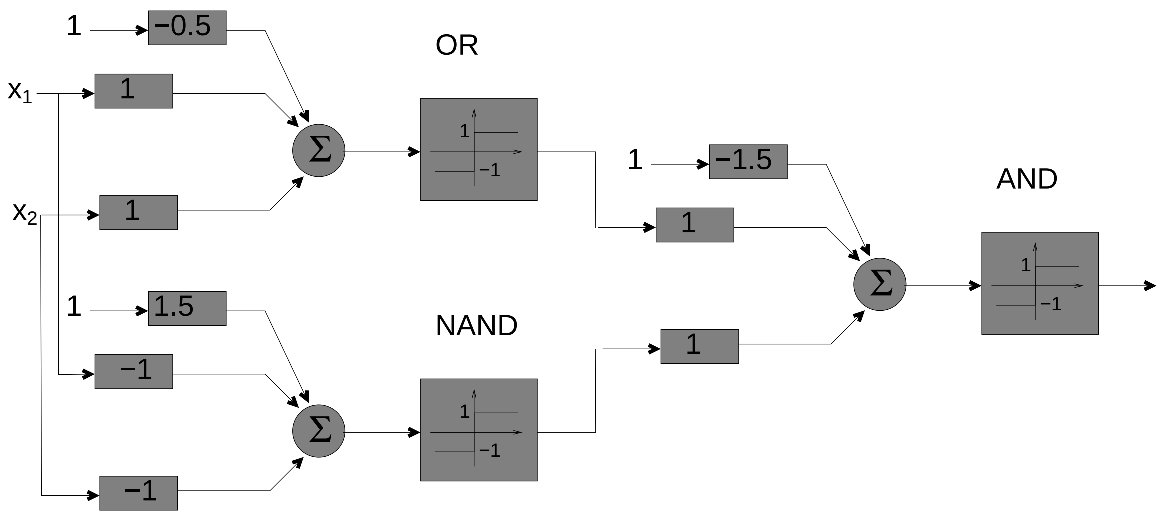
University of Cambridge Engineering Part IIB

Module 4F10: Statistical Pattern Processing

Handout 5: Single Layer Perceptron & Estimating Linear Classifiers



Phil Woodland
pcw@eng.cam.ac.uk

Lent 2007

Introduction

So far we have concentrated on classifier design by explicit modelling of class-conditional probability density functions.

An alternative to modelling the class conditional probability density functions is to decide on some functional form for a discriminant function and attempt to construct a classifier directly.

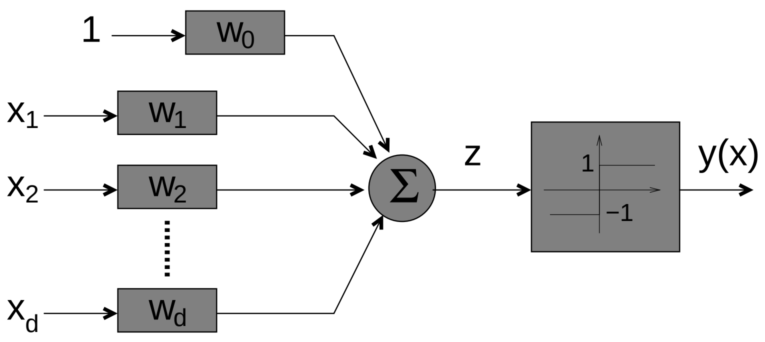
Here we will concentrate on the construction of *linear classifiers*: however it is also possible to construct quadratic or other non-linear decision boundaries (this is how some types of neural networks operate).

There are several methods of classifier construction for a linear classifier. We will examine

- The single layer perceptron. Iterative solution for the weights is via the perceptron algorithm which directly minimises the number of misclassifications
- Least squares estimation (viewing classification as function interpolation)
- Fisher linear discriminant which aims directly to maximise a measure of class separability

Single Layer Perceptron

A single layer perceptron is shown below. The typical form examined uses a *threshold activation function*:



The d-dimensional input vector $\mathbf{x}$ and scalar value z are related by

$$z = \mathbf{w}'\mathbf{x} + w_0$$

z is then fed to the activation function to yield $y(\mathbf{x})$. The parameters of this system are

- **weights**: $\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}$, selects the *direction* of the decision boundary
- **bias**: w_0, sets the *position* of the decision boundary.

