School of Computer Science

UNIVERSITY OF PERTOLEUM AND ENERGY STUDIES

DEHRADUN, UTTARAKHAND

Intro to Artificial Intelligence Lab

B.Tech CSE AIML B2 (Non Honours)

Lab File

Submitted to:

Mr. Biswa Mohan Sahoo

Submitted by:

Aryan Mohan

500092142

INDEX

S.NO.	TOPIC	PAGE NUMBER	DATE	SIGNATURE

AIML LAB

Experiment – 1

Ques) Write a program to solve different set operations using python.

Code:

```
set1={3,4,13,645,78,23,76,13,69,78,798}
set2={13,45,685,134,574,76,23,12,4,3,}


print("1 for Union")
print("2 for Intersection")
print("3 for Difference")
choice=int(input("Enter your choice: "))


if(choice==1):
   set1=set1.union(set2)
   print("Union between set1 and set2 is: ", set1)


elif(choice==2):
   set1=set1.intersection(set2)
   print("Intersection between set1 and set2 is: ", set1)


elif(choice==3):
   set1=set2.difference(set1)
   print("Difference between set1 and set2 is: ", set1)
```

Output:

```
1 for Union
2 for Intersection
3 for Difference
Enter your choice: 2
Intersection between set1 and set2 is:  {3, 4, 76, 13, 23}
```

AIML LAB

Experiment – 3

Ques) Write a program to implement Tautologies, Contradiction, and Satisfiability.

Code:

```
p=[True, True, False, False]
q=[True, False, True, False]

def not_op(a):
   z=[]
   for i in a:
      if i==True:
         z.append(False)
      else:
         z.append(True)
   return z

np=not_op(p)
nq=not_op(q)

def imp(p,q):
   z=[]
   for i in range(0,len(p)):
      if (not_op(p)[i] or q[i] == True):
         z.append(True)
      else:
```

```
        z.append(False)
    return z

piq=imp(p,q)
qip=imp(q,p)

def status(l):
    x=set(l)
    if x=={True,False}:
        print("Contigency")
    elif x=={True}:
        print("Tautology")
    elif x=={False}:
        print("Contradiction")

print("For P =",p,"and Q =",q)

# (p->q) or (q->p)

exp1=[]
for i in range(0,len(p)):
    if(piq[i] or qip[i] == True):
        exp1.append(True)
    else:
        exp1.append(False)
print("(P->Q)v(Q->P) is a",end=" ")
status(exp1)

# p or q

exp2=[]
```

```
for i in range(0,len(p)):
    if(p[i] or q[i]==True):
        exp2.append(True)
    else:
        exp2.append(False)
print("P v Q is a",end=" ")
status(exp2)

# not(p->q) and (q->p)

npiq=not_op(piq)
nqip=not_op(qip)
exp3=[]
for i in range(0,len(p)):
    if(npiq[i] and nqip[i]==True):
        exp3.append(True)
    else:
        exp3.append(False)
print("not(P->Q)^(Q->P) is a",end=" ")
status(exp3)
```

Output:

```
For P = [True, True, False, False] and Q = [True, False, True, False]
(P->Q)v(Q->P) is a Tautology
P v Q is a Contigency
not(P->Q)^(Q->P) is a Contradiction
```

AIML Lab Experiment – 5

Ques) Write a program to implement Bayesian Network in a given real-world problem. Compute the accuracy of the Bayesian Network, considering few test data sets.

