# 192GEO206T MACHINE LEARNING

## Unit I INTRODUCTION

### 1.1 Learning

Humans and other animals can display behaviors that we label as intelligent by learning from experience. Learning is what gives us flexibility in our life; the fact that we can adjust and adapt to new circumstances, and learn new tricks. The important parts of animal learning are remembering, adapting, and generalizing: recognizing that last time we were in this situation (saw this data) we tried out some particular action (gave this output) and it worked (was correct), so we'll tryit again, or it didn't work, so we'll try something different. The last word, generalizing, is about recognizing similarity between different situations, so that things that applied in one place can be used in another. This is what makes learning useful, because we can use our knowledge in lots of different places. There has also been a lot of interest in making computers reason and deduce facts. This was the basis of most early Artificial Intelligence, and is sometimes known as symbolic processing because the computer manipulates symbols that reflect the environment. In contrast, machine learning methods are sometimes called sub symbolic because no symbols or symbolic manipulation are involved.

**Machine Learning**

Machine learning, then, is about making computers modify or adapt their actions (whether these actions are making predictions, or controlling a robot) so that these actions get more accurate, where accuracy is measured by how well the chosen actions reflect the correct ones. Imagine that you are playing Scrabble (or some other game) against a computer. You might beat it every time in the beginning, but after lots of games it starts beating you, until finally you never win. Either you are getting worse, or the computer is learning how to win at Scrabble. Having learnt to beat you, it can go on and use the same strategies against other

players, so that it doesn't start from scratch with each new player; this is a form of generalisation. It is only over the past decade or so that the inherent multi-disciplinarily of machine learning has been recognised. It merges ideas from neuroscience and biology, statistics, mathematics, and physics, to make computers learn.

## 1.2 Types of Machine Learning

- **Supervised learning:**

  A training set of examples with the correct responses (targets) is provided and, based on this training set, the algorithm generalizes to respond correctly to all possible inputs. This is also called learning from exemplars.

- **Unsupervised learning:**

  Correct responses are not provided, but instead the algorithm tries to identify similarities between the inputs so that inputs that have something incommon are categorized together. The statistical approach to unsupervised learning is known as density estimation.

- **Reinforcement learning:**

  This is somewhere between supervised and unsupervised learning. The algorithm gets told when the answer is wrong, but does not get told how to correct it. It has to explore and try out different possibilities until it works out how to get the answer right. Reinforcement learning is sometime called learning with a critic because of this monitor that scores the answer, butdoes not suggest improvements.

- **Evolutionary learning:**

  Biological evolution can be seen as a learning process: biological organisms adapt to improve their survival rates and chance of having offspring in their environment.

### 1.3 Supervised Learning

The webpage example is a typical problem for supervised learning. There is a set of data (the training data) that consists of a set of input data that has target data, which is the answer that the algorithm should produce, attached. This is usually written as a set of data (xi, ti), where the inputs are xi, the targets are ti, and the i index suggests that we have lots of pieces of data, indexed by i running from 1 to some upper limit N.

If we had examples of every possible piece of input data, then we could put them together into a big look-up table, and there would be no need for machine learning at all. The thing that makes machine learning better than that is generalisation: the algorithm should produce sensible outputs for inputs that weren't encountered during learning. This also has the result that the algorithm candeal with noise, which is small inaccuracies in the data that are inherent in measuring any real world process

#### 1.3.1 Regression

Suppose that I gave you the following data points and asked you to tell me the value of the output (which we will call y since it is not a target data point) when x = 0.44 (here, x, t, and y are not written in boldface font since they are scalars, as opposed to vectors).

In figure 1.3.1 Top left: A few data points from a sample problem. Bottom left: Two possible ways to predict the values between the known data points: connecting the points with straight lines, or using a cubic approximation (which in this case misses all of the points). Top and bottom right: Two more complex approximators (see the text for details) that pass through the points, although the lower one is rather better than the top.

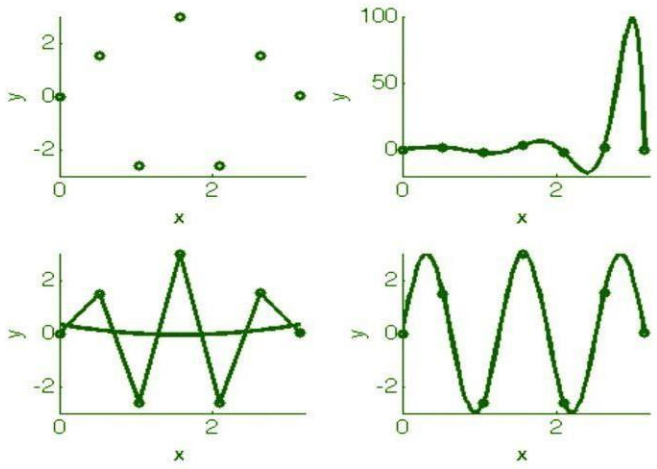



Fig 1.3.1

Since the value x = 0.44 isn't in the examples given, we need to find some way to predict what value it has. Assume that the values come from some sort of function, and try to find out what the function is. Then you'll be able to give the output value y for any given value of x. This is known as a regression problem in statistics: fit a mathematical function describing a curve, so that the curve passesas close as possible to all of the datapoints.It is generally a problem of function approximation or interpolation, working out the value between values that we know.

The problem is how to work out what function to choose. Have a look at Figure

| $x$ | $t$ |
|---|---|
| 0 | 0 |
| 0.5236 | 1.5 |
| 1.0472 | -2.5981 |
| 1.5708 | 3.0 |
| 2.0944 | -2.5981 |
| 2.6180 | 1.5 |
| 3.1416 | 0 |

1.3.1 The top-left plot shows a plot of the 7 values of x and y in the table, while the other plots show different attempts to fit a curve through the data points. The

bottom-left plot shows two possible answers found by using straight lines toconnect up the points, and also what happens if we try to use a cubic function (something that can be written as $ax^3 + bx^2 + cx + d = 0$). The top-right plot shows what happens when we try to match the function using a different polynomial, this time of the form $ax^{10} + bx^9 + \ldots + jx + k = 0$, and finally the bottom-right plot shows the function $y = 3 \sin(5x)$. In fact, the data were made with the sine functionplotted on the bottom-right, so that is the correct answer in this case, but the algorithm doesn't know that, and to it the two solutions on the right both look equally good. The only way we can tell which solution is better is to test how well they generalise. This will tell us that the bottom-right curve is better in the example. So one thing that our machine learning algorithms can do is interpolate between data points. This might not seem to be intelligent behaviour, or even very difficult in two dimensions, but it is rather harder in higher dimensional spaces.

### 1.3.2 Classification

The classification problem consists of taking input vectors and deciding which of N classes they belong to, based on training from exemplars of each class. The most important point about the classification problem is that it is discrete— each example belongs to precisely one class, and the set of classes covers the whole possible output space. These two constraints are not necessarily realistic; sometimes examples might belong partially to two different classes. There are many places where we might not be able to categorise every possible input. For example, consider a vending machine, where we use a neural network to learn to recognise all the different coins. We train the classifier to recognise all New Zealand coins, but what if a British coin is put into the machine? In that case, the classifier will identify it as the New Zealand coin that is closest to it in appearance,but this is not really what is wanted: rather, the classifier should identify that it is not one of the coins it was trained on. This is called novelty detection.

Let’s consider how to set up a coin classifier. When the coin is pushed into the slot, the machine takes a few measurements of it. These could include the diameter, the weight, and possibly the shape, and are the features that will generateour input vector. In this case, our input vector will have three elements, each of which will be a number showing the measurement of that feature (choosing a number to represent the shape would involve an encoding, for example that 1=circle, 2=hexagon, etc.). Of course, there are many other features that we could measure. If our vending machine included an atomic absorption spectroscope, thenwe could estimate the density of the material and its composition, or if it had a camera, we could take a photograph of the coin and feed that image into the classifier.

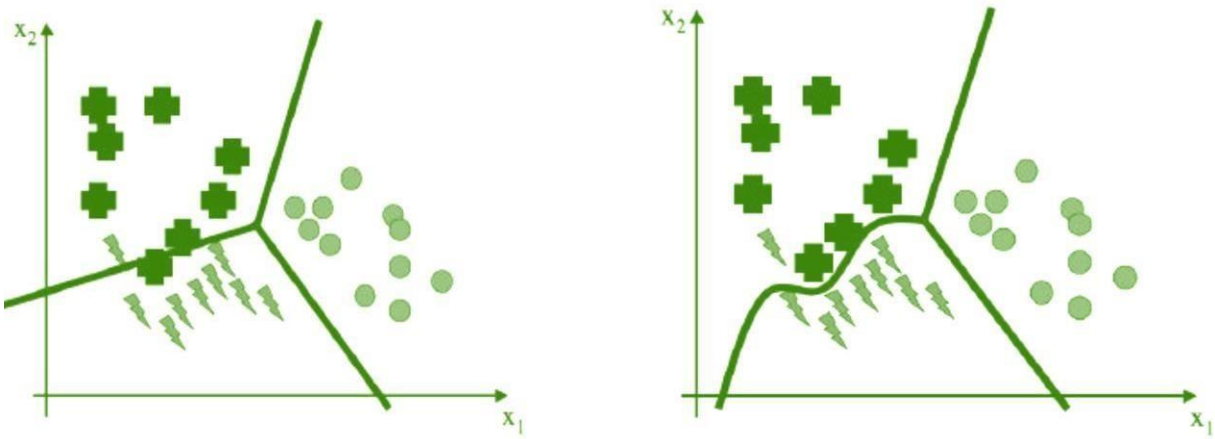

Fig 1.3.2

As the number of input dimensions grows, the number of data points required increases faster; this is known as the curse of dimensionality but we need to make sure that we can reliably separate the classes based on those features. For example, if we tried to separate coins based only on color, we wouldn’t get very far, because the 20 ¢ and 50 ¢ coins are both silver and the $1 and $2 coins both bronze. However, if we use color and diameter, we can do a pretty good job of the coin classification problem for NZ coins.

There are some features that are entirely useless. For example, knowing that the coin is circular doesn’t tell us anything about NZ coins, which are all circular

(see Figure 1.3.2). In other countries, though, it could be very useful. Given the features that are used as inputs to the classifier, we need to identify some values of those features that will enable us to decide which class the current input is in. Figure 1.3.3 shows a set of 2D inputs with three different classes shown, and two different decision boundaries; on the left they are straight lines, and are therefore simple, but don't categorize as well as the non-linear curve on the right

FIGURE 1.3.3 Left:A set of straight line decision boundaries for a classification problem. Right: An alternative set of decision boundaries that separate the plusses from the lightening strikes better, but requires a line that isn't straight.




## 1.4 The Brain and the Neuron

- The brain is an impressively powerful and complicated system, the basic building blocks that it is made up of are fairly simple and easy to understand.
- These are nerve cells called neurons. (100 billion = $10^{11}$ is the figure that is often given) and they come in lots of different types, depending upon their particular task.
- However, their general operation is similar in all cases: transmitter chemicals within the fluid of the brain raise or lower the electrical potential inside the bodyof the neuron.

