CMSC828L: DEEP LEARNING

PROBLEM SET 1

Implementing Neural Networks from scratch

By:
Anirudh Nakra
UID: 118134444

CMSC-828L Problem Set 1
Implementing an NN from scratch

Anirudh Nakra

February 18, 2022

1 Just a Linear and a Bias Layer !

1.1 Questions

Q1. Generate some random 1D test data according to a simple linear function, with Gaussian noise added. For example, your data might be generated with: y = 7x + 3 + ϵ, where ϵ is a Gaussian random variable. Include a plot showing the training data and the function that your network computes. (You can plot the function by evaluating it on a range of different inputs). This is all 1D, so easy to visualize.

Ans1. Data generation:

The method for the data generations was as specified in the the Problem Set. I used a linear function in x of the type 7x +3 for x in the range (-2,2) and generated randomly. In addition to make the data noisy simulating real world data, I added some additive white gaussian noise epsilon which was was sampled from a standard normal distribution. To generate the held out dataset, I split the created dataset with the equation into two sets, each of them having half of the total created data. One of them was used for the training and the other was used for testing.

The neural network created is very simplistic and only contains an input and an output layer without any nonlinearities present. So essentially what the NN learns is the best linear combination of the inputs to get the optimal regression output. The weight and bias initialization are a matter of great importance in any NN setup. The first simplistic weight initialization I used was to setup it as a matrix of all ones. This was easier to look at and see if the system was working or not since everything was very visually comprehensible with all weights being ones. It allowed me to verify that my feedforward was working or not and if the weights were being updated. However as postulated by many researchers, this way is far from rigorous enough and is not optimal. The next choice for the weight initialisation I used was it being sampled from a standard normal distribution. This worked very well but there were spurious cases where the initialisations were higher than I wanted them to be. All of this led me to the Kaiming He initialisation which is a standard initialisation scheme many ML researchers used nowadays. I removed the sqrt 2 term in the initialisation to guarantee lower variance and reduce spurious errors. There are still some unforeseen and random errors from time to time but that is believed to be a result of a specific random seed in the data and not the initialization scheme per say.

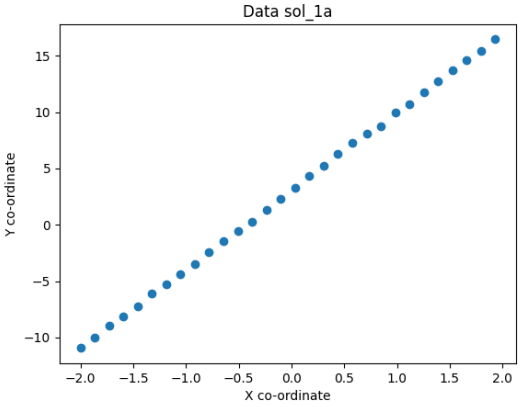


Fig 1: Data Q1a

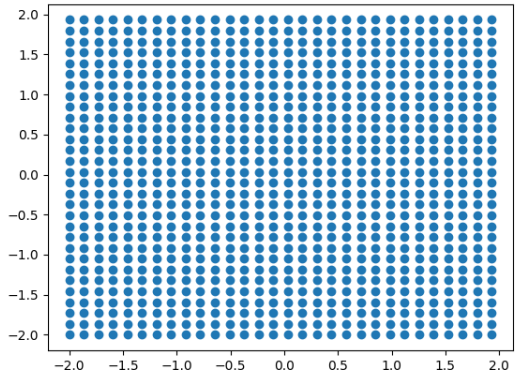


Fig 2: Data Q1b

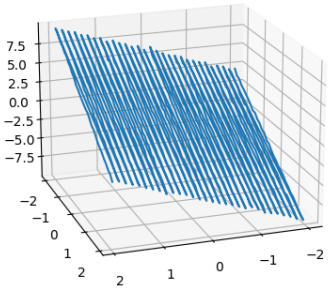


Fig 3: Data Q1b 3D

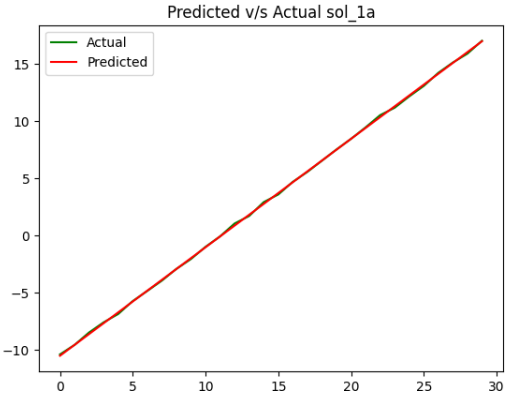


Fig 4: Predicted and Testing Data Q1a

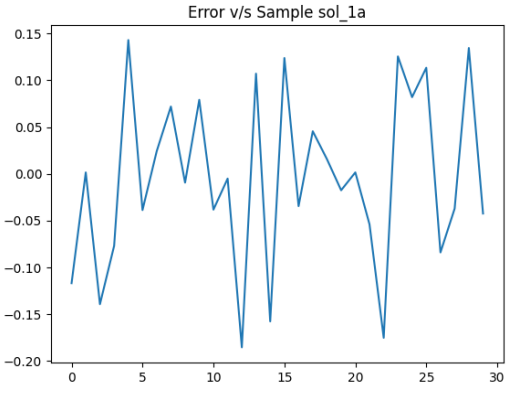


Fig 5: Absolute error for testing Q1a

Training v/s Predicted Data:

The function that the Neural network computes is as shown. As visible, the neural network converges to almost the same function as our data creates and it does so really fast. A couple thousand iterations with a learning rate of 0.5 is enough for the system to pass the MSE threshold in the public tests and its visible from the above graph that the approximation is near identical (the green graph which represents the actual data is invisible but some noisy green bumps can be seen). For matters of visual clarity, I have also plotted the absolute error v/s the sample being predicted and we see that on average the error is at max 0.15 which is quite a low error when compared with the actual values of the signal which is 2 orders of magnitudes higher than that.

Q2. Perform a similar experiment using higher-dimensional input. Demonstrate that your trained network is computing a reasonable function by evaluating it on some held-out test data, and comparing the computed values to ground truth values.

Ans2. Data generation:

The data generation for the high dimensional case is much more complicated. My data generators creates 2D training data for every sample as needed. The first step in creating the data is to create an array of two variables that would essentially act as the predictors to the regression problem. The data code then creates a mesh of these training samples and these can be seen very clearly in the figure shown below. Furthermore in the 3d plot, the regressor has been plotted with the predictors in a 3d graph so that the actual planar regression going on is easily visualisable. The 1b problem is then essentially reducible to a linear regression plane problem with the plane taking on a similar structure to the y=x line as was in 1a.

The weight initialisation that is being followed is still the Kaiming He intilaisation without the sqrt 2 factor. Something I didnt mention in the last section was the initialisation of biases which are done using the primitive initialization method i mentioned earlier i.e being sampled from a standard normal distribution.

Training v/s Predicted Data:

The function that the neural network computes is shown in the figure. Again we can see that the red curve almost completely covers the green curve which means our predicted function is very very close to the actual curve. The only place the green curve is visible is at the edges of the sawtooth like pattern. The curve can be further analysed using the absolute error between the predictions and the actual regressor values. The observations by themselves are between -10 to 10 and the error is in the range of 0.3 to -0.3 which is as in 1a about 2 orders of magnitudes lower. Also on average the error trend seems to have a rolling mean of around 0 which makes us more confident about the accuracy of our model.

Q3. How difficult was it to find an appropriate set of hyperparameters to solve this problem?

Ans2. I mentioned the initialisation of the weights and the biases. They are implicit hyperparameters that I use in every subproblem as is. Other than these two, the more explicit hyperparameters are the learning rate and the number of iterations we run the NN training for.

I conducted the analysis of the learning rate and the number of iterations as separate to each other keeping one parameter fixed when analysis the other. This kind of analysis is perhaps not

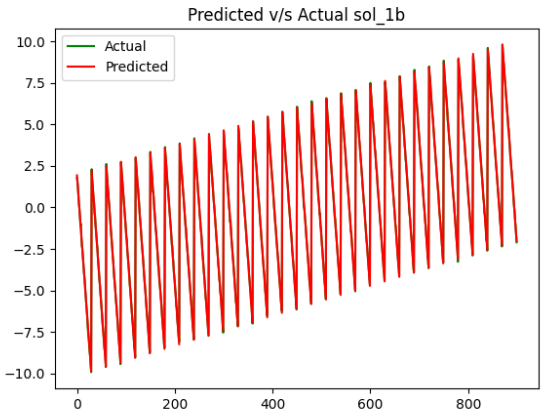


Fig 6: Predicted v/s Testing Data Q1b

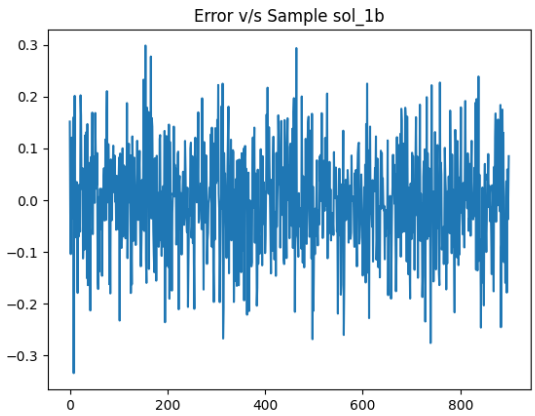


Fig 7: Absolute error Q1b

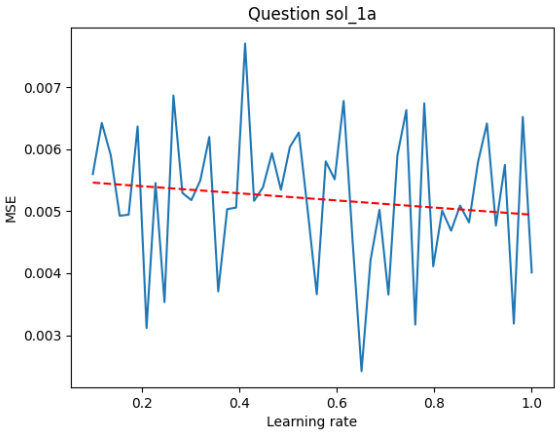


Fig 8: MSE v/s LR Q1a

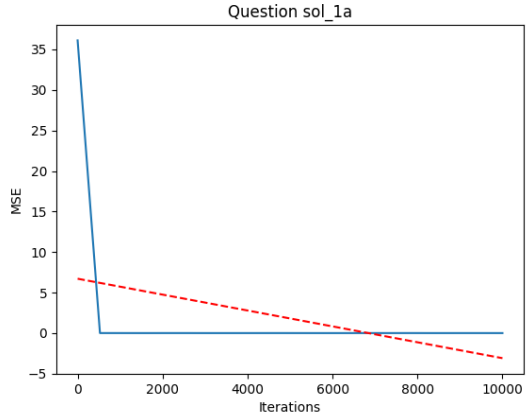


Fig 9: MSE v/s Iter Q1a

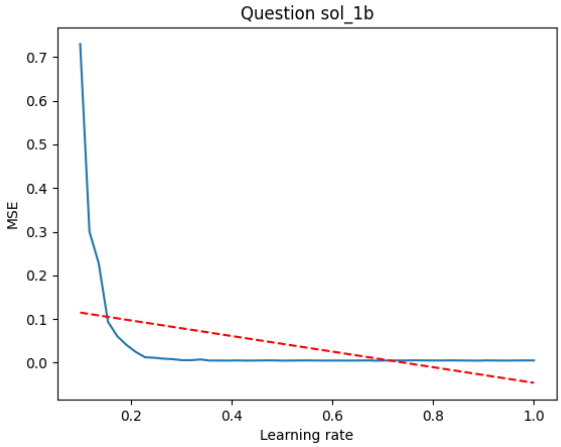


Fig 10: MSE v/s LR Q1b

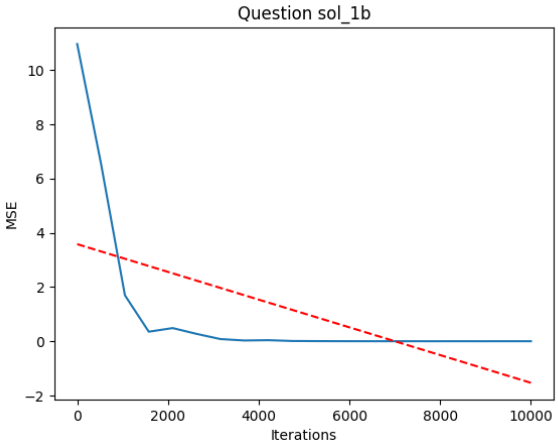


Fig 11: MSE v/s Iter Q1b

the most rigorous since the LR and the iterations we run for have an implicit dependence. Of course if the learning rate is really small then it might need a huge amount of iterations to converge to the global minima. Similarly if we only vary the iterations, a high LR can enable overshooting of the global minima since even a small gradient will have a large effect during the weight updates. Obviously there needs to be a tradeoff between these two. The added benefit, though, which I had when conducting these analysis was that of past literature which enabled me to make a wise choice about the hyperparameters so I could analyse the LR and iterations as a separate entity.

