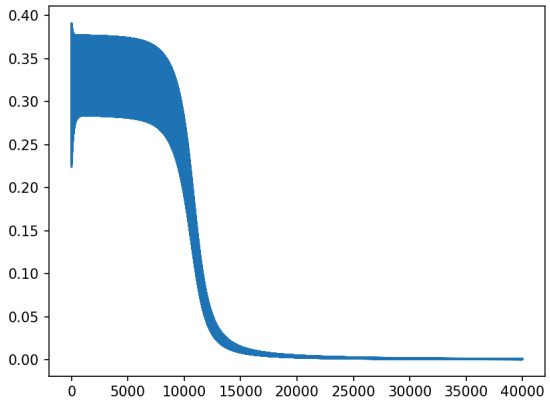
FINAL PROJECT
Michael Rios
Machine Learning
Professor Leeds

Introduction:
We are using the SPECTF Heart Data Set data set provided by University of California, Irvine. It contains 44 features taken from Single Proton Emission Computed Tomography (SPECTF) images that are used to diagnose a heart as normal or abnormal. We have a training set of 80 rows, half normal and half abnormal. I assume some of the abnormal training ones are synthetically created, but I am not sure. Then a testing set of 187 rows, 172 normal and 15 abnormal.

I built what is hopefully a functioning neural network. This will be discussed at greater length in the methods section, but I seem to have gotten decent results so it is probably working correctly. My first attempts only involved adding new neurons, either by increasing the number of layers or by making each layer longer. This did not do much. Then I tried adjusting the learning rate (epsilon) . I use epsilon and learning rate interchangeably in this paper. This did show considerable improvement. Finally I added the neural suppression, which again seemed to improve the results . In this case only weights were suppressed not an entire neuron, still it points to the power of fewer, but presumably correct, connections.

Methods:
I have written a program that seems to be a functioning neural network. I hand checked one round of calculations, forward then back propagation, using a simple data set for a single hidden layer and then two hidden layers. The calculations were correct both times, after first getting many errors.

I then used the program to come up with the correct weights to separate out classes based on the OR truth table



We see that the mean squared error decreases to almost nothing so that is good. In this case we had 10,000 epochs each looping through the four possibilities of an OR truth table. An error was recorded at each, hence the 40,000 on the x-axis. Also for error is cumulative over each four sets of the OR truth table. So it's the error of the first guess, then the error of the first and second guesses, etc. and only gets reset to zero at each new epoch hence the thick line. You will see this same pattern later in the results section. After that I was not entirely sure what to do in terms of testing the code so I stopped, and hence my slight hesitancy to say that my code is fully functioning.

My program is designed so that a user can decide how many layers they want, how many neurons in each layer. I am pretty proud that I was able to do this, though when it came to SPECTF Heart Data Set, adding more neurons seems to have done little to improve the outcome.

Using a small learning rate and many epochs was much more effective and the neural suppression only improved it. So it looks like I could have just built a single hidden layer and spent more of my time on cleaning up the code, but such is life. Also the code only works for a single output neuron, I don't think this would be hard to change so that more output neurons could be added but at the moment that is not how it is written.

With more time I would continue to refine the code but it seems to work.

Right now there is one main function: **neural_net**. It takes in the number of epochs you want to run (integer), the learning rate (epsilon) to use (real number), your data divided into features and classes, the number of hidden layers and size of each hidden layer to use (as an np array of the size you want). The final layer does need to have only one neuron so only two classes can be dealt with. I feel confident I could change this but I have not changed it so that is where it stands at the moment. The threshold at which to suppress a connection (supress_threshold any real number) and the number of epoch at which to check for suppression (k_iter, an integer)

There are also two helper functions: one to calculate the sigmoid and one for pruning. The sig (sigmoid) function is literally just the sigmoid equation. While the pruning function takes in our weights and a matrix of equal size to suppress the weights. The weight suppression matrix starts out as all ones and as a weight falls below the suppression threshold, the corresponding indice is turned to 0. We can then just multiply our weight matrix by the suppression matrix to make sure the suppressed weight stays suppressed.

The user can just go to the section marked **user input** to adjust the parameters and hyperparameters, to **data input** adjust the training data, and the **checking test data** for changing the test data. After that nothing needs to be changed.

The code started off much stiffer. At first, both the forward and backward propagation had every layer hard coded, now for the most part they just go through a loop which allows users to add on

as many layers as they want. I still have the very top layer of the backward propagation outside of the main backward propagation loop. I am sure this could be fixed but I doubt I will have time to do it.

The weights are stored in numpy arrays of size length of layer above times the length of layer below, so for an input of two features going to two neurons we have a 2 by 2 array, and so on and so forth, these arrays are all stuffed into one bigger array. I do not know if this is the best way to do this and my code seems like it could be condensed but it seems to work, and I am loath to change it at the moment. It also allowed me to for the most part just use matrix multiplication and avoid for loops.

