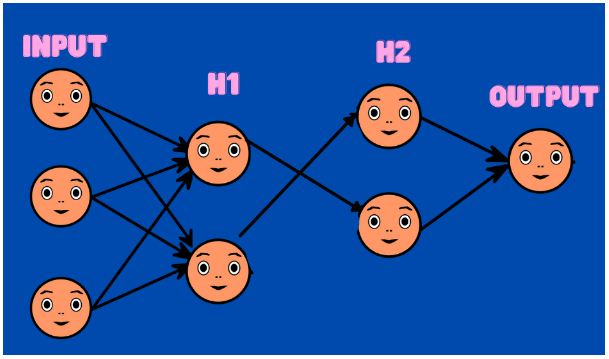
**Figure (a) -> Training Set is Linearly Separable**

**Figure (b) -> Training Set is non-linearly Separable**

A multi-layer Neural Network has two hidden layers. Hidden layers, whose neurons are not directly linked to the output, are used in multilayer networks to address the classification issue for non-linear data.

The hidden layers can be understood geometrically as extra hyper-planes that increase the network’s separation capability. Typical multilayer network designs are seen in the Figure below.

This new design raises a new challenge: how to train concealed units whose expected output is unknown. This problem can be solved using the **Backpropagation technique**.



**Backpropagation technique:**

Given a network with a defined set of units and linkages, the backpropagation algorithm learns the weights for a multilayer network.

It uses gradient descent to try to reduce the squared error between the network output values and the outputs’ goal values.

We begin by redefining E to total the errors over all of the network output units because we are examining networks with multiple output units rather than single units as previously.

The technique presented here is applicable to layered feed forward networks with two levels of sigmoid units, each layer’s units being linked to all units from the previous layer.

Each node in the network is given an index (for example, an integer), where a “node” is either a network input or the output of a network unit.

The input from node i to unit j is denoted by , while the associated weight is denoted by

The error term associated with unit n is denoted by  . It functions similarly to the amount (t – o) from our previous explanation of the delta training rule. We’ll see what happens afterward

**BACKPROPOGATION ALGORITHM:**

The approach starts by building a network with the necessary number of hidden and output units, as well as setting all network weights to tiny random values.

The main loop of the algorithm then iterates over the training instances using this fixed network topology.

It applies the network to each training example, determines the network output error for this example, computes the gradient with regard to the error for this example, and then updates all network weights.

This gradient descent phase is repeated until the network performs satisfactorily (sometimes thousands of times, using the same training samples each time).

The delta training rule is comparable to the gradient descent weight-update rule. It changes each weight according to the learning rate , the input value , to which the weight is applied, and the error in the unit’s output, just as the delta rule.

The main change is that in the delta rule, the error (t – o) is substituted with a more complicated error term  , whose exact form of   derives from the weight tuning rule’s derivation.

Consider how    is computed for each network output unit k to get a sense of how it works.

is simply from the delta rule multiplied by the quantity    that is the sigmoid squashing function’s derivative.

The     value for each hidden unit h  However, because target values  are only provided for network outputs in training instances, no target values are explicitly accessible to signal the inaccuracy of concealed unit values.

Rather, the error term for hidden unit h is determined by adding the error     terms for each output unit impacted by h and weighting each    by , the weight from hidden unit h to output unit k. The degree to which hidden unit h is “responsible” for the inaccuracy in output unit k is represented by this weight.

***BACKPROPAGATION(training\_examples,* , *)***

*Each training example is a pair of the form* **(, *),*** *where* **,** *is the vector of network input*

*values, and is the vector of target network output values.*

*is the learning rate (e.g.,* ***.O5).*** *, is the number of network inputs, the number of*

*units in the hidden layer, and , the number of output units.*

*The input from unit i into unit j is denoted , and the weight from unit* ***i*** *to unit j is denoted*

* Create a feed-forward network with *,* inputs, hidden units, and 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 **(, *),*** in *training\_examples,* **Do**

