1

Artificial Intelligence

Dr. Tran Quang Huy

HUYTRAN

ARTIFICIAL INTELLIGENCE

OUTLINE

2

Chapter 1: Overview of AI

Chapter 2: Artificial Neural Networks

Chapter 3: Searching, Knowledge, Reasoning, and Planning

Chapter 4: Machine learning

W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
L	L	L	I-T	L	L	L	L	P	P

L: Lesson; I-T: In-class Test; P: Project

HUYTRAN

ARTIFICIAL INTELLIGENCE

Objectives

3

1. Understand the basics of Neural Networks

2. Being able to move on the more advanced Convolutional Neural Networks

HUYTRAN

ARTIFICIAL INTELLIGENCE

Main contents

4

1. Artificial Neural Networks (ANN) and their relation to biology
2. The seminal Perceptron algorithm
3. Back propagation
4. How to train Neural Networks using Keras library

HUYTRAN

ARTIFICIAL INTELLIGENCE

What are Neural Networks?

5

Question:

- How does your family dog recognize you, the owner, versus a complete and total stranger?
- How does a small child learn to recognize the difference between a school bus and a transit bus?
- How do our own brains subconsciously perform complex pattern recognition tasks each and every day without us even noticing?

HUYTRAN

ARTIFICIAL INTELLIGENCE

What are Neural Networks?

6

Answer: Each of us contains a real-life biological neural networks that is connected to our nervous systems – this network is made up of a large number of interconnected neurons (nerve cells).

The word "neural" is the adjective form of "neuron", and "network" denotes a graph-like structure; therefore, an "Artificial Neural Network" is a computation system that attempts to mimic (or at least, is inspired by) the neural connections in our nervous system. Artificial neural networks are also referred to as "neural networks" or "artificial neural systems".

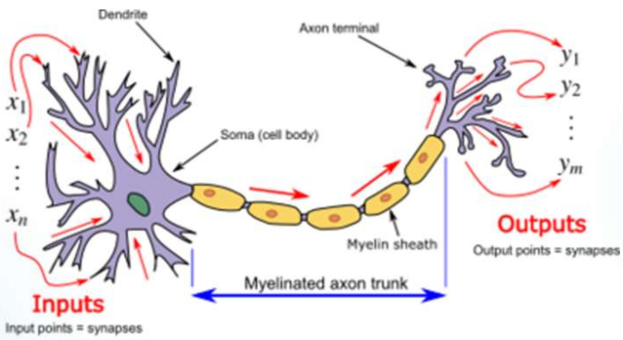
It is common to abbreviate Artificial Neural Network and refer to them as "ANN" or simply "NN"

HUYTRAN

ARTIFICIAL INTELLIGENCE

ANN

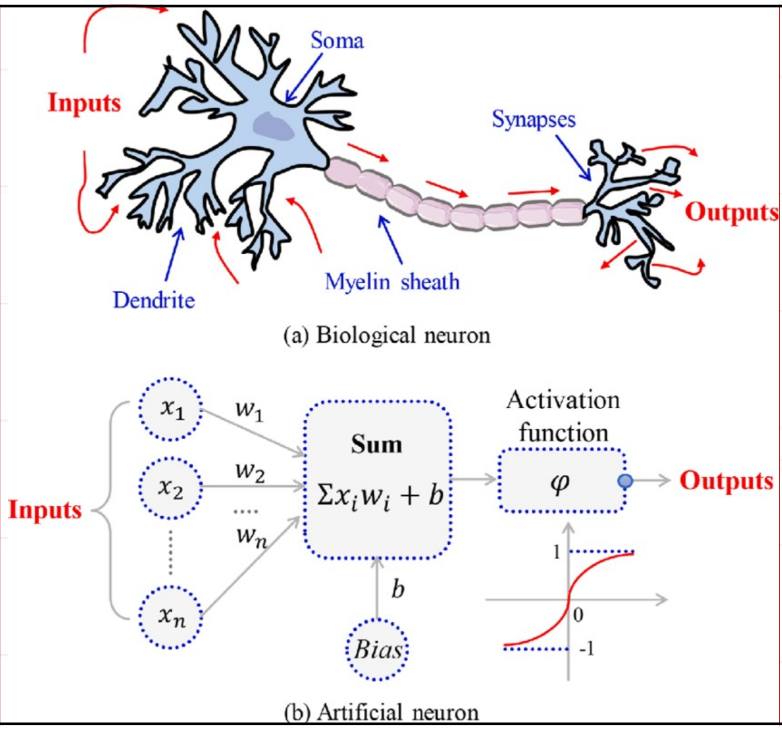
7



HUYTRAN

ARTIFICIAL INTELLIGENCE

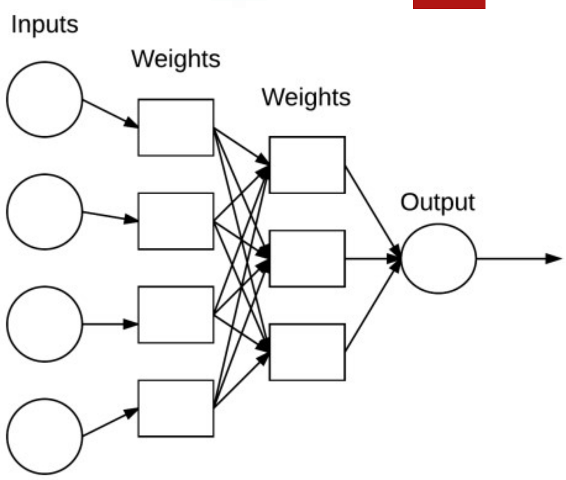
Biological Neuron	Artificial Neuron
Cell Nucleus (Soma)	Node
Dendrites	Input
Synapse	Weights or interconnections
Axon	Output



ANN

9

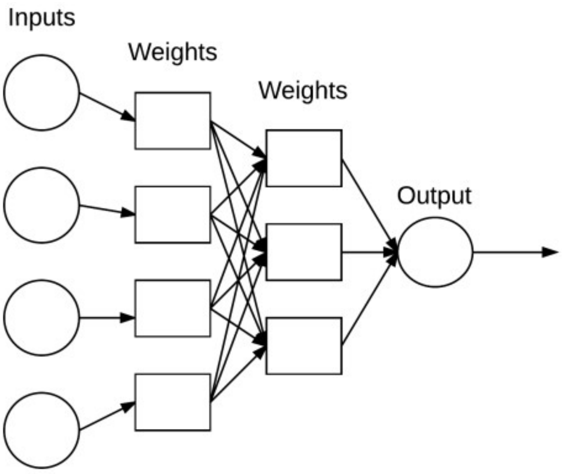
A simple neural network architecture. Inputs are presented to the network. Each connection carries a signal through the two hidden layers in the network. A final function computes the output class label.