Code:

```
import pgmpy.models

import pgmpy.inference

import pgmpy.factors.discrete


alarm_model=pgmpy.models.BayesianNetwork([("Burglary","Alarm"),("Earthquake","Alarm"),("Alar
m","John"),("Alarm","Mary")])


cpd_burglary=pgmpy.factors.discrete.TabularCPD("Burglary",2 , [[0.99],[0.01]])

cpd_earthquake=pgmpy.factors.discrete.TabularCPD("Earthquake", 2, [[0.99],[0.01]])

cpd_alarm=pgmpy.factors.discrete.TabularCPD("Alarm", 2,
[[0.99,0.71,0.06,0.05],[0.01,0.29,0.94,0.95]],evidence=["Burglary","Earthquake"],evidence_card=[2,2
])

cpd_john=pgmpy.factors.discrete.TabularCPD("John",
2,[[0.95,0.1],[0.05,0.9]],evidence=["Alarm"],evidence_card=[2])

cpd_mary=pgmpy.factors.discrete.TabularCPD("Mary",
2,[[0.1,0.7],[0.9,0.3]],evidence=["Alarm"],evidence_card=[2])

alarm_model.add_cpds(cpd_alarm,cpd_burglary,cpd_earthquake,cpd_john,cpd_mary)
```

```
print(alarm_model.check_model())
print(cpd_burglary)
print(cpd_earthquake)
print(cpd_alarm)
print(cpd_john)
print(cpd_mary)
print(alarm_model)

infer = pgmpy.inference.VariableElimination(alarm_model)
posterior_probability = infer.query(['Earthquake'], evidence={'John': 0, 'Mary': 0})
posterior_probability1 = infer.query(['Burglary'], evidence={'John': 0, 'Mary': 0})
posterior_probability2 = infer.query(['Alarm'], evidence={'Earthquake': 0, 'Burglary': 0})
posterior_probability3 = infer.query(['Earthquake','Alarm'], evidence={'John': 1, 'Mary': 1})
posterior_probability4 = infer.query(['Alarm'], evidence={'John': 1, 'Mary': 0})

print("Probability of Earthquake if John and Marry calls")
print(posterior_probability)
print("Probability of Burglary if John and Marry calls")
print(posterior_probability1)
print("Probability of Alarm if Burglary or Earthquake Happens")
print(posterior_probability2)
print("Probability of Alarm due to Earthquake if John and Marry calls")
print(posterior_probability3)
print("Probability of Earthquake if John and Marry calls")
print(posterior_probability4)
```

Output:

```
True
+---------------+------+
| Burglary(0)   | 0.99 |
+---------------+------+
| Burglary(1)   | 0.01 |
+---------------+------+
+----------------+------+
| Earthquake(0)  | 0.99 |
+----------------+------+
| Earthquake(1)  | 0.01 |
+----------------+------+
+------------+---------------+---------------+---------------+---------------+
| Burglary   | Burglary(0)   | Burglary(0)   | Burglary(1)   | Burglary(1)   |
+------------+---------------+---------------+---------------+---------------+
| Earthquake | Earthquake(0) | Earthquake(1) | Earthquake(0) | Earthquake(1) |
+------------+---------------+---------------+---------------+---------------+
| Alarm(0)   | 0.99          | 0.71          | 0.06          | 0.05          |
+------------+---------------+---------------+---------------+---------------+
| Alarm(1)   | 0.01          | 0.29          | 0.94          | 0.95          |
+------------+---------------+---------------+---------------+---------------+
+---------+----------+----------+
| Alarm   | Alarm(0) | Alarm(1) |
+---------+----------+----------+
| John(0) | 0.95     | 0.1      |
+---------+----------+----------+
| John(1) | 0.05     | 0.9      |
+---------+----------+----------+
+---------+----------+----------+
| Alarm   | Alarm(0) | Alarm(1) |
+---------+----------+----------+
| Mary(0) | 0.1      | 0.7      |
+---------+----------+----------+
| Mary(1) | 0.9      | 0.3      |
+---------+----------+----------+
```

```
BayesianNetwork with 5 nodes and 4 edges
Finding Elimination Order: : 100%|          | 2/2 [00:00<?, ?it/s]
Eliminating: Alarm: 100%|          | 2/2 [00:00<00:00, 1003.18it/s]
Finding Elimination Order: : 100%|          | 2/2 [00:00<00:00, 1982.65it/s]
Eliminating: Alarm: 100%|          | 2/2 [00:00<00:00, 994.15it/s]
Finding Elimination Order: : : 0it [00:00, ?it/s]
0it [00:00, ?it/s]
Finding Elimination Order: : 100%|          | 1/1 [00:00<00:00, 998.88it/s]
Eliminating: Burglary: 100%|          | 1/1 [00:00<00:00, 1004.38it/s]
Finding Elimination Order: : 100%|          | 2/2 [00:00<00:00, 2015.04it/s]
Eliminating: Earthquake: 100%|          | 2/2 [00:00<00:00, 996.98it/s]
Probability of Earthquake if John and Marry calls
+---------------+-------------------+
| Earthquake    |   phi(Earthquake) |
+===============+===================+
| Earthquake(0) |            0.9907 |
+---------------+-------------------+
| Earthquake(1) |            0.0093 |
+---------------+-------------------+
Probability of Burglary if John and Marry calls
+-------------+-----------------+
| Burglary    |   phi(Burglary) |
+=============+=================+
| Burglary(0) |          0.9924 |
+-------------+-----------------+
| Burglary(1) |          0.0076 |
+-------------+-----------------+
```

```
Probability of Alarm if Burglary or Earthquake Happens
+----------+--------------+
| Alarm    |   phi(Alarm) |
+==========+==============+
| Alarm(0) |       0.9900 |
+----------+--------------+
| Alarm(1) |       0.0100 |
+----------+--------------+
Probability of Alarm due to Earthquake if John and Marry calls
+----------+---------------+-------------------------+
| Alarm    | Earthquake    |   phi(Alarm,Earthquake) |
+==========+===============+=========================+
| Alarm(0) | Earthquake(0) |                  0.8744 |
+----------+---------------+-------------------------+
| Alarm(0) | Earthquake(1) |                  0.0063 |
+----------+---------------+-------------------------+
| Alarm(1) | Earthquake(0) |                  0.1032 |
+----------+---------------+-------------------------+
| Alarm(1) | Earthquake(1) |                  0.0160 |
+----------+---------------+-------------------------+
Probability of Earthquake if John and Marry calls
+----------+--------------+
| Alarm    |   phi(Alarm) |
+==========+==============+
| Alarm(0) |       0.2601 |
+----------+--------------+
| Alarm(1) |       0.7399 |
+----------+--------------+
```

