Chapter 3

Classification

Neural networks and deep learning

Classification Example

Classification is to distinguish *ballet dancers* from *rugby players*.

Two distinctive *features* that can aid in classification:

- ***weight***
- ***height***

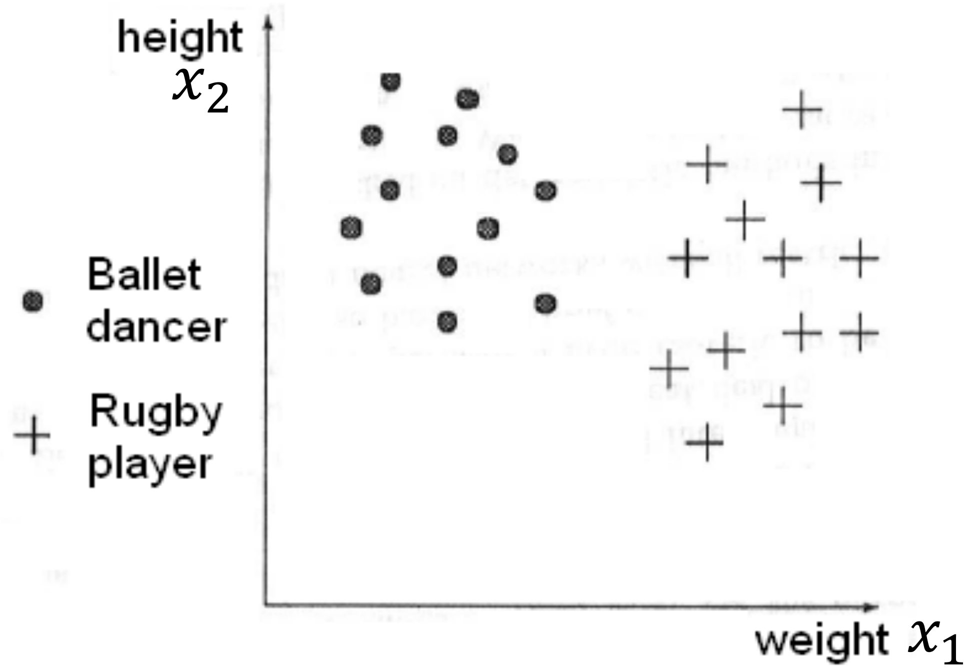


Figure: A 2-dimensional feature space

Let x_1 denote weight and x_2 denote height. Each individual is represented as a point $\boldsymbol{x} = (x_1, x_2)$ in the feature vector.

The figure shows the distribution of height and weight measurements on each individual in the feature space.

Classification Example

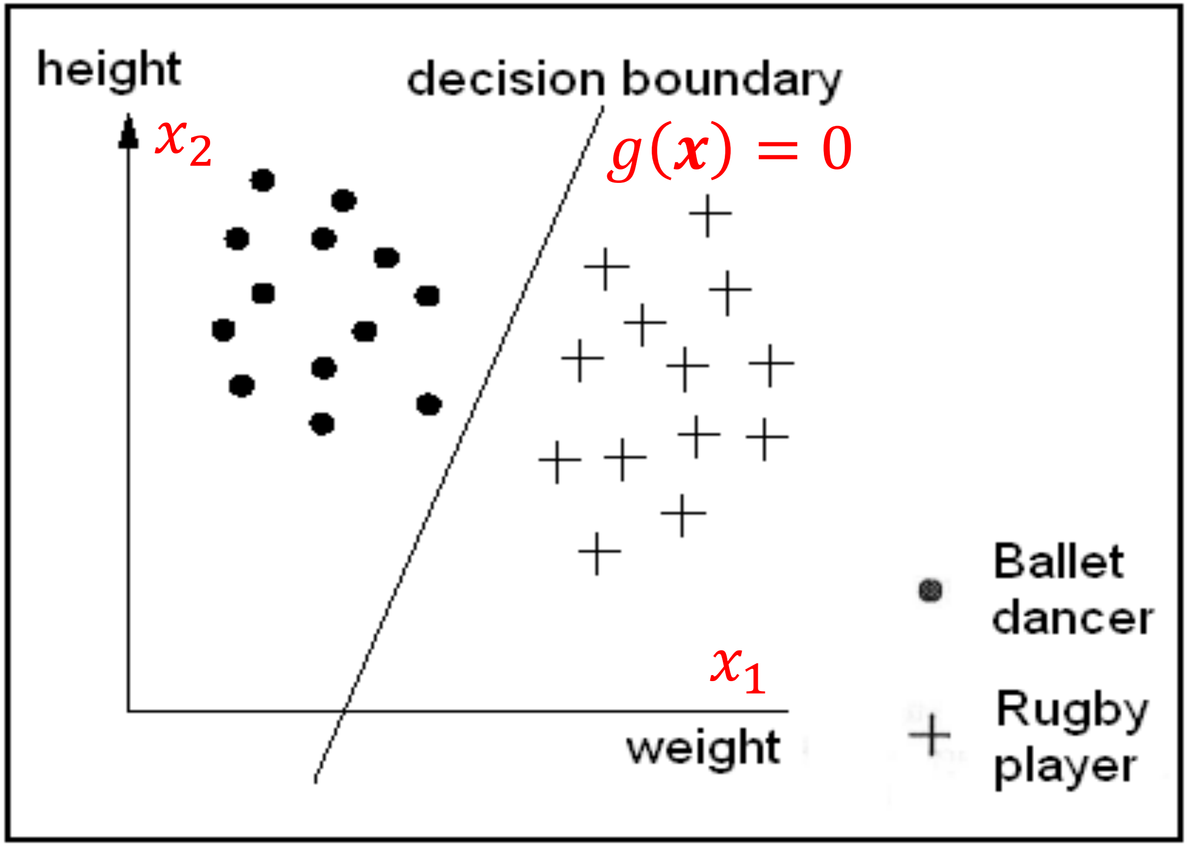


Figure: A linear classification decision boundary

Discriminant Function

The **decision boundary** of the classifier is given by $g(\boldsymbol{x}) = 0$ where $g(\boldsymbol{x})$ is referred to as the **discriminant function**, a function of feature vector $\boldsymbol{x}$.

In the example, the decision boundary separates the two classes. On one side of the decision boundary, discriminant function is positive and on other side, discriminant function is negative.

Therefore, the following class definition may be employed:

If $g(\mathbf{x}) > 0$ => Ballet dancer

If $g(\mathbf{x}) \leq 0$ => Rugby player

Linear Classifier

The objective of classification is to determine a decision boundary that would separate the two clusters, i.e., rugby players and ballet dances.

Humans can easily decide some line or curve drawn between the two classes to separate them (See the previous Figure). But how can such a decision boundary or discrimination function g between them be computed ?

If the two classes can be separated by a straight line, the classification is said to be **linearly separable**.

For linear separable classes, one can design **a linear classifier**. Two class linear classifier can be designed with an artificial neuron with a threshold activation function

Linear Classifier

A linear classifier implements discriminant function or a decision boundary that is represented by a straight line (hyper plane) in the multidimensional **feature space**.

Generally, the feature space is multidimensional. In the multidimensional space, a straight line (hyper plane) is indicated by a linear sum of coordinates.

Given an input (features), $\boldsymbol{x} = (x_1 \quad x_2 \quad \cdots \quad x_n)^T$. The discriminant function of **a linear classifier** is given by a linear function of features:

$$g(\boldsymbol{x}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

where $\mathbf{w} = (w_1 \quad w_2 \quad \cdots \quad w_n)^T$ are the coefficient/weights and w_0 is the constant term.

Linear classifier with an artificial neuron

The linear discriminant function can be implemented by the synaptic input to a neuron

$$g(\boldsymbol{x}) = u = \boldsymbol{x}^T\boldsymbol{w} + b$$

And with a threshold activation function:

$$u = g(\boldsymbol{x}) > 0 \ \rightarrow y = 1 \ \rightarrow class1$$
$$u = g(\boldsymbol{x}) \leq 0 \ \rightarrow y = 0 \rightarrow class2$$

That is, two-class linear classifier can be implemented with an artificial neuron with <u>a threshold (unit step) activation function</u>. The output of the binary neuron represents the **label** of the class.

Linear Classifier

Consider two feature inputs x_1 and x_2 to an artificial neuron. The discriminant boundary in 2-dimensions can be written as a straight line:

$$g(\boldsymbol{x}) = w_1 x_1 + w_2 x_2 + b = 0$$

Or $x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$.

