CSCE 478 Machine Learning Homework 2

Jakob Snyder(jsnyde) and Clayton Henderson(chenders)

October 17, 2016

1 Problem

1.1 What is the 90% two-sided confidence interval for the true error rate?

$error_v(h) = \frac{errors}{examples} = \frac{10}{65} = 0.1538$

$errorf_D(h) = error_v(h) \pm 1.64\sqrt{\frac{error_v(h)(1-error_v(h))}{65}} = error_v(h) \pm 0.0734 = [0.0805, 0.2272]$

1.2 What is the 95% one-sided interval?

$error_D(h) = error_v(h) \pm 1.64\sqrt{\frac{error_v(h)(1-error_v(h))}{65}} = error_v(h) \pm 0.0734 = [0.0805, 0.2272]$

1.3 What is the 90% one-sided interval?

$error_D(h) = error_v(h) \pm 1.28\sqrt{\frac{error_v(h)(1-error_v(h))}{65}} = error_v(h) \pm 0.0573 = [0.0966, 0.2111]$

2 Problem

2.1 A two-layer feedforward ANN and the effects of backpropagation on the network weights

Table 1: My caption

	Trial 1	Trial 2	Trial 3	Trial 4	Final Weights
W_ca	.2	.202	.202	.2043	.2043
W_cb	.2	.2	.1985	.1985	.1947
Wc0	.2	.2024	.1984	.2007	.1970
Wdc	.2	.2306	.1876	.2189	.1762
Wd0	.2	.2512	.1794	.2317	.1604

Table 2: A table of the weight values before each trial and the final values calculated using the equations below.

$$\delta_k^t \leftarrow y_k^t\,(1-y_k^t)\,(r_k^t - y_k^t) \qquad \delta_h^t \leftarrow y_h^t\,(1-y_h^t) \sum w_{k,h}^t\,\delta_k^t \qquad w_{j,i}^t \leftarrow w_{j,i}^t + \Delta w_{j,i}^t$$

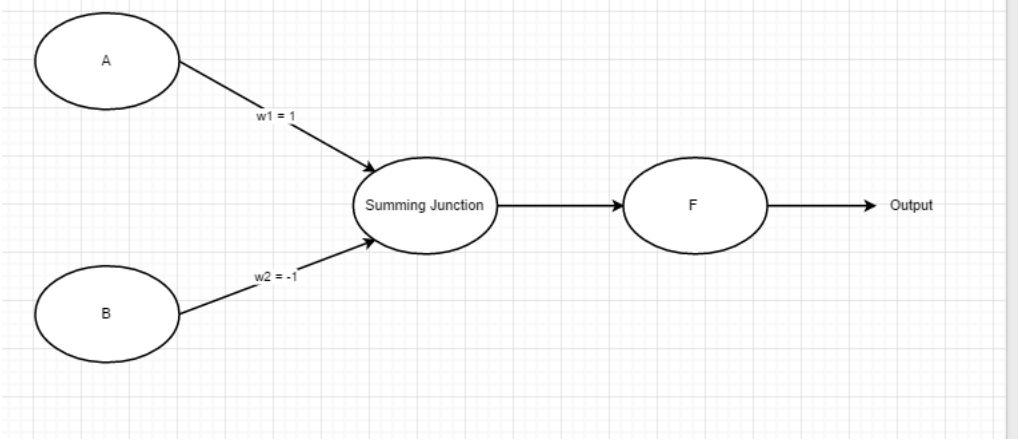
3 Problem

3.1 Design a single-layer, two-input perceptron that implements the boolean function A $\wedge$ [$\sim$ B], where $\sim$ is logical negation.

A	B	F
0	0	0
0	1	0
1	0	1
1	1	0

Table 3: Truth table of A $\wedge$ [$\sim$B]

Figure 1: A diagram of the network, A and B being the inputs.



3.2 Design a multi-layer network of perceptrons to implement [A $\oplus$ B] $\oplus$ C, where $\oplus$ represents exclusive OR.