AIML LAB Experiment – 7

Ques) Write a program to solve the Travelling Salesperson algorithm for a real-world problem.

Code:

```
import random as r
chromosome = 100 # total number of chromosome in a generation
city = 10     # total cities
generations=50
mutation_rate=0.01 #less than 1 always

"""     this is the mutation part of code, swap city """
def mutate(sequence):
    i=r.randint(0,len(sequence)-1)
    j=r.randint(0,len(sequence)-1)

    temp=sequence[i]
    sequence[i]=sequence[j]
    sequence[j]=temp
    return sequence

"""     this is for selecting and doing crossover the parents (K elements tournament selection)"""
def Sort(L1):   #sorts a given list based on its 2nd element
    l = len(L1)
    for i in range(0, l):
        for j in range(0, l - i - 1):
            if (L1[j][1] > L1[j + 1][1]):
                tempo = L1[j]
```

```
            L1[j] = L1[j + 1]

            L1[j + 1] = tempo

    return L1

def parents(L1):    #selects parents from a population

    a=[]

    a.append(L1[r.randint(0, chromosome - 1)])

    a.append(L1[r.randint(0, chromosome - 1)])

    if a[0][1]>a[1][1]:

        return a[1]

    else:

        return a[0]


def offsprings(a,b):    #using the parents, it creates offsprings(2 offsprings)

    a=a[1:len(a)-1]

    b=b[1:len(b)-1]

    offspring = a[0:5]

    for city in b:

        if not city in offspring:

            offspring.append(city)

    return offspring

def crossover(List_cd):     # this creates parents, their offsprings, mutates them and mames the next
generation

    p1 = parents(List_cd)

    p2 = parents(List_cd)

    a = p1[0]

    b = p2[0]

    c1 = offsprings(a, b)

    c2 = offsprings(b, a)

    if r.random()<mutation_rate:

        c1=mutate(c1)

    c1 = [1] + c1 + [1]
```

```
    c2 = [1] + c2 + [1]
    return [[c1,distance(L1, c1)],[c2,distance(L1, c2)]]
def new_population(L1): # returns a new population list based on the previous one
    new_pop=[]
    for i in range(50):
        new_pop += crossover(L1)
    return new_pop

"""    this is for making the inital population        """
def matrix():
    L1=[]
    for i in range(city):
        L2=[]
        while True:
            x=r.randint(10,99)
            if (len(L2)==city):
                break
            elif (x in L2):
                continue
            else:
                L2.append(x)
        L1.append(L2)
    for i in range(10):
        L1[i][i]=0
    return L1
def create_chromosome():
    str1 =[1]
    while True:
        str2 = r.randint(2,10)
        if str2 not in str1:
            str1.append(str2)
```

```
        if len(str1) == 10:
            break
    str1.append(1)
    return str1
def distance(L1, node):
    node_distance = 0

    for i in range(0, len(node) - 1):
        x = node[i] - 1
        y = node[i + 1] - 1
        node_distance += L1[x][y]
    return node_distance
def create_population(L1):
    population = []  # list of population which will have all the chromosome for a given generation
    for i in range(chromosome):
        node = create_chromosome()  # randomly creates a combination
        dist = distance(L1, node)  # finds distance for the given combination
        population.append([node, dist])
    return population
# creates the inital population and measures their fitness value
L1 = matrix()
List_cd = create_population(L1)
print("Least distance in 1st generation :",Sort(List_cd)[0])

for i in range(generations):
    List_cd=new_population(List_cd)

print("Least distance in last generation :",Sort(List_cd)[0])
```

