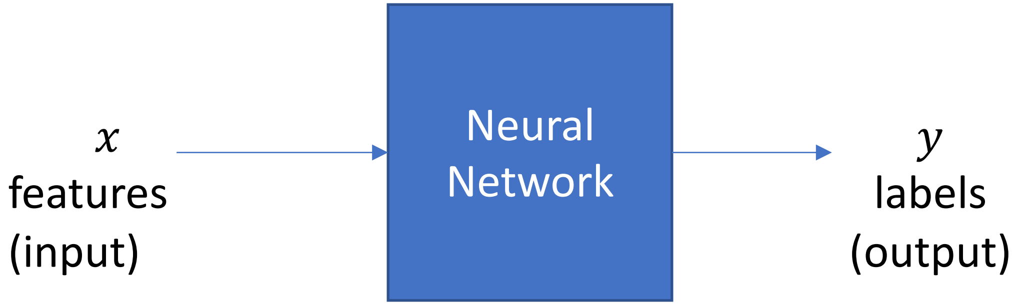
Chapter 2

Regression

Neural networks and deep learning

Regression and classification



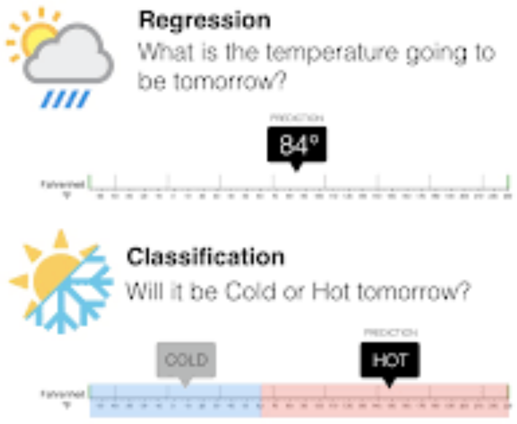
Primarily, neural network are used to *predict* output **labels** from input **features**.

Prediction tasks can be classified into two categories:

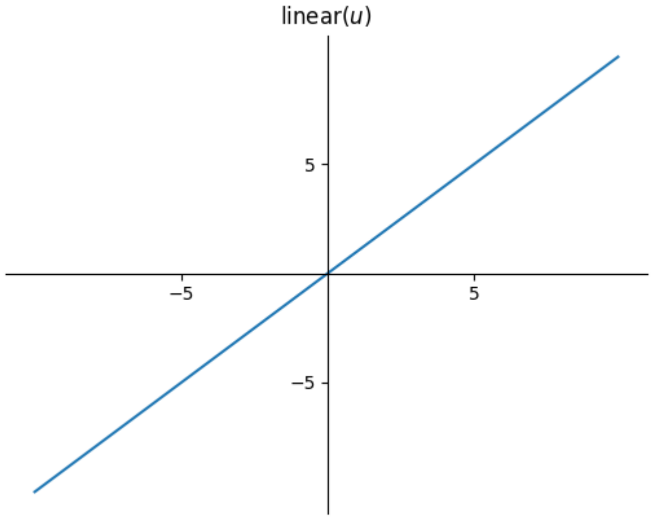
Regression: the labels are continuous (age, income, height, etc.)

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

Supervised learning finds network weights and biases that are optimal for **prediction** of labels from features.



Linear neuron



Synaptic input u to a neuron is given by

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

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

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

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

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

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

Linear neuron performs linear regression

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

$$y = w_1 x_1 + w_2 x_2 \cdots + w_n x_n + b$$

where $x_1, x_2, \cdots x_n$ are the inputs.

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

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

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

Stochastic gradient descent (SGD) for linear neuron

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

Given a training pattern $(\boldsymbol{x}, d)$, $\frac{1}{2}$ square error cost J is defined as

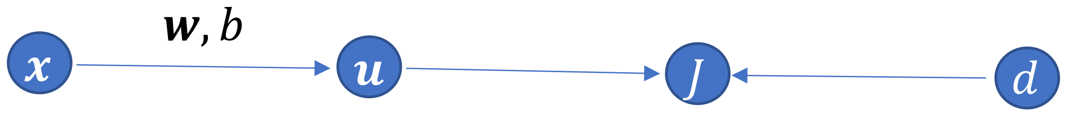
$$J = \frac{1}{2}(d - y)^2$$

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

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

The ½ 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



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

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

$$\nabla_{\boldsymbol{w}} J = \frac{\partial J}{\partial \boldsymbol{w}} = \frac{\partial J}{\partial u}\frac{\partial u}{\partial \boldsymbol{w}} \qquad \text{(B)}$$

SGD for linear neuron

$$u = \boldsymbol{w}^T\boldsymbol{x} + b = w_1x_1 + w_2x_2 \cdots + w_nx_n + b$$

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

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

$$\nabla_{\boldsymbol{w}} J = -(d - y)\boldsymbol{x} \qquad \text{(D)}$$

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

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

SGD for linear neuron

Gradient learning equations:

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

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

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha(d - y)\boldsymbol{x}$$
$$b \leftarrow b + \alpha(d - y)$$

SGD algorithm for linear neuron

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

Set learning parameter α

Initialize $\boldsymbol{w}$ and b

Repeat until convergence:

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

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

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

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

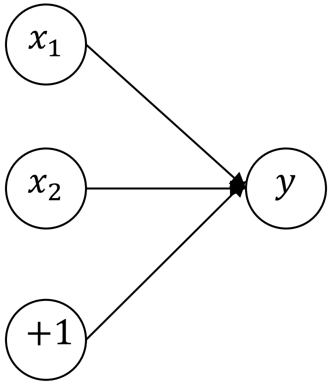
Example 1

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

$\boldsymbol{x = (x_1, x_2)}$	$\boldsymbol{d}$
$(0.54, -0.96)$	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

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

$\alpha = 0.01$

Example 1: epoch 1

Epoch 1 begins

Shuffle the patterns ….