These parameters are often combined into a single composite vector, $\tilde{\mathbf{w}}$, and the input vector extended, $\tilde{\mathbf{x}}$.

$$\tilde{\mathbf{w}} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \\ w_0 \end{bmatrix}; \qquad \tilde{\mathbf{x}} = \begin{bmatrix} x_1 \\ \vdots \\ x_d \\ 1 \end{bmatrix}$$

Single Layer Perceptron (cont)

We can then write

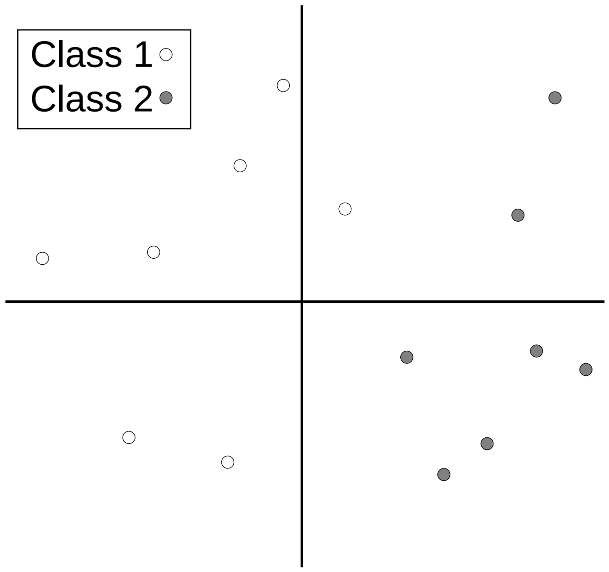
$$z = \tilde{\mathbf{w}}'\tilde{\mathbf{x}}$$

The task is to train the set of model parameters $\tilde{\mathbf{w}}$. For this example a *decision boundary* is placed at $z = 0$. The decision rule is

$$y(\mathbf{x}) = \begin{cases} 1, & z \geq 0 \\ -1, & z < 0 \end{cases}$$

If the training data is linearly separable in the d-dimensional space then using an appropriate training algorithm perfect classification (on the training data at least!) can be achieved.

Is the solution unique?



The precise solution selected depends on the training algorithm used.

Parameter Optimisation

First a *cost function* of the set of the weights must be defined, $E(\tilde{\mathbf{w}})$. Some *learning process* which minimises the cost function is then used. One basic procedure is **gradient descent**:

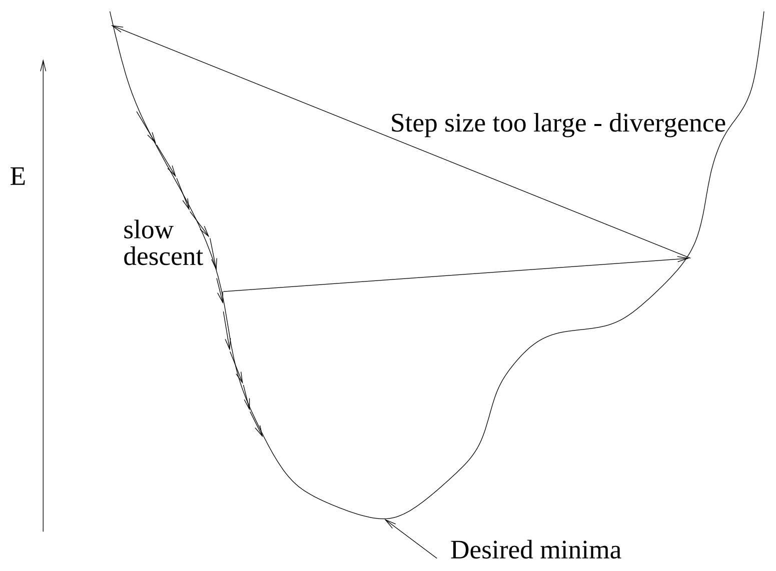
1. Start with some initial estimate $\tilde{\mathbf{w}}[0]$, $\tau = 0$.
2. Compute the gradient $\nabla E(\tilde{\mathbf{w}})|_{\tilde{\mathbf{w}}[\tau]}$
3. Update weight by moving a small distance in the steepest downhill direction, to give the estimate of the weights at iteration $\tau + 1$, $\tilde{\mathbf{w}}[\tau + 1]$,
$$\tilde{\mathbf{w}}[\tau + 1] = \tilde{\mathbf{w}}[\tau] - \eta \nabla E(\tilde{\mathbf{w}})|_{\tilde{\mathbf{w}}[\tau]}$$
This can be written for just the i^{th} element
$$\tilde{w}_i[\tau + 1] = \tilde{w}_i[\tau] - \eta \left. \frac{\partial E(\tilde{\mathbf{w}})}{\partial \tilde{w}_i} \right|_{\tilde{\mathbf{w}}[\tau]}$$
Set $\tau = \tau + 1$
4. Repeat steps (2) and (3) until convergence, or the optimisation criterion is satisfied

One of the restrictions on using gradient descent is that the cost function $E(\tilde{\mathbf{w}})$ *must* be differentiable (and hence continuous). This means *mis-classification* cannot be used as the cost function for gradient descent schemes.

Gradient descent is *not* usually guaranteed to decrease the cost function.

Choice of η

In the previous slide the *learning rate* term η was used in the optimisation scheme.



η is positive. When setting η we need to consider:

- if η is too small, convergence is slow;
- if η is too large, we may overshoot the solution and diverge.

Later in the course we will examine improvements for gradient descent schemes for highly complex schemes. Some of these give automated techniques for setting η.