Output:

```
Least distance in 1st generation : [[1, 10, 4, 7, 9, 6, 2, 5, 8, 3, 1], 374]
Least distance in last generation : [[1, 8, 3, 7, 4, 9, 10, 5, 6, 2, 1], 328]
```

AIML Lab Experiment – 11

Ques) **Write a program to implement the linear regression, multiple linear regression algorithm, and Logistic Regression algorithms to fit data points. Select the appropriate data set for your experiment and draw graphs.**

Code:

#Simple Linear Regression

```
import numpy as np
from sklearn.linear_model import LinearRegression
x = np.array([5,15,25,35,45,55]).reshape((-1,1))
y = np.array([5,20,14,32,22,38])
model = LinearRegression().fit(x,y)
r_square = model.score(x,y)
print("Coefficient of determination: ", r_square)
est_coef = model.coef_
print("Estimated coefficients: ",est_coef)
ind_term = model.intercept_
print("Independent term in the linear model: ", ind_term)

pred_y=model.predict(np.array([30]).reshape((-1,1)))
print("Predicted value: ",pred_y)
```

```
Coefficient of determination:  0.7158756137479542
Estimated coefficients:  [0.54]
Independent term in the linear model:  5.633333333333329
Predicted value:  [21.83333333]
```

#Multiple Linear Regression

```
a = np.array([[1,1],[1,2],[2,2],[2,3]])

b = np.dot(a, np.array([1,2]))+3


print(a)

print(b)


model = LinearRegression().fit(a,b)

r_square = model.score(a,b)

print("Coefficient of determination: ", r_square)

est_coef = model.coef_

print("Estimated coefficients: ",est_coef)

ind_term = model.intercept_

print("Independent term in the linear model: ", ind_term)


pred_b=model.predict(np.array([[4,5]]))

print("Predicted value: ",pred_b)
```

```
[[1 1]
 [1 2]
 [2 2]
 [2 3]]
[ 6  8  9 11]
Coefficient of determination:  1.0
Estimated coefficients:  [1. 2.]
Independent term in the linear model:  3.0000000000000018
Predicted value:  [17.]
```

#Simple Linear Regression without sklearn

```
def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)

x = np.array([5,15,25,35,45,55])
y = np.array([5,20,14,32,22,38])

# estimating coefficients
b = estimate_coef(x, y)
print("Estimated coefficients: = {} \nIndependent Term = {}".format(b[0], b[1]))
```

```
Estimated coefficients: = 5.633333333333329
Independent Term = 0.54
```