For subproblem a, the following hyperparameters concerns were addressed.

The given experiments were performed at a fixed learning rate of 0.5. As can be seen from the graphs, the neural network structure for the given problem 1a converges really fast. At 1000-2000, it already achieves the desired MSE threshold of 0.01. In fact at iterations around 5000, the MSE is usually around 0.005 which is half the desired threshold and foreshadows an extremely good reconstruction of the data. Along with the main plot, I have also plotted the general trend in the graph which is a good indicator for inferring observations where there are lots of spurious errors/spiky behaviour. The trend reasserts our observation that the general trend from 0-10000 iterations is a net decrease in the MSE with the rise in number of iterations. Along with the MSE trend, I have also plotted the number of times the public tests pass with respect to the changing number of iterations. As is clearly visible, after 1000 iterations the MSE tests never fail signifying true convergence.

The given experiments were performed at a fixed amount of 5000 iterations. For 5000 iterations, the experiments signify the possibility of 0.5-0.8 LR being the more conducive to true convergence with more minimas in this region of the spiky behaviour than any other. From the MSE test pass/fail scatter plot we can see that even for really low LRs, the MSE test doesnt fail and the actual MSE is around 0.005 which is as mentioned in the section before half of the threshold and really close to true convergence. There is also an observable general trend downwards with LR but since the graph is impulsive and the values of the MSE are near identical, more analysis needs to be done before making any bold claim.

For subproblem b, the following hyperparameters concerns were addressed.

It is obvious that the learning process for the higher dimensional data would be more complicated due to the introduction of a new predictor and this would complicate the earlier analyzed results. This is visibly true. For a fixed LR of 0.5, we can see that the MSE converges slower as compared to 1a and the threshold for good convergence is visibly higher in terms of the iterations it takes. It is also worth noting that the MSE bound for the problem 1b is noticeably looser than the one for 1a. Using the experimental data, 5000 again seems like a good choice to use but its important to acknowledge that the lower bound of iterations for good MSE convergence is higher in 1b than in 1a.

Correspondingly, in the scatter plot that gives us the pass/fail, we can visualise that after 3000 iterations the MSE bound is passed satisfactorily for the given LR. However it fails for all iterations before it and the setup fails more times for the low iterations (200-3000) than it did for 1a.

Fixed at 10,000 iterations, there seems to be a clear trend of decreasing MSE with increasing LR. The trend is really strong with it saturating above LRs of 0.3. A possible reason for this is just because 10,000 iterations is probably very small for convergence with the low LRs of 0.01-0.2. The saturation after 0.3 upto 1 implies that the NN successfully reaches the convergence threshold for every possible value. The pass/fail scatter plot seems to imply the same conclusion about LRs

below 0.2 not being able to converge to the local minima in the 10,000 iterations. These situations are where the tradeoffs between the LRs and the iterations need to be visualised.

Armed with the abovementioned plots, choosing the require hyperparameters (LR and iterations) was not hard since there are clear trends visible in the graphs and there are only two variables we have to analyse at a particular time making it easy to study the impact of the tradeoff side by side.

For Q1a, I chose the LR as 0.5 with 5000 iterations and it converges consistently.

For Q1b, I chose the LR as 0.5 with 10000 iterations and it converges consistently. The reason for choosing the high number of iterations even though its not immediately obvious from the curves is the issue of consistency and random seeds as mentioned at the starting.

2 Implementing an NN with one hidden layer

2.1 Questions

Q1. 1D. Generate training data from a simple 1D function, such as a sine wave. You do not need to add noise to the function. Train your network to fit this data. Turn in a plot that shows the training data, along with a curve showing the function that your network computes.

Ans1.

The data for Q2a is a simple sampled sine wave with additive white gaussian noise added to it. Again the training testing is a 50:50 split as in the previous questions. The data is visualised. It can be seen that the predicted curve closely follows the actual training samples when interpolated. The convergence here is significantly slower than the problems in Q1 though, presumably due to the added complexity of having a hidden layer with additional weights to be computed. Its empirically verified and will be shown later on that at an LR of 0.5, the network takes about 25000 iterations to converge to the desired threshold consistently. The absolute error has also been plotted and it can be seen that on average the absolute error magnitude is about 2 orders lower than the actual data.

Q2. Perform a similar experiment using higher-dimensional input. Demonstrate that your trained network is computing a reasonable function by evaluating it on some held-out test data, and comparing the computed values to ground truth values.

Ans2.

Generating the data for Q2b is again a complicated procedure, most of which follows the algorithm outlined in the data generation for Q1b. The two predictors are individually intialised as a constant step linspace in the function. Then they are combined to make a meshgrid creating a square grid of sampled 2d data. This meshgrid of data is then passed on to create a quadratic spatial hyperplane with some additive white gaussian noise sampled from a standard normal added to it. As seen below, the predicted curve again follows the actual curve pretty closely. The convergence of the high dimensional data is even slower than the scalar part of this question. The most intuitive explanation is just the addition of an additional feature enabling more calculations and complicated behaviour. An interesting thing to note from the absolute error curve is the nature of

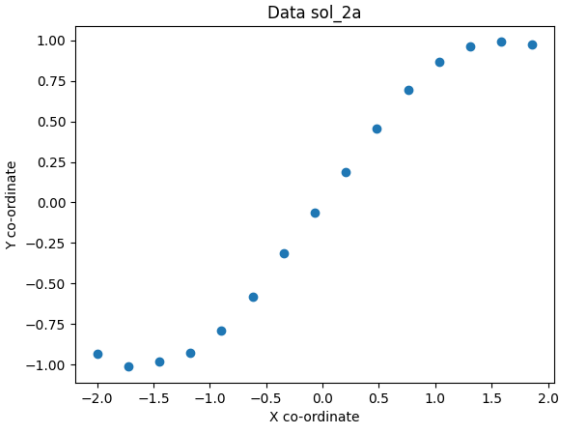


Fig 12: Q2a Data

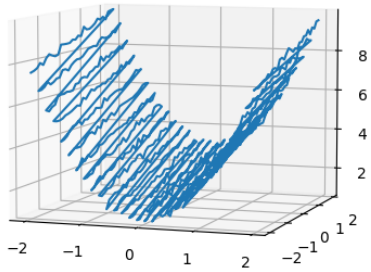


Fig 13: Q2b Data

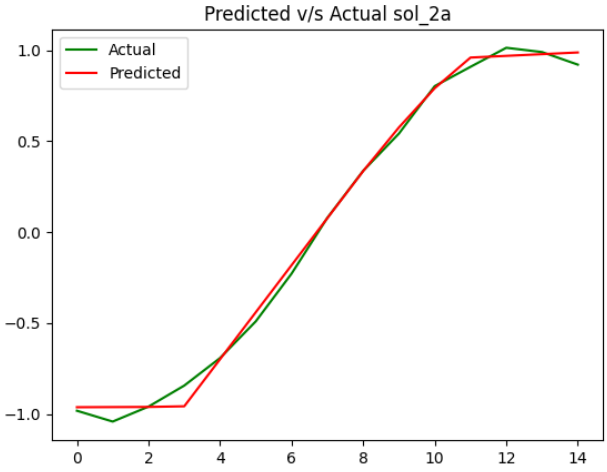


Fig 14: Predicted v/s Test Data Q2a

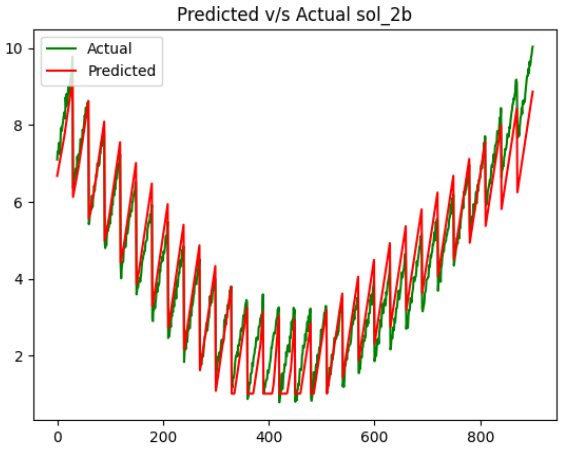


Fig 15: Predicted v/s Test Q2b

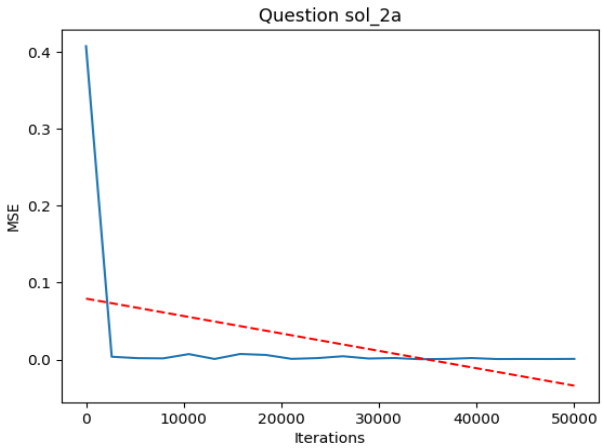


Fig 16: MSE v/s LR Q2a

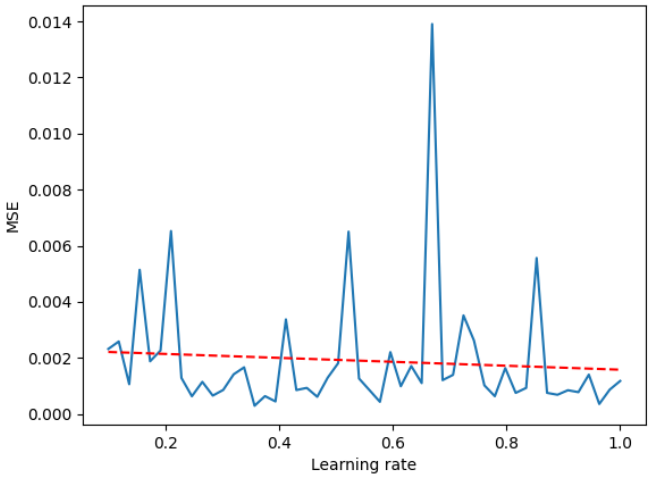


Fig 17: MSE v/s Iter Q2a

the noise present. The absolute error by itself resembles a standard normal gaussian. I postulate that the shape of the error curve is this way due to two possible reasons. One is the intrinsic nature of the noise we added at the time of data generation and the other is the possibility that the NN is learning the actual values of the regression problem but has an issue with learning the curvature.