- If this membrane potential reaches some threshold, the neuron spikes or fires, and a pulse of fixed strength and duration is sent down the axon.
- The axons divide (arborise) into connections to many other neurons, connecting to each of these neurons in a synapse.
- Each neuron is typically connected to thousands of other neurons, so that it is estimated that there are about 100 trillion (= $10^{14}$) synapses within the brain. After firing, the neuron must wait for some time to recover its energy (the refractory period) before it can fire again.
- Each neuron can be viewed as a separate processor, performing a very simple computation: deciding whether or not to fire. This makes the brain a massively parallel computer made up of $10^{11}$ processing elements.

### 1.4.1 Hebb's Rule

- Hebb's rule says that the changes in the strength of synaptic connections are proportional to the correlation in the firing of the two connecting neurons. So if two neurons consistently fire simultaneously, then any connection betweenthem will change in strength, becoming stronger.
- However, if the two neurons never fire simultaneously, the connection between them will die away. The idea is that if two neurons both respond to something, then they should be connected.
- There are other names for this idea that synaptic connections between neurons and assemblies of neurons can be formed when they fire together and can become stronger. It is also known as long-term potentiation and neural plasticity, and it does appear to have correlates in real brains.

### 1.4.2 McCulloch and Pitts Neurons

McCulloch and Pitts produced a perfect example of this when they modelled a neuron as:

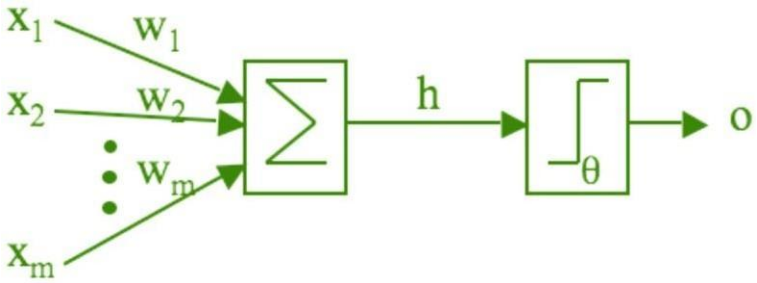



FIGURE 1.4.1 A picture of McCulloch and Pitts' mathematical model of a neuron. The inputs $x_i$ are multiplied by the weights $w_i$, and the neurons sum their values. If this sum is greater than the threshold then the neuron fires; otherwise it does not.

(1) a set of weighted inputs $w_i$ that correspond to the synapses

(2) an adder that sums the input signals (equivalent to the membrane of the cell that collects electrical charge)

(3) an activation function (initially a threshold function) that decides whether the neuron fires ('spikes') for the current inputs

- A picture of their model is given in Figure 3.1, and we'll use the picture to write down a mathematical description. On the left of the picture are a set of input nodes (labelled $x_1$, $x_2$, . . . $x_m$).
- These are given some values, and as an example we'll assume that there are three inputs, with $x_1 = 1$, $x_2 = 0$, $x_3 = 0.5$.
- In real neurons those inputs come from the outputs of other neurons. So the 0 means that a neuron didn't fire, the 1 means it did, and the 0.5 has no biological meaning.

- Each of these other neuronal firings flowed along a synapse to arrive at our neuron, and those synapses have strengths, called weights. The strength of the synapse affects the strength of the signal, so we multiply the input by the weight of the synapse (so we get $x_1 \times w_1$ and $x_2 \times w_2$, etc.). Now when all of these signals arrive into our neuron, it adds them up to see if there is enough strengthto make it fire.
- which just means sum (add up) all the inputs multiplied by their synaptic weights.

$$h = \sum_{i=1}^{m} w_i x_i,$$

- Assumed that there are m of them, where m = 3 in the example.
- If the synaptic weights are w1 = 1,w2 = −0.5,w3 = −1, then the inputs to our model neuron are h = 1 × 1 + 0 ×−0.5 + 0.5 × −1 = 1 + 0 + −0.5 = 0.5.
- Now the neuron needs to decide if it is going to fire. For a real neuron, this is a question of whether the membrane potential is above some threshold. Pick a threshold value (labelled Θ), say Θ = 0 as an example.
- Now, does our neuron fire? Well, h = 0.5 in the example, and 0.5 > 0, so the neuron does fire, and produces output 1.
- If the neuron did not fire, it would produce output 0.
- The McCulloch and Pitts neuron is a binary threshold device. It sums up the inputs (multiplied by the synaptic strengths or weights) and either fires (producesoutput 1) or does not fire (produces output 0) depending on whether the input is above some threshold.
- We can write the second half of the work of the neuron, the decision about whether or not to fire (which is known as an activation function), as:

$$o = g(h) = \begin{cases} 1 & \text{if } h > \theta \\ 0 & \text{if } h \leq \theta. \end{cases}$$

- This is a very simple model, but we are going to use these neurons, or very simple variations on them using slightly different activation functions (that is, we'll replace the threshold function with something else) for most of our study ofneural networks.
- In fact, these neurons might look simple, but as we shall see, a network of such neurons can perform any computation that a normal computer can, provided that the weights $w_i$ are chosen correctly.

### 1.4.3 Limitations of the McCulloch and Pitts Neuronal Model

- Real neurons are much more complicated. The inputs to a real neuron are not necessarily summed linearly: there may be non-linear summations.
- However, the most noticeable difference is that real neurons do not output a single output response, but a spike train, that is, a sequence of pulses, and it is this spike train that encodes information.
- This means that neurons don't actually respond as threshold devices, but produce a graded output in a continuous way.
- They do still have the transition between firing and not firing, though, but the threshold at which they fire changes over time.
- The neurons are not updated sequentially according to a computer clock, but update themselves randomly (asynchronously), whereas in many of our models we will update the neurons according to the clock.
- There are neural network models that are asynchronous, but for our purposes we will stick to algorithms that are updated by the clock.
- Note that the weights $w_i$ can be positive or negative. This corresponds to excitatory and inhibitory connections that make neurons more likely to fire and less likely to fire, respectively.

- Both of these types of synapses do exist within the brain, but with the McCulloch and Pitts neurons, the weights can change from positive to negative or vice versa, which has not been seen biologically—synaptic connections are either excitatory or inhibitory, and never change from one to the other.
- Additionally, real neurons can have synapses that link back to themselves in a feedback loop, but we do not usually allow that possibility when we make networks of neurons.
- McCulloch and Pitts neurons already provide a great deal of interesting behaviour that resembles the action of the brain, such as the fact that networks of McCulloch and Pitts neurons can memorise pictures and learn to represent functions and classify data.

## 1.5 Design a Learning System

### 1.5.1 Choosing the Training Experience

- The first design choice we face is to choose the type of training experience from which our system will learn.
- The type of training experience available can have a significant impact on success or failure of the learner.
- One key attribute is whether the training experience provides direct or indirect feedback regarding the choices made by the performance system.
- A second important attribute of the training experience is the degree to which the learner controls the sequence of training examples.
- A third important attribute of the training experience is how well it represents the distribution of examples over which the final system performance P must be measured.
- In general, learning is most reliable when the training examples follow a distribution similar to that of future test examples.

- In our checkers learning scenario, the performance metric P is the percent of games the system wins in the world tournament. If its training experience E consists only of games played against itself, there is an obvious danger that thistraining experience might not be fully representative of the distribution of situations over which it will later be tested.

**A checkers learning problem:**

- Task T: playing checkers
- Performance measure P: percent of games won in the world tournament
- Training experience E: games played against itself

In order to complete the design of the learning system, we must now choose

1. The exact type of knowledge to be learned
2. A representation for this target knowledge
3. A learning mechanism

### 1.5.2 Choosing the Target Function

- The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program.
- Let us begin with a checkers-playing program that can generate the legal moves from any board state.
- The program needs only to learn how to choose the best move from among these legal moves.
- This learning task is representative of a large class of tasks for which the legal moves that define some large search space are known a priori, but for which the best search strategy is not known. Many optimization problems fall into this class.
- Given this setting where we must learn to choose among the legal moves, the most obvious choice for the type of information to be learned is a program,or function, that chooses the best move for any given board state.

- Let us call this function **ChooseMove**and use the notation
- **ChooseMove**: B →M to indicate that this function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M.
- The choice of the target function will therefore be a key design choice.
- Although ChooseMove is an obvious choice for the target function in our example, this function will turn out to be very difficult to learn given the kind of indirect training experience available to our system.
- An alternative target function and one that will turn out to be easier to learn in this setting-is an evaluation function that assigns a numerical score to any given board state. Let us call this target function V and again use the notation V : B → R to denote that V maps any legal board state from the set B to some real value (we use R to denote the set of real numbers).
- If the system can successfully learn such a target function V, then it can easily use it to select the best move from any current board position.
- This can be accomplished by generating the successor board state produced by every legal move, then using V to choose the best successor state and therefore the best legal move.

Let us therefore define the target value V(b) for an arbitrary board state b in B, as follows:

1. if b is a final board state that is won, then V(b) = 100
2. if b is a final board state that is lost, then V(b) = -100
3. if b is a final board state that is drawn, then V(b) = 0
4. if b is a not a final state in the game, then V(b) = V(b'), where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game (assuming the opponent plays optimally, as well).

- While this recursive definition specifies a value of V(b) for every board state b, this definition is not usable by our checkers player because it is not efficiently computable.
- Except for the trivial cases (cases 1-3) in which the game has already ended, determining the value of V(b) for a particular board state requires (case 4) searching ahead for the optimal line of play, all the way to the end of the game! Because this definition is not efficiently computable by our checkers playing program, we say that it is a nonoperational definition.
- The goal of learning in this case is to discover an operational description of V ; that is, a description that can be used by the checkers-playing program toevaluate states and select moves within realistic time bounds.
- Thus, we have reduced the learning task in this case to the problem of discovering an operational description of the ideal target function V.
- It may be very difficult in general to learn such an operational form of V perfectly. In fact, we often expect learning algorithms to acquire only some approximation to the target function, and for this reason the process of learning the target function is often called function approximation.

### 1.5.3 Choosing a Representation for the Target Function

To keep the discussion brief, let us choose a simple representation: for any given board state, the function c will be calculated as a linear combination of the following board features:

- $x_1$: the number of black pieces on the board
- $x_2$: the number of red pieces on the board
- $x_3$: the number of black kings on the board
- $x_4$: the number of red kings on the board
- $x_5$: the number of black pieces threatened by red (i.e., which can be captured on red's next turn)

- $x_6$: the number of red pieces threatened by black

Thus, our learning program will represent c(b) as a linear function of the form

$$\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

- Where $w_o$ through $w_6$ are numerical coefficients, or weights, to be chosen by the learning algorithm.
- To summarize our design choices thus far, we have elaborated the original formulation of the learning problem by choosing a type of training experience, a target function to be learned, and a representation for this target function. Our elaborated learning task is now
- Partial design of a checkers learning program:
  - Task T: playing checkers
  - Performance measure P: percent of games won in the world tournament
  - Training experience E: games played against itself
  - Target function: V:Board □R
  - Target function representation

$$\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

The first three items above correspond to the specification of the learning task, whereas the final two items constitute design choices for the implementation of the learning program.

