TRIBHUVAN UNIVERSITY



**Sagarmatha College of Science &**

**Technology**

Lab Report On: Neural Network

Lab Report No.: 01

Date: 2077-07-23

**SUBMITTED BY SUBMITTED TO**

***Name: Ramesh Neupane CSIT Department***

***Roll no.: 37***

**OBJECTIVE**

To implement some basic activation function to single neuron perceptron

**LAB QUESTION**

Consider the following neural network. Compute output of the network by assuming each of the following activation function:

a. Step function with threshold 1.

b. Linear function 2x-1

c. Sigmoid Function

d. Tanh Function

b=-2

W1=1

X1

y

W2=1

X2

**SOURCE CODE AND OUTPUT**

1. **Threshold Activation Function**

import numpy as np

# a node to calculate the input for the activation function

def node(w, x, b):

v = w \* x

n = np.sum(v) + b

print(f"n: {n}")

return threshold(n)

# threshold activation function

def threshold(n):

if n >= 0:

return 1

else:

return 0

weights = np.array([1, 1]) # given in question

bias = np.array([-2.0]) # given in question

print("Enter the inputs: \n")

inputs = [] # store inputs x1 and x2

for i in range(2):

e = float(input(f"x[{i+1}]: "))

inputs.append(e)

inputs = np.array(inputs) # input list into np.array

output = node(weights, inputs, bias)

print(f"Output: {output}")

**Output:**

1. Enter the inputs:

x[1]: 1

x[2]: -1

n: [-2.]

Output: 0

1. Enter the inputs:

x[1]: 2

x[2]: 1

n: [1.]

Output: 1

1. **Linear Activation Function with 2x-1**

import numpy as np

# a node to calculate the input for the activation function

def node(w, x, b):

v = w \* x

n = np.sum(v) + b

print(f"n: {n}")

return linear(n)

# linear activation function

def linear(n):

return (2 \* n - 1)

weights = np.array([1, 1]) # given in question

bias = np.array([-2.0]) # given in question

print("Enter the inputs: \n")

inputs = [] # store inputs x1 and x2

for i in range(2):

e = float(input(f"x[{i+1}]: "))

inputs.append(e)

inputs = np.array(inputs) # input list into np.array

output = node(weights, inputs, bias)

print(f"Output: {output}")

**Output:**

Enter the inputs:

x[1]: 1

x[2]: 0.5

n: [-0.5]

Output: [-2.]

1. **Sigmoid Activation Function**

import numpy as np

from math import exp

# a node to calculate the input for the activation function

def node(w, x, b):

v = w \* x

n = np.sum(v) + b

print(f"n: {n}")

return sigmoid(n)

# sigmoid activation function

def sigmoid(n):

return (1/(1 + exp(-n)))

weights = np.array([1, 1]) # given in question

bias = np.array([-2.0]) # given in question

print("Enter the inputs: \n")

inputs = [] # store inputs x1 and x2

for i in range(2):

e = float(input(f"x[{i+1}]: "))

inputs.append(e)

inputs = np.array(inputs) # input list into np.array

output = node(weights, inputs, bias)

print(f"Output: {output}")

**Output:**

Enter the inputs:

x[1]: 2

x[2]: 1

n: [1.]

Output: 0.73106

1. **Tanh Activation Function**

import numpy as np

from math import tanh

# a node to calculate the input for the activation function

def node(w, x, b):

v = w \* x

n = np.sum(v) + b

print(f"n: {n}")

return hyp\_tangent(n)

# tanh activation function

def hyp\_tangent(n):

return tanh(n)

weights = np.array([1, 1]) # given in question

bias = np.array([-2.0]) # given in question

print("Enter the inputs: \n")

inputs = [] # store inputs x1 and x2

for i in range(2):

e = float(input(f"x[{i+1}]: "))

inputs.append(e)

inputs = np.array(inputs) # input list into np.array

output = node(weights, inputs, bias)

print(f"Output: {output}")

**Output:**

Enter the inputs:

x[1]: -2

x[2]: -1

n: [-5.]

Output: -0.999909

**CONCLUSION**

Hence, we are able to implement some basic activation functions (i.e. threshold, linear, sigmoid, tanh) to single neuron perceptron.