First pattern $p = 1$ is applied:

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

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

s.e. = $\left(d_p - y_p\right)^2 = 2.292$

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

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

Example 1: epoch 1

Second pattern $p = 2$ is applied:

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

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

s.e. = $\left(d_p - y_p\right)^2 = 0.01$

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

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

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

the second epoch starts …..
Shuffle the patterns
Apply patterns $p = 1,2, \cdots 6$

Training epochs continue until convergence.

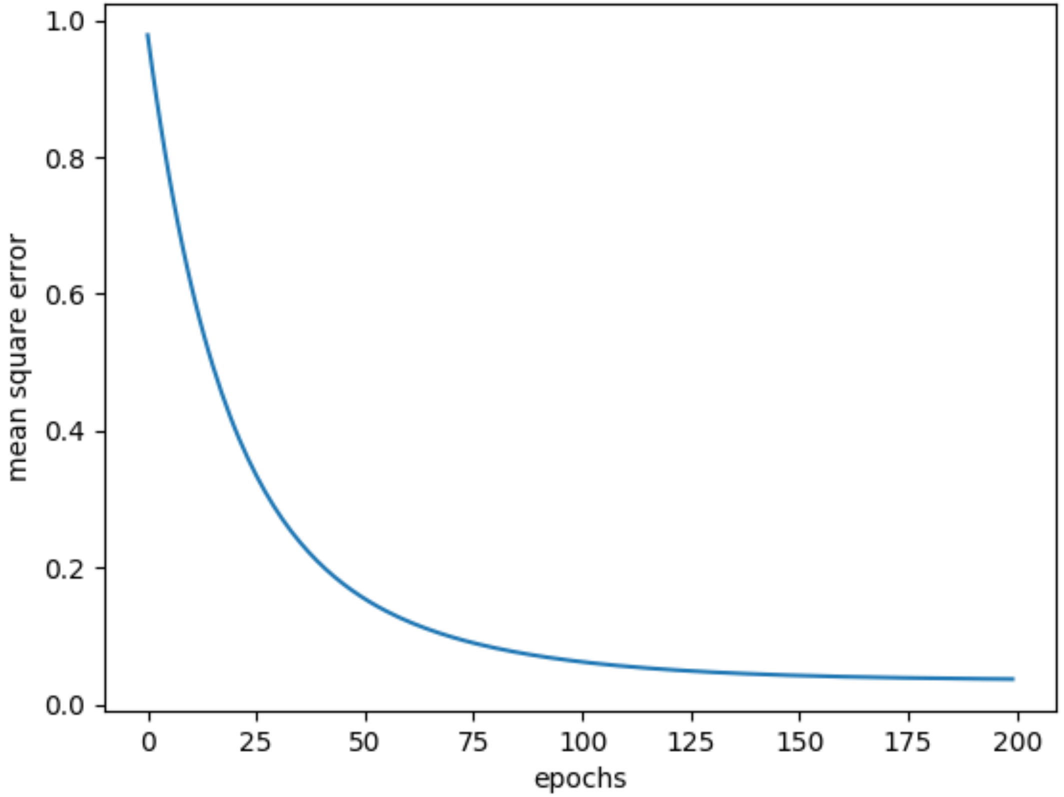
Example 1: epoch 1

$\boldsymbol{x}_p$	y_p	s.e.	$\boldsymbol{w}$	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

$\boldsymbol{x}_p$	y_p	s.e.	$\boldsymbol{w}$	b
$\boldsymbol{x}_1 = \begin{pmatrix} 0.54 \\ -0.96 \end{pmatrix}$	1.49	0.03	$\begin{pmatrix} 2.00 \\ -0.44 \end{pmatrix}$	-0.01
$\boldsymbol{x}_2 = \begin{pmatrix} 0.00 \\ -0.55 \end{pmatrix}$	0.22	0.12	$\begin{pmatrix} 2.00 \\ -0.44 \end{pmatrix}$	-0.01
$\boldsymbol{x}_3 = \begin{pmatrix} -0.66 \\ -0.82 \end{pmatrix}$	-0.98	0.01	$\begin{pmatrix} 2.00 \\ -0.44 \end{pmatrix}$	-0.01
$\boldsymbol{x}_4 = \begin{pmatrix} 0.37 \\ 0.91 \end{pmatrix}$	0.33	0.00	$\begin{pmatrix} 2.00 \\ -0.44 \end{pmatrix}$	-0.01
$\boldsymbol{x}_5 = \begin{pmatrix} 0.27 \\ 0.50 \end{pmatrix}$	0.30	0.02	$\begin{pmatrix} 2.00 \\ -0.44 \end{pmatrix}$	-0.01
$\boldsymbol{x}_6 = \begin{pmatrix} -0.60 \\ 0.52 \end{pmatrix}$	-1.45	0.04	$\begin{pmatrix} 2.00 \\ -0.44 \end{pmatrix}$	-0.01

