CHAPTER 8

Linear Discriminants for Machine Learning

Learning Objectives

At the end of this chapter, you will be able to:

- Explain the concept of linear discriminants
- Describe important models such as perceptron, support vector machine, logistic regression and multi-layer perceptron network

8.1 Introduction to Linear Discriminants

In the previous chapters, we discussed classifiers that work on classes which are separated by linear, non-linear and piece-wise linear boundaries. Specifically, k-nearest neighbor (KNN) and its variants are easy to understand and implement. They can deal with **non-linear decision boundaries between classes**. They typically work on numerical features, even though distances like Gower distance can be used to classify pattern vectors that are of mixed type, that is, they have both numerical and categorical features. They are ideally suited for applications that have low-dimensional small-size training data sets.

Decision tree classifier (DTC) can capture **piece-wise linear boundaries.** It can deal with pattern vectors that have mixed type features. However, it is ideally suited for low-dimensional applications only because of its greedy nature in building the decision tree. Combinational classifiers, based on DTC, like random forest and gradient boost are more versatile and are suited for dealing with large-scale data sets. So, their implementations are popularly used in several practical applications of current interest. DTC and its variants are embedded-feature-selection schemes.

Bayes' classifier and its variants can deal with mixed type data and can handle classes separated by **non-linear boundaries.** Bayes' classifier is optimal and can be used when the underlying probability structure is known. Its variant, the naïve Bayes classifier (NBC), is a simplified version and can work on mixed type data. It deals with classes having **linear decision boundaries.** It can also be used as an embedded method for feature selection. All these models can also be used for function prediction or regression.

In this chapter, we deal with a collection of ML models that are ideally suited for dealing with large-scale data sets; they are chosen when the classes are **linearly separable**. They are designed to deal with classification of patterns that have only numerical features as these models depend on

If you are unable to redeem the licence key, you may have a pirated or second-hand book. To buy an original book with a valid licence key, please go to www.orientblackswan.com or contact customercare@orientblackswan.com.

dot product computation between pattern vectors. Learning in these cases involves starting with some weight structure and updating the weights based on training data. The specific models are:

- **Perceptron:** It is the earliest ML model and also forms the building block for deep neural networks that are popular state-of-the-art ML models. It provides the appropriate decision boundary in a reasonable time when the classes are linearly separable. It can be used when the classes are not linearly separable, provided the functional form of the non-linearity is known.
- **Support Vector Machine (SVM):** This can be viewed as an artificial neural network but a major strength of SVM is that it offers global optimal solutions to a well-formulated optimization problem that maximizes the margin of separation between two classes. The simple model can be extended to deal with even non-linear decision boundaries between classes with the help of the well-known *kernel trick*.
  It captures the non-linear boundaries in the given input data by viewing it as though the data is projected into a higher-dimensional, theoretically even infinite-dimensional, space called the feature space, where the classes are linearly separable. Under some acceptable conditions, it permits us to make all the computations in the low-dimensional input space instead of in the high-dimensional feature space.
- **Logistic Regression:** A simple interpretation is that logistic regression is a variant of perceptron. Perceptron typically employs a threshold function, the linear threshold function, at the output that gives a binary output of 0 or 1, whereas logistic regression employs a function that can provide values in the range [0,1]; these values can be interpreted as probabilities. A more general view is that logistic regression is a linear model; this generalized linear model enables us to deal with non-linear boundaries.
- **Artificial Neural Networks (ANNs):** An ANN is made up of multiple levels of perceptrons that learn the required input–output mapping. It employs a simple gradient-descent scheme for learning or updating the weights. An edge between a pair of neurons is associated with a weight; learning in ANNs pertains to learning the appropriate weight matrix/structure. It is said to have the capacity to learn the features automatically, unlike the other ML models.

We will study all these ML models in this chapter. One unifying thread behind all these models is the notion of a **linear discriminant function**, which we consider next.

8.2 Linear Discriminants for Classification

Let $w = (w(1), w(2), \ldots, w(l))$ be an l-dimensional weight vector and let $x = (x(1), x(2), \ldots, x(l))$ be a pattern represented as a feature vector. Then

$$g(w, x) = \sum_{i=1}^{l} w(i)x(i) + b$$

is called a **linear discriminant function**, where b is a scalar that is called the **bias or threshold**. Sometimes, $w(0)$ is used instead of b in the literature. However, b is more popular, so we use b in this chapter.

Example 1: Let $x(1)+x(2) > 0$ or $x(1)+x(2)-0.5 > 0$ be a linear discriminant function; here $w = (1, 1)$ and $b = 0$ for the former and $w = (1, 1)$ and $b = -0.5$ for the latter.

Both of them represent the Boolean OR function. Consider Fig. 8.1, which shows the Boolean OR on the left and the Boolean AND on the right. We can appreciate them better by looking at their truth table, given in Table 8.1.

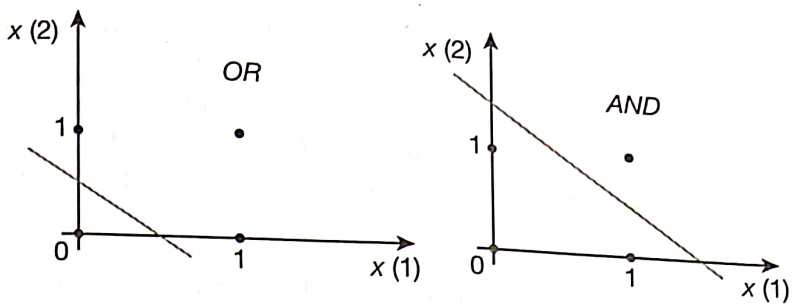

FIG. 8.1 Boolean functions: *OR* and *AND*

TABLE 8.1 Truth table for Boolean *OR* and *AND*

$x(1)$	$x(2)$	$OR(x(1), x(2))$	$x(1)+x(2)-0.5>0$	$AND(x(1), x(2))$	$x(1)+x(2)>1.5$
0	0	0	0	0	0
0	1	1	1	0	0
1	0	1	1	0	0
1	1	1	1	1	1

Here, $x(1)$ and $x(2)$ are binary features taking values of either 1 (*True*) or 0 (*False*). Note that $OR(x(1), x(2))$ is 0 only when both $x(1)$ and $x(2)$ are 0, else it is 1.

Consider $x(1)+x(2)-0.5>0$. It is 0 (*False*) only when both $x(1)$ and $x(2)$ are 0, else it is 1 (*True*). So, Boolean *OR* is captured by $x(1)+x(2)-0.5>0$. Similarly, $AND(x(1), x(2))$ is 1 (*True*) only when both $x(1)$ and $x(2)$ are 1 (*True*), else it is 0. Now consider $x(1)+x(2)>1.5$. It is 1 (*True*) only when both $x(1)$ and $x(2)$ are 1, else 0. So, Boolean *AND* is captured by $x(1)+x(2)>1.5$.

Note that in Fig. 8.1, output value 1 is shown by a black dot and output value 0 is indicated by a grey dot for both *OR* and *AND* functions. In each case, the line that separate 0s from 1s is the decision boundary. It separates the two classes of points; points falling above the line are 1s and those falling below are 0s.

In the case of *OR*, the decision boundary (line) characterizes points that satisfy $x(1)+x(2)=0.5$. Points that give output 1 satisfy the property that $x(1)+x(2)>0.5$ and a point for which output is 0 satisfies $x(1)+x(2)<0.5$. In the case of *AND*, a point, $x=(x(1), x(2))$, on the line satisfies $x(1)+x(2)=1.5$. A point that gives 1 as output satisfies $x(1)+x(2)>1.5$ and points that give 0 as output satisfy $x(1)+x(2)<1.5$.

Note that there could be infinite ways of representing the *OR* and *AND* functions. $OR(x(1), x(2))$ can be represented using $x(1)+x(2)-\alpha>0$, where $\alpha \in [0,1)$; α can take any value in the interval [0,1] excluding the value 1. Similarly, $AND(x(1), x(2))$ can be represented using $x(1)+x(2)>\beta$, where $\beta \in [1,2)$; β can take any value in the interval [1,2] excluding the value 2.

8.2.1 Parameters Involved in the Linear Discriminant Function

The generic form of any linear discriminant function is

$$g(w,x)=w^t x+b=\sum_{i=1}^{l} w(i)x(i)+b$$

It is linear because the components $x(i)$ are used in their linear form, that is, with exponent 1. There are no non-linear terms like the product $x(1)x(2)$ or terms with exponents like $x(1)^2$. $g(x) = 0$ or equivalently $w^t x + b = 0$ characterizes the *decision boundary* between the *positive class* ($g(x) > 0$) and the *negative class* ($g(x) < 0$). So, it can be used as a linear classifier; once we know the weight vector w and the threshold or bias b, we decide a given x to be from the positive class if $w^t x + b > 0$, else x is from the negative class.

EXAMPLE 2: Let $w = (-1, 2)$ and $b = 1$ for a linear discriminant. Now we can classify patterns as follows:

Let $x_1 = (1, 1)$. $g(w, x_1) = w^t x_1 + b = -1 \times 1 + 2 \times 1 + 1 = 2$. So, $g(w, x_1) > 0 \Rightarrow$ Assign x_1 to the positive class.

Let $x_2 = (4, 1)$. $g(w, x_2) = w^t x_2 + b = -1 \times 4 + 2 \times 1 + 1 = -1$. So, $g(w, x_2) < 0 \Rightarrow$ Assign x_2 to the negative class.

We can visualize the boundary and other regions associated with the linear discriminant function as shown in Fig. 8.2.