Perceptron Criterion

The decision boundary should be contructed to minimise the number of misclassified training examples. For each training example $\tilde{\boldsymbol{x}}_i$

$$\begin{array}{rcll} \tilde{\boldsymbol{w}}'\tilde{\boldsymbol{x}}_i & > & 0 & \tilde{\boldsymbol{x}}_i \in \omega_1 \\ \tilde{\boldsymbol{w}}'\tilde{\boldsymbol{x}}_i & < & 0 & \tilde{\boldsymbol{x}}_i \in \omega_2 \end{array}$$

The problem is that we cannot simply sum the values of $\tilde{\mathbf{w}}'\tilde{\mathbf{x}}$ as a cost function since the sign depends on the class. For training, we replace all the observations of class ω_2 by their negative value. Thus

$$\overline{\mathbf{x}} = \begin{cases} \tilde{\mathbf{x}}; & \text{belongs to } \omega_1 \\ -\tilde{\mathbf{x}}; & \text{belongs to } \omega_2 \end{cases}$$

This means that for a correctly classified symbol

$$\tilde{\mathbf{w}}'\overline{\mathbf{x}} > 0$$

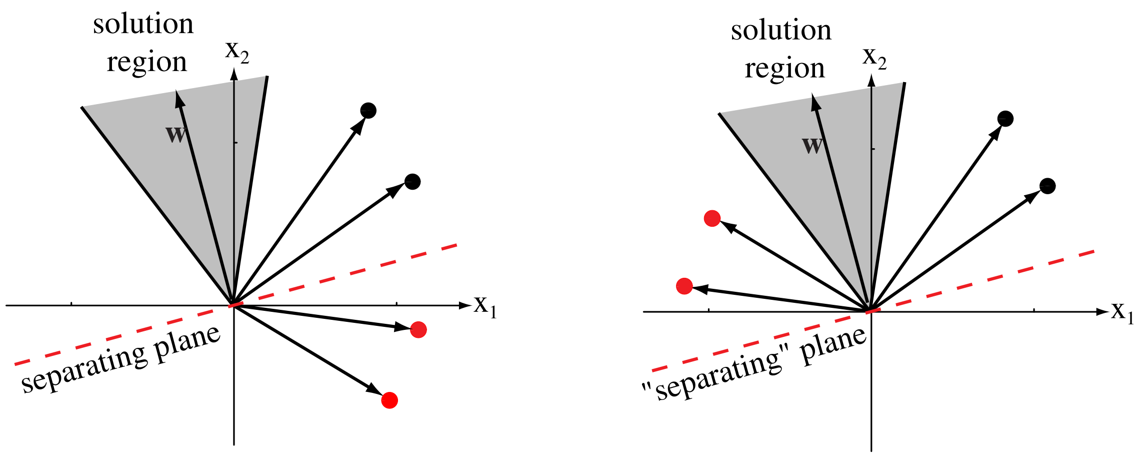
and for mis-classified training examples

$$\tilde{\mathbf{w}}'\overline{\mathbf{x}} < 0$$

We will refer to this as "normalising" the data.

Perceptron Solution Region

- Each sample $\tilde{\boldsymbol{x}}_i$ places a constraint on the possible location of a solution vector that classifies all samples correctly.
- $\tilde{\boldsymbol{w}}'\tilde{\boldsymbol{x}}_i = 0$ defines a hyperplane through the origin on the "weight-space" of $\tilde{\boldsymbol{w}}$ vectors with $\tilde{\boldsymbol{x}}_i$ as a normal vector.
- For normalised data, the solution vector must be on the positive side of every such hyperplane.
- The solution vector, if it exists, is not unique and lies anywhere within the solution region
- Figure below (from DHS) shows 4 training points and the solution region for both normalised data and un-normalised data.



Perceptron Criterion (cont)

The perceptron criterion may be expressed as

$$E(\tilde{\mathbf{w}}) = \sum_{\overline{\mathbf{x}} \in \mathcal{Y}} (-\tilde{\mathbf{w}}' \overline{\mathbf{x}})$$

where $\mathcal{Y}$ is the set of *miss-classified* points. We now want to minimise the *perceptron criterion*. We can use gradient descent. It is simple to show that

$$\boldsymbol{\nabla} E(\tilde{\mathbf{w}}) = \sum_{\overline{\mathbf{x}} \in \mathcal{Y}} (-\overline{\mathbf{x}})$$

Hence the GD update rule is

$$\tilde{\mathbf{w}}[\tau + 1] = \tilde{\mathbf{w}}[\tau] + \eta \sum_{\overline{\mathbf{x}} \in \mathcal{Y}[\tau]} \overline{\mathbf{x}}$$

where $\mathcal{Y}[\tau]$ is the set of mis-classified points using $\tilde{\mathbf{w}}[\tau]$. For the case when the samples are linearly separable using a value of $\eta = 1$ is guaranteed to converge. The basic algorithm is:

1. Take the extended observations $\tilde{\mathbf{x}}$ for class ω_2 and invert the sign to give $\overline{\mathbf{x}}$.
2. Initialise the weight vector $\tilde{\mathbf{w}}[0]$, $\tau = 0$.
3. Using $\tilde{\mathbf{w}}[\tau]$ produce the set of mis-classified samples $\mathcal{Y}[\tau]$.
4. Use update rule

   $$\tilde{\mathbf{w}}[\tau + 1] = \tilde{\mathbf{w}}[\tau] + \sum_{\overline{\mathbf{x}} \in \mathcal{Y}[\tau]} \overline{\mathbf{x}}$$

   then set $\tau = \tau + 1$.
5. Repeat steps (3) and (4) until the convergence criterion is satisfied.

