MODULE 4

**ARTIFICIAL NEURAL NETWORKS**

# INTRODUCTION

Artificial neural networks (ANNs) provide a general, practical method for learning real-valued, discrete-valued, and vector-valued target functions.

## Biological Motivation

* The study of artificial neural networks (ANNs) has been inspired by the observation that biological learning systems are built of very complex webs of interconnected ***Neurons***
* Human information processing system consists of brain ***neuron***: basic building block cell that communicates information to and from various parts of body

## Facts of Human Neurobiology

* Number of neurons ~ 1011
* Connection per neuron ~ 10 4 – 5
* Neuron switching time ~ 0.001 second or 10 -3
* Scene recognition time ~ 0.1 second
* 100 inference steps doesn’t seem like enough
* Highly parallel computation based on distributed representation

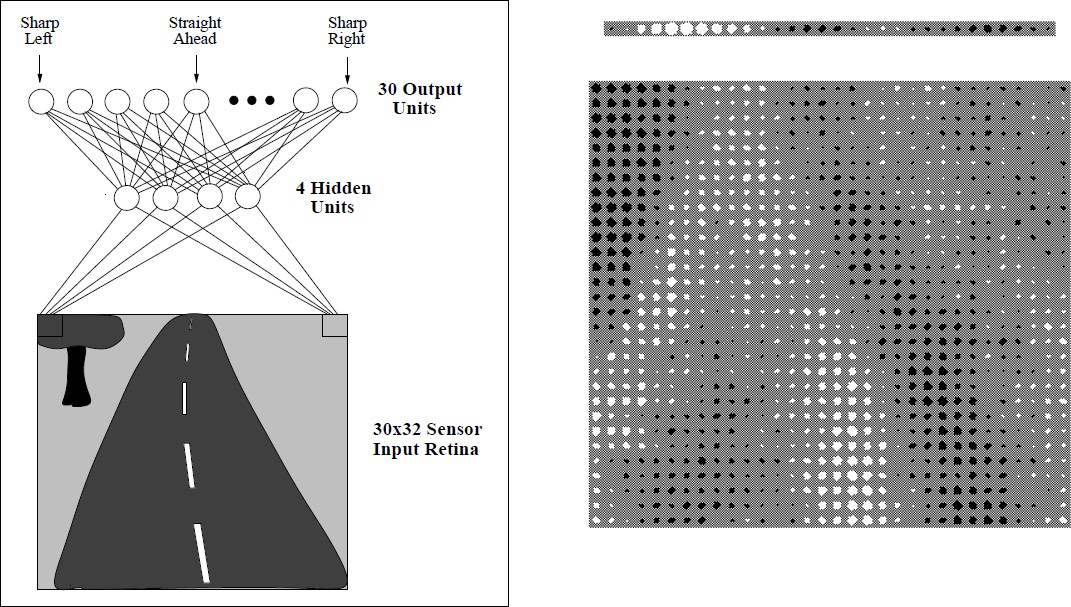
## Properties of Neural Networks

* Many neuron-like threshold switching units
* Many weighted interconnections among units
* Highly parallel, distributed process
* Emphasis on tuning weights automatically
* Input is a high-dimensional discrete or real-valued (e.g, sensor input )

# NEURAL NETWORK REPRESENTATIONS

* A prototypical example of ANN learning is provided by Pomerleau's system ALVINN, which uses a learned ANN to steer an autonomous vehicle driving at normal speeds on public highways
* The input to the neural network is a 30x32 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





**Figure:** Neural network learning to steer an autonomous vehicle.

* Figure illustrates the neural network representation.
* The network is shown on the left side of the figure, with the input camera image depicted below it.
* Each node (i.e., circle) in the network diagram corresponds to the output of a single network unit, and the lines entering the node from below are its inputs.
* There are four units that receive inputs directly from all of the 30 x 32 pixels in the image. These are called "hidden" units because their output is available only within the network and is not available as part of the global network output. Each of these four hidden units computes a single real-valued output based on a weighted combination of its 960 inputs
* These 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 diagrams on the right side of the figure depict the learned weight values associated with one of the four hidden units in this ANN.
* The large matrix of black and white boxes on the lower right depicts the weights from the 30 x 32 pixel inputs into the hidden unit. Here, a white box indicates a positive weight, a black box a negative weight, and the size of the box indicates the weight magnitude.
* The smaller rectangular diagram directly above the large matrix shows the weights from this hidden unit to each of the 30 output units.

# APPROPRIATE PROBLEMS FOR NEURAL NETWORK LEARNING

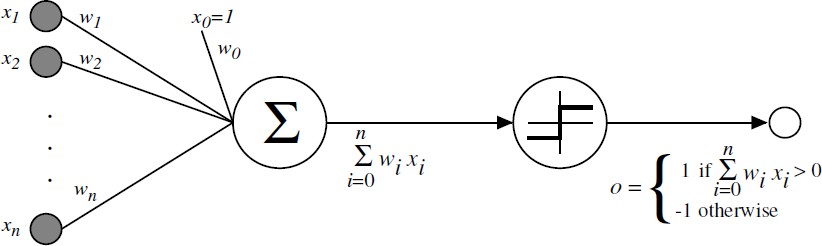
ANN learning is well-suited to problems in which the training data corresponds to noisy, complex sensor data, such as inputs from cameras and microphones.

ANN is appropriate for problems with the following characteristics:

1. Instances are represented by many attribute-value pairs.
2. The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.
3. The training examples may contain errors.
4. Long training times are acceptable.
5. Fast evaluation of the learned target function may be required
6. The ability of humans to understand the learned target function is not important

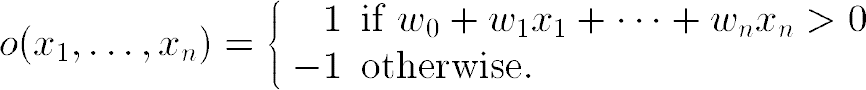
# PERCEPTRON

* One type of ANN system is based on a unit called a perceptron. Perceptron is a single layer neural network.



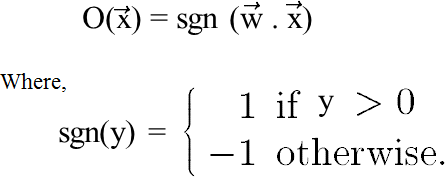
**Figure:** A perceptron

* A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise.
* Given inputs ***x*** through ***x,*** the output **O*(x1,*** . . . , ***xn)*** computed by the perceptron is



* Where, each wi is a real-valued constant, or weight, that determines the contribution of input xi to the perceptron output.
* -w0 is a threshold that the weighted combination of inputs w1x1 + . . . + wnxn must surpass in order for the perceptron to output a 1.

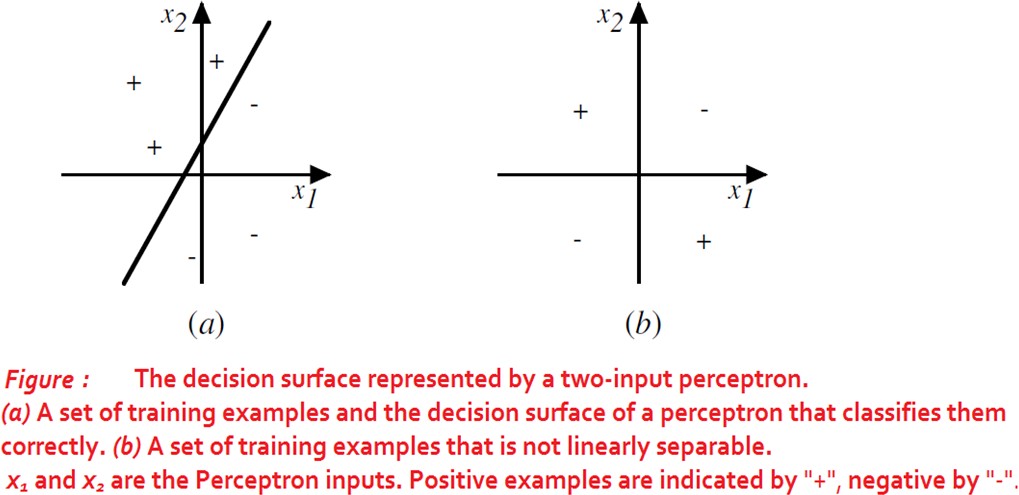
Sometimes, the perceptron function is written as,



Learning a perceptron involves choosing values for the weights w0 , . . . , wn . Therefore, the space H of candidate hypotheses considered in perceptron learning is the set of all possible real-valued weight vectors

## Representational Power of Perceptrons

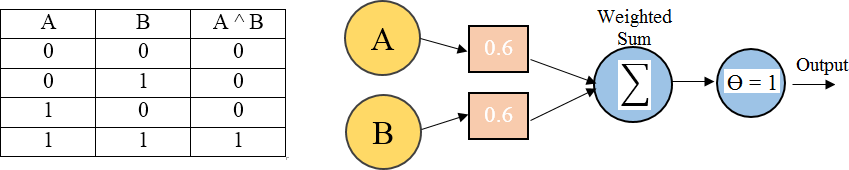
* The perceptron can be viewed as representing a hyperplane decision surface in the n- dimensional space of instances (i.e., points)
* The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side, as illustrated in below figure



Perceptrons can represent all of the primitive Boolean functions AND, OR, NAND (~ AND), and NOR (~OR)

Some Boolean functions cannot be represented by a single perceptron, such as the XOR function whose value is 1 if and only if x1 ≠ x2

**Example: Representation of AND functions**



If A=0 & B=0 → 0\*0.6 + 0\*0.6 = 0.

This is not greater than the threshold of 1, so the output = 0.

If A=0 & B=1 → 0\*0.6 + 1\*0.6 = 0.6.

This is not greater than the threshold, so the output = 0.

If A=1 & B=0 → 1\*0.6 + 0\*0.6 = 0.6.

This is not greater than the threshold, so the output = 0.

If A=1 & B=1 → 1\*0.6 + 1\*0.6 = 1.2.

This exceeds the threshold, so the output = 1.

