Assignment 01: Fitting and generalization of regression models via simulation
Cristopher McIntyre Garcia

PART A: Generate data

The generated datasets are comprised of (x, y) coordinates. The x values are generated using the following commend *'random.uniform(0, 1).'* To generate the y values, we first generated a designated z value using the following commend *'random.gauss(0, math.pow(sigma, 2))'* and proceed to generate the y values with the following commend *'yi = math.cos(2 * math.pi * xi) + zi.'* Here is an example of a generated dataset following these distributions.

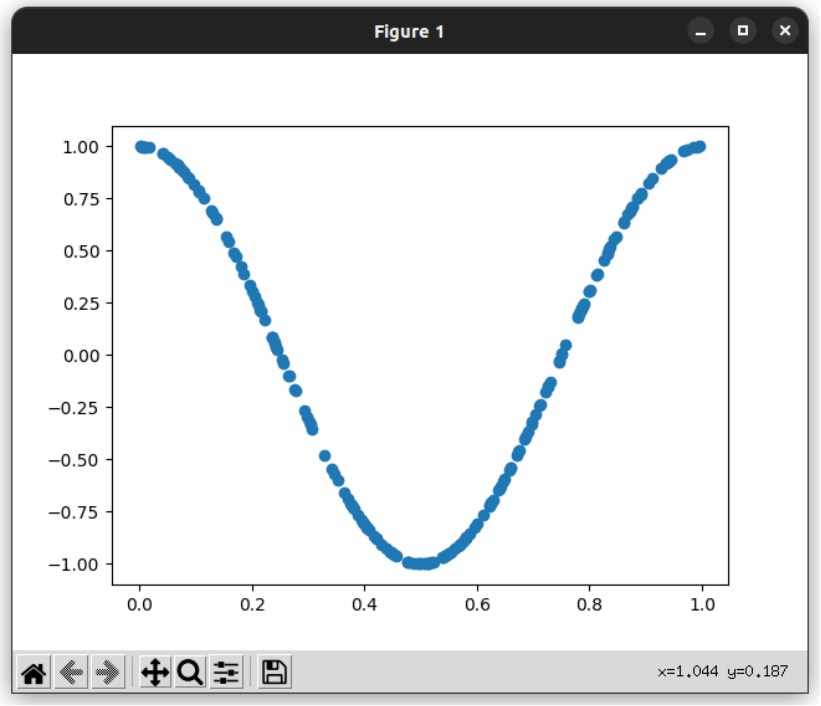

Figure 1. Generated dataset

PART B: Mean Squared Error

The mean squared error is calculated using the following equation:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Equation 1. Mean Squared Error

where n is the size of the dataset (batch size), y_i is the real output value, and $\tilde{y}_i$ is the predicted output. The predicted output is computed with the following equation:

$$Y = a_0 + a_1X + a_2X^2 + \ldots + a_dX^d$$

Equation 2. Predicted value

Where a_i are provided coefficients and X is an input value. The number of coefficients provided dictates the complexity of the model. If there are *d* coefficients, then the function in equation 2 will be

of degree *d*. In this report different notation is used for different error rates. *E_in* is the training error, *E_out* is testing error, *E_gen* is mean error between the training error and testing error, *E_bias* is the error of an average model over many runs. We train many models over different datasets, and then use the average between their coefficients to produce a new model, which is use to calculate the *E_bias* error.

PART C: Estimates polynomial coefficients by fitting to data

Firstly a set of random coefficients must be generated if none are provided. The amount of coefficients generated is determined by the degree provided. Once the coefficients are generated, or if the coefficients were provided, the batch size used for estimating the new coefficients is determined. If regular gradient decent is being used, the entire dataset provided will be used. If stochastic gradient decent is being used, a random data point will be chosen from the dataset. If mini-batched stochastic gradient decent is being used, a sample of a provided size will be used to calculate the new coefficients.

