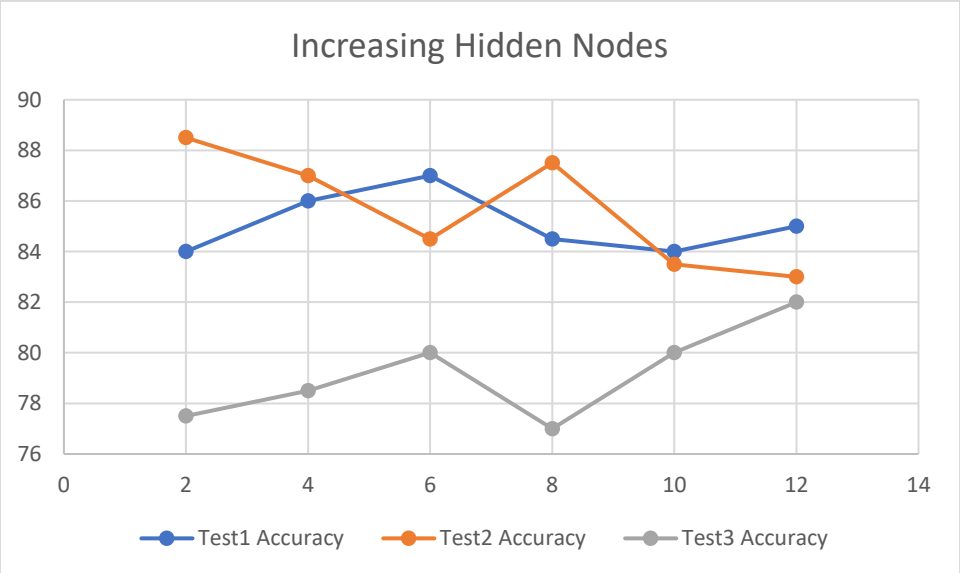
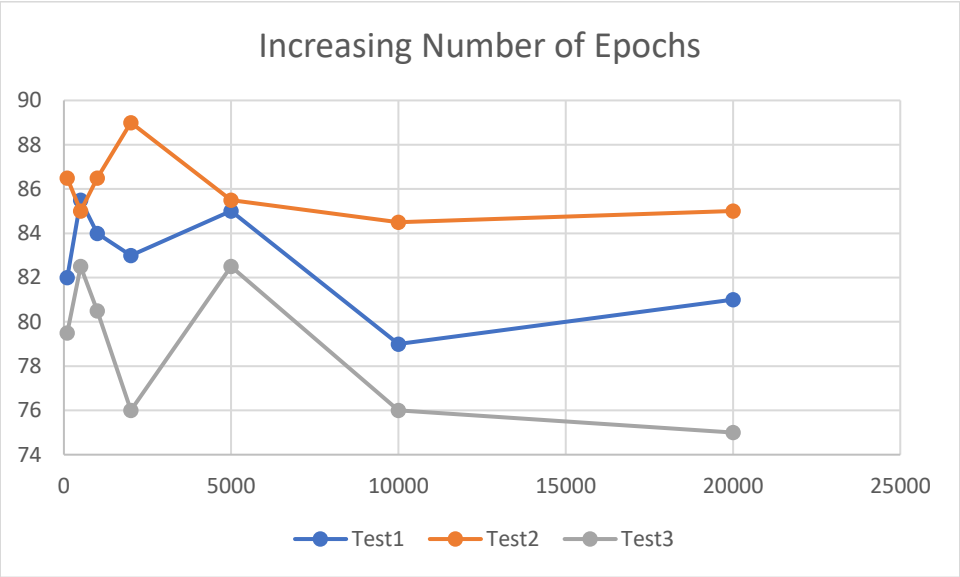
Introduction:

In this first homework, I attempt to construct a single hidden-layer, feed forward neural net to train on provided data and test its classification ability with test data. I decided to implement this neural net in MatLab, as it is the language I am most familiar with currently. This neural net receives five inputs (x1:x5), and outputs two values, y1 and y2. If y1 is 1 and y2 is zero, the classification is "fail". If y1 is 0 and y2 is 1, then it is a pass. There is a single hidden layer, but the number of hidden layer units was coded to be variable, as are many of the variables that determine neural net performance. When the training set is iterated through all 200 of its data segments, I randomly sorted them each epoch to eliminate dependence on the order of delivery during the training of the net. In the following questions, I examine the effects these different variables have and observe how changing then can change the performance of my neural net.