*Propagate the input forward through the network:*

1. Input the instance  to the network and compute the output *,* of every unit ***u*** in the network.

*Propagate the errors backward through the network:*

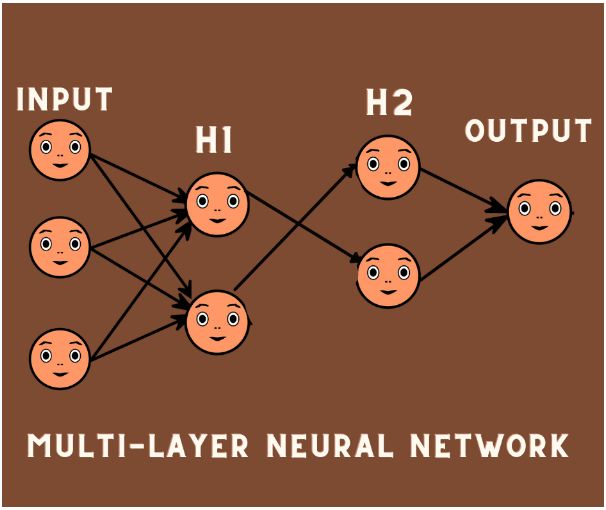
1. For each network output unit k, calculate its error term
2. For each hidden unit *h,* calculate its error term
3. Update each network weight

Where

The Stochastic gradient version of back propagation algorithm is as shown above

**Derivation of Backpropogation Rule**

Backpropagation’s purpose is to find the partial derivatives of the cost function C for every weight w or bias b in the network. It is a supervised learning algorithm used for Multilayer Perceptrons (Artificial Neural Networks).



We’ll use the product of some constant alpha and the partial derivative of that quantity with respect to the cost function to update the weights and biases in the network once we get these partial derivatives. This is the famously-known gradient descent method.

The partial derivatives provide the largest ascending direction. As a result, we take a modest step in the opposite direction — the route of greatest descent, that is, the direction that will lead us to the cost function’s local minima.

**What is Backpropagation, and how does it work?**

Using a concept known as the delta rule or gradient descent, the Backpropagation algorithm hunts for the least value of the error function in weight space. The weights that minimize the error function are therefore regarded as a learning problem solution.

Let’s look at an example,

Consider you have a labeled data set.

|  |  |
| --- | --- |
| Input | Desired Output |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |

When the value of “W” is 3, the following is the output of your model:

|  |  |  |
| --- | --- | --- |
| Input | Desired Output | Model Output(W=3) |
| 0 | 0 | 0 |
| 1 | 2 | 3 |
| 2 | 4 | 6 |

The absolute error for three instances is 0, 1, 2 and the squared error is 0, 1, 4. As we increase the value of W, the error increases. But if you decrease the value of W, the error decreases.

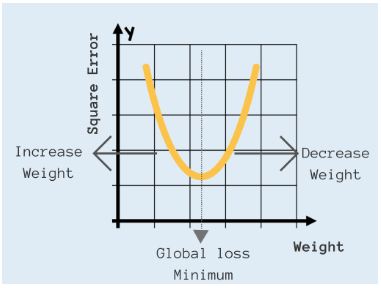
**Steps:**

* We started by setting a random value to ‘W’ and then propagated forward.
* Then we realized there was a mistake. We propagated backward and raised the value of ‘W’ to lessen the mistake.
* We also found that the error had risen after that. We discovered that we are unable to increase the ‘W’ value.
* As a result, we propagated backward one more and lowered the ‘W’ value.
* We have now found that the error has decreased.

As a result, we’re attempting to find a weight value that minimizes the inaccuracy. Essentially, we must determine if the weight value should be increased or decreased.

Once we know that, we continue to update the weight value in that direction until the mistake is as little as possible. You may reach a point where updating the weight further increases the inaccuracy. You must stop at that point, and that is your ultimate weight value.

Consider the following graph:



We must achieve the ‘Global Loss Minimum.’ This is called Back Propagation.

**Derivation of the BACKPROPAGATION Rule:**

For each training example d descending the gradient of the error Ed with respect to this single example. In other words, for each training example d, every weight is updated by adding to it

...(1)

Where  is the total of all output units in the network’s error on training example d.