Example 1



m.s.e. = $\frac{1}{6}\sum_{p=1}^{6}(d-y)^2$

Example 1

At convergence:

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

Mean square error = 0.037

The regression equation:

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

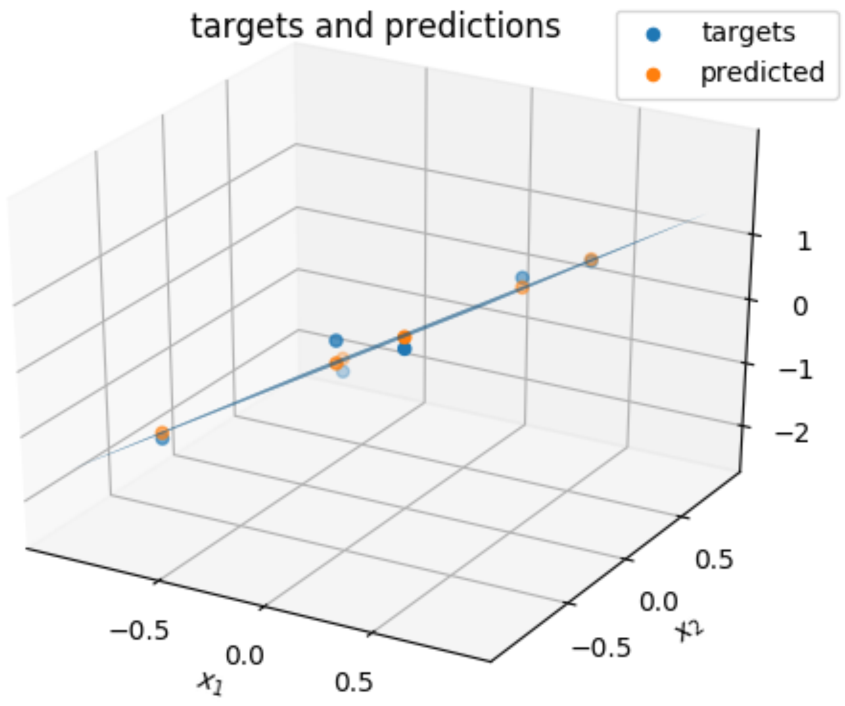
The mapping learnt by the linear neuron:

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

Example 1

inputs $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$	*predictions* $y = 2.00x_1 - 0.44x_2 - 0.01$	***targets*** $\boldsymbol{d}$
$(0.54, -0.96)$	1.49	1.33
$(0.27, 0.50)$	0.30	0.45
$(0.00, -0.55)$	0.22	0.56
$(-0.60, 0.52)$	-1.45	-1.66
$(-0.66, -0.82)$	-0.98	-1.07
$(0.37, 0.91)$	0.33	0.30

Example 1



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

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

```python
# class for a linear neuron
class Linear(object):
    def __init__(self):
        self.w = tf.Variable(np.random.rand(2), dtype=tf.float64)
        self.b = tf.Variable(0., dtype=tf.float64)

    def __call__(self, x):
        return tf.tensordot(x ,self.w, axes=1) + self.b


# squared error as the loss
def loss(predicted_y, target_y):
    return tf.square(predicted_y - target_y)


# function executing a training step
def train_step(model, x, d, learning_rate):
    y = model(x)
    grad_w = -(d - y)*x
    grad_b = -(d - y)
   model.w.assign(model.w - learning_rate * grad_w)
   model.b.assign(model.b - learning_rate * grad_b)
```

```python
model = Linear()

# keep an index for training
idx = np.arange(len(X))

# training epochs iterate
err = []
for epoch in range(no_epochs):
    np.random.shuffle(idx)
    X, Y = X[idx], Y[idx]

    for p in np.arange(len(X)):
        train_step(model, X[p], Y[p], learning_rate=lr)
```

Gradient descent (GD) for linear neuron

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

$$J = \sum_{p=1}^{P} J_p \qquad \text{(F)}$$

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

GD for linear neuron

From (F):

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

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

GD for linear neuron

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

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

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

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

GD for linear neuron

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

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

We get GD learning equations for the linear neuron as

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \boldsymbol{X}^{T}(\boldsymbol{d} - \boldsymbol{y})$$
$$b \leftarrow b + \alpha \boldsymbol{1}_{P}{}^{T}(\boldsymbol{d} - \boldsymbol{y})$$

And α is the learning factor.

Where

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} + b\,\boldsymbol{1}_{P}$$

GD for linear neuron

Given a training dataset$(\boldsymbol{X}, \boldsymbol{d})$
Set learning parameter α
Initialize $\boldsymbol{w}$ and b
Repeat until convergence:

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

GD and SGD for a linear neuron

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

Perceptron

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

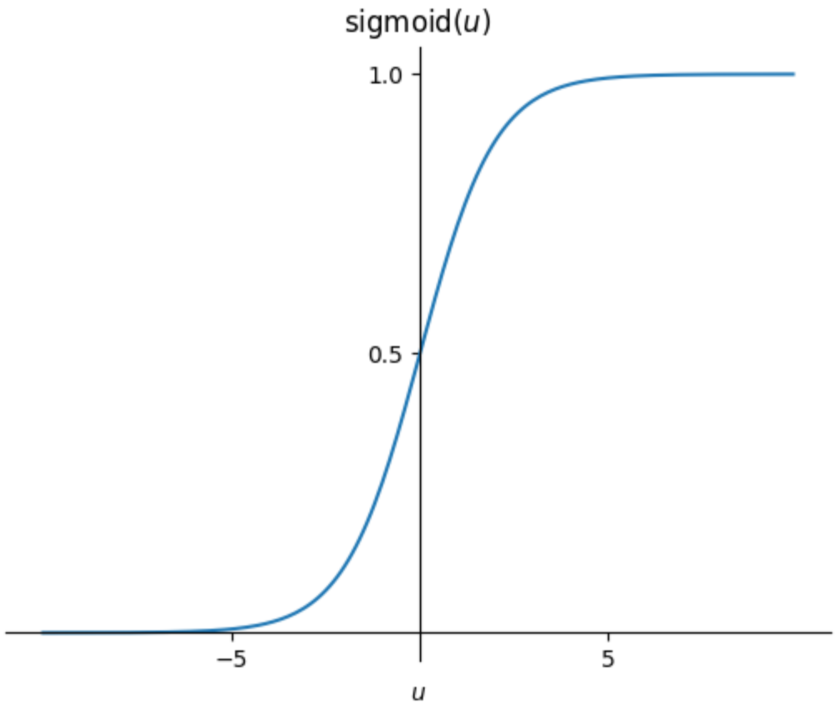
$$y = f(u)$$

where

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

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

The square error is used as cost function for learning.



Perceptron

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

where $\boldsymbol{x}_p = (x_{p1}, x_{p2}, \cdots x_{pn})^T \in \boldsymbol{R}^n$ and $d_p \in \boldsymbol{R}$.

The continuous perceptron finds a functional mapping:

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

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

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

SGD for perceptron

$$\boldsymbol{x} \xrightarrow{\boldsymbol{w}, b} \boldsymbol{u} \longrightarrow y \longrightarrow J \longleftarrow d$$

Cost function J is given by

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

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

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), $\frac{\partial u}{\partial \boldsymbol{w}} = \boldsymbol{x}$ and $\frac{\partial u}{\partial b} = 1$.

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

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

SGD for perceptron

Gradient learning equations:

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

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

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

SGD algorithm for perceptron

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

Set learning parameter α

Initialize $\boldsymbol{w}$ and b

Repeat until convergence:

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

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

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

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

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

GD for perceptron

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

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

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

GD for perceptron

From (F):

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

$$= -\sum_{p=1}^{P} (d_p - y_p) f'(u_p) \boldsymbol{x}_p \qquad \text{From (J)}$$

$$= -\left((d_1 - y_1) f'(u_1) \boldsymbol{x}_1 + (d_2 - y_2) f'(u_2) \boldsymbol{x}_2 + \cdots + (d_P - y_P) f'(u_P) \boldsymbol{x}_p\right)$$

$$= -(\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \cdots \quad \boldsymbol{x}_p) \begin{pmatrix} (d_1 - y_1) f'(u_1) \\ (d_2 - y_2) f'(u_2) \\ \vdots \\ (d_P - y_P) f'(u_P) \end{pmatrix}$$

$$= -\boldsymbol{X}^T (\boldsymbol{d} - \boldsymbol{y}) \cdot f'(\boldsymbol{u}) \qquad \text{(K)}$$

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

GD for perceptron

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

$$\nabla_{\boldsymbol{b}} J = -\mathbf{1}_P{}^T(\boldsymbol{d} - \boldsymbol{y}) \cdot f'(\boldsymbol{u}) \tag{L}$$

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

The gradient descent learning is given by

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

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

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

Note that $\cdot$ is the element-wise product.

GD for perceptron

Given a training dataset$(\boldsymbol{X}, \boldsymbol{d})$
Set learning parameter α
Initialize $\boldsymbol{w}$ and b
Repeat until convergence:

$$\boldsymbol{u} = \boldsymbol{X}\boldsymbol{w} + b\mathbf{1}_P$$
$$\boldsymbol{y} = f(\boldsymbol{u}) = \frac{1}{1+e^{-\boldsymbol{u}}}$$
$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\boldsymbol{X}^T(\boldsymbol{d} - \boldsymbol{y}) \cdot f'(\boldsymbol{u})$$
$$b \leftarrow b + \alpha\mathbf{1}_P{}^T(\boldsymbol{d} - \boldsymbol{y}) \cdot f'(\boldsymbol{u})$$

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 2

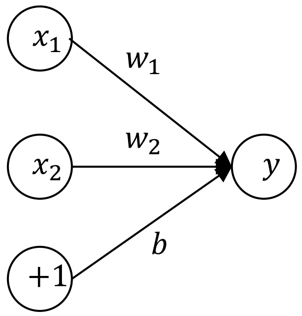
Design a perceptron to learn the following mapping by using gradient descent (GD):

$\boldsymbol{x} = (\boldsymbol{x_1}, \boldsymbol{x_2})$	$\boldsymbol{d}$
$(0.77, 0.02)$	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 $\alpha = 0.01$.