Drawback of perceptron

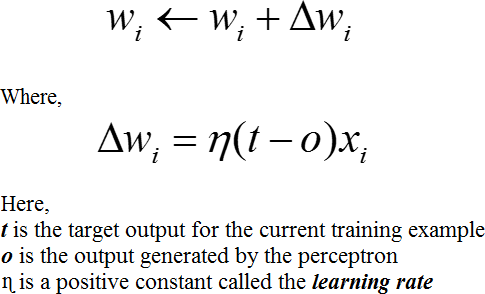
* The perceptron rule finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the examples are not linearly separable

## The Perceptron Training Rule

The learning problem is to determine a weight vector that causes the perceptron to produce the correct + 1 or - 1 output for each of the given training examples.

To learn an acceptable weight vector

* Begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example.
* This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly.
* Weights are modified at each step according to the perceptron training rule, which revises the weight wi associated with input xi according to the rule.



* The role of the learning rate is to moderate the degree to which weights are changed at each step. It 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

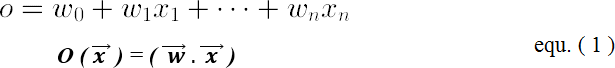
Drawback:

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

## Gradient Descent and the Delta Rule

* If the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept.
* The key idea behind the delta rule is to use ***gradient descent*** to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.

To understand the delta training rule, consider the task of training an unthresholded perceptron. That is, a linear unit for which the output ***O*** is given by



To derive a weight learning rule for linear units, specify a measure for the ***training error*** of a hypothesis (weight vector), relative to the training examples.

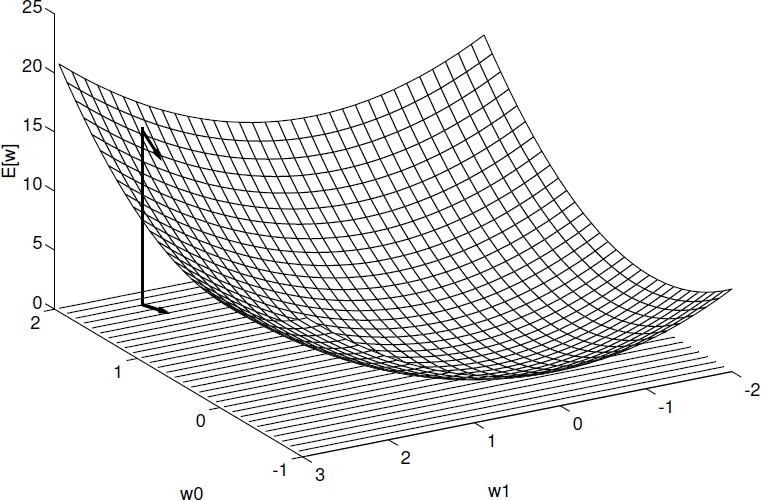


Where,

* D is the set of training examples,
* td is the target output for training example d,
* od is the output of the linear unit for training example d
* E ( w⃗⃗⃗→ ) is simply half the squared difference between the target output td and the linear unit output od, summed over all training examples.

# Visualizing the Hypothesis Space

* To understand the gradient descent algorithm, it is helpful to visualize the entire hypothesis space of possible weight vectors and their associated E values as shown in below figure.
* Here the axes w0 and wl represent possible values for the two weights of a simple linear unit. The w0, wl plane therefore represents the entire hypothesis space.
* The vertical axis indicates the error E relative to some fixed set of training examples.
* The arrow shows the negated gradient at one particular point, indicating the direction in the w0, wl plane producing steepest descent along the error surface.
* The error surface shown in the figure thus summarizes the desirability of every weight vector in the hypothesis space



* Given the way in which we chose to define E, for linear units this error surface must always be parabolic with a single global minimum.

Gradient descent search determines a weight vector that minimizes E by starting with an arbitrary initial weight vector, then repeatedly modifying it in small steps.

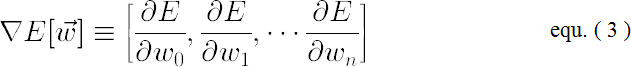
At each step, the weight vector is altered in the direction that produces the steepest descent along the error surface depicted in above figure. This process continues until the global minimum error is reached.

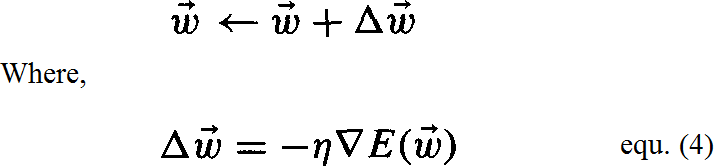
# Derivation of the Gradient Descent Rule

### How to calculate the direction of steepest descent along the error surface?

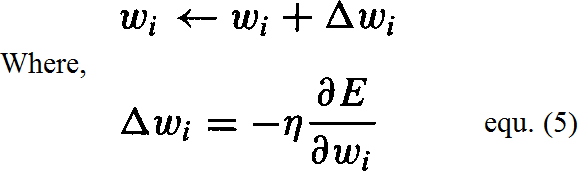
The direction of steepest can be found by computing the derivative of E with respect to each component of the vector w⃗⃗⃗→ . This vector derivative is called the gradient of E with respect to

⃗w⃗⃗→ , written as



The gradient specifies the direction of steepest increase of E, the training rule for gradient descent is

* Here η is a positive constant called the learning rate, which determines the step size in the gradient descent search.
* The negative sign is present because we want to move the weight vector in the direction that decreases E.

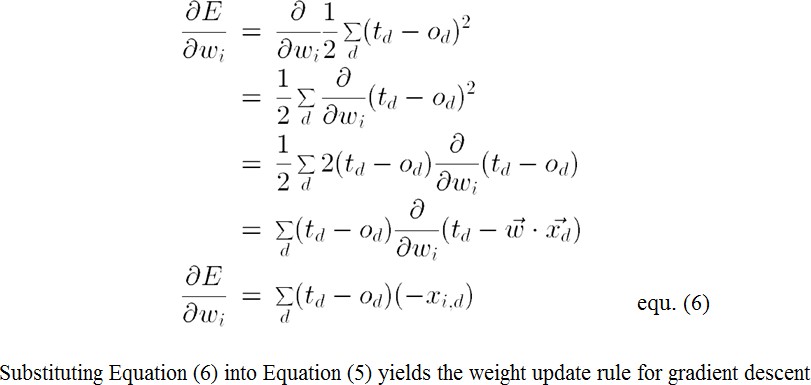
This training rule can also be written in its component form

Calculate the gradient at each step. The vector of 𝜕𝐸

𝜕𝑤𝑖

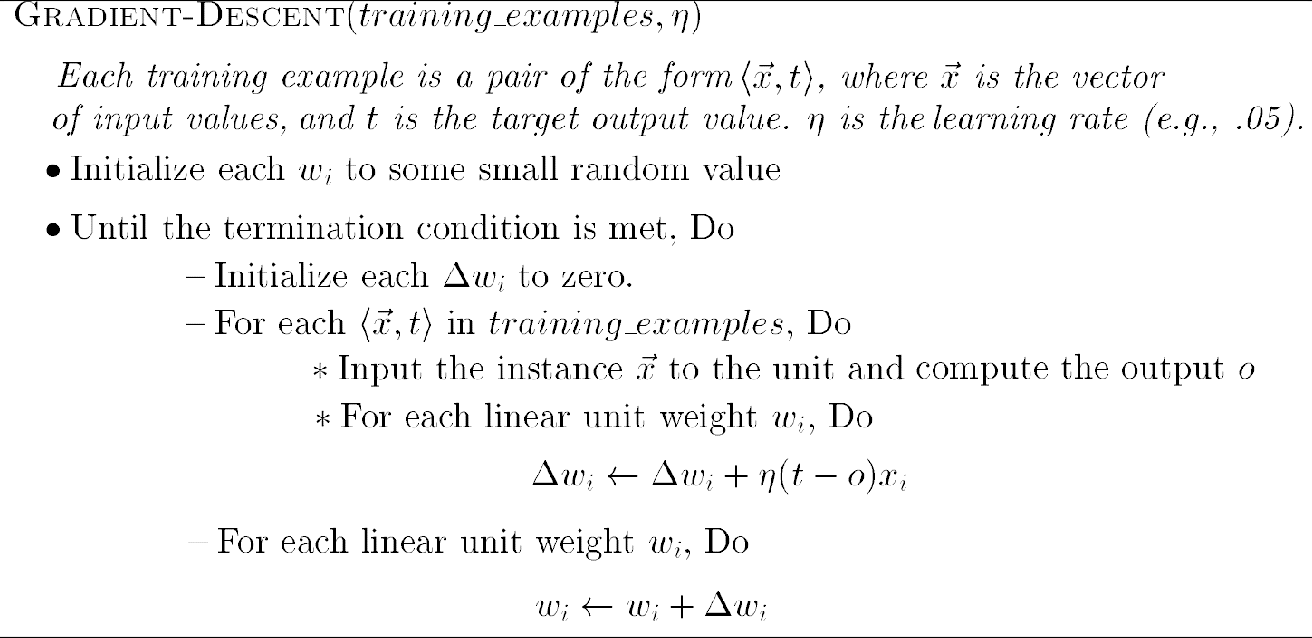
derivatives that form the

gradient can be obtained by differentiating E from Equation (2), as





# GRADIENT DESCENT algorithm for training a linear unit



To summarize, the gradient descent algorithm for training linear units is as follows:

* Pick an initial random weight vector.
* Apply the linear unit to all training examples, then compute Δwi for each weight according to Equation (7).
* Update each weight wi by adding Δwi, then repeat this process

## Issues in Gradient Descent Algorithm

Gradient descent is an important general paradigm for learning. It is a strategy for searching through a large or infinite hypothesis space that can be applied whenever

1. The hypothesis space contains continuously parameterized hypotheses
2. The error can be differentiated with respect to these hypothesis parameters

