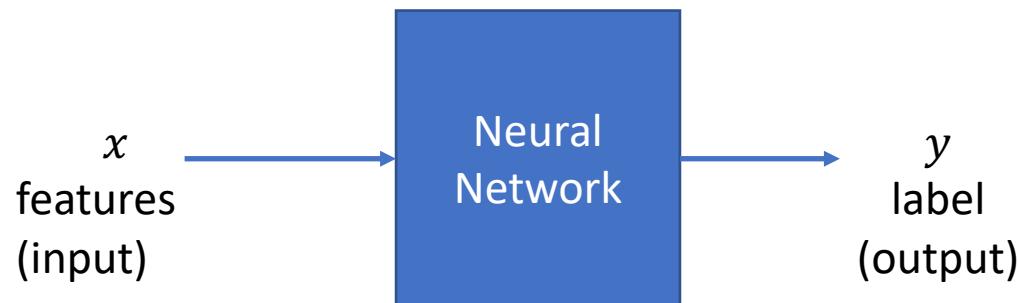


Chapter 2

Regression

Neural networks and deep learning

Regression and classification



The primary application of neural network is to predict **output labels** from **input features**.

In supervised learning, the ability to predict labels from features is achieved by learning from a set of training patterns where target label for every training input is known. Learning finds the weights and biases that are optimal of the network for the **prediction task**.

Prediction tasks can be classified into two categories:

Regression: the labels are continuous (age, income, height, etc.)

Classification: the labels are discrete (sex, digits, type of flowers, etc.),

Linear neuron

Synaptic input u to a neuron is given by

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

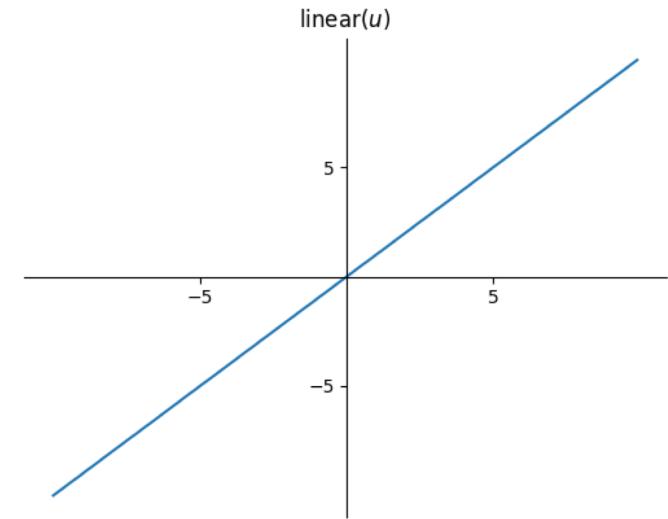
A linear neuron has a *linear activation function*. That is,

$$y = f(u) = u$$

A linear neuron with weight vector $\mathbf{w} = (w_1 \quad w_2 \quad \cdots \quad w_n)^T$ and bias b has an output:

$$y = \mathbf{w}^T \mathbf{x} + b$$

where input $\mathbf{x} = (x_1 \quad x_2 \quad \cdots \quad x_n)^T \in \mathbf{R}^n$ and output $y \in \mathbf{R}$.



Linear regression with linear neuron

Representing a dependent (output) variable as a linear combination of independent (input) variables is known as **linear regression**.

The output of a linear neuron can be written as

$$y = w_1x_1 + w_2x_2 \cdots + w_nx_n + b$$

where x_1, x_2, \dots, x_n are the inputs.

That is, a linear neuron performs linear regression and the weights and biases (that is, b and w_1, \dots, w_n) act as regression coefficients.

Given a training dataset $\{(x_p, d_p)\}_{p=1}^P$ where input $x_p \in \mathbf{R}^n$ and target $d_p \in \mathbf{R}$, training a linear neuron finds a regression function $\phi: \mathbf{R}^n \rightarrow \mathbf{R}$, that is given by the linear mapping:

$$y = \mathbf{w}^T \mathbf{x} + b$$

Stochastic gradient descent (SGD) for linear neuron

The cost function $J(\mathbf{w}, b)$ for regression is usually given as the *square error* (s.e.) between neuron outputs and targets.

Given a training pattern (\mathbf{x}, d) , the square error cost J is defined as

$$J = \frac{1}{2} (d - y)^2$$

where y is neuron output for input pattern \mathbf{x} and

$$y = \mathbf{x}^T \mathbf{w} + b$$

The $\frac{1}{2}$ in the cost function is introduced to simplify learning equations and does not affect the optimal values of the parameters (weights and bias).

SGD for linear neuron



$$J = \frac{1}{2}(d - y)^2$$
$$y = u = \mathbf{x}^T \mathbf{w} + b$$

$$\frac{\partial J}{\partial u} = \frac{\partial J}{\partial y} = -(d - y) \tag{A}$$

$$\nabla_{\mathbf{w}} J = \frac{\partial J}{\partial \mathbf{w}} = \frac{\partial J}{\partial u} \frac{\partial u}{\partial \mathbf{w}} \tag{B}$$

SGD for linear neuron

$$u = \mathbf{w}^T \mathbf{x} + b = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b$$

$$\frac{\partial u}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial u}{\partial w_1} \\ \frac{\partial u}{\partial w_2} \\ \vdots \\ \frac{\partial u}{\partial w_n} \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \mathbf{x} \quad (C)$$

Substituting (A) and (C) in (B),

$$\nabla_{\mathbf{w}} J = -(d - y) \mathbf{x} \quad (D)$$

Similarly, since $\frac{\partial u}{\partial b} = 1$,

$$\nabla_b J = \frac{\partial J}{\partial u} \frac{\partial u}{\partial b} = -(d - y) \quad (E)$$

SGD for linear neuron

Gradient learning equations:

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} J \\ b &\leftarrow b - \alpha \nabla_b J\end{aligned}$$

By substituting from (D) and (E), SGD equations for a linear neuron are given by

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} + \alpha(d - y)\mathbf{x} \\ b &\leftarrow b + \alpha(d - y)\end{aligned}$$

SGD algorithm for linear neuron

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

Set learning parameter α

Initialize w and b

Repeat until convergence:

For every training pattern (x_p, d_p) :

Synaptic input $y_p = x_p^T w + b$

$w \leftarrow w + \alpha(d_p - y_p)x_p$

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

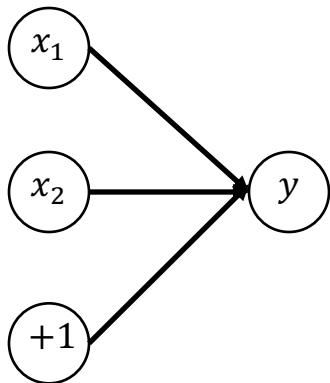
Example 1

Train a linear neuron to perform the following mapping, using stochastic gradient descent (SGD) learning:

$x = (x_1, x_2)$	y
(0.54, -0.96)	1.33
(0.27, 0.50)	0.45
(0.00, -0.55)	0.56
(-0.60, 0.52)	-1.66
(-0.66, -0.82)	-1.07
(0.37, 0.91)	0.30

Use a learning factor $\alpha = 0.01$.