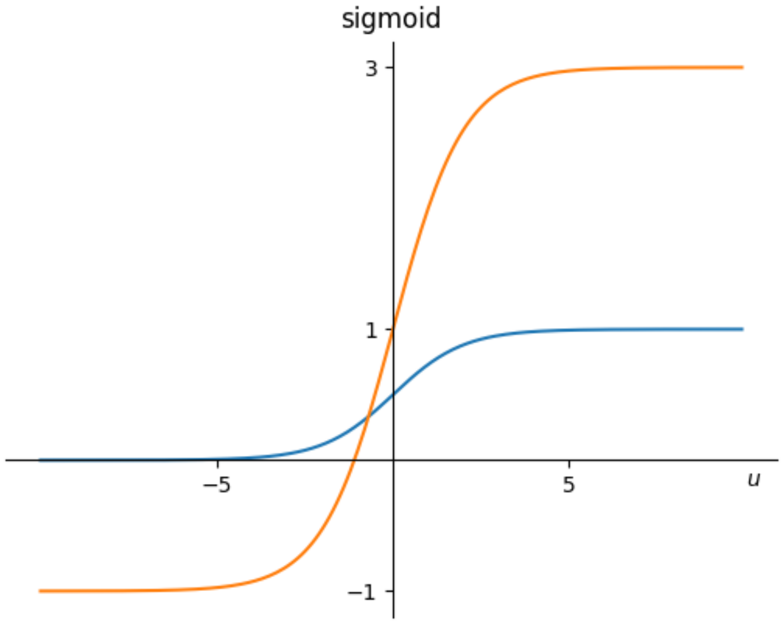
Example 2

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



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

$\alpha = 0.01$



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

Note that the sigmoidal should have an amplitude = 4 and shifted downwards by 1.0.

So, the activation function should be

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

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

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

Example 2

Epoch 1 begins …

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

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

$$m.s.e. = \frac{1}{7} \sum_{p=1}^{7} \left(d_p - y_p\right)^2 = 2.11$$

Example 2

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

Example 2

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

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

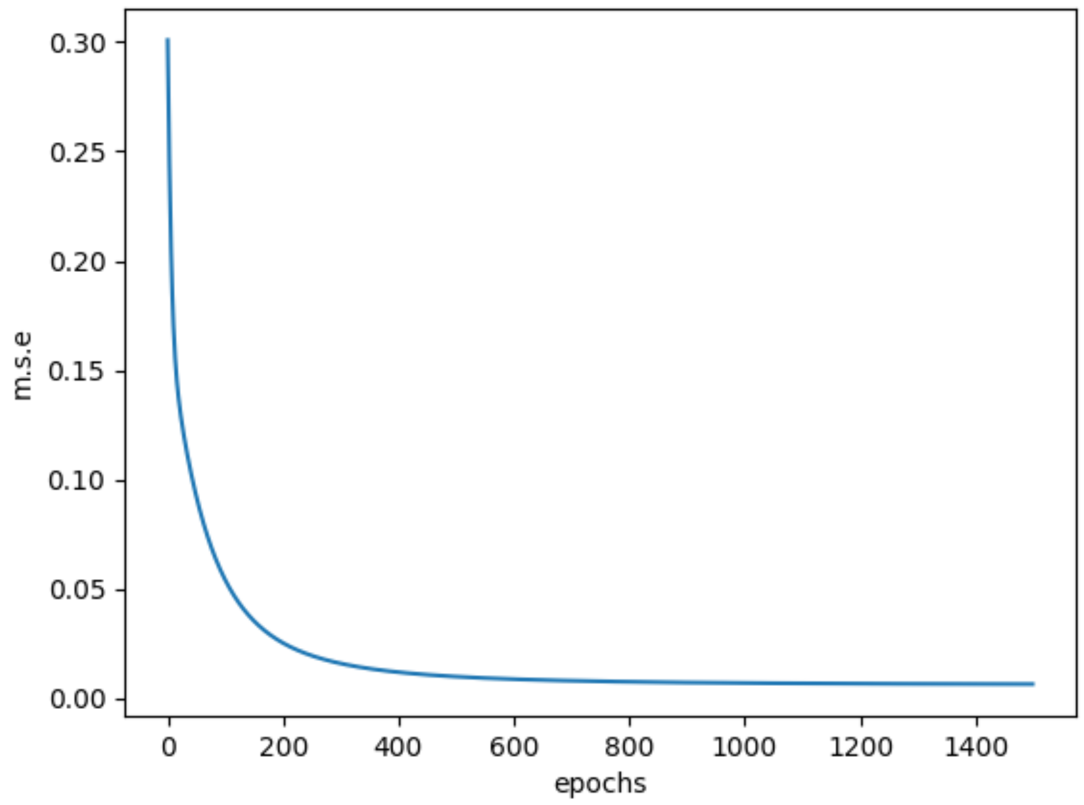
$$b = b + \alpha \boldsymbol{1}_P{}^T(\boldsymbol{d} - \boldsymbol{y}) \cdot f'(\boldsymbol{u})$$

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

Example 2

$iter$	u	y	$f'(u)$	mse	w	b
1	$\begin{pmatrix}0.64\\0.97\\0.54\\0.63\\0.19\\1.14\\0.32\end{pmatrix}$	$\begin{pmatrix}1.61\\1.90\\1.53\\1.61\\1.19\\2.03\\1.31\end{pmatrix}$	$\begin{pmatrix}0.90\\0.80\\0.93\\0.91\\0.99\\0.73\\0.98\end{pmatrix}$	2.11	$\begin{pmatrix}0.80\\0.57\end{pmatrix}$	-0.05
2	$\begin{pmatrix}0.57\\0.88\\0.47\\0.54\\1.04\\1.95\\0.24\end{pmatrix}$	$\begin{pmatrix}1.56\\1.83\\1.46\\1.52\\1.13\\1.95\\1.24\end{pmatrix}$	$\begin{pmatrix}0.92\\0.83\\0.95\\0.93\\1.00\\0.77\\0.99\end{pmatrix}$	1.96	$\begin{pmatrix}0.79\\0.52\end{pmatrix}$	-0.11
1500	$\begin{pmatrix}2.05\\-0.45\\0.56\\-1.94\\-0.16\\-0.85\\-1.89\end{pmatrix}$	$\begin{pmatrix}2.54\\0.56\\1.55\\-0.50\\0.84\\0.20\\-0.48\end{pmatrix}$	$\begin{pmatrix}0.40\\0.95\\0.92\\0.44\\0.99\\0.84\\0.45\end{pmatrix}$	0.046	$\begin{pmatrix}3.35\\-2.80\end{pmatrix}$	-0.47