### 1.5.4 Choosing a Function Approximation Algorithm

In order to learn the target function V^ we require a set of training examples, each describing a specific board state b and the training value $V_{train}(b)$ for b. In other words, each training example is an ordered pair of the form (b, $V_{train}(b)$). For instance, the following training example describes a board state b in which black has won the game (note $x_2 = 0$ indicates that red has no remaining pieces) and for which the target function value $V_{train}(b)$ is therefore +100.

$((x_1 = 3, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 0, x_6 = 0), +100)$

The procedure :-

- Derives such training examples from the indirect training experience available to the learner
- adjusts the weights $w_i$ to best fit these training examples.

### 1.5.4.1 ESTIMATING TRAINING VALUES

Assign the training value of $V_{train}(b)$ for any intermediate board state b to be V^(successor(b)) where V^ is the learner's current approximation to V and where Successor(b) denotes the next board state following b for which it is again the program's turn to move (i.e., the board state following the program's move and the opponent's response). This rule for estimating training values canbe summarized as Rule for estimating training values.

$V_{train}(b) \leftarrow$ V^(successor(b))

### 1.5.4.2 ADJUSTING THE WEIGHTS

Several algorithms are known for finding weights of a linear function that minimize E defined in this way. In our case, we require an algorithm that will incrementally refine the weights as new training examples become available and that will be robust to errors in these estimated training values. One such algorithm is called the least mean squares, or LMS training rule. For each observed training example it adjuststhe weights a small amount in the direction that reduces the error on this training example.

$$E \equiv \sum_{\langle b, V_{train}(b)\rangle \in \text{ training examples}} (V_{train}(b) - \hat{V}(b))^2$$

**LMS weight update rule.**

For each training example (b ,$V_{train}(b)$)

- Use the current weights to calculate V^(b)
- For each weight $w_i$, update it as

$$w_i \leftarrow w_i + \eta \, (V_{train}(b) - \hat{V}(b)) \, x_i$$

Here η is a small constant (e.g., 0.1) that moderates the size of the weight update. To get an intuitive understanding for why this weight update rule works, notice that when the error ($V_{train}$(b) – V^(b)) is zero, no weights are changed. When error ($V_{train}$(b) – V^(b)) is positive (i.e., when V^b) is too low), then each weight is increased in proportion to the value of its corresponding feature. This will raise the value of V^(b), reducing the error.

### 1.5.5 The Final Design

The final design of our checkers learning system can be naturally described by four distinct program modules that represent the central components in many learning systems. These four modules, summarized in Figure 1.5.5, are as follows:

The Performance System is the module that must solve the given performance task, in this case playing checkers, by using the learned target function(s). It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output. In our case, the

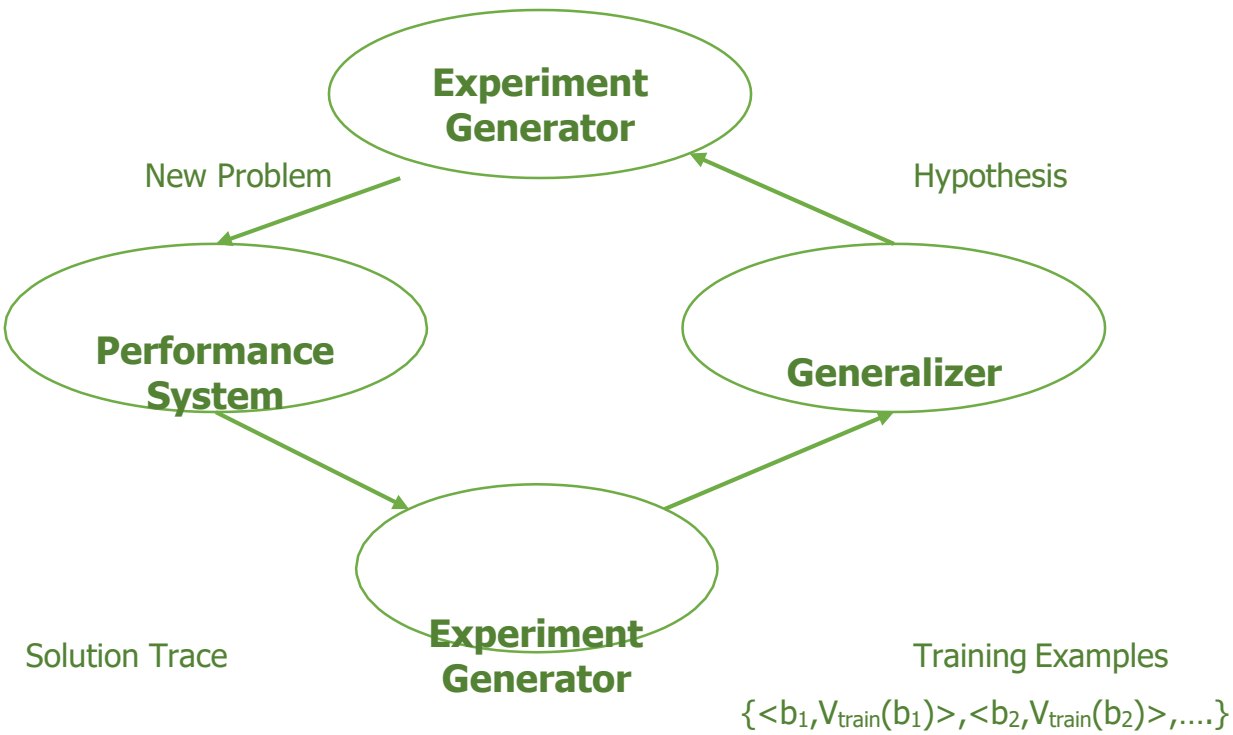


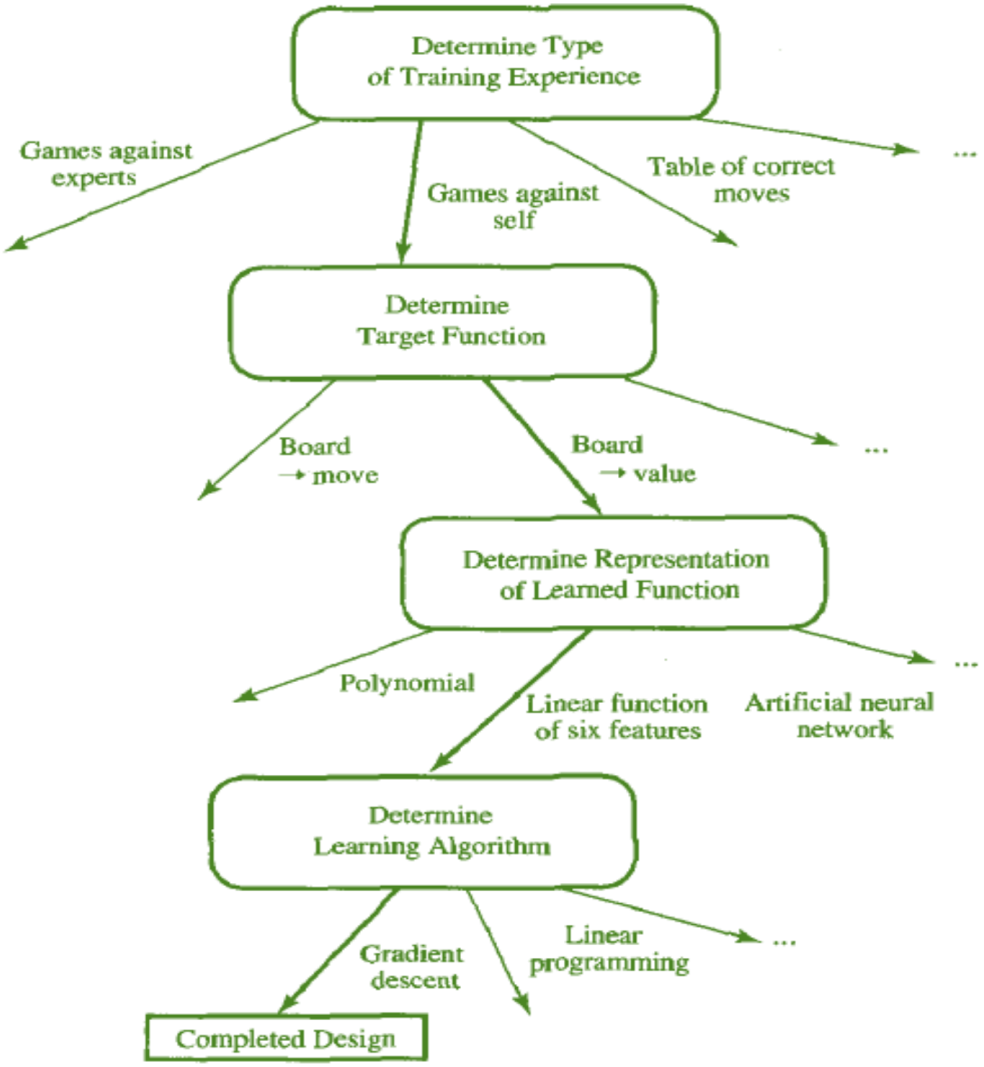

**Fig 1.5.5 Final design of the checkers learning program.**

•The Performance System is the module that must solve the given performance task, in this case playing checkers, by using the learned target function(s). It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output.

•The Critic takes as input the history or trace of the game and produces as output a set of training examples of the target function.

•The Generalizer takes as input the training examples and produces an output hypothesis that is its estimate of the target function. It generalizes from the specific training examples, hypothesizing a general function that covers these examples and other cases beyond the training examples.

•The Experiment Generator takes as input the current hypothesis (currently learned function) and outputs a new problem (i.e., initial board state) for the Performance System to explore. Its role is to pick new practice problems that will maximize the learning rate of the overall system.




## 1.6 Perspectives and Issues in Machine Learning

One useful perspective on machine learning is that it involves searching a very large space of possible hypotheses to determine one that best fits the observed data and any prior knowledge held by the learner. For example, consider the space of hypotheses that could in principle be output by the above checkers learner. This hypothesis space consists of all evaluation functions that can be represented by some choice of values for the weights $w_o$ through $w_6$.

The learner's task is thus to search through this vast space to locate the hypothesis that is most consistent with the available training examples. The LMS algorithm for fitting weights achieves this goal by iteratively tuning the weights,

adding a correction to each weight each time the hypothesized evaluation function predicts a value that differs from the training value. This algorithm works well when the hypothesis representation considered by the learner defines a continuously parameterized space of potential hypotheses.

### 1.6.1 Issues in Machine Learning

Our checkers example raises a number of generic questions about machine learning.

- What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?
- How much training data is sufficient? What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experienceand the character of the learner's hypothesis space?
- When and how can prior knowledge held by the learner guide the process of generalizing from examples? Can prior knowledge be helpful even when it is only approximately correct?
- What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?
- What is the best way to reduce the learning task to one or more function approximation problems? Put another way, what specific functions should the system attempt to learn? Can this process itself be automated?
- How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

## 1.7 Concept Learning Task

consider the example task of learning the target concept "days on which my friend Aldo enjoys his favorite water sport." Table 1.7 describes a set of example days, each represented by a set of attributes. The attribute EnjoySport indicates whether or not Aldo enjoys his favorite water sport on this day. The task is to learn to predict the value of EnjoySport for an arbitrary day, based on the values of its other attributes.