...(2)

Let

* = the input to unit j
* = the weight associated with input to unit j
* (the weight sum of inputs for unit j)
* = output computed by unit j
* = target output for unit j
* σ =  the sigmoid function
* outputs = the set of units in the final layer of the network
* Downstream(j) = the set of units whose immediate inputs include the output of unit j

In order to implement the stochastic gradient descent rule, we must first develop an equation for

To begin, keep in mind that weight  can only affect the rest of the network via As a result, we may write using the chain rule.

**.....(3)**

Our final objective is to construct a suitable expression for  given the above equation.

We investigate two scenarios in turn: one in which unit j is a network output unit, and another in which unit j is an internal unit.

**Case1: Training Rule for Output unit weights**

can only impact the rest of the network through , and  can only influence the rest of the network through . As a result, we may use the chain rule to write again

....(4)

To begin, consider just the first term in the above equation,

The derivatives   will be zero for all output units k except when k = j. We, therefore, drop the summation over output units and simplify set k = j.

........(5)

Next, consider the second term in Equation(4). Since  σ  , the derivative   is just the derivative of the sigmoid function, which we have already noted is equal to σ

.....(6)

Substitute equation (5) and (6) in equation (4)

......(7)

The stochastic gradient descent rule for output units is obtained by combining Equations (1) and (3).

**Case 2: Training rule for hidden unit weights**

When j is a network’s internal, or hidden, unit, the training rule for i must account for the indirect ways in which  might impact the network outputs, and so **.**

Downstream is the name given to this group of components ( j). can only control the network outputs (and hence **.**) through the units in the Downstream (j). As a result, we may write

Rearranging terms and using to denote we have

and

**Bayesian Learning:**

Bayesian machine learning is a subset of probabilistic machine learning approaches (for other probabilistic models, see Supervised Learning).

In Bayesian learning, model parameters are treated as random variables, and parameter estimation entails constructing posterior distributions for these random variables based on observed data.

**Why Bayesian Learning Algorithms?**

For two reasons, Bayesian learning approaches are relevant to machine learning.

* To begin, Bayesian learning algorithms compute explicit probabilities for hypotheses.
* The second reason is that they aid comprehension of various learning methods that do not involve probability manipulation.

**Features of Bayesian learning methods include:**

* Each observed training example can reduce or enhance the estimated chance that a hypothesis is correct by a small amount.
* This is more flexible than methods that fully discard a hypothesis if it is discovered to be inconsistent with any single example. To assess the final probability of a hypothesis, prior knowledge can be merged with observed data.
* Hypotheses that make probabilistic predictions can be accommodated by Bayesian approaches (e.g., hypotheses such as “this pneumonia patient has a 93 percent chance of complete recovery”).
* The validity of a proposition is calculated via Bayesian Estimation.
* The proposition’s validity is determined by two factors:
* i). Preliminary Estimate
* ii). New evidence that is relevant.

**Practical Issues:**

* One practical issue with using Bayesian methods is that they often require prior knowledge of a large number of possibilities. When these probabilities aren’t known ahead of time, they’re calculated using prior knowledge, data, and assumptions about the shape of the underlying distributions.
* The substantial processing cost necessary to determine the Bayes optimum hypothesis in the general case is a second practical obstacle

**BAYES THEOREM:**

Consider a typical machine learning task. You have a set of training data, inputs, and outputs, and you’d like to figure out how to map them together.

As a result, you piece together a model and soon have a deterministic way of making predictions for a target variable y given an unknown input x.

There’s only one problem: you have no method of explaining what’s going on inside your model! You just know it was trained to minimize some loss function on your training data, but that’s not much information. In an ideal world, you’d have an objective summary of your model’s parameters, complete with confidence intervals and other statistical morsels, and you’d be able to reason about them in probability terms.

This is where Bayesian Machine Learning enters the picture.

The Bayes theorem is a method for calculating a hypothesis’s probability based on its prior probability, the probabilities of observing specific data given the hypothesis, and the seen data itself.