Example 1



Let's initialize weights randomly and biases to zeros

$$\mathbf{w} = \begin{pmatrix} 0.92 \\ 0.71 \end{pmatrix} \text{ and } b = 0.0$$

$$\alpha = 0.01$$

Example 1: epoch 1

Epoch 1 begins

Shuffle the patterns

First pattern $p = 1$ is applied:

$$\mathbf{x}_p = \begin{pmatrix} 0.54 \\ -0.96 \end{pmatrix} \text{ and } d_p = 1.33$$

$$y_p = \mathbf{x}_p^T \mathbf{w} + b = (0.54 \quad -0.96) \begin{pmatrix} 0.92 \\ 0.71 \end{pmatrix} + 0.0 = -0.19$$

$$\text{s.e.} = (d_p - y_p)^2 = 2.292$$

$$\mathbf{w} = \mathbf{w} + \alpha(d_p - y_p)\mathbf{x}_p = \begin{pmatrix} 0.92 \\ 0.71 \end{pmatrix} + 0.01 \times (1.33 + 0.19) \begin{pmatrix} 0.54 \\ -0.96 \end{pmatrix} = \begin{pmatrix} 0.93 \\ 0.70 \end{pmatrix}$$

$$b = b + \alpha(d_p - y_p) = 0 + 0.01 \times (1.33 + 0.19) = 0.02$$

Example 1: epoch 1

Second pattern $p = 2$ is applied:

$$\mathbf{x}_p = \begin{pmatrix} -0.66 \\ -0.82 \end{pmatrix} \text{ and } d_p = -1.07$$

$$y_p = \mathbf{x}_p^T \mathbf{w} + b = (-0.66 \quad -0.82) \begin{pmatrix} 0.93 \\ 0.70 \end{pmatrix} + 0.02 = -0.19$$

$$\text{s.e.} = (d_p - y_p)^2 = 0.01$$

$$\mathbf{w} = \mathbf{w} + \alpha(d_p - y_p)\mathbf{x}_p = \begin{pmatrix} 0.93 \\ 0.70 \end{pmatrix} + 0.01 \times (-1.07 + 0.19) \begin{pmatrix} -0.66 \\ -0.82 \end{pmatrix} = \begin{pmatrix} 0.93 \\ 0.70 \end{pmatrix}$$

$$b = b + \alpha(d_p - y_p) = 0.02 + 0.01 \times (1.33 + 0.19) = 0.02$$

Iterations continues for patterns $p = 3, \dots, 6$.

the second epoch starts $p = 1, 2, \dots, 6$

Training epochs continue until convergence.

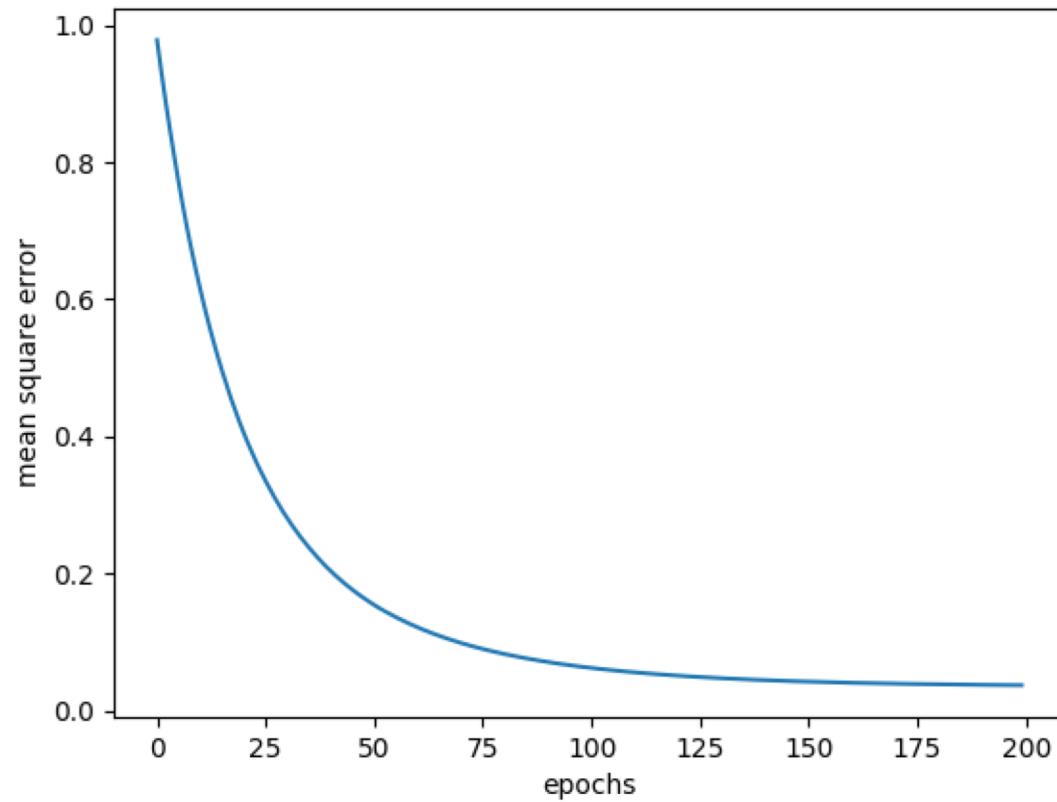
Example 1: epoch 1

x_p	y_p	s.e.	w	b
$\boldsymbol{x}_1 = \begin{pmatrix} 0.54 \\ -0.96 \end{pmatrix}$	-0.19	2.29	$\begin{pmatrix} 0.93 \\ 0.70 \end{pmatrix}$	0.02
$\boldsymbol{x}_2 = \begin{pmatrix} -0.66 \\ -0.82 \end{pmatrix}$	-1.17	0.01	$\begin{pmatrix} 0.93 \\ 0.70 \end{pmatrix}$	0.03
$\boldsymbol{x}_3 = \begin{pmatrix} 0.00 \\ -0.55 \end{pmatrix}$	-0.37	0.87	$\begin{pmatrix} 0.93 \\ 0.69 \end{pmatrix}$	0.03
$\boldsymbol{x}_4 = \begin{pmatrix} 0.27 \\ 0.50 \end{pmatrix}$	0.62	0.03	$\begin{pmatrix} 0.92 \\ 0.69 \end{pmatrix}$	0.02
$\boldsymbol{x}_5 = \begin{pmatrix} -0.60 \\ 0.52 \end{pmatrix}$	-0.17	2.21	$\begin{pmatrix} 0.93 \\ 0.69 \end{pmatrix}$	0.01
$\boldsymbol{x}_6 = \begin{pmatrix} 0.37 \\ 0.91 \end{pmatrix}$	0.98	0.45	$\begin{pmatrix} 0.93 \\ 0.68 \end{pmatrix}$	0.00

Example 1: epoch 200

x_p	y_p	s.e.	w	b
$x_1 = \begin{pmatrix} 0.54 \\ -0.96 \end{pmatrix}$	1.49	0.03	$\begin{pmatrix} 2.00 \\ -0.44 \end{pmatrix}$	-0.01
$x_2 = \begin{pmatrix} 0.00 \\ -0.55 \end{pmatrix}$	0.22	0.12	$\begin{pmatrix} 2.00 \\ -0.44 \end{pmatrix}$	-0.01
$x_3 = \begin{pmatrix} -0.66 \\ -0.82 \end{pmatrix}$	-0.98	0.01	$\begin{pmatrix} 2.00 \\ -0.44 \end{pmatrix}$	-0.01
$x_4 = \begin{pmatrix} 0.37 \\ 0.91 \end{pmatrix}$	0.33	0.00	$\begin{pmatrix} 2.00 \\ -0.44 \end{pmatrix}$	-0.01
$x_5 = \begin{pmatrix} 0.27 \\ 0.50 \end{pmatrix}$	0.30	0.02	$\begin{pmatrix} 2.00 \\ -0.44 \end{pmatrix}$	-0.01
$x_6 = \begin{pmatrix} -0.60 \\ 0.52 \end{pmatrix}$	-1.45	0.04	$\begin{pmatrix} 2.00 \\ -0.44 \end{pmatrix}$	-0.01

Example 1



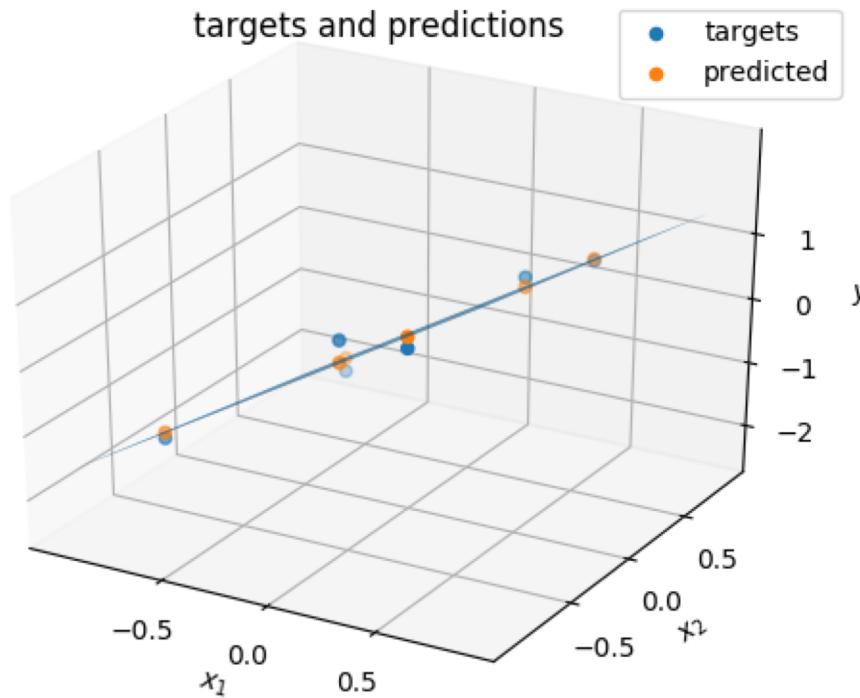
Example 1

At convergence:

$$\mathbf{w} = \begin{pmatrix} 2.00 \\ -0.44 \end{pmatrix}$$
$$b = -0.013$$

Mean square error = 0.037

Example 1



The regression equation:

$$y = \mathbf{x}^T \mathbf{w} + b = (x_1 \quad x_2) \begin{pmatrix} 2.00 \\ -0.44 \end{pmatrix} - 0.013$$

