ESP3201

ANN MINI-ASSIGNMENT

Goh Kheng Xi, Jevan
A0199806L

Contents

1. Introduction

Artificial neural networks (ANNs) are machine learning methods modelled and inspired by the human brain, where the neuron of the ANN performs similarly to the biological neuron [1].

1.1 Aim

This assignment analyses the training and architecture of an ANN. Task A requires the designing of the ANN model and setting of the hyperparameters. Task B studies the mathematics and equations behind the training of the ANN through an optimisation algorithm, while tasks C and D access the training algorithm itself.

2. Task A

The task given is to draw and label an ANN with one input layer, one hidden layer and one output layer, where the number of neurons in the input layer is 2, number of neurons in the hidden layer is 3 and the output layer will have only 1 neuron. The activation function for the hidden neuron is sigmoid function while the activation function for the output layer is any linear function and the input layer neurons have no activation function.

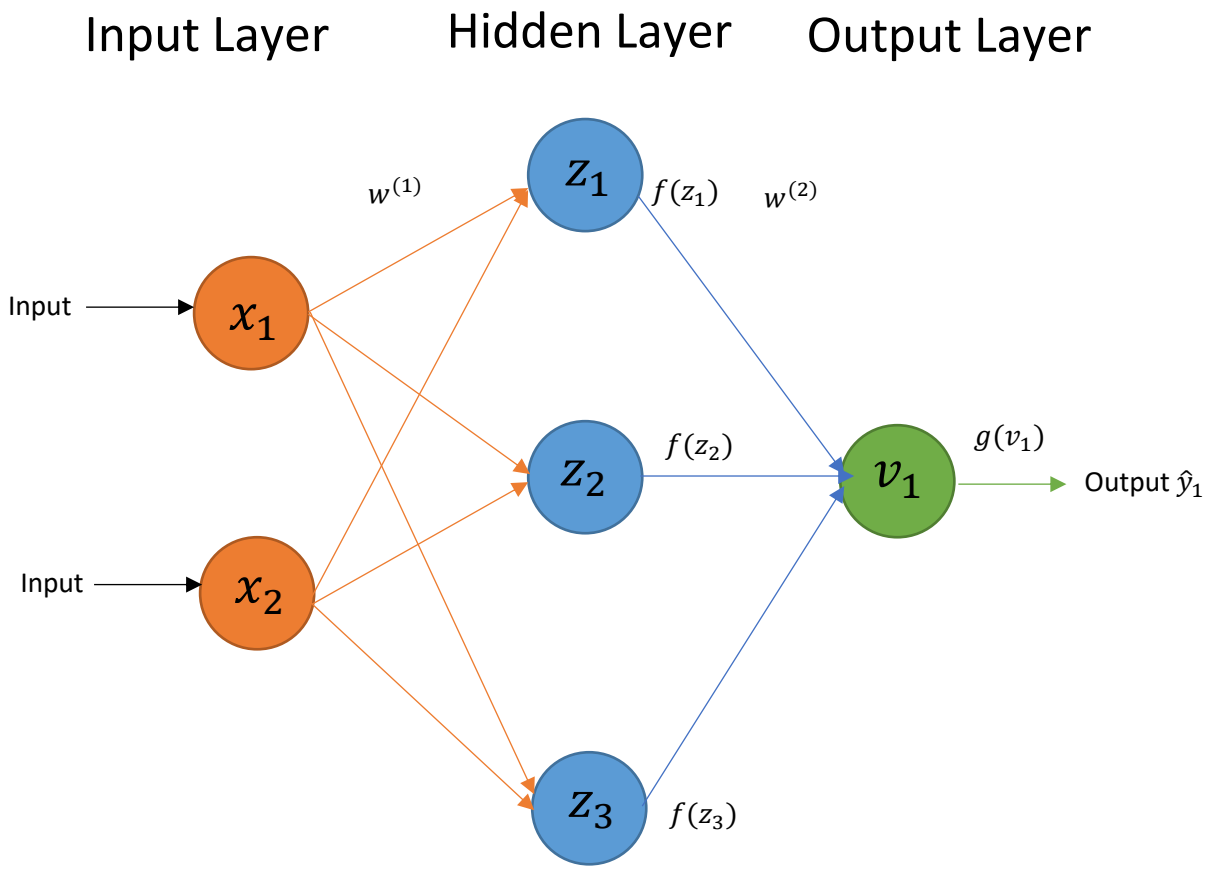

Figure 1. ANN model

$w^{(i)}$: *parameter matrix that contains the weights and biases of the* i^{th} *layer*

x_i: i^{th} *neuron in input layer*

z_i: i^{th} *neuron in hidden layer*

v_1: *neuron in output layer*

$$\hat{y}_1: output\ value$$

$$\text{f(z)}:\ sigmoid\ activation\ function\ for\ the\ neurons\ in\ hidden\ layer, \frac{1}{1+e^{-z}}$$

$$g(y):\quad linear\ activation\ function\ for\ the\ neurons\ in\ output\ layer, Av + B$$

3. Task B

In the training of an ANN, a full update iteration consists of a forward pass and a backward pass [2]. The problem statement given is to list the equations involved in the forward and backward pass.