The biases have their own array of weights though I am sure they could be included into the regular weights. I know they can with forward propagation. I was worried that it might create havoc with the back propagation and so kept them separate, but I am pretty sure that both could be placed together.

Reading in the data is done with a single line data = np.genfromtxt("final_project/SPECTF.train", delimiter=",")

This data set has the classes in the first (0) column, so that it is separated out and placed in its own np array called classes. It is then removed from the rest of the data, so we are left with only the 44 features, this np array is called features. If someone wanted to use a different data set they would need to modify this slightly.

I am using the Single Proton Emission Computed Tomography (SPECT) heart images. It has 44 features and categorizes folks into normal and abnormal. I read the paper this data set was taken from “Knowledge discovery approach to automated cardiac SPECT diagnosis”. This is not my area of expertise so I may have misunderstood but it sounds like they literally take the images produced by the SPECT and then somehow extract the 44 features from that, which is then used to determine if blood flow through the heart is normal or abnormal.

Results:

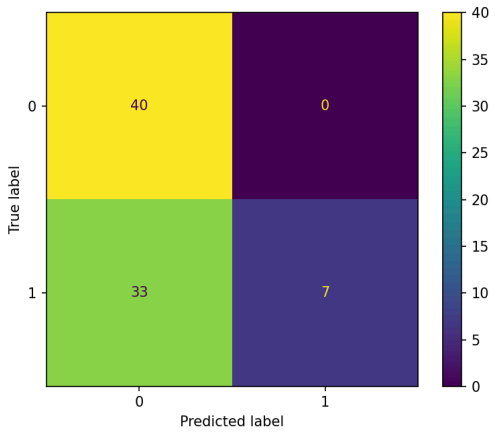
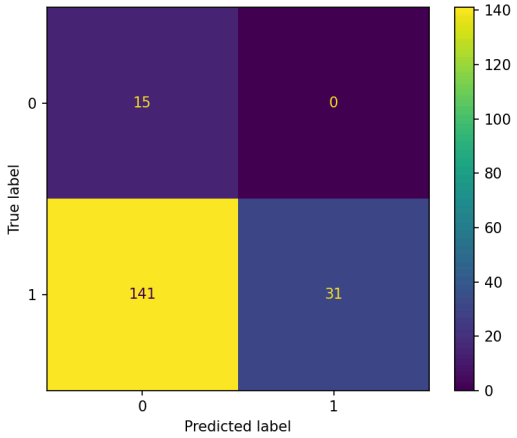
My results were not stunning, especially when all I was doing was adding more neurons. Either with more layers or fatter layers, I feared that there was a mistake in my code and there certainly could be, but I seem to have had more luck with using a smaller learning rate and more epochs. I then had even more luck by suppressing weights so perhaps the code is fine and in this case less neurons is more knowledge.

We are given a set of 80 training data, 40 normal and 40 abnormal. Then 187 test data rows that are heavily skewed towards normal, which makes sense as something that is abnormal cannot be common.

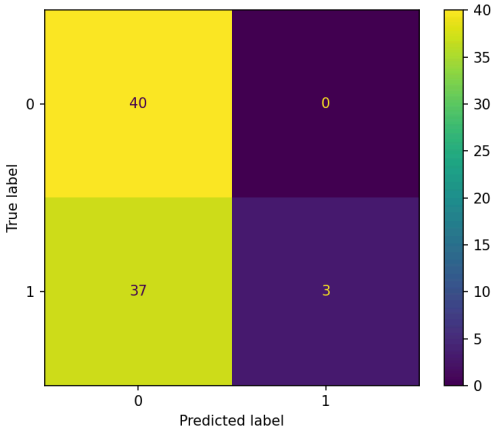
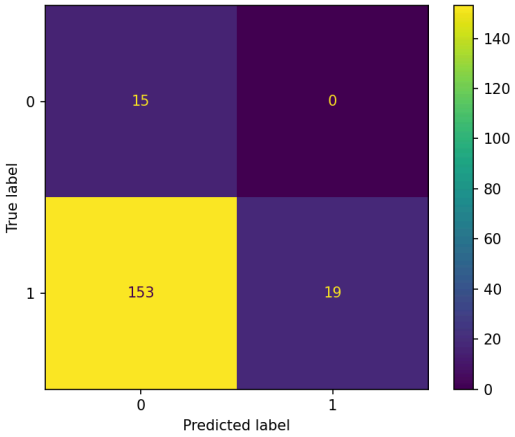
Here I use my 5 hidden layers each with 44 neurons, the length of one row of data, going to a single output, with epsilon of .1 and 5,000 epochs, and all 80 of the training set. It ends up mostly just guessing that everyone is abnormal. I will continually take away layers and not much will change.

Note abnormal is represented by 0 and normal by 1 these are just a bunch of confusion matrices.