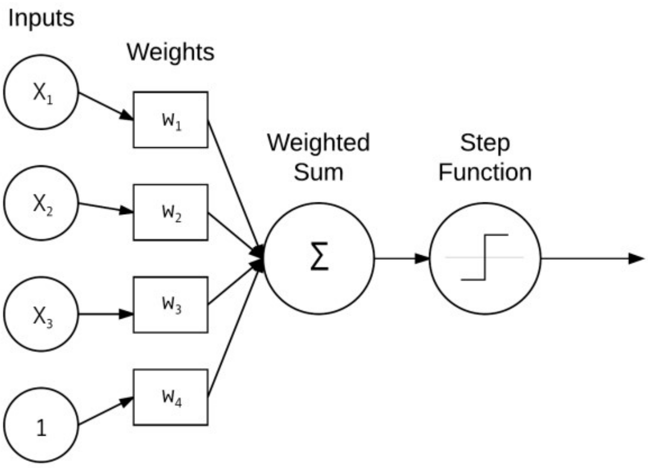


HUYTRAN

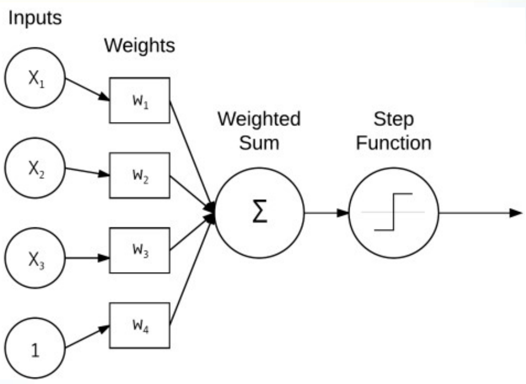
ARTIFICIAL INTELLIGENCE

ANN





Read the following and explain the meaning of each part in the figure and equations

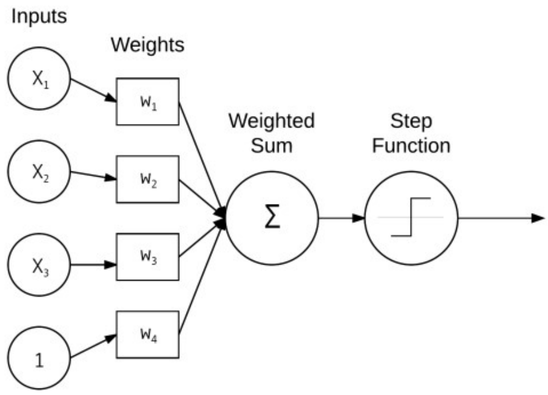


- $f(w_1x_1 + w_2x_2 + \cdots + w_nx_n)$
- $f(\sum_{i=1}^{n} w_ix_i)$
- Or simply, $f(net)$, where $net = \sum_{i=1}^{n} w_ix_i$

Activation Functions

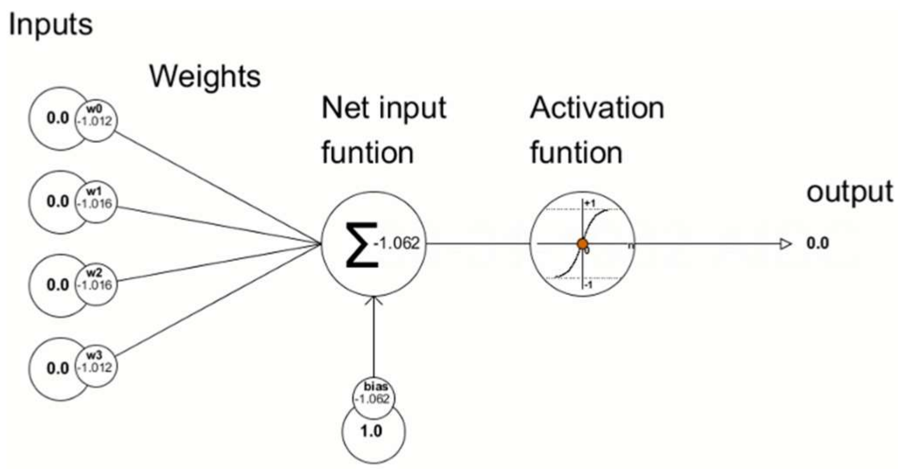
13

- What is activation function?
- How does the activation function work?
- Why do we use activation functions?
- List some types of popular activation functions?



- What is activation function?
- How does the activation function work?

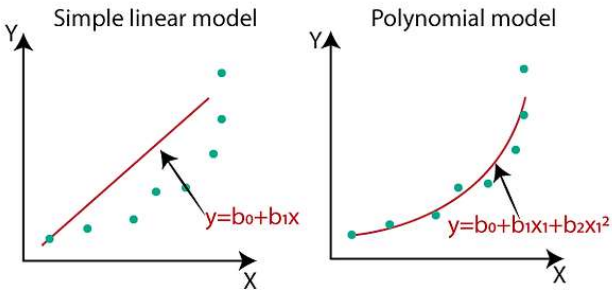
14

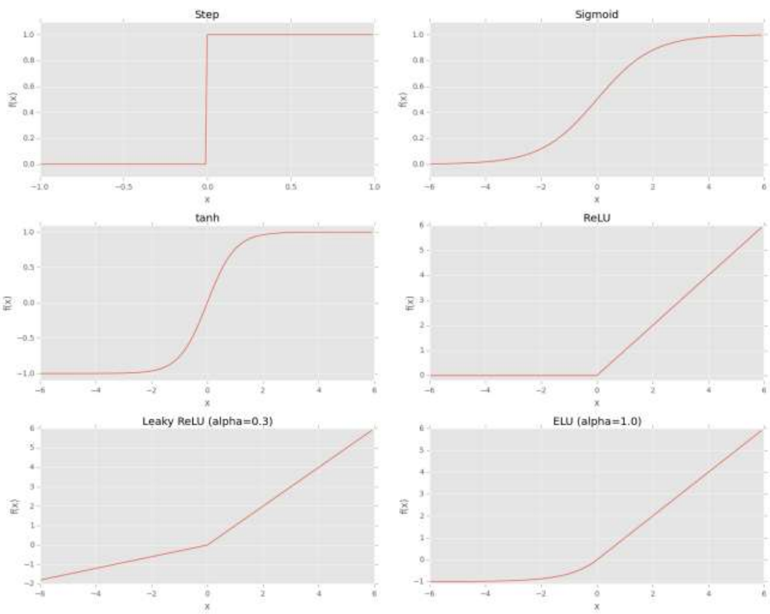


▶Why do we use activation functions?

