BME 6717 Dataset 3 Aritificial Neural Networks

Gilgal Ansah

March 2022

Basic Artificial Neural Network By Hand

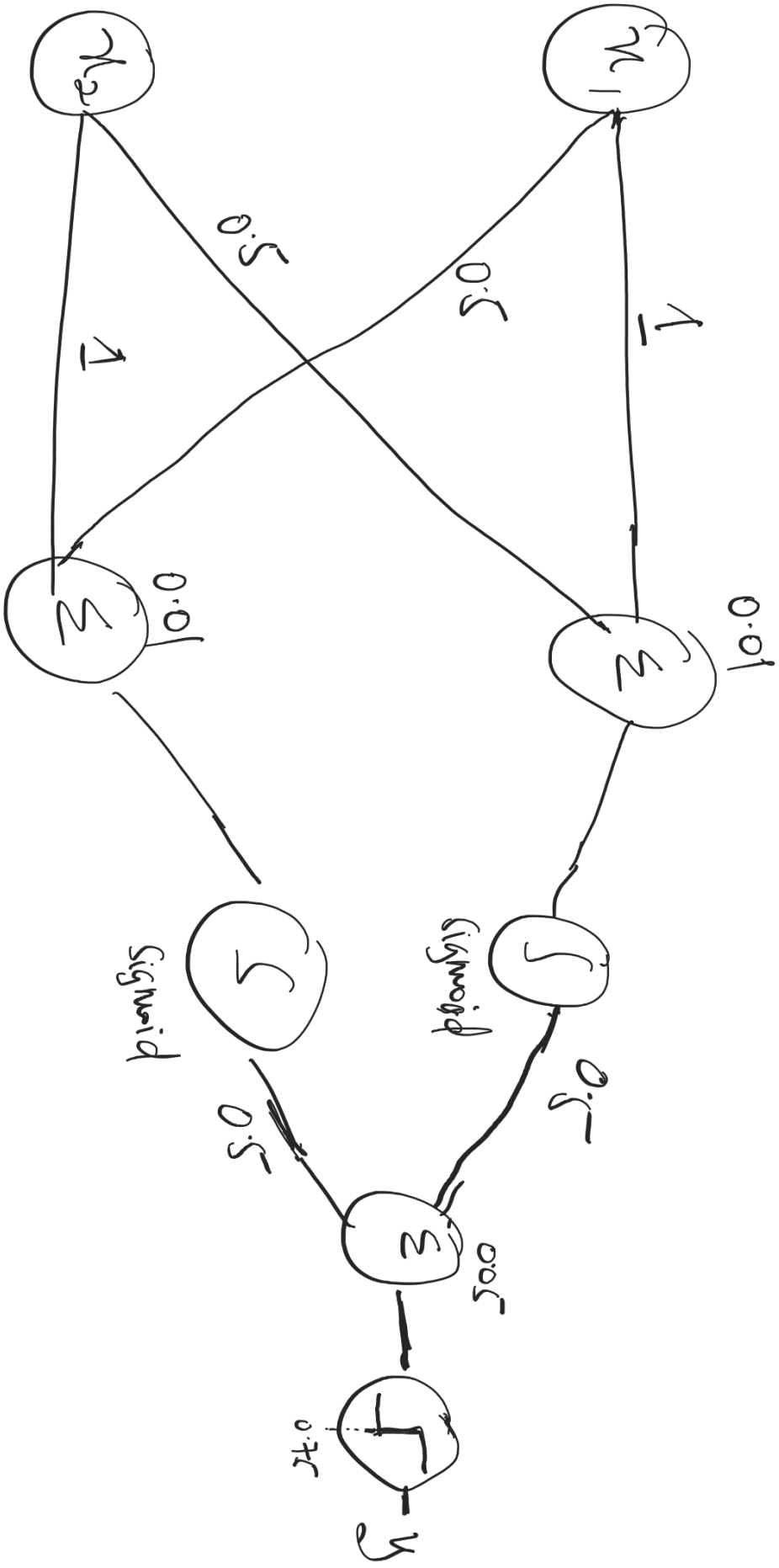
A MATLAB script was written to test the effect of different weights and activation functions on the network. 4 networks were built, one of which had no weights or hidden layers. The others had one hidden layer each.