Q3. How difficult was it to find an appropriate set of hyperparamters to solve this problem? Do you notice any difference in the difficulty of solving the 1D problem and solving the higher dimensional problem?

Ans3.

For Q2a, there is a clear and consistent trend of decreasing MSE with an increase in the number of iterations for a given learning rate of 0.5. From the pass/fail scatter plot, we can see that till the number of iterations exceeds 20,000 there is a lot of volatile behaviour. It is presumed this is because of the strict threshold on the scalar subproblems and the chosen LR is not able to find a minima immediately but over time it is able to slowly go back and forth and converge to a minima below the specified threshold. The architecture used for this experiment is with 5 hidden units. We will talk about this in a future section but 5 nodes in the hidden layer is not the best we can do with this architecture. By changing the hidden units, we can enable the network to converge on the test set faster. It is also worth mentioning that the experimental observations made by the graphs are checking the rates of convergence and tuned hyperparameters with respect to the testing set and not necessarily the training set (MSE of the test set). This is done to be able to check the generalisation behaviours of the network and to not get to a point where the network is overfitting.

It is easily inferable from the graph that for a fixed iteration number of 25000, the MSE is mostly insensitive to the change in learning rate. For the fixed iteration choice, most of the choices of the LR converge to the MSE. The choice in my program of 0.5 from this set of selection is due to the consistency of the public tests which was right in the middle of the stretch from 0.2 to 0.6 which are the LRs that reached the specified MSE threshold the fastest. It can be seen that other than the outliers (which is assumed to be a random seed error after consultation with the Instructor team) the LRs seem to be passing the thresholds consistently for the choice of iterations from the scatter pass/fail plot.

For Q2a, it can be observed from the curve that the best choices for the number of hidden units is between 10 and 20 with it remaining stable upto 30 hidden units. The choice of 15 hidden units thus is both practical as to not introduce huge computational expenses while maintaining good convergence on the test set. This is in comparison to 5 hidden units that we were talking about earlier and it can be clearly seen that the hidden units ¿10 outperform 5 hidden nodes which is very close to the initial drop of the curve tracking MSE v/s number of hidden units and might not be consistent enough. Consequently, it can also be seen that the scatter pass/fail plot tells us that the convergence is consistent for the hidden units with respect to the public test thresholds.

For Q2b, It is visibly obvious from the graph that with an increase in iterations for a fixed LR of 0.5 results in a net net decrease in the test set MSE. There is a threshold of around 1500 iterations after which as visible from the pass/fail plot, the MSE converges every single test run. As expected with an increase in the number of iterations, the LR of 0.5 guarantees the minima every time. The curve for the LR follows the behaviour set by the iterations plot. For a fixed iteration of 20000, as the learning rate increases, the test MSE follows a net negative trend. In fact all LRs above 0.05 are able to converge to the minima in the given number of iterations. It is however important to notice that the volatility in the behaviour also increases as the LR increases. This is possibly

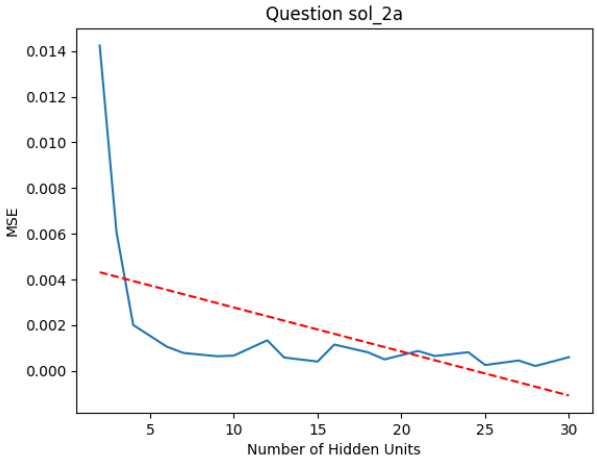
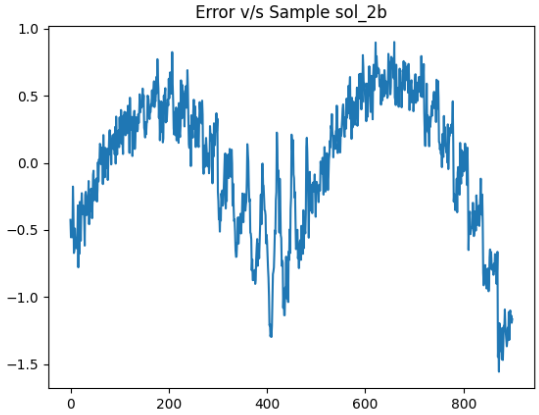


Fig 18: MSE vs Hidden Units Q2a



Fog 19: Abs error Q2b

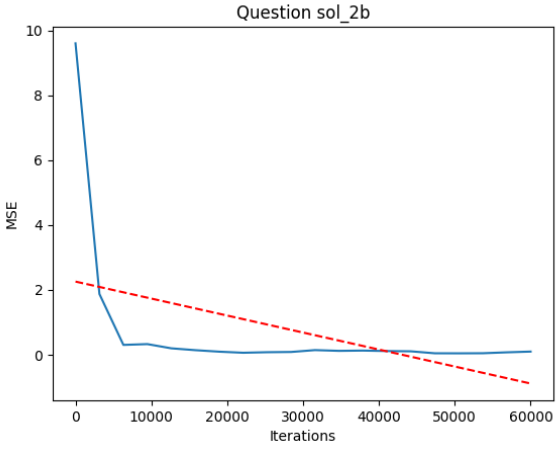


Fig 20: MSE v/s Iterations Q2b

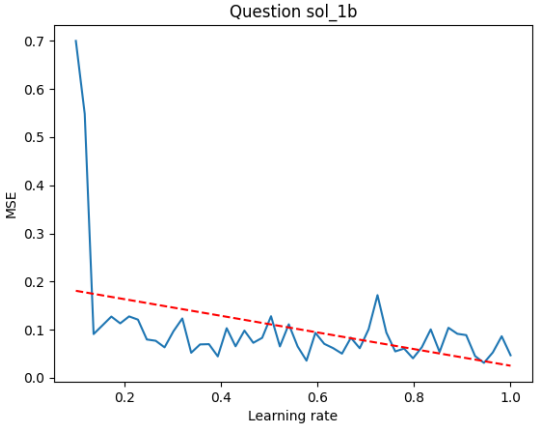


Fig 21: MSE v/s LR Q2b

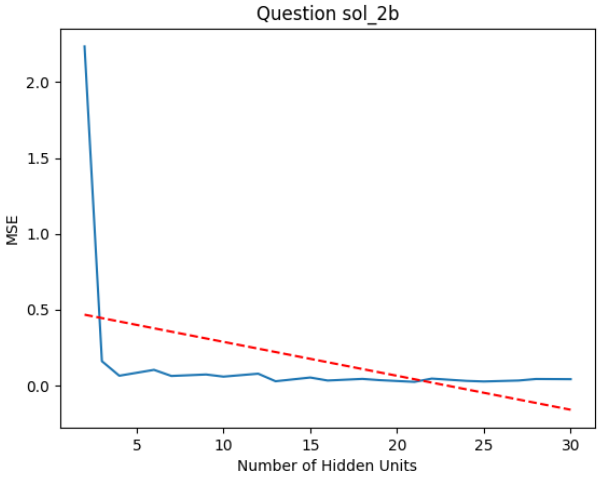


Fig 21: MSE v/s Hidden Units Q2b

due to the high learning rate enabling the gradient to jump around the minima long enough to produce unreasonable behaviour. However, despite the volatility the number of iterations are large enough to guarantee convergence. The number of hidden units for the high dimensional data follows the curve that we got for the scalar case. Hidden units between 15-20 seem ideal for the usecase.

The introduction of the hidden units hyperparameter made the balancing of three hyperparameters more difficult compared to just the Iteration-LR tradeoff. Getting the correct parameter set involved running multiple major experiments like varying LR for fixed iterations and hidden units, varying Iterations keeping other things constant, varying hidden units keeping others constant and other pairwise dependencies.

The scalar 1D data was simpler to finetune than the high dimensional data. This is because the 2D data produced MSE values that were more volatile to change in the parameter set. Even though practically, the MSE convergence threshold for the high dimensional case is quite looser when compared to the 1D case, the average variance when looking at the change in MSE when compared to the learning rates is higher. This is surprising because the graph for the learning rate v/s MSE for sol 2a looks more volatile but it can be seen that the dynamic range of the curve is much lower than the equivalent changes for sol 2b. It is also inferable from the curves that the sensitivity to hidden units and iterations does not seem to change much with the dimensionality mostly because I postulate that the chosen number of iterations for consistency in public tests is high enough to guarantee 1D and 2D convergence.

3 General Deep NNs

3.1 Questions

Q1. Test your network with the same training data that you used in Problem 2, using both 1D and higher dimensional data. Experiment with using 3 and 5 hidden layers. Evaluate the accuracy of your solutions in the same way as Problem 2.

Ans1.

The data generation for the 3rd problem is the same as the data generated for the second problem. The scalar data is a sine wave with additive gaussian noise sampled from a standard normal distribution added to it. The multidimensional data is a 2D data which was visualised as in the subsequent figures. It is formed using the above mentioned method of creating a meshgrid and subsequently forming a sinusoidal hyperplane which needs to be recreated by the regression problem.