Example 2



0.30
0.25
0.20
0.15
0.10
0.05
0.00
m.s.e
0
200
400
600
800
1000
1200
1400
epochs

Example 2

At convergence:

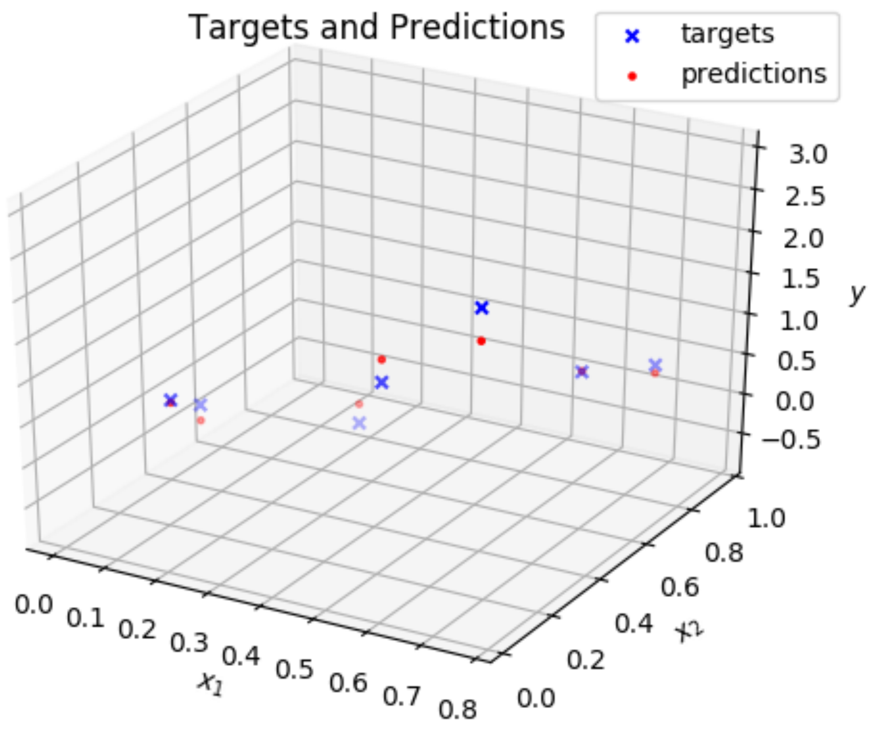
$$\boldsymbol{w} = \begin{pmatrix} 3.35 \\ -2.80 \end{pmatrix}$$
$$b = -0.47$$

Mean square error = 0.01

$$u = \boldsymbol{x}^T \boldsymbol{w} + b = (x_1 \quad x_2) \begin{pmatrix} 3.35 \\ -2.80 \end{pmatrix} - 0.47 = 3.35x_1 - 2.8x_2 - 0.47$$

$$y = \frac{4.0}{1 + e^{-u}} - 1.0$$
$$y = \frac{4.0}{1 + e^{-3.35x_1 + 2.8x_2 + 0.47}} - 1.0$$

Example 2

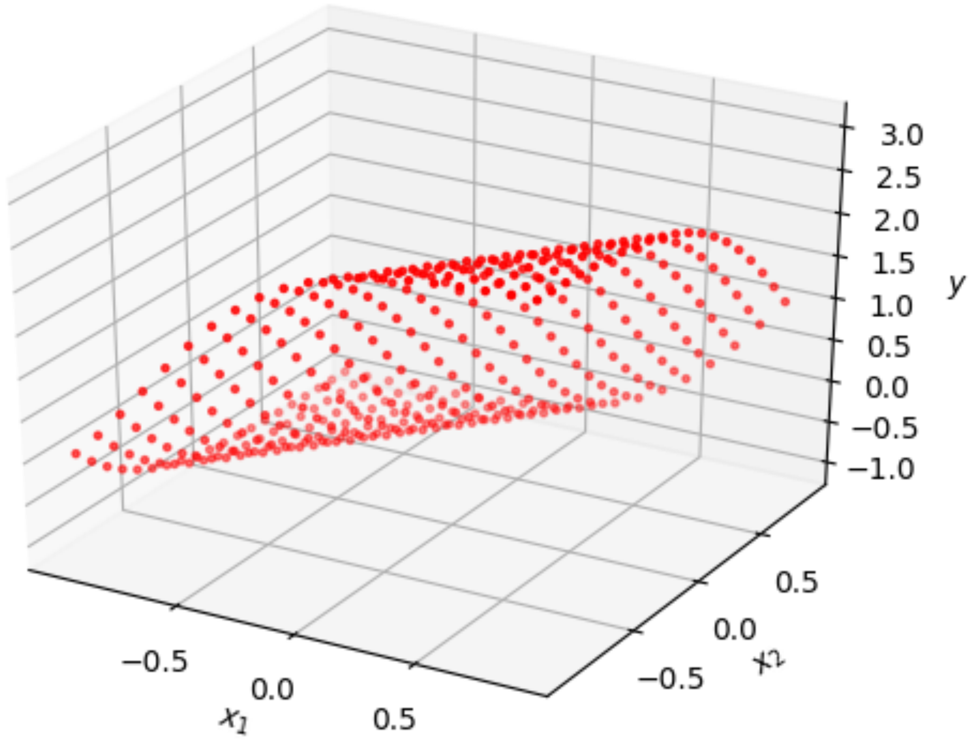


Targets and Predictions
targets
predictions
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
y
1.0
0.8
0.6
0.4
0.2
0.0
x_2
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
x_1