Comparing the equation to the Cartesian system of a straight line (y = mx + c), the slope of line is controlled by ratio of the weight values, w_1 and w_2 and intercept is controlled by the bias term b.

Note that the constant term is needed. Otherwise, the decision boundary always goes through the origin.

With the appropriate values for the weight vector leant, we can indeed determine the position of the decision boundary.

Weight vector is normal to the decision boundary

At the decision boundary:

$$g(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{w} + b = \mathbf{0}$$

Consider two points, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, on the decision surface.

Then $\boldsymbol{x}_1{}^T\boldsymbol{w} + b = \boldsymbol{x}_2{}^T\boldsymbol{w} + b = 0$

and therefore

$$(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T\boldsymbol{w} = 0$$

That is, the weight vector $\boldsymbol{w}$ is normal to the decision boundary (given by the direction of $(\boldsymbol{x}_1 - \boldsymbol{x}_2)$).

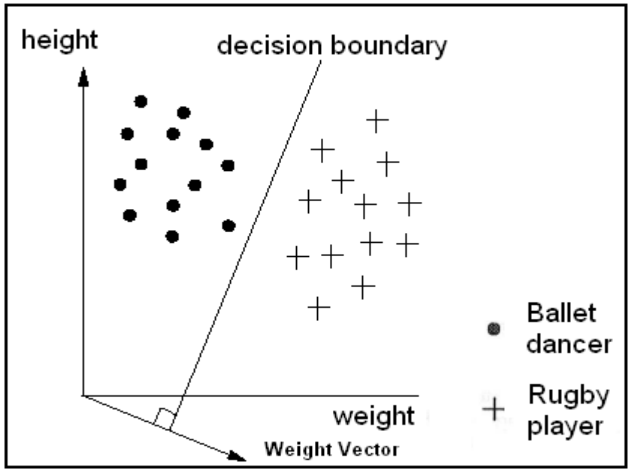
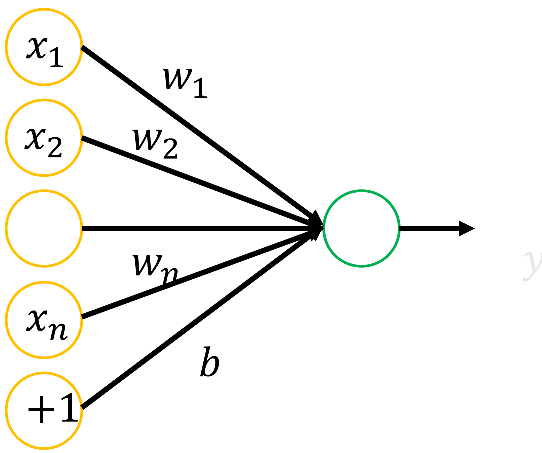


Figure: A linear classification decision boundary

Discrete Perceptron

Simple (or *discrete*) *perceptron* is a neuron that has a **threshold** or **unit-step** activation function.



Output:

$$y = f(u) = \begin{cases} 1, & u > 0 \\ 0, & u \leq 0 \end{cases}$$

where

$$u = \boldsymbol{x}^T \boldsymbol{w} + b$$

Discrete perceptron classifies input patterns into two classes with a linear discriminant function. That is, discrete perceptron is capable of classifying two linearly separable classes (e.g., $y = 0$ for class 1 and $y = 1$ for class 2). It acts as a **two-class classifier** or **dichotomizer**.

Indicator function $1(\cdot)$

Indictor function $1(\cdot)$ takes value 1 when the condition given is True and value 0 when the condition is False.

$$1(x) = \begin{cases} 1, & x \text{ is True} \\ 0, & x \text{ is False} \end{cases}$$

Using the indicator function, the output of a discrete perceptron can be written as:

$$y = 1(u > 0)$$

where $u = \boldsymbol{x}^T \boldsymbol{w} + b$

Discrete Perceptron Learning Algorithm

Discrete perceptron learning algorithm is a supervised scheme. It was proposed by Minsky in 1950 and its convergence can be proved. However, because of non-differentiable characteristics of the activation function, the discrete perceptron learning algorithm cannot be derived from a cost function.

Given P training pairs $\{(\boldsymbol{x}_p, d_p)\}_{p=1}^{P}$
where $\boldsymbol{x}_p \in \boldsymbol{R}^n$ is the n-dimensional input and $d_p \in \{0, 1\}$ is the binary target (desired) output of p th training pattern.

Discrete perceptron leaning algorithm finds **a linear decision boundary** in the feature space.

Discrete Perceptron Learning Algorithm

The change of weights is proportional to the difference (error) between the desired output d and perceptron output y.

For training pattern $(\boldsymbol{x}, d)$:

$$u = \boldsymbol{x}^T \boldsymbol{w} + b$$
$$y = 1(u > 0)$$
$$\delta = d - y$$

Learning:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\delta\boldsymbol{x}$$
$$b \leftarrow b + \alpha\delta$$

Note that $\delta = \{-1, 0, 1\}$.
The learning rate $\alpha \in (0, 1]$
When $\alpha = 1.0$, learning equations are referred to as *simple perceptron rule.*

Discrete Perceptron Learning Algorithm

Given a training dataset $\{(\boldsymbol{x}_p, d_p)\}_{p=1}^{P}$

Set the learning parameter α

Initialize $\boldsymbol{w}$ and b

Repeat until convergence:

For every training pattern $(\boldsymbol{x}_p, d_p)$:

$$u_p = {\boldsymbol{x}_p}^T \boldsymbol{w} + b$$

$$y_p = 1(u_p > 0)$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha(d_p - y_p)\boldsymbol{x}_p$$

$$b \leftarrow b + \alpha(d_p - y_p)$$

Discrete Perceptron Learning

Batch learning	Stochastic learning
$(\boldsymbol{X}, \boldsymbol{d})$	$(\boldsymbol{x}_p, d_p)$
$\boldsymbol{u} = \boldsymbol{X}\boldsymbol{w} + b\boldsymbol{1}_P$	$u_p = \boldsymbol{x}_p{}^T\boldsymbol{w} + b$
$\boldsymbol{y} = 1(\boldsymbol{u} > 0)$	$y_p = 1(u_p > 0)$
$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\boldsymbol{X}^T(\boldsymbol{d} - \boldsymbol{y})$	$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha(d_p - y_p)\boldsymbol{x}_p$
$b \leftarrow b + \alpha\boldsymbol{1}_P{}^T(\boldsymbol{d} - \boldsymbol{y})$	$b \leftarrow b + \alpha(d_p - y_p)$

The learning equations have the same form of linear neurons.
Note that $\boldsymbol{d}$ and $\boldsymbol{y}$ are vectors of binary numbers.

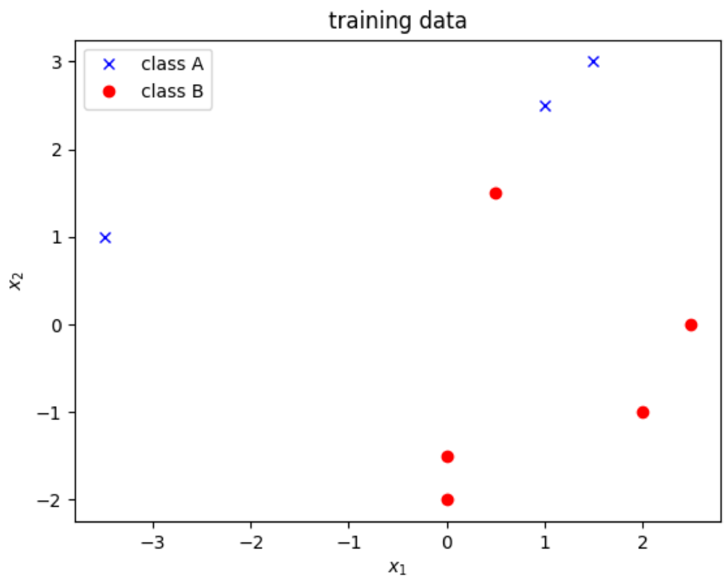
Example 1

Train a perceptron to classify the following 2-dimensional patterns, using the discrete perceptron learning algorithm:

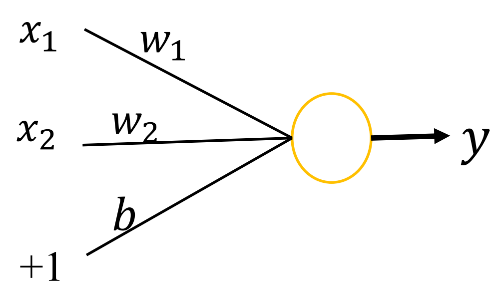
$$(1.0 \quad 2.5) \rightarrow class\ B$$
$$(2.0 \quad -1.0) \rightarrow class\ A$$
$$(1.5 \quad 3.0) \rightarrow class\ B$$
$$(0.0 \quad -1.5) \rightarrow class\ A$$
$$(-3.5 \quad 1.0) \rightarrow class\ B$$
$$(2.5 \quad 0.0) \rightarrow class\ A$$
$$(0.5 \quad 1.5) \rightarrow class\ A$$
$$(0.0 \quad -2.0) \rightarrow class\ A$$

Show two iterations of SGD learning with a learning parameter $\alpha = 0.4$.