**What hypothesis representation shall we provide to the learner in this case?**

Many possible representations

- in the following: h is conjunction of constraints on attributes
- Each constraint can be
  - a specfic value (e.g., Water = Warm)
  - don't care (e.g., "Water =?")
  - no value allowed (e.g.,"Water=∅")
- For example,

  Sky AirTemp Humid Wind Water Forecst

  <Sunny ? ? Strong ? Same>
- We write h(x) = 1 for a day x, if x satisfies the description
- Note that much more expressive languages exists.
- Most general hypothesis: (?, ?, ?, ?, ?)
- Most specific hypothesis: (∅,∅,∅,∅,∅)

**Table 1.7 Positive and negative training examples for the target concept EnjoySport.**

| Example | *Sky* | *AirTemp* | *Humidity* | *Wind* | *Water* | *Forecast* | *EnjoySport* |
|---|---|---|---|---|---|---|---|
| 1 | Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| 2 | Sunny | Warm | High | Strong | Warm | Same | Yes |
| 3 | Rainy | Cold | High | Strong | Warm | Change | No |
| 4 | Sunny | Warm | High | Strong | Cool | Change | Yes |

### 1.7.1 Prototypical Concept Learning Task

Given:

- Instances X: Possible days, each described by the attributes

  Sky, AirTemp, Humidity, Wind, Water, Forecast
- Target function c: EnjoySport : X → {0, 1}
- Hypotheses H: Conjunctions of literals. E.g.

  <?,Cold,High, ?, ?, ?>
- Training examples D: Positive and negative examples of the target function

  $<x_1, c(x_1)i, \ldots <x_m, c(x_m)>$

- Determine: A hypothesis h in H with h(x) = c(x) for all x in D.
- To summarize, the EnjoySport concept learning task requires learning the set of days for which EnjoySport = yes, describing this set by a conjunction of constraints over the instance attributes.

### 1.7.2 Notation

The set of items over which the concept is defined is called the set of instances, which we denote by X. In the current example, X is the set of all possible days, each represented by the attributes Sky, AirTemp, Humidity, Wind, Water, and Forecast. The concept or function to be learned is called the target concept, which we denote by c. In general, c can be any Boolean valued function defined over the instances X; that is, c : X + {O, 1). In the current example, the target concept corresponds to the value of the attribute EnjoySport

(i.e., c(x) = 1 if EnjoySport = Yes, and c(x) = 0 if EnjoySport = No).

### 1.7.3 The Inductive Learning Hypothesis

The inductive learning hypothesis: Any hypothesis found to approximate thetarget function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.

- I.e. the training set needs to ’represent’ the whole domain (which may be infinite)
- Even if we have a ’good’ training set, we can still construct bad hypotheses!

### 1.8 Concept Learning as Search

- Concept learning can be viewed as the task of searching through a large space of hypotheses implicitly defined by the hypothesis representation.
- The goal of this search is to find the hypothesis that best fits the training examples.

Example:

Consider the instances X and hypotheses H in the EnjoySport learning task. The attribute Sky has three possible values, and AirTemp, Humidity, Wind, Water, Forecast each have two possible values, the instance space X contains exactly

3.2.2.2.2.2 = 96 distinct instances

5.4.4.4.4.4 = 5120 syntactically distinct hypotheses within H.

Every hypothesis containing one or more "Φ" symbols represents the empty set of instances; that is, it classifies every instance as negative.

1 + (4.3.3.3.3.3) = 973. Semantically distinct hypotheses

#### 1.8.1 General-to-Specific Ordering of Hypotheses

Consider the two hypotheses $h_1$ =

(Sunny, ?, ?, Strong, ?, ?) $h_2$

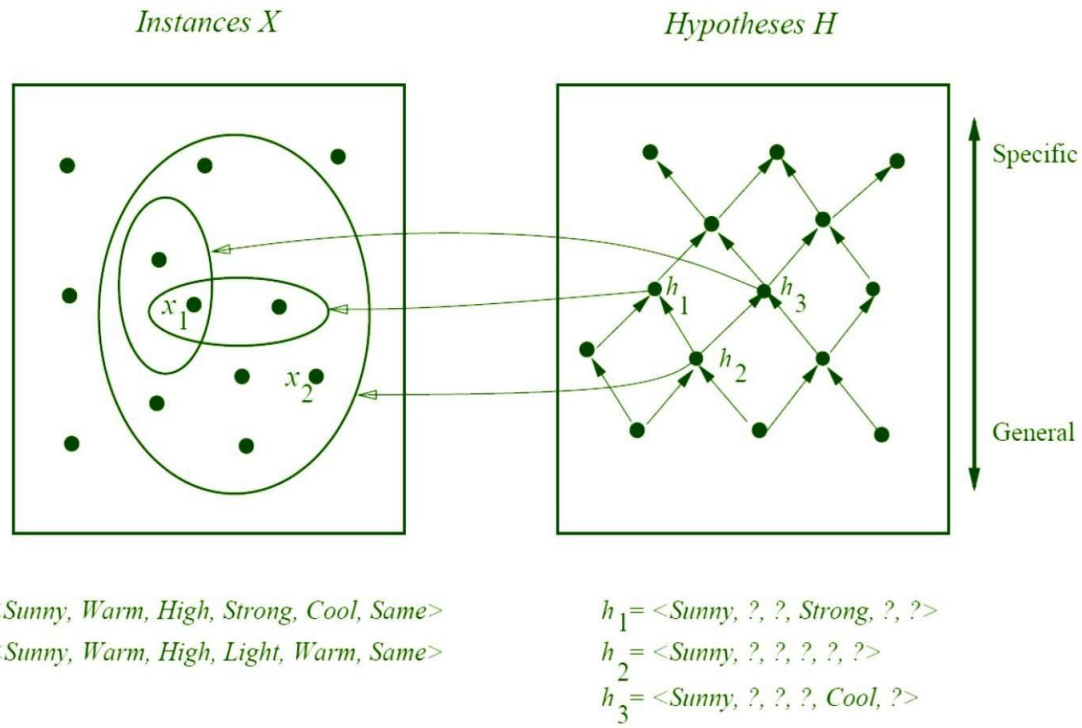
= (Sunny, ?, ?, ?, ?, ?)

- Consider the sets of instances that are classified positive by $h_l$ and by $h_2$.
- $h_2$ imposes fewer constraints on the instance, it classifies more instances as positive. So, any instance classified positive by $h_l$ will also be classified positive by $h_2$. Therefore, $h_2$ is more general than hl.

Given hypotheses $h_j$ and $h_k$, $h_j$ is more-general-than or- equal do $h_k$ if and only if any instance that satisfies $h_k$ also satisfies $_{hi}$

Definition: Let $h_j$ and $h_k$ be Boolean-valued functions defined over X. Then $h_j$ is more general-than-or-equal-to $h_k$ (written $h_j \geq h_k$) if and only if

$$(\forall\, x \in X)\, [(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$




- In the figure, the box on the left represents the set X of all instances, the box on the right the set H of all hypotheses.
- Each hypothesis corresponds to some subset of X-the subset of instances that it classifies positive.
- The arrows connecting hypotheses represent the more - general -than relation, with the arrow pointing toward the less general hypothesis.
  - Note the subset of instances characterized by $h_2$ subsumes the subset characterized by $h_1$ , hence $h_2$ is more - general– than $h_1$

## 1.9 Finding a Maximally Specific Hypothesis

FIND-S Algorithm

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x For each attribute constraint ai in h

   If the constraint ai is satisfied by x

   Then do nothing

   Else replace $a_i$ in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

To illustrate this algorithm, assume the learner is given the sequence of training examples from the EnjoySport task

| Example | Sky | AirTemp | Humidity | Wind | Water | Forecast | EnjoySport |
|---|---|---|---|---|---|---|---|
| 1 | Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| 2 | Sunny | Warm | High | Strong | Warm | Same | Yes |
| 3 | Rainy | Cold | High | Strong | Warm | Change | No |
| 4 | Sunny | Warm | High | Strong | Cool | Change | Yes |

- The first step of FIND-S is to initialize h to the most specific hypothesis in H h – (Ø, Ø, Ø, Ø, Ø, Ø)
- Consider the first training example x1 = <Sunny Warm Normal Strong Warm Same>, +

Observing the first training example, it is clear that hypothesis h is too specific. None of the "Ø" constraints in h are satisfied by this example, so each is replaced by the next more general constraint that fits the example

h1 = <Sunny Warm Normal Strong Warm Same>

- Consider the second training example x2 = <Sunny, Warm, High, Strong, Warm, Same>, +

The second training example forces the algorithm to further generalize h, this time substituting a "?" in place of any attribute value in h that is not satisfied by the new example h2 = <Sunny Warm ? Strong Warm Same>

- Consider the third training example x3 = <Rainy,

  Cold, High, Strong, Warm, Change>, -

  Upon encountering the third training the algorithm makes no change to h. The FIND-S algorithm simply ignores every negative example. h3 = < Sunny Warm ? Strong Warm Same>

- Consider the fourth training example x4 = <Sunny

  Warm High Strong Cool Change>, +

  The fourth example leads to a further generalization of

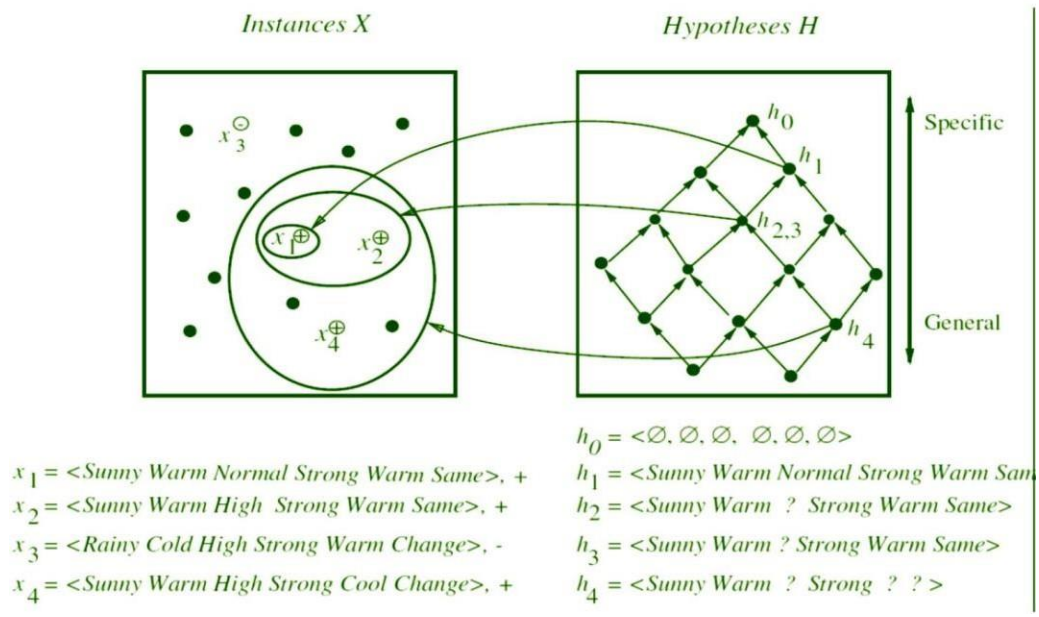
  h h4 = < Sunny Warm ? Strong ? ? >

- Consider the fourth training example x4 = <Sunny

  Warm High Strong Cool Change>, +

  The fourth example leads to a further generalization of

  h h4 = < Sunny Warm ? Strong ? ? >



**The key property of the FIND-S algorithm**

FIND-S is guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples

FIND-S algorithm's final hypothesis will also be consistent with the negative examples provided the correct target concept is contained in H, and provided the training examples are correct.