A Simple Example

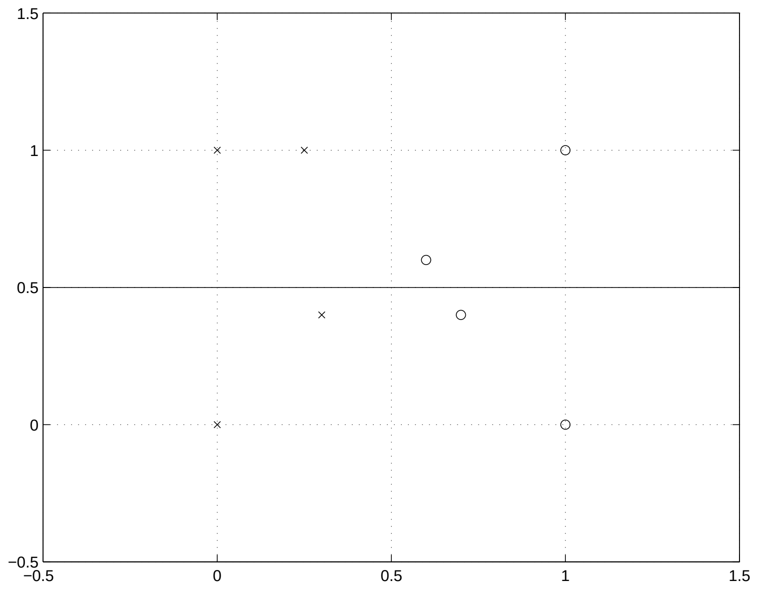
Consider a simple example:

Class 1 has points $\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.6 \\ 0.6 \end{bmatrix}, \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix}$

Class 2 has points $\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.25 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.3 \\ 0.4 \end{bmatrix}$

Initial estimate: $\tilde{\mathbf{w}}[0] = \begin{bmatrix} 0 \\ 1 \\ -0.5 \end{bmatrix}$

This yields the following initial estimate of the decision boundary.



Given this initial estimate we need to train the decision boundary.

Simple Example (cont)

The first part of the algorithm is to use the current decision boundary to obtain the set of mis-classified points.

For the data from Class ω_1

Class ω_1	1	2	3	4
z	-0.5	0.5	0.1	-0.1
Class	2	1	1	2

and for Class ω_2

Class ω_2	1	2	3	4
z	-0.5	0.5	0.5	-0.1
Class	2	1	1	2

The set of mis-classified points, $\mathcal{Y}[0]$, is

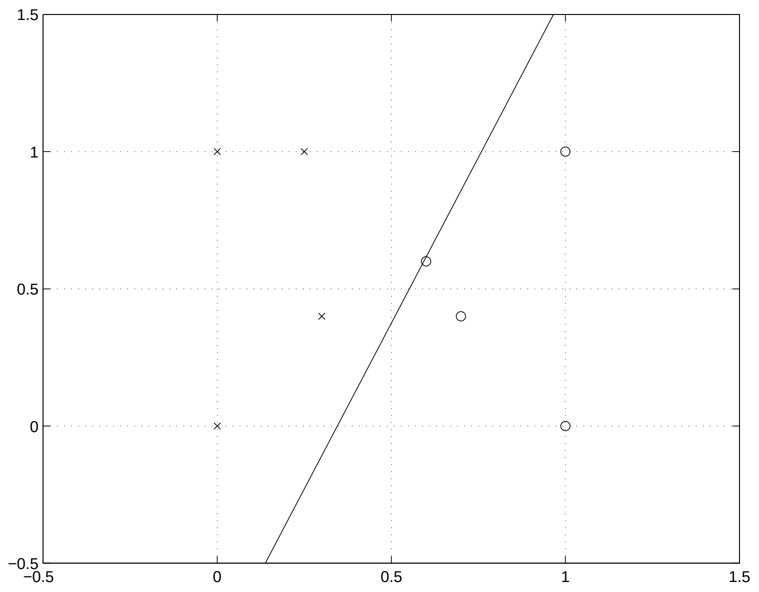
$$\mathcal{Y}[0] = \{1_{\omega_1}, 4_{\omega_1}, 2_{\omega_2}, 3_{\omega_2}\}$$

From the perceptron update rule this yields the updated vector

$$\tilde{\mathbf{w}}[1] = \tilde{\mathbf{w}}[0] + \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.7 \\ 0.4 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \\ -1 \end{bmatrix} + \begin{bmatrix} -0.25 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1.45 \\ -0.6 \\ -0.5 \end{bmatrix}$$

Simple Example (cont)

This yields the following decision boundary



Again applying the decision rule to the data we get for class ω_1

Class 1	1	2	3	4
z	0.95	0.35	0.01	0.275
Class	1	1	1	1

and for class ω_2

Class 2	1	2	3	4
z	-0.5	-1.1	-0.7375	-0.305
Class	2	2	2	2

All points correctly classified the algorithm has converged.