The mapping portrays a hyperplane in the 3-dimensional space:

$$y = 2.00x_1 - 0.44x_2 - 0.013$$

Gradient descent (GD) for linear neuron

Given a training dataset $\{(\mathbf{x}_p, d_p)\}_{p=1}^P$, cost function J is given by the sum of square errors (s.s.e):

$$J = \frac{1}{2} \sum_{p=1}^P (d_p - y_p)^2$$

where y_p is the neuron output for input pattern \mathbf{x}_p .

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

where $J_p = \frac{1}{2} (d_p - y_p)^2$ is the square error for the p th pattern.

GD for linear neuron

From (F):

$$\begin{aligned}\nabla_{\mathbf{w}} J &= \sum_{p=1}^P \nabla_{\mathbf{w}} J_p \\ &= - \sum_{p=1}^P (d_p - y_p) \mathbf{x}_p \\ &= -((d_1 - y_1) \mathbf{x}_1 + (d_2 - y_2) \mathbf{x}_2 + \cdots + (d_P - y_P) \mathbf{x}_P) \\ &= -(\mathbf{x}_1 \quad \mathbf{x}_2 \quad \cdots \quad \mathbf{x}_P) \begin{pmatrix} (d_1 - y_1) \\ (d_2 - y_2) \\ \vdots \\ (d_P - y_P) \end{pmatrix} \\ &= -\mathbf{X}^T (\mathbf{d} - \mathbf{y})\end{aligned}\tag{G}$$

from (D)

where $\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_P^T \end{pmatrix}$ is the data matrix, $\mathbf{d} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_P \end{pmatrix}$ and $\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_P \end{pmatrix}$

GD for linear neuron

Similarly, $\nabla_b J$ can be obtained by considering inputs of +1 and substituting +1 in (G), is given by

$$\nabla_b J = -\mathbf{1}_P^T (\mathbf{d} - \mathbf{y}) \quad (\text{H})$$

where $\mathbf{1}_P = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$ has P elements of 1.

The vector of outputs for the batch of P patterns is given by

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_P \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_P \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1^T \mathbf{w} + b \\ \mathbf{x}_2^T \mathbf{w} + b \\ \vdots \\ \mathbf{x}_P^T \mathbf{w} + b \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_P^T \end{pmatrix} \mathbf{w} + b \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} = \mathbf{X}\mathbf{w} + b \mathbf{1}_P$$

GD for linear neuron

Substituting (G) and (H) in gradient descent equations:

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} J \\ b &\leftarrow b - \alpha \nabla_b J\end{aligned}$$

We get GD learning equations for the linear neuron as

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} + \alpha \mathbf{X}^T (\mathbf{d} - \mathbf{y}) \\ b &\leftarrow b + \alpha \mathbf{1}_P^T (\mathbf{d} - \mathbf{y})\end{aligned}$$

And α is the learning factor.

Where

$$\mathbf{y} = \mathbf{Xw} + b \mathbf{1}_P$$

GD for linear neuron

Given a training dataset(X, d)

Set the learning parameter α

Initialize w and b

Repeat until convergence:

$$y = Xw + b\mathbf{1}_P$$

$$w \leftarrow w + \alpha X^T(d - y)$$

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

Example 2

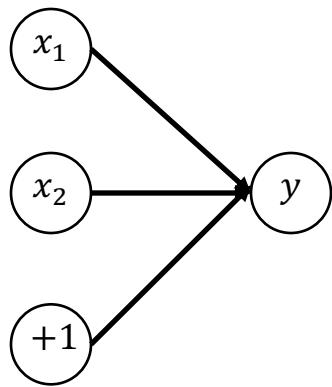
Train a linear neuron to perform the following mapping, using gradient descent learning:

$x = (x_1, x_2)$	y
(0.54, -0.96)	1.33
(0.27, 0.50)	0.45
(0.00, -0.55)	0.56
(-0.60, 0.52)	-1.66
(-0.66, -0.82)	-1.07
(0.37, 0.91)	0.30

Use a learning factor $\alpha = 0.01$.

Example 2

Gradient descent learning:



$$X = \begin{pmatrix} 0.54 & -0.96 \\ 0.27 & 0.50 \\ 0.0 & -0.55 \\ -0.60 & 0.52 \\ -0.66 & -0.82 \\ 0.37 & 0.91 \end{pmatrix}, \mathbf{d} = \begin{pmatrix} 1.33 \\ 0.45 \\ 0.56 \\ -1.66 \\ -1.07 \\ 0.33 \end{pmatrix}$$

Let's initialize:

$$\mathbf{w} = \begin{pmatrix} 0.92 \\ 0.71 \end{pmatrix} \text{ and } b = 0.0$$

$$\alpha = 0.01$$

Example 2

$$\text{Output } \mathbf{y} = \mathbf{X}\mathbf{w} + b\mathbf{1}_P = \begin{pmatrix} 0.54 & -0.96 \\ 0.27 & 0.50 \\ 0.0 & -0.55 \\ -0.60 & 0.52 \\ -0.66 & -0.82 \\ 0.37 & 0.91 \end{pmatrix} \begin{pmatrix} 0.92 \\ 0.71 \end{pmatrix} + 0.0 \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -0.19 \\ 0.60 \\ -0.40 \\ -0.18 \\ -1.20 \\ 0.99 \end{pmatrix}$$

$$\begin{aligned} \text{Mean square error } J &= \frac{1}{6} \sum_{p=1}^6 (d_p - y_p)^2 \\ &= \frac{1}{6} ((1.33 + 0.19)^2 + (0.45 - 0.60)^2 + \dots + (0.33 - 0.99)^2) \\ &= 0.98 \end{aligned}$$

Example 2

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

$$= \begin{pmatrix} 0.92 \\ 0.71 \end{pmatrix} + 0.01 \times \begin{pmatrix} 0.54 & 0.27 & 0.00 & -0.60 & -0.66 & 0.37 \\ -0.96 & 0.50 & -0.55 & 0.52 & -0.82 & 0.91 \end{pmatrix} \left(\begin{pmatrix} 1.33 \\ 0.45 \\ 0.56 \\ -1.66 \\ -1.07 \\ 0.33 \end{pmatrix} - \begin{pmatrix} -0.19 \\ 0.60 \\ -0.40 \\ -0.18 \\ -1.20 \\ 0.99 \end{pmatrix} \right) = \begin{pmatrix} 0.93 \\ 0.68 \end{pmatrix}$$