15

1. Create non-linear characteristic for model
2. Keep the output in a specific range, such as [0, 1]; [-1, 1]





16

Popular Activation Functions

Find the equation of each activation function.

Activation Functions

17

Step function:

$$f(net)\begin{cases}1 & if net > 0\\ 0 & otherwise\end{cases}$$

Sigmoid function: $s(t) = 1/(1+e^{-t})$

ReLU function: $f(x) = max(0,x)$

HUYTRAN

ARTIFICIAL INTELLIGENCE

Activation Functions

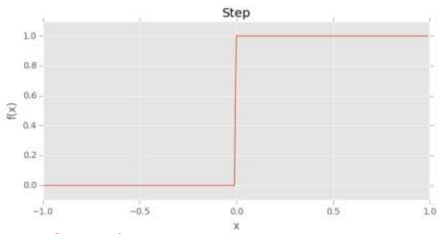
18

Step function:

$$f(net)\begin{cases}1 & if net > 0\\ 0 & otherwise\end{cases}$$

This is a very simple threshold function. If the weighted sum: $\sum_{i=1}^{n} w_i x_i > 0$, otherwise, we output 0.

The output of *f* is always zero when net is less than or equal zero. If net is greater than zero, then *f* will return one.



What is the problems of step function?

HUYTRAN

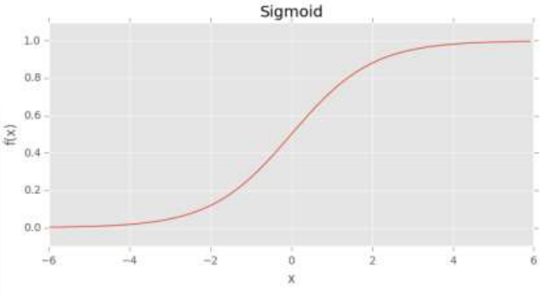
ARTIFICIAL INTELLIGENCE

Activation Functions

19

Sigmoid function: $s(t) = 1/(1+e^{-t})$

Sigmoid function is a more common activation function used in the history of NN.



HUYTRAN

ARTIFICIAL INTELLIGENCE

Activation Functions

20

Sigmoid function: $s(t) = 1/(1+e^{-t})$

Sigmoid function is a more common activation function used in the history of NN.

Why???

The primary advantage here is that the smoothness of the sigmoid function makes it easier to devise learning algorithms.

The sigmoid function is a better choice for learning than the simple step function since it:

1. Is continuous and differentiable everywhere.
2. Is symmetric around the y-axis.
3. Asymptotically approaches its saturation values.

HUYTRAN

ARTIFICIAL INTELLIGENCE

Activation Functions

21

Sigmoid function: $s(t) = 1/(1+e^{-t})$

Disadvantage of Sigmoid function:

1. The outputs of the sigmoid are not zero centered.
2. Saturated neurons essentially kill the gradient, since the delta of the gradient will be extremely small.

HUYTRAN

ARTIFICIAL INTELLIGENCE

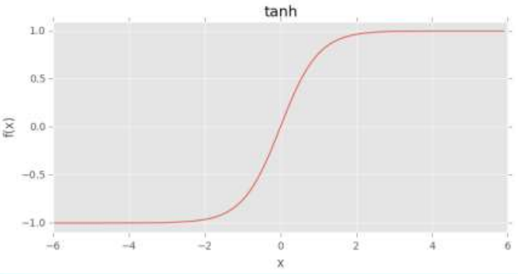
Activation Functions

22

Tanh function:

$$f(z) = tanh(z) = (e^{z} - e^{-z})/(e^{z} + e^{-z})$$

The hyperbolic tangent, or tanh (with a similar shape of the sigmoid) was also heavily used as an activation function up until the late 1990s.
The tanh function is zero centered, but the gradients are still killed when neurons become saturated



HUYTRAN

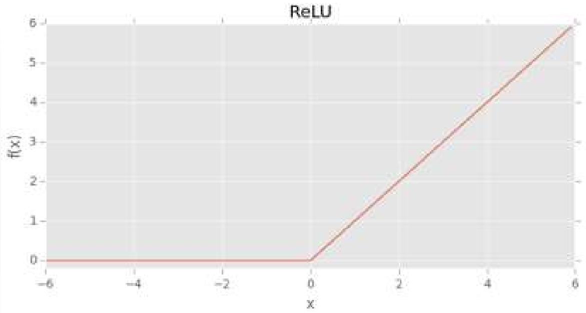
ARTIFICIAL INTELLIGENCE

Activation Functions

23

ReLU function: $f(x) = max(0, x)$

Rectified Linear Unit (ReLU) is also called "ramp functions" due to how they look when plotted.



HUYTRAN

ARTIFICIAL INTELLIGENCE

Activation Functions

24

ReLU function: $f(x) = max(0, x)$

Note:

Notice how the function is zero for negative inputs but then linearly increases for positive values. The ReLU function is not saturable and is also extremely computationally efficient.

The ReLU activation function tends to outperform both the sigmoid and tanh functions in nearly all applications.

HUYTRAN

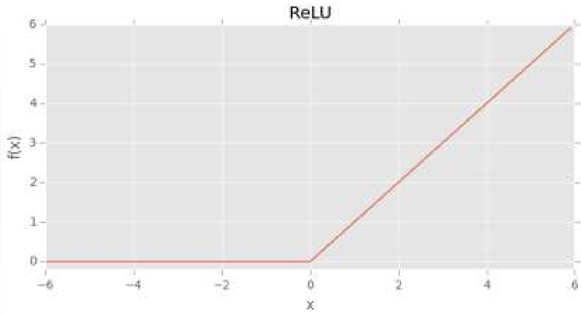
ARTIFICIAL INTELLIGENCE

Activation Functions

25

ReLU function: $f(x) = max(0, x)$

As of 2015, ReLU is the most popular activation function used in deep learning. However, a problem arises when we have a value of zero – the gradient cannot be taken.



