Chapter 3

Classification

Neural networks and deep learning

Classification Example

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

Two distinctive *features* that can aid in classification:

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

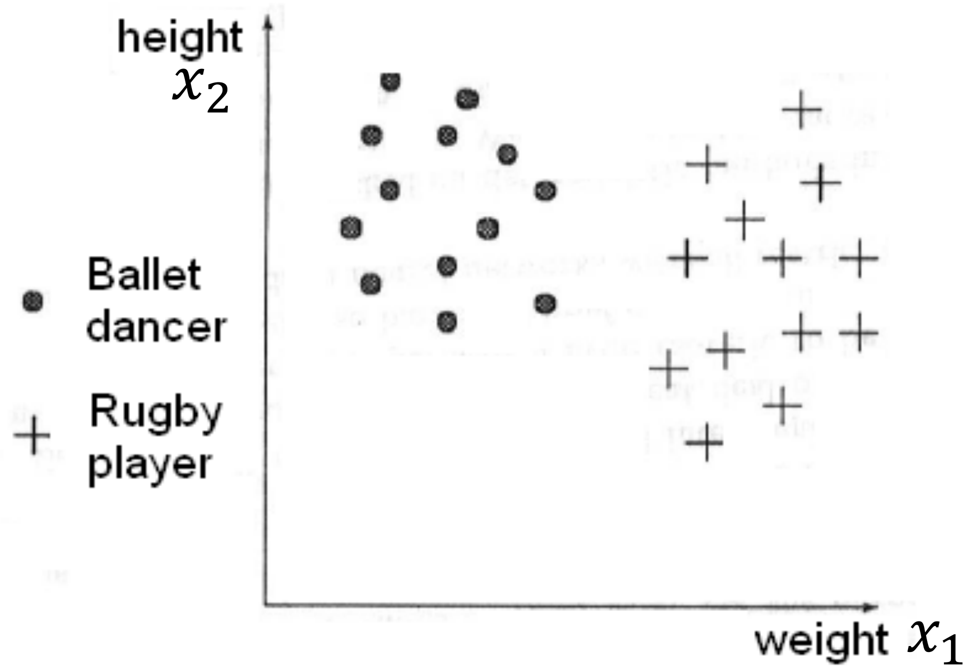


Figure: A 2-dimensional feature space

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 feature space.

