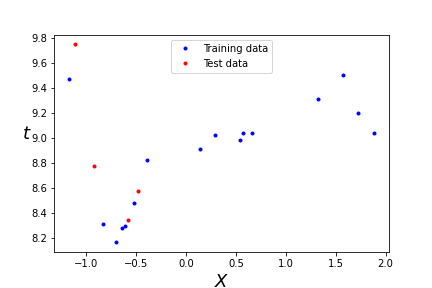
ELL409   
Assignment 1

Vansh Gupta  
2019EE10143

**Part 1A**

* Using only 20 data points

  
Fig 1. Randomly sampled 20 data points

* Using Moore-Penrose pseudoinverse:

I run my code for different values of (regularization constant) and epochs (number of iterations). [Fig 2.]

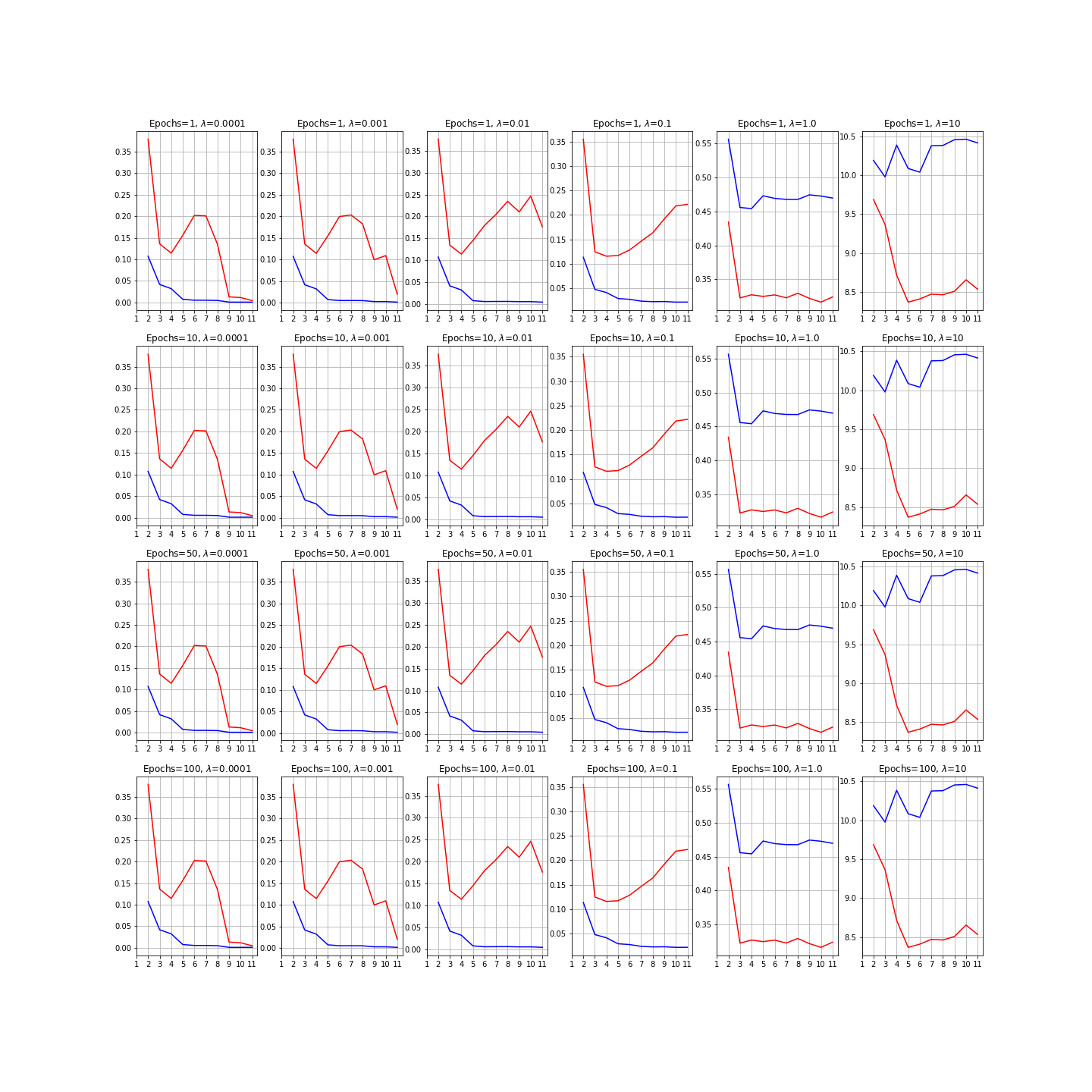
After that, I observed that the number of epochs have no effect on the losses, which is also expected since our dataset is very small for the model to have to learn the same data point again. Thus, I iterate over different . We can also see that while the training error has an elbow point at m=5 (m is the highest degree of polynomial), the test set is truly reduced for m=9. Thus, for further iterations, I will fix m at 9. [Fig 3]