3.1 Forward pass

A forward pass is performed during the training of an ANN to propagate the input and using the output, calculate the loss function, thus, finding the gradient and step-size of each update. Equations 1 to 4 are the equations involved in the forward pass of this ANN.

$$z_i = w_{i,0}^{(1)} + \sum_{j=1}^{2} x_j w_{i,j}^{(1)} = w_{i,0}^{(1)} + x_1 w_{i,1}^{(1)} + x_2 w_{i,2}^{(1)} \quad \text{(equation 1)}$$

$$f(z) = \frac{1}{1+e^{-z}} \quad \text{(equation 2)}$$

$$v_1 = w_{1,0}^{(2)} + \sum_{j=1}^{3} f(z_j) w_{1,j}^{(2)} = w_{1,0}^{(2)} + f(z_1) w_{1,1}^{(2)} + f(z_2) w_{1,2}^{(2)} + f(z_3) w_{1,3}^{(2)} \quad \text{(equation 3)}$$

$$\hat{y} = g(v) = Av + B \quad \text{(equation 4)}$$

3.2 Back pass

For the back pass, a loss function, J, which is an indicator of the distance (loss) between the model's output and the expected output (ground truth). Mean squared error (MSE) is one of the commonly used loss functions and its equation is given below.

$$J(y - \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \quad \text{(equation 5)}$$

Using equation 5, the loss function for the ANN in Task A is given by:

$$J(y - \hat{y}) = (y_1 - \hat{y}_1)^2 \quad \text{(equation 6)}$$

Next, the gradient ($\frac{dJ}{dw}$) of the loss function(J) with respect to each parameter ($w_{j,k}^{(i)}$), is calculated during the back pass which following partial derivatives (similar to finding a Jacobian matrix) are required:

$$\frac{\partial J}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}} (y - \hat{y})^2$$

$$= -2(y - \hat{y}) \quad \text{(equation 7)}$$

$$\frac{\partial \hat{y}}{\partial v} = \frac{\partial g(v)}{\partial v} = \frac{d}{dv}(Av + B)$$
$$= v \qquad \text{(equation 8)}$$

$$\frac{\partial v}{\partial w_{i,j\neq 0}^{(2)}} = \frac{\partial}{\partial w_{i,j}^{(1)}}\left(w_{i,0}^{(2)} + \sum_{n=1}^{3} f(z_n) \cdot w_{i,n}^{(2)}\right)$$
$$= f(z_j)$$
$$= \frac{1}{1+e^{-z_j}} \qquad \text{(equation 9)}$$

$$\frac{\partial v_1}{\partial w_{i,0}^{(2)}} = \frac{\partial}{\partial w_{i,0}^{(2)}}\left(w_{i,0}^{(2)} + \sum_{n=1}^{3} f(z_n) \cdot w_{i,n}^{(2)}\right) = 1 \qquad \text{(equation 10)}$$

$$\frac{\partial v_1}{\partial f(z_i)} = \frac{\partial}{\partial f(z_i)}\left(w_{1,0}^{(2)} + \sum_{n=1}^{3} f(z_n) \cdot w_{1,n}^{(2)}\right)$$
$$= w_{1,i}^{(1)} \qquad \text{(equation 11)}$$

$$\frac{\partial f(z_i)}{\partial z_i} = \frac{\partial}{\partial z_i}\left(\frac{1}{1+e^{-z_i}}\right)$$
$$= \frac{e^{-z_i}}{(1+e^{-z_i})^2} \qquad \text{(equation 12)}$$

$$\frac{\partial z_i}{\partial w_{i,j\neq 0}^{(1)}} = \frac{\partial}{\partial w_{i,j\neq 0}^{(1)}}\left(w_{i,0}^{(1)} + \sum_{n=1}^{2} x_n w_{i,n}^{(1)}\right)$$
$$= x_j \qquad \text{(equation 13)}$$

$$\frac{\partial z_i}{\partial w_{i,0}^{(1)}} = \frac{\partial}{\partial w_{i,0}^{(1)}}\left(w_{i,0}^{(1)} + \sum_{n=1}^{2} x_n w_{i,n}^{(1)}\right) = 1 \qquad \text{(equation 14)}$$

Using equations 1 to 14, the gradients of the loss function with respect to each parameter of each layer, $w_{j,k}^{(i)}$ can be determined.