HUYTRAN

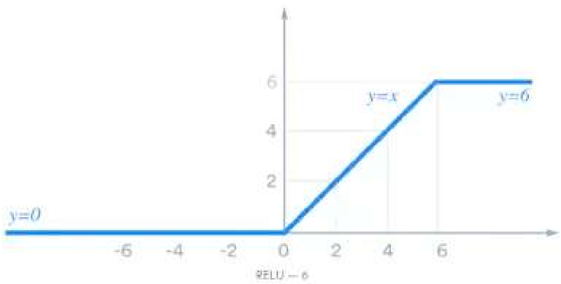
ARTIFICIAL INTELLIGENCE

Activation Functions

26

ReLU6 function: $f(x) = \min(\max(0, x), 6)$

This function limits the problem of exploding gradients



HUYTRAN

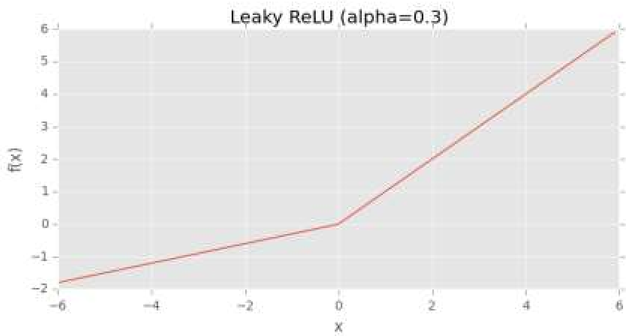
ARTIFICIAL INTELLIGENCE

Activation Functions

27

Leaky ReLU function: $$f(net) = \begin{cases} net & if\, net >= 0 \\ \alpha \times net & otherwise \end{cases}$$

Leaky ReLUs allow for a small, non-zero gradient when the unit is not active



HUYTRAN

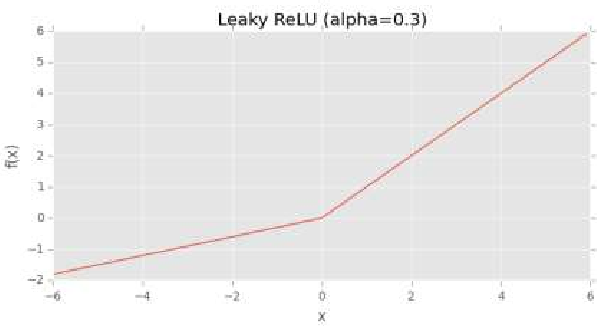
ARTIFICIAL INTELLIGENCE

Activation Functions

28

Leaky ReLU function: $$f(net) = \begin{cases} net & if\, net >= 0 \\ \alpha \times net & otherwise \end{cases}$$

The function is indeed allowed to take on a negative value, unlike traditional ReLUs which “clamp" the function output at zero. Parametric ReLUs build on Leaky ReLUs and allow the parameter α to be learned on an activation-by-activation basis, implying that each node in the network can learn a different “coefficient of leakage” separate from the other nodes.

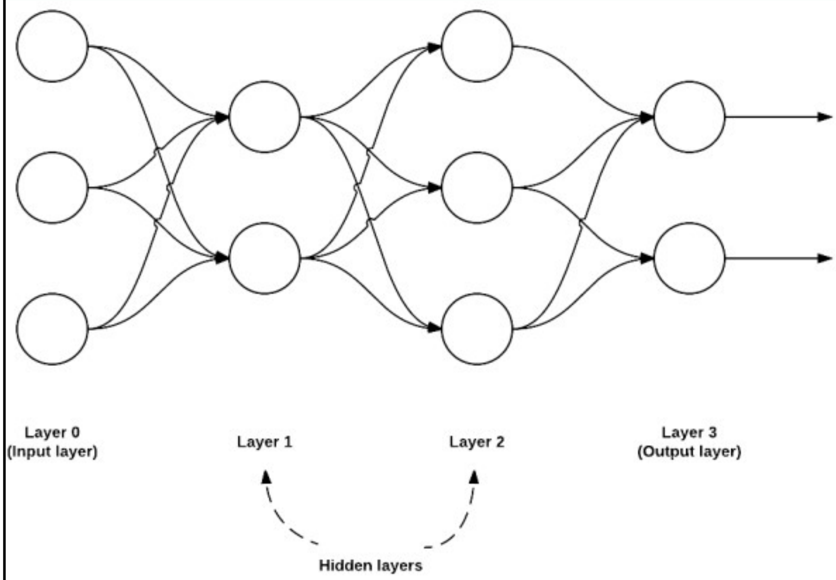


HUYTRAN

ARTIFICIAL INTELLIGENCE

Feedforward Network Architectures

29



In this type of architecture, a connection between nodes is only allowed from nodes in layer i to nodes in layer i+1 (hence the term, feedforward). There are no backward or inter-layer connections allowed. When feedforward networks include feedback connections (output connections that feed back into the inputs) they are called recurrent neural networks.

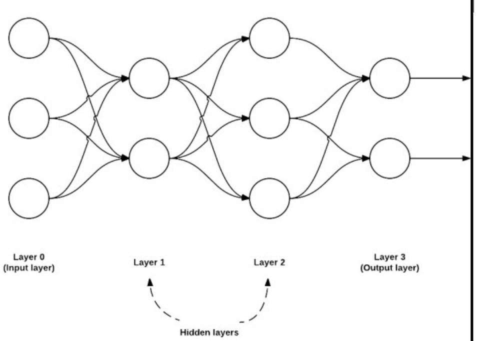
HUYTRAN

ARTIFICIAL INTELLIGENCE

Feedforward Network Architectures

30

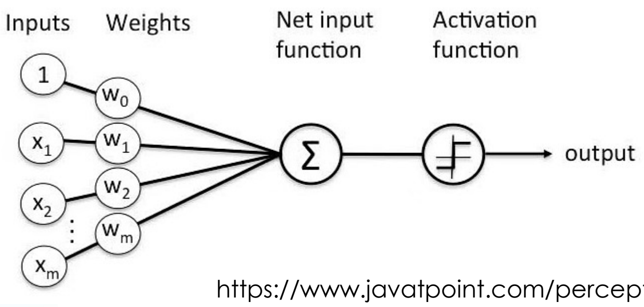
- This figure is a 3-2-3-2 feedforward network
- Layer 0 contains 3 inputs, our x_i values. These could be raw pixel intensities of an image or a feature vector extracted from the image.
- Layers 1 and 2 are hidden layers containing 2 and 3 nodes, respectively.
- Layer 3 is the output layer or the visible layer – there is where we obtain the overall output classification from our network.
- The output layer typically has as many nodes as class labels; one node for each potential output.
- For example, if we were to build an NN to classify handwritten digits, our output layer would consist of 10 nodes, one for each digit 0-9.