For Q3a: The architecture used in the experimentation for the training v/s predicted curve is [10,8,6] with a depth of 3 hidden layers. I chose the learning rate to be 0.4 and the number of iterations to be 40000. Using these hyperparameters, I plot the testing data overlaid with the predicted data. Upon inspection of the curves and their corresponding absolute error plots, it is easily shown that the NN learns the sinusoidal regression problem really well. Analysis similar to Q2 can be made regarding the nature of the noise since it looks piecewise gaussian but this will be easier to visualise in the case of the 2D data. The plots for 3 hidden layers versus 5 hidden layers make intuitive sense. For a fixed number of iterations, the NN with the lower number of hidden layers converges to a solution that follows the actual test curve more closely than the NN with the hidden depth 5. However upon experimental observation, it is found that both the NNs satisfy the MSE threshold so there is not much difference in the convergence criterion but on average for the

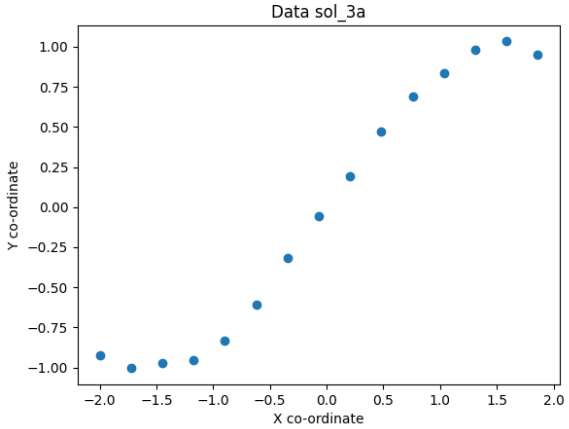


Fig 22: Data 3a

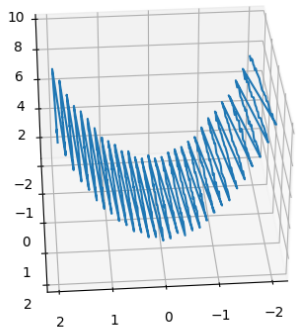


Fig 23: Data 3b

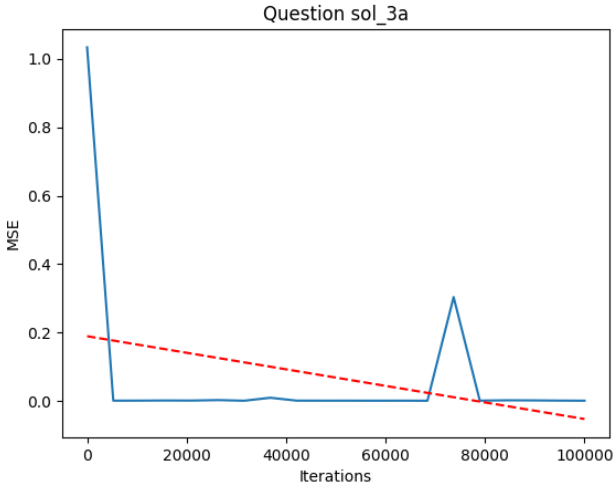


Fig 24: MSE v/s Iter Q3a

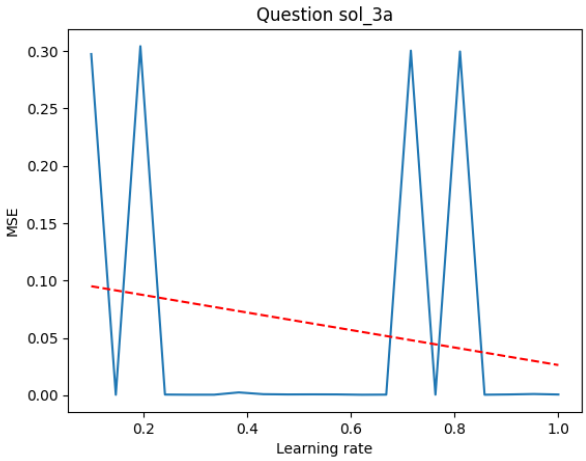


Fig 25: MSE v/s LR Q3a

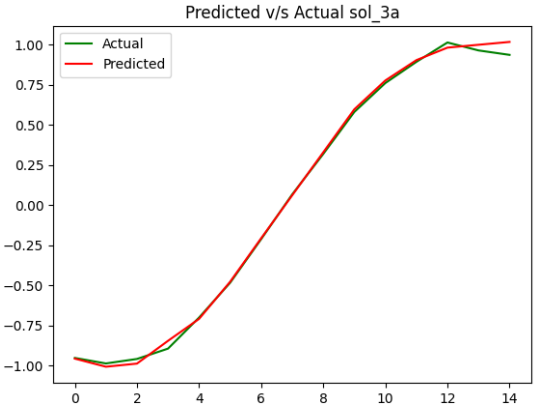


Fig 26: Predicted v/s Testing Q3a

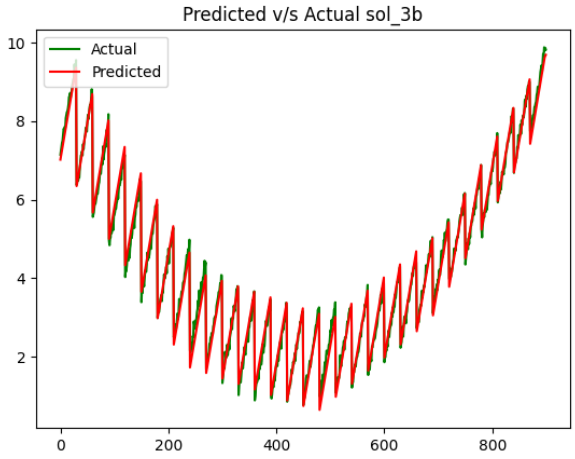


Fig 27: Predicted v/s Testing Q3b

fixed number of iterations, a 3 hidden layer NN outperforms for the 50000 iterations chosen. It has been empirically shown that 3 layer fully connected NNs perform generally well when compared to other depths but this is a factor also dependent on the problem complexity.

For Q3b: The architecture used in the experimentation for the training v/s predicted curve is again [10,8,6] with a depth of 3 hidden layers. I chose the learning rate to be 0.5 and the number of iterations to be 60000. Using these hyperparameters, I plot the testing data overlaid with the predicted data. Upon inspection of the curves and their corresponding absolute error plots, it is easily shown that the NN learns the region of a sinusoidal hyperplane regression problem really well. It is much easier to visualise the nature of the noise in the subproblem as it looks like a mixture of gaussians. Surprisingly, for the higher dimensional problem, 50000 iterations are enough for the regression problem to converge for both the 3 and 5 hidden layer networks. The 5 hidden layer network converges so well as to outperform the three hidden layer network as well as shown in the graph. This implies that the problem complexity is equal to or less than the capacity of the 5 hidden layer network and the model does not overfit.

Q2. Play around with different choices of hyperparameters. Based on your experience, Do you think it is easier or harder to choose effective hyperparameters for a deeper network than for the shallow network? Explain?

Ans2.

For both the subproblems, random seeds create a lot of issues for the 3rd problem, this is perhaps the only problem where the inconsistency of the seed behaviour creates a visible and noticeable change. This is also enhanced by the fact that there are visible discontinuities in the parameters I sweep for. For these reasons, I will attempt to analyse the general trends rather than try to explain the momentary transient characteristics.

Solving the Hidden Layer/ Hidden unit problem for deep general NNs is a difficult problem. There is no theoretical proof for how many hidden units are adequate for a problem and will lead to good generalisation. I created a framework that allowed me to analyse the effect of changing hidden layers and units efficiently. I created a nested loop for hidden layers and hidden units. I selected to run my experiments on three sets of hidden layer depth i.e 3,4 and 5 layers. For each of these architectures I created an architecture that is empirically and theoretically sound. I started with the initial layer having from 50-100 nodes. Using a factor of 0.6, I decreased the number of hidden units in the second layer by almost half and thus created a geometric progression for each starting hidden unit the loop iterated over. Thus if the hidden layer depth was 5, the starting hidden units which I refer to as hidden unit max is 50 in the first iteration of the nested loop. The next layer would thus have 50*0.6=30 hidden nodes. The subsequent layer would then have 30*0.6=18 and so on and so forth. Since I ran this experiment over 3,4 and 5 layers, the situation where the hidden nodes would be 0 was never encountered.

For Q3a: For an increasing number of iterations, it is seen that there is a region of sweep of iterations between 20,000-60,000 iterations that result in a consistent test MSE convergence. The behaviour between 65,000 to 80,000 iterations is assumed to be a random seed error and therefore there is a net decrease in the MSE the more iterations we do for the fixed learning rate of 0.5. This is in accordance with previous questions and our earlier analysis. The corresponding public test pass/fail has also been plotted for convenience. For a fixed iteration set of 50,000 iterations (the number was chosen to enable effective computation for both the sub problems due to consistency issues), LRs between 0.2 and 0.6 proved to be the most conducive for the MSE convergence to take

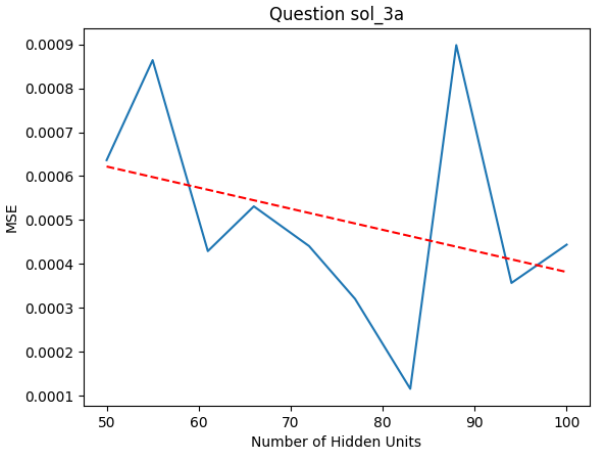


Fig 28: MSE v/s Hidden Unit Max 3 Layer

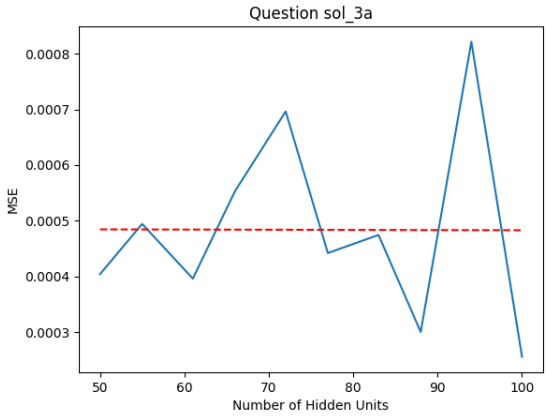


Fig 29: MSE v/s Hidden Unit Max 4 Layer

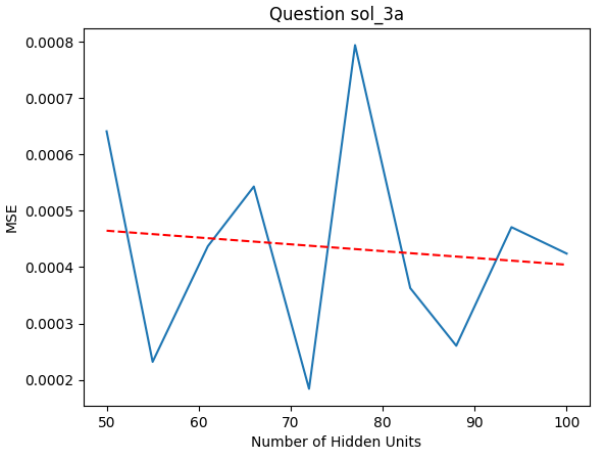


Fig 30: MSE v/s Hidden Unit Max 5 Layer

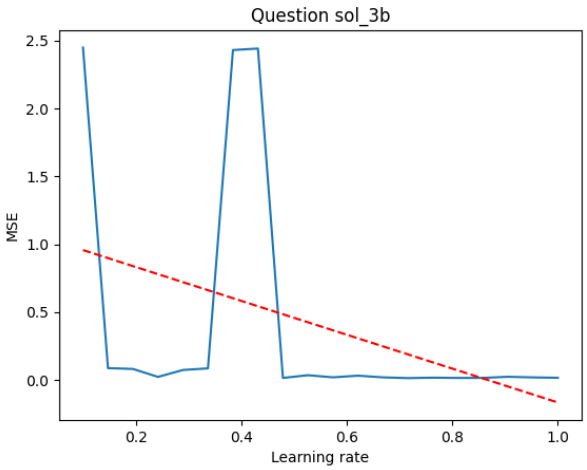


Fig 31: MSE v/s LR Q3b

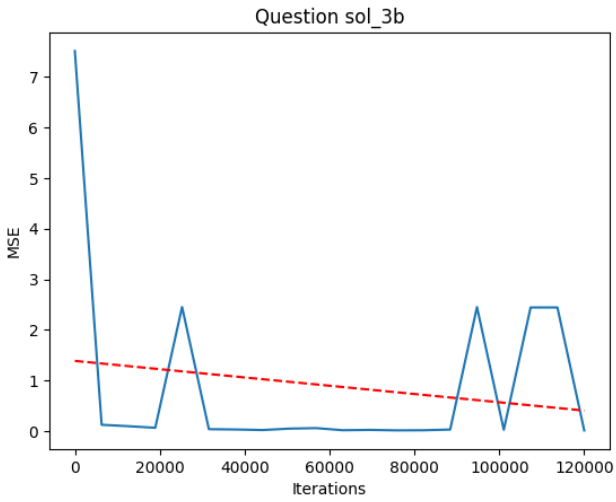


Fig 32: MS v/s Iter Q3b

place. These LRs lead to an MSE of 0.1 which is two orders of magnitudes lesser than the regressor value. Consequently the pass/fail test also passes 75 % of the time with the 25 % being attributed to the random seed errors.

