

ANN PROGRAM

CODE

```
import numpy as np
import time
X = np.array([[2, 9], [1, 5], [3, 6]], dtype=float)
y = np.array([[92], [86], [89]], dtype=float)
y = y/100
def sigmoid (x):
    return 1/(1 + np.exp(-x))
epoch=50000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
start_time=time.process_time()
for i in range(epoch):
    #Forward Propagation
    neth1=np.dot(X,wh)+bh
    outh1 = sigmoid(neth1)
    neto1=np.dot(outh1,wout)+bout
    outo1 = sigmoid(neto1)

    #Backpropagation
    deltak=(y-outo1)*(outo1)*(1-outo1)

    deltaj = deltak.dot(wout.T)*(outh1)*(1-outh1)

    wout += outh1.T.dot(deltak) *lr
    #bout += np.sum(deltak, axis=0,keepdims=True) *lr
    wh += X.T.dot(deltaj) *lr
    #bh += np.sum(deltaj, axis=0,keepdims=True) *lr
end_time=time.process_time()-start_time
print(end_time,"seconds")
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,outo1)
```

ANN Program

```
import numpy as np
import time
X = np.array([[2, 3], [1, 5], [3, 6]], dtype = float)
y = np.array([[32], [86], [83]], dtype = float)
y = y/100
```

```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

```
epoch = 50000
```

```
lr = 0.1
```

```
inputlayer_neurons = 2
```

```
hiddenlayer_neurons = 3
```

```
output_neurons = 1
```

```
wh = np.random.uniform(size = (inputlayer_neurons, hiddenlayer_neurons))
```

```
bh = np.random.uniform(size = (1, hiddenlayer_neurons))
```

```
wout = np.random.uniform(size = (hiddenlayer_neurons, output_neurons))
```

```
bout = np.random.uniform(size = (1, output_neurons))
```

```
start_time = time.process_time()
```

```
for i in range(epoch):
```

```
    net_h1 = np.dot(X, wh) + bh
```

```
    out_h1 = sigmoid(net_h1)
```

```
    net_o1 = np.dot(out_h1, wout) + bout
```

```
    out_o1 = sigmoid(net_o1)
```

```
    deltak = (y - out_o1) * (out_o1) * (1 - out_h1)
```

```
    deltaj = deltak.dot(wout.T) * (out_h1) * (1 - out_h1)
```

```
    wout += out_h1.T.dot(deltak) * lr
```

```
    wh += X.T.dot(deltaj) * lr
```

```
end_time = time.process_time() - start_time
```

```

print(end_time, "seconds")
print("Input:\n" + str(X))
print("Actual Output:\n" + str(y))
print("Predicted Output:\n" + str(out01))

```

Output: 2.875 seconds

Input:

[2. 9.]

[1. 5.]

[3. 6.]

Actual Output:

[0.92]

[0.86]

[0.89]

Predicted Output:

[0.8989852]

[0.87688401]

[0.89206533]

OUTPUT

The screenshot displays a Jupyter Notebook interface for a neural network implementation. The notebook is titled "MLL-4 Last Checkpoint: 3 hours ago (autosaved)". The code is organized into several cells:

- Cell 1:** Imports `numpy` as `np` and `time`. It initializes input `X` and output `y` arrays.

```
import numpy as np
import time
X = np.array([[2, 9], [1, 5], [3, 6]], dtype=float)
y = np.array([[92], [86], [89]], dtype=float)
y = y/100
```
- Cell 2:** Defines a `sigmoid` function.

```
def sigmoid(x):
    return 1/(1 + np.exp(-x))
```
- Cell 3:** Sets training parameters.

```
epoch=50000 #Setting training iterations
lr=0.1 #Setting Learning rate
inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
```
- Cell 4:** Initializes weights and biases.

```
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
```
- Cell 5:** Starts the training loop for 50,000 epochs, including forward and backward propagation.

```
start_time=time.process_time()
for i in range(epoch):
    #Forward Propagation
    neth1=np.dot(X,wh)+bh
    outh1 = sigmoid(neth1)
    neto1=np.dot(outh1,wout)+bout
    outo1 = sigmoid(neto1)

    #Backpropagation
    deltak=(y-outo1)*(outo1)*(1-outo1)
    deltax = deltak.dot(wout.T)*(outh1)*(1-outh1)
    wout += outh1.T.dot(deltak) *lr
    #bout += np.sum(deltak, axis=0,keepdims=True) *lr
    wh += X.T.dot(deltaj) *lr
    #bh += np.sum(deltaj, axis=0,keepdims=True) *lr
```
- Cell 6:** Prints the end time, input, actual output, and predicted output.

```
end_time=time.process_time()-start_time
print(end_time,"seconds")
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,outo1)
```

The output of the notebook shows the training time as 2.875 seconds and the predicted output for the input `[[2.9], [1.5], [3.6]]` as `[[0.89898562], [0.87688401], [0.89206593]]`.

Output for 100000 epochs

The screenshot shows a Jupyter Notebook titled "MLL-4b" running on a local host. The notebook contains a single code cell with the following Python code:

```
#Dn += np.sum(delta, axis=0, keepdims=True) * Lr

In [10]: end_time=time.process_time()-start_time
print(end_time,"seconds")
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n",out01)
```

The output of the code cell is as follows:

```
5.359375 seconds
Input:
[[2. 9.]
 [1. 5.]
 [3. 6.]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
[[0.90140242]
 [0.87500281]
 [0.89124675]]
```

The Jupyter Notebook interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations and execution, and a status bar at the bottom showing the Windows taskbar with the search bar and system clock.

Output for iris dataset

The screenshot shows a Jupyter Notebook interface with the following content:

```
In [13]: end_time=time.process_time()-start_time
print(end_time,"seconds")
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n",out1)
```

The output of the code cell is as follows:

```
[0.3]
[0.3]
[0.3]
[0.3]
[0.3]
[0.3]
[0.3]
[0.3]
[0.3]
[0.3]
[0.3]]
Predicted Output:
[[0.02015423]
[0.02029723]
[0.01980895]
[0.01980627]
[0.01979263]
[0.01980788]]
```