**Bayes theorem definition**,

* Before we view the training data, we use P(h) to signify the starting probability that hypothesis h holds.
* P(h) is also known as the prior probability of h, and it might reflect whatever prior knowledge we have about the likelihood that h is right.
* If we don’t have any prior knowledge, we may just give each candidate’s hypothesis the same prior probability.
* Similarly, P(D) is the prior probability of observing training data D. (i.e., the probability of D given no knowledge about which hypothesis holds).
* The likelihood of seeing data D in some environment where hypothesis h holds is denoted by P(D/h).
* Given the observed training data D, the probability P(h/D) that h holds.
* The posterior probability of h is designated P(h/D) because it represents our confidence that h holds after seeing the training data D.
* In contrast to the prior probability P(h), which is independent of D, the posterior probability P(h/D) indicates the influence of the training data D.
* From the prior probability P(h), as well as P(D) and P(D/h), the Bayes theorem can be used to compute the posterior probability P(h/D).

Bayes theorem:

* According to Bayes theorem, P(h/D) grows with P(h) and P(D/h).

As P(D) grows, P(h/D) drops.

* In many learning scenarios, the learner considers a collection of candidate hypotheses H and is looking for the most likely hypothesis hH given the observed data D, or at least one of the most likely if there are numerous.
* A maximum a posteriori (MAP) hypothesis is any maximally likely hypothesis.

* The posterior probability of each candidate hypothesis can be calculated using the Bayes method to identify the MAP hypotheses.
* P(D) was deleted because it is a constant.
* We’ll suppose that every hypothesis in H is equally likely a priori in some instances (P() = P() for all and in H).
* In this scenario, we may simplify the equation even more by simply considering the quantity P(D/h) when determining the most likely hypothesis.
* The likelihood of the data D given h is typically referred to as P(D/h), and any hypothesis that maximizes P(D/h) is referred to as a maximum likelihood (ML) hypothesis.

We’ve already seen one use of Bayes Theorem in the analysis of Knowledge Cascades, we discovered that based on the conditional probabilities computed using Bayes’ Theorem, reasonable decisions may be made where one’s own personal information is omitted.

**Application of the Bayes Theorem:**

* The theorem has a wide range of applications that aren’t confined to finance.
* Bayes’ theorem, for example, can be used to estimate the accuracy of medical test findings by taking into account how probable any specific person is to have a condition as well as the test’s overall accuracy.

**Example:**

Consider a medical diagnosis problem, There are two different hypotheses:

(1) the patient has a certain type of cancer

(2) the patient does not have cancer.

The information is based on a laboratory test with two possible outcomes: + (positive) and – (negative) (negative).

We already knew that only.008 percent of the population is infected with this disease. Furthermore, the laboratory test is merely an imperfect indicator of disease.

Only 98 percent of the time does the test give an accurate positive result when the disease is present, and only 97 percent of the time does it give a valid negative result when the disease is not there. In other circumstances, the test yields the opposite outcome.

That is

Let’s say we come across a new patient who has a positive lab test result. Should we give the patient a cancer diagnosis or not? Below equation can be used to find the maximal a posteriori hypothesis

(.98)(.008)=.0078

Thus, . The exact posterior probabilities can also be determined by normalizing the above quantities so that they sum to 1 (e.g., == .21). This step is warranted because Bayes theorem states that the posterior probabilities are just the above quantities divided by the probability of the data, P(). Although P() was not provided directly as part of the problem statement, we can calculate it in this fashion because we know that P() and P() must sum to 1 (i.e., either the patient has cancer or they do not). Notice that while the posterior probability of cancer is significantly higher than its prior probability, the most probable hypothesis is still that the patient does not have cancer.

Let’s have a look at the Problem of Probability Density Estimation,

Given a sample of observations (X) from a domain (), each observation is taken independently from the domain with the same probability distribution (so-called independent and identically distributed, i.i.d., or close to it).

Density estimation entails choosing a probability distribution function and its parameters that best explain the observed data’s joint probability distribution (X).

