Iran university of science and technology

October 19, 2024

ANN ASSIGNMENT 1

Presented To

Nafiseh Ahmadi

Presented By

Hessam Koucheki

403 7239 87

Table of Contents

Problem 1

We have these data to start with:

Learning rate (η)	b (weight)	W1	W2	Bias (constant)
0.05	0.1	0.2	-0.1	1

As a result, we don't have to randomly initialize the weights.
Considering the truth table for AND function, we start doing the math.

X1	X2	Bias	Target
1	1	1	**1**
1	0	1	**0**
0	1	1	**0**
0	0	1	**0**

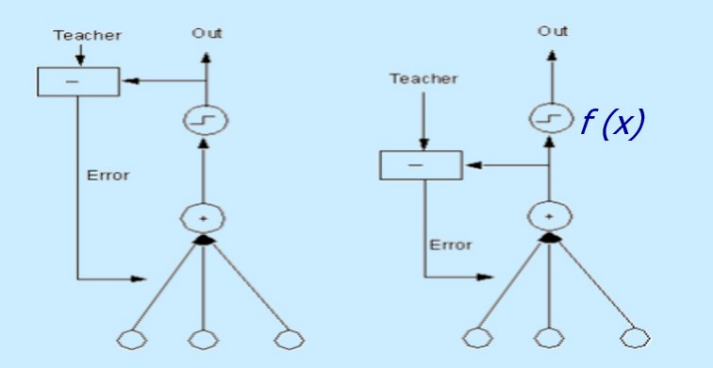
The formula for calculating neuron output is:

$$y = b + \sum x_i w_i$$

And the formulas for updating *b* and weights are:

$$b(new) = b(old) + \eta(d - y)$$
$$w_i(new) = w_i(old) + \eta(d - y)x_i$$

Activation function



While we solve this question using both activation functions, the best choice pf activation function for this question would be a Step function. That's because we want to output the 1 if the answer of the AND function is placed in the second half of the range. I mean, we have a range from -1 to 1 for output; the first half which is -1 to 0 is relatively belongs to -1; and the second half which is 0 to 1 is for output of 1.) Using an activation function such as ReLU will not generate good result to come handy

for calculating the error, and thus, training the network. As a result, we have to choose the Step function which is a Hard Limiter that brings the outputs to the desired results. Moreover, the calculation of the loss and error does not require the function to be derivative since we are using Adaline and calculating the error before applying the activation function.

Applying (1, 1) with output of 1 (epoch 1)

Output: y = 0.1 + 1*0.2 + 1*-0.1 = **0.2**
Relu: 0.2
Step: 1

Note: the activation fuction should be applied after calculating the error, but as dear TAs told us, we apply activation function before loss calculations.

Updating weights:
Relu:

b = 0.1 + 0.05(1-0.2) = **0.14**
w1 = 0.2 + 0.05(1-0.2) = **0.24**
w2 = -0.1 + 0.05(1-0.2) = **-0.06**

Step:

b = 0.1 + 0.05(1-1) = **0.1**
w1 = 0.2 + 0.05(1-1) = **0.2**
w2 = -0.1 + 0.05(1-1) = **-0.1**

So, weights now are:

	b	W1	W2
Relu	0.14	0.24	-0.06
Step	0.1	0.2	-0.1

Applying (1, 0) with output of 0 (epoch 1)

Relu: 0.14 + 0*0.22 + -1*-0.08 = **0.22**
Step: 0.1 + 0*0.2 + -1*-0.1 = 0.2 = **1**

Updating weights:
Relu:

b = 0.14 + 0.05*(0-0.22) = **0.129**
w1 = 0.24 + 0.05*(0-0.22) = **0.229**
w2 = -0.06 + 0.05*(0-0.22) = **-0.071**

Step:

b = 0.1 + 0.05*(0-1) = **0.05**
w1 = 0.2 + 0.05*(0-1) = **0.15**
w2 = -0.1 + 0.05*(0-1) = **-0.15**

So, weights now are:

	b	W1	W2
Relu	0.129	0.229	-0.071
Step	0.05	0.15	-0.15

Applying (0, 1) with output of 0 (epoch 1)

Relu: y = 0.129 + 0*0.229 + 1*-0.071 = **0.058**
Step: y = 0.05 + 0*0.15 + 1*-0.15 = -0.01 = **-1**

Updating weights:
Relu:

b = 0.129 + 0.05*(0-0.058) = **0.1261**
w1 = 0.229 + 0.05*(0-0.058) = **0.2261**
w2 = -0.071 + 0.05*(0-0.058) = -**0.0739**