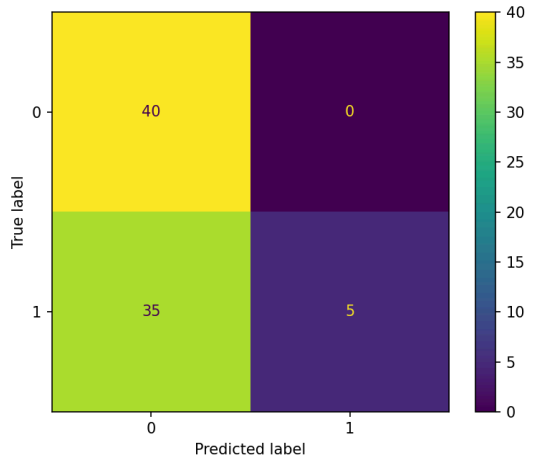
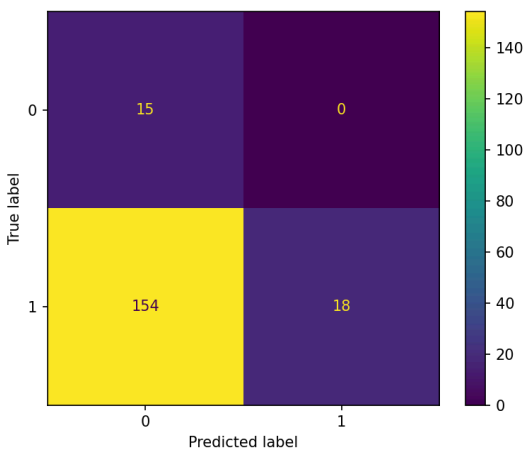
Approximately 3 minutes of run time to train

Training	Test
	

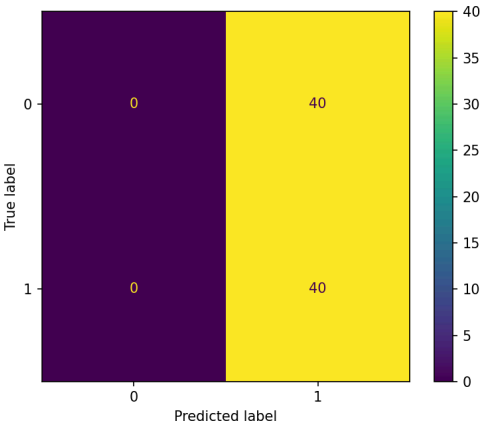
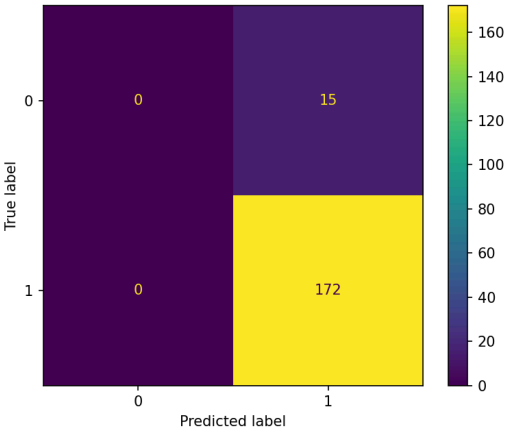
If we only use 4 hidden layers we seem to do slightly worse. Still approximately 3 minutes to train.

Training	Test
	

Three hidden layers, approximately 2 minutes to train. Not much has changed

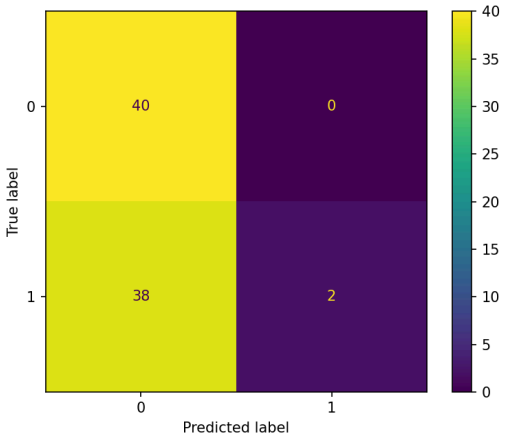
Training	Test
	

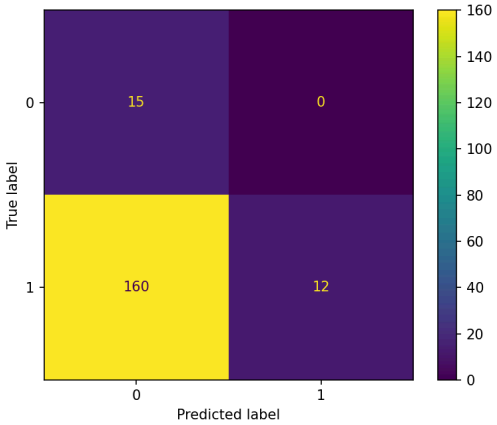
Two hidden layers, just guess everyone is normal which for the test data might look fairly accurate, correct 172 out of 187 times, but as the point of the test is to catch abnormalities is quite bad. Approximate run time 2 minutes to train.