Example 1



Note that the two classes are linearly separable.



Let $y = 0$ for $class\ A$
$y = 1$ for $class\ B$

Initially, $\boldsymbol{w} = \begin{pmatrix} 0.77 \\ 0.02 \end{pmatrix}$, $b = 0.0$

Learning rate $\alpha = 0.4$

Example 1

$$\boldsymbol{w} = \begin{pmatrix} 0.77 \\ 0.02 \end{pmatrix}, b = 0.0$$
$$y = 1(u > 0)$$

Iteration 1:
Shuffle the inputs

$p = 1$:
$\boldsymbol{x}_1 = \begin{pmatrix} 1.5 \\ 3.0 \end{pmatrix}$, d_1=1:

$u_1 = \boldsymbol{x}_1{}^T \boldsymbol{w} + b = (1.5 \quad 3.0) \begin{pmatrix} 0.77 \\ 0.02 \end{pmatrix} + 0.0 = 1.22$
$y_1 = 1.0$

$\mathbf{w} = \mathbf{w} + \alpha(d_1 - y_1)\boldsymbol{x}_1 = \begin{pmatrix} 0.77 \\ 0.02 \end{pmatrix} + 0.4\times(1-1) \begin{pmatrix} 1.5 \\ 3.0 \end{pmatrix} = \begin{pmatrix} 0.77 \\ 0.02 \end{pmatrix}$
$b = b + \alpha(d_1 - y_1) = 0.0 + 0.4\times(1-1) = 0.0$

Example 1

$p = 2$:

$\boldsymbol{x}_2 = \begin{pmatrix} 0.0 \\ -2.0 \end{pmatrix}$, d_2=0:

$u_2 = \boldsymbol{x}_2{}^T \boldsymbol{w} + b = (0.0 \quad -2.0) \begin{pmatrix} 0.77 \\ 0.02 \end{pmatrix} + 0.0 = -0.04$

$y_2 = 0.0$

$\mathbf{w} = \mathbf{w} + \ \alpha(d_2 - y_2)\boldsymbol{x}_1 = \begin{pmatrix} 0.77 \\ 0.02 \end{pmatrix} + 0.4\times(0 - 0) \begin{pmatrix} 0.0 \\ -2.0 \end{pmatrix} = \begin{pmatrix} 0.77 \\ 0.02 \end{pmatrix}$

$b = b + \alpha(d_1 - y_1) = 0.0 + 0.4\times(0 - 0) = 0.0$

Apply $\boldsymbol{x}_3, \boldsymbol{x}_4, \ldots \boldsymbol{x}_8$, and update **w** and b for each pattern.

Example 1

$\boldsymbol{x}$	$\boldsymbol{d}$	$\boldsymbol{u}$	$\boldsymbol{y}$	$\boldsymbol{w}^{\boldsymbol{new}}$	$b^{\mathbf{new}}$
$\begin{pmatrix} 1.5 \\ 3.0 \end{pmatrix}$	1	1.22	1	$\begin{pmatrix} 0.77 \\ 0.02 \end{pmatrix}$	0.0
$\begin{pmatrix} 0.0 \\ -2.0 \end{pmatrix}$	0	-0.04	0	$\begin{pmatrix} 0.77 \\ 0.02 \end{pmatrix}$	0.0
$\begin{pmatrix} 2.5 \\ 0.0 \end{pmatrix}$	0	1.93	1	$\begin{pmatrix} -0.23 \\ 0.02 \end{pmatrix}$	-0.4
$\begin{pmatrix} 0.5 \\ 1.5 \end{pmatrix}$	0	-0.48	0	$\begin{pmatrix} 0.23 \\ 0.02 \end{pmatrix}$	-0.4
$\begin{pmatrix} -3.5 \\ 1.0 \end{pmatrix}$	1	0.43	1	$\begin{pmatrix} 0.23 \\ 0.02 \end{pmatrix}$	-0.4
$\begin{pmatrix} 0.0 \\ -1.5 \end{pmatrix}$	0	-0.43	0	$\begin{pmatrix} 0.23 \\ 0.02 \end{pmatrix}$	-0.4
$\begin{pmatrix} 0.5 \\ 1.5 \end{pmatrix}$	0	-0.88	0	$\begin{pmatrix} 0.23 \\ 0.02 \end{pmatrix}$	0.4
$\begin{pmatrix} 1.0 \\ 2.5 \end{pmatrix}$	1	-0.58	0	$\begin{pmatrix} 0.17 \\ 1.02 \end{pmatrix}$	0.0

Classification error = $\sum_{p=1}^{8} 1\left(d_p \neq y_p\right) = 2$

Example 1

Then epoch 2 begins,
Shuffle the inputs
apply $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_8$, in random order and for each input, update $\mathbf{w}$ and b

Then epoch 3 begins, apply $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_8$, in a random order and for each input, update $\mathbf{w}$ and b

.

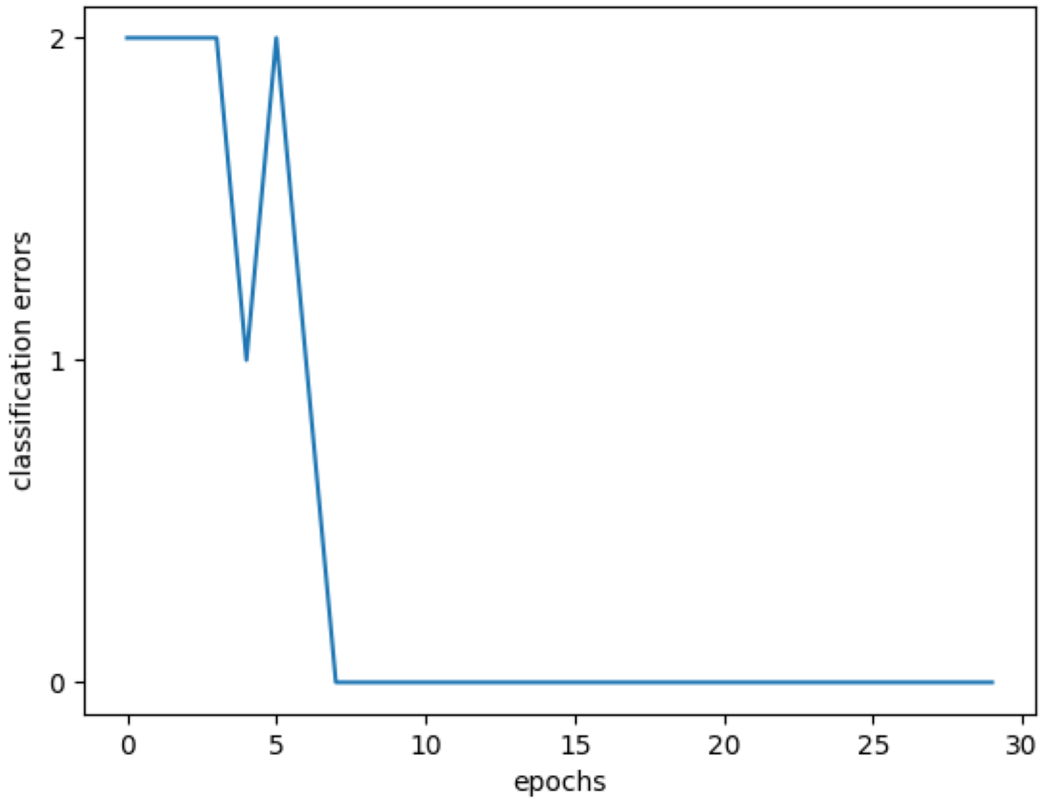
.