Once again, we see that since number of data points is only 20, there is no over-fitting in the first place that should be fixed with regularization

A final plot for = 0, number of epochs = 1, over different values of m is shown in Fig. 4

Therefore, the best guess of the polynomial, for m = 9 and = 0 is:

8.99654324 0.32089103 0.30898259**2** 4.58312801**3** 3.91859305 6.12614492**5** 7.53175079**6** 0.885168573.20999694**8** 0.88288049

  
Fig 2. The red line depicts the test error and the blue line shows the training error

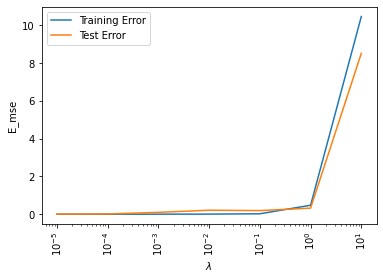
  
Fig 3.

  
Fig 4.

* Using Gradient Descent:

I followed the exact same procedure as listed for the Moore-Penrose pseudoinverse.

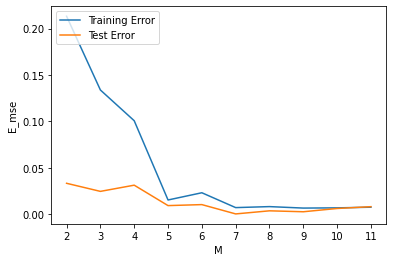
The inference for was absolute and not related to the method, therefore for this set of experiments, I keep to be 0. I also keep my epochs i.e., the number of iterations to be at a fixed value of 1500. This is because the data is small, and for gradient descent which does not produce a closed form solution, needs to go over the data multiple times to not succumb to underfitting

Thus, here my main focus is on the maximum degree of the polynomial (which should be the same), the learning rate and the size of our mini-batch.

First, for the learning rate, after trying various different values for , I came to the conclusion that there is no single good value for the learning rate and that it needs to change with time. Thus, once again after extensive experimentation, I found the value of to work best for the given data. Here, “” is the starting index for a certain mini-batch.

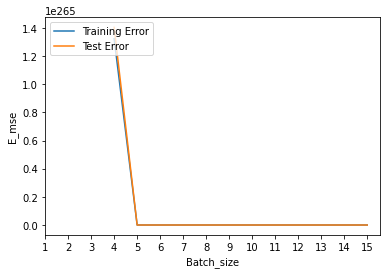
For the order of polynomial, I take number of epochs and learning rate value as described above and do mini-batch gradient descent on it for a batch size of 10. The results are in Fig. 5.

Note: For all models of gradient descent, I have used a min-max normalization followed by subtracting each dimension by its mean for faster and better convergence. The same normalizing parameters (Minimum, maximum, mean of min-max normalized training set) is used to normalize the test set

   
Fig. 6: Blue line is training error while the orange line is test error

This is not something very good, but keeping in mind that there are only 20 data points, we will have to work with it. Highest degree 7 seems good enough for our gradient descent model.