PERCEPTRON ALGORITHM

31

Perceptron was introduced by Frank Rosenblatt in 1957. He proposed a Perceptron learning rule based on the original MCP neuron. A Perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and processes elements in the training set one at a time.



https://www.javatpoint.com/perceptron-in-machine-learning

HUYTRAN

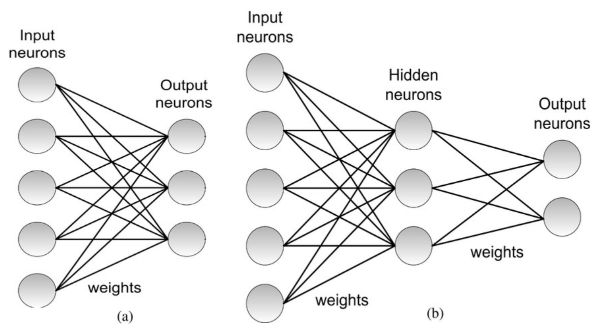
ARTIFICIAL INTELLIGENCE

TYPES OF PERCEPTRON

32

- 1. Single layer (a): Single layer perceptron can learn only linearly separable patterns.

- 2. Multilayer (b): Multilayer perceptrons can learn about two or more layers having a greater processing power.



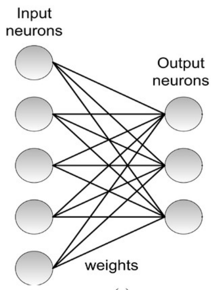
https://www.javatpoint.com/perceptron-in-machine-learning

TYPES OF PERCEPTRON

33

A single-layered perceptron model consists feed-forward network and also includes a threshold transfer function inside the model. The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes.

A single layer perceptron model do not contain recorded data, so it begins with inconstantly allocated input for weight parameters. Further, it sums up all inputs (weight). After adding all inputs, if the total sum of all inputs is more than a pre-determined value, the model gets activated and shows the output value as +1.
If the outcome is same as pre-determined or threshold value, then the performance of this model is stated as satisfied, and weight demand does not change. However, this model consists of a few discrepancies triggered when multiple weight inputs values are fed into the model. Hence, to find desired output and minimize errors, some changes should be necessary for the weights input.

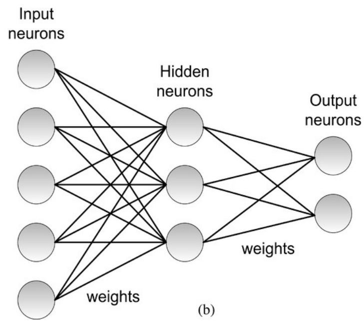


(a)

TYPES OF PERCEPTRON

34

The multi-layer perceptron model is also known as the Backpropagation algorithm, which executes in two stages as follows:
•**Forward Stage:** Activation functions start from the input layer in the forward stage and terminate on the output layer.
•**Backward Stage:** In the backward stage, weight and bias values are modified as per the model's requirement. In this stage, the error between actual output and demanded originated backward on the output layer and ended on the input layer.



(b)

TYPES OF PERCEPTRON

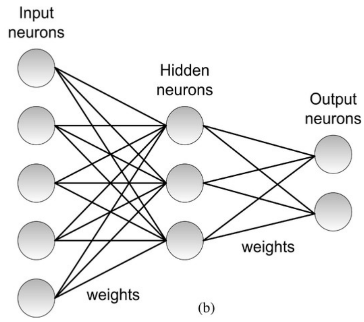
35

Advantages of Multi-Layer Perceptron:

•A multi-layered perceptron model can be used to solve complex non-linear problems.
•It works well with both small and large input data.
•It helps us to obtain quick predictions after the training.
•It helps to obtain the same accuracy ratio with large as well as small data.

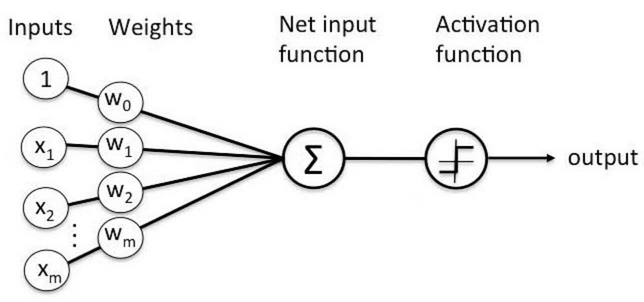
Disadvantages of Multi-Layer Perceptron:

• Computations are difficult and time-consuming.
•It is difficult to predict how much the dependent variable affects each independent variable.
•The model functioning depends on the quality of the training.



Basic Components of Perceptron

36



Mr. Frank Rosenblatt invented the perceptron model as a binary classifier which contains three main components:

- Input Nodes or Input Layer
- Weight and Bias
- Activation Function

https://www.javatpoint.com/perceptron-in-machine-learning

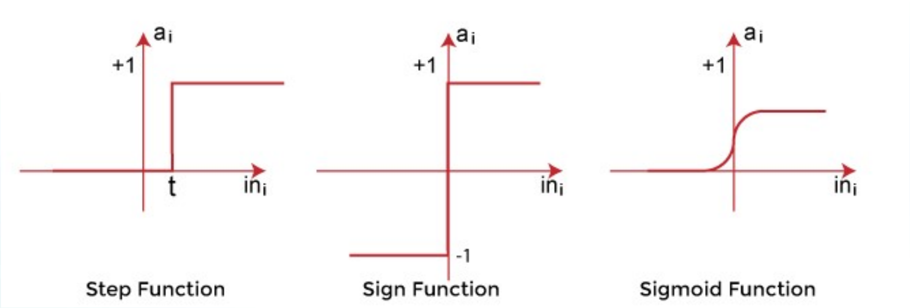
HUYTRAN

ARTIFICIAL INTELLIGENCE

Basic Components of Perceptron

37

Types of Activation functions:

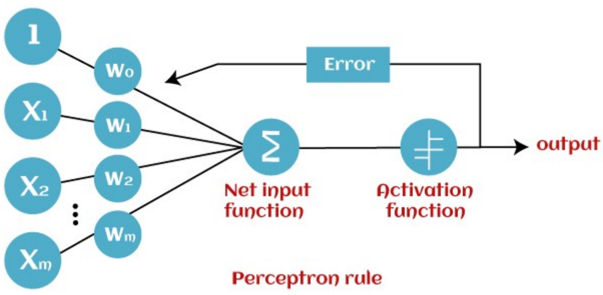


https://www.javatpoint.com/perceptron-in-machine-learning

How does Perceptron work?