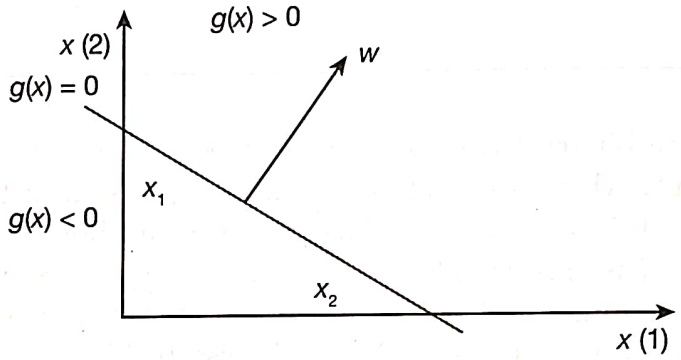

FIG. 8.2 Geometric visualization of the linear discriminant function

Some observations:

- It corresponds to two-dimensional data classification.
- The grey line indicates the decision boundary between two classes. Any point x on this line satisfies the property $g(x) = w^t x + b = 0$. In a higher dimensional space, that is, when the number of features is more than 2, the decision boundary will be a hyperplane.
- The decision boundary splits the whole space into two halves that are characterized by $g(x) > 0$ and $g(x) < 0$. Any point x in the *positive half space* satisfies the property that $g(x) > 0$. Similarly, any point x in the *negative half space* satisfies the property that $g(x) < 0$.
- Consider the two points x_1 and x_2 located on the decision boundary.

$$g(x_1) = g(x_2) = 0 = w^t x_1 + b = w^t x_2 + b \Rightarrow w^t (x_1 - x_2) = 0$$

This means w is orthogonal to $(x_1 - x_2)$ and $x_1 - x_2$ is the line joining the two points x_1 and x_2. So, w is orthogonal to the decision boundary.

- Consider the origin, that is, $x = 0$. So, $g(x) = w^t x + b = b$. So, $g(x) = b$.
  If $b > 0$, then origin is in the positive half space as $g(0) = b > 0$.
  If $b < 0$, then origin is in the negative half space as $g(0) = b < 0$.
  If $b = 0$, then origin is on the decision boundary as $g(x) = b = 0$.
- Consider the case where $b = 0$, as shown in Fig. 8.3. Consider the points x_3 and x_4. Let α be the angle between w and x_3 and β be the angle between w and x_4, as shown in the figure.

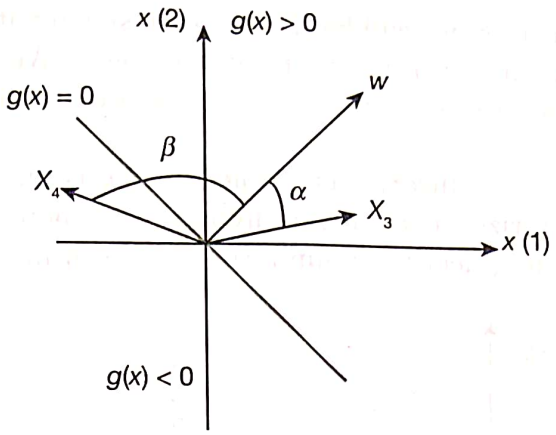

FIG. 8.3 Decision boundary passing through the origin ($b = 0$)

Note that x_3 is in the positive half space, so $g(x_3) = w^t x_3 > 0$ (as $b = 0$). We know that

$$cos\alpha = \frac{w^t x_3}{||w||||x_3||} \Rightarrow cos\alpha > 0 \Rightarrow |\alpha| < 90,$$

because the numerator is positive and both terms in the denominator are norms and so are positive. This holds good for any x in the positive half space. The angle between w and such an x is in the range $[-90,90]$.
However, x_4 is in the negative half space. So, $g(x_4) = w^t x_4 < 0$.

$$cos\beta = \frac{w^t x_4}{||w||||x_4||} \Rightarrow cos\beta < 0 \Rightarrow |\beta| > 90$$

So, w is orthogonal to the decision boundary and points towards the positive half space, as shown in Fig. 8.3.

- On the other hand, b indicates the location of the decision boundary. If $b = 0$, then the decision boundary passes through the origin. If $b > 0$, then $g(0) = w^t 0 + b = b > 0$. So, the origin will be in the positive half space. Similarly, if $b < 0$, then the origin will be in the negative half space.

EXAMPLE 3: Consider a linear discriminant function $g_1(x) = x(1) - x(2) = w^t x + b$, where $x = \begin{pmatrix} x(1) \\ x(2) \end{pmatrix}$, $w = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$ and $b = 0$. Here, a two-dimensional vector x such that

- $x(1) = x(2)$ will be on the decision boundary as $g_1(x) = 0$.
- $x(1) < x(2)$ will be in the negative half space as $g_1(x) < 0$.
- $x(1) > x(2)$ will be in the positive half space as $g_1(x) > 0$.

8.2.2 Learning w and b

We have seen that the weight vector w and b can be used to classify a test pattern x. However, such a classification scheme needs w and b to be obtained; a learning scheme is used to get these entities. This learning is achieved with the help of training data.

Instead of learning w and b separately, one can learn them together in one shot by augmenting. Note that $w^t x + b$ can be equivalently written as $w_a^t x_a$, where $w_a = [b, w]$ and $x_a = [1, x]$. That is,

if w is an l-dimensional vector, then w_a will be $(l+1)$-dimensional with the first component being b; similarly, x_a is $(l+1)$-dimensional with the first entry being 1. We use x for x_a and w for w_a for the sake of brevity; whether we are referring to the augmented vectors or the original vectors will be clear from the context.

So, $\{(x_i, y_i), i = 1, 2, \ldots, n\}$ is linearly separable if there is w_* such that $w_* x_i > 0$ if $y_i = 1$, else $y_i = 0$. Such a w_* characterizes a separating hyperplane; there could be infinitely many such w_* vectors and the corresponding decision boundaries as shown in Fig. 8.4.

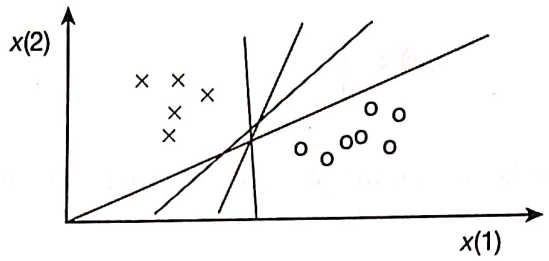

FIG. 8.4 Multiple decision boundaries (for colour figure, please see Colour Plate 2)

EXAMPLE 4: Consider $\{((1,1), l_1 = 0), ((2,2), l_2 = 0), ((4,4), l_3 = 1)\}$, where the first two vectors are from class 0 and the third one is from class 1.

The augmented vectors in the resulting three-dimensional space are $x_1 = (1,1,1), x_2 = (1,2,2), x_3 = (1,4,4)$. A possible augmented w is $(-6,1,1)$ that can separate the three patterns. Note that $w^t x_1 = -4(< 0)$, $w^t x_2 = -2(< 0)$ and $w^t x_3 = 2(> 0)$.

All the three are correctly classified; the first two patterns are from the negative half space (class 0) and the third from the positive half space (class 1). It is possible to check if $(-12, 2, 3)$ also correctly classifies all the three patterns. We leave it as an exercise.

We can compute the normal distance of a data point x from the decision boundary, as shown in Fig. 8.5. Let r be the magnitude of the distance between x and the decision boundary. We can view the vector x as a sum of two vectors x_p and $\frac{rw}{||w||}$, where x_p is the orthogonal projection of x onto the decision boundary and r is the unit of distance of point x from the decision boundary.

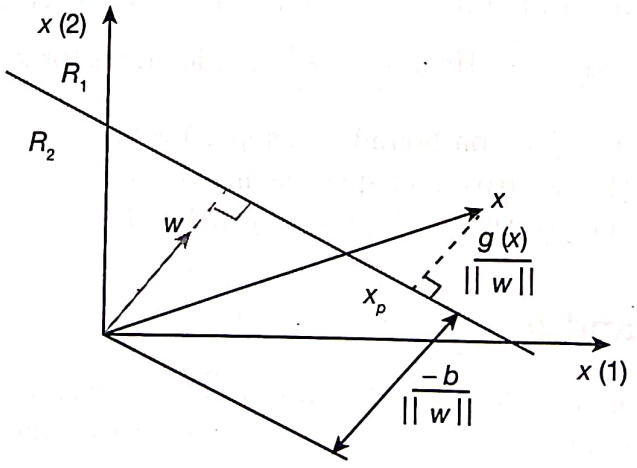

FIG. 8.5 Distance of a point from the decision boundary (for colour figure, please see Colour Plate 2)

The quantity $\frac{w}{||w||}$ gives us the unit vector in the direction of w. So, $r \times \frac{w}{||w||} = \frac{rw}{||w||}$ indicates the vector which when added to x_p gives us x.

We know that $g(x) = w^t x + b = w^t x_p + b + w^t \frac{rw}{||w||} = 0 + r||w||$ as x_p is on the decision boundary and so $w^t x_p + b = 0$. So, $r||w|| = g(x) \Rightarrow r = \frac{g(x)}{||w||}$.