To fit the coefficients to the data, the partial derivative of the polynomial equation are first found with respect to each individual coefficient. Once this is found, the data points from the dataset sample are put through the partial derivatives and the results are summed. The sum is then divided by the sample size, multiplied by a learning rate, and added to the old respective coefficient, which will be the new coefficient. For every run in this report, regular gradient decent is used, and so, the following equation is followed:

$$\theta^{\text{new}} := \theta^{\text{old}} - \lambda \frac{d\mathcal{L}}{d\theta}(\theta^{\text{old}})$$

Equation 3. Gradient decent

Where theta is the coefficient that is being updated, lambda is a learning rate, and L is a loss function (MSE in our case).

The coefficients are fit to the data by following the above for a provided number of iterations. With every iteration, the model better represents the training data. Here is an example of a sequence of iterations to show how the model is fitting the data (current iteration in the title of the figure):

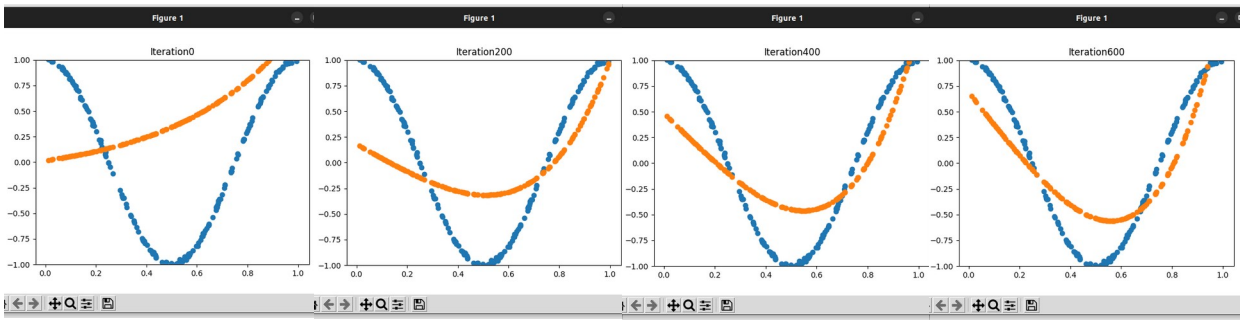

Figure 2. Model being fit to the data

As can be seen in the images above, the model seems to be fitting well to the training data with each 200 iteration jump. One way of knowing how the model is performing is by calculating its loss with the mean squared error. The error on the training dataset and a newly generated testing dataset is calculated. The testing dataset is generated using the same distributions as that of the training dataset. Shown bellow is how the error changes over the iterations in both training and testing.