Classification Example

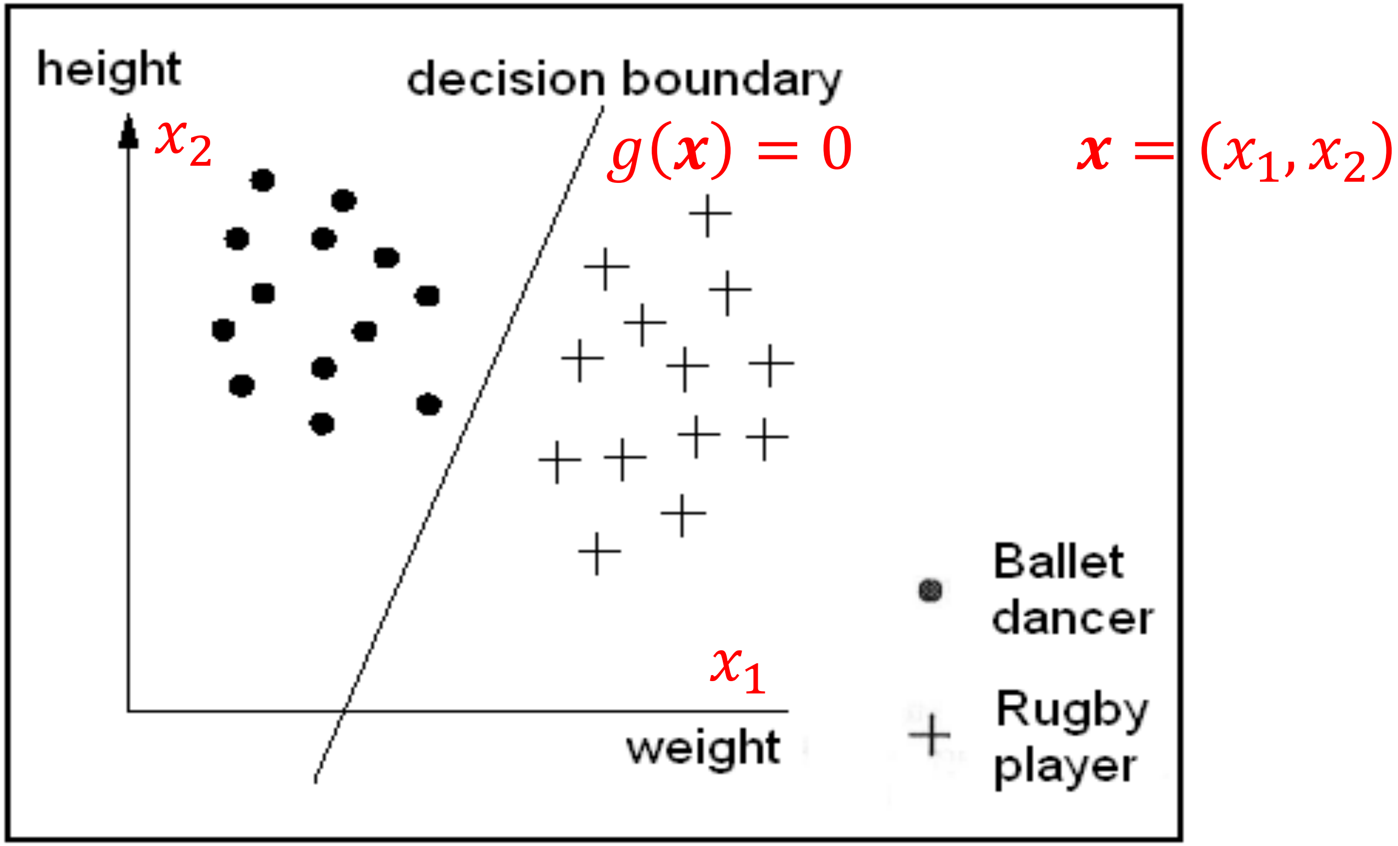


Figure: A linear classification decision boundary

Decision boundary and discriminant function

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

A classifier finds a decision boundary separating the two classes in the feature space. 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

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**.

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 or **hyper plane** is indicated by a linear sum of coordinates.

Given an input (features), $\boldsymbol{x} = (x_1 \quad x_2 \quad \cdots \quad x_n)^T$. A linear description function is given by

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

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 a discrete perceptron

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

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

And with a threshold activation function $1(u)$:

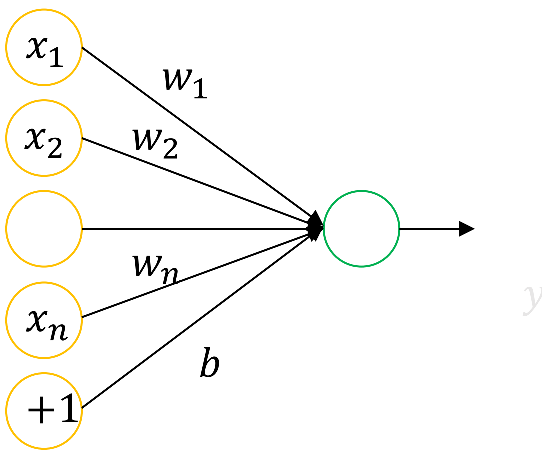
$$u = g(\boldsymbol{x}) > 0 \ \rightarrow y = 1 \ \rightarrow class1$$
$$u = g(\boldsymbol{x}) \leq 0 \ \rightarrow y = 0 \rightarrow class2$$

That is, two-class linear classifier (or a dichotomizer) can be implemented with an artificial neuron with a threshold (unit step) activation function (discrete percepton).

The output, 0 or 1, of the binary neuron represents the **label** of the class.

Discrete Perceptron

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



Output:

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

where

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

Discrete perceptron classifies input patterns into two classes with a linear discriminant function. Discrete perceptron acts as a linear **two-class classifier** or **dichotomizer**.

Discrete Perceptron Learning Algorithm

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 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.

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 $(\boldsymbol{x}, d)$:

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

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

Note that error $\delta = \{-1, 0, 1\}$.
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 $\left\{\left(\boldsymbol{x}_p, d_p\right)\right\}_{p=1}^{P}$

Set the learning parameter α

Initialize $\boldsymbol{w}$ and b

Repeat until convergence:

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

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

$$y_p = 1\left(u_p > 0\right)$$

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

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

Discrete Perceptron Learning

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

The learning equations have a form similar to that of linear neurons. Note that $\boldsymbol{d}$ and $\boldsymbol{y}$ are binary vectors.