Gradient of J(w) with respect to weights in output layer using equation 7, 8, 9:

$$\frac{dJ}{dw_{i,j\neq 0}^{(1)}} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial v} \cdot \frac{\partial v}{\partial w_{i,j\neq 0}^{(1)}}$$

$$= -2(y - \hat{y}) \cdot (v) \cdot \left(\frac{1}{1+e^{-z_j}}\right) = \frac{-2(y-\hat{y})v}{1+e^{-z_j}} \qquad \text{(equation 15)}$$

Gradient of J(w) with respect to bias in output layer using equation 7, 8, 10:

$$\frac{dJ}{dw_{i,0}^{(2)}} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial v_1} \cdot \frac{\partial v_1}{\partial w_{i,j\neq 0}^{(2)}}$$

$$= -2(y - \hat{y}) \cdot (v) \cdot (1) = 2\hat{y}v - 2yv \quad \text{(equation 16)}$$

Gradient of J(w) with respect to weights in hidden layer using equation 7, 8, 11, 12, 13:

$$\frac{dJ}{dw_{i,j\neq 0}^{(1)}} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial v_1} \cdot \frac{\partial v_1}{\partial f(z_i)} \cdot \frac{\partial f(z_i)}{\partial z_i} \cdot \frac{\partial z_i}{\partial w_{i,j\neq 0}^{(1)}}$$

$$= -2(y - \hat{y}) \cdot (v) \cdot \left(w_{1,i}^{(2)}\right) \cdot \left(\frac{e^{-z_i}}{(1+e^{-z_i})^2}\right) \cdot (x_j)$$

$$= -\frac{2(y-\hat{y})\cdot v \cdot w_{1,i}^{(2)} \cdot e^{-z_i} \cdot x_j}{(1+e^{-z_i})^2} \text{ (equation 17)}$$

Gradient of J(w) with respect to biases in hidden layer using equation 7, 8, 11, 12, 14:

$$\frac{dJ}{dw_{i,0}^{(1)}} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial v_1} \cdot \frac{\partial v_1}{\partial f(z_i)} \cdot \frac{\partial f(z_i)}{\partial z_i} \cdot \frac{\partial z_i}{\partial w_{i,0}^{(1)}}$$

$$= -2(y - \hat{y}) \cdot (v) \cdot \left(w_{1,i}^{(2)}\right) \cdot \left(\frac{e^{-z_i}}{(1+e^{-z_i})^2}\right) \cdot (1)$$

$$= -\frac{2(y-\hat{y})\cdot v \cdot e^{-z_i} \cdot w_{1,i}^{(2)}}{(1+e^{-z_i})^2} \quad \text{(equation 18)}$$

4. Task C and D

The training of an ANN is an optimisation algorithm (gradient descent) where the parameters are optimised when gradient of the loss against each parameters reaches a global minimum [2]. To demonstrate this process, the training of the ANN to perform a binary classification is explained. The two classes are 1 and 0 which corresponds to their y values.

Table 1. Training set of data

Data point $(x^{(i)})$	$x_1^{(i)}$	$x_2^{(i)}$	$y^{(i)}$
1	0.98	0.23	0
2	0.72	0.05	0
3	0.84	0.17	0
4	0.63	0.15	0
5	0.88	0.26	0
6	0.14	0.88	1
7	0.32	0.93	1
8	0.37	0.82	1

9	0.28	0.79	1
10	0.09	0.75	1

A training set of normalised data points is generated in table 1 and the learning rate, e, is set as 0.01 and the linear activation function for the neuron in the output layer is g(v) = 2v. However, the A and B values in the linear activation function g(v) = Av + B should not affect the training of the ANN as the gradient is independent of them.

4.1 Gradient descent for $w_{1,1}^{(2)}$

Since the gradient descent is the optimisation of the loss function for all dataset, the chosen cost function (equation 15) for the entire training set is the sum of the loss function for each data point.

$$\frac{1}{n}\sum_{i=1}^{n}\left(y_1^{(i)}-\hat{y}_1^{(i)}\right)^2=\frac{1}{10}\sum_{i=1}^{10}\left(y_1^{(i)}-\hat{y}_1^{(i)}\right)^2 \qquad \text{(equation 19)}$$

$$i: each\ data\ point$$

Firstly, a gradient descent is performed to tune $w_{1,1}^{(2)}$, and table 2 shows the parameters in the 0th update where the values of each parameter are randomly initialised.