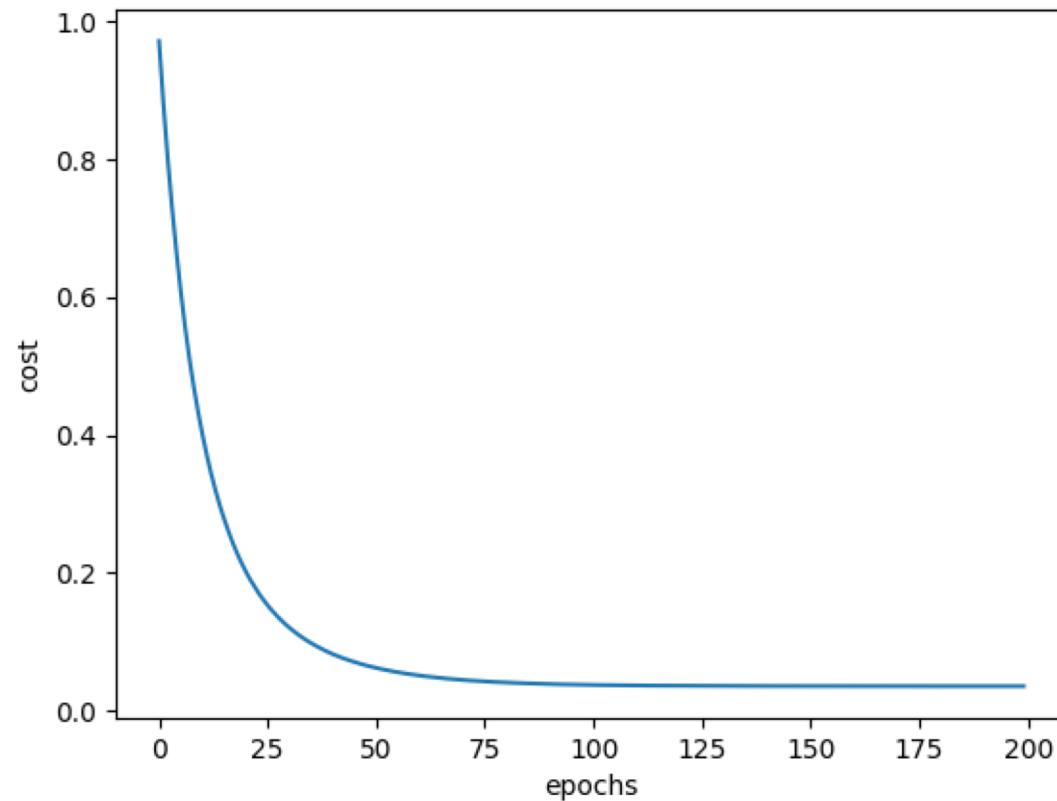
$$b = b + \alpha \mathbf{1}_P^T (\mathbf{d} - \mathbf{y}) = 0.0 + 0.01 \times (1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1) \left(\begin{pmatrix} 1.33 \\ 0.45 \\ 0.56 \\ -1.66 \\ -1.07 \\ 0.33 \end{pmatrix} - \begin{pmatrix} -0.19 \\ 0.60 \\ -0.40 \\ -0.18 \\ -1.20 \\ 0.99 \end{pmatrix} \right) = 0.0$$

Epochs continue with new weights and biases

Example 2

iteration	y	s.e.	w	b
1	$\begin{pmatrix} -0.19 \\ 0.60 \\ -0.40 \\ -0.18 \\ -1.20 \\ 0.99 \end{pmatrix}$	0.98	$\begin{pmatrix} 0.93 \\ 0.68 \end{pmatrix}$	0.0
2	$\begin{pmatrix} -0.14 \\ 0.59 \\ -0.37 \\ -0.21 \\ -1.17 \\ 0.96 \end{pmatrix}$	0.94	$\begin{pmatrix} 0.94 \\ 0.64 \end{pmatrix}$	0.01
200	$\begin{pmatrix} 1.49 \\ 0.30 \\ 0.22 \\ -1.45 \\ -0.98 \\ 0.33 \end{pmatrix}$	0.03	$\begin{pmatrix} 2.00 \\ -0.44 \end{pmatrix}$	-0.01

Example 2



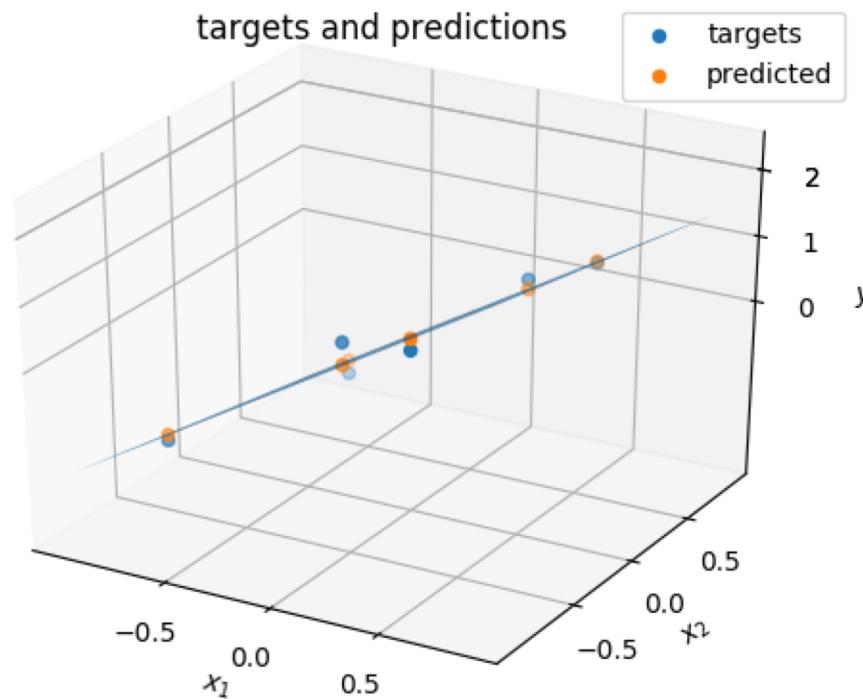
Example 2

At convergence:

$$\mathbf{w} = \begin{pmatrix} 2.00 \\ -0.44 \end{pmatrix}$$
$$b = -0.013$$

Mean square error = 0.037

Example 2



The regression equation:

$$y = \mathbf{x}^T \mathbf{w} + b = (x_1 \quad x_2) \begin{pmatrix} 2.00 \\ -0.44 \end{pmatrix} - 0.013$$

The mapping portrays a hyperplane in the 3-dimensional space:

$$y = 2.00x_1 - 0.44x_2 - 0.013$$

GD and SGD for a linear neuron

GD	SGD
(X, d)	(x_p, d_p)
$J = \frac{1}{2} \sum_{p=1}^P (d_p - y_p)^2$	$J = \frac{1}{2} (d_p - y_p)^2$
$\mathbf{y} = \mathbf{u} = \mathbf{X}\mathbf{w} + b\mathbf{1}_P$	$y_p = u_p = \mathbf{x}_p^T \mathbf{w} + b$
$\mathbf{w} \leftarrow \mathbf{w} + \alpha \mathbf{X}^T (\mathbf{d} - \mathbf{y})$	$\mathbf{w} \leftarrow \mathbf{w} + \alpha (d_p - y_p) \mathbf{x}_p$
$b \leftarrow b + \alpha \mathbf{1}_P^T (\mathbf{d} - \mathbf{y})$	$b \leftarrow b + \alpha (d_p - y_p)$

Perceptron

Perceptron is a neuron having the **sigmoid** activation function and has an output

.

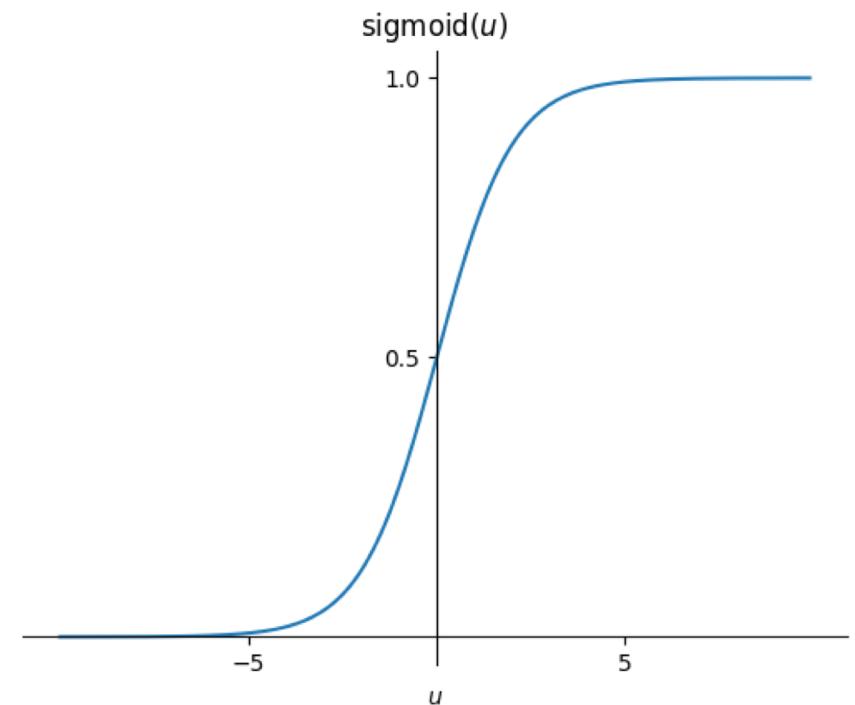
$$y = f(u)$$

where

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

$$\text{And } u = \mathbf{x}^T \mathbf{w} + b$$

The square error is used as cost function for learning.