There are several approaches to tackling this problem, but two of the most common are:

A Bayesian approach is called Maximum a Posteriori (MAP).

The small p indicates the probability density function. Maximum likelihood estimation (MLE) is a statistical technique for estimating the parameters of a probability distribution based on observed data.

This is accomplished by maximizing a probability function such that the observed data is most likely under the assumed statistical model. The maximum likelihood estimate is the point in the parameter space that maximizes the likelihood function.

**Bayes Theorem and Concept Learning:**

The steps for brute force concept learning:

We can design a straightforward concept learning algorithm to output the maximum a posterior hypothesis based on Bayes theorem as follows

Given the training data, the Bayes theorem determines the posterior probability of each hypothesis. It calculates the likelihood of each conceivable hypothesis before determining which is the most likely.

1. For each hypothesis h in H calculate the posterior probability
2. Output the hypothesis  with the highest posterior probability.

To calculate, we need to know the values of P(h) and P(D/h). To choose these to be consistent with the following assumptions:

1. There is no noise in the training data D (i.e., = .
2. The hypothesis space H contains the goal notion c.
3. We have no reason to conclude that one hypothesis is more likely than another based on prior evidence.

* Since we’re assuming the training data to be noise-free, the chances of observing classification given h are 1 if  = h() and 0 if  != ()). Therefore,

To put it another way, the probability of data D given hypothesis h is 1 if D agrees with h and 0 otherwise.

* Given no previous information of which hypothesis is more likely, it is fair to give each hypothesis h in H the same prior probability. We should require that these prior probabilities amount to 1 because we presume the target notion is contained in H.

Now, let’s consider two cases:

**Case 1:** h is inconsistent with the training data D.

Here, since we know that P(D/h) = 0 when h is inconsistent with D. We have,

That is, The posterior probability of a hypothesis inconsistent with D is zero

**Case 2:** Consider a case where h is consistent with D. and since we know for h consistent with D, the value of P(D/h) = 1, we have,

if h is consistent with D

is the subset of hypotheses from H that are consistent with D. It’s easier to verify that P(D) = | | / | H | above. Since the add up over all the hypotheses of P(h/D) must be one and because the number of hypotheses from H consistent with D is by definition |  |.

 Alternatively, we can derive P(D) from the theorem of total probability and the fact that the hypothesis is mutually exclusive.

To summarize, Bayes theorem implies that the posterior probability p(h/D) under the assumed P(h) and P(D/h) is

where | VSH,D | is the number of hypotheses from H consistent with D.

## Bayes Optimal Classifier and Naive Bayes Classifier

The Bayes Optimal Classifier is a probabilistic model that predicts the most likely outcome for a new situation.

The Bayes theorem is a method for calculating a hypothesis’s probability based on its prior probability, the probabilities of observing specific data given the hypothesis, and the seen data itself.

**BAYES OPTIMAL CLASSIFIER**

The Bayes Theorem, which provides a systematic means of computing a conditional probability, is used to describe it. It’s also related to Maximum a Posteriori (MAP), a probabilistic framework for determining the most likely hypothesis for a training dataset.

Take a hypothesis space that has 3 hypotheses , , and .

The posterior probabilities of the hypotheses are as follows:

🡪0.4

🡪0.3

🡪 0.3

Hence, is the MAP hypothesis. (MAP => max posterior)

Suppose a new instance x is encountered, which is classified negative by and but positive by .

Taking all hypotheses into account, the probability that x is positive is 0.4 and the probability that it is negative is therefore 0.6.

The classification generated by the MAP hypothesis is different from the most probable classification in this case which is negative.

**The most probable classification of the new instance is obtained by combining the predictions of all hypotheses, weighted by their posterior probabilities.**

If the new example’s probable classification can be any value vj from a set V, the probability P(/D) that the right classification for the new instance is is merely

The denominator is omitted since we’re only using this for comparison and all the values of will have the same denominator.

The value, for which is maximum, is the best classification for the new instance.