The key practical difficulties in applying gradient descent are

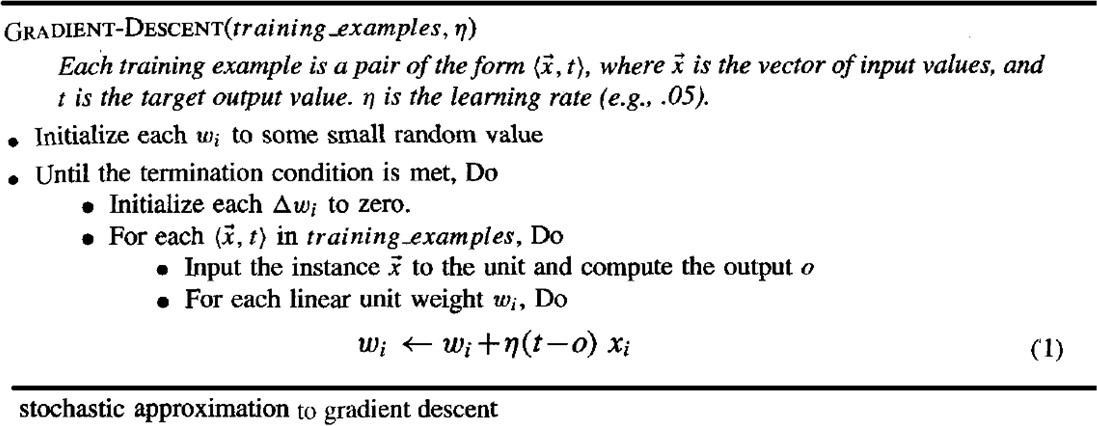
1. Converging to a local minimum can sometimes be quite slow
2. If there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minimum

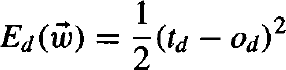
## Stochastic Approximation to Gradient Descent

* The gradient descent training rule presented in Equation (7) computes weight updates after summing over all the training examples in D
* The idea behind stochastic gradient descent is to approximate this gradient descent search by updating weights incrementally, following the calculation of the error for each individual example

∆wi = η (t – o) xi

* where t, o, and xi are the target value, unit output, and ith input for the training example in question



One way to view this stochastic gradient descent is to consider a distinct error function Ed( ⃗w⃗⃗→ ) for each individual training example d as follows

* Where, td and od are the target value and the unit output value for training example d.
* Stochastic gradient descent iterates over the training examples d in D, at each iteration altering the weights according to the gradient with respect to Ed( w⃗⃗⃗→ )
* The sequence of these weight updates, when iterated over all training examples, provides a reasonable approximation to descending the gradient with respect to our original error function Ed( w⃗⃗⃗→ )
* By making the value of η sufficiently small, stochastic gradient descent can be made to approximate true gradient descent arbitrarily closely

*The key differences between standard gradient descent and stochastic gradient descent are*

* 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.
* Summing over multiple examples in standard gradient descent requires more computation per weight update step. On the other hand, because it uses the true gradient, standard gradient descent is often used with a larger step size per weight update than stochastic gradient descent.
* In cases where there are multiple local minima with respect to stochastic gradient descent can sometimes avoid falling into these local minima because it uses the various

∇E ( ⃗w⃗⃗→ ) rather than ∇ E( w⃗⃗⃗→ ) to guide its search

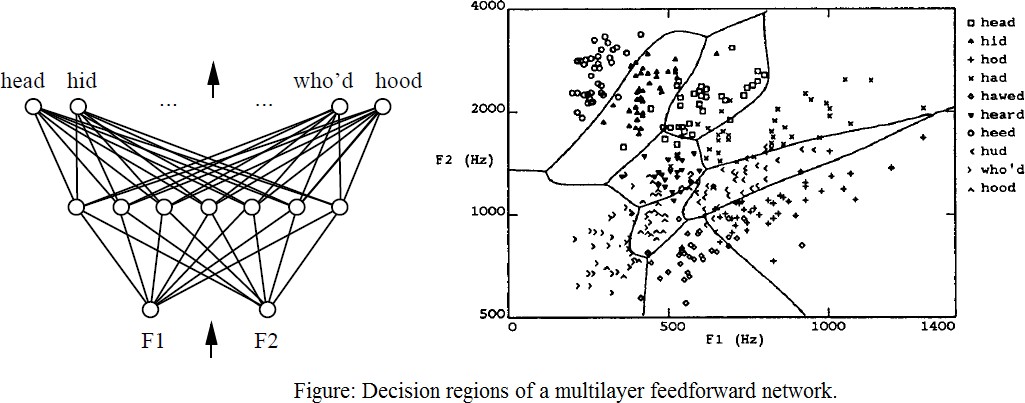
d

# MULTILAYER NETWORKS AND THE BACKPROPAGATION ALGORITHM

Multilayer networks learned by the BACKPROPAGATION algorithm are capable of expressing a rich variety of nonlinear decision surfaces.

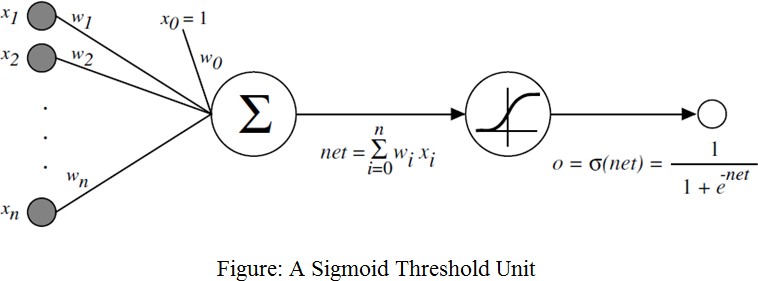
## Consider the example:

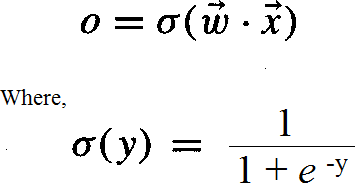
* Here the speech recognition task involves distinguishing among 10 possible vowels, all spoken in the context of "h\_d" (i.e., "hid," "had," "head," "hood," etc.).
* The network input consists of two parameters, F1 and F2, obtained from a spectral analysis of the sound. The 10 network outputs correspond to the 10 possible vowel sounds. The network prediction is the output whose value is highest.
* The plot on the right illustrates the highly nonlinear decision surface represented by the learned network. Points shown on the plot are test examples distinct from the examples used to train the network.



## A Differentiable Threshold Unit (Sigmoid unit)

* Sigmoid unit-a unit very much like a perceptron, but based on a smoothed, differentiable threshold function.

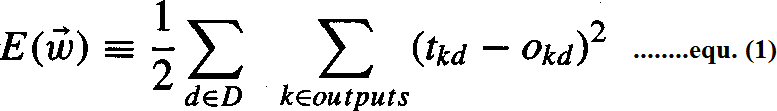


* The sigmoid unit first computes a linear combination of its inputs, then applies a threshold to the result and the threshold output is a continuous function of its input.
* More precisely, the sigmoid unit computes its output O as

σ is the sigmoid function

## The BACKPROPAGATION Algorithm

* The BACKPROPAGATION Algorithm learns the weights for a multilayer network, given a network with a fixed set of units and interconnections. It employs gradient descent to attempt to minimize the squared error between the network output values and the target values for these outputs.
* In BACKPROPAGATION algorithm, we consider networks with multiple output units rather than single units as before, so we redefine E to sum the errors over all of the network output units.



where,

* ***outputs*** - is the set of output units in the network
* ***tkd*** and ***Okd*** - the target and output values associated with the ***kth*** output unit
* ***d*** - training example

### Algorithm:

BACKPROPAGATION (*training\_example, ƞ, nin, nout, nhidden )*

*Each training example is a pair of the form (*𝑥⃗ →, 𝑡→ *), where (*𝑥→ *) is the vector of network*

*input values, (*𝑡→ *) and is the vector of target network output values.*

*ƞ is the learning rate (e.g., .05). ni, is the number of network inputs, nhidden the number of units in the hidden layer, and nout the number of output units.*

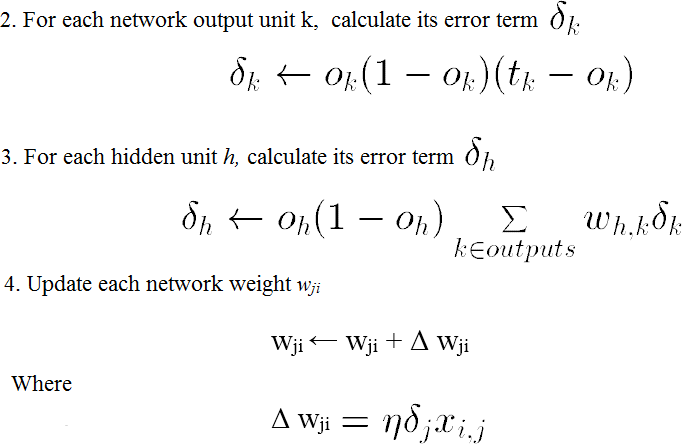
*The input from unit i into unit j is denoted xji, and the weight from unit i to unit j is denoted wji*

* Create a feed-forward network with ni inputs, nhidden hidden units, and nout output units.
* Initialize all network weights to small random numbers
* Until the termination condition is met, Do
  + For each (⃗𝑥→,

𝑡→ ), in training examples, Do

*Propagate the input forward through the network:*

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

*Propagate the errors backward through the network:*

**Adding Momentum**

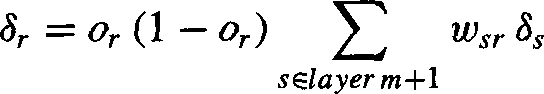
Because BACKPROPAGATION is such a widely used algorithm, many variations have been developed. The most common is to alter the weight-update rule the equation below

by making the weight update on the nth iteration depend partially on the update that occurred during the (n - 1)th iteration, as follows:



**Learning in arbitrary acyclic networks**

* BACKPROPAGATION algorithm given there easily generalizes to feedforward networks of arbitrary depth. The weight update rule is retained, and the only change is to the procedure for computing δ values.
* In general, the ***δ***, value for a unit ***r*** in layer ***m*** is computed from the ***δ*** values at the next deeper layer m + 1 according to



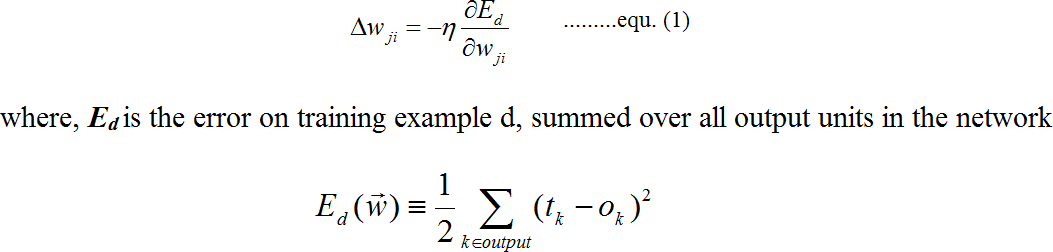
* The rule for calculating ***δ*** for any internal unit