.

Iterations (epochs) continue until convergence is achieved.

Note that in each iteration, the patterns are shuffled randomly to change the presentation order.

Example 1

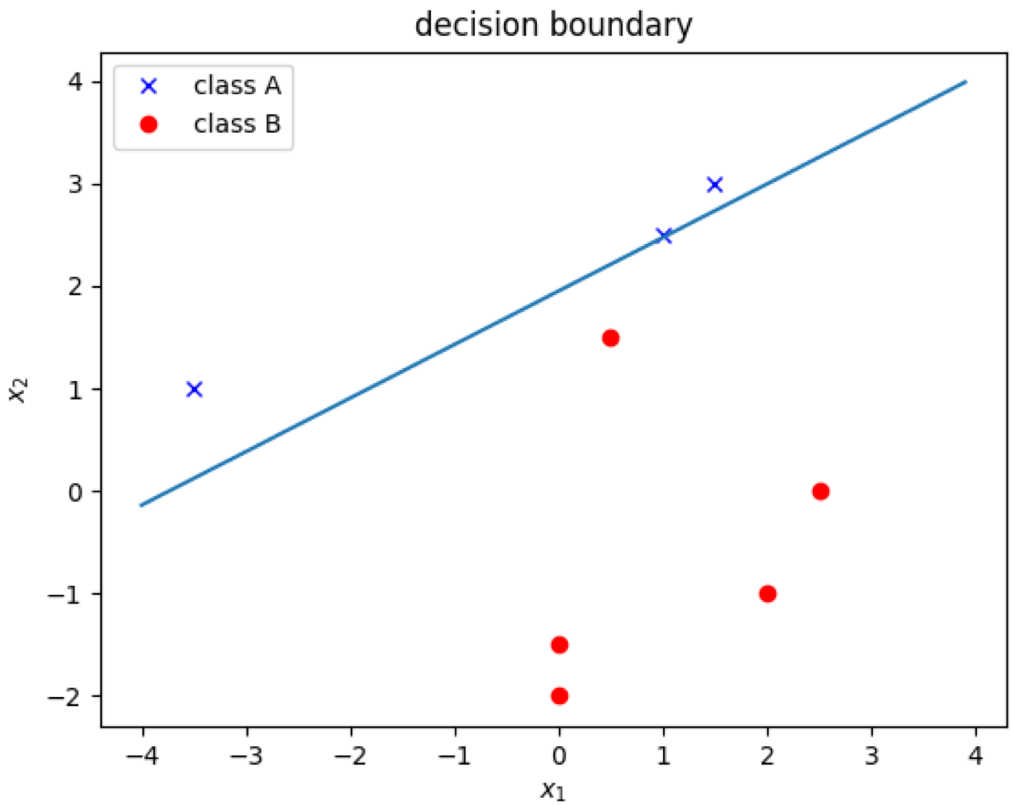


At convergence, $\boldsymbol{w} = \begin{pmatrix} -0.43 \\ 0.82 \end{pmatrix}, b = -1.6$

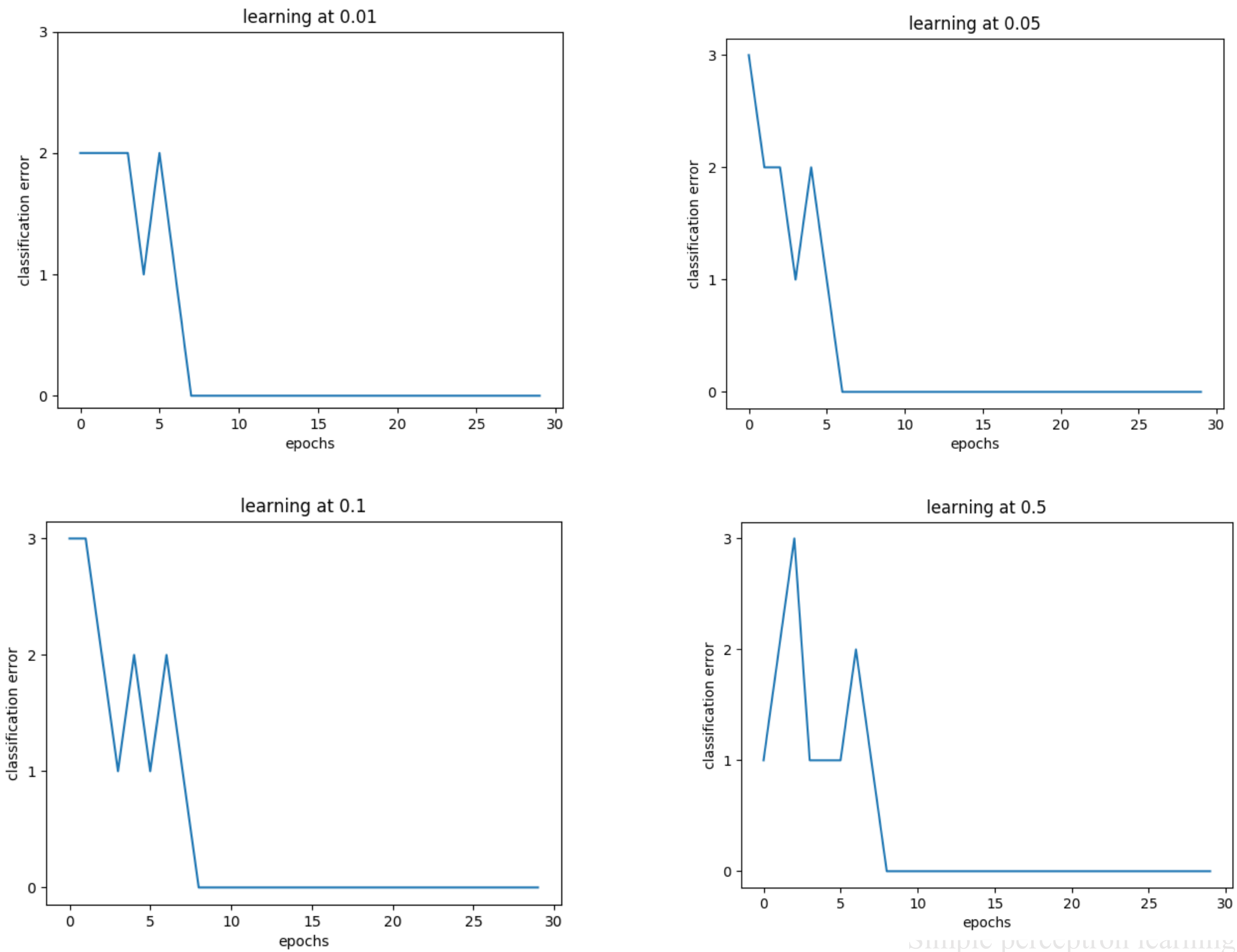
The decision boundary is given by: $\boldsymbol{x}^T \boldsymbol{w} + b = 0$

$$\begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} -0.43 \\ 0.82 \end{pmatrix} - 1.6 = 0$$

$$-0.43x_1 + 0.82x_2 - 1.6 = 0$$



Example 1: learning at different rates



Cross-entropy

Consider a model that produces data belonging to K classes with labels $\{1, 2, \cdots K\}$ at class probabilities $p_1, p_2, \cdots p_K$.

Assume $n_1, n_2, \cdots n_K$ number of data points were observed from each class and data points are independent to one another.

The **likelihood** $p(data|\ model)$ of data given by the model:

$$p(data|model) = {p_1}^{n_1} {p_2}^{n_2} \cdots {p_K}^{n_K} = \prod_{k=1}^{K} {p_k}^{n_k}$$

The **negative log-likelihood** or **cross entropy** is given by:

$$-log\big(p(data|model)\big) = -\sum_{k=1}^{K} n_k log\, p_k$$

Cross-entropy

The **negative log-likelihood** or **cross entropy** is given by:

$$-log\big(p(data|model)\big) = -\sum_{k=1}^{K} n_k log\, p_k$$

The maximum likelihood principal states that the likelihood of data should be maximized by the model.

That is, the cross-entropy needs to be minimized.

Cross-entropy is often used as cost function for neural networks learning classification tasks.

Cross-entropy

Dividing cross-entropy by the total number $N = n_1 + n_2 + \cdots + n_K$ of data points:

$$cross - entropy \propto -\sum_{k=1}^{K} \frac{n_k}{N} log\, p_k = -\sum_{k=1}^{K} q_k log\, p_k$$

where $q_k = \frac{n_k}{N}$ gives the probabilities of class k, given by data.