Table 2. Initial parameters

Neuron/input	$w_1^{(1)}$	$w_2^{(1)}$	$w_3^{(1)}$	$w_1^{(2)}$
w_0 (bias)	0	0	0	0
w_1	0.6	0.2	0.05	0.5
w_2	0.5	0.3	0.01	0.2
w_3	N.A.	N.A.	N.A.	0.3

1st update:

Table 3. Results of first iteration

Sample	$\hat{y}^{(i)}$	y^i	$y^i-\hat{y}^i$	$\mathcal{L}_1$
1	1.20	0	1.20	0.695
2	1.13	0	1.13	
3	1.17	0	1.17	
4	1.13	0	1.13	
5	1.19	0	1.19	
6	1.15	0	0.15	
7	1.19	1	0.19	
8	1.18	1	0.18	
9	1.16	1	0.16	
10	1.13	1	0.13	

Forward propagation of all the data points, $x^{(i)}$, is performed using the parameters set in table 1 to obtain the output, $\hat{y}$(table 3). Next the cost value, $\mathcal{L}_1 = 6.95$ is calculated as shown in

table 3. Next, back propagation is performed to find the gradient of the cost function with respect to the parameter, $w_{1,1}^{(2)}$, using equation 15.

$$\frac{d\mathcal{L}}{dw_{1,1}^{(2)}} = \frac{1}{10}\sum_{n=1}^{10} -\frac{2\left(y^{(n)} - \hat{y}^{(n)}\right)v^{(n)}}{1 + e^{-z_j^{(n)}}} = 0.499$$

Finally, the parameter, $w_{1,1}^{(1)}$, is updated using the gradient found in the back propagation through the following equation:

$$w_{k+1} = w_k - \frac{d\mathcal{L}}{dw_k} \cdot e \qquad \text{(equation 20)}$$

$$= 0.5 - 0.499 \cdot 0.01 = 0.495$$

The algorithm then runs again with the new updated parameter and terminates via a terminating condition. Such terminating conditions including maximum number of updates, gradient tolerance, and maximum performance gain [3]. The algorithm is coded on python to generate the rest of the w and J values for the remaining updates. The terminating condition used is gradient is within $\pm$ 0.01 of 0 which is the global minimum. The loss function is plotted against the parameter in figure 2 below where the loss function converges to 2.48 as the $w_{1,1}^{(2)}$ converges to 0.000419 and terminates after 108 loops.

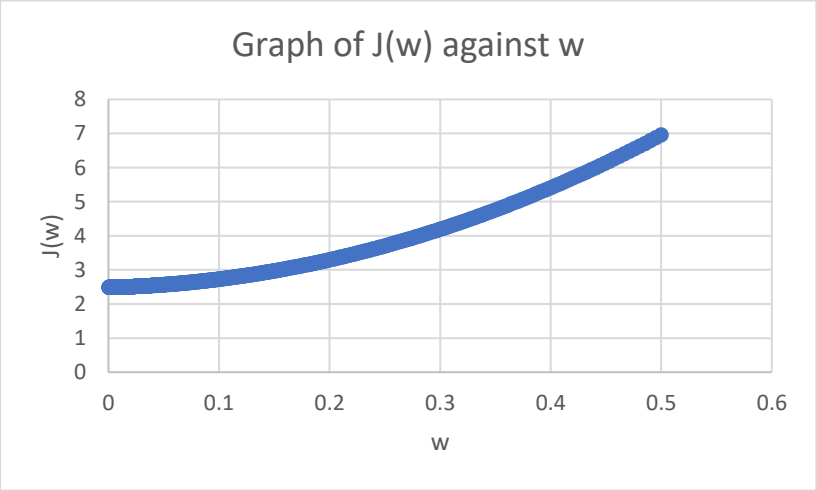

Figure 2. Gradient descent for $w_{1,1}^{(2)}$

4.2 Final ANN model

By performing a similar gradient descent algorithm using the corresponding equations (equation 15, 16, 17, 18) for the rest of the parameters. The optimal values for each of the parameters in the ANN model shown in table 4.

Table 4. Tuned parameters

Neuron/input	$w_1^{(1)}$	$w_2^{(1)}$	$w_3^{(1)}$	$w_1^{(2)}$
w_0 (bias)	3.76	2.76	3.64	-0.324
w_1	5.25	3.19	4.33	0.000419
w_2	6.79	0.3	4.77	-0.364

w_3	N.A.	N.A.	N.A.	-0.321

This is the completion of an epoch where all training samples have undergone an iteration of gradient descent (all parameters converge once), and all the parameters are updated.