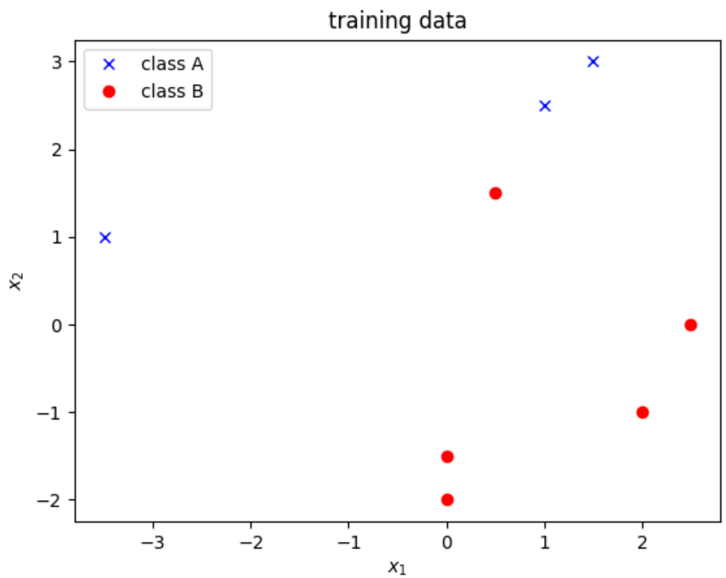
Example 1

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

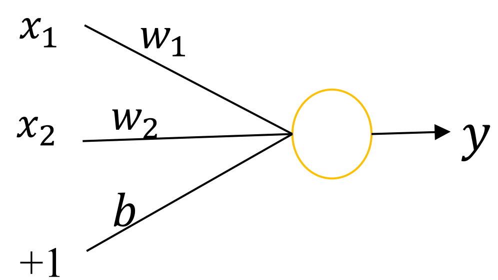
$$(1.0 \quad 2.5) \rightarrow class\ B$$
$$(2.0 \quad -1.0) \rightarrow class\ A$$
$$(1.5 \quad 3.0) \rightarrow class\ B$$
$$(0.0 \quad -1.5) \rightarrow class\ A$$
$$(-3.5 \quad 1.0) \rightarrow class\ B$$
$$(2.5 \quad 0.0) \rightarrow class\ A$$
$$(0.5 \quad 1.5) \rightarrow class\ A$$
$$(0.0 \quad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

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 $class\ B$

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

Learning rate $\alpha = 0.4$

Example 1

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

Epoch 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

$\boldsymbol{x}$	d	u	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 $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_8$, 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

.

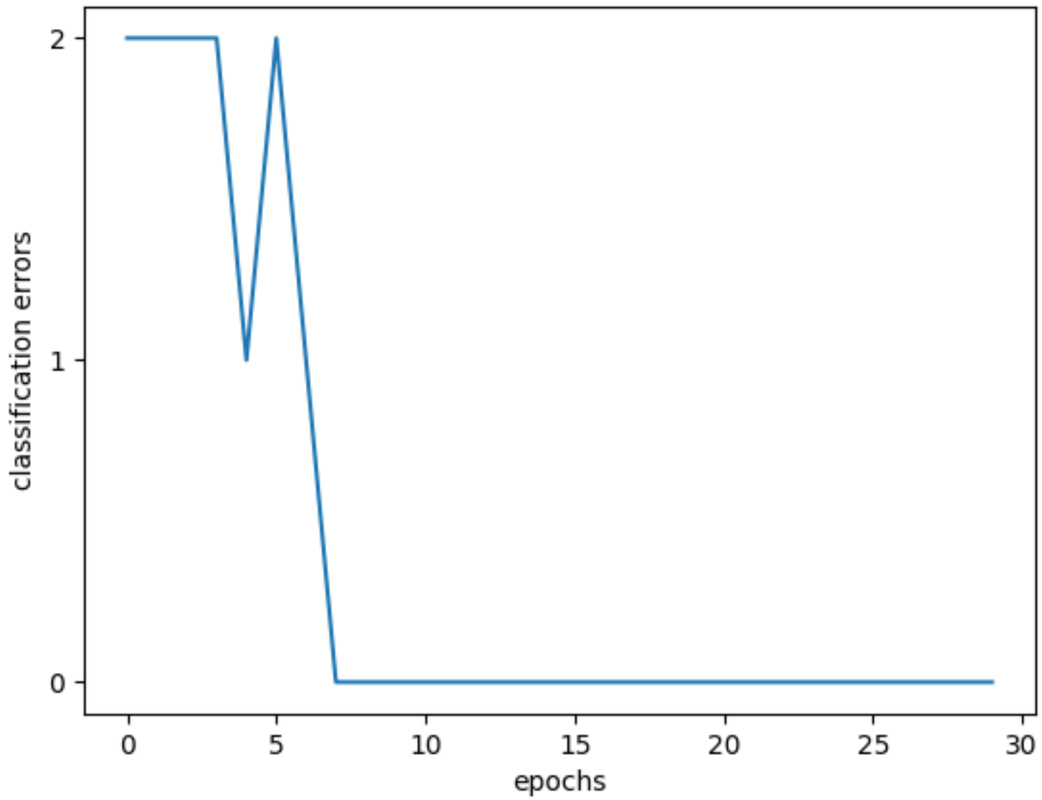
.

.

Iterations (epochs) continue until convergence is achieved.