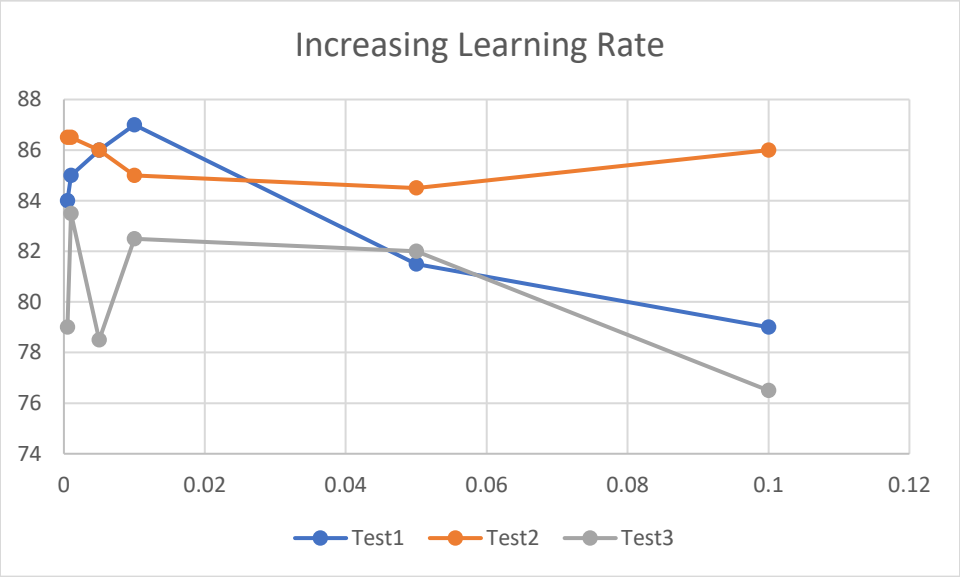
1) a) Shown in the graph below, increasing the number of hidden units seemed to be generally beneficial for Test3 data, but slightly less beneficial for Test 1 and 2. As I increase the number of units from 2 to 12, Test data 1 and 2 trend downward while Test 3 trends somewhat upward. It would seem the sweet spot for hidden units, based on this chart, might be around 5-6 hidden units. This is where there appears to be a highest performance percentage balance between the values of Test 1 and 2 and a local peak for Test 3 data.



b) As I increase the number of epochs, each set of test data adapts relatively similarly, with the percentage correct trending negatively with greater epoch lengths. In addition to the obvious downside of greatly increased processing time, going past 5,000 epochs doesn't seem to create any benefit for performance. I choose to settle on an epoch length of 5000 for the best results for data trained on training set 1.