Step:

b = 0.05 + 0.05*(0--1) = **0.1**
w1 = 0.015 + 0.05*(0--1) = **0.065**
w2 = -0.015 + 0.05*(0--1) = **0.035**

So, weights now are:

	b	W1	W2
Relu	0.1261	0.2261	-0.0739
Step	0.1	0.065	0.035

Applying (0, 0) with output of 0 (epoch 1)

Output:
Relu: y = 0.1261 + 0*0.2261 + 0*-0.0739 = **0.1261**
Step: y = 0.1 + 0*0.065 + 0*0.035 = 0.1 = **1**

Updating weights:
Relu:

b = 0.1261 + 0.05*(0-0.1261) = **0.1197**
w1 = 0.2261 + 0.05*(0-0.1261) = **0.2191**
w2 = -0.0739 + 0.05*(0-0.1261) = **-0.0802**

Step:

b = 0.1 + 0.05*(0-1) = **-0.0802**
w1 = 0.065 + 0.05*(0-1) = **0.015**
w2 = -0.035 + 0.05*(0-1) = **-0.085**

So, weights now are:

	b	W1	W2
Relu	0.1197	0.2191	-0.0802
Step	-0.0802	0.015	-0.085

This is the end of epoch number: one. Let's test the output to check if it is giving correct answers or not.

Table 1 Weights and outputs after epoch 1

AF	X1	W1	X2	W2	B * b	Destination	Output	Out w/ step function
Relu	1	0.2191	1	-0.0802	0.1197	1	0.2586	**0.2586**
Relu	1	0.2191	0	-0.0802	0.1197	0	0.3388	**0.3388**
Relu	0	0.2191	1	-0.0802	0.1197	0	0.0395	**0.0395**
Relu	0	0.2191	0	-0.0802	0.1197	0	0.1197	**0.1197**
Step	1	0.015	1	-0.085	-0.802	1	-0.872	**-1**
Step	1	0.015	0	-0.085	-0.802	0	-0.878	**-1**
Step	0	0.015	1	-0.085	-0.802	0	-0.887	**-1**
Step	0	0.015	0	-0.085	-0.802	0	-0.802	**-1**

Applying (1, 1) with output of 1 (epoch 2)

Relu: 0.1197 + 1*0.2191 + 1*-0.0802 = **0.2586**
Step: -0.0802 + 1*0.015 + 1*-0.085 = -0.15 = **-1**

Updating weights:

Relu:

b = 0.1197 + 0.05*(1-0.2586) = **0.1567**
w1 = 0.2191 + 0.05*(1-0.2586) = **0.2561**
w2 = -0.08 + 0.05*(1-0.2586) = **-0.042**

Step:

b = -0.8 + 0.05*(1--1) = **-0.7**

w1 = 0.015 + 0.05*(1--1) = **0.115**
w2 = -0.085 + 0.05*(1--1) = **0.015**

So, weights now are:

	b	W1	W2
Relu	0.1567	0.2561	-0.042
Step	-0.7	0.115	0.015

Applying (1, 0) with output of 0 (epoch 2)

Relu: 0.1567 + 1*0.2561 + 0*-0.042 = **0.4128**
Step: -0.7 + 1*0.115 + 0*0.15 = -0.585 = **-1**

Updating weights:
Relu:

b = 0.1567 + 0.05*(0-0.4128) = **0.1360**
w1 = 0.2561 + 0.05*(0-0.4128) = **0.2354**
w2 = -0.042 + 0.05*(0-0.4128) = **-0.062**

Step:

b = -0.7 + 0.05*(0--1) = **-0.65**
w1 = 0.115 + 0.05*(0--1) = **0.165**
w2 = 0.015 + 0.05*(0--1) = **0.065**

So, weights now are:

	b	W1	W2
Relu	0.1360	0.2354	-0.062
Step	-0.65	0.165	0.065

Applying (0, 1) with output of 0 (epoch 2)

Relu: 0.1567 + 0*0.2561 + 1*-0.042 = **0.4128**
Step: -0.7 + 0*0.115 + 1*0.15 = -0.585 = **-1**

Updating weights:
Relu:

b = 0.1360 + 0.05*(0-0.4128) = **0.1153**
w1 = 0.2354 + 0.05*(0-0.4128) = **0.2147**
w2 = -0.062 + 0.05*(0-0.4128) = **-0.082**

Step:

b = -0.65 + 0.05*(0--1) = **-0.6**
w1 = 0.165 + 0.05*(0--1) = **0.215**
w2 = 0.065 + 0.05*(0--1) = **0.115**