Baye’s Optimal Classification:

....(1)

To illustrate in terms of the above example, the set of possible classifications of the new instance is and

**=0.4 =0, =1**

**=0.3 =1, =0**

**=0.3 =1, =0**

A Bayes optimal classifier is a system that classifies new cases according to Equation(1). This strategy increases the likelihood that the new instance will be appropriately classified.

Consider an example for Bayes Optimal Classification,

Let there be 5 hypotheses  through .

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 0.4 | 1 | 0 | 0 |
| 0.2 | 0 | 1 | 0 |
| 0.1 | 0 | 0 | 1 |
| 0.1 | 0 | 1 | 0 |
| 0.2 | 0 | 1 | 0 |

The MAP theory, therefore, argues that the robot should proceed forward (F). Let’s see what the Bayes optimal procedure suggests.

Thus, the Bayes optimal procedure recommends the robot turn left.

**Naives Bayes Classifier**

The Naive Bayes classifiers, which are a set of classification algorithms, are created using the Bayes’ Theorem. ‘Each pair of features categorized is independent of the others. Naive Bayes Classifier is a group of algorithms that all work on the above principle.

The naive Bayes classifier is useful for learning tasks in which each instance x is represented by a set of attribute values and the target function f(x) can take any value from a finite set V.

A set of target function training examples is provided, as well as a new instance specified by the tuple of attribute values

The learner is given the task of estimating the goal value. The most likely target value  is assigned in the Bayesian strategy to classify the new instance.

**........(1)**

Simply count the number of times each target value  appears in the training data to estimate each P().

The naive Bayes classifier is based on the simplifying assumption that the attribute values are conditionally independent given the target value. In other words, the assumption is that given the target value of the instance, the probability of observing the conjunction is just the product of the probabilities for the individual attributes: Substituting this into Equation **(1),** we have the approach used by the naive Bayes classifier.

**Naive Bayes Classifier:** ..........(2)

Where stands for the Naive Bayes classifier’s target value.

The naive Bayes learning approach includes a learning stage in which the different P().and variables are estimated using the training data’s frequency distribution.

The learned hypothesis is represented by the set of these estimations.

The basic Naive Bayes **assumption** is that each feature has the following effect.

Contribution to the ultimate product that is both independent and equal.

Let’s understand the concept of the Naive Bayes classifier better, with the help of an example.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Day** | **Outlook** | **Temperature** | **Humidity** | **Wind** | **Play Tennis** |
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D3 | Overcast | Hot | High | Weak | Yes |
| D4 | Rain | Mild | High | Weak | Yes |
| D5 | Rain | Cool | Normal | Weak | Yes |
| D6 | Rain | Cool | Normal | Strong | No |
| D7 | Overcast | Cool | Normal | Strong | Yes |
| D8 | Sunny | Mild | High | Weak | No |
| D9 | Sunny | Cool | Normal | Weak | Yes |
| D10 | Rain | Mild | Normal | Weak | Yes |
| D11 | Sunny | Mild | Normal | Strong | Yes |
| D12 | Overcast | Mild | High | Strong | Yes |
| D13 | Overcast | Hot | Normal | Weak | Yes |
| D14 | Rain | Mild | High | Strong | No |

Let’s use the naive Bayes classifier to solve a problem we discussed during our decision tree learning discussion: classifying days based on whether or not someone will play tennis.

Above table shows 14 training instances of the goal concept PlayTennis, with the characteristics Outlook, Temperature, Humidity, and Wind describing each day. To categorize the following novel instance, we utilize the naive Bayes classifier and the training data from this table:

<Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong>

Our task is to predict the target value (yes or no) of the target concept

PlayTennis for this new instance. Instantiating Equation (2) to fit the current task, the target value is given by

..........(3)