38

In Machine Learning, Perceptron is considered as a single-layer neural network that consists of four main parameters named input values (Input nodes), weights and Bias, net sum, and an activation function.
The perceptron model begins with the multiplication of all input values and their weights, then adds these values together to create the weighted sum. Then this weighted sum is applied to the activation function 'f' to obtain the desired output. This activation function is also known as the **step function** and is represented by **'f'**.

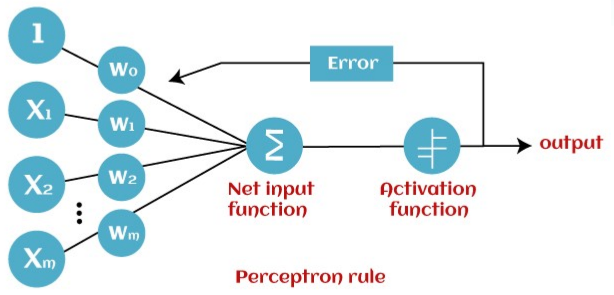


Write the final equation based on these info?

https://www.javatpoint.com/perceptron-in-machine-learning

How does Perceptron work?

39



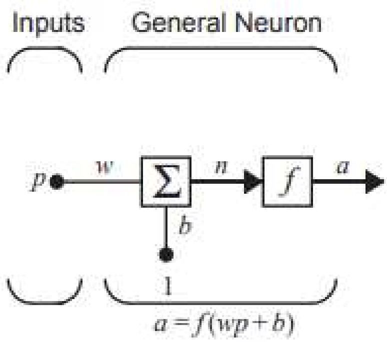
For example, $x_1 = 2$, $x_2 = 3$, $x_3 = 1$, w_n are certain numbers in the range [0,1], step function is used. Estimate the ouput.

HUYTRAN

ARTIFICIAL INTELLIGENCE

40

Single-Input Neuron



HUYTRAN

ARTIFICIAL INTELLIGENCE

Problem 1: The input to a single-input neuron is 2.0, its weight is 2.3 and its bias is -3.

i. What is the net input to the transfer function?

ii. What is the neuron output?

Transfer Functions

Name	Input/Output Relation	Icon	MATLAB Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Linear	$a = n$		purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$		satlins
Log-Sigmoid	$a = \frac{1}{1 + e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		poslin

43

Problem 2: The input to a single-input neuron is 2.0, its weight is 2.3 and its bias is -3. What is the output of the neuron if it has the following transfer functions?

i. Hard limit

ii. Linear

iii. Log-sigmoid

44

Problem 3:

Given a two-input neuron with the following parameters: $b = 1.2$, $W = [3\ 2]$, and $p = [-5\ 6]^T$, calculate the neuron output for the following transfer functions:

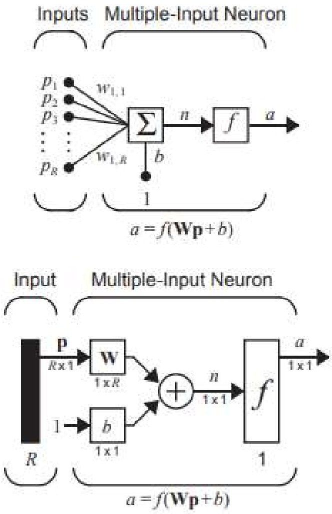
i. A symmetrical hard limit transfer function

ii. A saturating linear transfer function

iii. A hyperbolic tangent sigmoid (tansig) transfer function

45

Multiple-Input Neuron



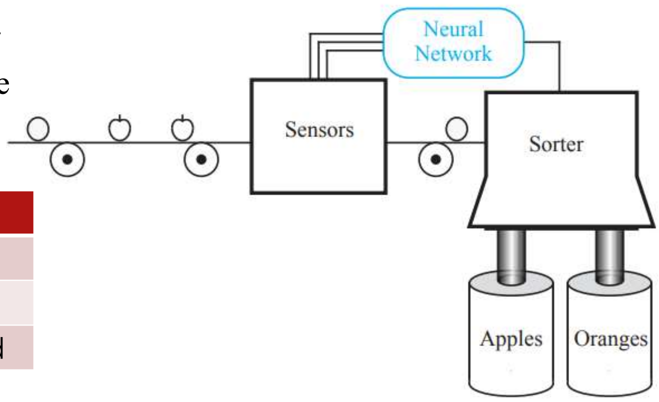
HUYTRAN

ARTIFICIAL INTELLIGENCE

An illustrative example

46

There is a conveyer belt on which the fruit is loaded. This conveyer passes through a set of sensors, which measure three properties of the fruit: shape, texture and weight.



Value	1	-1
Shape	round	elliptical
Texture	smooth	rough
Weight	>1 pound	<=1 pound

The three sensor outputs will then be input to a neural network. The purpose of the network is to decide which kind of fruit is on the conveyor. Let's assume that there are only two kinds of fruit on the conveyor: apples and oranges.