Is this a good decision boundary?

The LMS Algorithm

An alternative optimisation criterion is to use *least mean squares* estimation. This uses the following cost function

$$E(\tilde{\mathbf{w}}) = \frac{1}{2}\sum_{\forall \mathbf{x}} \left(g(\mathbf{x}) - t(\mathbf{x})\right)^2$$

where $t(\mathbf{x})$ is the target value for observation $\mathbf{x}$ and $g(\mathbf{x})$ is a function of the weights to be optimised. For the specific case of a linear decision boundary

$$E(\tilde{\mathbf{w}}) = \frac{1}{2}\sum_{\forall \tilde{\mathbf{x}}} \left(\tilde{\mathbf{w}}'\tilde{\mathbf{x}} - t(\tilde{\mathbf{x}})\right)^2$$

where $t(\tilde{\mathbf{x}})$ is the target value for observation $\tilde{\mathbf{x}}$ (so $t(\tilde{\mathbf{x}}) = t(\mathbf{x})$). For LMS *all* observation contribute to the cost function (contrast to the perceptron algorithm). Again using gradient descent, we find

$$\nabla E(\tilde{\mathbf{w}}) = \sum_{\forall \tilde{\mathbf{x}}} \left(\tilde{\mathbf{w}}'\tilde{\mathbf{x}} - t(\tilde{\mathbf{x}})\right)\tilde{\mathbf{x}}$$

and the update rule is

$$\tilde{\mathbf{w}}[\tau + 1] = \tilde{\mathbf{w}}[\tau] - \eta\sum_{\forall \tilde{\mathbf{x}}} \left(\tilde{\mathbf{w}}[\tau]'\tilde{\mathbf{x}} - t(\tilde{\mathbf{x}})\right)\tilde{\mathbf{x}}$$

This algorithm does not need the training data to be recognised.

Note, LMS is a general algorithm that is not restricted to estimating linear decision boundaries.

Least Mean Squares Estimate

For the case of the linear perceptron we can optimise the least mean squares estimate using the *pseudo-inverse*. Label the extended training sample $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n$. Rather than considering the summation we can form composite matrices from the observation vectors, $\mathbf{X}$, and target values, $\mathbf{t}$. Let

$$\mathbf{X} = \begin{bmatrix} \tilde{\mathbf{x}}_1' \\ \vdots \\ \tilde{\mathbf{x}}_n' \end{bmatrix}$$

and vector of target values

$$\mathbf{t} = \begin{bmatrix} t(\mathbf{x}_1) \\ \vdots \\ t(\mathbf{x}_n) \end{bmatrix}$$

Thus $\mathbf{X}$ is a $n \times (d+1)$ matrix and $\mathbf{t}$ is n-dimensional vector. The least squares criterion may also be written as

$$E(\tilde{\mathbf{w}}) = \frac{1}{2}(\mathbf{X}\tilde{\mathbf{w}} - \mathbf{t})'(\mathbf{X}\tilde{\mathbf{w}} - \mathbf{t})$$

Differentiating gives

$$\nabla E(\tilde{\mathbf{w}}) = \mathbf{X}'(\mathbf{X}\tilde{\mathbf{w}} - \mathbf{t})$$

Equating to zero , the solution $\tilde{\mathbf{w}}[1]$ satisfies

$$\mathbf{X}'\mathbf{X}\tilde{\mathbf{w}}[1] = \mathbf{X}'\mathbf{t}$$

Solutions to LMS

The exact solution obtained with LMS depends on the choice of the target values. The simplest choice of target values is to use

- $t(\mathbf{x}_i) = 1$ if observation $\mathbf{x}_i$ belongs to class ω_1.
- $t(\mathbf{x}_i) = -1$ if observation $\mathbf{x}_i$ belongs to class ω_2.

The standard classification rule may then used.

There are three situations to consider

1. $n < (d+1)$ Solution is *under-specified*, i.e. a set of possible solutions exist to perfectly yield the target values ($E(\tilde{\mathbf{w}}) = 0$).
2. $n = (d+1)$ Assuming that $\mathbf{X}$ is not singular, there is a unique solution to exactly yield the target values ($E(\tilde{\mathbf{w}}) = 0$).
3. $n > (d+1)$ The problem is *over-specified*, it is not possible to exactly get the target values.