#Gradient Descent

```
import numpy as np

def mean_squared_error(y_true, y_predicted):

    # Calculating the loss or cost
    cost = np.sum((y_true-y_predicted)**2) / len(y_true)
    return cost

# Gradient Descent Function
# Here iterations, learning_rate, stopping_threshold
# are hyperparameters that can be tuned
def gradient_descent(x, y, iterations = 1000, learning_rate = 0.0001,
                                        stopping_threshold = 1e-6):

    # Initializing weight, bias, learning rate and iterations
    current_weight = 0.1
    current_bias = 0.01
    iterations = iterations
    learning_rate = learning_rate
    n = float(len(x))

    costs = []
    weights = []
    previous_cost = None

    # Estimation of optimal parameters
    for i in range(2000):

        # Making predictions
        y_predicted = (current_weight * x) + current_bias
```

```
        # Calculationg the current cost
        current_cost = mean_squared_error(y, y_predicted)

        # If the change in cost is less than or equal to
        # stopping_threshold we stop the gradient descent
        if previous_cost and abs(previous_cost-current_cost)<=stopping_threshold:
            break

        previous_cost = current_cost

        costs.append(current_cost)
        weights.append(current_weight)

        # Calculating the gradients
        weight_derivative = -(2/n) * sum(x * (y-y_predicted))
        bias_derivative = -(2/n) * sum(y-y_predicted)

        # Updating weights and bias
        current_weight = current_weight - (learning_rate * weight_derivative)
        current_bias = current_bias - (learning_rate * bias_derivative)

        # Printing the parameters for each 1000th iteration
        print(f"Iteration {i+1}: Cost {current_cost}, Weight \
        {current_weight}, Bias {current_bias}")


    # Visualizing the weights and cost at for all iterations

    return current_weight, current_bias

X = np.array([5,15,25,35,45,55])
```

```
Y = np.array([5,20,14,32,22,38])

# Estimating weight and bias using gradient descent
estimated_weight, eatimated_bias = gradient_descent(X, Y, iterations=2000)
print(f"Estimated Weight: {estimated_weight}\nEstimated Bias: {eatimated_bias}")

# Making predictions using estimated parameters
Y_pred = estimated_weight*X + eatimated_bias
```

```
Estimated Weight: 0.6680634595104552
Estimated Bias: 0.5474122810062275
```

AIML Lab Experiment – 11(Logistic Regression)

Code:

```
from sklearn.linear_model import LogisticRegression

import numpy as np

x = np.array([5,15,25,35,45,55]).reshape((-1,1))

y = np.array([5,20,14,32,22,38])

model = LogisticRegression(solver='liblinear').fit(x,y)

r_square = model.score(x,y)

print("Coefficient of determination: ", r_square)

est_coef = model.coef_

print("Estimated coefficients: ",est_coef)

ind_term = model.intercept_

print("Independent term in the linear model: ", ind_term)


pred_y=model.predict(np.array([30]).reshape((-1,1)))

print("Predicted value: ",pred_y)
```

```
Coefficient of determination:  0.16666666666666666
Estimated coefficients:  [[-0.15356611]
 [-0.04398542]
 [-0.07247123]
 [-0.01180555]
 [-0.02585367]
 [ 0.00023683]]
Independent term in the linear model:  [ 0.41541363 -0.21423242  0.0406801  -0.63610267 -0.4334289  -0.83039622]
Predicted value:  [38]
```

AIML Lab Experiment – 9

Ques) Design and implement a Perceptron learner and Neural Networks learner and test on real-world problem data sets.

Code:

```
x1 = 0.5
x2 = 0.2
w11 = 0.7
w12 = 0.3
w21 = 0.14
w22 = -0.6
w31 = 0.9
w32 = 0.8
def sigmoid(a):
   return (1/(1 + 2.71**(0-a)))
hn1 = x1 * w11 + x2 * w21
z1 = sigmoid(hn1)
hn2 = x1 * w12 + x2 * w22
z2 = sigmoid(hn2)

o1 = z1 * w31 + z2 * w32
o=sigmoid(o1)
print(o)
```

Output:

```
0.7184760218226367
```

AIML Lab Experiment – 10

Ques) Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the NN using appropriate data sets.