8.3 Perceptron Classifier

Perceptron is the earliest ML model that was used in classifying linearly separable data. Some related observations are as follows:

- We learn the augmented weight vector w with the help of some two-class training data.
- If the classes are linearly separable, then the perceptron learning algorithm will learn the augmented weight vector in a finite number of steps.
- The obtained weight vector w will classify any augmented pattern x to the negative class if $w^t x < 0$ and assign x to the positive class if $w^t x > 0$. We assume that the final w will classify every training pattern to one of the two classes and there is no x that gives $w^t x = 0$.
- It is likely that there are infinite possible solutions to the learning problem; that is, there could be possibly infinite w vectors. The perceptron learning algorithm will find one of them. It is based on starting with an initial weight vector and updating it until all the training patterns are correctly classified.
- If the weight vector at step k, w_k, misclassifies an x from the negative class, that is, $w_k^t x > 0$, then it is updated as $w_{k+1} = w_k - x$. Consider $w_{k+1}^t x = w_k^t x - x^t x$. So, $w_{k+1}^t x$ can be negative even if $w_k^t x$ is positive because $x^t x \geq 0$ and $w_{k+1}^t x \leq w_k^t x$.
- If the weight vector at step k, w_k, misclassifies an x from the positive class, that is, $w_k^t x < 0$, then it is updated as $w_{k+1} = w_k + x$. Consider $w_{k+1}^t x = w_k^t x + x^t x$. So, $w_{k+1}^t x$ can be positive even if $w_k^t x$ is negative because $x^t x \geq 0$ and $w_{k+1}^t x \geq w_k^t x$.
- So, this update scheme to get w_{k+1} from w_k is justified because w_{k+1} is in a better position, compared to w_k, to classify x correctly.
- There could be different ways of selecting the initial weight vector w_0. But a simple choice for theoretical analysis is to let $w_0 = 0$; so w_0 is chosen to be the zero vector initially.

8.3.1 Perceptron Learning Algorithm

The algorithm is very simple and it can be shown that the algorithm stops after a finite number of steps to give us a correct w if the classes are linearly separable.

Let the training data set be $\{(x_1, l_1), (x_2, l_2), \ldots, (x_n, l_n)\}$, where x_j is the j^{th} pattern, l_j is its class label and $l_j \in \{-1, +1\}$ based on whether x_j is from the negative class or positive class. The algorithm follows the steps given below:

1. Let $i = 0$. Let the initial augmented weight vector w_i be the zero vector. Let $j = 1$. (Note that j is the pattern index and i is the weight vector index).
2. If x_j is incorrectly classified by w_i, then $i = i + 1$ and $w_i = w_{i-1} + l_j x_j$, set $j = (j + 1) mod\ n$ and go to step 2. Else set $j = (j + 1) mod\ n$ and go to Step 2.
3. Terminate the algorithm if there is no change in weight vector w_i for n successive steps.

Some observations:

- x_j is incorrectly classified by w_i (Step 2) if $l_j(w_i^t x_j) \leq 0$).
- Updating j in steps 2 and 3 involves using addition modulo n, that is, after considering the n^{th} pattern, x_n, it considers the first pattern, x_1. So, it considers the data points cyclically till the termination condition, specified in Step 3, is satisfied.
- If point x_j from the negative class is misclassified, then $w_i = w_{i-1} - x_j$ as $l_j = -1$ and if x_j from the positive class is misclassified, then $w_i = w_{i-1} + x_j$ as $l_j = +1$.
- If there is no update for n successive steps, it means that the current w_i has classified all the patterns correctly and so the algorithm did not visit Step 2 in the recent n steps.
- If a pattern is correctly classified, then we do not update the weight vector and consider the next pattern in the sequence cyclically.

We illustrate the working of the algorithm using an example.

EXAMPLE 5: Let the training set be $\{((-1,1), l_1 = -1), ((-2,-2), l_2 = -1), ((4,4), l_3 = 1)\}$. So, the augmented patterns are $(1,-1,1)$, $(1,-2,-2)$ and $(1,4,4)$ with their respective labels.

1. $w_0 = (0,0,0)$. It misclassifies $(1,-1,1)$ as the dot product between w_0 and $(1,-1,1)$ is 0 (we expect it to be less than 0). So, $w_1 = w_0 - (1,-1,1) = (-1,1,-1)$.
2. w_1 correctly classifies the next pattern $(1,-2,-2)$ as the dot product is -1 (< 0).
3. w_1 misclassifies $(1,4,4)$ as the dot product with $(1,4,4)$ is -1 (we expect it to be greater than 0). So, $w_2 = w_1 + (1,4,4) = (-1,1,-1) + (1,4,4) = (0,5,3)$.
4. Note that $w_2 = (0,5,3)$ correctly classifies all the three augmented patterns as
   - The dot product of w_2 with $(1,-1,1)$ is -2 (< 0).
   - The dot product of w_2 with $(1,-2,-2)$ is -16 (< 0).
   - The dot product of w_2 with $(1,4,4)$ is 32 (> 0).
5. So, the algorithm terminates and gives us the weight vector $(0,5,3)$.

We consider one more example. It deals with learning of Boolean *OR*.

EXAMPLE 6: Consider the truth table of Boolean *OR* shown in the first three columns of Table 8.1. The inputs are $x(1)$ and $x(2)$ and the output is either a 0 or 1; we have a two-class problem based on these two output values. So, there is one pattern from Class 0 and three patterns from Class 1. The augmented patterns are shown in Table 8.2.

TABLE 8.2 Augmented patterns for Boolean *OR*

1	$x(1)$	$x(2)$	$OR(x(1), x(2))$
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

We show the details of the updating of the weight vector starting with $w_0 = (0,0,0)$ in Table 8.3. It takes 9 steps of weight updates to give the weight vector $w_9 = (-1,2,2)$ which classifies all the four augmented patterns correctly.

TABLE 8.3 Updating the weight vector

Weight vector	Pattern misclassified	$w^t x$	Weight vector	Pattern misclassified	$w^t x$
$w_0 = (0,0,0)$	(1,0,0)	0	$w_1 = (-1,0,0)$	(1,0,1)	-1
$w_2 = (0,0,1)$	(1,1,0)	0	$w_3 = (1,1,1)$	(1,0,0)	-1
$w_4 = (0,1,1)$	(1,0,0)	0	$w_5 = (-1,1,1)$	(1,0,1)	0
$w_6 = (0,1,2)$	(1,0,0)	0	$w_7 = (-1,1,2)$	(1,1,0)	0
$w_8 = (0,2,2)$	(1,0,0)	0	$w_9 = (-1,2,2)$	none	*correct* $\forall x$

Note that in the process, the algorithm subtracted the pattern $(1,0,0)$ five times, added the pattern $(1,1,0)$ twice and the pattern $(1,0,1)$ twice to the initial weight vector w_0. So, $w = w_9 = (0,0,0) - 5(1,0,0) + 2(1,1,0) + 2(1,0,1) = (-1,2,2)$. So, it is convenient to view the learnt vector as a linear combination of the training data points. Generically, $w = \sum_{i=1}^{n} \alpha_i l_i x_i$, where $\alpha_i \geq 0$. Note that in Example 1,

- $x_1 = (1,0,0), \alpha_1 = 5,$ *and* $l_1 = -1$
- $x_2 = (1,0,1), \alpha_2 = 2,$ *and* $l_2 = 1$
- $x_3 = (1,1,0), \alpha_3 = 2,$ *and* $l_3 = 1$
- $x_4 = (1,1,1), \alpha_4 = 0,$ *and* $l_4 = 1$

8.3.2 Convergence of the Learning Algorithm

It is possible to show that the learning algorithm finds the correct weight vector, a weight vector that classifies all the patterns correctly, if the classes are linearly separable.

Let the augmented patterns be $x_1, x_2, \ldots, x_n$ with their respective class labels being $l_1, l_2, \ldots, l_n$ and $l_i \in \{-1, +1\}$ for all i. If they are linearly separable, then there exists a vector w that correctly classifies all the patterns. That is, if x is in the positive class, then $w^t x > 0$, and if x is in the negative class, then $w^t x < 0$. In either case, $w^t l x > 0$, where l is the class label of x because $l = +1$ if x is in the positive class and $l = -1$ if x is in the negative class.

We have $w_0 = 0$. Let the first pattern misclassified by w_0 be x^0 and let $l(0)$ be its class label. So, $w_1 = w_0 + l(0)x^0$. In general, if augmented pattern x^{k-1} is misclassified by w_{k-1}, then $w_k = w_{k-1} + l(k-1)x^{k-1}$. By expanding recursively, we have $w_k = w_0 + l(0)x^0 + l(1)x^1 + \cdots + l(k-1)x^{k-1}$, where $w_0 = 0$.

Considering the dot product between the correct w and w_k, we have

$$w^t w_k = w^t(l(0)x^0 + l(1)x^1 + \cdots + l(k-1)x^{k-1}$$

Let $\alpha_i = w^t l(i) x^i$. So, $w^t w_k = \alpha_0 + \alpha_1 + \cdots + \alpha_{k-1}$. We know that $\alpha_i > 0$ for $i = 0, 1, \ldots, k-1$ because of w. So, $w^t w_k > 0$.

Let α_p be the minimum$(\alpha_0, \alpha_1, \ldots, \alpha_{k-1})$. So, $w^t w_k > k\alpha_p$. Consider $w_k^t w_k = l(0)^2||x^0||^2 + l(1)^2||x^1||^2 + \cdots + l(k-1)^2||x^{k-1}||^2 = ||x^0||^2 + ||x^1||^2 + \cdots + ||x^{k-1}||^2$ as $l(i)^2 = 1$ for $i = 0, 1, \ldots, k-1$.

Let $\beta_i = ||x^i||^2$ for $i = 0, 1, 2, \ldots, k-1$. So, clearly $\beta_i > 0$ for all i. So, $w_k^t w_k = \beta_0 + \beta_1 + \cdots + \beta_{k-1}$ is positive. Let $\beta_q = maximum(\beta_0, \beta_1, \ldots, \beta_{k-1})$. Then $w_k^t w_k < k\beta_q$.

Let θ be the angle between w and w_k. We know that $cos\theta = \frac{w^t w_k}{||w||||w_k||}$. Note that both the numerator and each of the two terms in the denominator is positive as $w^t w_k > 0$; this is because

C_3 vs $\bar{C}_3$: Let x be assigned to $\bar{C}_3$.
C_4 vs $\bar{C}_4$: Let x be assigned to $\bar{C}_4$.