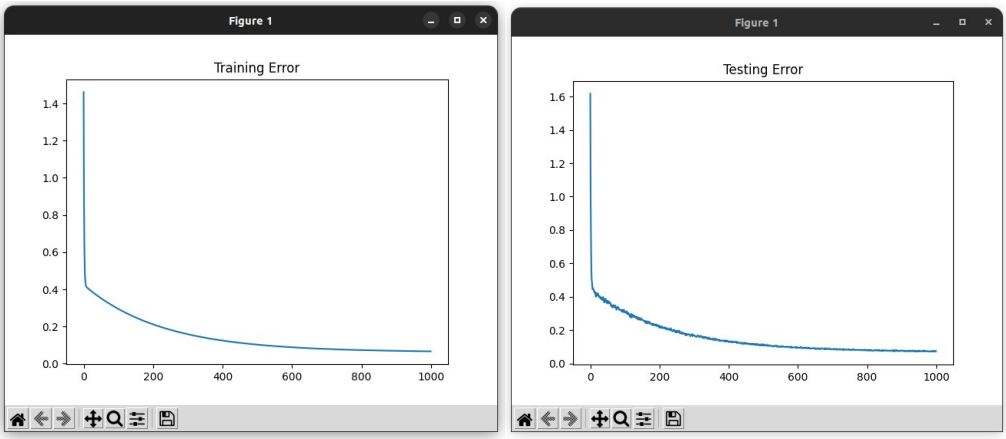

Figure 3. Training Error and Testing Error respectively

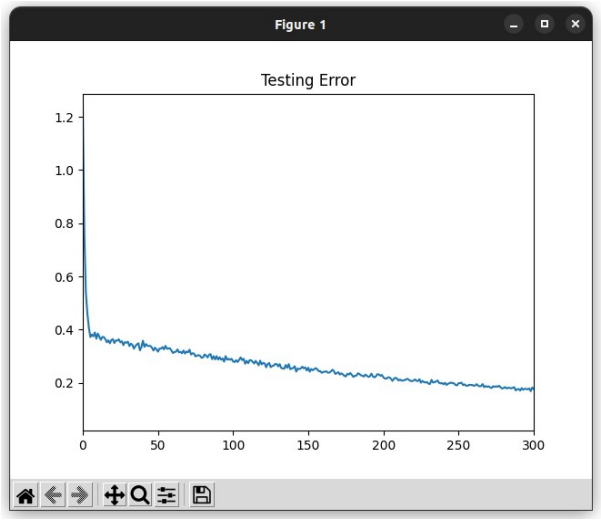

Figure 4. Zoom in of Testing Error

In figure 3 and 4, the y axis represents the error, and the x axis represents the number of iterations. As can be seen, the error for both the training and testing datasets is minimized as the model goes through the iterations. One important observation to note is the instability of the testing error, which is caused by the variation of the data between the testing and training sets. The model is fit according to the distribution of the training data, and though the testing data shares the same setting of variance, it in itself is not identical to the training data.

PART D: Experiment

So far, all the examples provided in this report have used the same parameters for fitting the model. These parameters are:
variance = 0.01^2,
polynomial degree = 5,
algorithm = GD,
training dataset size = 200,
learning rate = 0.1,
and iterations = 1000

For a good understanding of how a certain set of parameters affect the model, multiple trails must be run and evaluated. Each trial uses the same dataset size, polynomial degree, and noise variance. The amount of trials should be large, and in our case, we are going to keep the number of trials at 50. Each trial generates a new training dataset for the model to fit its coefficients, and a testing dataset. The training errors and testing errors are then averaged out, which gives a good representation of how the model performs on average. The coefficients will also be averaged and used on a separate testing dataset to determine the bias error. These metrics can be used to compare the effectiveness of different parameter combinations on the model and how it fits to data.

Using the previous parameters, and running them trough 50 trails, we get the following averages:
Average training error = 0.09013913320733793
Average testing error = 0.09355136396047045
Average coefficients testing error = 0.0942545281343233

The parameters chosen have yielded decent results, however they were chosen at random and so they might not be optimal. Also the point of this assignment has less to do with building a model, and more with understanding how different parameters will affect the learning of the model. For this reason, running more experiments with different parameter combinations is a must.

PART E: Results
The parameters that will be modified and evaluated are the size of the training dataset $N \in \{2, 5, 10, 20, 50, 100, 200\}$, the polynomial complexity/degree $d \in \{0, 1, 2, \ldots, 20\}$, and sigma $\sigma \in \{0.01, 0.1, 1\}$. Note, the iterations here will be lowered to 20 to reduce the computation time (unless otherwise specified).

First the impact of the variance on the error rates is examined.

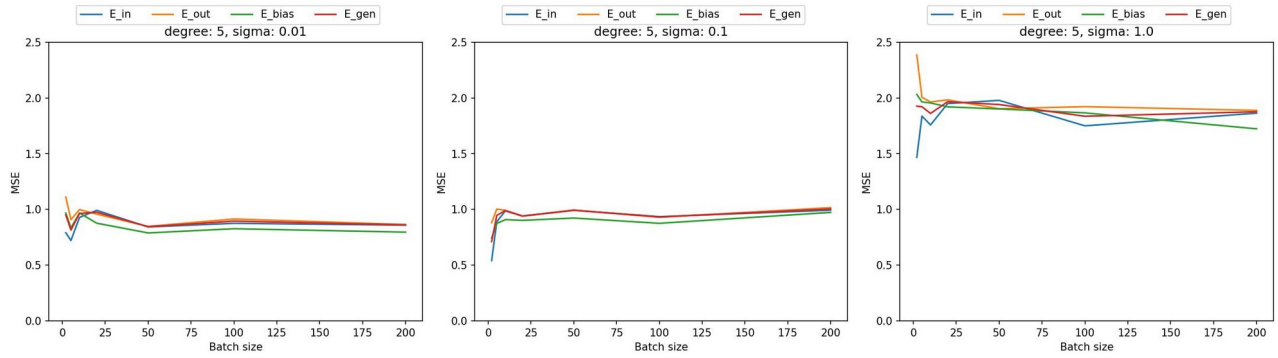

Figure 5. Error when sigma is modified (degree 5, over batch sizes)