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

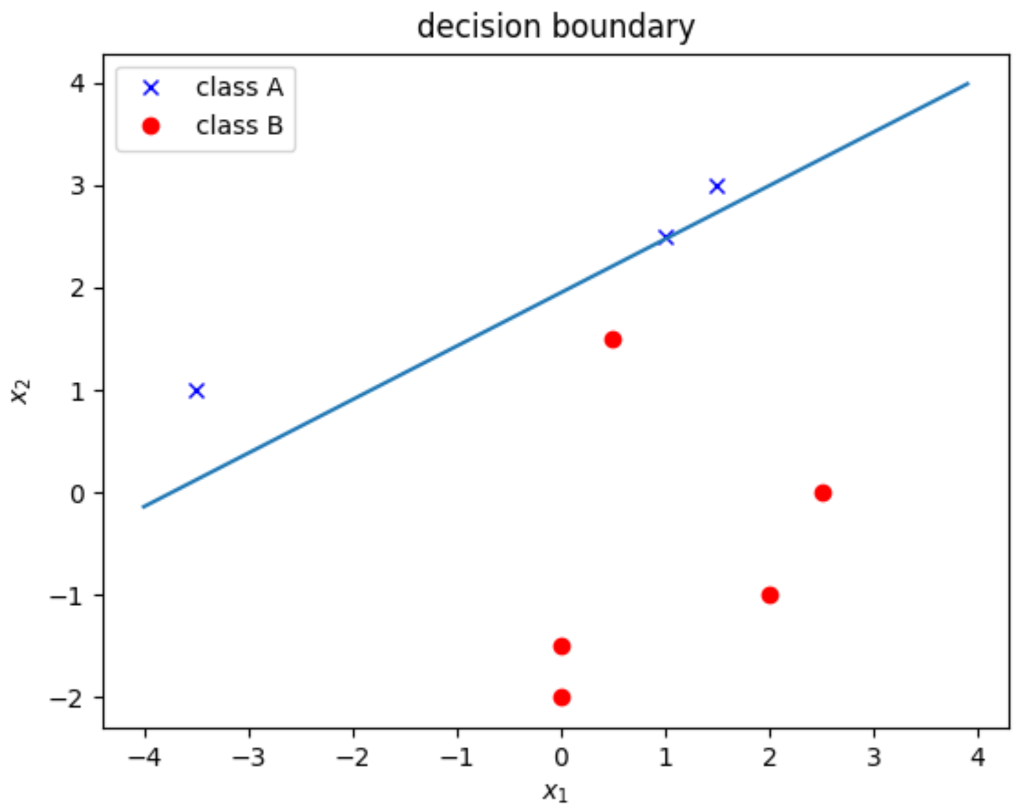
Example 1



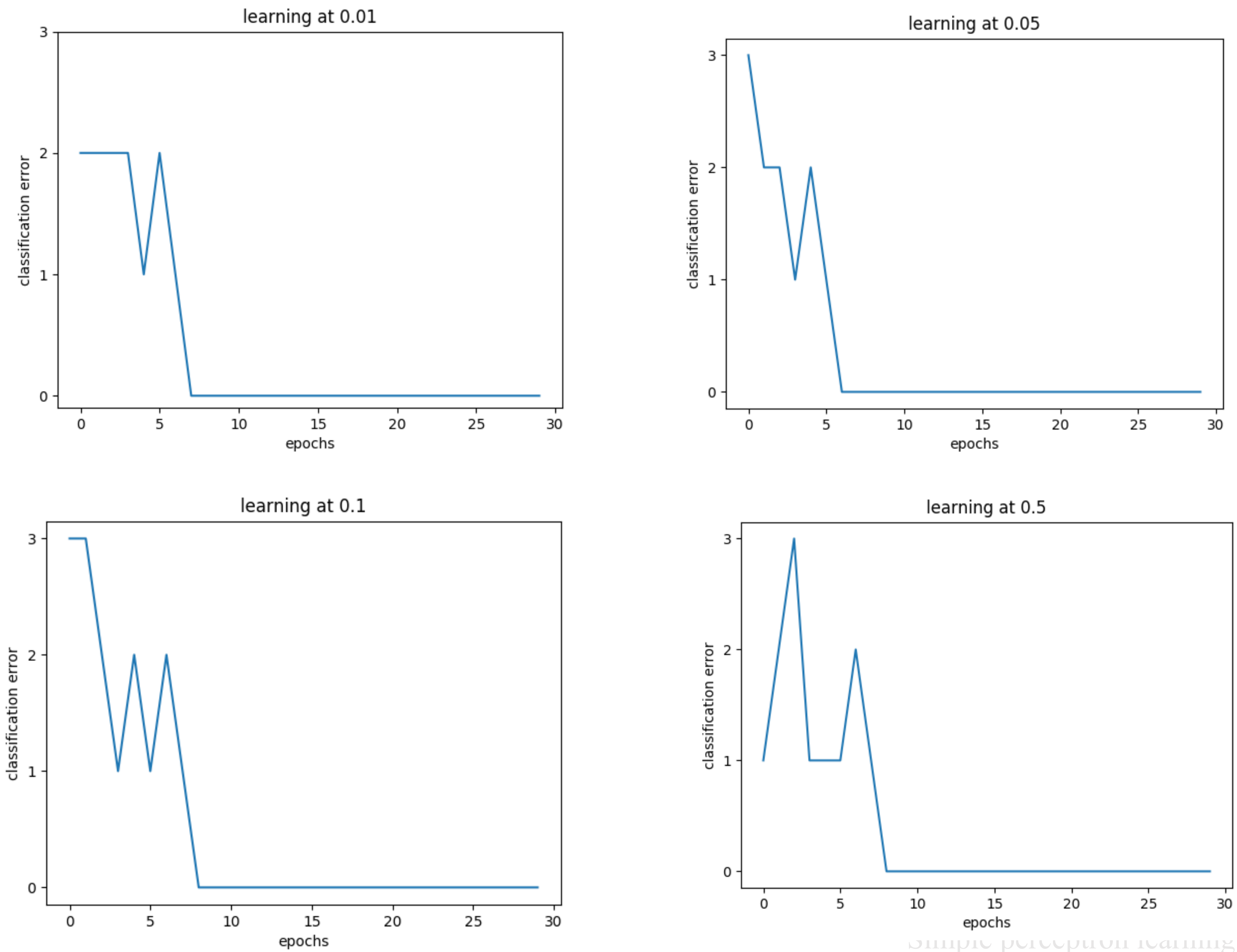
At convergence, $\boldsymbol{w} = \begin{pmatrix} -0.43 \\ 0.82 \end{pmatrix}, b = -1.6$

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 $p_1, p_2, \cdots p_K$.