**Unanswered by FIND-S**

1. Has the learner converged to the correct target concept?
2. Why prefer the most specific hypothesis?
3. Are the training examples consistent?
4. What if there are several maximally specific consistent hypotheses?

## 1.10 Version Spaces and the Candidate Elimination Algorithm

The key idea in the CANDIDATE-ELIMINATION algorithm is to output a description of the set of all hypotheses consistent with the training examples

**Representation**

Definition: consistent- A hypothesis h is consistent with a set of training examples D if and only if h(x) = c(x) for each example (x, c(x)) in D.

$$Consistent\ (h, D) \equiv (\forall\ \langle x, c(x)\rangle \in D)\ h(x) = c(x))$$

• Note difference between definitions of consistent and satisfies

- ✓ An example x is said to satisfy hypothesis h when h(x) = 1, regardless of whether x is a positive or negative example of the target concept.
- ✓ An example x is said to consistent with hypothesis h iff h(x) = c(x)

Definition: $VS_{H,D} \equiv \{h \in H \mid Consistent\ (h, D)\}$

version space- The version space, denoted $V\ S_{H,\ D}$ with respect to hypothesis space H and training examples D, is the subset of hypotheses from H consistent with the training examples in D

## The LIST-THEN-ELIMINATION algorithm

The LIST-THEN-ELIMINATE algorithm first initializes the version space to contain all hypotheses in H and then eliminates any hypothesis found inconsistent with any training example.

1. VersionSpace c a list containing every hypothesis in H
2. For each training example, (x, c(x))

   remove from VersionSpace any hypothesis h for which $h(x) \neq c(x)$
3. Output the list of hypotheses in VersionSpace

- List-Then-Eliminate works in principle, so long as version space is finite.
- However, since it requires exhaustive enumeration of all hypotheses in practice it is not feasible.

**1.10.1** A More Compact Representation for Version Spaces

The version space is represented by its most general and least general members. These members form general and specific boundary sets that delimit the version space within the partially ordered hypothesis space.

Definition: The general boundary G, with respect to hypothesis space H and training data D, is the set of maximally general members of H consistent with D

$$G \equiv \{g \in H \mid Consistent(g, D) \wedge (\neg \exists g' \in H)[(g' >_g g) \wedge Consistent(g', D)]\}$$

**Definition:** The **specific boundary** S, with respect to hypothesis space H and training data D, is the set of minimally general (i.e., maximally specific) members of H consistent with D.

$$S \equiv \{s \in H \mid Consistent(s, D) \wedge (\neg \exists s' \in H)[(s >_g s') \wedge Consistent(s', D)]\}$$

Theorem: Version Space representation theorem

Theorem: Let X be an arbitrary set of instances and Let H be a set of Boolean-valued hypotheses defined over X. Let c: $X \rightarrow \{O, 1\}$ be an arbitrary target concept defined

over X, and let D be an arbitrary set of training examples {(x, c(x))). For all X, H, c, and D such that S and G are well defined, To Prove:

1. Every h satisfying the right hand side of the above expression is in $VS_{H, D}$
2. Every member of $VS_{H, D}$ satisfies the right-hand side of the expression

$$VS_{H,D} = \{ h \in H \mid (\exists s \in S)(\exists g \in G)(g \geq_g h \geq_g s) \}$$

Sketch of proof:

1. let g, h, s be arbitrary members of G, H, S respectively with $g \geq_g h \geq_g s$

- By the definition of S, s must be satisfied by all positive examples in D. Because h $\geq_g$ s , h must also be satisfied by all positive examples in D.
- By the definition of G, g cannot be satisfied by any negative example in D, and because g $\geq_g$ h cannot be satisfied by any negative example in D. Because h is satisfied by all positive examples in D and by no negative examples in D, h is consistent with D, and therefore h is a member of $VS_{H,D}$.

2. It can be proven by assuming some h in $VS_{H,D}$,that does not satisfy the right-hand side

of the expression, then showing that this leads to an inconsistency

**1.10.2 CANDIDATE-ELIMINATION Learning Algorithm**

The CANDIDATE-ELIMINTION algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.

Initialize G to the set of maximally general hypotheses in H

Initialize S to the set of maximally specific hypotheses in

H For each training example d, do

- If d is a positive example
- Remove from G any hypothesis inconsistent with d
- For each hypothesis s in S that is not consistent with d
- Remove s from S

- Add to S all minimal generalizations h of s such that
- h is consistent with d, and some member of G is more general than h

• Remove from S any hypothesis that is more general than another hypothesis in S

- If d is a negative example
- Remove from S any hypothesis inconsistent with d
- For each hypothesis g in G that is not consistent with d
- Remove g from G
- Add to G all minimal specializations h of g such that
- h is consistent with d, and some member of S is more specific than h
- Remove from G any hypothesis that is less general than another hypothesis in G

**An Illustrative Example**

| Example | Sky | AirTemp | Humidity | Wind | Water | Forecast | EnjoySport |
|---|---|---|---|---|---|---|---|
| 1 | Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| 2 | Sunny | Warm | High | Strong | Warm | Same | Yes |
| 3 | Rainy | Cold | High | Strong | Warm | Change | No |
| 4 | Sunny | Warm | High | Strong | Cool | Change | Yes |

CANDIDATE-ELIMINTION algorithm begins by initializing the version space to the set of all hypotheses in H;

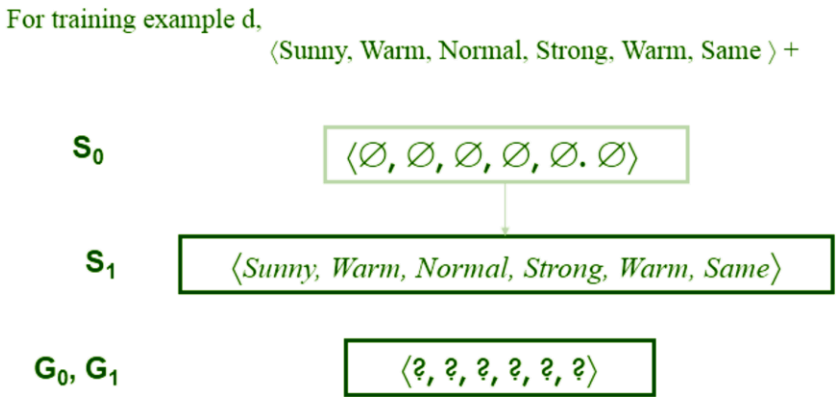
Initializing the G boundary set to contain the most general hypothesis in H

$G_0 < ?, ?, ?, ?, ?, ? >$

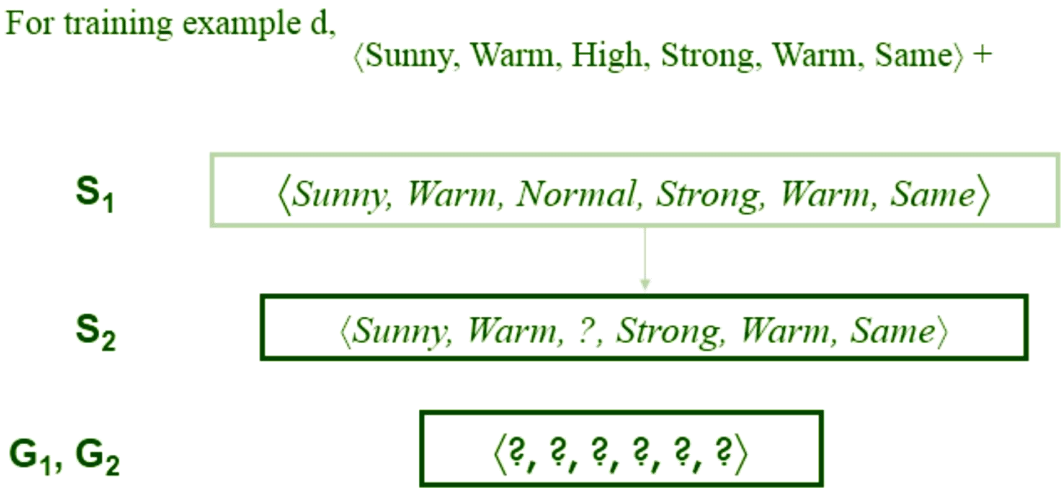
Initializing the S boundary set to contain the most specific (least general) hypothesis

$S_0 < Ø, Ø, Ø, Ø, Ø, Ø >$

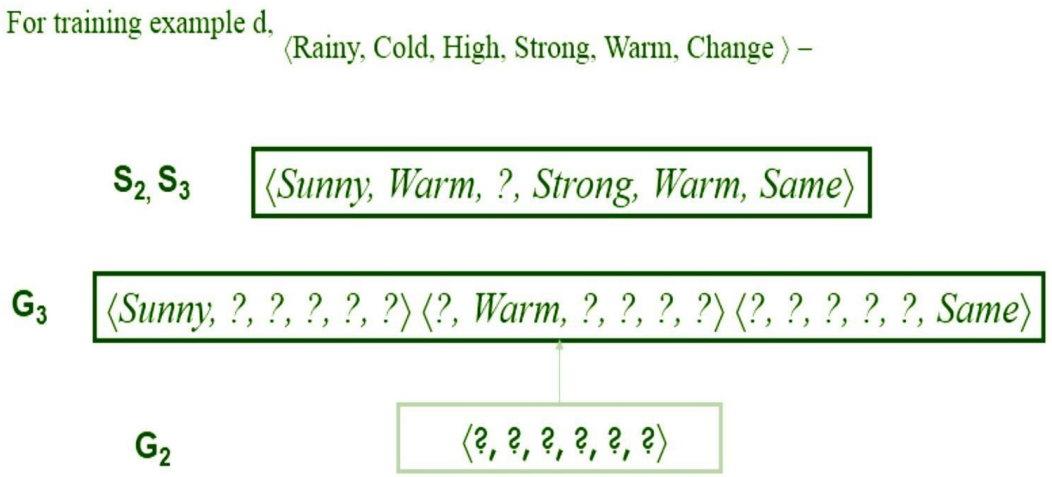
- When the first training example is presented, the CANDIDATE-ELIMINTION algorithm checks the S boundary and finds that it is overly specific and it fails to cover the positive example.
- The boundary is therefore revised by moving it to the least more general hypothesis that covers this new example. No update of the G boundary is needed in response to this training example because Go correctly covers this example