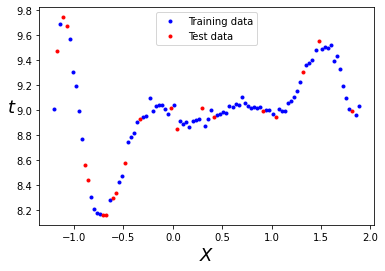
Using 7 as our M, we know vary batch size from 1 to 16 (Size of training set)

  
Fig 6. The error for batch size < 4 was out of bounds

Therefore, our initial batch size value of 10 was good for our final estimate of the polynomial which is:

8.88472159 0.8518583 1.09412644**2** 0.62121061**3** 2.52004379 0.22660583**5** 0.22836758**6** 2.82967896

* Using all 100 data points:

  
Fig. 7

* Using Moore-Penrose pseudoinverse:

I run my code for different values of (regularization constant) and epochs (number of iterations). [Fig 8.]

Once again, we observed that the number of epochs have no effect on the losses. Thus, I iterate over different . We can also see that while the training error has an elbow point at m=5 (m is the highest degree of polynomial), the test set is truly reduced for m=9. Thus, for further iterations, I will fix m at 9. [Fig 9]

This time, we see that since number of data points is 100, there is some improvement with regularization at = 0.01 after which it starts under-fitting.

A final plot for = 0.01, number of epochs = 1, over different values of m is shown in Fig. 10 which confirms m= 9 to be a good basis.

Therefore, the best guess of the polynomial, for m = 9 and = 0.01 is:

9.0140234 0.01824327 0.97251592**2** 2.73209287**3** 1.05359651 4.21682231**5** 4.04789536**6** 0.86240564 1.9421145**8** 0. 51181432

Finally, Fig 10 shows this polynomial on data-points from the dataset

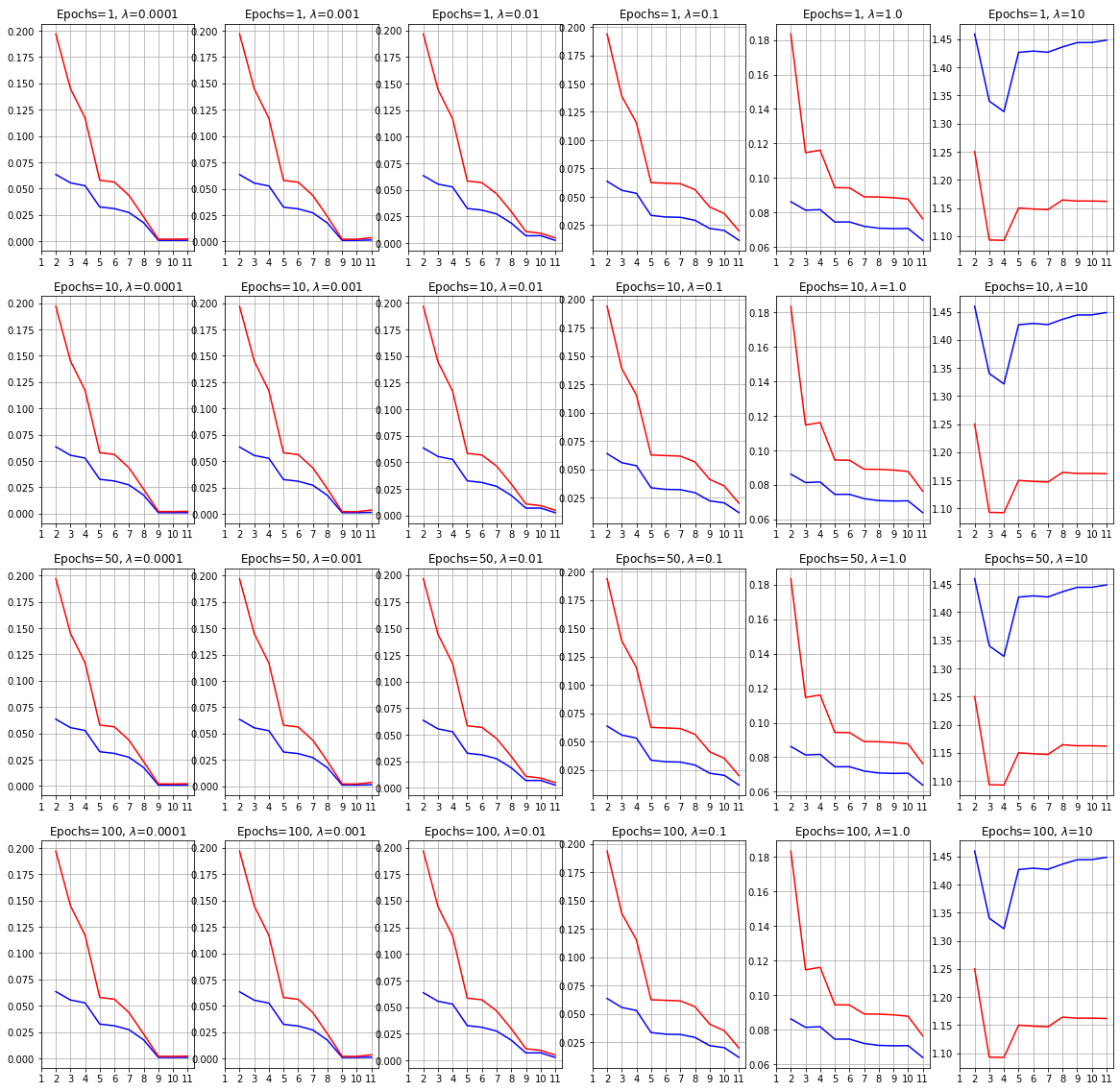
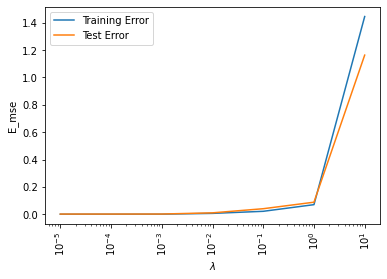
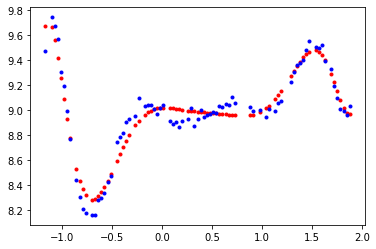
  