We decide to assign x to class C_1.

This scheme can also have problems:

- There could be ties. For example, in step d of Example 9, if the decision had gone in favour of C_4, there would have been a tie in taking majority voting between C_1 and C_4.
- If the number of classes C is large, there can class imbalance between each C_i and $\bar{C}_i$. Specifically, a class C_i could be a minority class having less than $\frac{1}{C}$ fraction of training data points and $\bar{C}_i$, the majority class, could have more than $\frac{C-1}{C}$ fraction. This leads to problems while using several classifiers.

Even though these techniques have been well understood for more than four decades, it is support vector machines (SVMs) (binary classifiers) that have popularized them.

8.4 Support Vector Machines

We have seen in Fig. 8.4 that there could be theoretically infinite decision boundaries if the classes are linearly separable. Perceptron selects one of them. However, if we want to select a decision boundary that is evenly centred between the two class boundaries, then support vector machine (SVM) is the best choice to obtain the optimal solution. Consider Fig. 8.6.

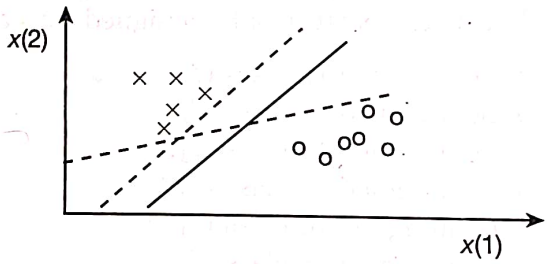

Fig. 8.6 Bad and good decision boundaries

There are two classes in a two-dimensional space, specified by $x(1)$ and $x(2)$. There are 5 Xs from the negative class and 7 circles from the positive class. These two classes are linearly separable.

There are two decision boundaries indicated by broken lines. Both of them are bad; one of them is very close to the negative class and the other one is unable to exploit the margin of separation between the two classes. The third decision boundary shown by a solid line appears to be good as it evenly splits the margin of separation between the two classes.

The SVM classifier formulates an optimization problem to maximize the margin between the two classes; the decision boundary will be evenly positioned between the two classes. Some of its important features are as follows:

- It considers a two-class problem with class label $y_i \in \{-1, +1\}$ for x_i in the training set for $i = 1, 2, \ldots, n$; if $y_i = -1$, then x_i is from the negative class and if $y_i = +1$, then x_i is from the positive class.

- Any point x on the decision boundary will satisfy $w^t x + b = 0$, where w is the weight vector and b is the bias.
- For any point x from the negative class, $w^t x + b < 0$ and any point x from the positive class, $w^t x + b > 0$.
- There is a finite number, n, of training patterns. So, one can identify an $\epsilon > 0$ such that:

  $w^t x_i + b \geq \epsilon$ for all i with $y_i = 1$
  $w^t x_i + b \leq -\epsilon$ for all i with $y_i = -1$

- By dividing both sides of the two inequalities by ϵ, we get

  $w'^t x + b' \geq 1$ for all i with $y_i = 1$
  $w'^t x + b' \leq -1$ for every x for all i with $y_i = -1$, where $w' = \frac{1}{\epsilon} w$ and $b' = \frac{b}{\epsilon}$
- We can combine the two inequalities into one by exploiting the value of $y_i \in \{-1, 1\}$. The resulting inequality is

$$y_i(w^t x_i + b) \geq 1, \forall i$$

  Note that we have dropped the superscripts for w and b for the sake of brevity. So, we have $w^t x + b = 1$ for a boundary pattern x from the positive class and $w^t x + b = -1$ for a boundary pattern x from the negative class.
  These two equalities characterize two parallel lines when w and x are two-dimensional vectors and they correspond to two parallel planes if the vectors are higher-dimensional (dimensionality > 2). The data points closest to the hyperplanes are the *support vectors.*
- The distance from any point x on a plane $w^t x + b = 1$ to the decision boundary given by $w^t x + b = 0$ is $\frac{1}{||w||}$, as discussed in Section 8.2.2. Similarly, the distance from any point x on plane $w^t x + b = -1$ to the decision boundary is $\frac{1}{||w||}$. So, the total distance between the two parallel planes is $\frac{2}{||w||}$. This is called the **margin.**
- The margin is maximized by SVM. This optimization problem has constraints in the form

$$y_i(w^t x_i + b) \geq 1, \ \forall i$$

  Instead of maximizing $\frac{2}{||w||}$, we can minimize $\frac{||w||^2}{2}$. So, the optimization problem with constraints is *minimize* $\frac{1}{2} \mid\mid w \mid\mid^2$ *subject to* $1 - y_i(w^t x_i + b) \leq 0$ *for* $i = 1, \ldots, n$.
  The Lagrangian is $\mathcal{L} = \frac{1}{2} w^t w + \sum_{i=1}^{n} \alpha_i (1 - y_i(w^t x_i + b))$.
- The necessary and sufficient conditions for a (w, b) pair to be the optimal solution are (by setting the gradient of $\mathcal{L}$ with respect to w to zero, we get)

$$w + \sum_{i=1}^{n} \alpha_i (-y_i) x_i = 0 \Rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

$$\frac{\delta \mathcal{L}}{\delta b} = 0 \Rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$1 - y_i(w^t x_i + b) \leq 0, \ \forall i$$

$$\alpha_i \geq 0 \text{ and } \alpha_i(1 - y_i(w^t x_i + b)) = 0, \ \forall i$$

- Consider the condition $\alpha_i(1 - y_i(w^t x_i + b)) = 0, \ \forall i$. If $\alpha_i > 0$, then $1 - y_i(w^t x_i + b) = 0 \Rightarrow w^t x_i + b = 1$ if $y_i = 1$ and $w^t x_i + b = -1$ *if* $y_i = -1$. If $\alpha_i = 0$ and $1 - y_i(w^t x_i + b) \neq 0$, then $w^t x_i + b > 1$ if $y_i = 1$ and $w^t x_i + b < -1$ if $y_i = -1$. Such points are called well-classified

points; these points are not on the boundary of the class and are within the proper region of the respective class. If $\alpha_i = 0$ and $1 - y_i(w^t x_i + b) = 0$, then such x_i lie on the support planes, but $\alpha_i = 0$.

- So, if a data point x_i lies away from the support plane, that is, $y_i(w^t x_i + b) > 1$, then $\alpha_i = 0$. Now consider $w = \sum_{i=1}^{n} \alpha_i y_i x_i$. If $\alpha_i = 0$ for x_i, then it will not contribute to w. So, well-classified points do not contribute to w. Only training data points on the support planes with $\alpha > 0$ can contribute to w. Such vectors are called **support vectors**. So, $w = \sum_{i=1}^{n} \alpha_i y_i x_i = \sum_{i \in S} \alpha_i y_i x_i$, where S is the set of support vectors and S is a subset of the training set.

EXAMPLE 10: Consider the training points from the positive and negative classes, as shown in Fig. 8.7. There are two points from the negative class and one point from the positive class. These are:

- Negative Class: $(2, 1)$ and $(1, 3)$
- Positive Class: $(6, 3)$

So, they need to satisfy:

$$2w(1) + w(2) + b = -1$$
$$w(1) + 3w(2) + b = -1$$
$$6w(1) + 3w(2) + b = 1$$

These are three equations in three unknowns. By solving them, we get

$$w(1) = \frac{2}{5}, w(2) = \frac{1}{5} \text{ and } b = -2$$

Note that the point $(4, 2)$ is on the boundary because $4 \times \frac{2}{5} + 2 \times \frac{1}{5} - 2 = 0$. Further, $\sum_{i=1}^{3} \alpha_i y_i = 0 \Rightarrow -\alpha_1 - \alpha_2 + \alpha_3 = 0 \Rightarrow \alpha_3 = \alpha_1 + \alpha_2$.

We know that $w = (\frac{2}{5}, \frac{1}{5}) = -\alpha_1 \times (2, 1) - \alpha_2 \times (1, 3) + \alpha_3 \times (6, 3) \Rightarrow \alpha_1 = \alpha_3 = \frac{1}{10}; \alpha_2 = 0$.

So, the set of support vectors is $S = \{(2, 1), (6, 3)\}$. The point $(1, 3)$ will not contribute to w as $\alpha_2 = 0$. So, a boundary point or a point on the hyperplane need not contribute to w and so need not be a support vector.

Consider the origin, that is, $x = (0, 0)$. Here, $w^t x + b = -2 < -1$. So, it cannot be a support vector even if it is added to the training data. Similarly, a point $(7, 6)$ will be a well-classified point.

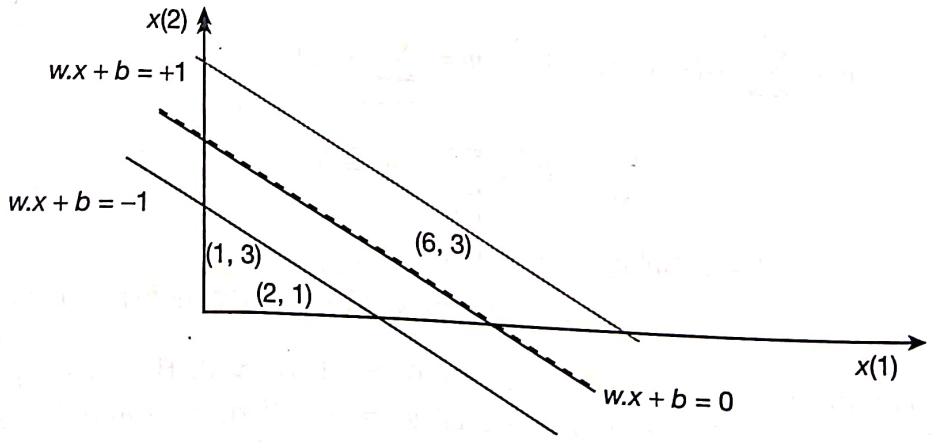

FIG. 8.7 An example to illustrate SVM (for colour figure, please see Colour Plate 2)

The SVM classifier can also be used as a feature selector. Let the w vector obtained at the end of training be $(0.001, -20, 10.9, 0)$ in a four-dimensional space. So, the second and the third features are important. The remaining two features are unimportant. The second feature is the most important and it votes in favour of the negative class. The next important feature is the third feature and it is in favour of the positive class.