Where, Downstream(r) is the set of units immediately downstream from unit ***r*** in the network: that is, all units whose inputs include the output of unit ***r***

**Derivation of the BACKPROPAGATION Rule**

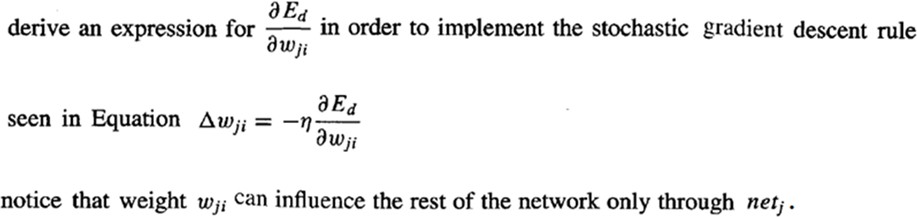
* Deriving the stochastic gradient descent rule: Stochastic gradient descent involves iterating through the training examples one at a time, for each training example d descending the gradient of the error Ed with respect to this single example
* For each training example d every weight wji is updated by adding to it Δwji

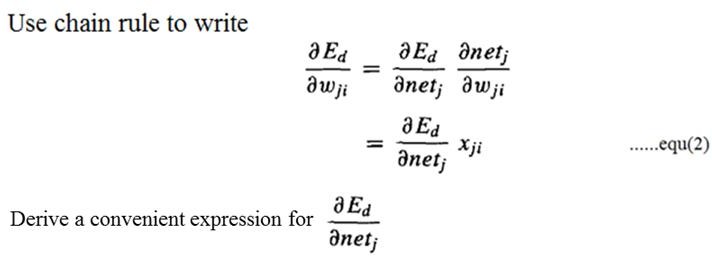


Here outputs is the set of output units in the network, ***tk*** is the target value of unit ***k*** for training example ***d***, and ***ok*** is the output of unit ***k*** given training example ***d***.

The derivation of the stochastic gradient descent rule is conceptually straightforward, but requires keeping track of a number of subscripts and variables

* xji = the ith input to unit j
* wji = the weight associated with the ith input to unit j
* netj = Σi wjixji (the weighted sum of inputs for unit j )
* oj = the output computed by unit j
* tj = the 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

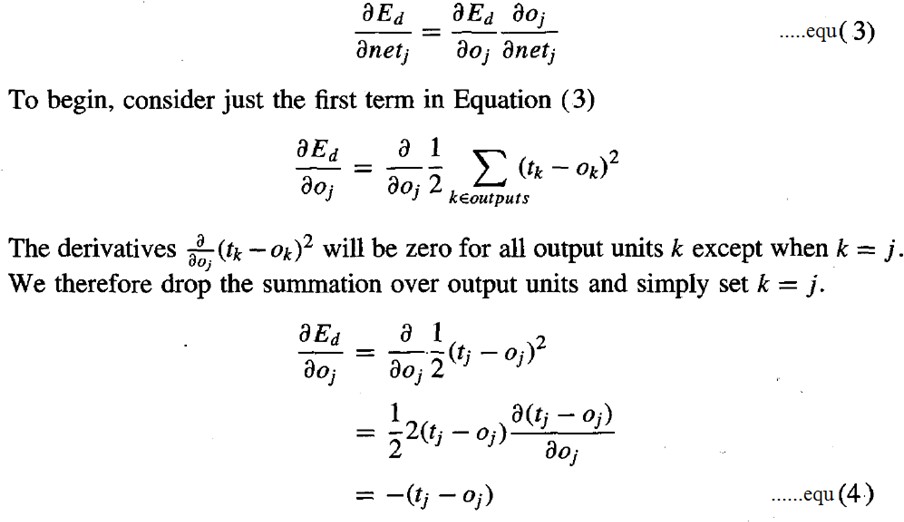


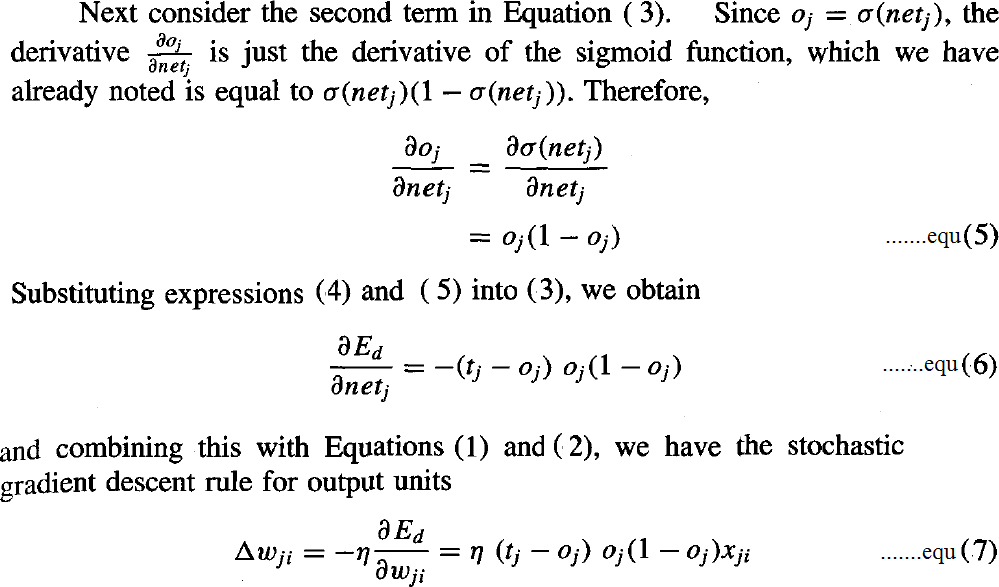


**Consider two cases:** The case where unit j is an output unit for the network, and the case where j is an internal unit (hidden unit).

## Case 1: Training Rule for Output Unit Weights.

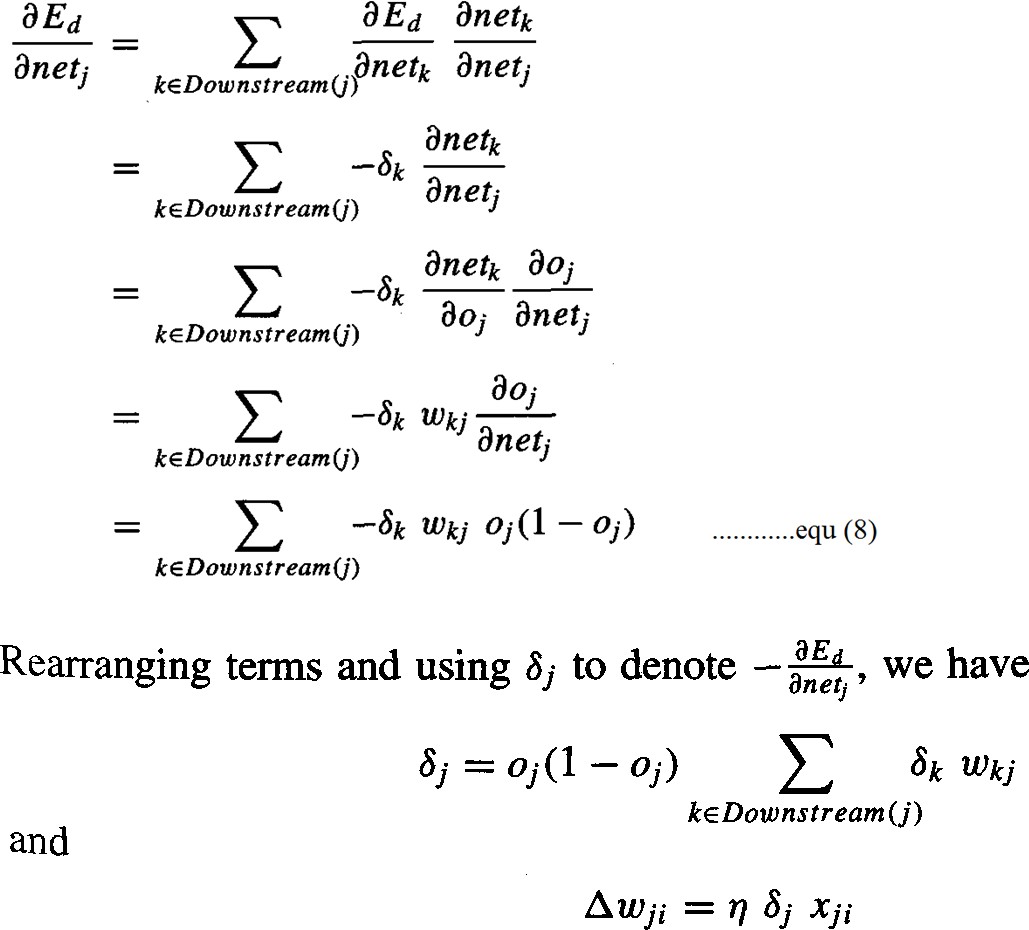
wji can influence the rest of the network only through netj , netj can influence the network only through oj. Therefore, we can invoke the chain rule again to write





## Case 2: Training Rule for Hidden Unit Weights.

* In the case where j is an internal, or hidden unit in the network, the derivation of the training rule for wji must take into account the indirect ways in which wji can influence the network outputs and hence Ed.
* For this reason, we will find it useful to refer to the set of all units immediately downstream of unit j in the network and denoted this set of units by Downstream( j).
* netj can influence the network outputs only through the units in Downstream(j). Therefore, we can write



# REMARKS ON THE BACKPROPAGATION ALGORITHM

## Convergence and Local Minima

* + The BACKPROPAGATION multilayer networks is only guaranteed to converge toward some local minimum in E and not necessarily to the global minimum error.
  + Despite the lack of assured convergence to the global minimum error, BACKPROPAGATION is a highly effective function approximation method in practice.
  + Local minima can be gained by considering the manner in which network weights evolve as the number of training iterations increases.