Fig 8. The red line depicts the test error and the blue line shows the training error  
  
Fig. 9



Fig 10.

  
Fig 11. Red dots showed the predicted target labels and blue dots are from the dataset

* Using Gradient Descent:

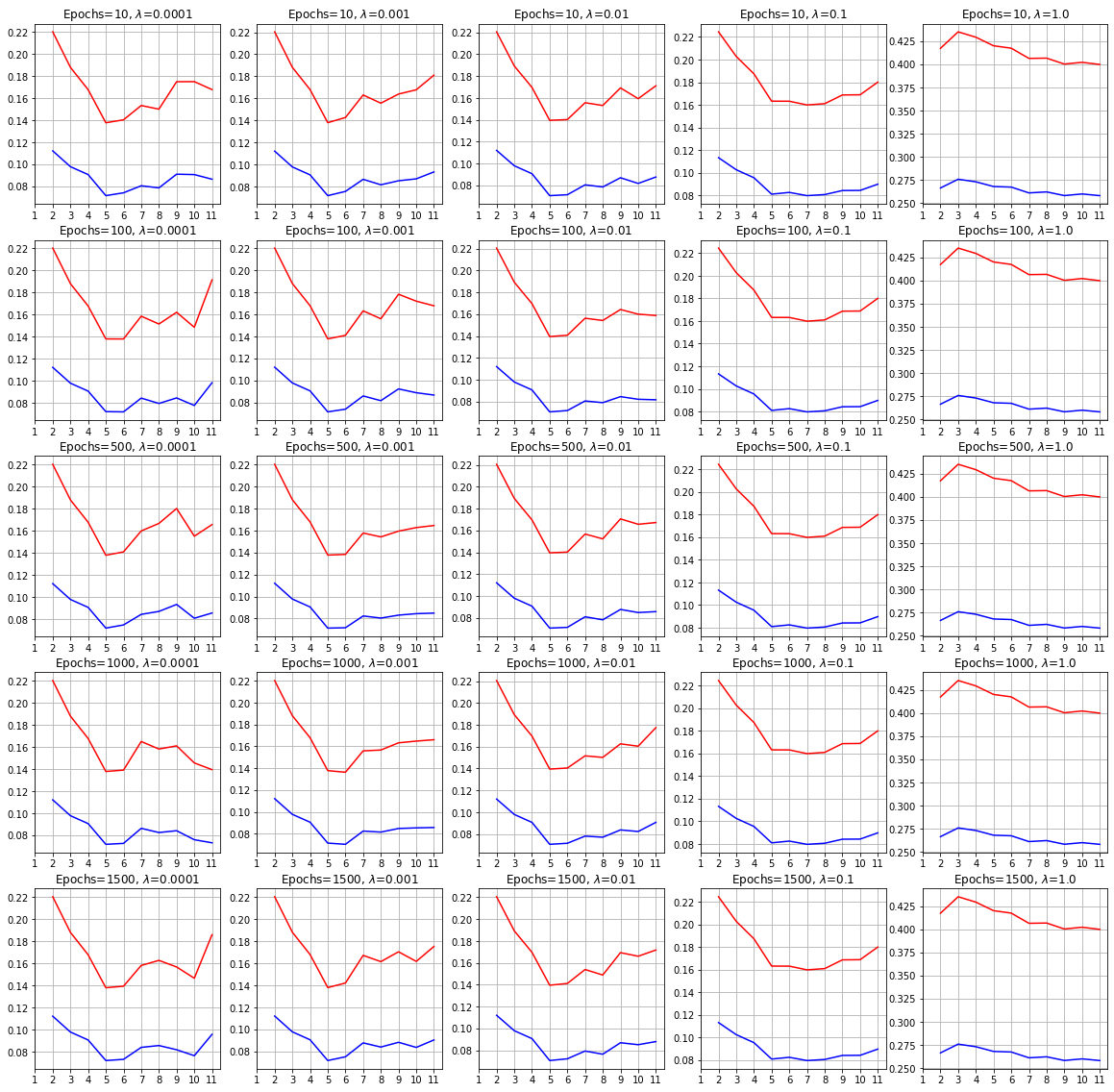
I followed the exact same procedure as listed for the 20 data points. However, I will need to check for appropriate value of as well.