It can be shown that the cross-entropy is minimum when $p_k = q_k$ for all k.

In other words, the probabilities given by the model (i.e., classifier) is same as those implied by data when cross-entropy is minimum.

That is, cross-entropy can be used as the cost function of a classifier. The optimal weights and biases obtained by minimizing the cross entropy learns a classifier that fits the model of data.

Logistic regression neuron

A **logistic regression neuron** performs a binary classification of inputs. That is, it classify inputs into two classes with labels 0 and 1'. The activation of a logistic regression neuron gives the probability of the neuron output belonging to class '1'.

Given an input $\boldsymbol{x}$, the activation of the neuron gives $P(y = 1|\boldsymbol{x})$:

$$f(u) = P(y = 1|\boldsymbol{x}) = \frac{1}{1 + e^{-u}}$$

where $u = \boldsymbol{w}^T\boldsymbol{x} + b$ is the synaptic input to the neuron and y is the output.

That is, the activation function of the neuron is given by the sigmoid (or logistic function). Note that the output y of the neuron is not equal to the activation.

The activation of neuron gives the probability of the input belonging to class "1".

And

$$P(y = 0|\boldsymbol{x}) = 1 - P(y = 1|\boldsymbol{x}) = 1 - f(u)$$

Logistic Regression Neuron

A logistic regression neuron receives an input $\boldsymbol{x} \in \boldsymbol{R}^n$ and produces a class label $y \in \{0, 1\}$ as the output.

The activation function give the probability that the output class label $y = 1$. That is,

$$f(u) = \frac{1}{1 + e^{-u}} = P(y = 1|\boldsymbol{x})$$

The output y of the neuron is given by:

$$y = 1(f(u) > 0.5)$$

Decision boundary is given by $u = 0$.

SGD for logistic regression neuron

Given a training pattern $(\boldsymbol{x}, d)$ where $\boldsymbol{x} \in \boldsymbol{R}^n$ and $d \in \{0,1\}$.

The cost function for classification is given by the *cross-entropy*:

$$J = -d\log\big(f(u)\big) - (1-d)\log\big(1-f(u)\big)$$

where $u = \boldsymbol{x}^T\boldsymbol{w} + b$ and $f(u) = \frac{1}{1+e^{-u}}$.

The cost function J is minimized using the gradient descent procedure.

GD for logistic regression neuron

$$J = -d\ log\big(f(u)\big) - (1-d)log\big(1-f(u)\big)$$

where $u = \boldsymbol{x}^T\boldsymbol{w} + b$ and $f(u) = \frac{1}{1+e^{(-u)}}$.

Gradient with respect to u :

$$\frac{\partial J}{\partial u} = -\frac{\partial}{\partial f(u)}\Big(dlog\big(f(u)\big) + (1-d)log\big(1-f(u)\big)\Big)\frac{\partial f(u)}{\partial u}$$

$$= -\left(\frac{d}{f(u)} - \frac{(1-d)}{1-f(u)}\right)f'(u)$$

Substituting $f'(u) = f(u)\big(1-f(u)\big)$ for sigmoid activation function,

$$\frac{\partial J}{\partial u} = -\frac{d-f(u)}{f(u)\big(1-f(u)\big)}f(u)\big(1-f(u)\big) = -\big(d-f(u)\big)$$

SGD for logistic regression neuron

Substituting $\frac{\partial J}{\partial u}$:

$$\nabla_{\boldsymbol{w}} J = \frac{\partial J}{\partial u}\frac{\partial u}{\partial \boldsymbol{w}} = -\big(d - f(u)\big)\boldsymbol{x} \qquad \text{(A)}$$

$$\nabla_b J = \frac{\partial J}{\partial u}\frac{\partial u}{\partial b} = -\big(d - f(u)\big) \qquad \text{(B)}$$

Gradient learning equations:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla_{\boldsymbol{w}} J$$

$$b \leftarrow b - \alpha \nabla_b J$$

Substituting $\nabla_{\boldsymbol{w}} J$ and $\nabla_b J$ for logistic regression neuron

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\big(d - f(u)\big)\boldsymbol{x}$$

$$b \leftarrow b + \alpha\big(d - f(u)\big)$$

SGD for logistic regression neuron

Given a training dataset $\{(\boldsymbol{x}_p, d_p)\}_{p=1}^{P}$

Set learning rate α

Initialize $\boldsymbol{w}$ and b

Iterate until convergence:

For every pattern $(\boldsymbol{x}_p, d_p)$:

Synaptic input $u_p = \boldsymbol{x}_p{}^T\boldsymbol{w} + b$

$f(u_p) = \frac{1}{1+e^{-u_p}}$

$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\left(d_p - f(u_p)\right)\boldsymbol{x}_p$

$b \leftarrow b + \alpha\left(d_p - f(u_p)\right)$

GD for logistic regression neuron

Given a training dataset $\{(\boldsymbol{x}_p, d_p)\}_{p=1}^{P}$ where $\boldsymbol{x}_p \in \boldsymbol{R}^n$ and $d_p \in \{0,1\}$.

The cost function for logistic regression is given by the *cross-entropy* (or *negative log-likelihood*) over all the training patterns:

$$J = -\sum_{p=1}^{P} d_p\, log\left(f(u_p)\right) + (1 - d_p) log\left(1 - f(u_p)\right)$$

where $u_p = {\boldsymbol{x}_p}^T \boldsymbol{w} + b$ and $f(u_p) = \frac{1}{1+e^{-u_p}}$.

The cost function J can be written as

$$J = \sum_{p=1}^{P} J_p$$

where $J_p = d_p log\left(f(u_p)\right) + (1 - d_p) log\left(1 - f(u_p)\right)$ is cross-entropy due to p th pattern.

GD for logistic regression neuron

$$\nabla_{\boldsymbol{w}} J = \sum_{p=1}^{P} \nabla_{\boldsymbol{w}} J_p$$

$$= -\sum_{p=1}^{P} \left(d_p - f(u_p)\right) \boldsymbol{x}_p$$

From (A)

$$= -\left(\left(d_1 - f(u_1)\right)\boldsymbol{x}_1 + \left(d_2 - f(u_2)\right)\boldsymbol{x}_2 + \cdots + \left(d_P - f(u_P)\right)\boldsymbol{x}_p\right)$$

$$= -(\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \cdots \quad \boldsymbol{x}_p) \begin{pmatrix} \left(d_1 - f(u_1)\right) \\ \left(d_2 - f(u_2)\right) \\ \vdots \\ \left(d_P - f(u_P)\right) \end{pmatrix}$$

$$= -\boldsymbol{X}^T\left(\boldsymbol{d} - f(\boldsymbol{u})\right)$$

where $\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1{}^T \\ \boldsymbol{x}_2{}^T \\ \vdots \\ \boldsymbol{x}_P{}^T \end{pmatrix}$, $\boldsymbol{d} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_P \end{pmatrix}$, and $f(\boldsymbol{u}) = \begin{pmatrix} f(u_1) \\ f(u_2) \\ \vdots \\ f(u_P) \end{pmatrix}$

GD for logistic regression neuron

Substituting the gradients in

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla_{\boldsymbol{w}} J$$
$$b \leftarrow b - \alpha \nabla_b J$$

the gradient descent learning for logistic regression neuron is given by

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \boldsymbol{X}^T \left(\boldsymbol{d} - f(\boldsymbol{u})\right)$$
$$b \leftarrow b + \alpha \mathbf{1}_P^T \left(\boldsymbol{d} - f(\boldsymbol{u})\right)$$

Note that $\boldsymbol{y}$ in the discrete perceptron is now replaced with $f(\boldsymbol{u})$ in logistic regression learning equations.