Common heuristics to attempt to alleviate the problem of local minima include:

1. Add a momentum term to the weight-update rule. Momentum can sometimes carry the gradient descent procedure through narrow local minima
2. Use stochastic gradient descent rather than true gradient descent
3. Train multiple networks using the same data, but initializing each network with different random weights

## Representational Power of Feedforward Networks

*What set of functions can be represented by feed-forward networks?*

The answer depends on the width and depth of the networks. There are three quite general results are known about which function classes can be described by which types of Networks

* 1. Boolean functions – Every boolean function can be represented exactly by some network with two layers of units, although the number of hidden units required grows exponentially in the worst case with the number of network inputs
  2. Continuous functions – Every bounded continuous function can be approximated with arbitrarily small error by a network with two layers of units
  3. Arbitrary functions – Any function can be approximated to arbitrary accuracy by a network with three layers of units.

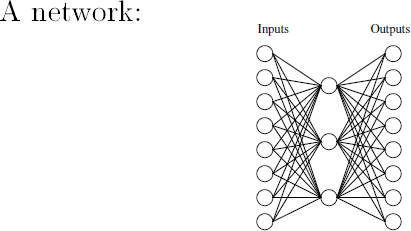
## Hypothesis Space Search and Inductive Bias

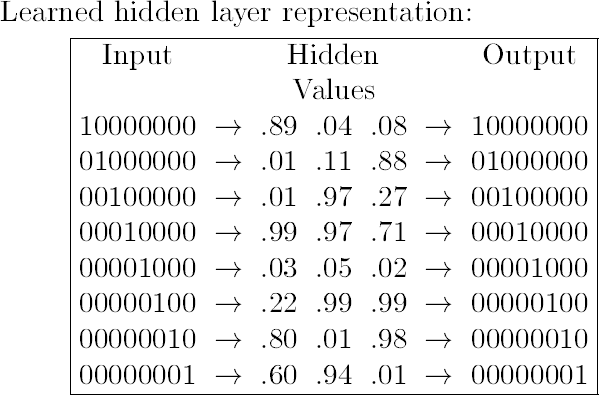
* Hypothesis space is the n-dimensional Euclidean space of the ***n*** network weights and hypothesis space is continuous.
* As it is continuous, E is differentiable with respect to the continuous parameters of the hypothesis, results in a well-defined error gradient that provides a very useful structure for organizing the search for the best hypothesis.
* It is difficult to characterize precisely the inductive bias of BACKPROPAGATION algorithm, because it depends on the interplay between the gradient descent search and the way in which the weight space spans the space of representable functions. However, one can roughly characterize it as smooth interpolation between data points.

## Hidden Layer Representations

BACKPROPAGATION can define new hidden layer features that are not explicit in the input representation, but which capture properties of the input instances that are most relevant to learning the target function.

Consider example, the network shown in below Figure





* Consider training the network shown in Figure to learn the simple target function f (x)

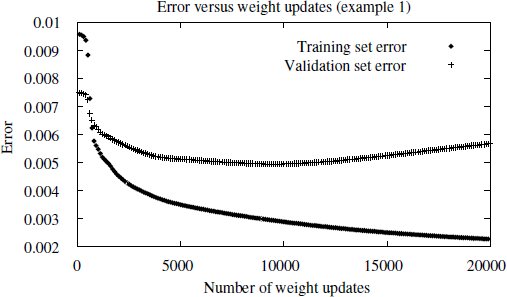
= x, where x is a vector containing seven 0's and a single 1.

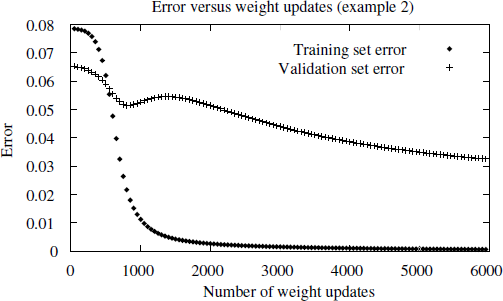
* The network must learn to reproduce the eight inputs at the corresponding eight output units. Although this is a simple function, the network in this case is constrained to use only three hidden units. Therefore, the essential information from all eight input units must be captured by the three learned hidden units.
* When BACKPROPAGATION applied to this task, using each of the eight possible vectors as training examples, it successfully learns the target function. By examining the hidden unit values generated by the learned network for each of the eight possible input vectors, it is easy to see that the learned encoding is similar to the familiar standard binary encoding of eight values using three bits (e.g., 000,001,010,. . . , 111). The exact values of the hidden units for one typical run of shown in Figure.
* This ability of multilayer networks to automatically discover useful representations at the hidden layers is a key feature of ANN learning

## Generalization, Overfitting, and Stopping Criterion

What is an appropriate condition for terminating the weight update loop? One choice is to continue training until the error E on the training examples falls below some predetermined threshold.

To see the dangers of minimizing the error over the training data, consider how the error E varies with the number of weight iterations





* Consider first the top plot in this figure. The lower of the two lines shows the monotonically decreasing error E over the training set, as the number of gradient descent iterations grows. The upper line shows the error E measured over a different validation set of examples, distinct from the training examples. This line measures the generalization accuracy of the network-the accuracy with which it fits examples beyond the training data.
* The generalization accuracy measured over the validation examples first decreases, then increases, even as the error over the training examples continues to decrease. How can this occur? This occurs because the weights are being tuned to fit idiosyncrasies of the training examples that are not representative of the general distribution of examples. The large number of weight parameters in ANNs provides many degrees of freedom for fitting such idiosyncrasies
* Why does overfitting tend to occur during later iterations, but not during earlier iterations?

By giving enough weight-tuning iterations, BACKPROPAGATION will often be able to create overly complex decision surfaces that fit noise in the training data or unrepresentative characteristics of the particular training sample.

**BAYESIAN LEARNING**

Bayesian reasoning provides a probabilistic approach to inference. It is based on the assumption that the quantities of interest are governed by probability distributions and that optimal decisions can be made by reasoning about these probabilities together with observed data

# INTRODUCTION

Bayesian learning methods are relevant to study of machine learning for two different reasons.

1. First, Bayesian learning algorithms that calculate explicit probabilities for hypotheses, such as the naive Bayes classifier, are among the most practical approaches to certain types of learning problems
2. The second reason is that they provide a useful perspective for understanding many learning algorithms that do not explicitly manipulate probabilities.

# Features of Bayesian Learning Methods

* Each observed training example can incrementally decrease or increase the estimated probability that a hypothesis is correct. This provides a more flexible approach to learning than algorithms that completely eliminate a hypothesis if it is found to be inconsistent with any single example
* Prior knowledge can be combined with observed data to determine the final probability of a hypothesis. In Bayesian learning, prior knowledge is provided by asserting (1) a prior probability for each candidate hypothesis, and (2) a probability distribution over observed data for each possible hypothesis.
* Bayesian methods can accommodate hypotheses that make probabilistic predictions
* New instances can be classified by combining the predictions of multiple hypotheses, weighted by their probabilities.
* Even in cases where Bayesian methods prove computationally intractable, they can provide a standard of optimal decision making against which other practical methods can be measured.

# Practical difficulty in applying Bayesian methods

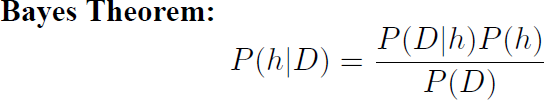
1. One practical difficulty in applying Bayesian methods is that they typically require initial knowledge of many probabilities. When these probabilities are not known in advance they are often estimated based on background knowledge, previously available data, and assumptions about the form of the underlying distributions.
2. A second practical difficulty is the significant computational cost required to determine the Bayes optimal hypothesis in the general case. In certain specialized situations, this computational cost can be significantly reduced.

# BAYES THEOREM

Bayes theorem provides a way to calculate the probability of a hypothesis based on its prior probability, the probabilities of observing various data given the hypothesis, and the observed data itself.

Notations

* P(h) prior probability of h, reflects any background knowledge about the chance that h is correct
* P(D) prior probability of D, probability that D will be observed
* P(D|h) probability of observing D given a world in which h holds
* P(h|D) posterior probability of h, reflects confidence that h holds after D has been observed

Bayes theorem is the cornerstone of Bayesian learning methods because it provides a way to calculate the posterior probability P(h|D), from the prior probability P(h), together with P(D) and P(D|h).

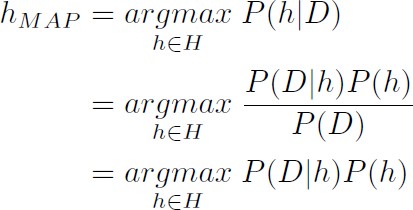
* P(h|D) increases with P(h) and with P(D|h) according to Bayes theorem.
* P(h|D) decreases as P(D) increases, because the more probable it is that D will be observed independent of h, the less evidence D provides in support of h.

## Maximum a Posteriori (MAP) Hypothesis

* In many learning scenarios, the learner considers some set of candidate hypotheses H and is interested in finding the most probable hypothesis h ∈ H given the observed data

D. Any such maximally probable hypothesis is called a maximum a posteriori (MAP) hypothesis.

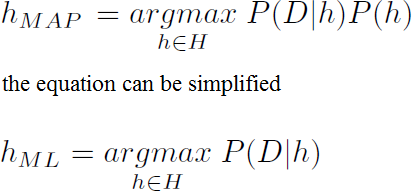
* Bayes theorem to calculate the posterior probability of each candidate hypothesis is hMAP is a MAP hypothesis provided



* P(D) can be dropped, because it is a constant independent of h

## Maximum Likelihood (ML) Hypothesis

* In some cases, it is assumed that every hypothesis in H is equally probable a priori (P(hi) = P(hj) for all hi and hj in H).
* In this case the below equation can be simplified and need only consider the term P(D|h) to find the most probable hypothesis.