So, weights now are:

	b	W1	W2
Relu	0.1153	0.2147	-0.082
Step	-0.6	0.215	0.115

Applying (0, 0) with output of 0 (epoch 2)

Relu: 0.1153 + 0*0.2147 + 0*-0.082 = **0.1153**
Step: -0.6 + 0*0.215 + 0*0.115 = -0.6 = **-1**

Updating weights:
Relu:

b = 0.1153 + 0.05*(0-0.1153) = **0.1095**
w1 = 0.2147 + 0.05*(0-0.1153) = **0.2089**
w2 = -0.082 + 0.05*(0-0.1153) = **-0.0877**

Step:

b = -0.6 + 0.05*(0--1) = **-0.55**
w1 = 0.215 + 0.05*(0--1) = **0.265**
w2 = 0.115 + 0.05*(0--1) = **0.165**

So, weights now are:

	b	W1	W2
Relu	0.1095	0.2089	-0.0877
Step	-0.55	0.265	0.165

This is the end of epoch number: Two. Let's test the output to check if it is giving correct answers or not.

Table 2 Weights and outputs after epoch 2

AF	X1	W1	X2	W2	B * b	Destination	Output	Out w/ step function
Relu	1	0.2089	1	-0.0877	0.1095	1	0.2307	**0.2307**
Relu	1	0.2089	0	-0.0877	0.1095	0	0.3184	**0.3184**
Relu	0	0.2089	1	-0.0877	0.1095	0	0.0218	**0.0218**
Relu	0	0.2089	0	-0.0877	0.1095	0	0.1095	**0.1095**

Step	1	0.265	1	0.165	-0.55	1	-0.12	**-1**
Step	1	0.265	0	0.165	-0.55	0	-0.285	**-1**
Step	0	0.265	1	0.165	-0.55	0	-0.385	**-1**
Step	0	0.265	0	0.165	-0.55	0	-0.55	**-1**

Problem 2

Adding neurons

In this question, I guess, we can increase the count of neurons at the first layer if we want to eliminate the second layer. That's because when we are putting away a full layer, it means that we are decreasing the capacity of the network to investigate and find patterns and weights; thus, if we want to compensate for the lost capacity resulted by eliminating the second layer, we must increase the number of neurons at the first layer.

Considering that we had about 1,200 weights at the second layer, we can add the neurons at the first layer by 3 (math: 50 is input of the first layer so new number of neurons is 53, therefore 3 * 728 2,100; which is an increase close to the missing 1,200 weights). However, this gives us less accuracy (about 93%). And seems not working. So, let's see some other techniques. [1]

Reducing the input size (reshape)

Another approach that we can proceed with is to reduce the size of inputs, i.e. reshaping (reducing the dimenstions of) the images. This is pretty helpful as we had lost lots of learning capacity by eliminating some neurons at the second layer. If we resize the image from 28*28 to 14*14, we can reduce 75% of the input size. Training with that model (input 14*14 and output 50), we come up with the accuracy of 93.6%, which is still pretty low. As we extremely reduced the number of parameters, we can add a little to the output and make it 100. With doubling the outputs of the first hidden layer, we still have the half of the parameters at the base model; but, the accuracy is not satisfying yet: 94.9%.

Downsampling

I don't want to waste your time describing what the downsampling is, here. Just knowing that the accuracy of the model with 128 neurons at the first layer and downsampling with the kernel size and stride of 2 and 2, respectively, on original images (28*28). Note that we still have the final 14*14

[1] Disclaimer: some of these ideas were generated with the aid of AI

images after applying downsampling; but here, in contradiction to the image resizing, we lose less data.
While observing the loss during the training process, I noticed that the loss is not staying at the same number at ending epochs, so, I doubled the number of epochs and I am waiting for the result. Ok results are ready now: 96.69% for 10 epochs, downsampling and one hidden layer with the output of 128 neurons. Let's see what will happen if we double the epochs again and reach 20 opochs... Well, The accuracy using 20 epochs and early stopping is still 96.6%.

Early stopping

While looking at loss numbers while training, I understand that the model, sometimes goes above 0.02. Thus, I implemented the mechanism for early stopping. The ZigZag pattern (going up and down for loss) is giving clues that we may can change the learning rate and reducing it.

Dropout, Scheduling Learning Rate, Batch Normalization

Having all of these combined, we can have the accuracy of 97.8%. Which is perfect. Here because we have used the dropout, the number of parameters is halfed.

Observing the wrong prediction of the model and decide based on that

One the of best ways to increase the accuracy while maintaining the low number of parameters, is to plot the predictions and actually "see" why the model is predicting them wrong. In one of my previous works on Persian handwritten digits, we were annoyed by the low accuracy. So, we plot the images with their corresponding labels. We observed that the shapes of the 0 and 5 was very similar in shape and even size! As a result, we started to preprocessing the image using erosion and dilation and thresholding. And the accuracy of model spikes. Here we can do this too.