8.4.1 Linearly Non-Separable Case

Consider the data shown in Fig. 8.8.

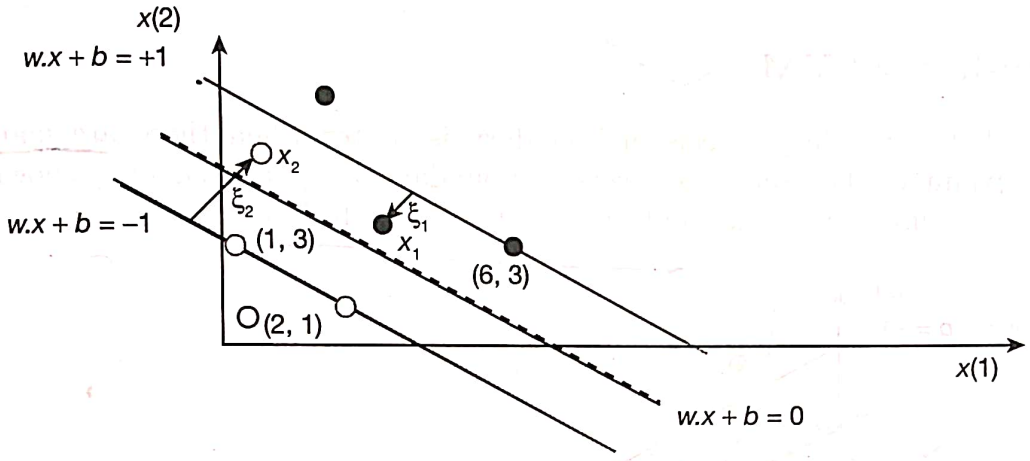

FIG. 8.8 An example to illustrate the margin violators (for colour figure, please see Colour Plate 2)

There are three points from the positive class; each of these is indicated by a blackened circle. There are four data points from the negative class; each of these is depicted using a whitened circle. There are two violators; they are labelled x_1 and x_2.

Data point x_1 is from the positive class and it violates the constraint given by $w^t x_1 + b \geq 1$ as it falls to the left of the positive support line. However, it is still on the correct side of the decision boundary. Similarly, data point x_2 is from the negative class and it violates the constraint $w^t x_2 + b \leq 1$. It clearly violates both the support line and the decision boundary.

The extent of violation is captured by values ξ_1 and ξ_2, respectively, for the two violators x_1 and x_2. Even though x_1 is a violator, it is still correctly classified as belonging to the positive class. However, x_2 is classified incorrectly as a member of the positive class. So, the violators need to be handled properly. Violations need to be tolerated to some extent.

This forms the basis for the so-called *soft-margin formulation* of the SVM given by

$$\begin{aligned} &\textit{minimize } \frac{1}{2} w^t w + C \sum_{i=1}^{n} \xi_i \\ &\textit{subject to } 1 - \xi_i - y_i(w^t x_i + b) \leq 0, \ i = 1, \ldots, n \\ &\qquad -\xi_i \leq 0, \ i = 1, \ldots, n. \end{aligned}$$

The Lagrangian is $\mathcal{L} = \frac{1}{2} w^t w + C \sum_{i=1}^{n} \xi_i + \sum_{i=1}^{n} \alpha_i (1 - \xi_i - y_i(w^t x_i + b)) - \sum_{i=1}^{n} \lambda_i \xi_i$. So, the necessary and sufficient conditions are

- $\nabla_w \mathcal{L} = 0 \Rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i$
- $\frac{\delta \mathcal{L}}{\delta b} = 0 \Rightarrow \sum \alpha_i y_i = 0$

- $\frac{\delta \mathcal{L}}{\delta \xi_i} = 0 \Rightarrow \alpha_i + \lambda_i = C$
- $1 - \xi_i - y_i(w^t x_i + b) \leq 0;\ \xi_i \geq 0,\ \forall i$
- $\alpha_i \geq 0;\ \lambda_i \geq 0,\ \forall i$
- $\alpha_i(1 - \xi_i - y_i(w^t x_i + b)) = 0;\ \lambda_i \xi_i = 0,\ \forall i$

So, the expressions for w and b have not changed. In the hard-margin (earlier) case, the Lagrange multipliers satisfied $\alpha_i \geq 0$. Here, it is possible to show that they will satisfy $\alpha_i \in [0, C]$; they are bounded by C; as $C \to \infty$, the soft-margin solution collapses to the earlier hard-margin one. If $C \to 0$, we tolerate any amount of violation (or any value of ξ). So, we need to find the appropriate value of C to obtain a good performance.

8.4.2 Non-linear SVM

It is possible that a non-linear decision boundary is better when there are margin violators. Consider, for example, the non-linear decision boundary (grey broken line) shown in Fig. 8.9. We will consider a simple example to illustrate a possible solution.

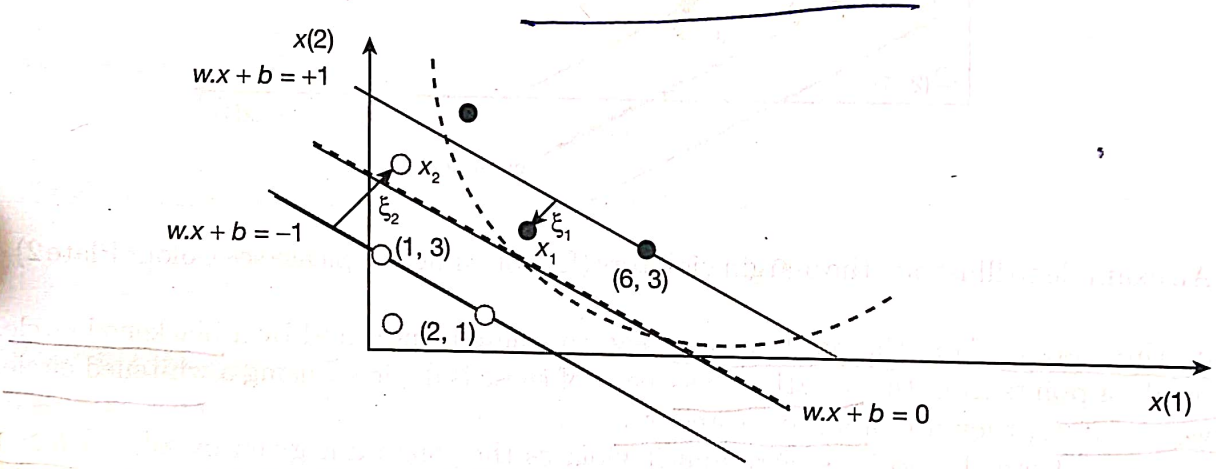

FIG. 8.9 An example to illustrate a non-linear SVM (for colour figure, please see Colour Plate 2)

EXAMPLE 11: Consider the truth table of the Boolean exclusive OR (XOR) shown in Table 8.6. Let us see whether we can have a linear decision boundary to separate the two classes based on outputs 0 and 1.

TABLE 8.6 Truth table of XOR

$x(1)$	$x(2)$	$XOR(x(1), x(2))$
0	0	0
0	1	1
1	0	1
1	1	0

Let the 0 output value correspond to the negative class and output 1 correspond to the positive class. Let the linear function be of the form $ax(1) + bx(2) + c$.

Input: $(0, 0)$ - Output: 0. So, $a \times 0 + b \times 0 + c < 0 \Rightarrow c < 0$.

Input: $(0,1)$ - Output: 1. So, $a \times 0 + b \times 1 + c > 0 \Rightarrow b + c > 0 \Rightarrow b = -c + \delta_1$, where $\delta_1 > 0$.
Input: $(1,0)$ - Output: 0. So, $a \times 1 + b \times 0 + c > 0 \Rightarrow a + c > 0 \Rightarrow a = -c + \delta_2$, where $\delta_2 > 0$.
Input: $(1,1)$ - Output: 0. So, $a \times 1 + b \times 1 + c < 0 \Rightarrow a + b + c < 0$. Plugging in for b and a, we get $-c + \delta_1 - c + \delta_2 + c = -c + \delta_1 + \delta_2 < 0$. This is not possible because $c < 0 \Rightarrow -c > 0$ and $\delta_1 > 0$ and $\delta_2 > 0$.

So, it is not possible to capture XOR using a linear function in $x(1)$ and $x(2)$.

Let us include the third input in the form of $x(1)x(2)$, the conjunction of $x(1)$ and $x(2)$. The resulting truth table along with a linear form in the three variables is shown in Table 8.7. A linear form $x(1) + x(2) - 2x(1)x(2)$ using the variables $x(1)$, $x(2)$ and $x(1)x(2)$ is shown in the fifth column of the table. Note that the output of XOR and $x(1) + x(2) - 2x(1)x(2)$ are identical in all the four cases. So, XOR can be represented as a linear function in a three-dimensional space based on variables $x(1)$, $x(2)$ and $x(1)x(2)$.