ANN by Hand

```
function ANN(x1,x2)
%% Basic Artificial Neural Network 'By Hand'
% This program takes a 2-dimensional input and predicts an output based on weights and
% functions

%% 0 layers; no weights; no biases

%summation node
out = x1*x2;

%output activation function
y=out;

%output
% ([0,0] --> [0])
% ([1,0] --> [0])
% ([0,1] --> [0])
% ([1,1] --> [1])
fprintf('OUTPUT1 = %d \n',y)

%% 1 layer (2 nodes); 6 weights; 3 biases

%weights to hidden layer
%wij = input i; node j
w11=1;
w12=1;
w21=1;
w22=1;

%weights from hidden layer
%wj = node j
w1=1;
w2=1;

%biases
b1=1;
b2=1;
b3=0;

%hidden layer computation with sin activation function
h1 = sin(x1*w11 + x2*w21 + b1);
h2 = sin(x1*w12 + x2*w22 + b2);

%output layer with rounding activation function
out = h1*w1 + h2*w2 +b3;
y=round(out);

%output
% ([0,0] --> [2])
% ([1,0] --> [2])
% ([0,1] --> [2])
% ([1,1] --> [0])
fprintf('OUTPUT2 = %d \n',y)

%% 1 layer (2 nodes); 6 weights; 3 biases
```

```
56
57 %weights to hidden layer
58 %wij = input i; node j
59 w11=0.1;
60 w12=0.5;
61 w21=0.1;
62 w22=0.5;
63
64 %weights from hidden layer
65 %wj = node j
66 w1=1;
67 w2=1;
68
69 %biases
70 b1=1;
71 b2=0.1;
72 b3=0.4;
73
74 %hidden layer computation with tanh activation function
75 h1 = tanh(x1*w11 + x2*w21 + b1);
76 h2 = tanh(x1*w12 + x2*w22 + b2);
77
78 %output layer
79 out = h1*w1 * h2*w2 +b3;
80
81 %output layer function
82 if out<1
83     y=0;
84 else
85     y=1;
86 end
87
88 %output
89 % ([0,0] --> [0])
90 % ([1,0] --> [0])
91 % ([0,1] --> [0])
92 % ([1,1] --> [1])
93 fprintf('OUTPUT3 = %d \n',y)
94
95 %% 1 layer (2 nodes); 6 weights; 3 biases
96
97 %weights to hidden layer
98 %wij = input i; node j
99 w11=1;
100 w12=0.5;
101 w21=0.5;
102 w22=1;
103
104 %weights from hidden layer
105 %wj = node j
106 w1=0.5;
107 w2=0.5;
108
109 %biases
110 b1=0.01;
111 b2=0.01;
112 b3=0.05;
113
114 %hidden layer computation with sigmoid activation function
115 h1 = 1/(1+exp(-(x1*w11 + x2*w21 + b1)));
116 h2 = 1/(1+exp(-(x1*w12 + x2*w22 + b2)));
117
118 %output layer
119 out = h1*w1 + h2*w2 + b3;
120
121 %output layer function
122 if out<0.75
123     y=0;
124 else
125     y=1;
126 end
127
128 %output
129 % ([0,0] --> [0])
130 % ([1,0] --> [0])
131 % ([0,1] --> [0])
132 % ([1,1] --> [1])
133 fprintf('OUTPUT4 = %d \n',y)
```

The diagram for the 4th network is shown below.



Discussion

It was observed that a ReLU activation function was best suited for the output layer given the nature of the outputs. In addition, the input weights seemed to influence the nature of the final input fed into the output layer. For instance, the input to the output layer for $1, 1$ and $1, 0$ were comparable when the weights w_{11} and w_{12} were significantly larger. Whereas, that of $1, 1$ and $0, 1$ were comparable when the weight w_{21} and w_{22} were significantly larger.

Pathology Classification

With a model of 3 hidden layers of neurons [1,5,9], I was able to achieve a 90% test accuracy.
The network used only *ReLU* activation functions. This was because the task is a classification task that requires only two outputs. *ReLU* was able to produce one of two outputs whereas the others gave multiple outputs.
I randomized the number of layers and nodes per layer as well as the training and testing data sets continually until I got a good model.
The training data was 80% of all data points and 20% of all observations was used to test the model.
The MATLAB program is shown below.

MATLAB Code

```
%loading data
load('BreastCancerData.mat');

Data= BreastCancerData.tissueMeasures;

Pathology= BreastCancerData.pathology;

%converting pathology to numeric targets
targets= zeros(1,length(Pathology));

for i=1:length(targets)
    if string(Pathology{i})== 'Benign'
        targets(i)=1;
    else
        targets(i)=0;
    end
end


% partitioning data randomly into train and test sets;
[trainInd,testInd,valInd] = dividerand(664,0.8,0.2,0); %train and test indices


%creating a feedforward network with n (1 - 6) layers and a random number
%of nodes in each layer
n=randi(6);
layerNodes = zeros(1,n);
for i=1:n
    layerNodes(i)=randi(20);
end

%using the machine learning and statistical toolbox
Mdl=fitcnet(Data(:,trainInd)',targets(:,trainInd),LayerSizes=layerNodes,Activations="relu");

%predicting targets using ANN
netTest=predict(Mdl,Data(:,testInd)');

%real test targets
trueTest = targets(:,testInd);

% checking test accuracy
testAccuracy = 1 - loss(Mdl,Data(:,testInd)',targets(:,testInd), ...
    "LossFun","classiferror")


confusionchart(trueTest,netTest)
```

The confusion matrix is shown below.

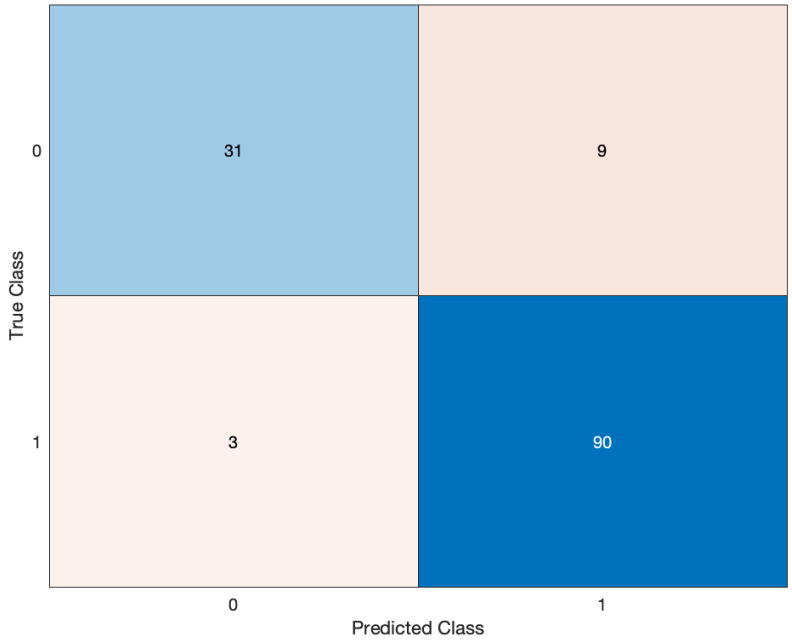


Figure 1: confusion matrix of predictions from network