Suppose $n_1, n_2, \cdots n_K$ number of data points were observed from K classes. Assuming that the 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 maximum likelihood principal states that an appropriate model should maximize the likelihood of data.

Cross-entropy

The **negative log-likelihood** is given by:

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

The term on the righthand side is referred to or **cross entropy.**
Maximizing the likelihood is equivalent for minimizing the cross-entropy.

Cross-entropy is often used as the cost function for neural networks for learning classification tasks. Weights and biases are determined by minimizing the cross-entropy for classifiers

Logistic regression neuron

A **logistic regression neuron** performs a binary classification of inputs. That is, it classifies inputs into two classes with labels '0' and '1'.

The activation function of the logistic regression neuron is given by the sigmoid function:

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

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

The activation of a logistic regression neuron gives the probability of the neuron output belonging to class '1'.

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

Then

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

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.

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

When $u = 0$, $f(u) = P(y = 1|\boldsymbol{x})$= $P(y = 0|\boldsymbol{x})$ = 0.5. That is, the decision boundary is given by $u = 0$.

The output y of the neuron is given by:

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

Note that for logistic neuron, the output and activation are different.

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

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

SGD for logistic regression neuron

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

where $u = \boldsymbol{w}^T\boldsymbol{x} + b$ 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 $\frac{\partial J}{\partial u}$ and $\frac{\partial u}{\partial \boldsymbol{w}} = \boldsymbol{x}$:

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

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

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

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

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

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

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

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

$$
\begin{aligned}
\nabla_{\boldsymbol{w}} J &= \sum_{p=1}^{P} \nabla_{\boldsymbol{w}} J_p \\
&= -\sum_{p=1}^{P} \left(d_p - f(u_p)\right) \boldsymbol{x}_p \\
&= -\left(\left(d_1 - f(u_1)\right)\boldsymbol{x}_1 + \left(d_2 - f(u_2)\right)\boldsymbol{x}_2 + \cdots + \left(d_P - f(u_P)\right)\boldsymbol{x}_p\right) \\
&= -(\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \cdots \quad \boldsymbol{x}_p) \begin{pmatrix} \left(d_1 - f(u_1)\right) \\ \left(d_2 - f(u_2)\right) \\ \vdots \\ \left(d_P - f(u_P)\right) \end{pmatrix} \\
&= -\boldsymbol{X}^T\left(\boldsymbol{d} - f(\boldsymbol{u})\right)
\end{aligned}
$$

From (A)

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

By substituting $\mathbf{1}_P$ for $\boldsymbol{X}$ in above equation:

$$\nabla_{\boldsymbol{b}} J = -\mathbf{1}_P{}^T(\boldsymbol{d} - f(\boldsymbol{u}))$$