From the curve, it is easily justifiable through the regression line for the 3 layer NN that there is a net negative slope and therefore there is a decrease in the MSE as we increase the number of hidden units. The curve is pretty volatile but since the corresponding variance in the MSE is not orders of magnitudes larger in between two observations, the trend line is an adequate curve to analyse. For a 4 layered NN, with the same number of iterations (50000) the curve seems to be invariant/unbiased to the increase in the number of hidden units. One possible reason for this happening is the fact that the given number of iterations is not high enough for the 4 layer NN to converge/learn to generalise well. This however is a weird postulation since the 5 layer NN seems to generalise well with the given number of iterations and it can be seen that there is a small but net negative trend between the MSE and the number of hidden units for the 5 layer NN.

It is also observed (part of graph attached) that the 3 layered NN has the least minimum MSE out of all the architectures with an increase in the hidden units leading to a decreasing MSE and leading to better accuracies.

For Q3b: For an increasing number of iterations, it is seen that there is a region of sweep of iterations between 20,000-80,000 iterations that result in a consistent test MSE convergence. The behaviour below 20,000 iterations and above 80,000 iterations is assumed to be a random seed error and therefore there is a net decrease in the MSE the more iterations we do for the fixed learning rate of 0.5. It is noticeable that the problem converges for a higher iteration threshold than the scalar case. It is also noticeable from the pass/fail analysis that the 3b solution remains more consistent for a wider region of iterations. For a fixed iteration set of 50,000 iterations (the number was chosen to enable effective computation for both the sub problems due to consistency issues), LRs between 0.45 and 1 proved to be the most conducive for the MSE convergence to take place with these LRs being mostly insensitive to the increase. The MSE remaining identical implies that the problem is solvable enough for all of these LRs to achieve minima in the desired set iterations. These LRs lead to an MSE of 0.1 which is two orders of magnitudes lesser than the regressor value. Consequently the pass/fail test also passes 75 % of the time with the 25 % being attributed to the random seed errors. The curves for the high dimensional problem are quite similar to the curves for the scalar problem. For both the 3 and the 5 hidden layers network, we can see a very visible net negative trend in the MSE when compared to the number of maximum hidden units we start with. Correspondingly for the given LR=0.5 and Iterations =50000, I also visualise the minimum MSE encountered over 3,4 and 5 layer NNs. I find that the 3 hidden layers once again achieves the minimum out of all the architecture types. The 4 layer neural network seems to be an outlier with the MSE increasing with an increase in the number of maximum hidden units. I assume this because of the separability in the high dimensional space but again its hard to justify it since the problem seems to be solvable in the set hyperparameters by both the 3 and 5 hidden layer networks.

It is tougher to choose the parameters for the deeper network when compared to the shallow network purely based on the number of parameters that need to be sweeped. Fixing the LR and iterations by themselves are hard enough to balance but finding a convergent condition with a set architecture adds to the complexity even more. Due to the framework I outlined before, I was able to do a good parameter sweep to analyse the variables and their respective sensitivities well and using the values of LR and Iterations of the previous questions made things simpler. The sheer number of permutations needed to balance the MSE meeting the threshold set is an important deciding factor that makes the approach towards deeper networks convoluted and paved the way

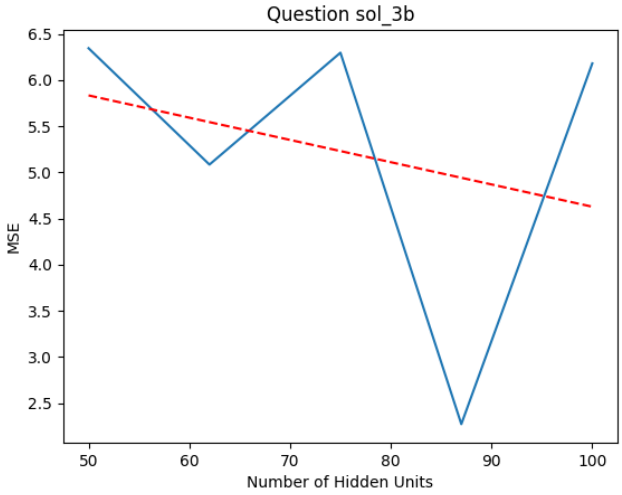


Fig 32: MSE v/s Hidden Units Q3b 3 layers

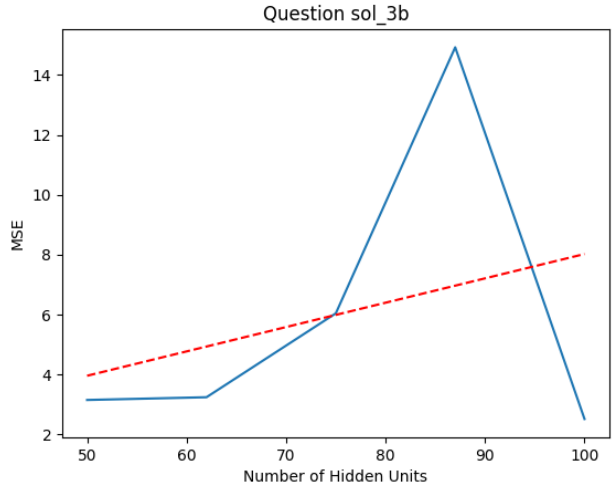


Fig 33: MSE v/s Hidden Units Q3b 4 layers

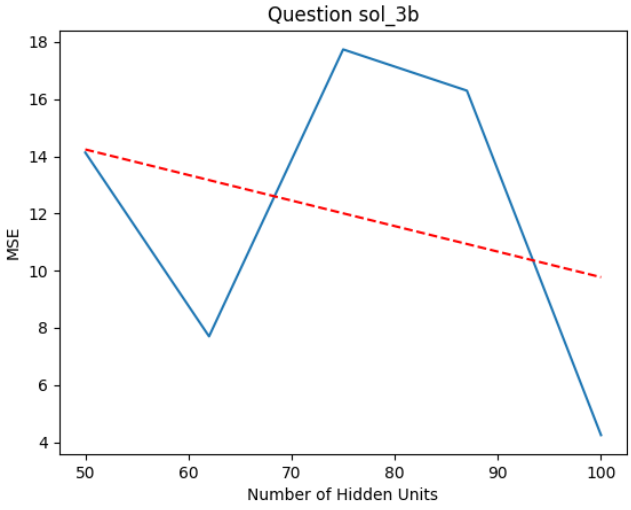


Fig 34: MSE v/s Hidden Units Q3b 5 layers

Depth Analysis:

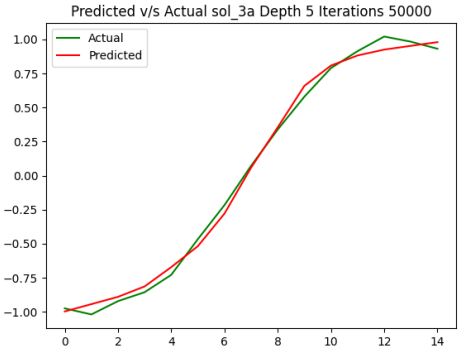


Fig 35: Q3a Predicted v/s Actual Depth 5

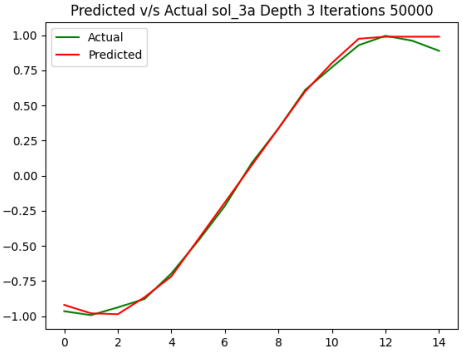


Fig 36: Q3a Predicted v/s Actual Depth 3

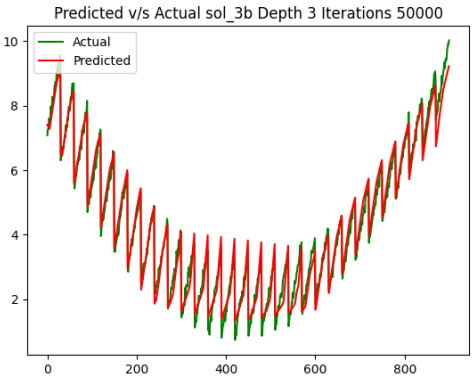


Fig 37: Q3b Predicted v/s Actual Depth 3

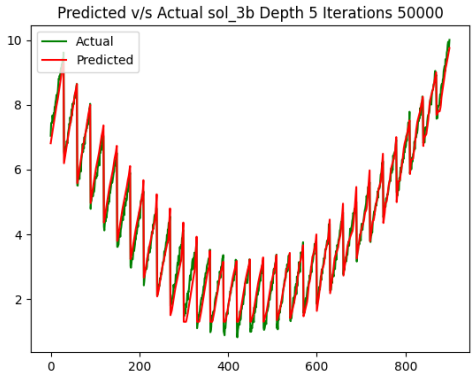
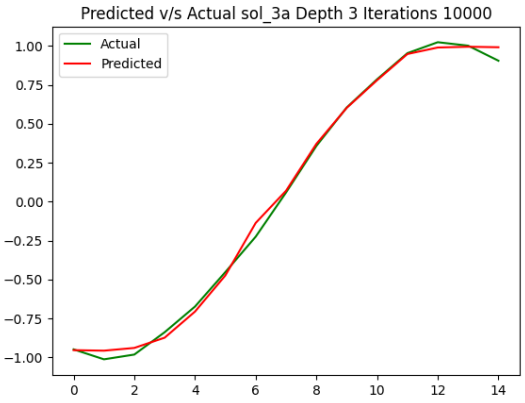


Fig 38: Q3b Predicted v/s Actual Depth 5

Convergence Analysis:



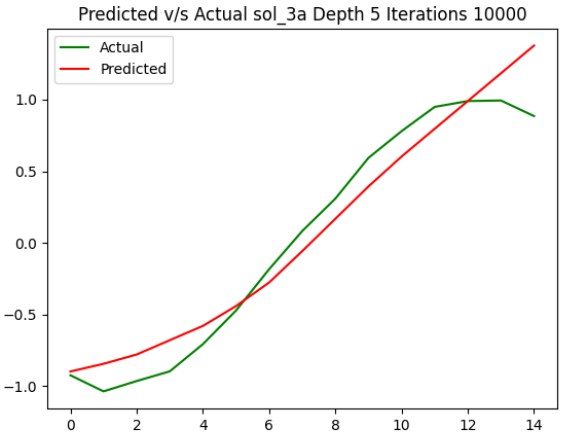
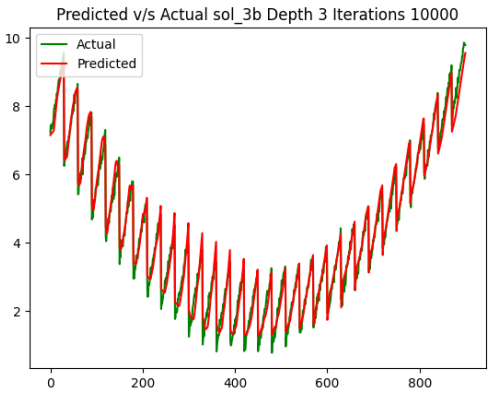


Fig 39 and 40: Comparison between the predicted data at Depth 3 and Depth 5 at 10,000 iterations. Notice the faster convergence of the shallower network



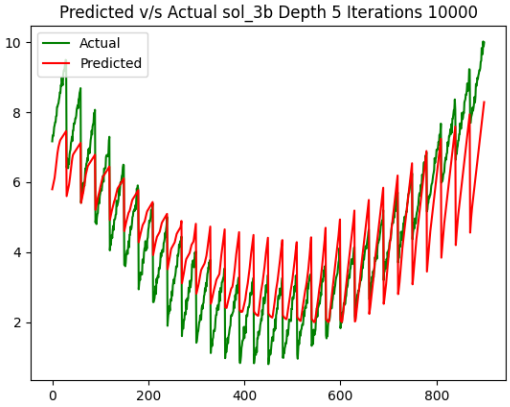


Fig 41 and 42: Comparison between the predicted data at Depth 3 and Depth 5 at 10,000 iterations for Q3b. Notice the faster convergence of the shallower network

forward for changes in the structures of the NN to things like CNNs and RNNs.