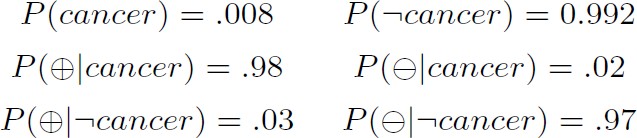
P(D|h) is often called the likelihood of the data D given h, and any hypothesis that maximizes P(D|h) is called a maximum likelihood (ML) hypothesis

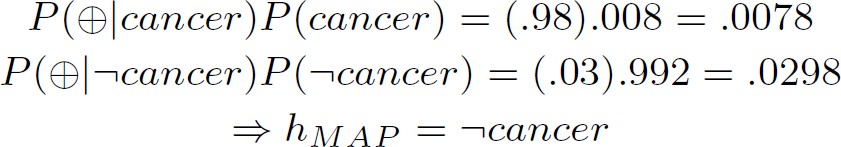
## Example

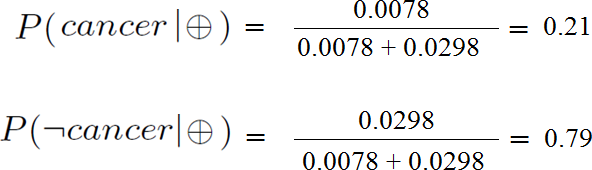
* Consider a medical diagnosis problem in which there are two alternative hypotheses:

(1) that the patient has particular form of cancer, and (2) that the patient does not. The available data is from a particular laboratory test with two possible outcomes: + (positive) and - (negative).

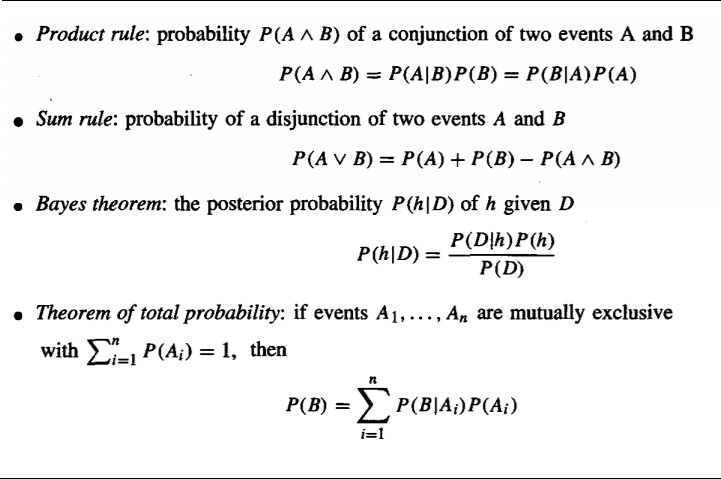
* We have prior knowledge that over the entire population of people only .008 have this disease. Furthermore, the lab test is only an imperfect indicator of the disease.
* The test returns a correct positive result in only 98% of the cases in which the disease is actually present and a correct negative result in only 97% of the cases in which the disease is not present. In other cases, the test returns the opposite result.
* The above situation can be summarized by the following probabilities:



Suppose a new patient is observed for whom the lab test returns a positive (+) result. Should we diagnose the patient as having cancer or not?

The exact posterior probabilities can also be determined by normalizing the above quantities so that they sum to 1

Basic formulas for calculating probabilities are summarized in Table



# BAYES THEOREM AND CONCEPT LEARNING

*What is the relationship between Bayes theorem and the problem of concept learning?*

Since Bayes theorem provides a principled way to calculate the posterior probability of each hypothesis given the training data, and can use it as the basis for a straightforward learning algorithm that calculates the probability for each possible hypothesis, then outputs the most probable.

# Brute-Force Bayes Concept Learning

Consider the concept learning problem

* Assume the learner considers some finite hypothesis space H defined over the instance space X, in which the task is to learn some target concept c : X → {0,1}.
* Learner is given some sequence of training examples ((x1, d1) . . . (xm, dm)) where xi is some instance from X and where di is the target value of xi (i.e., di = c(xi)).
* The sequence of target values are written as D = (d1 . . . dm).

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

**BRUTE-FORCE MAP LEARNING algorithm:**

1. For each hypothesis h in H, calculate the posterior probability



1. Output the hypothesis hMAP with the highest posterior probability

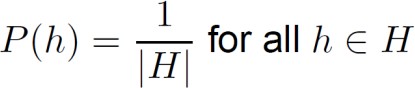


In order specify a learning problem for the BRUTE-FORCE MAP LEARNING algorithm we must specify what values are to be used for P(h) and for P(D|h) ?

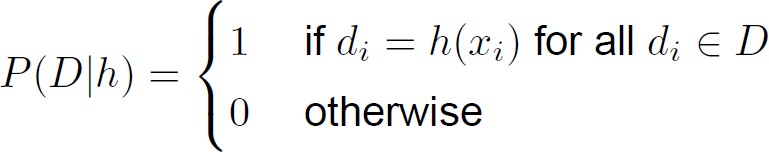
Let’s choose P(h) and for P(D|h) to be consistent with the following assumptions:

* The training data D is noise free (i.e., di = c(xi))
* The target concept c is contained in the hypothesis space H
* Do not have a priori reason to believe that any hypothesis is more probable than any other.

*What values should we specify for P(h)?*

* Given no prior knowledge that one hypothesis is more likely than another, it is reasonable to assign the same prior probability to every hypothesis h in H.
* Assume the target concept is contained in H and require that these prior probabilities sum to 1.

*What choice shall we make for P(D|h)?*

* P(D|h) is the probability of observing the target values D = (d1 . . .dm) for the fixed set of instances (x1 . . . xm), given a world in which hypothesis h holds
* Since we assume noise-free training data, the probability of observing classification di given h is just 1 if di = h(xi) and 0 if di ≠ h(xi). Therefore,

Given these choices for P(h) and for P(D|h) we now have a fully-defined problem for the above BRUTE-FORCE MAP LEARNING algorithm.

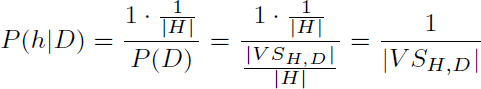
*Recalling Bayes theorem, we have*



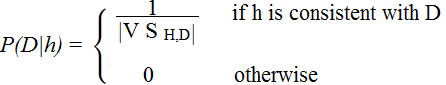
*Consider the case where h is inconsistent with the training data D*



The posterior probability of a hypothesis inconsistent with D is zero

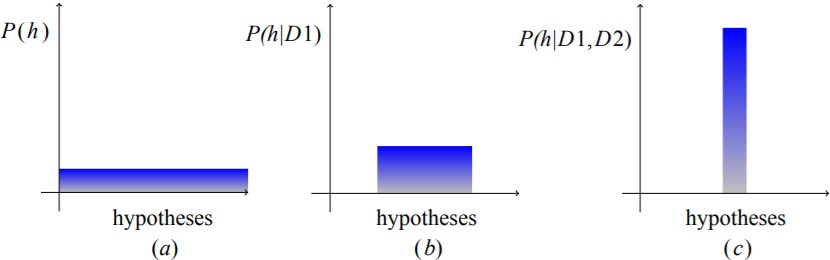
*Consider the case where h is consistent with D*

Where, VSH,D is the subset of hypotheses from H that are consistent with D

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

# The Evolution of Probabilities Associated with Hypotheses

* Figure (a) all hypotheses have the same probability.
* Figures (b) and (c), As training data accumulates, the posterior probability for inconsistent hypotheses becomes zero while the total probability summing to 1 is shared equally among the remaining consistent hypotheses.



# MAP Hypotheses and Consistent Learners

* A learning algorithm is a consistent learner if it outputs a hypothesis that commits zero errors over the training examples.
* Every consistent learner outputs a MAP hypothesis, if we assume a uniform prior probability distribution over H (P(hi) = P(hj) for all i, j), and deterministic, noise free training data (P(D|h) =1 if D and h are consistent, and 0 otherwise).

## Example:

* FIND-S outputs a consistent hypothesis, it will output a MAP hypothesis under the probability distributions P(h) and P(D|h) defined above.
* Are there other probability distributions for P(h) and P(D|h) under which FIND-S outputs MAP hypotheses? Yes.
* Because FIND-S outputs a maximally specific hypothesis from the version space, its output hypothesis will be a MAP hypothesis relative to any prior probability distribution that favours more specific hypotheses.

**Note**

* Bayesian framework is a way to characterize the behaviour of learning algorithms
* By identifying probability distributions P(h) and P(D|h) under which the output is a optimal hypothesis, implicit assumptions of the algorithm can be characterized (Inductive Bias)
* Inductive inference is modelled by an equivalent probabilistic reasoning system based on Bayes theorem

# MAXIMUM LIKELIHOOD AND LEAST-SQUARED ERROR HYPOTHESES

Consider the problem of learning a *continuous-valued target function* such as neural network learning, linear regression, and polynomial curve fitting

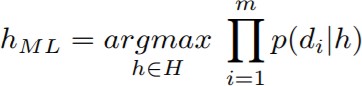
A straightforward Bayesian analysis will show that under certain assumptions any learning algorithm that minimizes the squared error between the output hypothesis predictions and the training data will output a *maximum likelihood (ML) hypothesis*

* Learner L considers an instance space X and a hypothesis space H consisting of some class of real-valued functions defined over X, i.e., (∀ h ∈ H)[ h : X → R] and training examples of the form <xi,di>
* The problem faced by L is to learn an unknown target function f : X → R
* A set of m training examples is provided, where the target value of each example is corrupted by random noise drawn according to a Normal probability distribution with zero mean (di = f(xi) + ei)
* Each training example is a pair of the form (xi ,di ) where di = f (xi ) + ei .
  + Here f(xi) is the noise-free value of the target function and ei is a random variable representing the noise.
  + It is assumed that the values of the ei are drawn independently and that they are distributed according to a Normal distribution with zero mean.
* The task of the learner is to *output a maximum likelihood hypothesis* or a *MAP hypothesis assuming all hypotheses are equally probable a priori*.

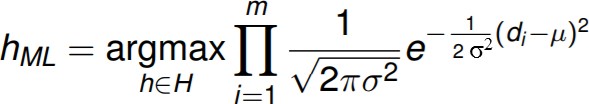
Using the definition of hML we have

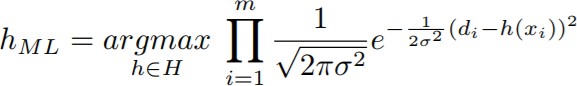


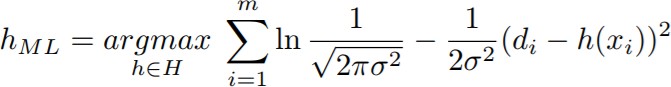
Assuming training examples are mutually independent given h, we can write P(D|h) as the product of the various (di|h)



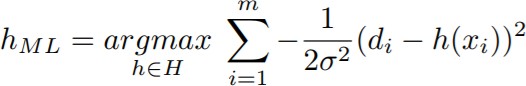
Given the noise ei obeys a Normal distribution with zero mean and unknown variance σ2 , each di must also obey a Normal distribution around the true targetvalue f(xi). Because we are writing the expression for P(D|h), we assume h is the correct description of f.