Training	Test
	

One hidden layer, better than 2 for some reason but essentially the same just guess one class all the time, run time about a minute

Train	Test

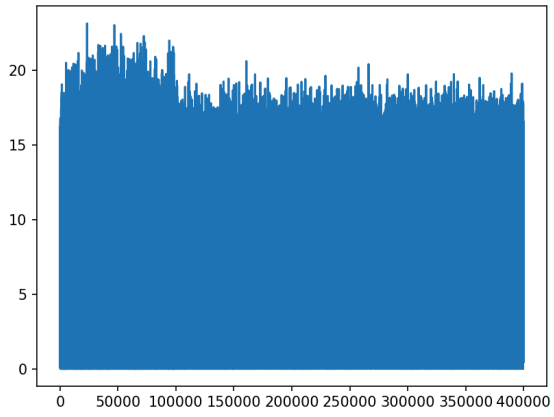


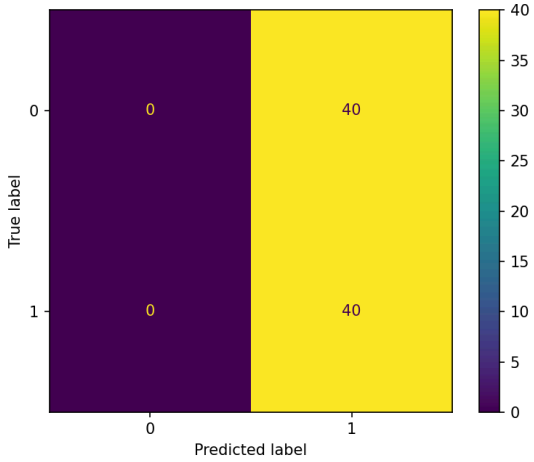


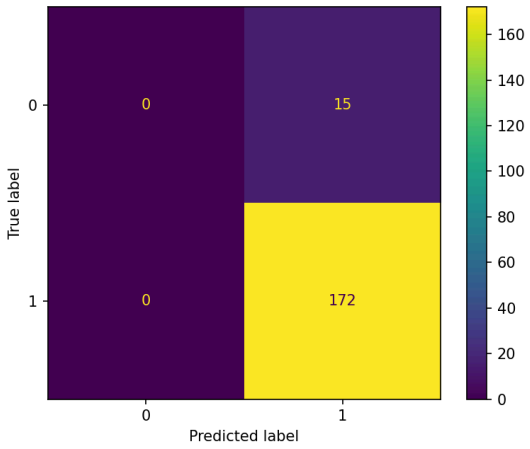
After that I tried with just one layer but kept making that layer have more neurons, it also did not work that well.

Finally I started changing the epsilon
Three layers all feature size, 44 neurons, learning rate 1 epochs 5000 all data. It that does badly, approx 3 minutes

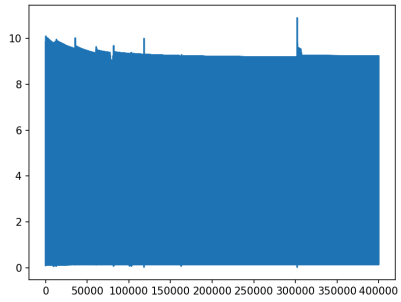
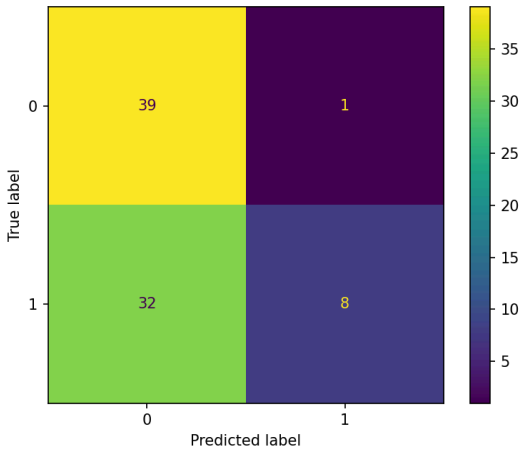
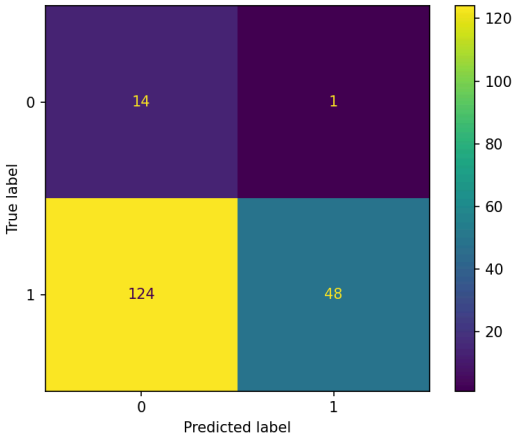
We now also see a graph of our errors