Perceptron

The training dataset $\{(\mathbf{x}_p, d_p)\}_{p=1}^P$
where $\mathbf{x}_p = (x_{p1}, x_{p2}, \dots x_{pn})^T \in \mathbf{R}^n$ and $d_p \in \mathbf{R}$.

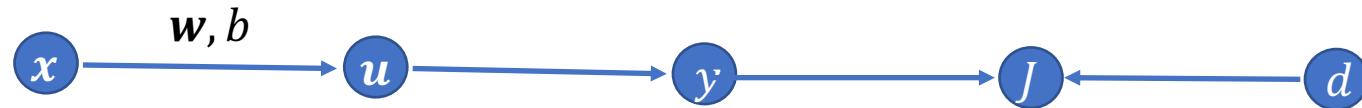
The continuous perceptron finds a functional mapping:

$$\phi: \mathbf{R}^n \rightarrow \mathbf{R}$$

by learning from training data. ϕ is a non-linear function.

Perceptron performs a *non-linear regression* of inputs.

SGD for perceptron



Cost function J is given by

$$J = \frac{1}{2} (d - y)^2$$

where $y = f(u)$ and $u = \mathbf{x}^T \mathbf{w} + b$

The gradient with respect to the synaptic input:

$$\frac{\partial J}{\partial u} = \frac{\partial J}{\partial y} \frac{\partial y}{\partial u} = -(d - y) f'(u)$$

From (C), $\frac{\partial u}{\partial \mathbf{w}} = \mathbf{x}$.

$$\nabla_{\mathbf{w}} J = \frac{\partial J}{\partial u} \frac{\partial u}{\partial \mathbf{w}} = -(d - y) f'(u) \mathbf{x} \quad (\text{I})$$

$$\nabla_b J = \frac{\partial J}{\partial u} \frac{\partial u}{\partial b} = -(d - y) f'(u) \quad (\text{J})$$

SGD for perceptron

Gradient learning equations:

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} J \\ b &\leftarrow b - \alpha \nabla_b J\end{aligned}$$

Substituting gradients from (I) and (J), SGD equations for a perceptron are given by

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} + \alpha(d - y)f'(u)\mathbf{x} \\ b &\leftarrow b + \alpha(d - y)f'(u)\end{aligned}$$

SGD algorithm for perceptron

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

Set learning parameter α

Initialize w and b

Repeat until convergence:

For every training pattern (x_p, d_p) :

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

$$y_p = f(u_p)$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(d_p - y_p)f'(u_p)\mathbf{x}_p$$

$$b \leftarrow b + \alpha(d_p - y_p)f'(u_p)$$

GD for perceptron

Given a training dataset $\{(x_p, d_p)\}_{p=1}^P$, cost function J is given by the sum of square errors (s.s.e) over all the patterns:

$$J = \frac{1}{2} \sum_{p=1}^P (d_p - y_p)^2 = \sum_{p=1}^P J_p \quad (\text{F})$$

where $J_p = \frac{1}{2} (d_p - y_p)^2$ is the square error for the p th pattern.

GD for perceptron

From (F):

$$\begin{aligned}\nabla_{\mathbf{w}} J &= \sum_{p=1}^P \nabla_{\mathbf{w}} J_p \\ &= - \sum_{p=1}^P (d_p - y_p) f'(u_p) \mathbf{x}_p && \text{From (J)} \\ &= - \left((d_1 - y_1) f'(u_1) \mathbf{x}_1 + (d_2 - y_2) f'(u_2) \mathbf{x}_2 + \dots + (d_P - y_P) f'(u_P) \mathbf{x}_P \right) \\ &= -(\mathbf{x}_1 \quad \mathbf{x}_2 \quad \dots \quad \mathbf{x}_P) \begin{pmatrix} (d_1 - y_1) f'(u_1) \\ (d_2 - y_2) f'(u_2) \\ \vdots \\ (d_P - y_P) f'(u_P) \end{pmatrix} \\ &= -\mathbf{X}^T (\mathbf{d} - \mathbf{y}) \cdot f'(\mathbf{u}) && (\text{K})\end{aligned}$$

where $\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_P^T \end{pmatrix}$, $\mathbf{d} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_P \end{pmatrix}$, $\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_P \end{pmatrix}$, and $f'(\mathbf{u}) = \begin{pmatrix} f'(u_1) \\ f'(u_2) \\ \vdots \\ f'(u_P) \end{pmatrix}$

GD for perceptron

Substituting X^T by $\mathbf{1}_P^T$ in (K), we get

$$\nabla_b J = -\mathbf{1}_P^T(\mathbf{d} - \mathbf{y}) \cdot f'(\mathbf{u}) \quad (\text{L})$$

where $\mathbf{1}_P = (1 \quad 1 \quad \dots \quad 1)^T$.

The gradient descent learning is given by

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} J \\ b &\leftarrow b - \alpha \nabla_b J\end{aligned}$$

Substituting (K) and (L), we get the learning equations:

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} + \alpha X^T(\mathbf{d} - \mathbf{y}) \cdot f'(\mathbf{u}) \\ b &\leftarrow b + \alpha \mathbf{1}_P^T(\mathbf{d} - \mathbf{y}) \cdot f'(\mathbf{u})\end{aligned}$$

Note that \cdot is the element-wise product.

GD for perceptron

Given a training dataset(\mathbf{X}, \mathbf{d})

Set learning parameter α

Initialize \mathbf{w} and b

Repeat until convergence:

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

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

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \mathbf{X}^T (\mathbf{d} - \mathbf{y}) \cdot f'(\mathbf{u})$$

$$b \leftarrow b + \alpha \mathbf{1}_P^T (\mathbf{d} - \mathbf{y}) \cdot f'(\mathbf{u})$$

Derivatives of Sigmoid

The activation function of the (continuous) perceptron is *sigmoid function* (i.e., unipolar sigmoidal function with $a = 1.0$ and $b = 1.0$) :

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

The derivative is given by

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

For *Tanh* function (bipolar sigmoid):

$$y = f(u) = \frac{e^{+u} - e^{-u}}{e^{+u} + e^{-u}}$$
$$f'(u) = \frac{(e^{+u}+e^{-u})(e^{+u}+e^{-u}) - (e^{+u}-e^{-u})(e^{+u}-e^{-u})}{(e^{+u}+e^{-u})^2} = \left(1 - \left(\frac{e^{+u}-e^{-u}}{e^{+u}+e^{-u}}\right)^2\right) = 1 - y^2$$

Example 3

Design a perceptron to learn the following mapping by using gradient descent (GD):

$x = (x_1, x_2)$	d
(0.77, 0.02)	2.91
(0.63, 0.75)	0.55
(0.50, 0.22)	1.28
(0.20, 0.76)	-0.74
(0.17, 0.09)	0.88
(0.69, 0.95)	0.30
(0.00, 0.51)	-0.28

Use learning factor $\alpha = 0.01$.