- When the second training example is observed, it has a similar effect of generalizing S further to S2, leaving G again unchanged i.e., G2 = G1 = G0



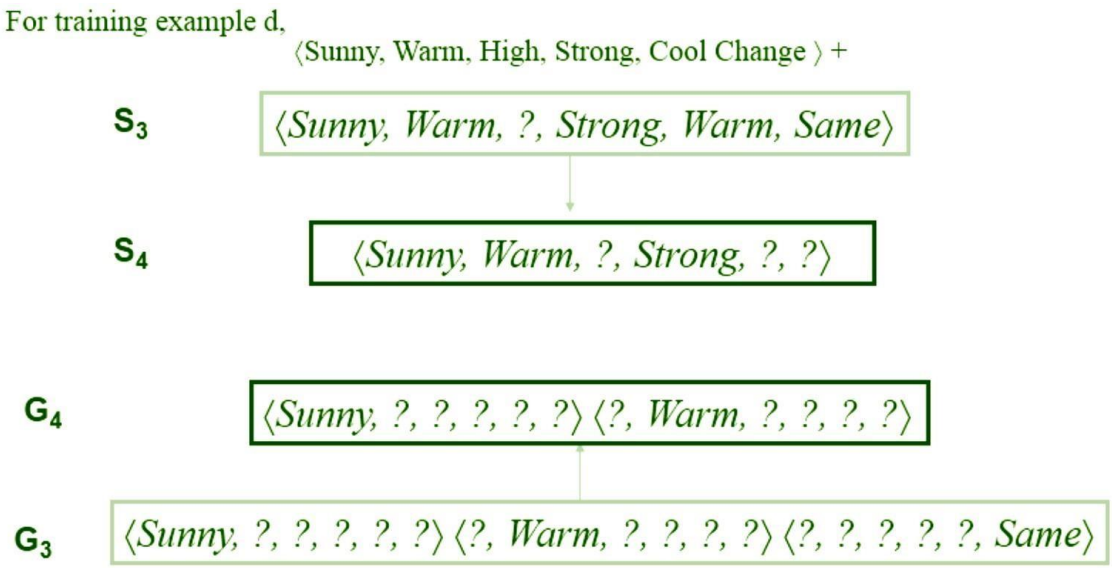

- Consider the third training example. This negative example reveals that the G boundary of the version space is overly general, that is, the hypothesis in G incorrectly predicts that this new example is a positive example.
- The hypothesis in the G boundary must therefore be specialized until it correctly classifies this new negative example




Given that there are six attributes that could be specified to specialize G2, why are there only three new hypotheses in G3?

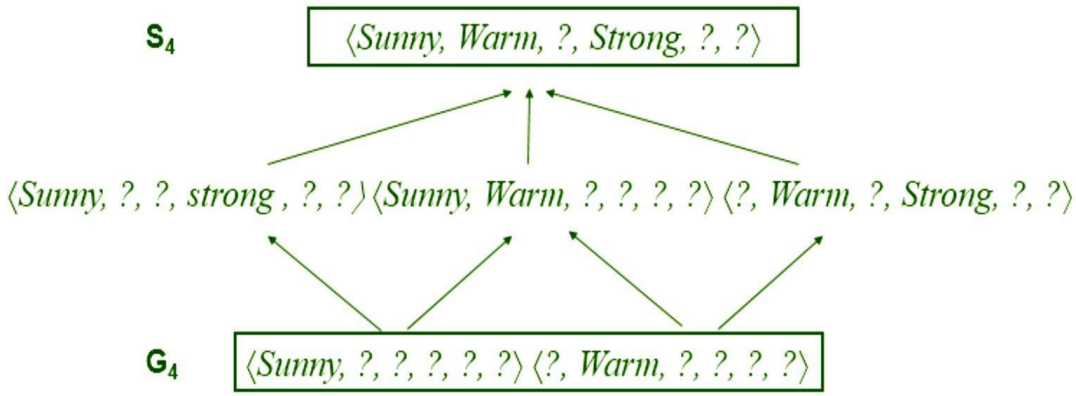
For example, the hypothesis h = (?, ?, Normal, ?, ?, ?) is a minimal specialization of G2 that correctly labels the new example as a negative example, but it is not included in G3. The reason this hypothesis is excluded is that it is inconsistent with the previously encountered positive examples

- **Consider the fourth training example.**




- This positive example further generalizes the S boundary of the version space. It also results in removing one member of the G boundary, because this member fails to cover the new positive example

- After processing these four examples, the boundary sets S4 and G4 delimit the version space of all hypotheses consistent with the set of incrementally observed training examples.




### 1.11 Linear Discriminants

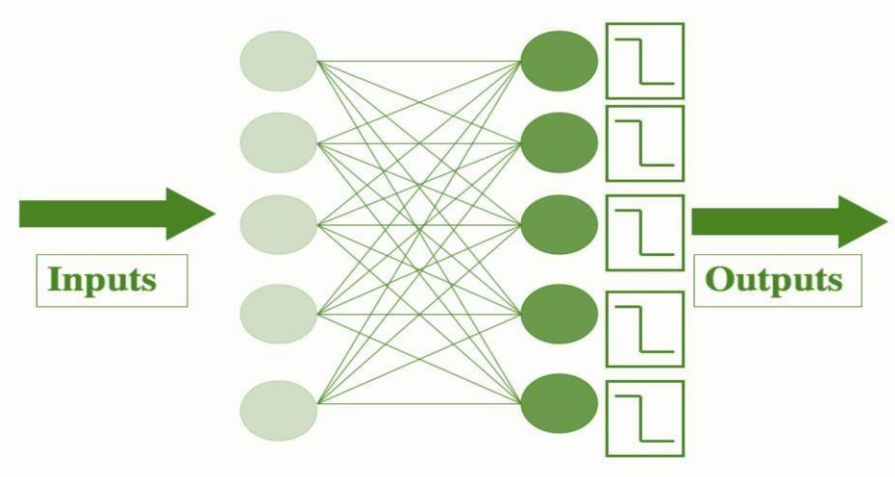
- To make the neuron a little more interesting we need to work out how to make it learn, and then we need to put sets of neurons together into neural networks so that they can do something useful.
- In order to make a neuron learn, the question that we need to ask is:
- How should we change the weights and thresholds of the neurons so that the network gets the right answer more often?
- Once we've worked out the algorithm and how it works, we'll look at what it can and cannot do, and then see how statistics can give us insights into learning as well.

### 1.12 Perceptron

- The Perceptron is nothing more than a collection of McCulloch and Pitts neurons together with a set of inputs and some weights to fasten the inputs to the neurons.
- The network is shown in Figure 1.12.1 On the left of the figure, shaded in light green, are the input nodes. These are not neurons.
- The neurons are shown on the right, and you can see both the additive part (shown as a circle) and the thresholder.
- the neurons in the Perceptron are completely independent of each other:

- it doesn't matter to any neuron what the others are doing, it works out whether or not to fire by multiplying together its own weights and the input, adding them together, and comparing the result to its own threshold, regardless of what the other neurons are doing.

FIGURE 1.12.1The Perceptron network, consisting of a set of input nodes (left) connected to McCulloch and Pitts neurons using weighted connections.




- Even the weights that go into each neuron are separate for each one, so the only thing they share is the inputs, since every neuron sees all of the inputs to the network.
- In Figure 3.2 the number of inputs is the same as the number of neurons. In general there will be m inputs and n neurons.
- When we looked at the McCulloch and Pitts neuron, the weights were labelled as $w_i$, with the i index running over the number of inputs. Here, we also need to work out which neuron the weight feeds into, so we label them as $w_{ij}$ , where the j index runs over the number of neurons.
- So $w_{32}$ is the weight that connects input node 3 to neuron 2. When we make an implementation of the neural network, we can use a two-dimensional array to hold these weights.

- There are m weights that are connected to that neuron, one for each of the input nodes. If we label the neuron that is wrong as k, then the weights that we are interested in are $w_{ik}$, where i runs from 1 to m.
- The first thing we need to know is whether each weight is too big or too small. This seems obvious at first: some of the weights will be too big if the neuron fired when it shouldn't have, and too small if it didn't fire when it should.
- So we compute $y_k - t_k$ (the difference between the output yk, This is a possible error function). If it is negative then the neuron should have fired and didn't, so we make the weights bigger, and vice versa if it is positive, which we can do by subtracting the error value.
- To get around this we'll multiply those two things together to see how we should change
- the weight: The final rule for updating a weight $w_{ij}$ :

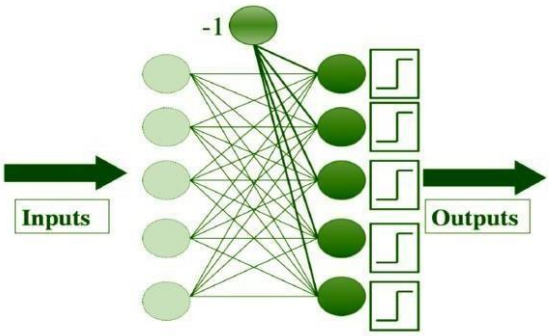
$$w_{ij} \leftarrow w_{ij} - \eta(y_j - t_j) \cdot x_i.$$

### 11.12.1 The Learning Rate η

- The cost of having a small learning rate is that the weights need to see the inputs more often before they change significantly, so that the network takes longer to learn. However, it will be more stable and resistant to noise (errors) and inaccuracies in the data.
- We therefore use a moderate learning rate, typically $0.1 < \boldsymbol{\eta} < 0.4$, depending upon how much error we expect in the inputs.

### 11.12.2 The Bias Input

- Changing the threshold requires an extra parameter.
- Fix the value of the threshold for the neuron at zero. Now, we add an extra input weight to the neuron, with the value of the input to that weight always being fixed (usually the value of -± is chosen; Use -1 to make it stand out, but any non-zero value will do).




**The Perceptron network again, showing the bias input.**

- We include that weight in our update algorithm (like all the other weights), so we don't need to think of anything new. And the value of the weight will change to make the neuron fire—or not fire, whichever is correct—when an input of all zeros is given,
- Since the input on that weight is always -1, even when all the other inputs are zero. This input is called a bias node, and its weights are usually given a 0 subscript, so that the weight connecting it to the $j^{th}$ neuron is $w_{0j}$ .