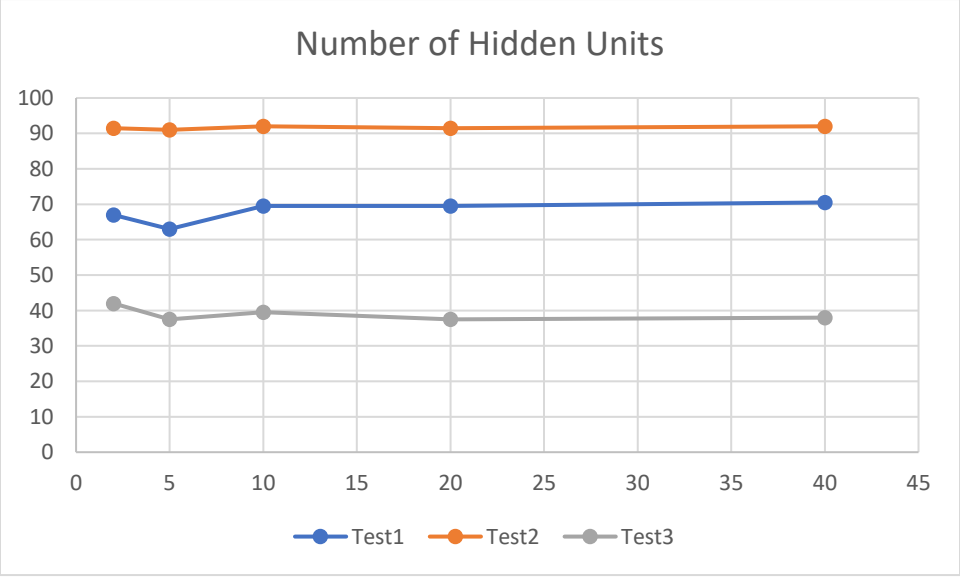
c) As I increased the learning rate from 0.0005 to 0.1, I observed a general decline across all three tests. Test 1 and 3 data display this trend the strongest, with Test 2 data being more variable. The highest performance balance for the three tests seemed centered around 0.005-0.01; I will use 0.005 for further tests.



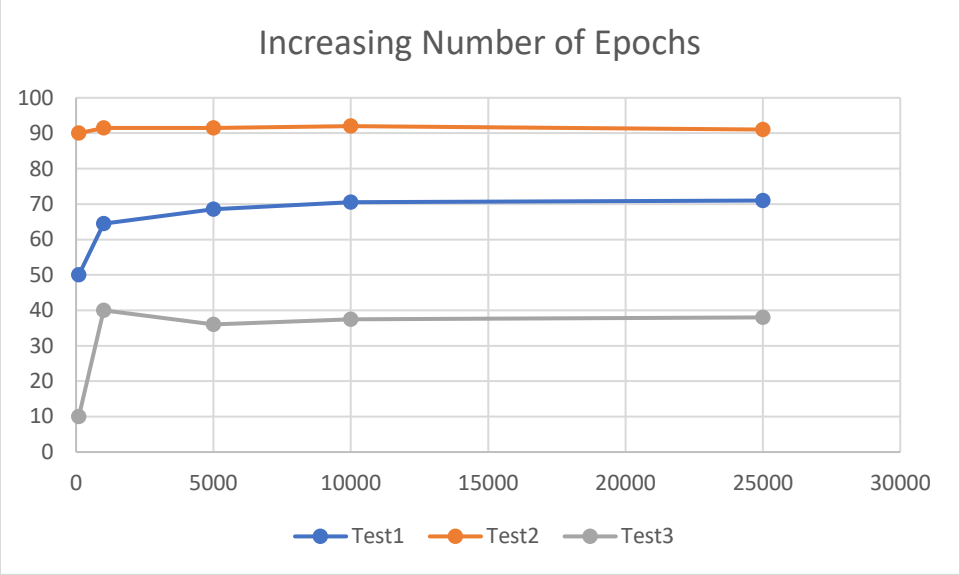
d) One area of significance in this training set is the cross-talk between training length and learning rate; if the learning rate is too slow and the training length is too short, the result is poor training/inconsistent results (for example, η=0.001 and epochs = 100 yields 90 percent correct for test3, 10 percent for test2, and 50 percent for test 1). Another observation is that the number of epochs is directly correlated to the time it took to run, while the number of hidden layers scaled more slowly, where doubling the number of hidden layer units would result in only 1.5x slower runtime (6units -> 12units went from 28sec to 40 sec). One factor that certainly affected my results was using the sigmoid cost function to calculate from; other functions would certainly change how the backpropagation works.

e) Test 1 and 2 were generally close to each other in performance, and performance trendlines seemed to mirror each other. Test 3 was typically lower than 1 and 2 by ~5-8% across all variables. These discrepancies between the testing sets correlate with their different number of pass and fail content. Test 1, which had an equal number of pass or fail outcomes, most closely resembled the training data found in training set 1. I would have expected it to perform the best as a result, due to the its similarity. However, it performed similarly to Test set 2, and sometimes less successfully while varying certain parameters such as number of epochs. Test set 2 has 180 fails and 20 passes, so for it to perform as well as test set 1 would imply my net is better at detecting failures than passes. This is reinforced in test set 3, which has 180 passes and only 20 failures, and it generally is less accurate than the other two tests.