GD for logistic regression neuron

Given training data $(\boldsymbol{X}, \boldsymbol{d})$
Set learning rate α
Initialize $\boldsymbol{w}$ and b
Iterate until convergence:

$$\boldsymbol{u} = \boldsymbol{X}\boldsymbol{w} + b\mathbf{1}_P$$

$$f(\boldsymbol{u}) = \frac{1}{1+e^{-\boldsymbol{u}}}$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \boldsymbol{X}^T\big(\boldsymbol{d} - f(\boldsymbol{u})\big)$$

$$b \leftarrow b + \alpha \mathbf{1}_P^T\big(\boldsymbol{d} - f(\boldsymbol{u})\big)$$

Learning for logistic regression neuron

GD	SGD
$(\boldsymbol{X}, \boldsymbol{d})$	$(\boldsymbol{x}_p, d_p)$
$J(\boldsymbol{w}, b) = -\sum_{p=1}^{P} d_p \log\left(f(u_p)\right) + (1 - d_p)\log\left(1 - f(u_p)\right)$	$J(\boldsymbol{w}, b) = -d_p \log\left(f(u_p)\right) - (1 - d_p)\log\left(1 - f(u_p)\right)$
$\boldsymbol{u} = \boldsymbol{X}\boldsymbol{w} + b\boldsymbol{1}_P$	$u_p = \boldsymbol{x}_p{}^T\boldsymbol{w} + b$
$f(\boldsymbol{u}) = \dfrac{1}{1 + e^{-\boldsymbol{u}}}$	$f(u_p) = \dfrac{1}{1 + e^{-u_p}}$
$\boldsymbol{y} = 1(f(\boldsymbol{u}) > 0.5)$	$y_p = 1\left(f(u_p) > 0.5\right)$
$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\boldsymbol{X}^T(\boldsymbol{d} - f(\boldsymbol{u}))$	$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\left(d_p - f(u_p)\right)\boldsymbol{x}_p$
$b \leftarrow b + \alpha\boldsymbol{1}_P{}^T(\boldsymbol{d} - f(\boldsymbol{u}))$	$b \leftarrow b + \alpha\left(d_p - f(u_p)\right)$

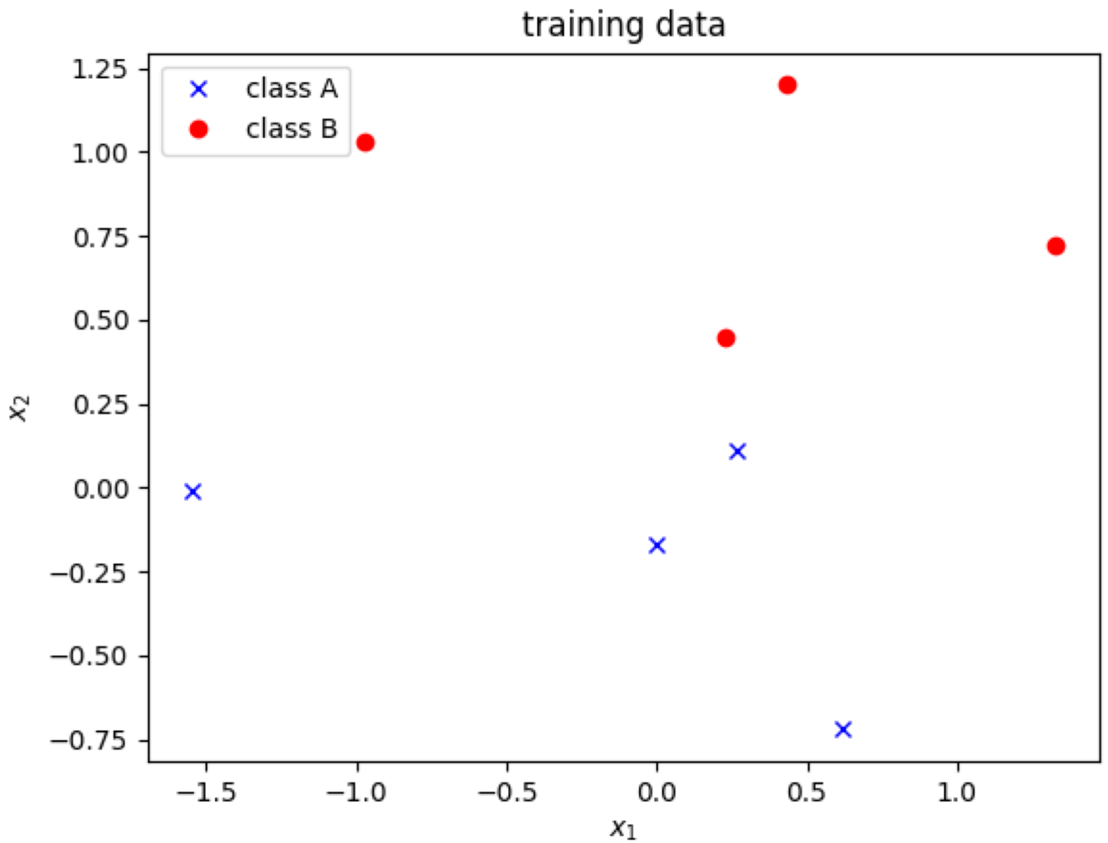
Example 2

Train a logistic regression neuron to perform the following classification:

$$
\begin{gathered}
(1.33 \quad 0.72) \rightarrow class\ B \\
(-1.55 \quad -0.01) \rightarrow class\ A \\
(0.62 \quad -0.72) \rightarrow class\ A \\
(0.27 \quad 0.11) \rightarrow class\ A \\
(0.0 \quad -0.17) \rightarrow class\ A \\
(0.43 \quad 1.2) \rightarrow class\ B \\
(-0.97 \quad 1.03) \rightarrow class\ B \\
(0.23 \quad 0.45) \rightarrow class\ B
\end{gathered}
$$

User a learning factor $\alpha = 0.4$.

Example 2



training data
class A
class B
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
x_2
−1.5
−1.0
−0.5
0.0
0.5
1.0
x_1

Example 2

Let $y = 1$ for class A and $y = 0$ for class B.

$$\boldsymbol{X} = \begin{pmatrix} 1.33 & 0.72 \\ -1.55 & -0.01 \\ 0.62 & -0.72 \\ 0.27 & 0.11 \\ 0.0 & -0.17 \\ 0.43 & 1.2 \\ -0.97 & 1.03 \\ 0.23 & 0.45 \end{pmatrix} \text{ and } \boldsymbol{d} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Initially, $w = \begin{pmatrix} 0.77 \\ 0.02 \end{pmatrix}$, $b = 0.0$ and $\alpha = 0.4$

Example 2

Iteration 1:

$$\boldsymbol{u} = \boldsymbol{X}\boldsymbol{w} + b\mathbf{1} = \begin{pmatrix} 1.33 & 0.72 \\ -1.55 & 0.01 \\ 0.62 & -0.72 \\ 0.27 & 0.11 \\ 0.0 & -0.17 \\ 0.43 & 1.2 \\ -0.97 & 1.03 \\ 0.23 & 0.45 \end{pmatrix} \begin{pmatrix} 0.77 \\ 0.02 \end{pmatrix} + 0.0 \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1.04 \\ -1.2 \\ 0.46 \\ 0.21 \\ 0.00 \\ 0.36 \\ -0.73 \\ 0.19 \end{pmatrix}$$

$$f(\boldsymbol{u}) = \frac{1}{1 + e^{(-\boldsymbol{u})}} = \begin{pmatrix} 0.74 \\ 0.23 \\ 0.61 \\ 0.55 \\ 0.5 \\ 0.59 \\ 0.33 \\ 0.55 \end{pmatrix}$$

Example 2

$$\boldsymbol{w} = \boldsymbol{w} + \alpha \boldsymbol{X}^T\left(\boldsymbol{d} - f(\boldsymbol{u})\right)$$

$$= \begin{pmatrix} 0.77 \\ 0.02 \end{pmatrix} + 0.4 \begin{pmatrix} 1.33 & -1.55 & 0.62 & 0.27 & 0 & 0.43 & -0.97 & 0.23 \\ 0.72 & -0.01 & -0.72 & 0.11 & -0.17 & 1.2 & 1.03 & 0.45 \end{pmatrix} \left(\begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 0.74 \\ 0.23 \\ 0.61 \\ 0.55 \\ 0.5 \\ 0.59 \\ 0.33 \\ 0.55 \end{pmatrix} \right)$$