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

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

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

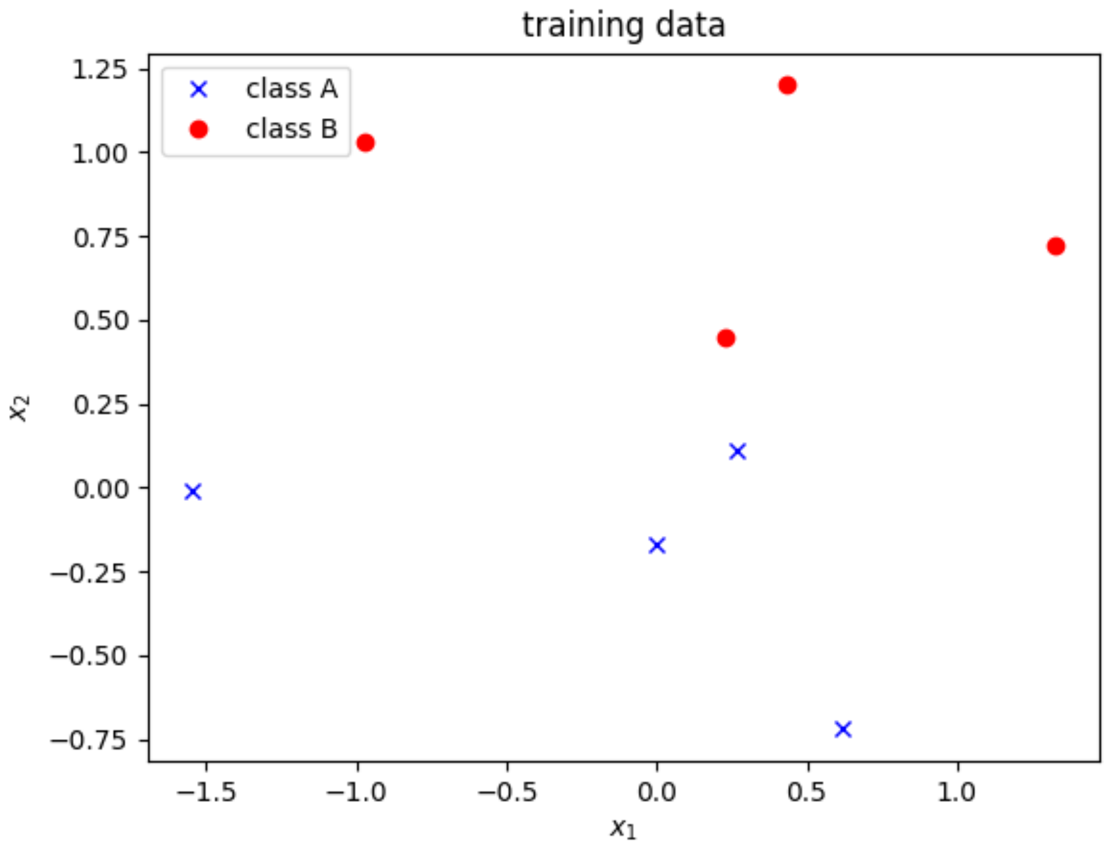
Example 2

Train a logistic regression neuron to perform the following classification, using GD:

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

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

Epoch 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

$$\boldsymbol{d} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

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

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

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

$= -log\left(1 - f(u_1)\right) - logf(u_2) - logf(u_3) - \cdots - log\left(1 - f(u_8)\right)$

$= -log(1 - 0.74) - log0.23 - log0.61 - \cdots - log(1 - 0.55)$

$= 6.653$

Example 2

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

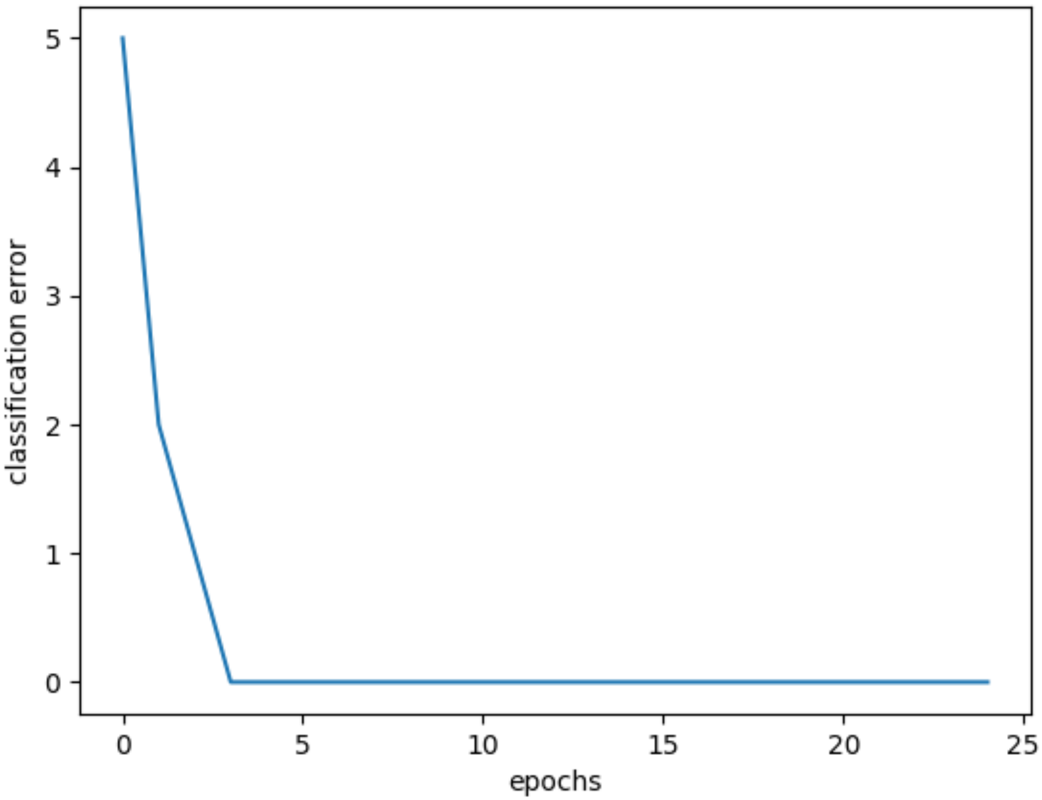
$$= \begin{pmatrix} 0.77 \\ 0.02 \end{pmatrix} + 0.4 \begin{pmatrix} 1.33 & -1.55 & 0.62 & 0.27 & 0 & 0.43 & -0.97 & 0.23 \\ 0.72 & -0.01 & -0.72 & 0.11 & -0.17 & 1.2 & 1.03 & 0.45 \end{pmatrix} \left(\begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 0.74 \\ 0.23 \\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