The case of most interest is (3) above. A common way of finding the decision boundary is to use the *pseudo-inverse*. Here

$$\tilde{\mathbf{w}}[1] = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{t}$$

Least Squares Example

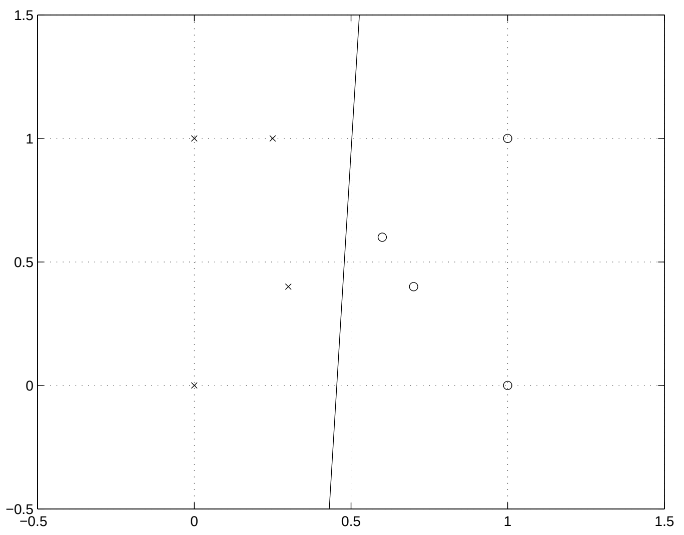
Using points from previous perceptron example we can form the following pair of matrices

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0.6 & 0.6 & 1 \\ 0.7 & 0.4 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0.25 & 1 & 1 \\ 0.3 & 0.4 & 1 \end{bmatrix}, \quad \mathbf{t} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

Using the pseudo inverse we obtain the following estimate

$$\tilde{\mathbf{w}}[1] = \begin{bmatrix} 2.3813 \\ -0.1143 \\ -1.0831 \end{bmatrix}$$

Which gives the decision boundary



Fisher's Discriminant Analysis

A different approach to training the parameters of the perceptron is to use Fisher's discriminant analysis. The basic aim here is to choose the projection that maximises the distance between the class means, whilst minimising the within class variance. Note only the projection $\mathbf{w}$ is determined. The following cost function is used

$$E(\mathbf{w}) = -\frac{(\overline{\mu}_1 - \overline{\mu}_2)^2}{\overline{s}_1 + \overline{s}_2}$$

where $\overline{s}_j$ and $\overline{\mu}_j$ are the projected scatter matrix and mean for class ω_j. The projected scatter matrix is defined as

$$\overline{s}_j = \sum_{\overline{x}_i \in \omega_j} (\overline{x}_i - \overline{\mu}_j)^2$$

The cost function may be expressed as

$$E(\mathbf{w}) = -\frac{\mathbf{w}'\mathbf{S}_B\mathbf{w}}{\mathbf{w}'\mathbf{S}_W\mathbf{w}}$$

where

$$\mathbf{S}_B = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)'$$

and

$$\mathbf{S}_W = \mathbf{S}_1 + \mathbf{S}_2$$

where

$$\mathbf{S}_j = \sum_{\mathbf{x}_i \in \omega_j} (\mathbf{x}_i - \boldsymbol{\mu}_j)(\mathbf{x}_i - \boldsymbol{\mu}_j)'$$

the mean of the class $\boldsymbol{\mu}_j$ is defined as usual.

Fisher's Discriminant Analysis (cont)

Differerentiating $E(\mathbf{w})$ with respect to the weights, we find that it is minimised when

$$(\hat{\mathbf{w}}'\mathbf{S}_B\hat{\mathbf{w}})\mathbf{S}_W\hat{\mathbf{w}} = (\hat{\mathbf{w}}'\mathbf{S}_W\hat{\mathbf{w}})\mathbf{S}_B\hat{\mathbf{w}}$$

From the definition of $\mathbf{S}_B$

$$\begin{aligned} \mathbf{S}_B\hat{\mathbf{w}} &= (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)'\hat{\mathbf{w}} \\ &= \left((\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)'\hat{\mathbf{w}}\right)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \end{aligned}$$

We therefore know that

$$\mathbf{S}_W\hat{\mathbf{w}} \propto (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

Multiplying both sides by $\mathbf{S}_w^{-1}$ yields

$$\hat{\mathbf{w}} \propto \mathbf{S}_W^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

This has given the direction of the decision boundary. However we still need the bias value w_0.

If the data is separable using the Fisher's discriminant it makes sense to select the value of w_0 that *maximises the margin*. Simply put this means that, given no additional information, the boundary should be equidistant from the two points either side of the decision boundary.