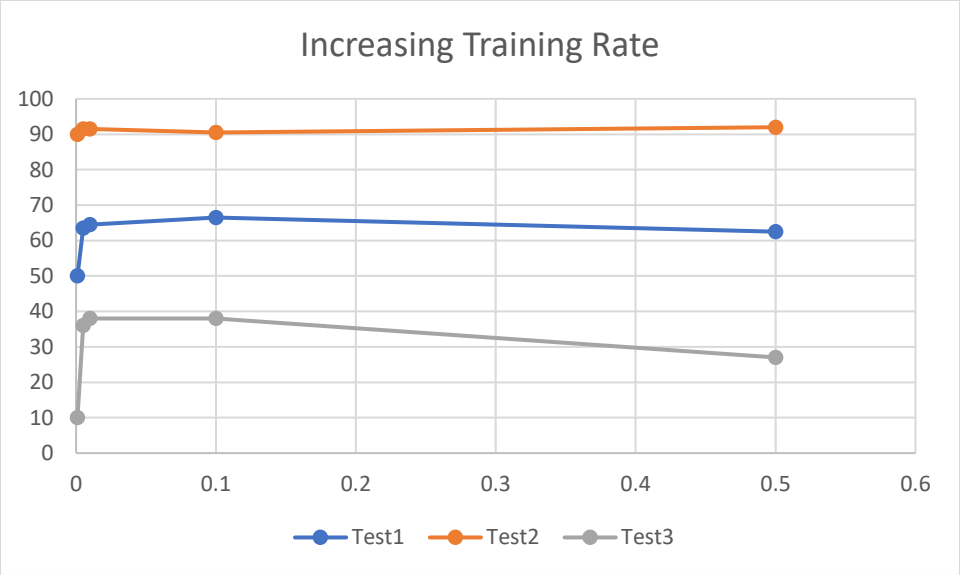
2) a) In this second training set, I repeated the experiments from question one with some modifications on variables and their spread. Shown below, having more units than 10 seems to little effect on the performance of the net. For Test 1 and 3, 5 seems to cause a dip in performance, but it is brought back up at 10. Test 2 seems virtually unaffected by changing the number of units, and does better than the other two for each case.



b) Starting at 100 epochs, the net fails to reach acceptable classification initially. The rates go up at 100 and stabilize around 5000, remaining stable for 10000 and beyond. Again, Test 2 seems mostly unaffected by the changes, with Test1 and Test3 data reacting greater at lower values. I chose to use 5000 for further experimentation afterwards.



c) Increasing the training rate from 0.001 to 0.5 started at failure, but gradually increased towards maximal values between 0.01 and 0.1. I believe an optimal value moving forward should be about 0.05. Again, Test 2 data was mainly unaffected, while Test 1 and 3 showed a decline in performance above 0.1.



d) As I discussed in question 1, the major adjustable component to this net besides the already discussed learning rate, epoch length, and number of hidden units would be the choice of cost function. I could use a radial basis function network instead, and it would change the performance. Another aspect of the training that could have impacted the results was the randomized starting values for the initial weights. If better starting values were known, it could improve the training; however, I started with a random value between -0.5 and 0.5. Finally, setting up the net so that each input layer was completely connected to every unit of the output layer was another assumption that could impact backpropagation and performance in a different situation. Choosing different connectivities would certainly impact the final performance and error tuning of my neural net.

e) There were even greater differences between the three tests using training set 2. This is due to the disproportionate number of fails compared to passes in the training set. As a result, the net is poorly optimized to recognize a given pass or fail, and is heavily biased towards characterizing test data as a fail. This is clearly seen in the performance, where Test 2 data (which has 180 fail, 20 pass) has the greatest performance and is less affected by changing variables, and Test 3 data (which has 20 fail and 180 pass) performs the worst of the three.