train	Test
	

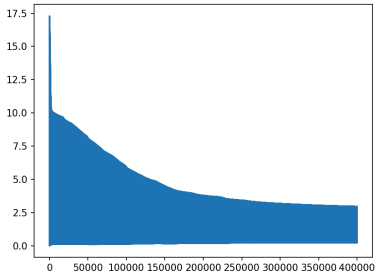
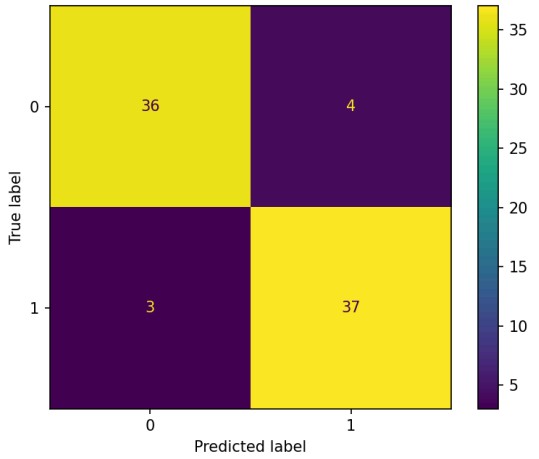
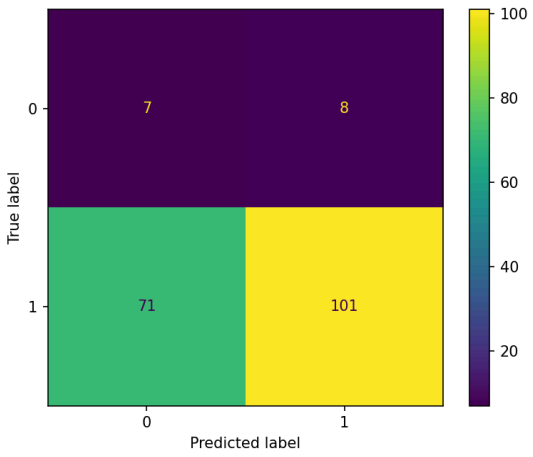




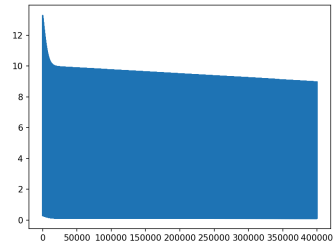
Everything else stayed the same but now the epsilon is .01. It's still mostly guess one class in this case abnormal but doing better than all previous results. Approximate run time, 2 minutes

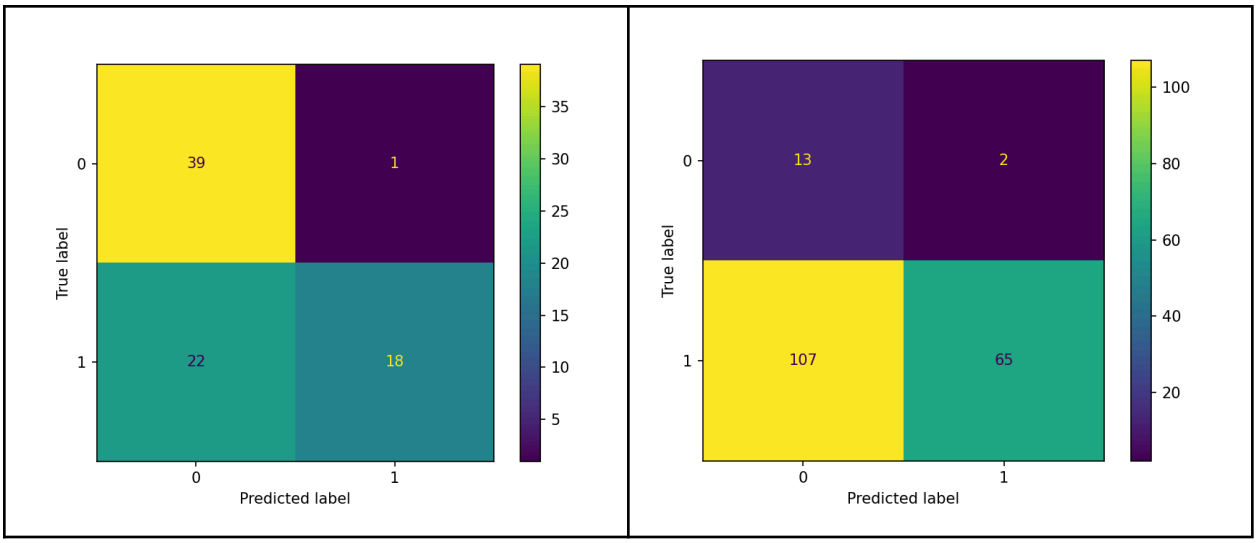
Train	Test
	
	

Three layers epsilon .001. It's kind of just splitting the testing data in two, i.e. half right for abnormal and normal but still far better than previous results yet. Approximate run time, 2 minutes.