First, for the learning rate, after trying various different values for , I came to the conclusion that there is no single good value for the learning rate and that it needs to change with time. Thus, once again after extensive experimentation, I found the value of to work best for the given data. Here, “” is the starting index for a certain mini-batch.

For the order of polynomial, I take number of epochs and lambda, and do mini-batch gradient descent on it for a batch size of 25. The results are in Fig. 12.

From Fig 12, we infer that the gradient descent seems to approximate the model to its best at M=5. Also, the model seems to perform well for and epochs = 1500

Note: For all models of gradient descent, I have used a min-max normalization followed by subtracting each dimension by its mean for faster and better convergence. The same normalizing parameters (Minimum, maximum, mean of min-max normalized training set) is used to normalize the test set

   
Fig 12.

Next, I plot the errors for different values of batch size from 1 to 80 (Size of training set) [Fig 13.]

Both batch gradient descent and SGD seem to show good performance, but for few values in the middle, the accuracy decreases. Since there is sharp decline around 35, we take that to be our batch size for final evaluation of degree of polynomial [Fig 14.]

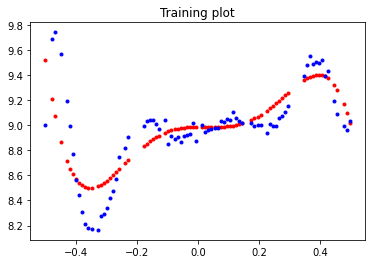
  
Fig 13.

  
Fig 14.

Therefore, our initial batch size value of 5 was good for our final estimate of the polynomial which is (For =0.001, epochs = 1500, batch size = 35):

8.97359663 0.59243457 0. 0.23539727**2** 2.35287087**3** 0.51712917 3.44087613**5**

The corresponding plot is:

  
Fig 15. Red dots are predicted values, blue dots are from dataset

Thus, we conclude that increasing the number of data points gives a much better model prediction. However, for a gradient descent model to work, we would still require much more data points

* Noise Estimate:

I have not calculated noise estimate for each of the 4 models, but only for Moore-Penrose pseudoinverse for all 100 data points

How to estimate variance of the noise?

*Var(Y) = E[(Y – E(Y))2]*

*Where E(Y) is our estimate of f(X)*

= 0.006940360733959994

Therefore, variance of the underlying noise is 0.006940360733959994