Example

Using the previously described data

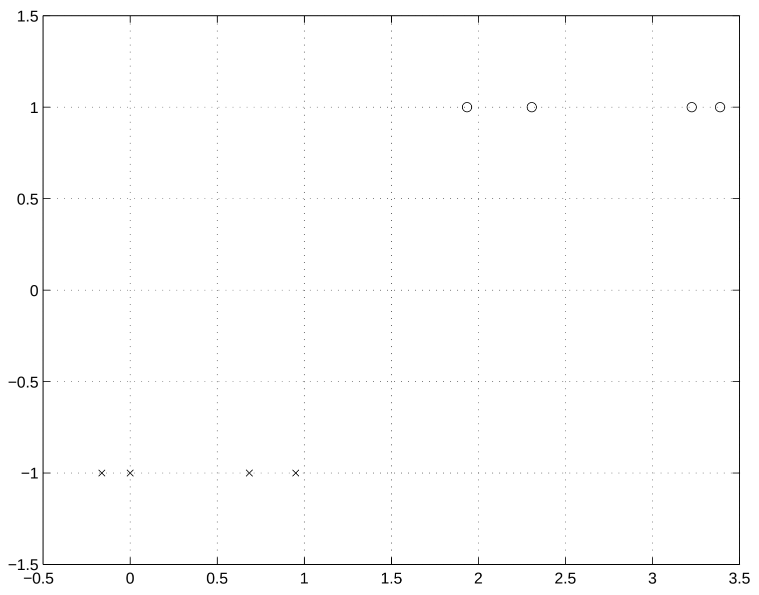
$$\boldsymbol{\mu}_1 = \begin{bmatrix} 0.825 \\ 0.500 \end{bmatrix}; \quad \boldsymbol{\mu}_2 = \begin{bmatrix} 0.1375 \\ 0.600 \end{bmatrix}$$

and

$$\mathbf{S}_W = \begin{bmatrix} 0.2044 & 0.0300 \\ 0.0300 & 1.2400 \end{bmatrix}$$

So solving this yields

$$\hat{\mathbf{w}} = \begin{bmatrix} 3.3878 \\ -0.1626 \end{bmatrix}$$

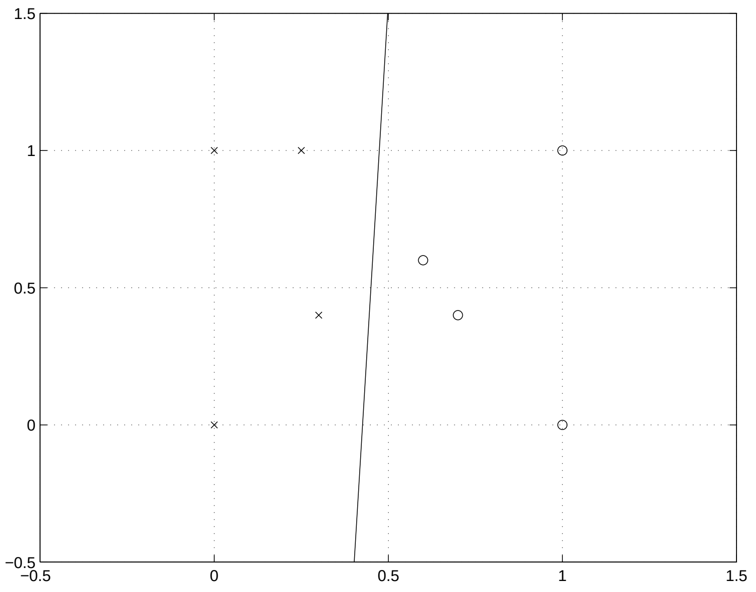


This projection is shown above (with offsets in the y-axis to aid visualisation!).

Example (cont)

It is now necessary to generate a decision boundary. For this separable case the (negative value of the) midpoint between the boundary observations was used.

$$\tilde{\mathbf{w}}[1] = \begin{bmatrix} 3.3878 \\ -0.1626 \\ -1.4432 \end{bmatrix}$$



The decision boundary is shown above. Confirm that you understand the direction of the decision boundary compared to the w.

Kesler's Construct

So far we have only examined binary clasifiers. The direct use of multiple binary classifiers can results in "no-decision" regions (see examples paper).

The multi-class problem can be converted to a 2-class problem. Consider an extended observation $\tilde{\mathbf{x}}$ which belongs to class ω_1. Then to be correctly classified

$$\tilde{\mathbf{w}}_1'\tilde{\mathbf{x}} - \tilde{\mathbf{w}}_j'\tilde{\mathbf{x}} > 0, \quad j = 2, \ldots, K$$

There are therefore $K-1$ inequalities requiring that the $K(d+1)$-dimensional vector

$$\boldsymbol{\alpha} = \begin{bmatrix} \tilde{\mathbf{w}}_1 \\ \vdots \\ \tilde{\mathbf{w}}_K \end{bmatrix}$$

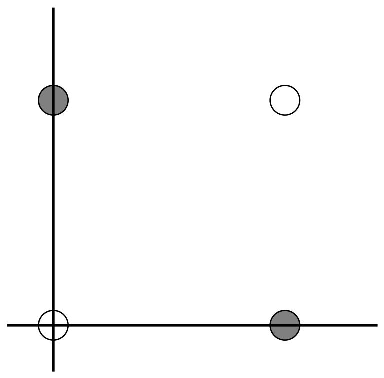
correctly classifies all $K-1$ set of $K(d+1)$-dimensional samples

$$\boldsymbol{\gamma}_{12} = \begin{bmatrix} \tilde{\mathbf{x}} \\ -\tilde{\mathbf{x}} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}, \quad \boldsymbol{\gamma}_{13} = \begin{bmatrix} \tilde{\mathbf{x}} \\ \mathbf{0} \\ -\tilde{\mathbf{x}} \\ \vdots \\ \mathbf{0} \end{bmatrix}, \quad \ldots, \quad \boldsymbol{\gamma}_{1K} = \begin{bmatrix} \tilde{\mathbf{x}} \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ -\tilde{\mathbf{x}} \end{bmatrix}$$