$$= \begin{pmatrix} 0.69 \\ -0.2 \end{pmatrix}$$

$$b = b + \alpha \mathbf{1}_P^T\left(\boldsymbol{d} - f(\boldsymbol{u})\right)$$

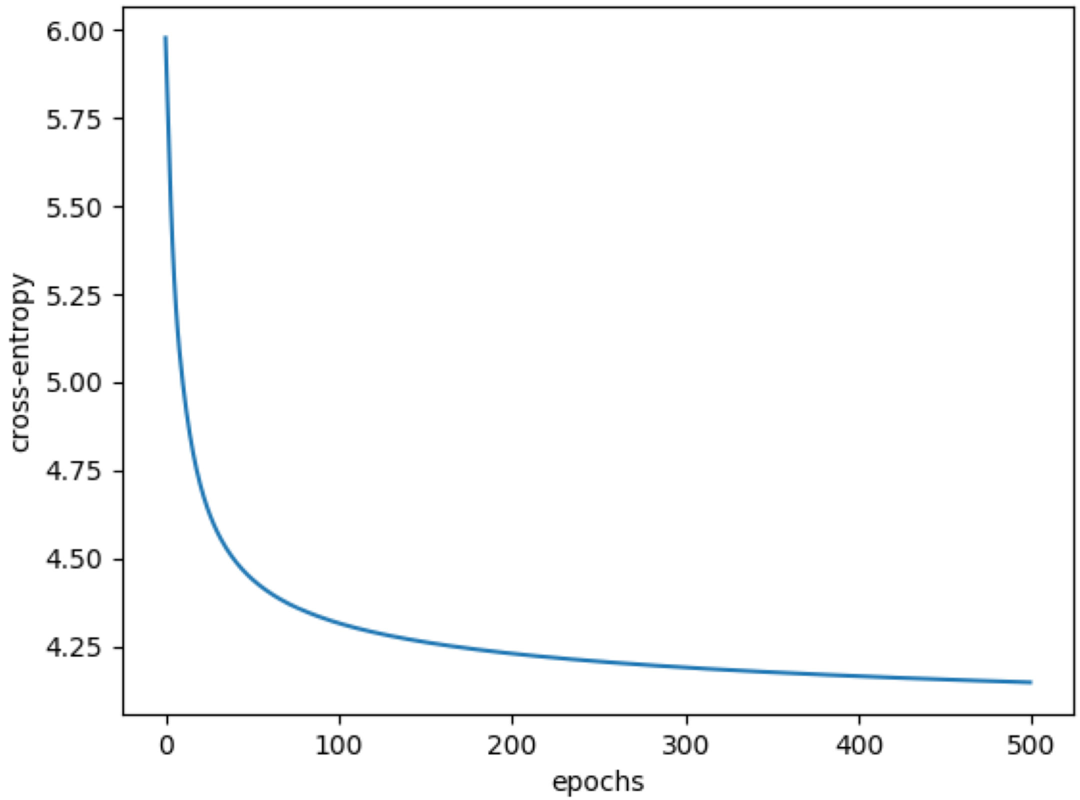
$$= 0.0 + 0.4 (1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1) \left(\begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 0.74 \\ 0.23 \\ 0.61 \\ 0.55 \\ 0.5 \\ 0.59 \\ 0.33 \\ 0.55 \end{pmatrix} \right) = -0.09$$

Example 2

$$\boldsymbol{y} = 1(f(\boldsymbol{u}) > 0.5) = 1\left(\begin{pmatrix} 0.74 \\ 0.23 \\ 0.61 \\ 0.55 \\ 0.5 \\ 0.59 \\ 0.33 \\ 0.55 \end{pmatrix} > 0.5\right) = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

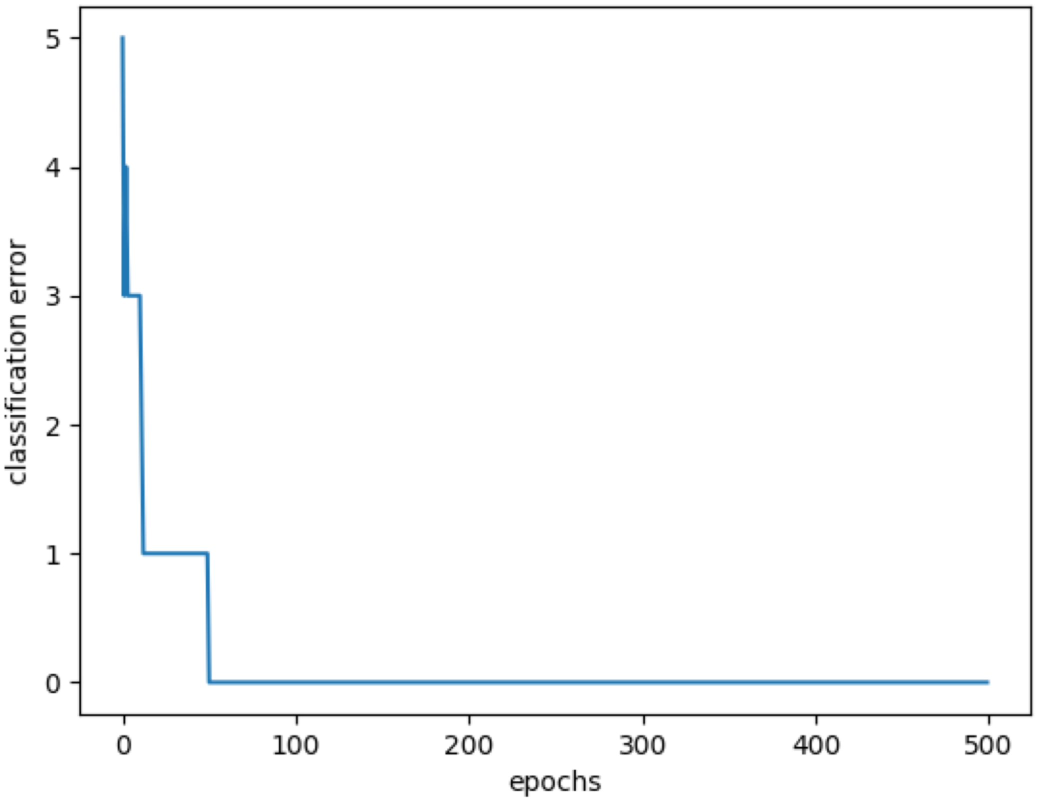
Classification error = $\sum_{p=1}^{8}\left(d_p \neq \; y_p\right) = 5$

Example 2



6.00
5.75
5.50
5.25
5.00
4.75
4.50
4.25
cross-entropy
0
100
200
300
400
500
epochs

Example 2

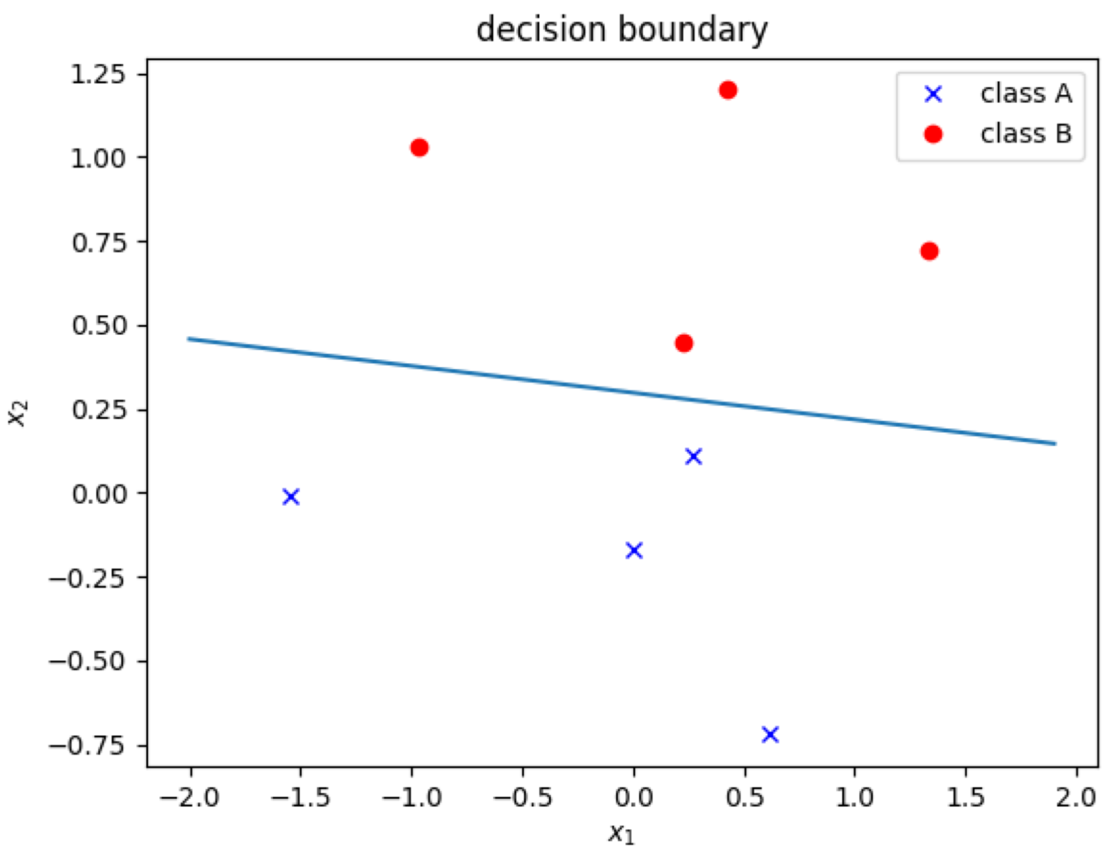


At convergence, $\boldsymbol{w} = \begin{pmatrix} -1.20 \\ -15.02 \end{pmatrix}, b = 4.47$