Example 3

$$X = \begin{pmatrix} 0.77 & 0.02 \\ 0.63 & 0.75 \\ 0.50 & 0.22 \\ 0.20 & 0.76 \\ 0.17 & 0.09 \\ 0.69 & 0.95 \\ 0.00 & 0.51 \end{pmatrix} \text{ and } \mathbf{d} = \begin{pmatrix} 2.91 \\ 0.55 \\ 1.28 \\ -0.74 \\ 0.88 \\ 0.30 \\ -0.28 \end{pmatrix}$$

Initially, $\mathbf{w} = \begin{pmatrix} 0.81 \\ 0.61 \end{pmatrix}$ and $b = 0.0$

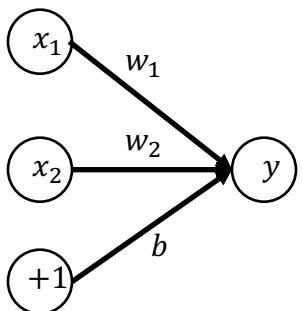
$\alpha = 0.01$

Output $y \in [-0.74, 2.91] \subset [-1.0, 3.0]$

So, the activation function should be

$$y = f(u) = \frac{4.0}{1 + e^{-u}} - 1.0$$

$$f'(u) = \frac{4e^{-u}}{(1+e^{-u})^2} = (y+1) \frac{e^{-u}}{(1+e^{-u})} = \frac{1}{4}(y+1)(3-y)$$



Example 3

$$\mathbf{u} = \mathbf{X}\mathbf{w} + b\mathbf{1}_P = \begin{pmatrix} 0.77 & 0.02 \\ 0.63 & 0.75 \\ 0.50 & 0.22 \\ 0.20 & 0.76 \\ 0.17 & 0.09 \\ 0.69 & 0.95 \\ 0.00 & 0.51 \end{pmatrix} \begin{pmatrix} 0.81 \\ 0.61 \end{pmatrix} + 0.0 \begin{pmatrix} 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \end{pmatrix} = \begin{pmatrix} 0.64 \\ 0.97 \\ 0.54 \\ 0.63 \\ 0.19 \\ 1.14 \\ 0.32 \end{pmatrix}$$

$$\mathbf{y} = f(\mathbf{u}) = \frac{4.0}{1 + e^{-\mathbf{u}}} - 1.0 = \begin{pmatrix} 1.61 \\ 1.90 \\ 1.53 \\ 1.61 \\ 1.19 \\ 2.03 \\ 1.31 \end{pmatrix}$$

$$m.s.e. = \frac{1}{7} \sum_{p=1}^7 (d_p - y_p)^2 = 0.30$$

Example 3

$$f'(\mathbf{u}) = \frac{1}{4}(\mathbf{y} + 1) \cdot (3 - \mathbf{y}) = \frac{1}{4} \left(\begin{pmatrix} 1.61 \\ 1.90 \\ 1.53 \\ 1.61 \\ 1.19 \\ 2.03 \\ 1.31 \end{pmatrix} + \begin{pmatrix} 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \end{pmatrix} \cdot \begin{pmatrix} 3.0 \\ 3.0 \\ 3.0 \\ 3.0 \\ 3.0 \\ 3.0 \\ 3.0 \end{pmatrix} - \begin{pmatrix} 1.61 \\ 1.90 \\ 1.53 \\ 1.61 \\ 1.19 \\ 2.03 \\ 1.31 \end{pmatrix} \right) = \begin{pmatrix} 0.90 \\ 0.80 \\ 0.93 \\ 0.91 \\ 0.99 \\ 0.73 \\ 0.98 \end{pmatrix}$$

Example 3

$$\mathbf{w} = \mathbf{w} + \alpha \mathbf{X}^T (\mathbf{d} - \mathbf{y}) \cdot f'(\mathbf{u})$$

$$= \begin{pmatrix} 0.81 \\ 0.61 \end{pmatrix} + 0.01 \begin{pmatrix} 0.77 & 0.63 & 0.50 & 0.20 & 0.17 & 0.69 & 0.00 \\ 0.02 & 0.75 & 0.22 & 0.76 & 0.09 & 0.95 & 0.51 \end{pmatrix} \left(\begin{pmatrix} 2.91 \\ 0.55 \\ 1.28 \\ -0.74 \\ 0.88 \\ 0.30 \\ -0.28 \end{pmatrix} - \begin{pmatrix} 1.61 \\ 1.90 \\ 1.53 \\ 1.61 \\ 1.19 \\ 2.03 \\ 1.31 \end{pmatrix} \right) \cdot \begin{pmatrix} 0.90 \\ 0.80 \\ 0.93 \\ 0.91 \\ 0.99 \\ 0.73 \\ 0.98 \end{pmatrix} = \begin{pmatrix} 0.80 \\ 0.57 \end{pmatrix}$$

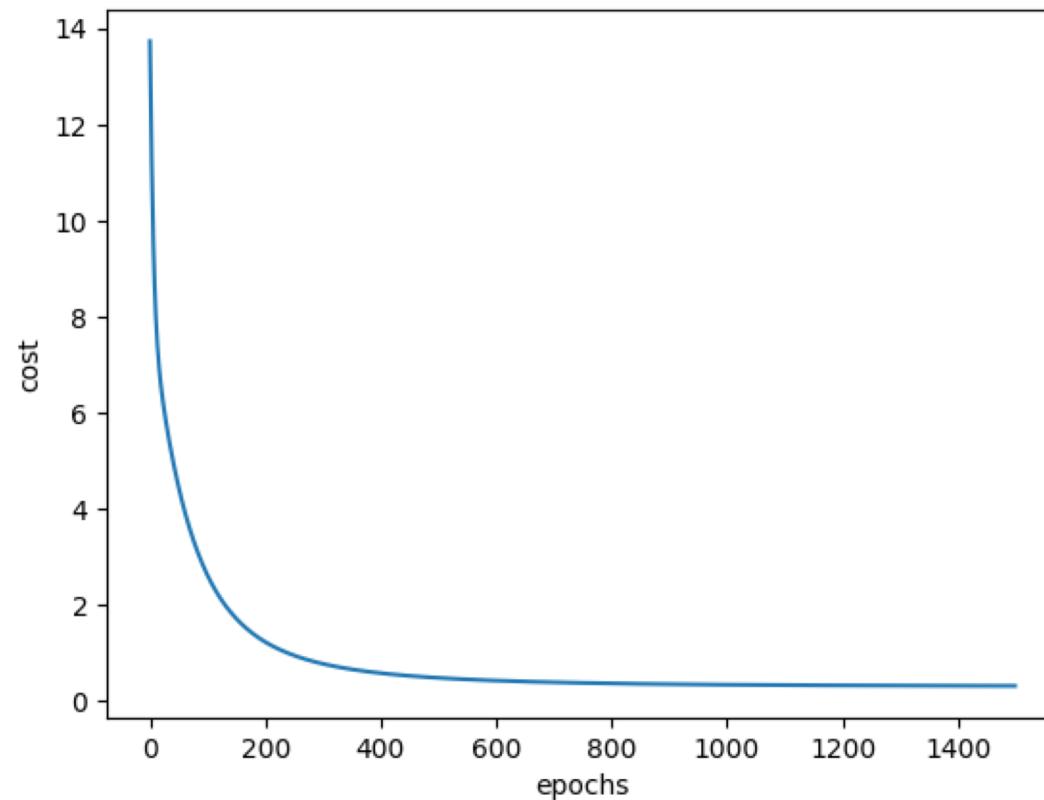
$$b = b + \alpha \mathbf{1}_P^T (\mathbf{d} - \mathbf{y}) \cdot f'(\mathbf{u})$$

$$= 0.0 + 0.01 \times (1.0 \quad 1.0 \quad 1.0 \quad 1.0 \quad 1.0 \quad 1.0) \left(\begin{pmatrix} 2.91 \\ 0.55 \\ 1.28 \\ -0.74 \\ 0.88 \\ 0.30 \\ -0.28 \end{pmatrix} - \begin{pmatrix} 1.61 \\ 1.90 \\ 1.53 \\ 1.61 \\ 1.19 \\ 2.03 \\ 1.31 \end{pmatrix} \right) \cdot \begin{pmatrix} 0.90 \\ 0.80 \\ 0.93 \\ 0.91 \\ 0.99 \\ 0.73 \\ 0.98 \end{pmatrix} = -0.05$$