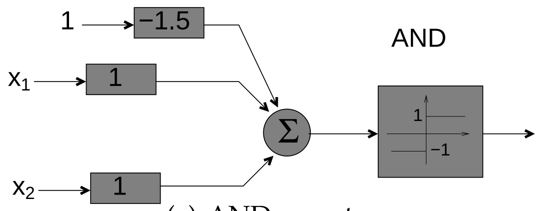
The multi-class problem has now been transformed to a two-class problem at the expense of increasing the effective dimensionality of the data and increasing the number of training samples. We now simply optimise for $\boldsymbol{\alpha}$.

Limitations of Linear Decision Boundaries

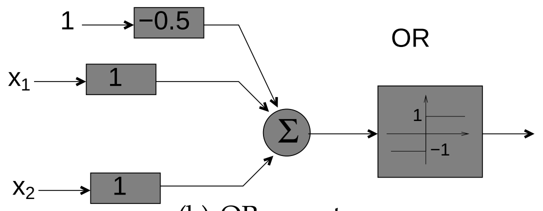
Perceptrons were very popular until the 1960′s when it was realised that it couldn′t solve the XOR problem.



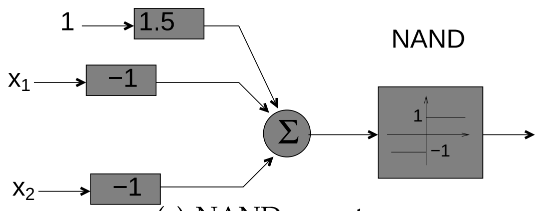
We can use perceptrons to solve the binary logic operators AND, OR, NAND, NOR.



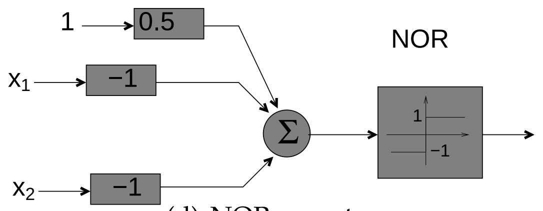
(a) AND operator



(b) OR operator



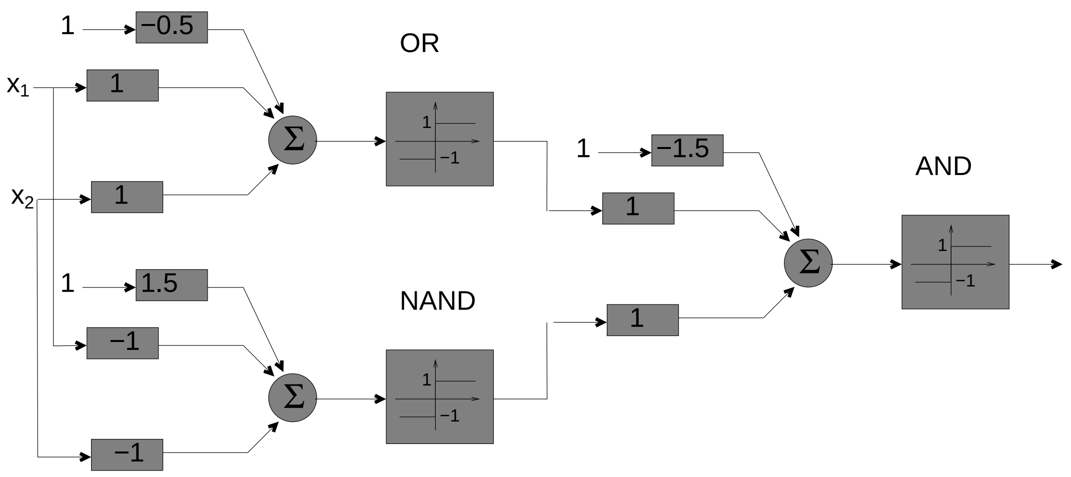
(c) NAND operator



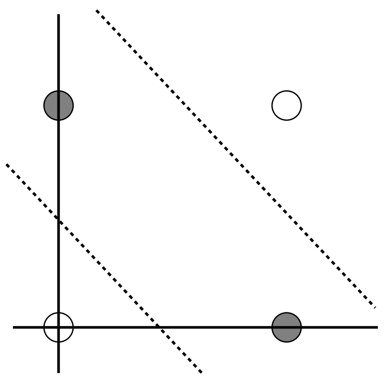
(d) NOR operator

XOR (cont)

But XOR may be written in terms of AND, NAND and OR gates



This yields the decision boundaries



So XOR can be solved using a two-layer network. The problem is how to train multi-layer perceptrons. In the 1980's an algorithm for training such networks was proposed, *error back propagation*.