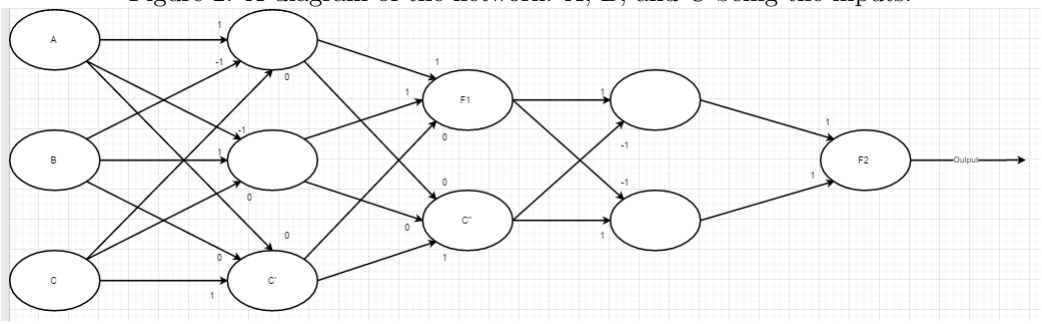
A	B	F1
0	0	0
0	1	1
1	0	1
1	1	0

Table 4: Truth table for A $\oplus$ B that results in F1

C	F1	F2
0	0	0
0	1	1
1	0	1
1	1	0

Table 5: Truth table for C $\oplus$ F1 that results in F2

Figure 2: A diagram of the network. A, B, and C being the inputs.



Here our strategy was to first compute A $\oplus$ B and have that be F1 and then to compute F1 $\oplus$ C and have that be F2. F2 would then be the final outcome that is wanted. This made the network much easier for us to understand at a high level.

4 Problem

4.1 Introduction

As we approached the task of implementing an artificial neural network and the backpropogation algorithm, it became clear that we would not only be challenged by the calculations to be completed, but also conceptualizing how our network would be formed and modified. The data sets used to train and test our implementation were: vote records, monks, and tic-tac-toe. We were initally confronted with an issue of data input into the network, but it was resolved using a method which will be covered in the implementation section. After comparing our implementation with an established ID3 algorithm, we found moderate performance advantages in using one algorithm over another depending upon the input data.

4.2 Implementation

The implementation of our ANN is based around the main objects we created to build our program: Neuron and Connection. Once we parsed in the data, the first objective was to create the skeleton of the network. We created the appropriate neurons for the input and hidden layers, and then procedurally created the connections that would be used to demonstrate the relationship between neurons. We were able to overcome our issues with our input data by creating seperate methods that would taylor the input neurons to the exact specifications required by the data set. While this does reduce the flexibility of our system, it ensures that our network is technically correct.

Once the network structure was complete, the training procedure could begin. This starts by first summing the input values multiplied by the connection weights for each neuron in the hiddlen layer. The summed value is then squashed using the sigmoid function and then passed along to the output neuron. The output neuron takes in values from each hidden neuron multiplied by the connection weights. Once again this resultant value is then squashed by the sigmoid function and we are left with the prediction value.

During this training process, we then used the prediction compared to the label of the training data in the backpropogation algorithm. This uses a variety of equations to modify the weights of the connections between neurons in order to make more accurate predictions.

4.3 Evaluation

We used the test data to evaluate our ANN implementaion. As mentioned before these data points were different every time we ran the program. To evaluate our algorithm we first trained the network. We then passed in the testing data one trial at a time. With each trial we kept track how far off the Sigmoid output was from the actual output.

Trial	Monks		Voting		Tic-Tac-Toe	
	Expected	Sigmoid	Expected	Sigmoid	Expected	Sigmoid
1	1	0.149345063217999	1	0.999998936779025	1	0.997862380653811
2	1	0.149340787014471	1	0.999999999999688	1	0.999887093839234
3	1	0.149340787014471	0	2.00359138551682E-07	1	0.999986404346442
4	0	0.149342019825897	0	2.19947982902309E-07	1	0.999995057553887
5	0	0.149340281655491	1	0.999999999999002	1	0.999993424571596
6	0	0.149336544906309	0	1.71856086104738E-07	1	0.999935069314365
7	0	0.156425501797286	1	0.999999999999002	1	0.999506367844569
8	0	0.149328888386015	0	3.05755211435162E-06	1	0.999874557880533
9	1	0.941570394550069	1	0.999999999998747	1	0.99025820185243
10	1	0.94499916328074	1	0.999999999997046	1	0.999678579853542
11	1	0.943443666763833	1	0.999999999999503	1	0.997736703539314
12	0	0.940374842437009	1	0.999999999991098	1	0.997932069278966
13	1	0.944876446019113	1	0.999999999977616	1	0.934576798245452
14	0	0.149326792921308	0	2.9921587968908E-07	1	0.999922290911806
15	0	0.149326792921308	0	1.62512850190188E-07	1	0.999998265383104
16	1	0.149324291445989	0	0.00343444000404882	1	0.999980621598179
17	0	0.155869844960005	1	0.999999993902445	1	0.999957670378707
18	0	0.155735360184005	1	0.996161327564584	0	0.00133611458688046
19	1	0.999999999929289	0	1.92005334913505E-07	0	0.0202347781923686
20	1	0.999999999927919	0	4.21896443536788E-06	0	0.000753705880621152
21	0	0.999999999926479	0	2.12988243098219E-07	0	0.00165669757956504
22	1	0.999999999924848	0	3.45070185954638E-07	0	0.00131265252342164
23	0	0.999999999924425	0	1.72469673988257E-07	0	0.000320323993263431
24	0	0.161183090026519	0	0.0800493135087444	0	1.52211269474586E-05
25	1	0.999999999972207	0	2.06228522040997E-07	0	8.24539973309907E-05
26	1	0.962253703701919	0	0.000102140735696355	0	0.00123392593375658
27	1	0.999999999972204	0	2.14012813974006E-07	0	3.88015106527371E-05
28	1	0.9603153774969	0	3.70677156588307E-06	0	5.06802505729303E-05
29	1	0.950442378859134	1	0.999966887232087	0	0.000793764263859285
30	1	0.999999999972214	0	1.64508463145999E-07	0	0.999999731411171