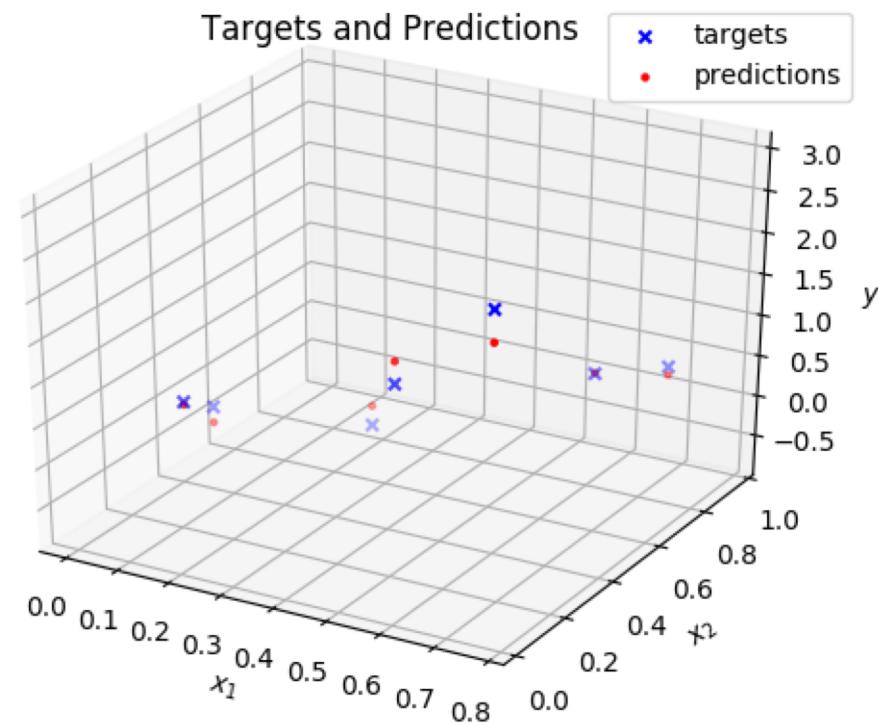
Example 3

<i>iter</i>	<i>u</i>	<i>y</i>	<i>f'(u)</i>	<i>mse</i>	<i>w</i>	<i>b</i>
1	$\begin{pmatrix} 0.64 \\ 0.97 \\ 0.54 \\ 0.63 \\ 0.19 \\ 1.14 \\ 0.32 \end{pmatrix}$	$\begin{pmatrix} 1.61 \\ 1.90 \\ 1.53 \\ 1.61 \\ 1.19 \\ 2.03 \\ 1.31 \end{pmatrix}$	$\begin{pmatrix} 0.90 \\ 0.80 \\ 0.93 \\ 0.91 \\ 0.99 \\ 0.73 \\ 0.98 \end{pmatrix}$	0.30	$\begin{pmatrix} 0.80 \\ 0.57 \end{pmatrix}$	-0.05
2	$\begin{pmatrix} 0.57 \\ 0.88 \\ 0.47 \\ 0.54 \\ 1.04 \\ 1.95 \\ 0.24 \end{pmatrix}$	$\begin{pmatrix} 1.56 \\ 1.83 \\ 1.46 \\ 1.52 \\ 1.13 \\ 1.95 \\ 1.24 \end{pmatrix}$	$\begin{pmatrix} 0.92 \\ 0.83 \\ 0.95 \\ 0.93 \\ 1.00 \\ 0.77 \\ 0.99 \end{pmatrix}$	0.28	$\begin{pmatrix} 0.79 \\ 0.52 \end{pmatrix}$	-0.11
1500	$\begin{pmatrix} 2.05 \\ -0.45 \\ 0.56 \\ -1.94 \\ -0.16 \\ -0.85 \\ -1.89 \end{pmatrix}$	$\begin{pmatrix} 2.54 \\ 0.56 \\ 1.55 \\ -0.50 \\ 0.84 \\ 0.20 \\ -0.48 \end{pmatrix}$	$\begin{pmatrix} 0.40 \\ 0.95 \\ 0.92 \\ 0.44 \\ 0.99 \\ 0.84 \\ 0.45 \end{pmatrix}$	0.01	$\begin{pmatrix} 3.35 \\ -2.80 \end{pmatrix}$	-0.47