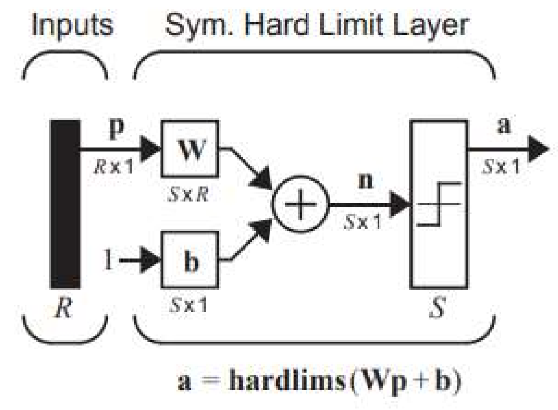
HUYTRAN

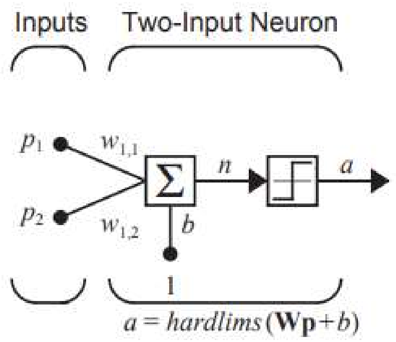
ARTIFICIAL INTELLIGENCE

An illustrative example

47

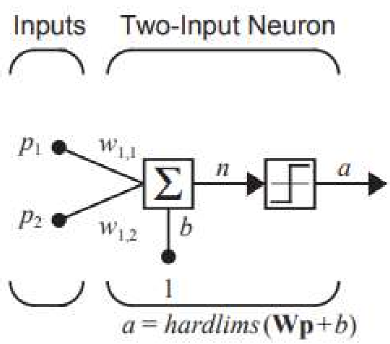
Apply the following perceptron model to the previous problem in the case of two inputs





An illustrative example

48



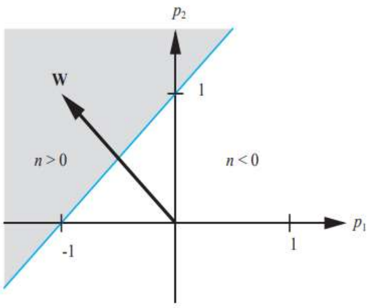
If $w_{1,1} = -1$, $w_{1,2} = 1$, find a?

An illustrative example

49

Therefore, if the inner product of the weight matrix (a single row vector in this case) with the input vector is greater than or equal to -b, the output will be 1. If the inner product of the weight vector and the input is less than -b, the output will be -1.
This divides the input space into two parts. The figure illustrates this for the case where b = -1. The blue line in the figure represents all points for which the net input is equal to 0:

$$n = [-1\ 1]p - 1 = 0$$



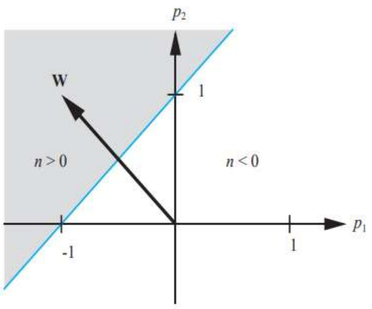
An illustrative example

50

The decision boundary between the categories is determined by the equation

$$\mathbf{W}p + b = 0$$

Because the boundary must be linear, the single-layer perceptron can only be used to recognize patterns that are linearly separable

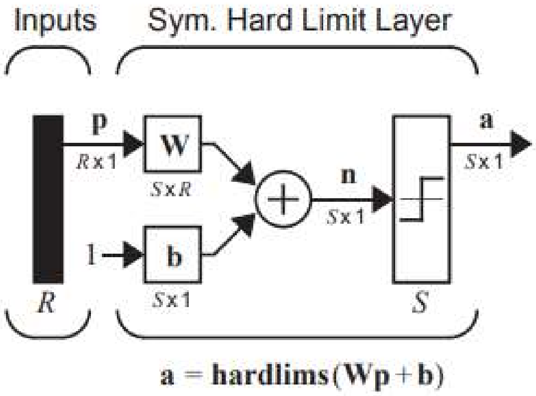


An illustrative example

51

Apply the following perceptron model to the previous problem in the case of three inputs

Find a



HUYTRAN

An illustrative example

52

We want to choose the bias and the elements of the weight matrix so that the perceptron will be able to distinguish between apples and oranges. For example, we may want the output of the perceptron to be 1 when an apple is input and -1 when an orange is input.

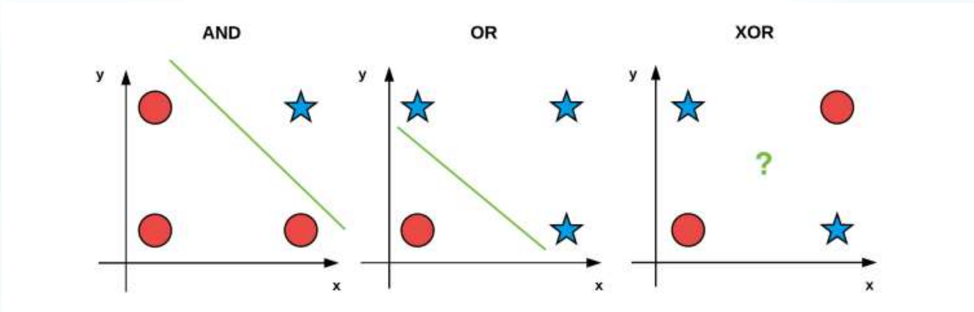
HUYTRAN

AND, OR, and XOR Datasets

$\boldsymbol{x}_0$	$\boldsymbol{x}_1$	$\boldsymbol{x}_0 \& \boldsymbol{x}_1$
0	0	0
0	1	0
1	0	0
1	1	1

$\boldsymbol{x}_0$	$\boldsymbol{x}_1$	$\boldsymbol{x}_0 \vert \boldsymbol{x}_1$
0	0	0
0	1	1
1	0	1
1	1	1

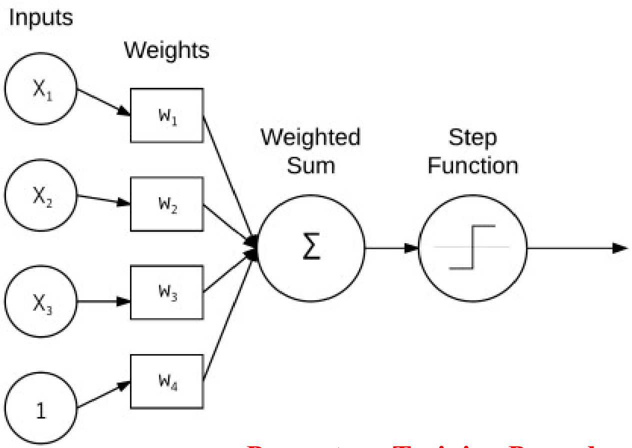
$\boldsymbol{x}_0$	$\boldsymbol{x}_1$	$\boldsymbol{x}_0 \wedge \boldsymbol{x}_1$
0	0	0
0	1	1
1	0	1
1	1	0