Table 6: Expected vs Actual outputs for all 3 data sets when using an ANN

Looking at the Table 6 you can see that while using this implementation Monks was not as accurate as the others. In this particular run Monks had 7 errors in classification while Voting had 1 and Tic-Tac-Toe had 0. With our decision tree we averaged 1 error on Monks, 7 with voting, and 0 on Tic-Tac-Toe. This leads us to conclude that for Monks a decision tree is more accurate. For Voting an ANN is more accurate. Lastly for Tic-Tac-Toe it is a toss up. We would say that the difference between using ID3 vs ANN for Monks has medium significance. We would say the same for Voting when considering ANN vs ID3. Lastly for Tic-Tac-Toe we would say there is little to no significance in the difference between the two algorithms.

The problems that we had with the voting data set were solved with an ANN. These problems rooted from having unknown values in the data set. With an ANN it was much easier to handle these values because the way we handled non-binary inputs. We also understood the problem of having "unknown" values better on this homework so that helped us tremendously while completing the algorithm.

While evaluating the differences between the two algorithms we ran into some problems while trying to use the tools we went over in class. We could not figure out how to correctly use them so we just compared them by looking at the average error percentage between the two.

We had some other output that we found to be useful while debugging our algorithm and we think that could be helpful while understanding how our code works. Here is an example of one output we added.

```
max passes
HiddenLayer Weights
w(0, 0) = -0.154142831431573 w(0, 1) = 4.59677003146844
w(0, 2) = -0.0515749393561321 w(0, 3) = 4.14385047943444
w(0, 4) = 0.00635306461924129 w(0, 5) = 0.0140560791051035
w(0, 6) = 0.0175565745578505 w(0, 7) = -8.84873138727427
w(0, 8) = -8.88300060652141 w(0, 9) = -0.0221341799989795
w(0, 10) = -1.3500080334387
w(1, 0) = -0.154142831431573 w(1, 1) = 4.59677003146844
w(1, 2) = -0.0515749393561321 w(1, 3) = 4.14385047943444
w(1, 4) = 0.00635306461924129 w(1, 5) = 0.0140560791051035
w(1, 6) = 0.0175565745578505 w(1, 7) = -8.84873138727427
```

w(1, 8) = −8.88300060652141 w(1, 9) = −0.0221341799989795

w(1, 10) = −1.3500080334387

OutputLayer Weights

w(2, 0) = 13.0243123174791 w(2, 1) = 13.0243123174791 w(2, 2) = −1.75022818155256 Errors =

The first line is the end condition that was hit for the training set. Next it shows the final weights of the hidden layer. This has the format of w(hidden layer neuron, input layer neuron) = value of weight. next is the output layer weights. This has a very similar output being w(output layer neuron, hidden layer neuron/bias). It then prints the number of errors when using the testing data set.

4.4 Conclusion

Our implementation of an ANN and the backprop algorithm produced generally accurate classifications. At one point we had the error rate on the Monks data set a little bit lower but we forgot to record the eta value we were using so we lost it. We had a few key struggles while implementing the ANN. The first one was again trying to implement multiclass outputs. This is something we could not achieve while implementing the ID3 algorithm and so instead of trying to implement it again we decided to change to a different data set after approval from Bahar. We decided to change to the Tic-Tac-Toe data set mostly because it was similar to our other data sets and it also had a binary output. Another key struggle that we faced was correctly calculating the δW Values. We overcame that struggle by asking Bahar a lot of questions to understand exactly what was being done on the example in the lecture slides. We left the implementation for the Poker data set included in the code and in order to run it just change the program to use dataset 4. This should just include a few line changes and the program should do the rest automatically.

4.5 References

Wikipedia: `https://en.wikipedia.org/wiki/Artificial_neuron`

Used to get a better general understanding of ANNs.

Wikipedia: `https://en.wikipedia.org/wiki/Backpropagation`

Used to get a better understanding of the Backprop Algorithm.