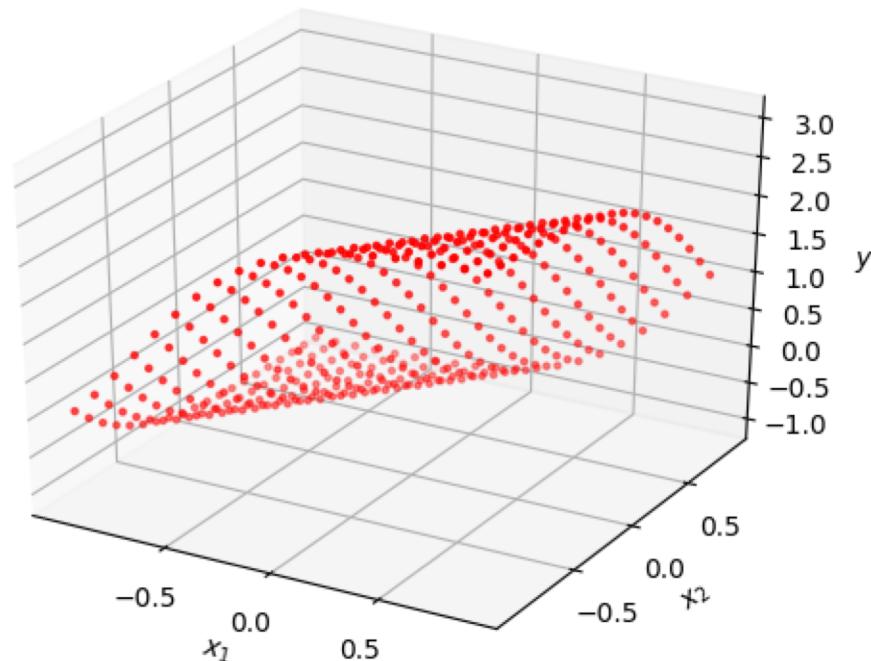
Example 3



Example 3



Example 3

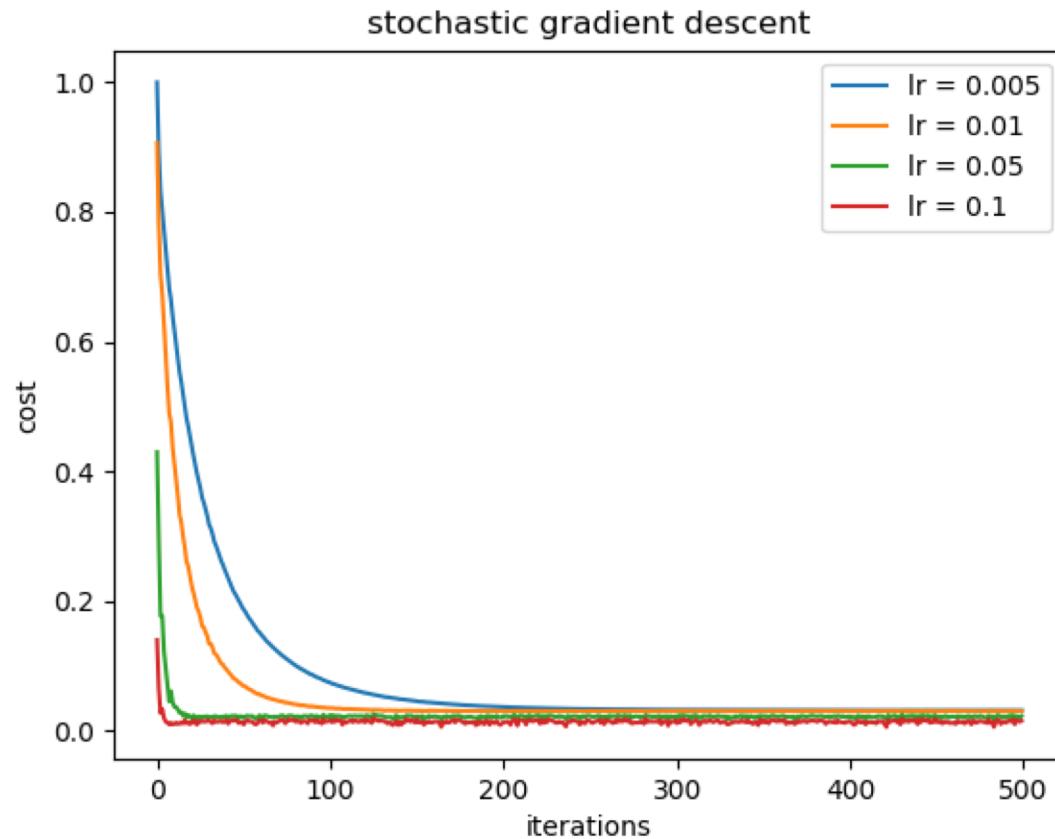


Non-linear function learnt by the perceptron

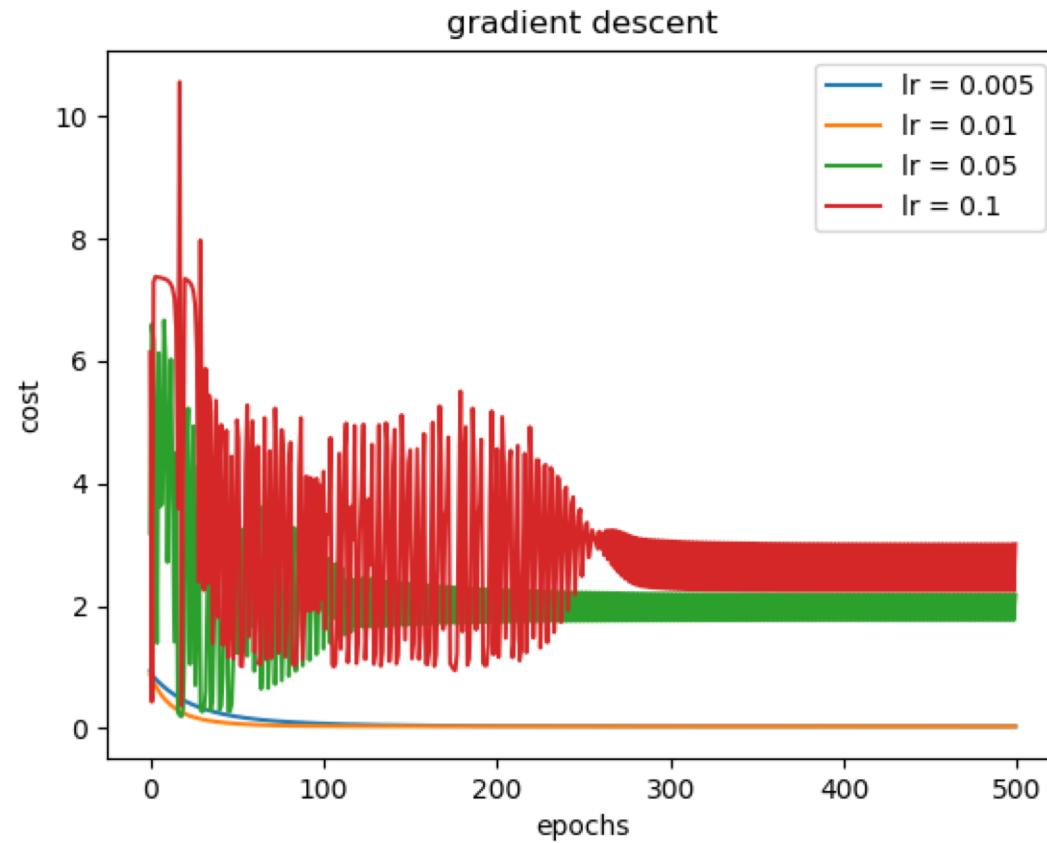
SGD for perceptron

GD	SGD
(X, d)	(x_p, d_p)
$J = \frac{1}{2} \sum_{p=1}^P (d_p - y_p)^2$	$J = \frac{1}{2} (d_p - y_p)^2$
$\mathbf{u} = X\mathbf{w} + b\mathbf{1}_P$	$u_p = \mathbf{x}_p^T \mathbf{w} + b$
$\mathbf{y} = f(\mathbf{u})$	$y_p = f(u_p)$
$\mathbf{w} = \mathbf{w} + \alpha X^T (\mathbf{d} - \mathbf{y}) \cdot f'(\mathbf{u})$	$\mathbf{w} = \mathbf{w} + \alpha (d_p - y_p) f'(u_p) \mathbf{x}_p$
$b = b + \alpha \mathbf{1}_P^T (\mathbf{d} - \mathbf{y}) \cdot f'(\mathbf{u})$	$b = b + \alpha (d_p - y_p) f'(u_p)$

Example 4: Learning rates with SGD



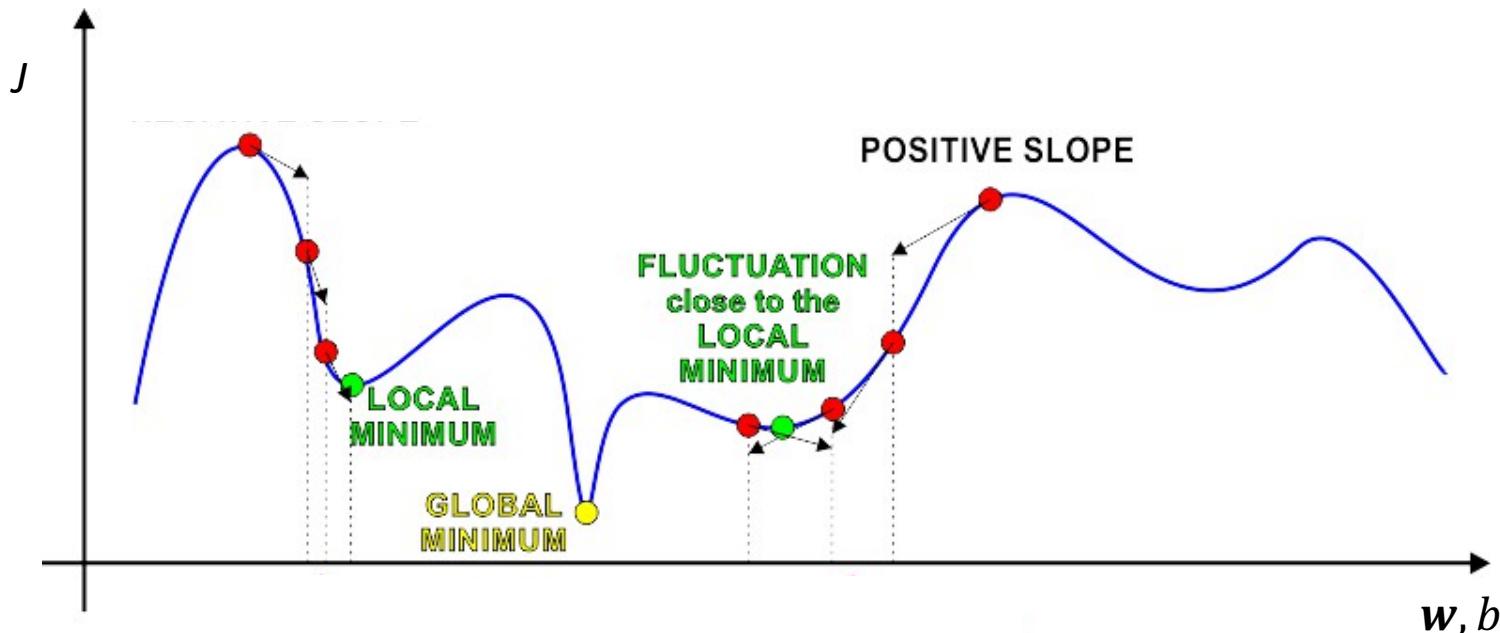
Example 4: Learning rates with GD



Learning rates

- At higher learning rates, convergence is faster but may not be stable.
- Generally,
 - SGD convergence to a better solution (lower error); SGD explores the randomness of data; SGD takes a longer time to converge
 - GD can use a higher learning rate is possible with GD; all the data is used in one batch. The time for one add/multiply computation is less when patterns are trained in a batch.
- In practice, *mini-batch SGD* is used.
- The optimal learning rate is the largest rate at which learning does not diverge. The optimum learning speed depends on
 - the learning rate
 - the batch size

Local minima problem in gradient descent learning



Algorithm may stuck in a local minimum of error function depending on the initial weights. Gradient descent gives a suboptimal solution and does not guarantee the optimal solution.

Summary of Chapter 2

- Regression with a linear neuron and a perceptron
- For a batch, The synaptic input $\mathbf{u} = \mathbf{X}\mathbf{w} + b\mathbf{1}_P$
- **Linear neuron** performs linear regression: $\mathbf{y} = \mathbf{u}$

GD learning equations:

$$\begin{aligned}\mathbf{w} &= \mathbf{w} + \alpha \mathbf{X}^T (\mathbf{d} - \mathbf{y}) \\ b &= b + \alpha \mathbf{1}_P^T (\mathbf{d} - \mathbf{y})\end{aligned}$$

- **Perceptron** performs non-linear regression:

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

GD learning equations:

$$\begin{aligned}\mathbf{w} &= \mathbf{w} + \alpha \mathbf{X}^T (\mathbf{d} - \mathbf{y}) \cdot f'(\mathbf{u}) \\ b &= b + \alpha \mathbf{1}_P^T (\mathbf{d} - \mathbf{y}) \cdot f'(\mathbf{u})\end{aligned}$$