CNN

I don't know whether we are allowed to use these kinds of networks or not. But based on experience, using CNNs while working with handwritten digits is the best options and commonly gives us the highest accuracy.

While trainging the real model, I reached to these accuracies:
Model: 5 epoch, batch size = 128, learning rate: 0.001, adam optimizer, CrossEntropyLoss.

Base model with two layers: 95.24%

These are my tests. Please note that some of them are just for testing the network and are not necessarily satisfy the problem prerequisites.

# of neurons at first layer	Accuracy by %	Other descriptions
150	96.78	28*28, 5 epochs
300	97.03	28*28, 5 epochs
1000	97.32	28*28, 5 epochs
1250	97.32	Reach the maximum accuracy for that type of network. Adding neurons or opochs will not increase the accuracy
50	93.62	Reshaped to 14*14, 5 epochs
100	94.9	Reshaped to 14*14, 5 epochs
128	95.91	Downsampling to 14*14
128	97.6	10 epochs
128	97.6	20 epochs
128	97.4	Learning rate: ½ of the base
128	**97.8**	28*28, Dropout, Scheduling Learning Rate, Batch Normalization

CNN model after 10 epochs gives **99.24%** accuracy on test data. :)

Problem 3

Let's see the data in a neat form:

x1=0.35	x2=0.9
w13=0.1	w23=0.8
w14=0.4	w24=0.6
w35=0.3	w45=0.9

The formula for calculating neuron output is:

$$y = b + \sum x_i w_i$$

Inputs for the first hidden layer:

Epoch 1

a3 = w13.x1 + w23.x2 + b = *doing the math with calculator* = 1.755
a4 = w13.x1 + w24.x2 + b = math... = 1.68

Using ReLU for simplicity:
H3 = relu(a3) = 1.755
H4 = relu(a4) = 1.68

a5 = w35.a3 + w45.a4 + b = math = 3.0385
y5 = relu(a5) = 3.0385

Based on question, the expected output is 0.5. So, let's calculate the error. I use squared error loss here.
$E = \frac{1}{2} (y5-y)^2$ = mathing = 3.21958

Gradient of y5 = y5 – y = 3.0385 – 0.5 = 2.5385
Gradient of w45 = (∂ (E) / ∂ (y5)) . (∂ (z5) / ∂ (w45)) = 4.2677
Updating weights:

W35 = w35 – η . ∂E / ∂w35 = -4.1528
W45 = w45 - η . ∂E / ∂w45 = -3.3677

Propagating the error for updating the previous weights:
∂E / ∂z3 = -10.537
∂E / ∂z4 = -8.5433
∂E / z13 = -3.688
∂E / z23 = -9.483

Updating:

W13 = w13 - η . ∂E / ∂w13 = 3.788

W23 = w23 - η . ∂E / ∂w23 = 10.393

Gradient of others:

∂E / ∂ w14 = -2.99

∂E / ∂24 = -7.689

Updating them:

W14 = 3.39

W24 = 8.289

Results after first epoch:

W13 = 3.788	w23 = 10.283
W14 = 3.390	w24 = 8.289
W35 = -4.1528	w45 = -3.3677

Epoch 2

a3 = 11.5805

a4 = 9.6466

applying relu:

a3 = relu(a3) = 11.5805

a4 = relu(a4) = 9.6466

final neuron:

a5 = -79.5730

y5 = relu(a5) = 0

calculating the error:

$E = ½ (y5 – y)^2 = 0.125$

Backpropagation and gradient descent:

Since the output of the last layer was 0 and 0 has no derivative, no weights will be updated and all stay the same as before.

Results after second epoch:

W13 = 3.788	w23 = 10.283
W14 = 3.390	w24 = 8.289
W35 = -4.1528	w45 = -3.3677

Problem 4

This question had implementation which already done on the source file. It is amazing that the accuracy is pretty low in comparison to the first question which I had implemented to test my guesses.
Firstly , it is important to say that I changed the root path to be able to run the code on the Kaggle platform. So, if you encounter problems while running that source file, consider reverting that at the very first step.

Model Description

This is the most important part of the model which we actually design the architecture. Here, instead of using seperated variables to define each layer of the network, I use a list (nn.Sequential) to define all layers of the network in one step. As a result of this, I did not need to define the ReLU function like what you had placed at the end of the forward function, and I just placed the activation functions just after the definition of each layer.
The model will have an outcome with a low accuracy because of problems in designing the architecture of the model. That is probably because of the high usage of layers with lots of neurons in them. I guess, the model is somehow overfitting over the data and cannot generalize for unseen data. The model is shown on figure 1.