TABLE 8.7 Truth table of XOR in three variables

$x(1)$	$x(2)$	$x(1)x(2)$	$XOR(x(1), x(2))$	$x(1) + x(2) - 2x(1)x(2)$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	0

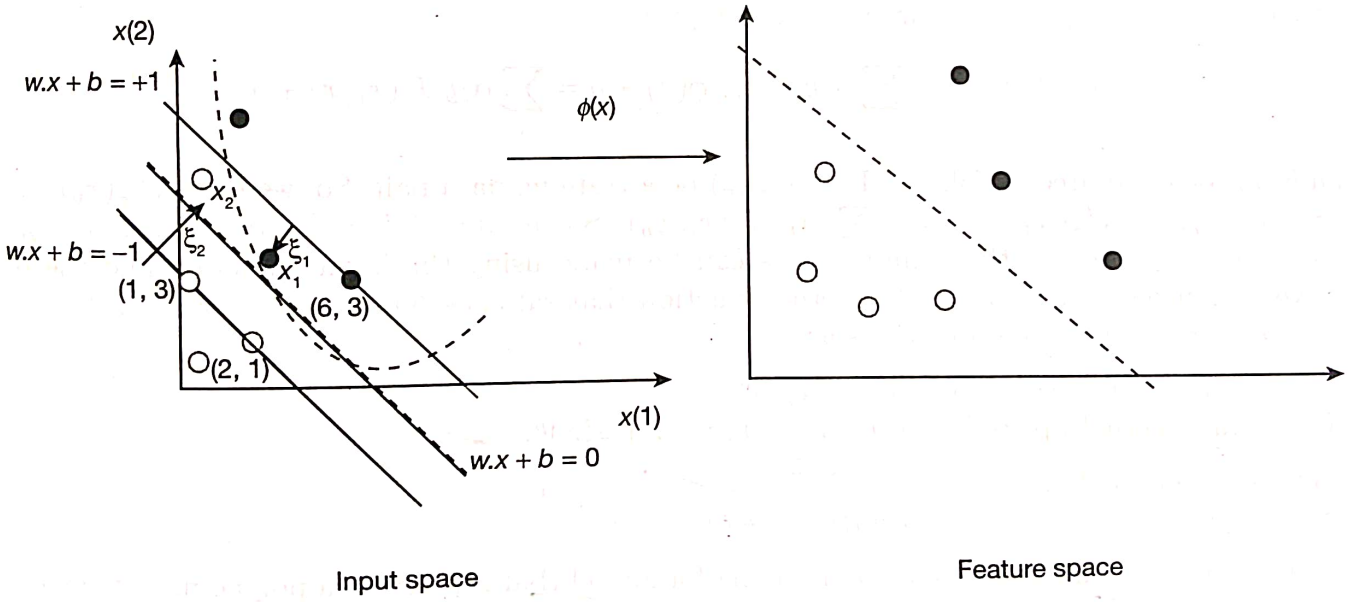

FIG. 8.10 Mapping the data to a high-dimensional feature space (for colour figure, please see Colour Plate 2)

So, we map the data points in the **input space** to a high-dimensional **feature space** using a mapping ϕ, as depicted in Fig. 8.10, with the hope that the data points are linearly separable in the higher-dimensional (feature) space. In the case of XOR, we have mapped the two-dimensional vector $(x(1), x(2))$ to a three-dimensional space given by $(x(1), x(2), x(1)x(2))$. Here, data points are linearly separable in the three-dimensional space.

We demonstrated the linear separability of XOR using the three-dimensional representation explicitly. In general, there are two problems associated with this approach:

- The data points may not be linearly separable in the feature space. We handle this issue by using the soft-margin formulation in the feature space.
- We may not know the explicit representation of the vectors in the feature space. Further, computation in the feature space may be unwieldy even if we know the explicit representation. This problem is handled using the **kernel trick**.

8.4.3 Kernel Trick

We are transforming a vector x in the input space to a higher dimensional space ($\phi(x)$) and look for a possible linear separation in the new space that is called the feature space. The most important observation is that we do not need to map ϕ explicitly. In theory, ϕ could map points into an infinite dimensional space. What we need is the inner product in the feature space. If x_i and x_j are two data points in the input space, then our computations will need terms of the form $\phi(x_i)^t\phi(x_j)$.

Many similarity functions can be viewed as inner products. If vectors $\phi(x_i)$ and $\phi(x_j)$ are unit norm vectors, then $cos\theta = \phi(x_i)^t\phi(x_j)$, where θ is the angle between the two vectors $\phi(x_i)$ and $\phi(x_j)$. We use the **kernel function**, $K(x_i, x_j)$, that maps a pair of vectors $x_i \in \Re^l$ and $x_j \in \Re^l$ to a real number. Specifically,

$$K(x_i, x_j) = \phi(x_i)^t\phi(x_j)$$

We know that $w = \sum \alpha_i y_i \phi(x_i)$ if we want to use the vectors of the form $\phi(x_i)$ explicitly. However, in order to classify a pattern x, we need to compute

$$w^t\phi(x) + b = \sum_i \alpha_i y_i \phi(x_i)^t\phi(x) + b = \sum_i \alpha_i y_i K(x_i, x) + b$$

Even b can be computed as follows. Let (x_p, y_p) be a training data pair. So, we have $w^t\phi(x_p) + b = y_p$. So, $b = y_p - w^t\phi(x_p) = y_p - \sum_i \alpha_i y_i K(x_i, x_p)$. So, $w^t\phi(x) + b = \sum_i \alpha_i y_i K(x_i, x) + y_p - \sum_i \alpha_i y_i K(x_i, x_p)$. So, all the computations can be made using the kernel function and the α_is. However, we need to have a suitable kernel function that satisfies $K(x_i, x_j) = \phi(x_i)^t\phi(x_j)$. Some popularly used kernel functions are:

- Polynomial of degree p: $K(x_i, x) = (x_i^t x)^p$
- Polynomial kernel up to degree p: $K(x_i, x) = (1 + x_i^t x)^p$.
- Gaussian kernel: $K(x_i, x) = e^{-\frac{||x_i - x||^2}{\sigma^2}}$.
- Sigmoidal kernel: $K(x_i, x) = tanh(ax_i^t x + b)$

We will see the underlying feature mapping for kernel that represents a polynomial of degree 2. Consider the kernel on two-dimensional patterns given by

$$K(x_i, x) = (x_i^t x)^2 = (x_i(1)x(1) + x_i(2)x(2))^2 = x_i(1)^2x(1)^2 + x_i(2)^2x(2)^2 + 2x_i(1)x_i(2)x(1)x(2).$$

This kernel computation can be achieved by using a feature mapping from the two-dimensional input space to a three-dimensional feature space given by:

$$\phi(\begin{pmatrix} x_i(1) \\ x_i(2) \end{pmatrix}) = \begin{pmatrix} x_i(1)^2 \\ x_i(2)^2 \\ \sqrt{2}x_i(1)x_i(2) \end{pmatrix}$$

Now we can verify that $K(x_i, x) = \phi(x_i)^t\phi(x)$ because the RHS is the dot product of $\phi(x_i)$ and $\phi(x)$ and

$$\phi(x_i)^t\phi(x) = x_i(1)^2x(1)^2 + x_i(2)^2x(2)^2 + 2x_i(1)x_i(2)x(1)x(2)$$

So, using the polynomial kernel of degree 2 is the same as applying the mapping ϕ that maps a two-dimensional vector to a three-dimensional vector as specified and computing the dot product in the three-dimensional space.

Note that the Gaussian kernel employs an exponential function and so its expansion can have infinite terms. So, the corresponding feature map ϕ can map a finite dimensional vector to an infinite dimensional vector. Because of the kernel trick, our computations are all performed in the finite dimensional input space but not in the feature space. So, the kernel function is a similarity function and the kernel trick can be employed even in the case of other ML models where similarity needs to be computed. We explore its use along with nearest neighbor classifier (NNC) in an exercise.

8.5 LOGISTIC REGRESSION

We have seen how linear discriminants are used by both perceptron and SVM in classification.

Perceptron employs a linear model $g(x) = w^tx + b$ and learns w and b using the training data. It can employ a generalized linear model to handle non-linear decision boundaries by augmenting w and x vectors suitably; it then uses a linear model on the augmented data.

SVM inherently deals with a linear model of the form $w^tx + b$, where w corresponds to the maximum margin given by $maximize \frac{2}{||w||}$. It employs a kernel to deal with non-linear decision boundaries by looking for a linear separation in the feature or kernel space. It can also combine a soft-margin formulation to permit some margin violators; this helps us in using a simpler model.

We will see that in logistic regression, another generalized linear model is employed. Before we get into the details of logistic regression, we will consider linear regression.

8.5.1 Linear Regression

Here, we assume that the model obeys a linear relationship. We would like to obtain the weight vector w from the data given. Specifically, let $A_{n\times l}$ be the data matrix having n patterns, each pattern forming a row, and l columns, one column per feature. Let $y_{n\times 1}$ be the vector of n target or the given output values. Let w be the vector of unknowns that linearly links x_i with $\hat{y}_i$, that is, an estimate of y_i. Note that the pattern x_i forms the i^{th} row of A and the value $\hat{y}_i$ is w^tx_i.

In this simple linear model, we are assuming that there is no noise and the bias (b) is not explicitly shown. We assume that x_i and w are suitably augmented to handle the bias term if required, like in the case of perceptron. We assume that the weight vector w is such that $w^tx_i = x_i^tw_i = y_i$; that is, x_i and y_i are linearly related. In practice, the estimated $\hat{y}_i$ can differ from its target value y_i. So, error vector e is given by $e = Aw - y$.

So, we consider the squared norm of e which is

$$||e||^2 = ||Aw - y||^2 = (Aw - y)^t(Aw - y) = w^tA^tAw - y^tAw - (Aw)^ty + y^ty$$

$= w^tA^tAw - 2y^tAw + y^ty$. We find w that minimizes this squared norm.