Code:

```
from math import exp
from random import seed
from random import random

def initialize_network(n_inputs, n_hidden, n_outputs):
	network = list()
	hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in range(n_hidden)]
	network.append(hidden_layer)
	output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in range(n_outputs)]
	network.append(output_layer)
	return network

def activate(weights, inputs):
	activation = weights[-1]
	for i in range(len(weights)-1):
		activation += weights[i] * inputs[i]
	return activation

def transfer(activation):
	return 1.0 / (1.0 + exp(-activation))
```

```
def forward_propagate(network, row):
	inputs = row
	for layer in network:
		new_inputs = []
		for neuron in layer:
			activation = activate(neuron['weights'], inputs)
			neuron['output'] = transfer(activation)
			new_inputs.append(neuron['output'])
		inputs = new_inputs
	return inputs

def transfer_derivative(output):
	return output * (1.0 - output)

def backward_propagate_error(network, expected):
	for i in reversed(range(len(network))):
		layer = network[i]
		errors = list()
		if i != len(network)-1:
			for j in range(len(layer)):
				error = 0.0
				for neuron in network[i + 1]:
					error += (neuron['weights'][j] * neuron['delta'])
				errors.append(error)
		else:
			for j in range(len(layer)):
				neuron = layer[j]
				errors.append(neuron['output'] - expected[j])
		for j in range(len(layer)):
			neuron = layer[j]
			neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])
```

```
def update_weights(network, row, l_rate):
	for i in range(len(network)):
		inputs = row[:-1]
		if i != 0:
			inputs = [neuron['output'] for neuron in network[i - 1]]
		for neuron in network[i]:
			for j in range(len(inputs)):
				neuron['weights'][j] -= l_rate * neuron['delta'] * inputs[j]
			neuron['weights'][-1] -= l_rate * neuron['delta']


def train_network(network, train, l_rate, n_epoch, n_outputs):
	for epoch in range(n_epoch):
		sum_error = 0
		for row in train:
			outputs = forward_propagate(network, row)
			expected = [0 for i in range(n_outputs)]
			expected[row[-1]] = 1
			sum_error += sum([(expected[i]-outputs[i])**2 for i in
range(len(expected))])
			backward_propagate_error(network, expected)
			update_weights(network, row, l_rate)
		print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))


seed(1)
dataset = [[2.7810836,2.550537003,0],
	[1.465489372,2.362125076,0],
	[3.396561688,4.400293529,0],
	[1.38807019,1.850220317,0],
	[3.06407232,3.005305973,0],
	[7.627531214,2.759262235,1],
```

```
	[5.332441248,2.088626775,1],
	[6.922596716,1.77106367,1],
	[8.675418651,-0.242068655,1],
	[7.673756466,3.508563011,1]]
n_inputs = len(dataset[0]) - 1
n_outputs = len(set([row[-1] for row in dataset]))
network = initialize_network(n_inputs, 2, n_outputs)
train_network(network, dataset, 0.5, 20, n_outputs)
for layer in network:
	print(layer)
```

Output:

```
>epoch=0, lrate=0.500, error=6.350
>epoch=1, lrate=0.500, error=5.531
>epoch=2, lrate=0.500, error=5.221
>epoch=3, lrate=0.500, error=4.951
>epoch=4, lrate=0.500, error=4.519
>epoch=5, lrate=0.500, error=4.173
>epoch=6, lrate=0.500, error=3.835
>epoch=7, lrate=0.500, error=3.506
>epoch=8, lrate=0.500, error=3.192
>epoch=9, lrate=0.500, error=2.898
>epoch=10, lrate=0.500, error=2.626
>epoch=11, lrate=0.500, error=2.377
>epoch=12, lrate=0.500, error=2.153
>epoch=13, lrate=0.500, error=1.953
>epoch=14, lrate=0.500, error=1.774
>epoch=15, lrate=0.500, error=1.614
>epoch=16, lrate=0.500, error=1.472
>epoch=17, lrate=0.500, error=1.346
>epoch=18, lrate=0.500, error=1.233
>epoch=19, lrate=0.500, error=1.132
[{'weights': [-1.4688375095432327, 1.850887325439514, 1.0858178629550297], 'output': 0.029980305604426185, 'delta': 0.0059546604162323625}, {'weights': [0.37711098142462157, -0.0
625909894552989, 0.2765123702642716], 'output': 0.9456229000211323, 'delta': -0.0026279652850863837}]
[{'weights': [2.515394649397849, -0.3391927502445985, -0.9671565426390275], 'output': 0.23648794202357587, 'delta': 0.04270059278364587}, {'weights': [-2.5584149848484263, 1.0036
422106209202, 0.42383086467582715], 'output': 0.7790535202438367, 'delta': -0.03803132596437354}]
```