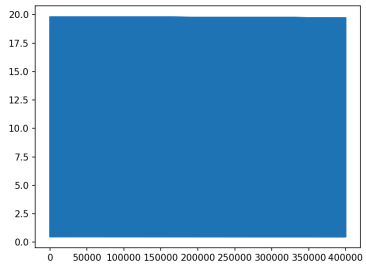
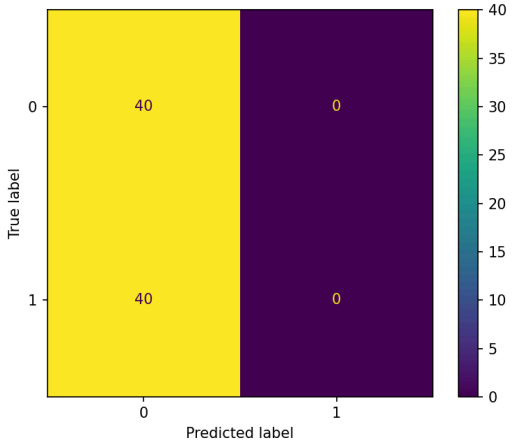
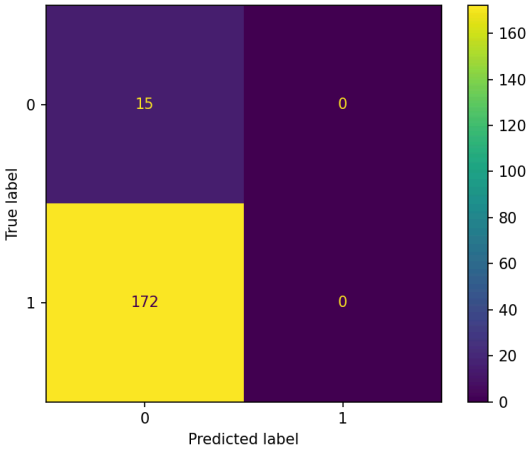
Train	Test
	
	

Three layers, epsilon .0001, does not seem to be doing as well as .001 but still better than just adding more neurons, approx 3 minutes

Train	Test
	

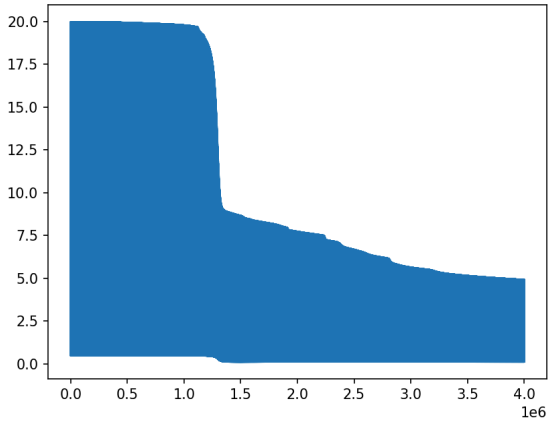
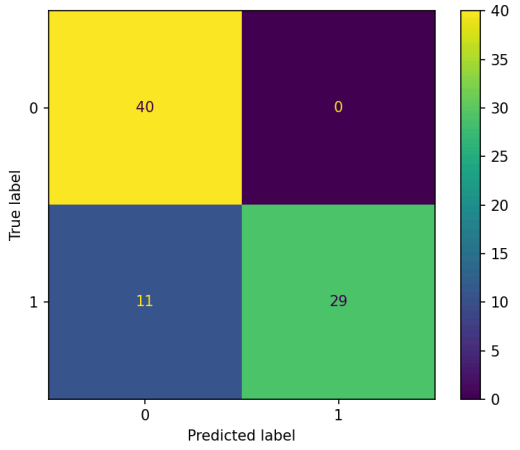
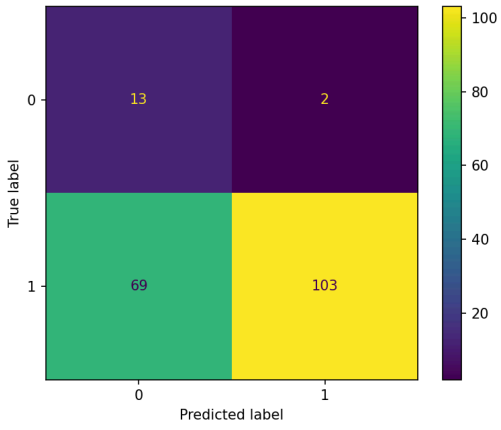


Three layers 0.00001 epsilon. Not good at all with such a small learning rate we probably need more epochs approximately 3 minutes

Train	Test
	
	

At this point I tried a couple more rounds with a smaller learning rate, more epochs and fewer neurons. I did get better results but more epochs means more running time. Learning rate = 0.00005, epochs = 50,000, 35 neurons and only 1 hidden layer. Run time approximately 10 minutes

It does well and is not an insane amount of run time.