We need to find w. So, by taking the gradient of the squared norm of e with w and equating to 0, we get

$$2A^tAw - 2A^ty = 0 \Rightarrow w = (A^tA)^{-1}A^ty$$

So, w gives us the estimates of the coefficients involved.

EXAMPLE 12: Let us consider a simple example of three vectors $x_1 = (1,1)$, $x_2 = (1,2)$ and $x_3 = (2,2)$. Let $y_1 = 3$, $y_2 = 5$ and $y_3 = 6$. So, the data matrix A consisting of the augmented vectors is: $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 2 & 2 & 1 \end{bmatrix}$ and $w = (w(1), w(2), w(3))$.

We have seen that $w = (A^tA)^{-1}A^ty$; so we have

$$A^tA = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 2 \\ 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 2 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 6 & 7 & 4 \\ 7 & 9 & 5 \\ 4 & 5 & 3 \end{bmatrix}.$$

and

$$(A^tA)^{-1} = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -2 \\ -1 & -2 & 5 \end{bmatrix}$$

$A^ty = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 2 \\ 1 & 1 & 1 \end{bmatrix} \begin{pmatrix} 3 \\ 5 \\ 6 \end{pmatrix} = \begin{pmatrix} 20 \\ 25 \\ 14 \end{pmatrix}$. So, $w = (A^tA)^{-1}A^ty =$

$$\begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -2 \\ -1 & -2 & 5 \end{bmatrix} \begin{pmatrix} 20 \\ 25 \\ 14 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix}$$

So, the coefficient corresponding to the augmented position is 0 ($b = 0$). Here, the linear form is $\hat{y} = 1x(1) + 2x(2)$. We can verify that for pattern $x_1 = (1,1)$, $\hat{y}_1 = 1 \times 1 + 2 \times 1 = 3 = y_i$. Similarly it is possible to verify for the other x_i also.

8.5.2 Sigmoid Function

We use a non-linear function of the linear form given by $f(w^tx+b)$, where f is the sigmoid function. The sigmoid or logistic function is given by $f(a) = \frac{1}{1+e^{-a}}$, where $a = w^tx+b$. Some of its properties are given below:

- $f : \Re \rightarrow [0,1]$. It maps a real number to a value in the interval [0,1].
- The function assumes its maximum value of 1 as $a \rightarrow \infty$ and its minimum value is reached as $a \rightarrow -\infty$.
- When $a = 0$, $f(a) = \frac{1}{2}$.
- If $y = f(a) = \frac{1}{1+e^{-a}}$, then $a = f^{-1}(y) = ln(\frac{y}{1-y})$. So, the function f is invertible.
- The derivative of $f(a)$ is

$$f'(a) = \frac{(-1)(-e^{-a})}{(1+e^{-a})^2} = f(a)(1-f(a))$$

- We can interpret $f(a)$ as probability by viewing $a = w^tx + b$.

$$f(a) = P(y = 1|x) = \frac{1}{1+e^{-w^tx-b}}$$

- If we let $p_1 = P(y = 1|x)$, then

$$p_1 = \frac{1}{1+e^{-w^tx-b}} \Rightarrow p_1(1+e^{-w^tx-b}) = 1 \Rightarrow w^tx + b = ln\left(\frac{p_1}{1-p_1}\right)$$

In the terminology of neural functions, it is called **sigmoid activation function.** It is also called logistic function and is denoted by $\sigma(a)$, where a is the activation input. Here, $a = w^t x$ assuming that w and x are augmented vectors.
We have seen that $\sigma'(a) = \sigma(a)(1 - \sigma(a))$. We are given training data in the form $\{(x_i, y_i), i = 1, 2, \ldots, n\}$. These y_is could be viewed as target outputs or required outputs; so, we can denote them as y_i^{tar}.
Based on the value of w, we obtain $y_i^{obt} = \sigma(w^t x)$, as shown in Fig. 8.11.

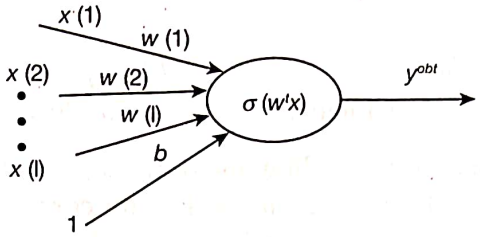

FIG. 8.11 Logistic regression using perceptron

The error is $E = \sum_{i=1}^{n}(y_i^{tar} - y_i^{obt})^2$ based on squared error loss; it is the square of the difference between the target output and the obtained output based on the current w. The partial derivative of E with respect to $w(j)$ is

$$\frac{\partial E}{\partial w(j)} = \sum_{i=1}^{n}(y_i^{tar} - y_i^{obt})\sigma'(w^t x)x_i(j) = \sum_{i=1}^{n}(y_i^{tar} - y_i^{obt})\sigma(w^t x)(1 - \sigma(w^t x))x_i(j)$$

Note that

$$\frac{\partial E}{\partial b} = \sum_{i=1}^{n}(y_i^{tar} - y_i^{obt})\sigma(w^t x)(1 - \sigma(w^t x)) \times 1$$

because $x_i(j) = 1$ in the augmented vector x for the position corresponding to b.

EXAMPLE 14: Let us consider the data used in Example 10.
$x_1 = (1, 2)$, $x_2 = (2, 1)$, $x_3 = (6, 7)$, $x_4 = (7, 6)$
$y_1 = 0$, $y_2 = 0$, $y_3 = 1$, $y_4 = 1$
Let $w_0 = (0, 0, 0)$ be the initial augmented vector. Using the update equation $w_1(j) = w_0(j) + \eta(\sum_{i=1}^{n}(y_i^{tar} - y_i^{obt})\sigma(w^t x)(1 - \sigma(w^t x))x_i(j)$ for $j = 1, 2$, we get $w_1(1) = 0.0125 = w_1(2)$ using $\eta = 0.01$, where η is the learning rate.
For $w_1(3) = b_1$, we get $w_1(3) = 0.01 \times (-0.5) \times 0.25 \times 2 + 0.01 \times (0.5) \times 0.25 \times 2 = 0$. So, $w_1 = (0.0125, 0.0125, 0)$.
The corresponding $\sigma(w_1^t x_i)$ values for the patterns are 0.509, 0.509, 0.541 and 0.541, assigning all the patterns to Class 1 as the output values are larger than 0.5 for all the four patterns.
After some iterations, we get $w_9 = (0.084, 0.084, -0.01)$. The corresponding $\sigma(w_9^t x_i)$ values are 0.2531,0.2531, 0.747, 0.747 for the four patterns, where the first two patterns are assigned to Class 0 and the remaining two patterns to Class 1 based on thresholding with 0.5; that is, if the value is less than 0.5, it is treated as 0, else as 1.

8.6 MULTI-LAYER PERCEPTRONS (MLPS)

An artificial neuron forms the building block of any MLP and is shown in Fig. 8.11. A neuron can have a finite number of inputs. The inputs are represented as $x(1), x(2), \ldots, x(l)$. Each input

is associated with a weight; input $x(i)$ uses weight $w(i)$ for $i = 1, 2, \ldots, l$. Bias threshold can be considered as an input with a fixed input value of 1. The corresponding weight is b or $w(0)$ or $w(l+1)$ based on how the augmentation is performed.

Output or input to the activation unit is $a = w^t x = w(1)x(1) + w(2)x(2) + \cdots + w(l)x(l) + w(l+1)1$. If $\sigma(a)$ is the activation output, then using sigmoid activation we have the output as $\frac{1}{1+e^{-a}} = \frac{e^a}{1+e^a}$. The problem is to learn the weights $w(1), w(2), \ldots, w(l+1)$ using a training data set.

In an MLP network, we typically have multiple layers of connected neurons, as shown in Fig. 8.12. It has an input layer where the augmented vector x is input. The number of input units is $l+1$ as there will be $l+1$ entries in the augmented vector. There are 5 input units in this example.

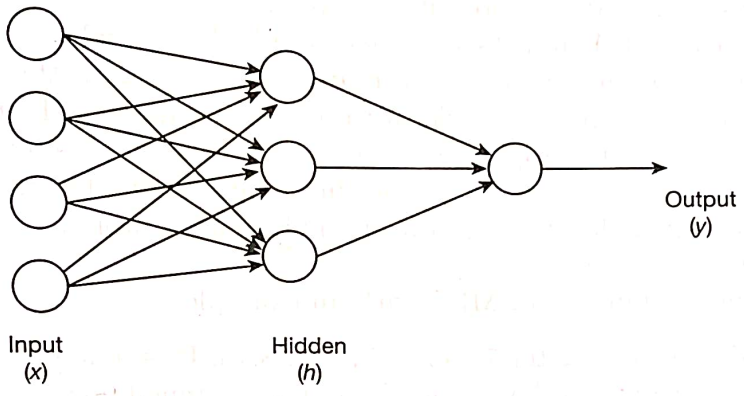

FIG. 8.12 An example of an MLP network

There is an output layer that outputs y. In general, there can be N_o output units. In this example, there is only one output node.

There is a layer of neurons in between the input and output layers in the example. It is called the **hidden layer** and is denoted by h; it is hidden from the input–output activity. However, learning representations from the input data takes place here. In the example, there are three nodes in the hidden layer. We also call it a **feed-forward neural network** because outputs of the neurons in each layer form the inputs for the neurons in the next layer. This forward propagation continues till we reach the output layer for which there will be no more layers.

In general, there could be more than one hidden layer and each of them can have a different number of processing elements or neurons. If there are p hidden layers, then we call them h_1, h_2, $\cdots$, h_p. There is no specific relationship between the number of neurons in the input, output and hidden layers. The number of output neurons can be different from the number of input neurons.