Example 2

Non-linear function leant by the perceptron

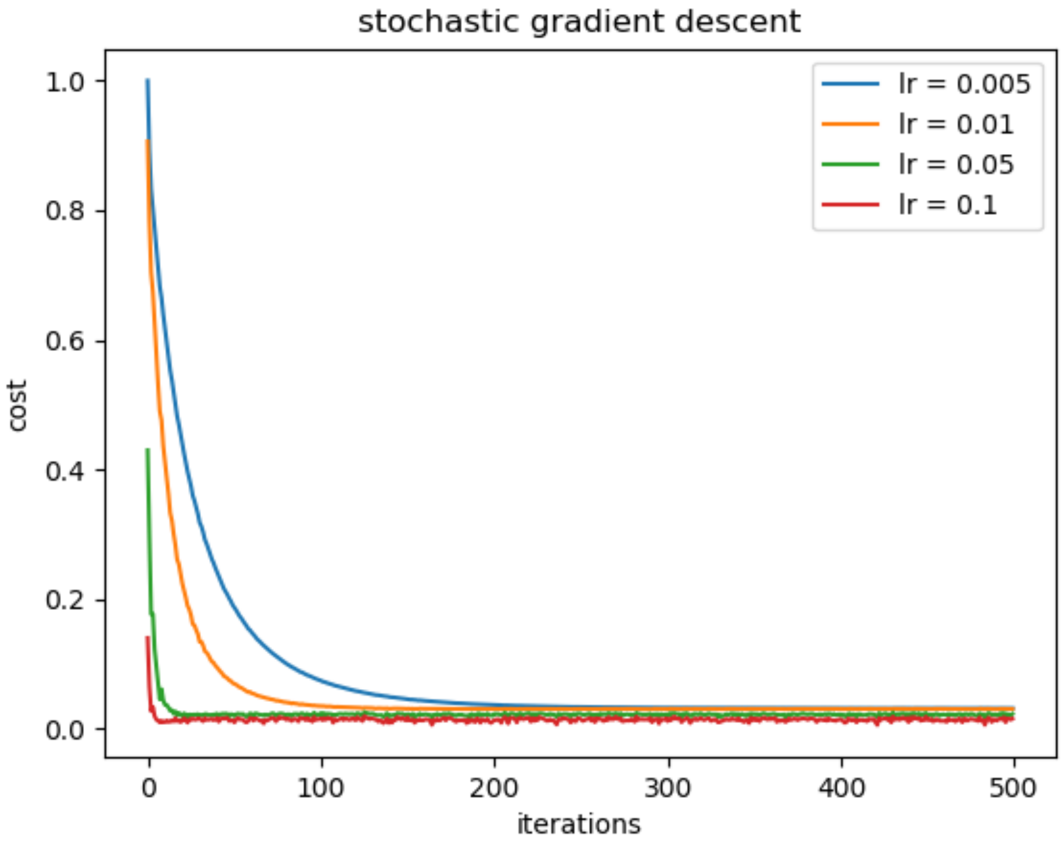
$$y = \frac{4.0}{1 + e^{-3.35x_1 + 2.8x_2 + 0.47}} - 1.0$$