Train	Test
	
	

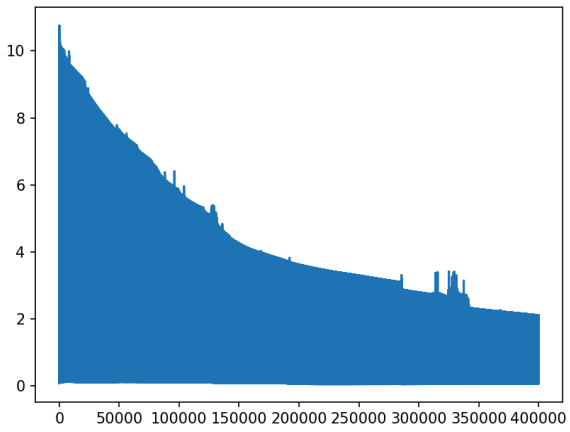
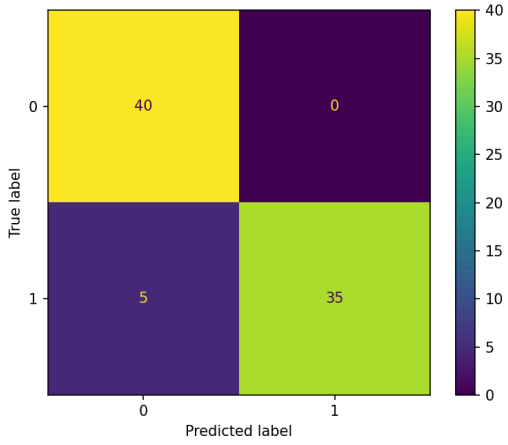
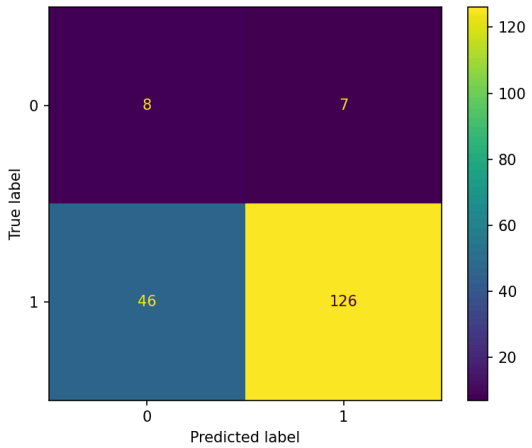
I did try running some variations for hours at a time, more neurons, more epochs without getting results much better than this.

Finally I implemented my weight suppression function.

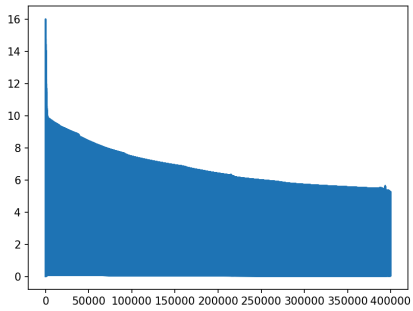
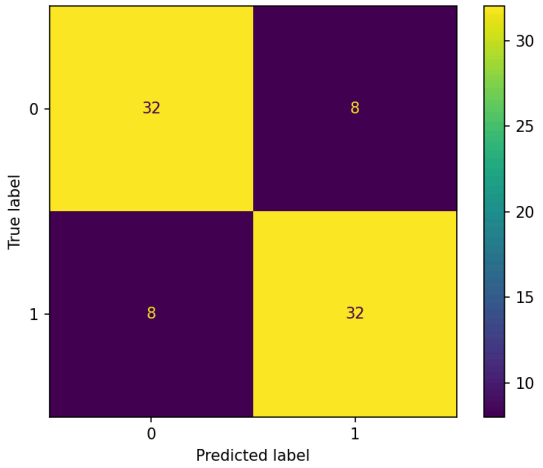
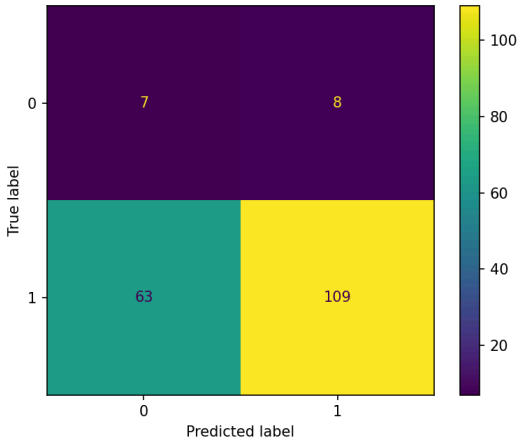
Here I am using three layers, 44 neurons each, 5,000 epochs and epsilon at 0.001 for consistency.