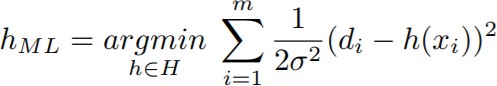
Hence, µ = f(xi) = h(xi)



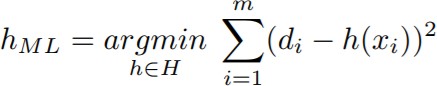
Maximize the less complicated logarithm, which is justified because of the monotonicity of function p

The first term in this expression is a constant independent of h, and can therefore be discarded, yielding



Maximizing this negative quantity is equivalent to minimizing the corresponding positive quantity

Finally, discard constants that are independent of h.



Thus, above equation shows that the maximum likelihood hypothesis hML is the one that minimizes the sum of the squared errors between the observed training values di and the hypothesis predictions h(xi)

**Note:**

Why is it reasonable to choose the Normal distribution to characterize noise?

* Good approximation of many types of noise in physical systems
* Central Limit Theorem shows that the sum of a sufficiently large number of independent, identically distributed random variables itself obeys a Normal distribution

Only noise in the target value is considered, not in the attributes describing the instances themselves

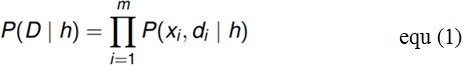
# MAXIMUM LIKELIHOOD HYPOTHESES FOR PREDICTING PROBABILITIES

* Consider the setting in which we wish to learn a nondeterministic (probabilistic) function f : X → {0, 1}, which has two discrete output values.
* We want a function approximator whose output is the probability that f(x) = 1. In other words, learn the target function f ` : X → [0, 1] such that f ` (x) = P(f(x) = 1)

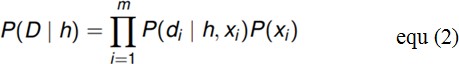
*How can we learn f ` using a neural network?*

* Use of brute force way would be to first collect the observed frequencies of 1's and 0's for each possible value of x and to then train the neural network to output the target frequency for each x.

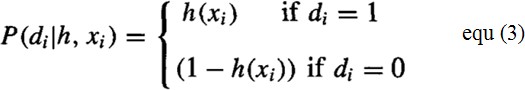
*What criterion should we optimize in order to find a maximum likelihood hypothesis for f' in this setting?*

* First obtain an expression for P(D|h)
* Assume the training data D is of the form D = {(x1, d1) . . . (xm, dm)}, where di is the observed 0 or 1 value for f (xi).
* Both xi and di as random variables, and assuming that each training example is drawn independently, we can write P(D|h) as

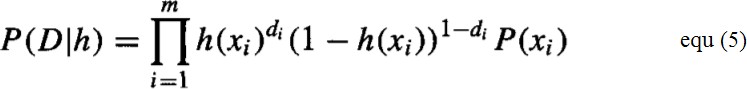
Applying the product rule



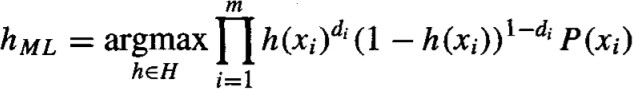
The probability P(di|h, xi)

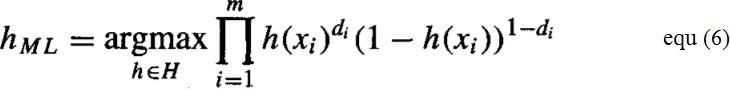


Re-express it in a more mathematically manipulable form, as

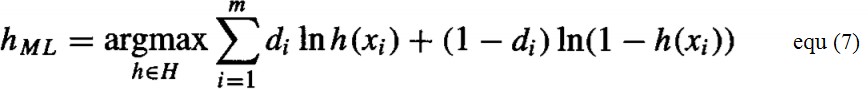
Equation (4) to substitute for P(di |h, xi) in Equation (5) to obtain

We write an expression for the maximum likelihood hypothesis



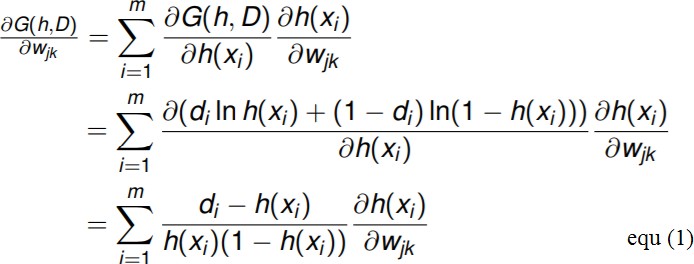
The last term is a constant independent of h, so it can be dropped

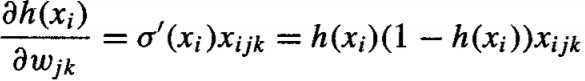
It easier to work with the log of the likelihood, yielding



Equation (7) describes the quantity that must be maximized in order to obtain the maximum likelihood hypothesis in our current problem setting

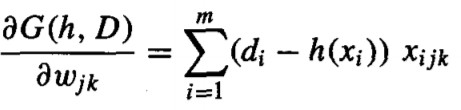
## Gradient Search to Maximize Likelihood in a Neural Net

* Derive a weight-training rule for neural network learning that seeks to maximize G(h,D) using gradient ascent
* The gradient of G(h,D) is given by the vector of partial derivatives of G(h,D) with respect to the various network weights that define the hypothesis h represented by the learned network
* In this case, the partial derivative of G(h, D) with respect to weight wjk from input k to unit j is
* Suppose our neural network is constructed from a single layer of sigmoid units. Then,

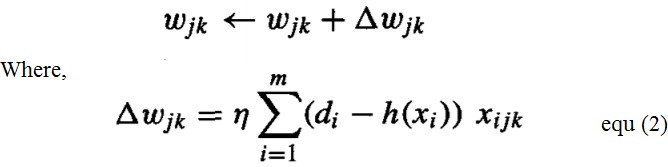


where xijk is the kth input to unit j for the ith training example, and d(x) is the derivative of the sigmoid squashing function.

* Finally, substituting this expression into Equation (1), we obtain a simple expression for the derivatives that constitute the gradient



Because we seek to maximize rather than minimize P(D|h), we perform gradient ascent rather than gradient descent search. On each iteration of the search the weight vector is adjusted in the direction of the gradient, using the weight update rule



Where, η is a small positive constant that determines the step size of the i gradient ascent search

# MINIMUM DESCRIPTION LENGTH PRINCIPLE

* A Bayesian perspective on Occam’s razor
* Motivated by interpreting the definition of hMAP in the light of basic concepts from information theory.



which can be equivalently expressed in terms of maximizing the log2



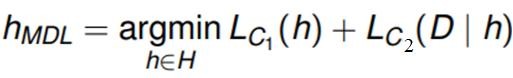
or alternatively, minimizing the negative of this quantity

This equation (1) can be interpreted as a statement that short hypotheses are preferred, assuming a particular representation scheme for encoding hypotheses and data

* -log2P(h): the description length of h under the optimal encoding for the hypothesis space H, LCH (h) = −log2P(h), where CH is the optimal code for hypothesis space H.
* -log2P(D | h): the description length of the training data D given hypothesis h, under the optimal encoding from the hypothesis space H: LCH (D|h) = −log2P(D| h) , where C D|h is the optimal code for describing data D assuming that both the sender and receiver know the hypothesis h.
* Rewrite Equation (1) to show that hMAP is the hypothesis h that minimizes the sum given by the description length of the hypothesis plus the description length of the data given the hypothesis.

Where, CH and CD|h are the optimal encodings for H and for D given h

The Minimum Description Length (MDL) principle recommends choosing the hypothesis that minimizes the sum of these two description lengths of equ.

Minimum Description Length principle:

Where, codes C1 and C2 to represent the hypothesis and the data given the hypothesis

The above analysis shows that if we choose C1 to be the optimal encoding of hypotheses CH, and if we choose C2 to be the optimal encoding CD|h, then hMDL = hMAP

## Application to Decision Tree Learning

Apply the MDL principle to the problem of learning decision trees from some training data.

*What should we choose for the representations C1 and C2 of hypotheses and data?*

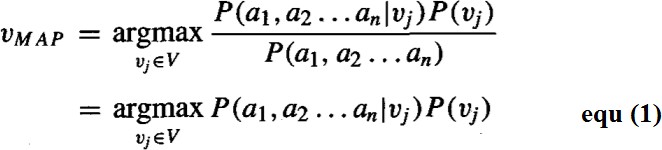
* For C1: C1 might be some obvious encoding, in which the description length grows with the number of nodes and with the number of edges
* For C2: Suppose that the sequence of instances (x1 . . .xm) is already known to both the transmitter and receiver, so that we need only transmit the classifications (f (x1) . . . f (xm)).
* Now if the training classifications (f (x1) . . .f(xm)) are identical to the predictions of the hypothesis, then there is no need to transmit any information about these examples. The description length of the classifications given the hypothesis ZERO
* If examples are misclassified by h, then for each misclassification we need to transmit a message that identifies which example is misclassified as well as its correct classification
* The hypothesis hMDL under the encoding C1 and C2 is just the one that minimizes the sum of these description lengths.

# NAIVE BAYES CLASSIFIER

* The naive Bayes classifier applies to learning tasks where each instance x is described by a conjunction of attribute values and where the target function f (x) can take on any value from some finite set V.
* A set of training examples of the target function is provided, and a new instance is presented, described by the tuple of attribute values (al, a2.. .am).
* The learner is asked to predict the target value, or classification, for this new instance.