### 1.12.3 The Perceptron Learning Algorithm

- Initialisation
  - set all of the weights $w_{ij}$ to small (positive and negative) random numbers
- Training
  - for T iterations or until all the outputs are correct:
    * for each input vector:

compute the activation of each neuron j using activation function g:

$$y_j = g\left(\sum_{i=0}^{m} w_{ij}x_i\right) = \begin{cases} 1 & \text{if } \sum_{i=0}^{m} w_{ij}x_i > 0 \\ 0 & \text{if } \sum_{i=0}^{m} w_{ij}x_i \leq 0 \end{cases}$$

update each of the weights individually using:

$$w_{ij} \leftarrow w_{ij} - \eta(y_j - t_j) \cdot x_i$$

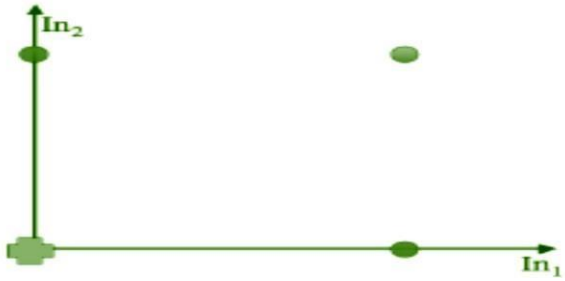
- Recall

– compute the activation of each neuron j using:

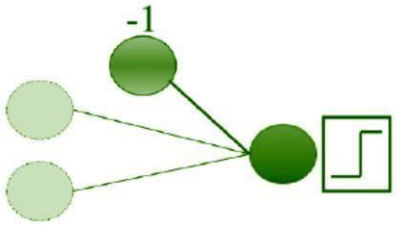
$$y_j = g\left(\sum_{i=0}^{m} w_{ij}x_i\right) = \begin{cases} 1 & \text{if } w_{ij}x_i > 0 \\ 0 & \text{if } w_{ij}x_i \leq 0 \end{cases}$$

### 1.12.4 An Example of Perceptron Learning: Logic Functions

| $In_1$ | $In_2$ | $t$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |




**Data for the OR logic function and a plot of the four datapoints.**




**The Perceptron network for the example**

- There are three weights. The algorithm tells us to initialize the weights to small random numbers, so we'll pick $w_0 = -0.05, w_1 = -0.02, w_2 = 0.02$.
- Now we feed in the first input, where both inputs are 0: (0, 0). Remember that the input to the bias weight is always −1, so the value that reaches the neuron is
- $-0.05 \times -1 + -0.02 \times 0 + 0.02 \times 0 = 0.05$. This value is above 0, so the neuron fires and the output is 1,which is incorrect according to the target.
- In update rule: use $\eta = 0.25$
  - $w_0$ : $-0.05 - 0.25 \times (1 - 0) \times -1 = 0.2$,
  - $w_1$ : $-0.02 - 0.25 \times (1 - 0) \times 0 = -0.02$,
  - $w_2$ : $0.02 - 0.25 \times (1 - 0) \times 0 = 0.02$.
- Feed in the next input (0, 1) and compute the output and then apply the learning rule again:

- $w_0 : 0.2 - 0.25 \times (0 - 1) \times -1 = -0.05$,
- $w_1 : -0.02 - 0.25 \times (0 - 1) \times 0 = -0.02$,
- $w_2 : 0.02 - 0.25 \times (0 - 1) \times 1 = 0.27$.

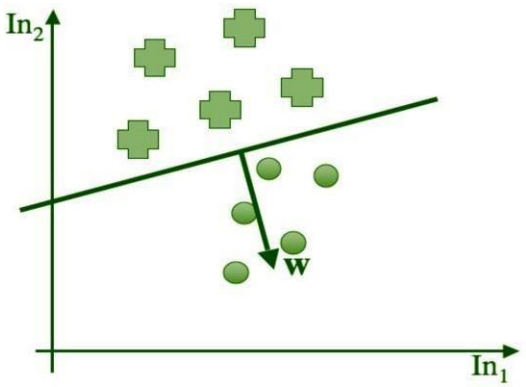
For the (1, 0) input the answer is already correct (you should check that you agree with this), so we don't have to update the weights at all, and the same is true for the (1, 1) input.

We now need to start going through the inputs again, until the weights settle down and stop changing, which is what tells us that the algorithm has finished. For real world applications the weights may never stop changing, which is why yourun the algorithm for some pre-set number of iterations, T.

## 1.13 Linear Separability

What does the Perceptron actually compute?

- It tries to find a straight line (in 2D, a plane in 3D, and a hyperplane in higher dimensions) where the neuron fires on one side of the line, and doesn't on the other. This line is called the decision boundary or discriminant function, and an example of one is given in Figure 1.13.1 A decision boundary separating two classes of data.




- To see this, think about the matrix notation we used in the implementation, but consider just one input vector x. The neuron fires if $x \cdot w^T \geq 0$ (where w is the row of W that connects the inputs to one particular neuron; they are the same for the OR example, since there is only one neuron, and $w^T$ denotes the transpose of w and is used to make both of the vectors into column vectors).

- The a · b notation describes the inner or scalar product between two vector.
- Getting back to the Perceptron, the boundary case is where we find an input vector $x_1$ that has $x_1 \cdot w^T = 0$.
- Now suppose that we find another input vector $x_2$ that satisfies $x_2 \cdot w^T = 0$. Putting these two equations together we get:

$$\begin{aligned} \mathbf{x}_1 \cdot \mathbf{w}^T &= \mathbf{x}_2 \cdot \mathbf{w}^T \\ \Rightarrow (\mathbf{x}_1 - \mathbf{x}_2) \cdot \mathbf{w}^T &= 0. \end{aligned}$$

- So given some data, and the associated target outputs, the Perceptron simply tries to find a straight line that divides the examples where each neuron fires from those where it does not. The cases where there is a straight line are called linearly separable cases.
- What happens if the classes that we want to learn about are not linearly separable?
- It turns out that making such a function is very easy: there is even one that matches a logic function. The weights for each neuron separately describe a straight line, so by putting together several neurons we get several straight lines that each try to separate different parts of the space.
- Figure1.13.2 shows an example of decision boundaries computed by a Perceptron with four neurons; by putting them together we can get good separation of the classes.

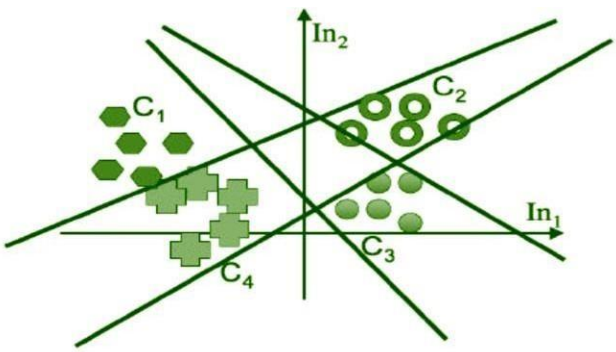



Figure 1.13.2 Different decision boundaries computed by a Perceptron with four neurons.

### 1.13.1 The Perceptron Convergence Theorem

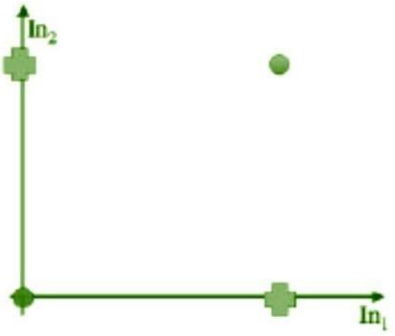
- First, we know that there is some weight vector w* that separates the data, since we have assumed that it is linearly separable. The Perceptron learning algorithm aims to find some vector w that is parallel to w*, or as close as possible.
- To see whether two vectors are parallel we use the inner product $w^* \cdot w$. When the two vectors are parallel, the angle between them is $\Theta = 0$ and so $\cos \Theta = 1$, and so the size of the inner product is a maximum. If we therefore show that at each weight update $w^* \cdot w$ increases, then we have nearly shown that thealgorithm will converge.
- $w^* \cdot w = ||w^*|| \, ||w|| \cos \Theta$, and so we also need to check that the length of w does not increase too much as well.
- Hence, considering a weight update, there are two checks that we need to make: the value of $w^* \cdot w$ and the length of w.
- Suppose that at the $t^{th}$ iteration of the algorithm, the network sees a particular input x that should have output y, and that it gets this input wrong, so $yw^{(t-1)} \cdot x < 0$, where the (t−1) index means the weights at the (t−1)st step.
- This means that the weights need to be updated. This weight update will be $w^{(t)} = w^{(t-1)} + yx$ (where we have set $\eta = 1$ for simplicity, and because it is fine for the Perceptron.

To see how this changes the two values we are interested in, we need to do some computation:

$$\begin{aligned} \mathbf{w}^* \cdot \mathbf{w}^{(t)} &= \mathbf{w}^* \cdot \left( \mathbf{w}^{(t-1)} + y\mathbf{x} \right) \\ &= \mathbf{w}^* \cdot \mathbf{w}^{(t-1)} + y\mathbf{w}^* \cdot \mathbf{x} \\ &\geq \mathbf{w}^* \cdot \mathbf{w}^{(t-1)} + \gamma \end{aligned}$$

Fig.1.13.2 Data for the XOR logic function and a plot of the four datapoints.

| $In_1$ | $In_2$ | $t$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |




The length of the weight vector after t steps is:

$$\begin{aligned} \|\mathbf{w}^{(t)}\|^2 &= \|\mathbf{w}^{(t-1)} + y\mathbf{x}\|^2 \\ &= \|\mathbf{w}^{(t-1)}\|^2 + y^2\|\mathbf{x}\|^2 + 2y\mathbf{w}^{(t-1)} \cdot \mathbf{x} \\ &\leq \|\mathbf{w}^{(t-1)}\|^2 + 1 \end{aligned}$$

### 1.13.2 The Exclusive Or (XOR) Function

- The XOR has the same four input points as the OR function, but looking at Figure 1.13.2, you should be able to convince yourself that you can't draw a straight line on the graph that separates true from false (crosses from circles).
- The XOR function is not linearly separable. If the analysis above is correct, then the Perceptron will fail to get the correct answer, and using the Perceptron code above we find: >>> targets = np.array([[0],[1],[1],[0]])

  >>> pcn.pcntrain(inputs,targets,0.25,15)
- The algorithm does not converge, but keeps on cycling through two different wrong solutions. Running it for longer does not change this behaviour.
- So even for a simple logical function, the Perceptron can fail to learn the correct answer. This is what was demonstrated by Minsky and Papert in "Perceptrons," and the discovery that the Perceptron was not capable of solving even these problems, let alone more interesting ones, is what halted neural network development for so long.

- There is an obvious solution to the problem, which is to make the network more complicated—add in more neurons, with more complicated connections between them, and see if that helps.

### 1.11.4 Linear Regression

- It is common to turn classification problems into regression problems.
- This can be done in two ways, first by introducing an indicator variable, which simply says which class each datapoint belongs to. The problem is now to use the data to predict the indicator variable, which is a regression problem.
- The second approach is to do repeated regression, once for each class, with theindicator value being 1 for examples in the class and 0 for all of the others. Since classification can be replaced by regression using these methods.
- The only real difference between the Perceptron and more statistical approaches is in
- the way that the problem is set up.
- For regression we are making a prediction about an unknown value y (such as the indicator variable for classes or a future value of some data) by computing some function of known values $x_i$.
- So the output y is going to be a sum of the $x_i$ values, each multiplied by a constant parameter:
- Figure 1.11.4 shows this in two and three dimensions.
- Just minimise the sum-of-squares of the errors, then we get the most common minimisation, which is known as least-squares optimisation. What we are doing is choosing the parameters in order to minimise the squared difference between the prediction and the actual data value, summed over all of the datapoints.