Q3. Do some experiments to determine whether the depth of a network has any significant effect on how quickly your network can converge to a good solution. Include at least one plot to justify your conclusions.

Ans3.

For analysing the relation between the depth and the convergence rate, I plotted the predicted function versus the actual test data. As we can see for a fixed iteration number of 10000, the three layer network tracks the test sinusoidal function really closely. Conversely, we can see that for the 5 layer network, the predicted function is closer to a line rather than a sinusoidal function.

Similarly in the problem 3b, we can see that the 3 layer NN at 10000 iterations generalises well and gives the sawtooth sinusoidal which is almost exactly overlaid on the original plot whereas the 5 layer NN can't train completely in the set amount of iterations (duration). These plots show that the rate of convergence is definitely dependent on the depth of the network. In fact during empirical observations it was found that the 3 layer NN converges consistently under 40,000 iterations whereas the 5 layer NN takes atleast 60,000 iterations to meet the MSE bound but even then it cant match the 3 layer NN test MSE.

4 General Deep NNs for Classification

4.1 Questions

Q1. Test your network with two 1D problems; that is, your input is just a scalar. First, choose two sets of points that are linearly separable. Vary the margin between the points and the number of layers in the network. Is it more difficult to find hyperparameters that solve a problem with a smaller margin? Does the speed with which the network converges to a good solution depend on the margin? Include a plot to support your answer to the second question. Second, test your network with some 1D data that is not linearly separable. What differences do you observe?

Ans1.

The data generation of the 1D data was not a very complicated process. I create two dummy variables x1 and x2. x1 is initialised as a linspace between -2 and 2 and 30 points in this range are stored to x1. These points will constitute one class. x2, the second dummy variable can be initialised as the data we will use for the second label. The x2 can be chosen as a linspace between 2+margin to 2+4+margin to still maintain the effective spread of the dummy variable but enable us to vary the level of separation between the two classes. We then concatenate the two respective dummy variables to create one scalar composite dataset. Labels corresponding to the corresponding dataset can be created using a decision hyperplane of x¿2 and this will in essence create the labels for x, the composite dataset, based on the ranges of x1 and x2, our dummy variables.

For the non separable case, the 1D data can be created by initialising one dummy variable x instead of the two required for the previous case. Now instead of choosing the labels to be dependent on a linear hyperplane, to be dependent on a quadratic equation. I use the expression

$$x^2 > 0$$

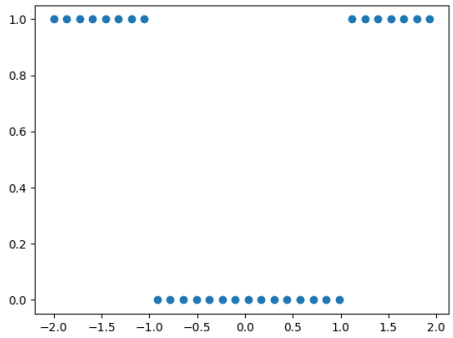


Figure 43: Non Sep 4a data

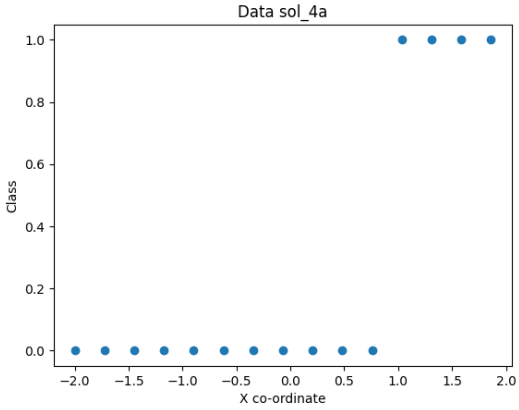


Figure 44: Separable 4a data

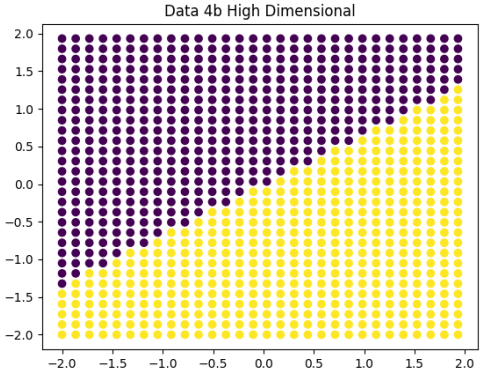


Fig 45: Meshgrid 4b data

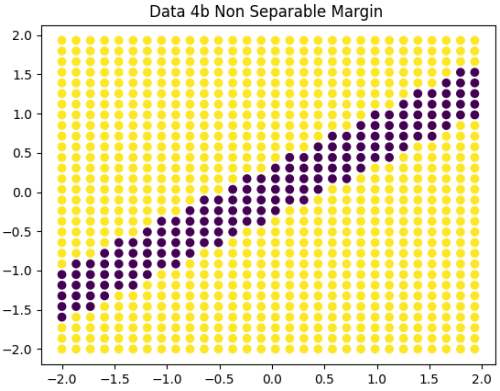


Fig 46: Non separable meshgrid 4b

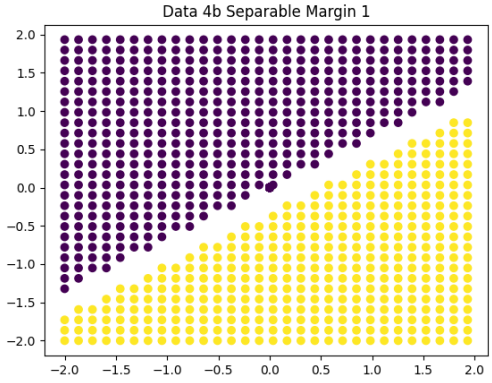


Fig 47: Margin =1 Separable mesh grid 4b data

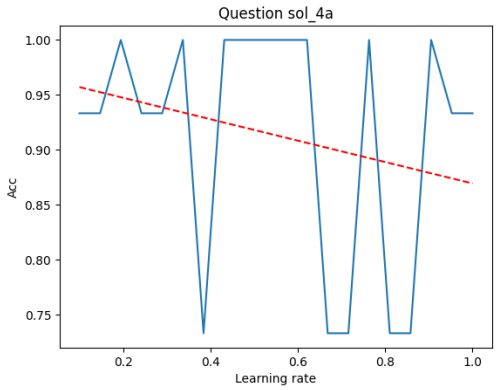


Fig 48: MSE v/s LR Q4a

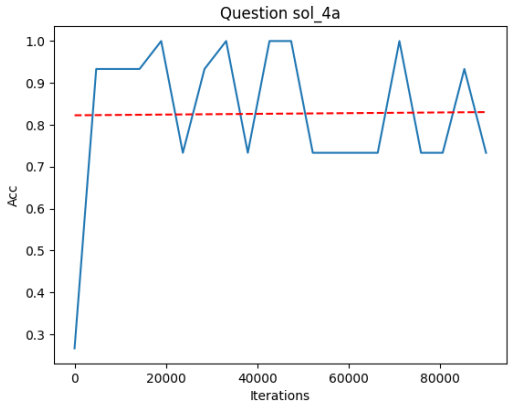


Fig 49: MSE v/s Iter Q4a

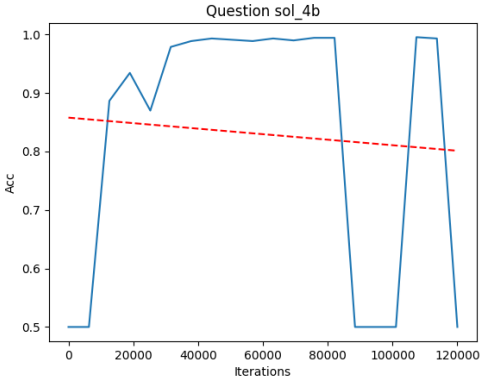


Fig 50: MSE v/s Iter Q4b

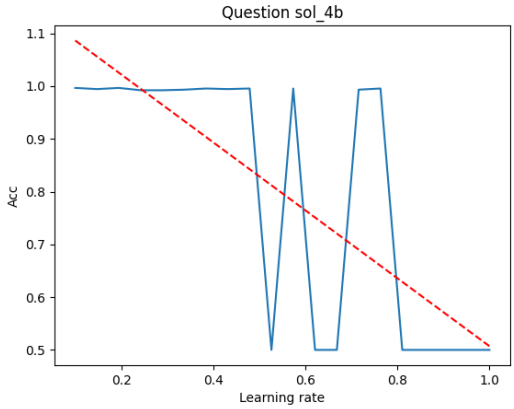
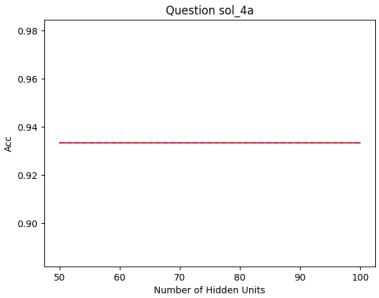
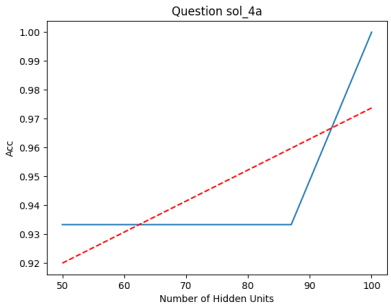
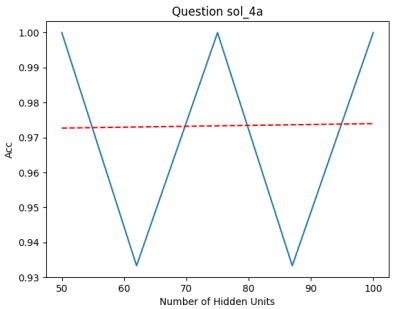


Fig 51: MSE v/s LR Q4b







Figs: Accuracy v/s Hidden Units Max for 3 layers, 4 layers and 5 layers respectively

which created a 1D XOR problem and is an interesting problem I would analyse.