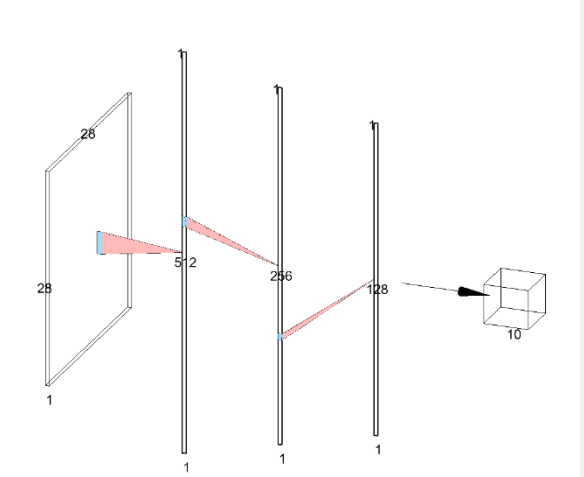


Figure 1 - size of the network

We can clearly see that the count of neurons is not reducing in a perfect manner, thus, making the feature extraction too hard. In other words, jumping from 128 neurons to 10 at the last layer is not commonly suggested. We could also use Dropout to reduce the number of weights.

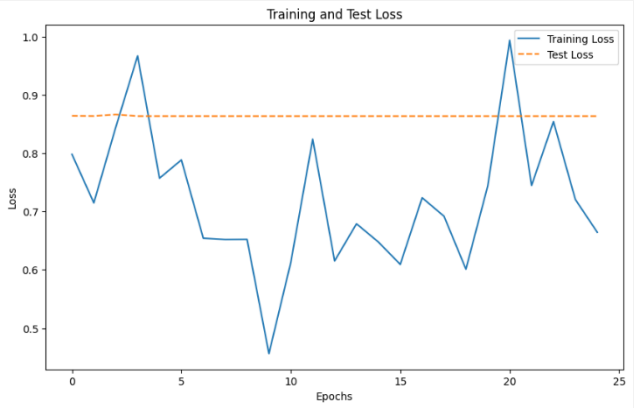


Figure 2 - Train and Test loss plot with lots of fluctuations

We used CrossEntropyLoss for criterion, learning rate of 0.001 which was pretty high. As we see on the loss plot at figure 2, the model is struggling to find a minimum point for loss and constantly fluctuate over the numbers. Reducing the learning rate or using other techniques such as Scheduling Learning Rate to gradually reduce the learning rate can be benficial to avoid this problem. The optimizer was set to Adam. The number of 25 was chosen for count of epochs.

Here, we can see the accuracy changed over the 25 epochs. We cannot see a gradual increase in this plot, which means the amount of Learning Rate is probably have not been chosen properly.

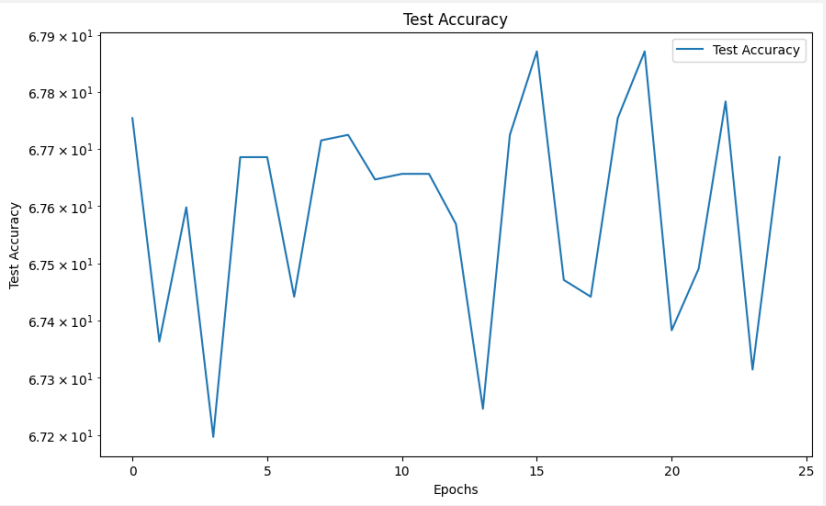


Figure 3 - Test Accuracy

Total Words in this document: 3022
Total editing time: 1965 Minutes (32.75 Hours)
Number of revisions: 169

With thanks and respect,
Hesam Koucheki
403 7239 87