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

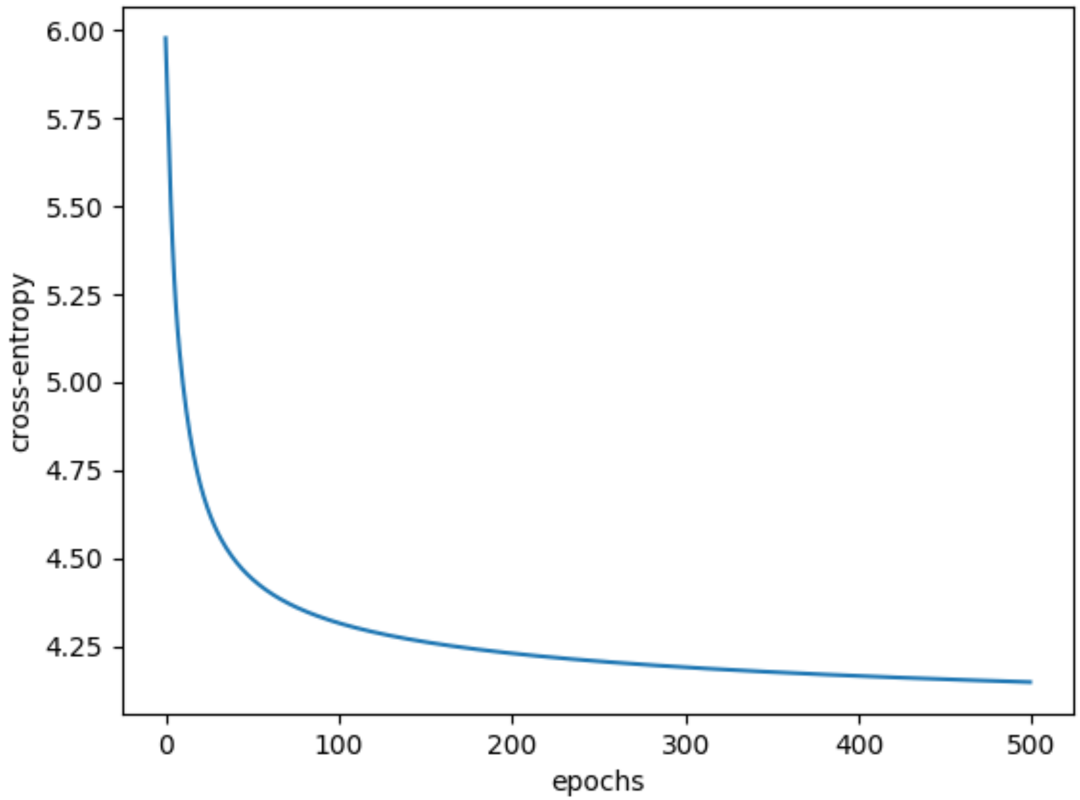
$$b = b + \alpha \mathbf{1}_P^T\left(\boldsymbol{d} - f(\boldsymbol{u})\right)$$

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

Example 2



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

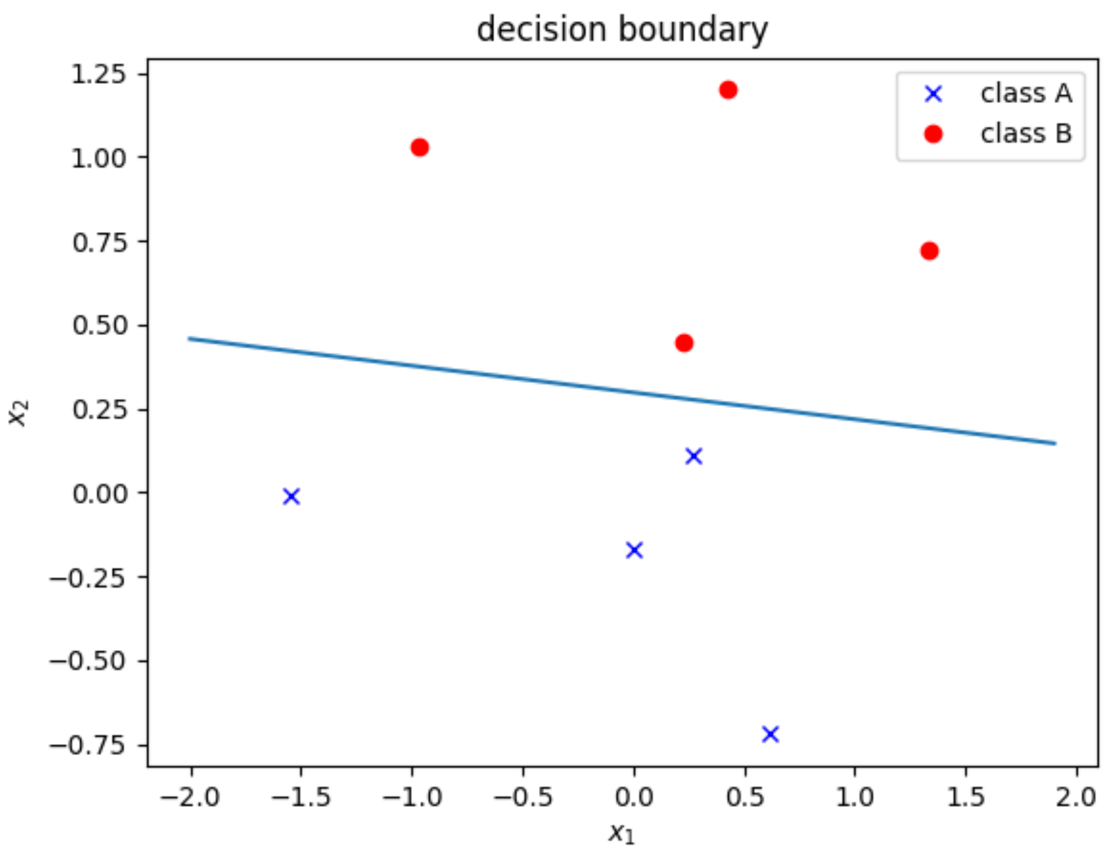
The decision boundary is given by: $u = \boldsymbol{x}^T \boldsymbol{w} + b = 0$