HUYTRAN

ARTIFICIAL INTELLIGENCE

54



Perceptron Training Procedure and the Delta Rule (step 2c)

1. Initialize our weight vector $\boldsymbol{w}$ with small random values
2. Until Perceptron converges:
   (a) Loop over each feature vector x_j and true class label d_i in our training set D
   (b) Take $\boldsymbol{x}$ and pass it through the network, calculating the output value: $y_j = f(\boldsymbol{w(t)} \cdot \boldsymbol{x}_j)$
   (c) Update the weights $\boldsymbol{w}$: $w_i(t+1) = w_i(t) + \alpha(d_j - y_j)x_{j,i}$ for all features $0 <= i <= n$

Implementing the Perceptron in Python

55

```
# import the necessary packages
import numpy as np

class Perceptron:
    def __init__(self, N, alpha=0.1):
        # initialize the weight matrix and store the learning rate
        self.W = np.random.randn(N + 1) / np.sqrt(N)
        self.alpha = alpha
```


Implementing the Perceptron in Python

56

```
# import the necessary packages
import numpy as np

class Perceptron:
    def __init__(self, N, alpha=0.1):
        # initialize the weight matrix and store the learning rate
        self.W = np.random.randn(N + 1) / np.sqrt(N)
        self.alpha = alpha
```

Implementing the Perceptron in Python

57

```
    def step(self, x):
        # apply the step function
        return 1 if x > 0 else 0
```

```
    def fit(self, X, y, epochs=10):
        # insert a column of 1's as the last entry in the feature
        # matrix -- this little trick allows us to treat the bias
        # as a trainable parameter within the weight matrix
        X = np.c_[X, np.ones((X.shape[0]))]
```


Implementing the Perceptron in Python

58

```
    def step(self, x):
        # apply the step function
        return 1 if x > 0 else 0
```

Implementing the Perceptron in Python


```
    def fit(self, X, y, epochs=10):
        # insert a column of 1's as the last entry in the feature
        # matrix -- this little trick allows us to treat the bias
        # as a trainable parameter within the weight matrix
        X = np.c_[X, np.ones((X.shape[0]))]
```


Implementing the Perceptron in Python


```
        # loop over the desired number of epochs
        for epoch in np.arange(0, epochs):
            # loop over each individual data point
            for (x, target) in zip(X, y):
                # take the dot product between the input features
                # and the weight matrix, then pass this value
                # through the step function to obtain the prediction
                p = self.step(np.dot(x, self.W))

                # only perform a weight update if our prediction
                # does not match the target
                if p != target:
                    # determine the error
                    error = p - target

                    # update the weight matrix
                    self.W += -self.alpha * error * x
```

```python
        # loop over the desired number of epochs
        for epoch in np.arange(0, epochs):
            # loop over each individual data point
            for (x, target) in zip(X, y):
                # take the dot product between the input features
                # and the weight matrix, then pass this value
                # through the step function to obtain the prediction
                p = self.step(np.dot(x, self.W))

                # only perform a weight update if our prediction
                # does not match the target
                if p != target:
                    # determine the error
                    error = p - target

                    # update the weight matrix
                    self.W += -self.alpha * error * x
```

61


```python
        # loop over the desired number of epochs
        for epoch in np.arange(0, epochs):
            # loop over each individual data point
            for (x, target) in zip(X, y):
                # take the dot product between the input features
                # and the weight matrix, then pass this value
                # through the step function to obtain the prediction
                p = self.step(np.dot(x, self.W))

                # only perform a weight update if our prediction
                # does not match the target
                if p != target:
                    # determine the error
                    error = p - target

                    # update the weight matrix
                    self.W += -self.alpha * error * x
```


62

63

```
    def predict(self, X, addBias=True):
        # ensure our input is a matrix
        X = np.atleast_2d(X)

        # check to see if the bias column should be added
        if addBias:
            # insert a column of 1's as the last entry in the feature
            # matrix (bias)
            X = np.c_[X, np.ones((X.shape[0]))]

        # take the dot product between the input features and the
        # weight matrix, then pass the value through the step
        # function
        return self.step(np.dot(X, self.W))
```


Evaluating the Perceptron Bitwise Datasets

64

```
# import the necessary packages
from pyimagesearch.nn import Perceptron
import numpy as np

# construct the OR dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [1]])

# define our perceptron and train it
print("[INFO] training perceptron...")
p = Perceptron(X.shape[1], alpha=0.1)
p.fit(X, y, epochs=20)
```

Evaluating the Perceptron Bitwise Datasets

65

```
# now that our perceptron is trained we can evaluate it
print("[INFO] testing perceptron...")

# now that our network is trained, loop over the data points
for (x, target) in zip(X, y):
    # make a prediction on the data point and display the result
    # to our console
    pred = p.predict(x)
    print("[INFO] data={}, ground-truth={}, pred={}".format(
        x, target[0], pred))
```


Evaluating the Perceptron Bitwise Datasets

66

```
$ python perceptron_or.py
[INFO] training perceptron...
[INFO] testing perceptron...
[INFO] data=[0 0], ground-truth=0, pred=0
[INFO] data=[0 1], ground-truth=1, pred=1
[INFO] data=[1 0], ground-truth=1, pred=1
[INFO] data=[1 1], ground-truth=1, pred=1
```

Evaluating the Perceptron Bitwise Datasets


```python
# import the necessary packages
from pyimagesearch.nn import Perceptron
import numpy as np

# construct the AND dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [0], [0], [1]])

# define our perceptron and train it
print("[INFO] training perceptron...")
p = Perceptron(X.shape[1], alpha=0.1)
p.fit(X, y, epochs=20)
```


Evaluating the Perceptron Bitwise Datasets


```python
# now that our perceptron is trained we can evaluate it
print("[INFO] testing perceptron...")

# now that our network is trained, loop over the data points
for (x, target) in zip(X, y):
	# make a prediction on the data point and display the result
	# to our console
	pred = p.predict(x)
	print("[INFO] data={}, ground-truth={}, pred={}".format(
		x, target[0], pred))
```

Evaluating the Perceptron Bitwise Datasets

69

```
$ python perceptron_and.py
[INFO] training perceptron...
[INFO] testing perceptron...
[INFO] data=[0 0], ground-truth=0, pred=0
[INFO] data=[0 1], ground-truth=0, pred=0
[INFO] data=[1 0], ground-truth=0, pred=0
[INFO] data=[1 1], ground-truth=1, pred=1
```