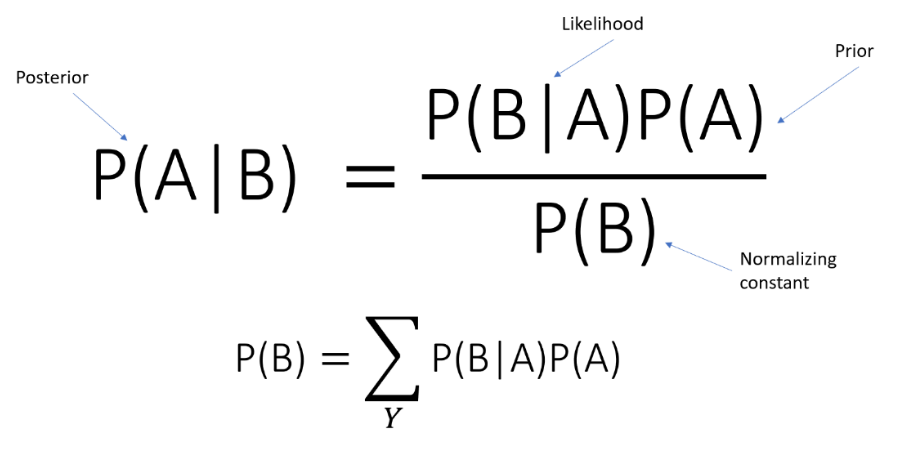
First, based on the frequencies of the 14 training instances, the probability of the various goal values may be easily determined.

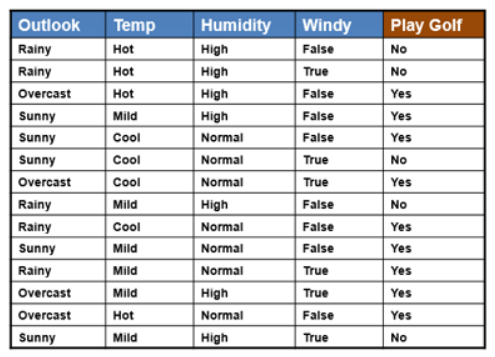
We can also estimate conditional probabilities in the same way. Those for Wind = strong, for example, include

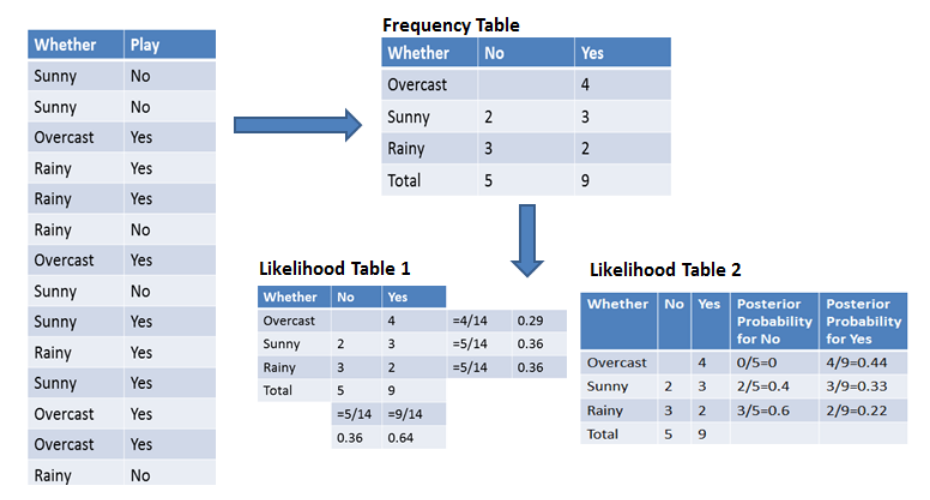
Using these probability estimates and similar estimates for the remaining attribute values, we calculate according to Equation ***(3)*** as follows (now omitting attribute names for brevity)

Based on the probability estimates learned from the training data, the naive Bayes classifier gives the goal value PlayTennis = no to this new occurrence. Furthermore, given the observed attribute values, we can determine the conditional probability that the target value is no by normalizing the above amounts to sum to one.