The decision boundary is given by: $u = \boldsymbol{x}^T \boldsymbol{w} + b = 0$

$$(x_1 \quad x_2)^T \begin{pmatrix} -1.20 \\ -15.02 \end{pmatrix} + 4.47 = 0$$

$$-1.20x_1 - 15.02x_2 + 4.47 = 0$$

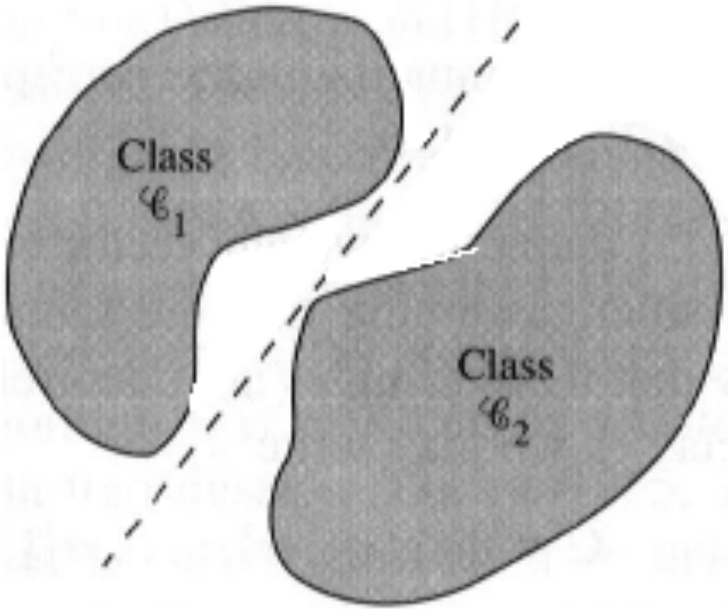


Limitations of Discrete Perceptron and Logistic neuron

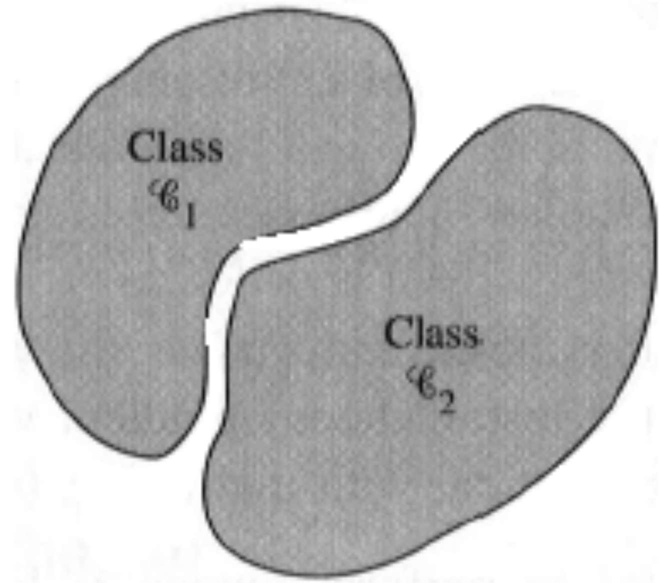
As long as an ANN consists of a *linear combiner* followed by a *non-linear element*, then regardless of the form of non-linearity used, a perceptron can perform pattern classification *only* on *linearly separable* patterns.

Linear separability requires that the patterns to be classified must be sufficiently separated from each other to ensure that the decision boundaries are hyperplanes.

Limitations of Discrete Perceptron and Logistic neuron



(a) Linearly Separable Pattern



(b) Non-Linearly Separable Pattern

Discrete perceptron and logistic regression neuron can create only linear decision boundaries.

Summary of Chapter 3

- Discrete perceptron and logistic regression implements *linear* binary classification (i.e., two class classification) and create linear decision boundaries

- $\boldsymbol{u} = \boldsymbol{X}\boldsymbol{w} + b\boldsymbol{1}_P$

Discrete perceptron:

$$\boldsymbol{y} = 1(\boldsymbol{u} > 0)$$

Learning:

$$\boldsymbol{w} = \boldsymbol{w} + \alpha \boldsymbol{X}^T(\boldsymbol{d} - \boldsymbol{y})$$

$$b = b + \alpha \boldsymbol{1}_P{}^T(\boldsymbol{d} - \boldsymbol{y})$$

Logistic regression:

$$P(\boldsymbol{y} = 1|\boldsymbol{x}) = f(\boldsymbol{u}) = \frac{1}{1+e^{-\boldsymbol{u}}}$$

$$\boldsymbol{y} = 1(f(\boldsymbol{u}) > 0.5\,)$$

Learning:

$$\boldsymbol{w} = \boldsymbol{w} + \alpha \boldsymbol{X}^T\big(\boldsymbol{d} - f(\boldsymbol{u})\big)$$

$$b = b + \alpha \boldsymbol{1}_P{}^T\big(\boldsymbol{d} - f(\boldsymbol{u})\big)$$

Summary: Neurons

Role	Neuron
Regression (one dimensional)	Linear neuron
	Perceptron
Classification (two class)	Discrete perceptron
	Logistic regression neuron

Summary: GD for a neuron

$$(\boldsymbol{X}, \boldsymbol{d})$$
$$\boldsymbol{u} = \boldsymbol{X}\boldsymbol{w} + b\mathbf{1}_P$$
$$\boldsymbol{w} = \boldsymbol{w} - \alpha \boldsymbol{X}^T \nabla_{\boldsymbol{u}} J$$
$$b = b - \alpha \mathbf{1}_P{}^T \nabla_{\boldsymbol{u}} J$$

neuron	$f(\boldsymbol{u}), \boldsymbol{y}$	$\nabla_{\boldsymbol{u}} J$
Discrete perceptron	$\boldsymbol{y} = 1(f(\boldsymbol{u}) > 0)$	$-(\boldsymbol{d} - \boldsymbol{y})$
Logistic regression neuron	$f(\boldsymbol{u}) = \frac{1}{1 + e^{-\boldsymbol{u}}}$ $\boldsymbol{y} = 1(f(\boldsymbol{u}) > 0.5)$	$-(\boldsymbol{d} - f(\boldsymbol{u}))$
Linear neuron	$\boldsymbol{y} = f(\boldsymbol{u}) = \boldsymbol{u}$	$-(\boldsymbol{d} - \boldsymbol{y})$
Perceptron	$\boldsymbol{y} = f(\boldsymbol{u}) = \frac{1}{1 + e^{-\boldsymbol{u}}}$	$-(\boldsymbol{d} - \boldsymbol{y}) \cdot f'(\boldsymbol{u})$

Summary: SGD for a neuron

$$(\boldsymbol{x}_p, d_p)$$
$$u_p = \boldsymbol{x}_p{}^T \boldsymbol{w} + b$$
$$\boldsymbol{w} = \boldsymbol{w} - \alpha \nabla_{u_p} J \boldsymbol{x}_p$$
$$b = b - \alpha \nabla_{u_p} J$$

neuron	$f(u_p),\ y_p$	$\nabla_{u_p} \boldsymbol{J}$
Discrete perceptron	$y_p = 1(u_p > 0)$	$-(d_p - y_p)$
Logistic regression neuron	$f(u_p) = \frac{1}{1 + e^{-u_p}}$ $y_p = 1(f(u_p) > 0.5)$	$-(d_p - f(u_p))$
Linear neuron	$y_p = f(u_p) = u_p$	$-(d_p - y_p)$
Perceptron	$y_p = f(u_p) = \frac{1}{1 + e^{-u_p}}$	$-(d_p - y_p) \cdot f'(u_p)$