5. Conclusion

The ANN model might not be as accurate due to the limited training data available, and the training data only undergo a single epoch of gradient descent. For modelling of the neural network after a more complex function, the number of layers can be increased alongside the number of neurons. Other activation functions such as the rectified linear unit can be explored as the sigmoid function have a vanishing gradient and explosive gradient due to the nature of the function itself. It would be more ideal to choose a normalized activation function at the output layer to normalize the output.

6. References

[1] IBM Cloud Education, "Neural Networks," IBM, 17 August 2020. [Online]. Available: https://www.ibm.com/sg-en/cloud/learn/neural-networks. [Accessed 5 September 2021].

[2] S. Kostadinov, "Understanding Backpropagation Algorithm," Towards Data Science, 8 August 2019. [Online]. Available: https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd. [Accessed 5 September 2021].

[3] N. Jain, "An overview of the Gradient Descent algorithm," Towards Data Science, 26 April 2019. [Online]. Available: https://towardsdatascience.com/an-overview-of-the-gradient-descent-algorithm-8645c9e4de1e. [Accessed 5 September 2021].

7. Appendices

Python code, coded from scratch

```
import math

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

def lin(x):
    return  2 * x


cin = [[0.98, 0.23], [0.72, 0.05],[0.84, 0.17],[0.63, 0.15],[0.88, 0.26],[0.14
, 0.88],[0.32, 0.93],[0.37, 0.82],[0.28, 0.79],[0.09, 0.75]]
loss = []
gradient = [5]
```

```
z = [[0 for i in range(3)] for j in range(10)]
f_z = [[0 for i in range(3)] for j in range(10)]
y = [0 for i in range(10)]
g_y = [0 for i in range(10)]
err = [0 for i in range(10)]
w = [[[0, 0.6, 0.5],[0, 0.2, 0.3],[0, 0.05, 0.01]], [[0, 0.5, 0.2, 0.3]]]
w_curr = []
y_gt = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]

counter = 0

while((gradient[-1]<-0.01 or gradient[-1] < 0.01)):
    w_curr.append(w[0][2][0]) #have to change the w[i][j][k] to the specific p
arameter to train
    print(" \n")

    print(f"========= iteration: {counter} ===========\n")
    for sample in range(10):
        z[sample][0] = w[0][0][1] * (cin[sample][0]) + w[0][0][2] * (cin[sampl
e][1]) + w[0][0][0]
        z[sample][1] = w[0][1][1] * (cin[sample][0]) + w[0][1][2]* (cin[sample
][1])+ w[0][1][0]
        z[sample][2] = w[0][2][1] * (cin[sample][0]) + w[0][2][2] * (cin[sampl
e][1])+ w[0][2][0]


    for sample in range(10):
        f_z[sample][0] = sigmoid(z[sample][0])
        f_z[sample][1] = sigmoid(z[sample][1])
        f_z[sample][2] = sigmoid(z[sample][2])


    for sample in range(10):
        y[sample] = w[1][0][1] * f_z[sample][0] + w[1][0][2] * f_z[sample][1]
+ w[1][0][3] * f_z[sample][2] + w[1][0][0]


    for sample in range(10):
        g_y[sample] = lin(y[sample])



    for i in range(10):
        if (i < 5):
            err[i]=g_y[i]
        else:
            err[i]= g_y[i] - 1

    print(" ")
```

```
    print("============ y_hat ===========")
    print(" ")
    for i in range(10):
        print(g_y[i])


    print(" ")
    print("============ y-y_hat ===========")
    print(" ")
    for i in range(10):
        print(err[i])


    loss.append(0)
    for i in range(10):
        loss[-1] += err[i]**2

    print(f"loss: {loss[-1]}\n" )


    gradient.append(0)
    for i in range(10):
        #have to manually change the gradient function for each neuron
        num = 2*(y_gt[i]-g_y[i]) * y[i] * w[1][0][3]*math.exp(-
z[i][2])#*cin[i][1]#(-2)*(y_gt[i]-g_y[i])*y[i]
        den = (1+math.exp(-z[i][2]))**2
        gradient[-1] += num/den

    gradient[-1] /= 10
    print(f"gradient: {gradient[-1]}\n")

    w[0][2][0] -= gradient[-
1]  * 1 #have to change the w[i][j][k] to the specific parameter to train


    counter += 1

print(" \n\n\n\n\n")
for i in range(len(loss)):
    print(f"{loss[i]}")

print(" \n\n\n\n\n")
for i in range(len(w_curr)):
    print(f"{w_curr[i]}")
```