In Fig. 8.12, we have two layers having weights. So, we say that it has two layers and one of them is a hidden layer. It is possible to show that one hidden layer or a network with two layers is adequate to approximate any continuous function for inputs in a specified range.
The architecture of MLP has:

- No connections among the neurons within a layer.
- Typically no direct connections between neurons in the input layer and neurons in the output layer.
- Neurons between layers that are fully connected. In the MLP network shown in Fig. 8.12, each of the five input layer neurons is connected to all the three neurons in the hidden layer. So, there are 15 weights between layers. Similarly, each of the three hidden neurons is connected to the output neuron; here we have 3 weights between the hidden and output layers.

- Each hidden neuron is a perceptron with output $y_i = f(w^t x_i)$, where x_i is its input. Even each of the output neurons is typically a perceptron.

Recall that we had a linear threshold function or a step function as the activation function in the case of perceptron. However, this function is not differentiable and we require a differentiable function to deal with the learning process. So, a simple activation function is the linear activation function. However, if we use the linear activation function throughout the MLP network, we have, with respect to Fig. 8.12,

$$h = A_1 x; y = A_2 h \Rightarrow y = A_2 A_1 X = Bx,$$

where A_1 is the matrix of weights in the layer between the input and hidden layers and A_2 is the matrix of weights between the hidden and output layers.

The size of A_1 is 5×3 and of A_2 it is 3×1 in the example. In general, A_1 can be of size $l + 1 \times n_h$ and A_2 can be of size $n_h \times n_o$, where n_h is the number of neurons in the hidden layer and n_o is the number of neurons in the output layer. So, if we let the product of the matrices $A_2 \times A_1 = B$, then effectively the input–output mapping will be $y = Bx$, that is, a simple linear mapping. So, non-linear mappings cannot be realized if we use linear activation throughout. Further, $y = Bx$ in effect corresponds to a single-layer computation. This is applicable not only to the example in Fig. 8.12, but for any general MLP.

We will illustrate the working of an MLP with an example.

EXAMPLE 15: Consider the character 7 shown in Fig. 8.13. It is in a grid of size 6×6 cells or 36 pixels. The grey portion indicates the background and the whitened portion represents character 7. We will look at an MLP network that can be used to detect whether the given image of 6×6 pixels is a 7 or not.

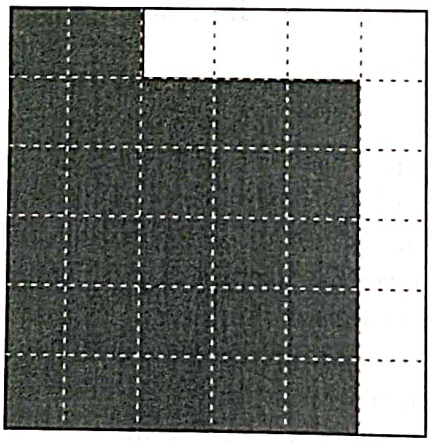

FIG. 8.13 The character 7 in a 6×6 grid

We have 36 pixels in the input image. The number of input neurons is 36, as shown in Fig. 8.14. We typically initialize the MLP with small random weights. The number of hidden nodes is m, where m is a tunable parameter. The training and test data sets consist of a number of patterns, each of size 6×6.

For a given initialization of weights in the MLP, we get an output y_i^{obt} when we input x_i. Based on the difference between y_i^{tar} and y_i^{obt}, the weights in the network are adjusted so that the MLP gives an output of 1 when the input character is 7 and 0 otherwise. This updating of weights is performed with the help of a training algorithm called backpropagation which we will consider next.

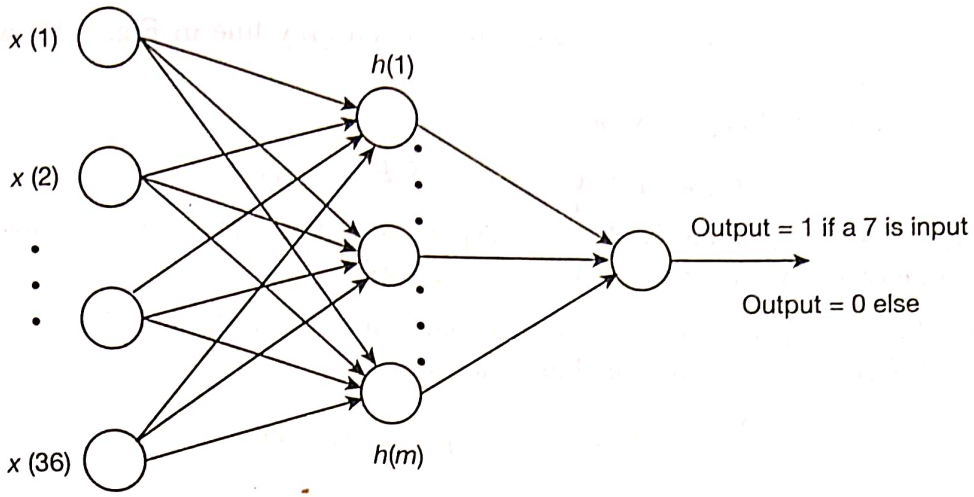

FIG. 8.14 MLP to classify 7

8.6.1 Backpropagation for Training an MLP

Backpropagation is a well-known and simple optimization algorithm that helps us in learning the augmented weight vectors in MLP.
It has two passes:

- **Forward Pass:** The network is initialized with some random weights. Input x is applied at the input layer. The output of a layer of neurons is computed and is forwarded to the next layer. This process is continued till the output of the entire MLP is computed. Let the obtained output be y^{obt}, where the required or target output is y^{tar}.
- **Backward Pass:** The error at the output is a function of the difference between y^{tar} and y^{obt}. Here, y^{obt} depends on the current weights of the network. An error is formulated as a function of current weights and these weights are updated to move in the direction of minimizing the error using gradient descent. This is achieved by backpropagating the error.

Backpropagation employs a **gradient descent** scheme. Gradient descent may be explained using Fig. 8.15. There are two local minima, $w*$ and w_3. If we start at w_2 and try to move in the negative

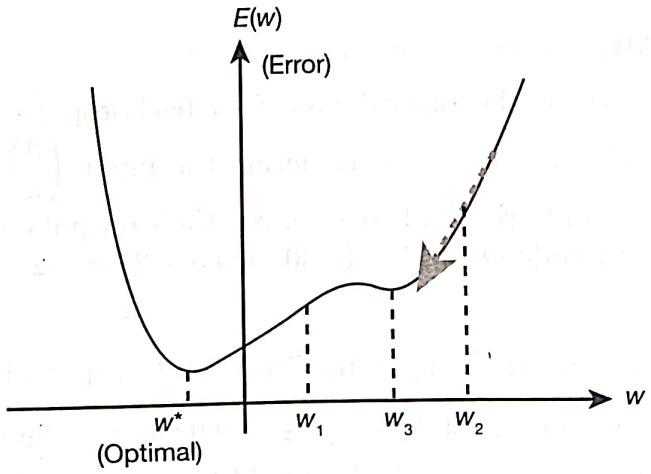

FIG. 8.15 Gradient descent for minimizing error

direction of the gradient at w_2, as shown using the broken grey line in Fig. 8.15, we keep moving towards the local minimum w_3.

The update equation will typically be

$$w_{updated} = w_{current} - \eta \nabla E(w_{current}),$$

where $\nabla E(w_{current})$ is the gradient of E with respect to entries in w and the result is evaluated at the current value of w. We illustrate it with an example.

Example 16: Let $w_{current} = (0.5, 0)$ and the pattern $x = (1, 2)$. Let $E = (w(1)x(1) + w(2)x(2) - 1)^2$. Here, $\nabla E(w_{current})$ is obtained as follows:

$$\frac{\delta E}{\delta w(1)} = 2(w(1)x(1) + w(2)x(2) - 1)x(1)$$

$$\frac{\delta E}{\delta w(2)} = 2(w(1)x(1) + w(2)x(2) - 1)x(2)$$

Plugging in the values of $w(1), w(2), x(1)$ and $x(2)$, we have, at the value of $w_{current} = (0.5, 0)$,

$$\frac{\delta E}{\delta w(1)} = -1$$

Similarly,

$$\frac{\delta E}{\delta w(2)} = -2$$

The gradient $\nabla E(w_{current})$ is a vector of size 2 as $w_{current}$ is a two-dimensional vector, where the first component is the partial derivative with respect to $w(1)$ evaluated at $w_{current}$ and the second component is the partial derivative with respect to $w(2)$ evaluated at $w_{current}$. In this example, $\nabla E(w_{current}) = (-1, -2)$.

If we start at a point like w_1 and move in the direction of the negative of the gradient, we may reach the local optimum $w*$ that is also the global optimum. η is the *learning rate*. The value of η plays an important role. If it is too small, then it takes a large number of steps and updates to reach the local minima. If the value of η is large, then it may oscillate around the local minimum point without reaching it.

Forward Pass

Let us consider a simple MLP network shown in Fig. 8.16.

Example 17: We will illustrate the forward pass of the backpropagation algorithm. The forward pass will give us the following outputs by considering the input $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$. The activation inputs to both C and D are the same and are equal to zero. So, their outputs are also equal and they are $\frac{1}{1+e^{-0}} = 0.5$. So, the input to node O is $0.5 \times (-30) + 0.5 \times 25 = -2.5$. So, the output of node O is $\frac{1}{1+e^{2.5}} = 0.07586$.

For the input $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$, the activation input to C is 1 and output of C is $\frac{1}{1+e^{-1}} = 0.731$. The activation input to D is 8 and output of D is $\frac{1}{1+e^{-8}} = 0.9996$. So, the input to node E is $0.731 \times (-30) + 0.9996 \times 25 = 3.06$. So, the output of node O is $\frac{1}{1+e^{-3.06}} = 0.9552$. Computation of the outputs in the forward pass for inputs $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ are left as part of an exercise.