For the first parameter sweep, I do a thorough analysis without taking into account the separability of the problem. I run the program and compute the accuracy versus different learning rates. For a fixed iteration number of 200000 (large value taken to guarantee consistency), the learning rates between 0.4 to 0.6 seem most conducive to the convergence. In addition, they guarantee perfect fit onto the testing dataset. Generally the trend is hard to decipher but from the LR pass/fail curve, it can be seen that these values are outliers and not necessarily a signifier of the overall trend. Most LR indices are generating accuracy values above 0.95 with the range between 0.4-0.6 being the best with consistent 1 accuracy values.

When deciphering the iteration curves, we can see that with an increase in the number of iterations, initially the accuracy increases but pretty fast it starts to converge to values between 0.95 and 1. It is also visualised that at high iterations, the random seed problem becomes more and more common with tests above 60,000 failing really easily with accuracies of 0.75. However this behaviour stops at iteration values above 100,000 with more consistent converging properties arising.

Due to the large parameter space and the high number of iterations required for the stuff to run consistently, I needed to restrict my hidden unit/ hidden layer sweep due to timing constraints. For each hidden layer in [3,4,5], I now only iterate over 5 points between 50-100 to set as the hidden unit max. I still use the same logic of creating the geometric series to identify a possible working architecture. Unfortunately, due to the restrictions imposed by the high number of iterations, there is an intrinsic inability to find any general trends. However, it does allow me to identify structures for each of the hidden layer iterators. For 3 hidden layers, the network consistently converges to 93.33% accuracy for all the architectures I iterate over. For a 4 layered network, the NN even achieves accuracy of 1 when the max hidden units equal 100. For a 5 layered network however there is very volatile behaviour and its hard to understand a general trend. However it can be seen that for the chosen LR and Iteration bound, the system converges and periodically goes back and forth between 0.94 to 1 accuracy. Thus our framework is still good enough for finding useful architectures even if the sweep is constrained.

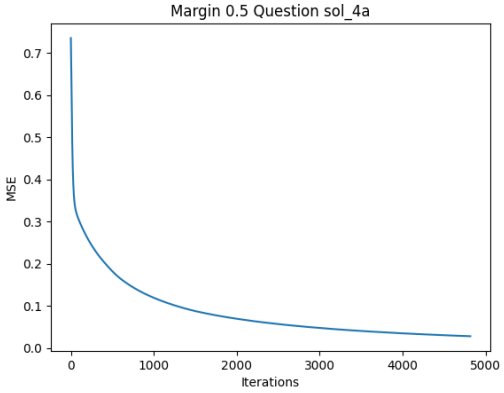
Due to the constrained sweep set, random seed issues and divide by zero warning along with tough decision regions chosen in the data generation scheme, the problem is harder to choose a consistent hyperparameter set for.

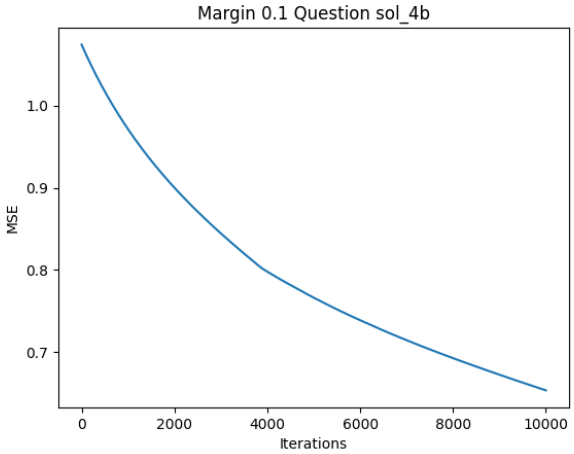
The speed with which the network converges is definitely dependent on the margin. To visualise this I plot curves that plot the training losses v/s the choice of the total number of iterations. Over a span of 10,000 iterations, it is observed that the NNs trained on larger margins generally exhibit faster convergence to a set MSE threshold. The converse accuracy threshold is also satisfied. We can see that for the set margin of 0.5 the curve is the flattest signifying lesser negative slope than when margin is 2 which consequently has lesser negative slope than when the linearly margin slope has margin 5. All this is clearly signifying a general trend in decrease in time to convergence with an increase in margin possibly due to the geometric simplification and stronger decision boundaries being learnt.

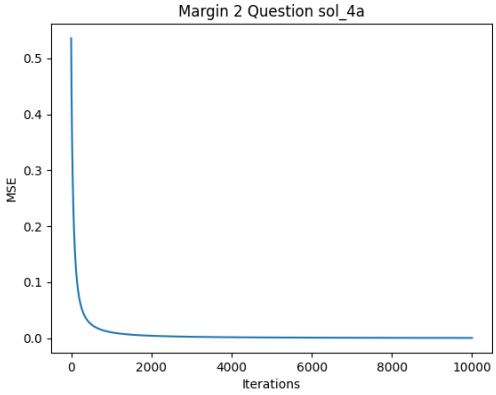
When experimenting with non linearly separable data, we see that the MSE converges for the 1D XOR problem. However it is easily found that the convergence for the non linear problem takes much longer than the linearly separable data. Also the non separable data exhibits more scattered behaviour with some iterations leading to diverging behaviour really fast. I postulate this is because of the strong decision boundary the NN needs to learn in order to model the behaviour exhibited by the data.

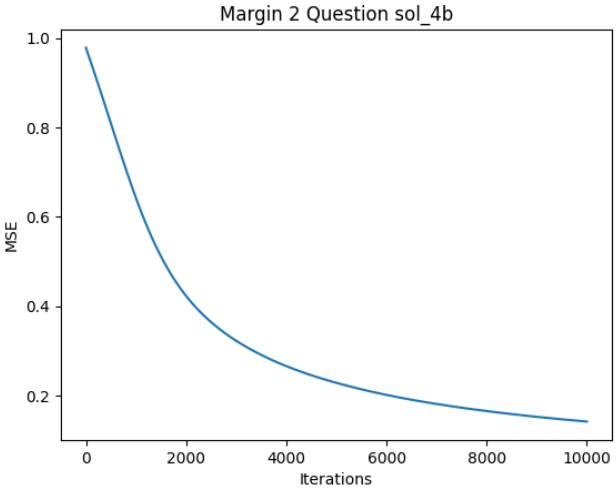
Convergence Analysis:

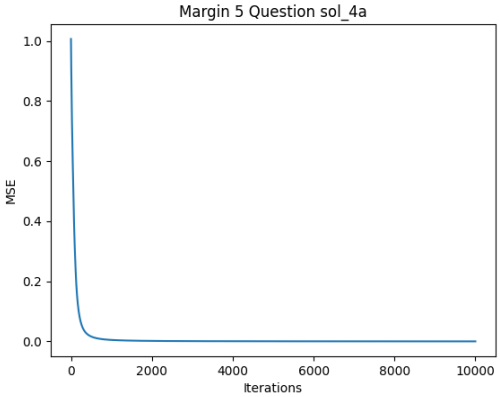
Fig 52,53,54,55,56: Faster convergence with bigger margins Q4a and Q4b

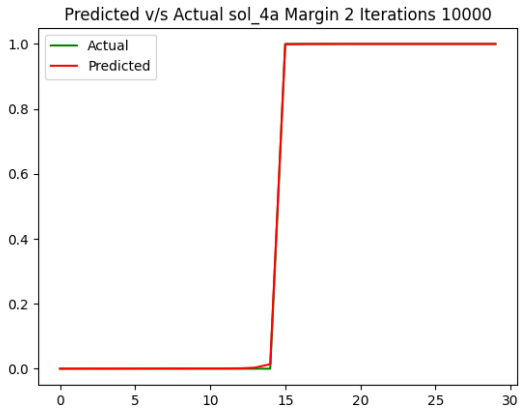


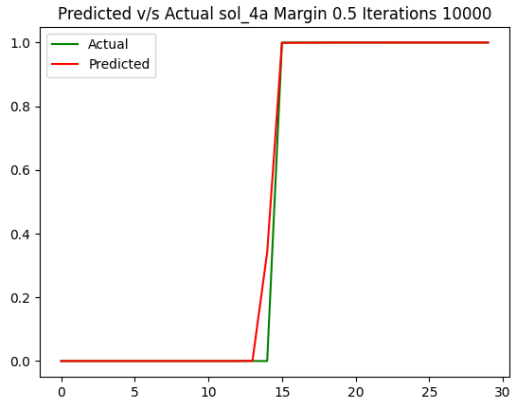












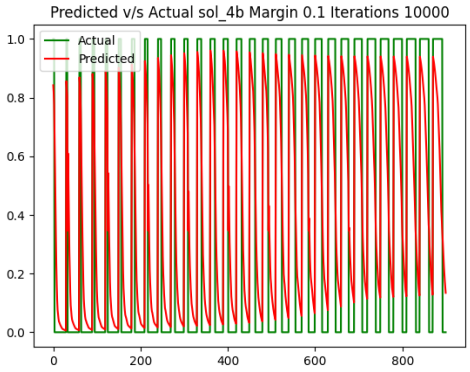
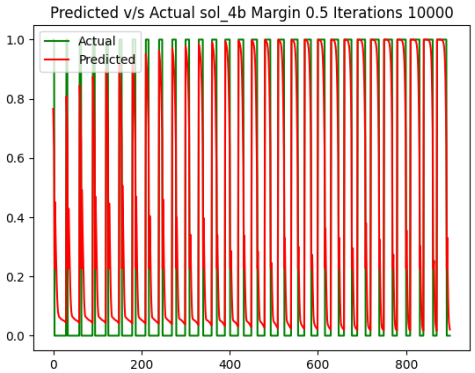
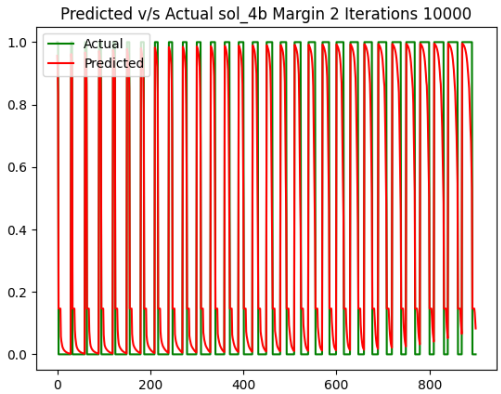
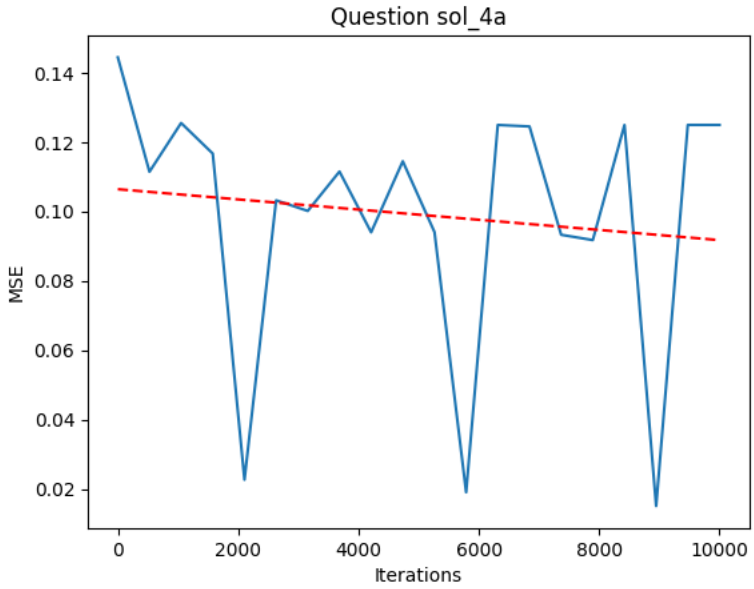