- That is, we have: $$\sum_{j=0}^{N}\left(t_j - \sum_{i=0}^{M}\beta_i x_{ij}\right)^2$$

This can be written in matrix form as: $$(\mathbf{t}-\mathbf{X}\beta)^T(\mathbf{t}-\mathbf{X}\beta),$$

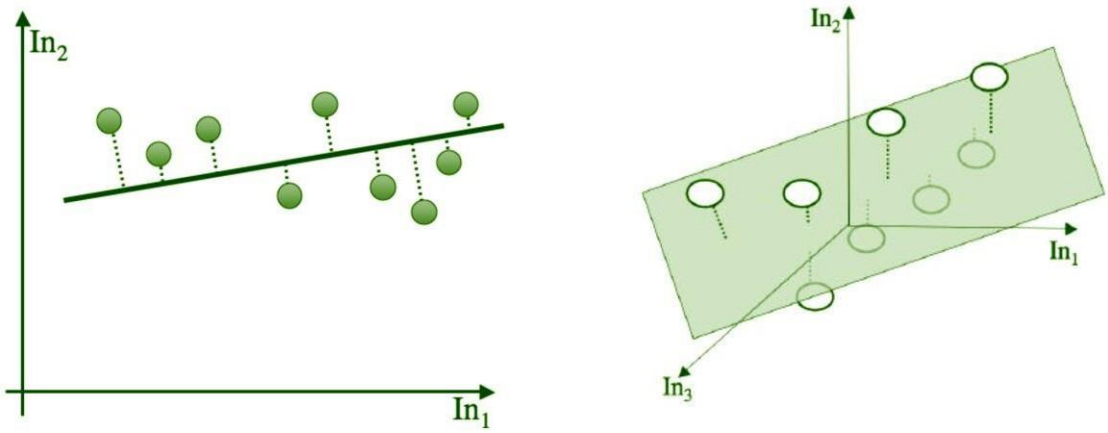



**Fig 1.11.4 Linear regression in two and three dimensions.**

### 1.11.4.1 Linear Regression Examples.

- Using the linear regressor on the logical OR function seems a rather strange thing to do, since we are performing classification using a method designed explicitly for regression, trying to fit a surface to a set of 0 and 1 points.
- Worse, we will view it as an error if we get say 1.25 and the output should be 1, so points that are in some sense too correct will receive a penalty! However, we can do it,and it gives the following outputs:

  [[ 0.25]
  [ 0.75]
  [ 0.75]
  [ 1.25]]

- If we threshold the outputs by setting every value less than 0.5 to 0 and every value above 0.5 to 1, then we get the correct answer. Using it on the XOR function shows that this is still a linear method:

  [[ 0.5]

[ 0.5]

[ 0.5]

[ 0.5]]

- A better test of linear regression is to find a real regression dataset. The UCI database is useful here. Use the auto-mpg dataset. This consists of a collection of a number of datapoints about certain cars (weight, horsepower, etc.), with the aim being to predict the fuel efficiency in miles per gallon (mpg).
- This dataset has one problem. There are missing values in it (labelled with question marks '?'). so after downloading the dataset, manually edit the file and delete all lines where there is a ? in that line. The linear regressor can't do much with the names of the cars either, but since they appear in quotes (")
- You should now separate the data into training and testing sets, and then use the training set to recover the vector.
- Then you use that to get the predicted values on the test set. However, the confusion matrix isn't much use now, since there are no classes to enable us to analyse the results. Instead, we will use the sum-of-squares error, which consists of computing the difference between the prediction and the true value, squaring them so that they are all positive, and then adding them up, as is used in the definition of the linear regressor.

**1. Give precise definition of learning. (CO1,K1)**

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

**2. Find T, P and E for checkers learning problem, handwriting recognition learning problem, robot driving learning problem (CO1,K1)** Checkers learning problem

- Task T: playing checkers
    - Performance measure P: percent of games won against opponents
    - Training experience E: playing practice games against itself. Handwriting recognition learning problem
- Task T: recognizing and classifying handwritten words within images
    - Performance measure P: percent of words correctly classified
    - Training experience E: a database of handwritten words with given classifications Robot driving learning problem
- Task T: driving on public four-lane highways using vision sensors
    - Performance measure P: average distance travelled before an error
    - Training experience E: a sequence of images and steering commands recorded while observing a human driver

**3. Define machine learning. (CO1,K1)**

Machine learning is a subfield of computer science (CS) and artificial intelligence (AI) that deals with the construction and study of systems that can **learn from data**, rather than follow only explicitly programmed instructions. (i.e. Machine learning is the science of getting computers to act without being explicitly programmed)

**4. What is the difference between artificial intelligence and machine learning methods? (CO1,K2)**

Artificial intelligence, and is sometimes known as symbolic processing because the computer manipulates symbols that reflect the environment. In contrast, machine learning methods are sometimes called sub symbolic because no symbols or symbolic manipulation are involved.

**6. What is supervised learning? (CO1,K1)**

A training set of examples with the correct responses (targets) are provided and, based on this training set, the algorithm generalizes to respond correctly to all possible inputs. This is also called learning from exemplars.

**7. What is unsupervised learning? (CO1,K1)**

Correct responses are not provided, instead the algorithm tries to identify similarities between the inputs so that inputs that have something in common are categorized together. The statistical approach to unsupervised learning is known as density estimation.

**8. What is reinforcement learning? (CO1,K1)**

This is somewhere between supervised and unsupervised learning. The algorithm gets told when the answer is wrong, but does not get told how to correct it. It has to explore and try out different possibilities until it works out how to get the answer right. Reinforcement learning sometimes called learning with critic because of this monitor that scores the answer, but does not suggest improvements.

**9. What is evolutionary learning? (CO1,K1)**

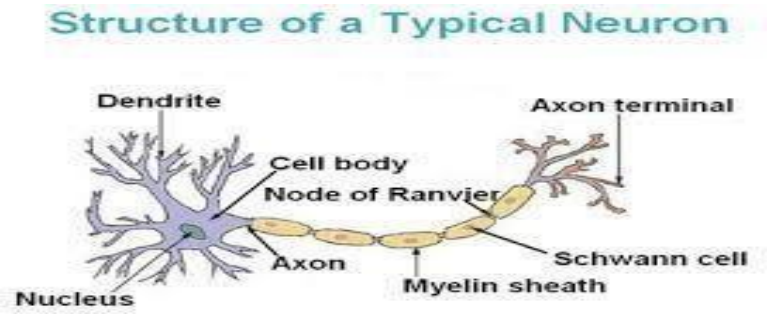
Biological evaluation can be seen as a learning process: biological organisms adapt to improve their survival rates and chance of having offspring in their environment.

**10. What are the basic functions of a neuron? (CO1,K1)**

- Receive signals (or information).

- Integrate incoming signals (to determine whether or not the information should be passed along).
- Communicate signals to target cells
- These neuronal functions are reflected in the anatomy of the neuron.

**11. Draw the basic structure of neuron (CO1,K1)**




**12. List the steps in designing a learning system. (CO1,K1)**

- Choosing the training experience
- Choosing the target function
- Choosing the representation for the target function
- Choosing a function approximation algorithm.
- Estimating training values and Adjusting the weights
- The final design

**13. State Hebb's rule. (CO1,K1)**

Hebb's rule says that the changes in the strength of synaptic connections are proportional to the correlation in the firing of the two connecting neurons. So if two neurons consistency fire simultaneously, then any connection between them will change in strength, becoming stronger. However, if the two neurons never fire simultaneously, the connection between them will die away.

**14. How McCulloch and Pitts modelled a neuron? (CO1,K2)**

McCulloch and Pitts modelled a neuron AS:

- A set of weighted inputs wi that correspond to the synapses

- An adder that sums the input signals (equivalent to the membrane of the cell that collects electrical charge)
- An activation function (initially a threshold function) that decides whether the neuron fires for the current inputs.

**15. Give the formal definition of concept Learning? (CO1,K1)**

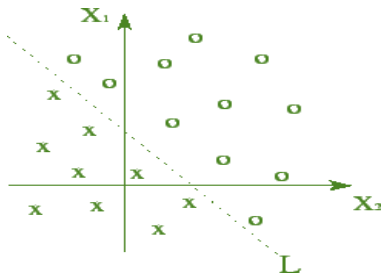
Inferring a boolean-valued function from training examples of its input and output.

**16. What is Perceptron? (CO1,K1)**

The Perceptron is nothing more than a collection of McCulloch and Pitts neurons together with a set of inputs and some weights to fasten the inputs to the neurons.

**17. What is linear separability? (CO1,K2)**

Consider two-input patterns (X1, X2) being classified into two classes (as shown in figure). Each point with either symbol of or represents a pattern with a set of values(X1, X2) . Each pattern is classified into one of two classes. Notice that these classes can be separated with a single line . They are known as linearly separable patterns. Linear separability refers to the fact that classes of patterns with -dimensional vector x = (x1,x2….,xn) can be separated with a single decisionsurface. In the case above, the line represents the decision surface




**18. What is linear regression? (CO1,K1)**

In statistics, linear regression is a linear approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X. The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called

multiple linear regression.

**19. Define the Perspective of Machine Learning. (CO1,K1)**

It involves searching a very large space of possible hypothesis to determine the one that best fits the observed data.

**20. Identify the Issues involved in Machine Learning? (CO1,K2) •**

Which algorithm performs best for which types of problems & representation?

- How much training data is sufficient?
- Can prior knowledge be helpful even when it is only approximately correct?
- The best strategy for choosing a useful next training experience.
- How can learner automatically alter it's representation to improve it's ability to represent and learn the target function?
- What specific function should the system attempt to learn?

## PART - B

1. Define learning. What are the three features of learning? Explain the three features of learning with the following problem. **(CO1,K2)**
    a) Checkers learning problem
    b) Handwriting recognition learning problem
    c) Robot driving learning problem.
2. Explain the various steps in designing a learning system. **(CO1,K2)**
3. Elaborate the Perspectives and issues in machine learning **(CO1,K1)**
4. Explain concept learning task with the example of ENJOYSPORT. **(CO1,K2)**
5. Explain concept learning as search with the example of ENJOYSPORT. **(CO1,K2)**

6. i) Explain in detail the FIND-S: FINDING A MAXIMALLY SPECIFIC HYPOTHESIS.

(ii) Conclude the key properties of FIND-S algorithm **(CO1,K2)**

7. Explain supervised learning with the concept of regression and classification. **(CO1,K2)**

**8.** Draw the neuron structure and explain the various parts. **(CO1,K1)**

**PART - C**

**9.** (i) Assess the Candidate-Elimination algorithm. **(CO1,K2)**

(ii)Explain the candidate elimination algorithm. **(CO1,K2)**

Apply the algorithm to obtain the final version space for the training example **(CO1,K3)**

| SI.No | Sky | Air temp | Humidity | Wind | Water | Forecast | Enjoy sport |
|---|---|---|---|---|---|---|---|
| 1 | Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| 2 | Sunny | Warm | High | Strong | Warm | Same | Yes |
| 3 | Rainy | Cold | High | Strong | Warm | Change | No |
| 4 | Sunny | Warm | High | Strong | Cold | Change | Yes |

**10.** Explain Perceptron learning algorithm with an example. **(CO1,K2)**