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

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

Decision boundary:

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

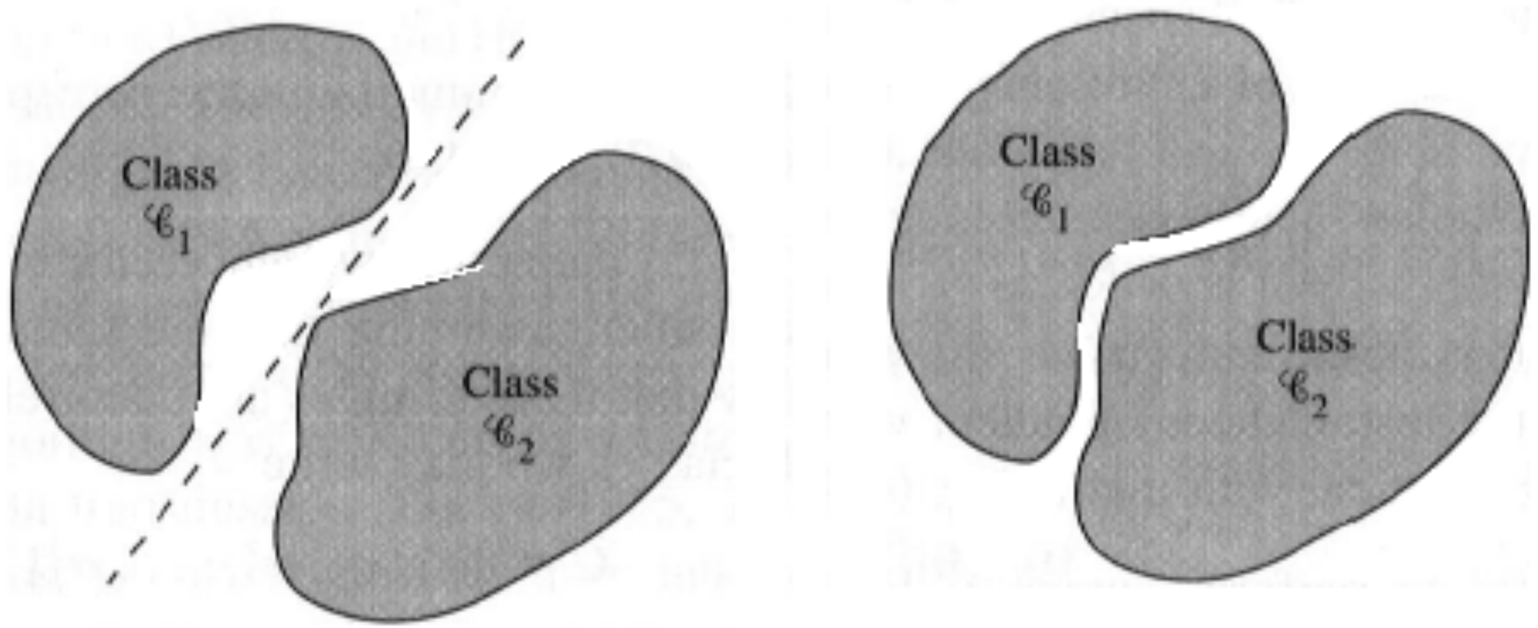


Limitations of Discrete Perceptron and Logistic neuron

As long as the neuron is a *linear combiner* followed by a *non-linear activation function*, then regardless of the form of non-linearity used, the neuron 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:

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

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

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	$\boldsymbol{y} = 1(\boldsymbol{u} > 0)$	$-(\boldsymbol{d} - \boldsymbol{y})$
Logistic regression neuron	$f(\boldsymbol{u}) = \frac{1}{1 + e^{-\boldsymbol{u}}}$ $\boldsymbol{y} = 1(f(\boldsymbol{u}) > 0.5)$	$-\left(\boldsymbol{d} - f(\boldsymbol{u})\right)$
Linear neuron	$\boldsymbol{y} = \boldsymbol{u}$	$-(\boldsymbol{d} - \boldsymbol{y})$
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

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

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	$y_p = u_p$	$-(d_p - y_p)$
Perceptron	$y_p = f(u_p) = \frac{1}{1 + e^{-u_p}}$	$-(d_p - y_p) \cdot f'(u_p)$