Fig 57,58: Bigger Margins lead to more confident decision regions and therefore more confident logistic evaluations Q4a

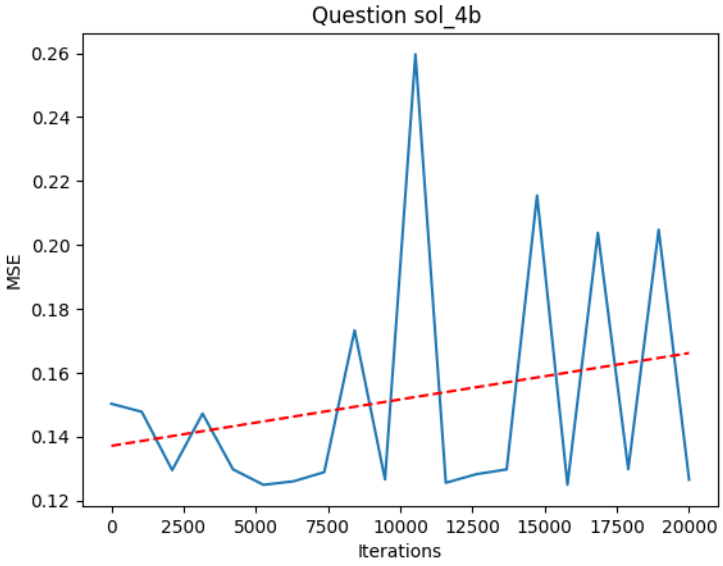
Fig 59,60,61: Bigger Margins lead to more confident decision regions and therefore more confident logistic evaluations Q4b







Figs 62,63: Difficulty in learning XOR problems with low margins (Non-Separability)



Q2. Repeat these experiments, using higher dimensional data. Again, select data that is linearly separable, and then select data that is not linearly separable. Include data to support your conclusions.

Ans2.

For the high dimensional case, the curves mostly follow a similar trend. As we increase the LRs, for a given iteration size of 60,000, the accuracy decreases from as high as 1 to as low as 0.5. Part of this can be attributed to the random seed errors but the curve implies a stronger behaviour than the graphs seen before. For iterations, the number of iterations between 40,000 and 60,000 seem to be the most stable and achieve accuracies of 1. A parameter sweep for Q4b for the hidden layer and units under the preconstructed framework was done and it led to similar results as in the scalar part so the architectures from 4a were chosen as is. The selection of hyperparameters is definitely more difficult in problem 4 but since the parameter sweep for Q4a allows us to take reasonably good architectures directly, the problem is simpler than it should be but due to the constrained parameter set and random seed issues and divide by zero warning along with tough decision regions chosen in the data generation scheme, the problem is harder than the previous ones to choose a consistent hyperparameter set for.

The multi dimensional data problem follows the trend set by the scalar problem. With an increase in the margin between the two classes, the MSE decreases and the accuracy increases. This is seen very easily using the curves plotted below. For a lower margin, the curve is flatter and for a higher margin the curve moves leftwards and converges faster symbolising an inverse proportionality between the margin and the number of iterations needed for the convergence MSE to be satisfied.

The 2D XOR problem is a very difficult one to solve for the parameter set selected. In fact in some test runs the tests diverge really fast and lead to the NN not training and thus not generalising well on the test set. It is also seen through the given curves that the nonlinear boundary in general takes more runs and iterations to converge.

5 MLP's on MNIST

5.1 General Discussion

The MNIST problem is really straightforward. The only problem of concern is the mini batching for the SGD solver. There are only two possible options where the batching can be done: the train() function and train step() function. I create a permutation of the whole training set that I load and shuffle the entire dataset into the desired variables. Further, I go on to index into batches of the selected batch size sequentially so that I can access the whole jumbled dataset over an epoch. The technical aspects of the code are far less interesting. The network works with a wide variety of architecture as tabulated below. I run the 5 layered NN for 10 epochs with a learning rate of 0.4 to make it converge while surprisingly the 3 layered NNs take longer to converge for some reason and they need to be run for at least 25 epochs for the threshold limit in accuracy to be passed.

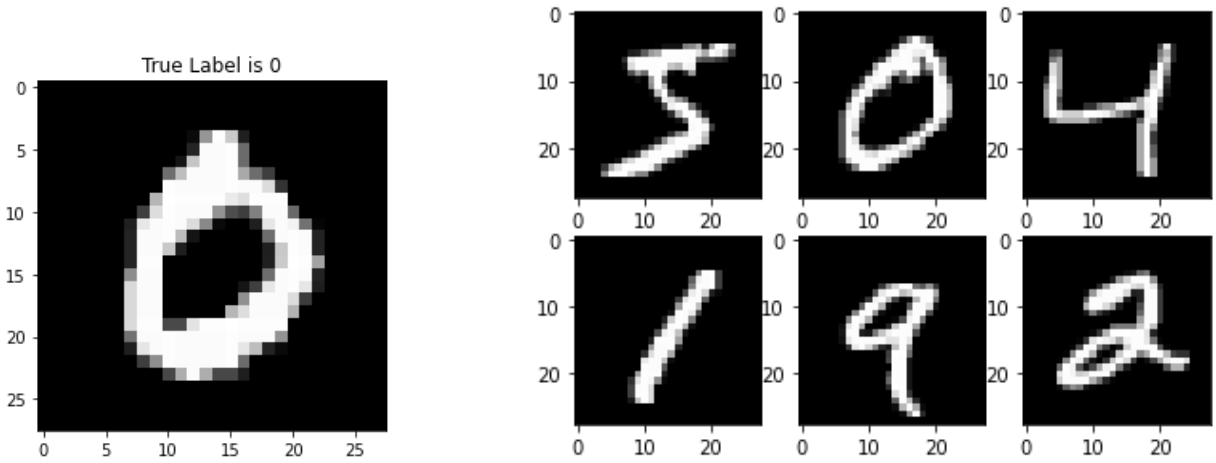


Fig 64 and 65: MNIST Data visualization

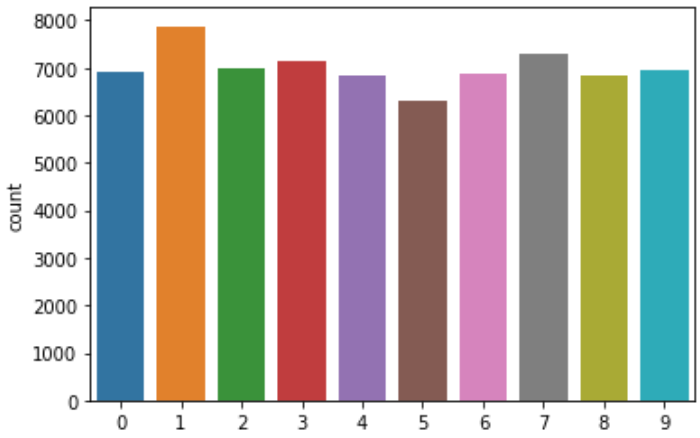


Fig 66: Class balance check using Histogram

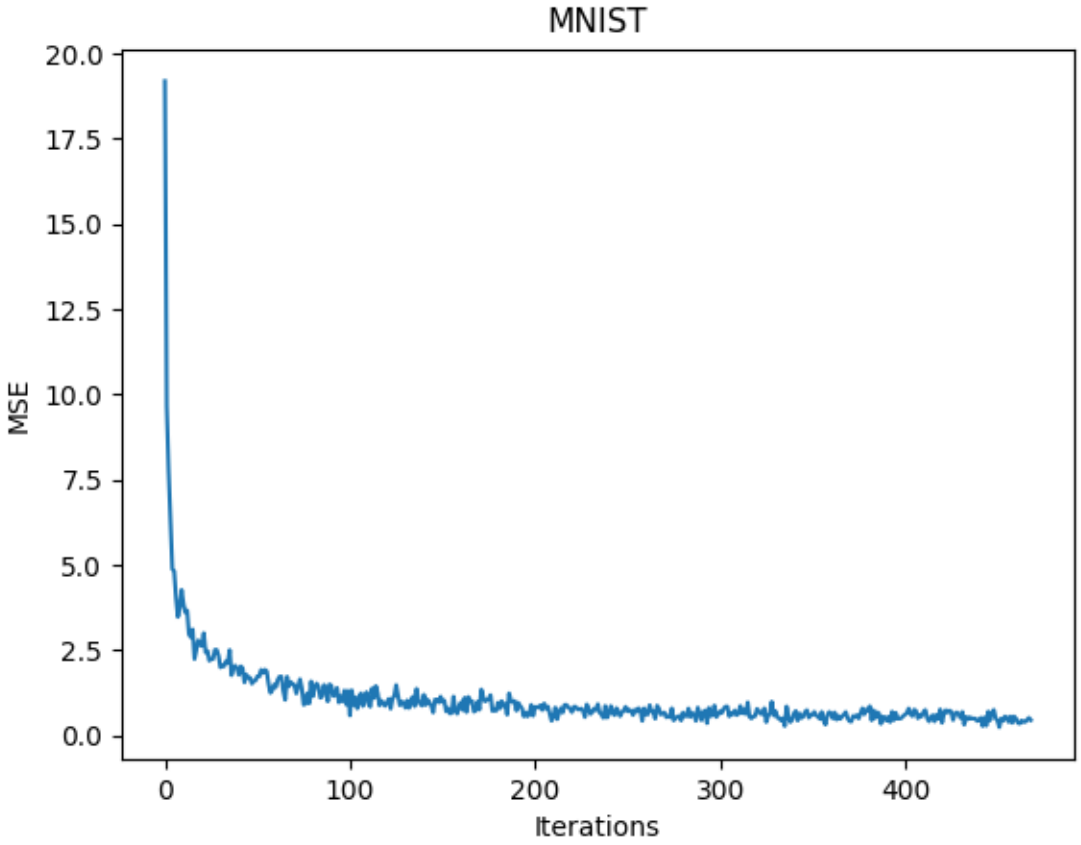


Table 1: MNIST Accuracy List v/s Architecture

Architecture	Accuracy
784,120,100,100,80,80,10	96.05
784,50,50,50,50,50,10	94.99
784,32,32,32,32,32,10	94.06
784,64,64,64,64,64,10	94.74
784,32,32,32,10	90.81
784,50,50,50,10	93.37

6 Appendix

Here I will talk a little more about the loss modules that are implemented in layers that are used in the neural network architectures I create for each problem.

- Cross Entropy I calculate the Cross Entropy loss using the conventional equation of the product of the true probability distribution and the models predicted probability distribution summed over all classes. The equation is as follows:

$$-(y \log(p) + (1-y) \log(1-p))$$

and it can more generally be evaluated as

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

- Sigmoid I implement both the naive and the numerically stable Sigmoid. The expression for the naive sigmoid is straight forward and is as follows:

$$S(z) = \frac{1}{1+e^{-z}}$$

For the numerically stable sigmoid, the expression changes to

$$S(z) = \frac{e^{-z}}{1+e^{-z}}$$

so that we dont exponentiate an arbitrarily large positive number.

- Cross Entropy and Softmax combined

I create a combined cross entropy softmax layer based on the following equation:

$$-\log\left(\frac{\exp(x_k)}{\sum_i \exp(x_i)}\right) = -x_k + \log(\sum_i \exp(x_i)).$$

- Square Loss

The implemented square loss is the following equation

$$l = \sum_{i=1}^{n} (y_i - \bar{y})^2$$