Conclusions:

As I wrote the code for the neural net, based on the class notes and discussion, I encountered a few major problems. Even though I did my best to duplicate the correct formulas, I accidentally created gradient *ascent*, by incorrectly setting the sign of η wrong. This caused the solutions to diverge instead of converging on the correct characterizations. One small sign, and the entire net was unusable. Neural nets can adapt to a wide variety of inputs, but they require careful planning and setup to ensure their functionality. As seen in questions 1 and 2, the training rate and number of epochs need to be chosen to ensure functionality by having enough time to properly train without going too far and becoming painfully slow. The number of hidden units is also important, needing to balance speed with being robust. From the experiments, the ideal number of hidden units seems to be between 5 and 10.

My takeaway from this experiment is that a balanced and varied training set is essential for correct performance. As seen with training set 2, a biased training set will create imbalances in the results that do not closely resemble the training set. Training set 1 allowed for much more robust performance, since it was trained with an even set of pass and fail data. Even still, it had a slight inclination towards fail characterization vs pass, but far less pronounced than when training set 2 was used. To fix the failing of training set 2, I would use more training data when possible, and attempt to even the number of pass and fail examples the net was given. I might duplicate the 20 pass examples in training set 2 to give me 180 passes to match the given 180 fails. Or, I could downsample to 20 fails to match my 20 passes. Neither work as well as having even amounts of data, but both would lessen the impact that the fail/pass ratio had on the training of the net.

Data Tables: Training Set 1

Epochs	N	Hidden	ProcessTime	Test1 Accuracy	Test2 Accuracy	Test3 Accuracy
2000	0.01	2	17.15	84	88.5	77.5
2000	0.01	4	21.72	86	87	78.5
2000	0.01	6	27.94	87	84.5	80
2000	0.01	8	31.34	84.5	87.5	77
2000	0.01	10	36.31	84	83.5	80
2000	0.01	12	40.83	85	83	82
100	0.01	5	1.2	82	86.5	79.5
500	0.01	5	6.09	85.5	85	82.5
1000	0.01	5	12.01	84	86.5	80.5
2000	0.01	5	24.45	83	89	76
5000	0.01	5	61.01	85	85.5	82.5
10000	0.01	5	120.17	79	84.5	76
20000	0.01	5	240	81	85	75
5000	0.1	5	60.08	79	86	76.5
5000	0.05	5	60.25	81.5	84.5	82
5000	0.01	5	60.27	87	85	82.5
5000	0.005	5	61.28	86	86	78.5
5000	0.001	5	60.69	85	86.5	83.5
5000	0.0005	5	60.95	84	86.5	79

Training Set 2:

Epochs	N	Hidden	ProcessTime	Test1 Accuracy	Test2 Accuracy	Test3 Accuracy
5000	0.005	2	43.2	67	91.5	42
5000	0.005	5	63	63	91	37.5
5000	0.005	10	90	69.5	92	39.5
5000	0.005	20	149	69.5	91.5	37.5
5000	0.005	40	284.95	70.5	92	38
100	0.005	10	1.8	50	90	10
1000	0.005	10	18.15	64.5	91.5	40
5000	0.005	10	92.69	68.5	91.5	36
10000	0.005	10	180	70.5	92	37.5
25000	0.005	10	454.49	71	91	38
1000	0.5	10	18.89	62.5	92	27
1000	0.1	10	18.15	66.5	90.5	38
1000	0.01	10	18.06	64.5	91.5	38
1000	0.005	10	18.67	63.5	91.5	36
1000	0.001	10	18.06	50	90	10