For the current example, this probability is,







**Bayseian belief network**

Bayesian belief network describes the probability distribution governing a set of variables by specifying a set of conditional independence assumptions along with a set of conditional

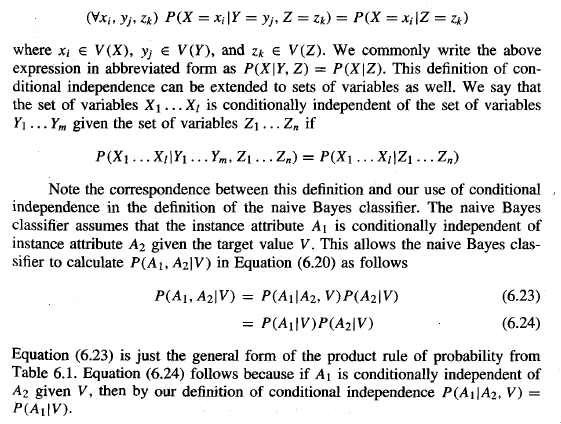
probabilities. In contrast to the naive Bayes classifier, which assumes that ***all*** the variables are conditionally independent given the value

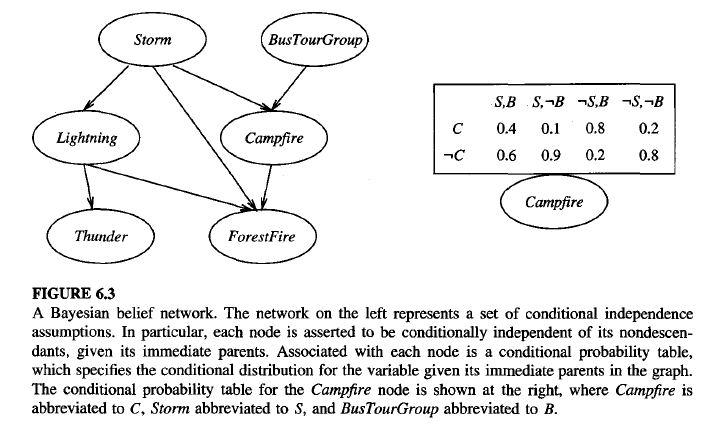
of the target variable, Bayesian belief networks allow stating conditional independence assumptions that apply to ***subsets*** of the variables. Thus, Bayesian belief networks provide an intermediate approach that is less constraining than the global assumption of conditional independence made by the naive Bayes classifier, but more tractable than avoiding conditional independence assumptions altogether.

In general, a Bayesian belief network describes the probability distribution over a set of variables. Consider an arbitrary set of random variables where each variable can take on the set of possible values . We define the ***joint space*** of the set of variables Y to be the cross product . In other words, each item in the joint space corresponds to one of the possible assignments of values to the tuple of variables .) The probability distribution over this joint' space is called the ***joint probability distribution.*** The joint probability distribution specifies the probability for each of the possible variable bindings for the tuple . A Bayesian belief network describes the joint probability distribution for a set of variables.

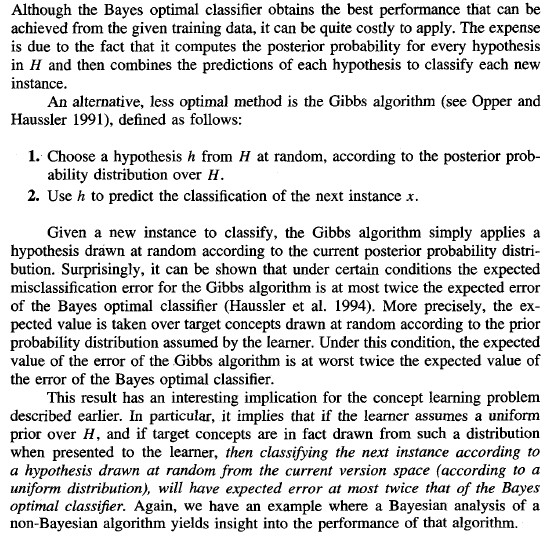
* **Conditional Independence**

Let us begin our discussion of Bayesian belief networks by defining precisely the notion of conditional independence. Let X, Y, and Z be three discrete-valued random variables. We say that X is ***conditionally independent*** of Y given Z if the probability distribution governing X is independent of the value of Y given a value for 2; that is, if





**GIBBS ALGORITHM**

****