The Bayesian approach to classifying the new instance is to assign the most probable target value, VMAP, given the attribute values (al, a2.. .am) that describe the instance

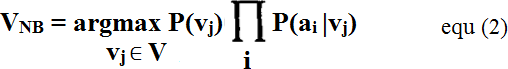
Use Bayes theorem to rewrite this expression as



* The naive Bayes classifier is based on the assumption that the attribute values are conditionally independent given the target value. Means, the assumption is that given the target value of the instance, the probability of observing the conjunction (al, a2.. .am), is just the product of the probabilities for the individual attributes:

Substituting this into Equation (1),

## Naive Bayes classifier:



Where, VNB denotes the target value output by the naive Bayes classifier

## An Illustrative Example

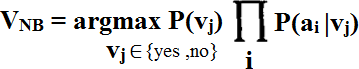
* Let us apply the naive Bayes classifier to a concept learning problem i.e., classifying days according to whether someone will play tennis.
* The below table provides a set of 14 training examples of the target concept ***PlayTennis***, where each day is described by the attributes Outlook, Temperature, Humidity, and Wind

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Day** | **Outlook** | **Temperature** | **Humidity** | **Wind** | **PlayTennis** |
| **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 |

* Use the naive Bayes classifier and the training data from this table to classify the following novel instance:

< 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

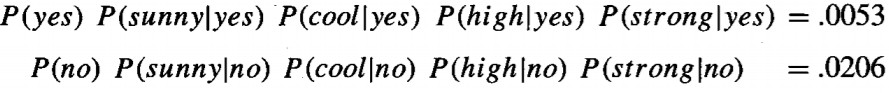


The probabilities of the different target values can easily be estimated based on their frequencies over the 14 training examples

* P(P1ayTennis = yes) = 9/14 = 0.64
* P(P1ayTennis = no) = 5/14 = 0.36

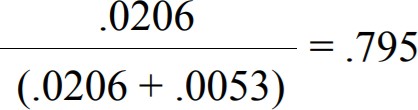
Similarly, estimate the conditional probabilities. For example, those for Wind = strong

* P(Wind = strong | PlayTennis = yes) = 3/9 = 0.33
* P(Wind = strong | PlayTennis = no) = 3/5 = 0.60

Calculate VNB according to Equation (1)

Thus, the naive Bayes classifier assigns the target value ***PlayTennis = no*** to this new instance, based on the probability estimates learned from the training data.

By normalizing the above quantities to sum to one, calculate the conditional probability that the target value is ***no***, given the observed attribute values



# Estimating Probabilities

* We have estimated probabilities by the fraction of times the event is observed to occur over the total number of opportunities.
* For example, in the above case we estimated P(Wind = strong | Play Tennis = no) by the fraction nc /n where, n = 5 is the total number of training examples for which PlayTennis = no, and nc = 3 is the number of these for which Wind = strong.
* When nc = 0, then nc /n will be zero and this probability term will dominate the quantity calculated in Equation (2) requires multiplying all the other probability terms by this zero value
* To avoid this difficulty we can adopt a Bayesian approach to estimating the probability, using the ***m-estimate*** defined as follows

## *m* -estimate of probability:



* p is our prior estimate of the probability we wish to determine, and m is a constant called the equivalent sample size, which determines how heavily to weight p relative to the observed data
* Method for choosing p in the absence of other information is to assume uniform priors; that is, if an attribute has k possible values we set p = 1 /k.

# BAYESIAN BELIEF NETWORKS

* The naive Bayes classifier makes significant use of the assumption that the values of the attributes a1 . . .an are conditionally independent given the target value v.
* This assumption dramatically reduces the complexity of learning the target function

A 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

Bayesian belief networks allow stating conditional independence assumptions that apply to subsets of the variables

## Notation

* Consider an arbitrary set of random variables Y1 . . . Yn , where each variable Yi can take on the set of possible values V(Yi).
* The joint space of the set of variables Y to be the cross product V(Y1) x V(Y2) x. . . V(Yn).
* In other words, each item in the joint space corresponds to one of the possible assignments of values to the tuple of variables (Y1 . . . Yn). 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 (Y1 . . . Yn).
* A Bayesian belief network describes the joint probability distribution for a set of variables.

## Conditional Independence

Let X, Y, and Z be three discrete-valued random variables. 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 Z, that is, if



Where,



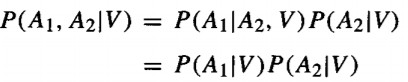
The above expression is written in abbreviated form as

P(X | Y, Z) = P(X | Z)

Conditional independence can be extended to sets of variables. The set of variables X1 . . . X*l* is conditionally independent of the set of variables Y1 . . . Ym given the set of variables Z1 . . . Zn if

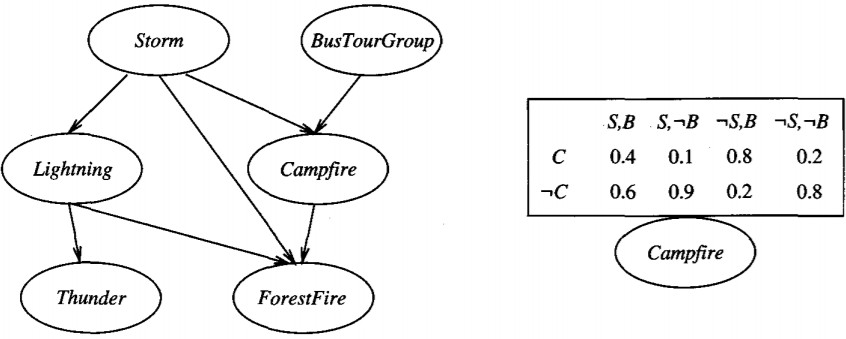


The naive Bayes classifier assumes that the instance attribute A1 is conditionally independent of instance attribute A2 given the target value V. This allows the naive Bayes classifier to calculate P(Al, A2 | V) as follows,



## Representation

A Bayesian belief network represents the joint probability distribution for a set of variables. Bayesian networks (BN) are represented by directed acyclic graphs.



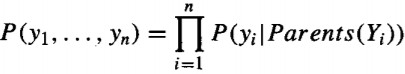
The Bayesian network in above figure represents the joint probability distribution over the boolean variables *Storm, Lightning, Thunder, ForestFire, Campfire,* and *BusTourGroup*

A Bayesian network (BN) represents the joint probability distribution by specifying a set of

*conditional independence assumptions*

* BN represented by a directed acyclic graph, together with sets of local conditional probabilities
* Each variable in the joint space is represented by a node in the Bayesian network
* The network arcs represent the assertion that the variable is conditionally independent of its non-descendants in the network given its immediate predecessors in the network.
* A ***conditional probability table* (CPT)** is given for each variable, describing the probability distribution for that variable given the values of its immediate predecessors

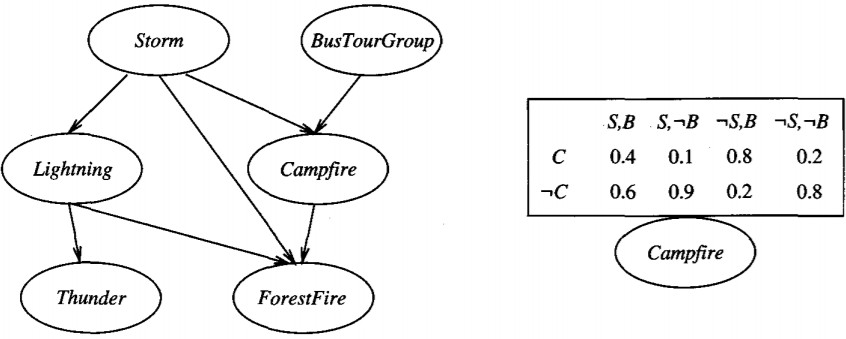
The joint probability for any desired assignment of values (y1, . . . , yn) to the tuple of network variables (Y1 . . . Ym) can be computed by the formula



Where, Parents(Yi) denotes the set of immediate predecessors of Yi in the network.

## Example:

Consider the node ***Campfire***. The network nodes and arcs represent the assertion that ***Campfire*** is conditionally independent of its non-descendants ***Lightning*** and ***Thunder***, given its immediate parents Storm and ***BusTourGroup***.



This means that once we know the value of the variables ***Storm*** and ***BusTourGroup***, the variables ***Lightning*** and ***Thunder*** provide no additional information about ***Campfire***

The conditional probability table associated with the variable ***Campfire.*** The assertion is P(Campfire = True | Storm = True, BusTourGroup = True) = 0.4

## Inference

* Use a Bayesian network to infer the value of some target variable (e.g., ForestFire) given the observed values of the other variables.
* Inference can be straightforward if values for all of the other variables in the network are known exactly.
* A Bayesian network can be used to compute the probability distribution for any subset of network variables given the values or distributions for any subset of the remaining variables.
* An arbitrary Bayesian network is known to be NP-hard

## Learning Bayesian Belief Networks

Affective algorithms can be considered for learning Bayesian belief networks from training data by considering several different settings for learning problem

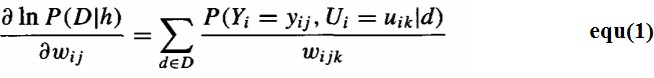
* First, the network structure might be given in advance, or it might have to be inferred from the training data.
* Second, all the network variables might be directly observable in each training example, or some might be unobservable.
  + In the case where the network structure is given in advance and the variables are fully observable in the training examples, learning the conditional probability tables is straightforward and estimate the conditional probability table entries
  + In the case where the network structure is given but only some of the variable values are observable in the training data, the learning problem is more difficult. The learning problem can be compared to learning weights for an ANN.

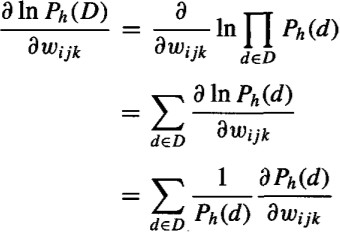
## Gradient Ascent Training of Bayesian Network

The gradient ascent rule which maximizes P(D|h) by following the gradient of ***ln P(D*|*h)*** with respect to the parameters that define the conditional probability tables of the Bayesian network.

Let wijk denote a single entry in one of the conditional probability tables. In particular wijk denote the conditional probability that the network variable Yi will take on the value yi, given that its immediate parents Ui take on the values given by uik.

The gradient of ***ln P(D|h)*** is given by the derivatives  for each of the wijk. As shown below, each of these derivatives can be calculated as



Derive the gradient defined by the set of derivatives for all ***i, j,*** and ***k***. Assuming the training examples ***d*** in the data set D are drawn independently, we write this derivative as

We write the abbreviation Ph(D) to represent P(D|h).