We start with a suppression value of 0.12, so any weight below that will be suppressed, we will check for this at every 100 epochs. The run time was approximately 2 minutes

We do see improvement. The error rate has gone down and we are getting fewer false positives, so that is good.

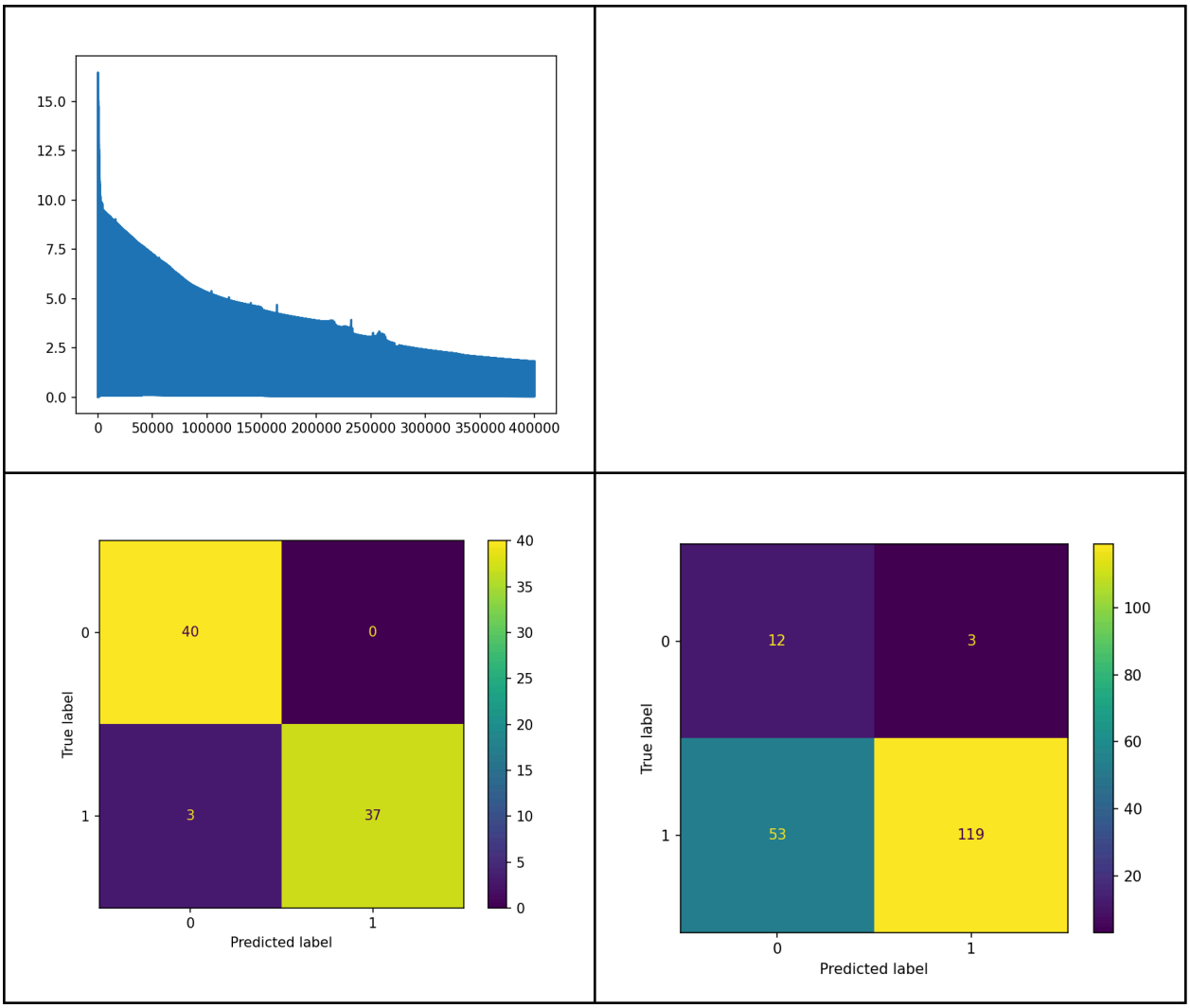
Train	Test
	
	

I tried to fine tune this and mostly what I learned is that is the suppression is too small i.e. 0.01, we do not see much improvement

Training	Test
	
	

I would continue to play around with it and my best results were found using three layers with 44 neurons, epsilon at 0.001, epochs = 5,000, suppression threshold at 0.1, and checks for suppression after 5 epochs.

Training	Test



We catch 12/15 of the abnormalities with only mislabeling about a third of the normal hearts as abnormal which is as good as I could make it.

I had many more experiments but erased most of them to save space.

Conclusion:
This does not seem like a simple problem, it becomes very easy for the model to only identify one class or the other. Just trying to tackle it with more and more neurons does little to help while also increasing run time, but using a small learning rate and neural suppression we do manage to get much better results. Therefore in neural networks it seems that sometimes fewer neurons is more knowledge.