Matlab Code for training:

```
%Neural Net for processing test data: Homework 1
clear all
clc

%%Import data
test1=csvread('test1.csv');
test2=csvread('test2.csv');
test3=csvread('test3.csv');
train1=csvread('train1.csv');
train2=csvread('train2.csv');

%% Establish Variables/Values/Functions
f = @(x) 1/(1+exp(-x));
Q = @(inputs,w,x,w0) f(sum(w(1:inputs).*x)+w0);

MSE=0;
hid_num=10;
N=0.005;
inputmat=train2;

w0(1:2)=rand(1,2)-0.5;
v0(1:hid_num)=rand(1,hid_num)-0.5;
w(1:(2*hid_num))=rand(1,(2*hid_num))-0.5;
v(1:hid_num*5)=rand(1,hid_num*5)-0.5;

%% Loop by epochs of 5000
tic

for g=1:1000

%randomize order
shuffledArray = inputmat(randperm(size(inputmat,1)),:);
inputmat = shuffledArray;

%%Go through all 200 rows
for i=1:length(inputmat)
    %Calculate output
    x_var=inputmat(i,1:5);
    y_var=inputmat(i,6:7);
    %fprintf('%f\n',y_var(1));

for i=1:hid_num
    H(i)=Q(5,v(1+(5*(i-1)):5+(5*(i-1))),x_var,v0(i));
end

for j=1:2
    Y(j)=Q(hid_num,w(1+(hid_num*(j-1)):hid_num+(hid_num*(j-1))),H,w0(j));
end

% Calc Error between calculated and known y
Error(1:2)=[Y(1)-y_var(1),Y(2)-y_var(2)];
%Error(1:2)=[Y(2)-y_var(2),Y(1)-y_var(1)];
```

```
%%Update weights
for a=1:2
    DeltaOut(a)=Error(a)*Y(a)*(1-Y(a));
end

W(1,1:hid_num)=w(1:hid_num);
W(2,1:hid_num)=w((hid_num+1):2*hid_num);
DeltaHid(1:hid_num)=mtimes(DeltaOut,W).*H.*(1-H);
%
% for b=1:hid_num
%     DeltaHid(b)=(w(b)*DeltaOut(1)+w(b+hid_num)*DeltaOut(2))*H(b)*(1-H(b));
% end

for c=1:2
w0(c)=w0(c)-N*DeltaOut(c);
w(1+(hid_num*(c-1)):hid_num+(hid_num*(c-1)))=w(1+(hid_num*(c-
1)):hid_num+(hid_num*(c-1)))-N*DeltaOut(c).*H;
end

for d=1:hid_num
v0(d)=v0(d)-N*DeltaHid(d);
v(1+(5*(d-1)):5+(5*(d-1)))=v(1+(5*(d-1)):5+(5*(d-1)))-N*DeltaHid(d).*x_var;
end

MSE=MSE+mean(Error(1:2).^2);
end

%MSE=mean(Error(1:2).^2);
% fprintf('%f, %f\n',Y(1),Y(2));
% fprintf('MSE is %4.2f\n',MSE);
% MSE=0;

end
toc
```

Code for Testing:

```
inputmat=test3;
comparison=0;
beans=0;

for i=1:length(inputmat)
    %% Calculate output
    x_var=inputmat(i,1:5);
    y_var=inputmat(i,6:7);

for i=1:hid_num
    H(i)=Q(5,v(1+(5*(i-1)):5+(5*(i-1))),x_var,v0(i));
end
```

```
for j=1:2
    Y(j)=Q(hid_num,w(1+(hid_num*(j-1)):hid_num+(hid_num*(j-1))),H,w0(j));
end

[M,I]=max(y_var);
[M2,I2]=max(Y);

if I == I2
     comparison=comparison+1;
end

%fprintf('%f, %f\n',Y(1),Y(2));
end

results=comparison/length(inputmat)*100;
fprintf('The results were %4.2f percent accurate.\n', results);
```