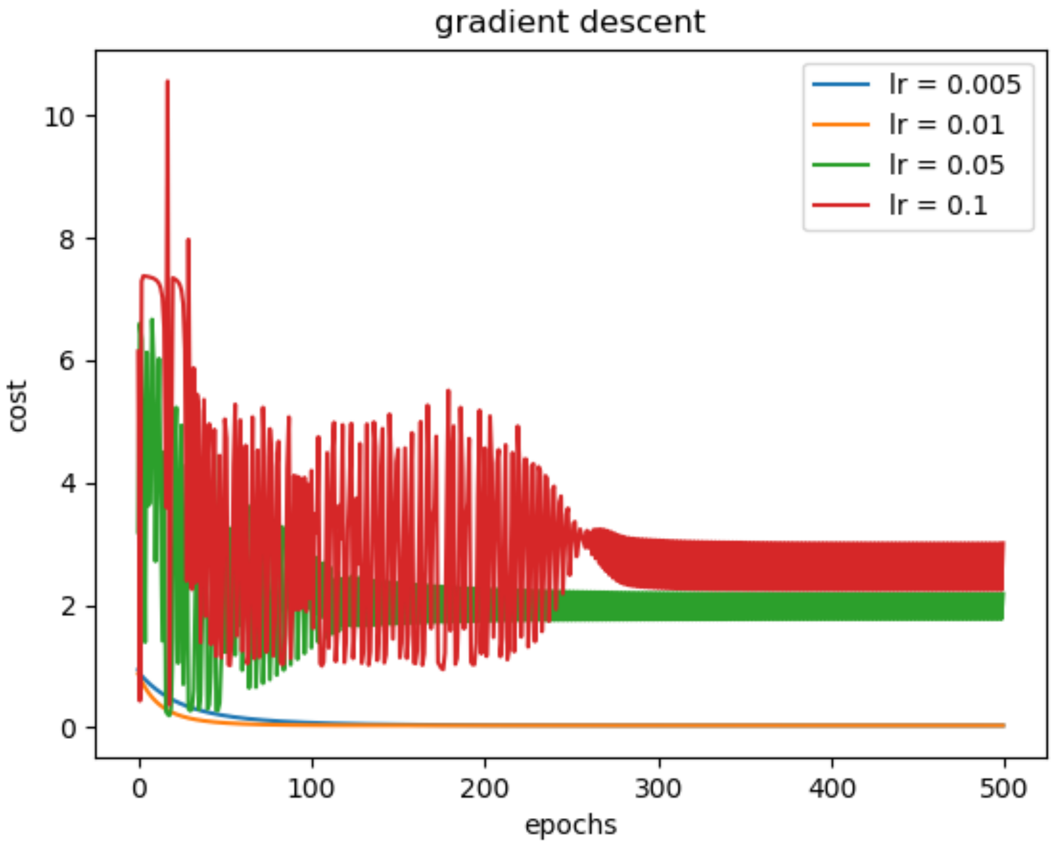
Gradient descent for perceptron

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

Example 3: Learning rates with SGD



Example 3: Learning rates with GD



Learning rates

- At higher learning rates, convergence is faster but may not be stable.

- The optimal learning rate is the largest rate at which learning does not diverge.

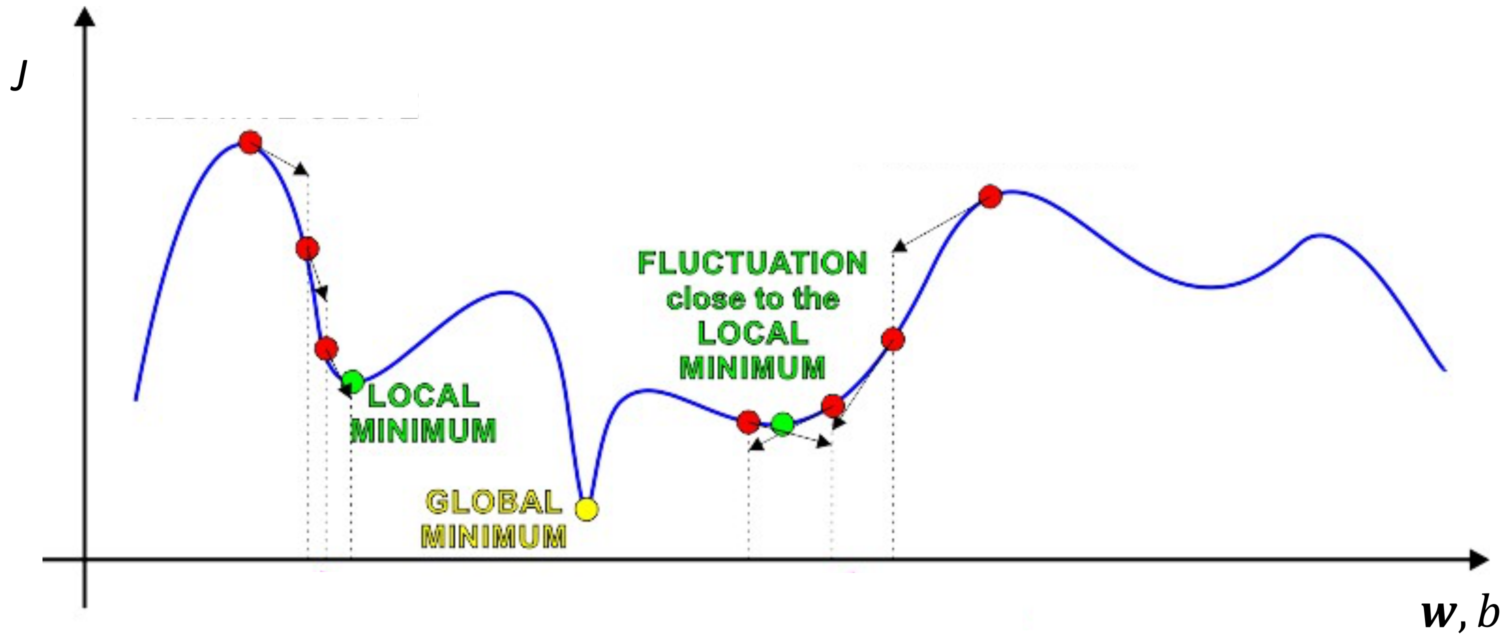
- Generally, SGD convergence to a better solution (lower error) as it capitalizes on randomness of data. SGD takes a longer time to converge

- Usually, GD can use a higher learning rate compared to SGD; The time for one add/multiply computation is less when patterns are trained in a batch.

- In practice, *mini-batch SGD* is used.

- Time to train a network is dependent upon
  - 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 $\boldsymbol{u} = \boldsymbol{X}\boldsymbol{w} + b\mathbf{1}_P$
- **Linear neuron** performs linear regression: $\boldsymbol{y} = \boldsymbol{u}$

  GD learning equations:

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

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

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

  GD learning equations:

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