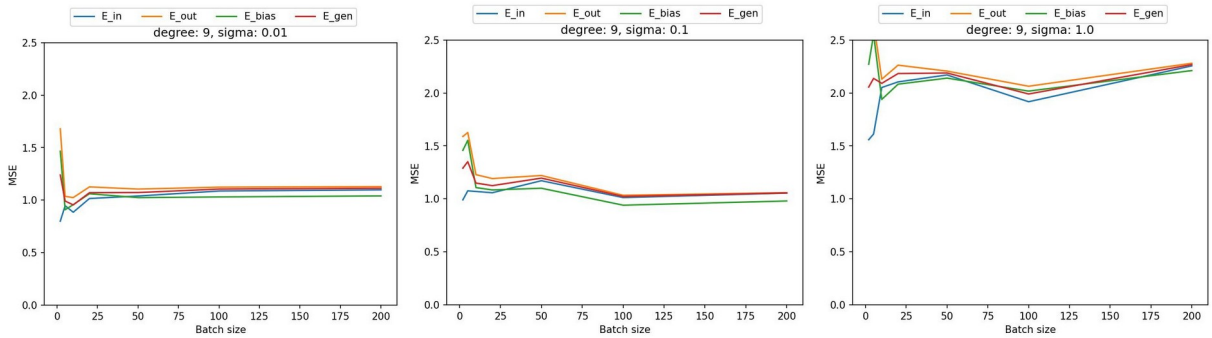

Figure 6. Error when sigma is modified (degree 9, over batch sizes)

In figure 5, it can be seen that when the variance is either 0.01 or 0.1, the error does not change very much, and is quite low. When the batch size is increased, the error between the two stays fairly consistent. When the variance is 1 however, the error increases by about a factor of two. This is seen all throughout, regardless of batch size and model complexity (as shown by comparing figure 5 and figure 6). For the reminder of the analysis, the focus will only be on the results using sigma = 0.01 and sigma = 0.1 due to their clear advantage over the higher value.

Next the complexity of the model and its effect on the error rates is analyzed.

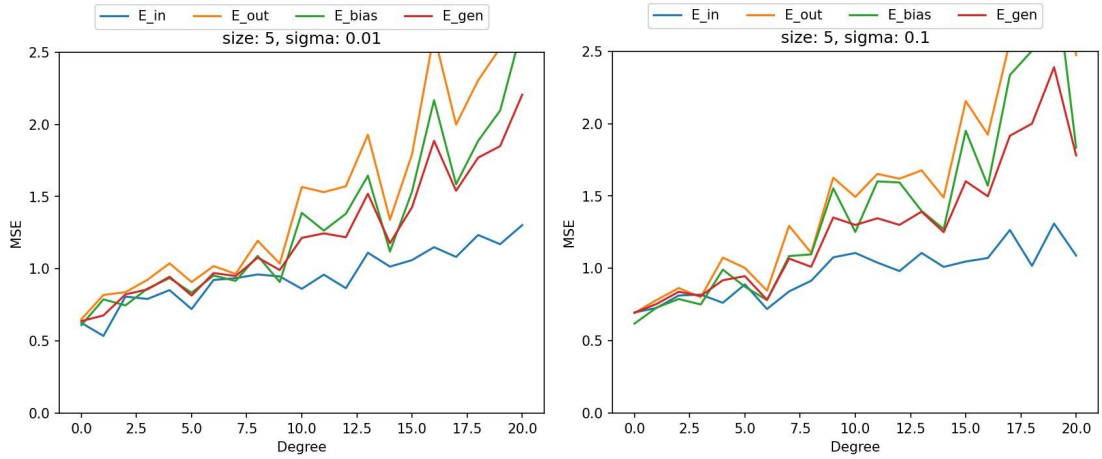

Figure 7. Error when size and variance are constants (size of 5)

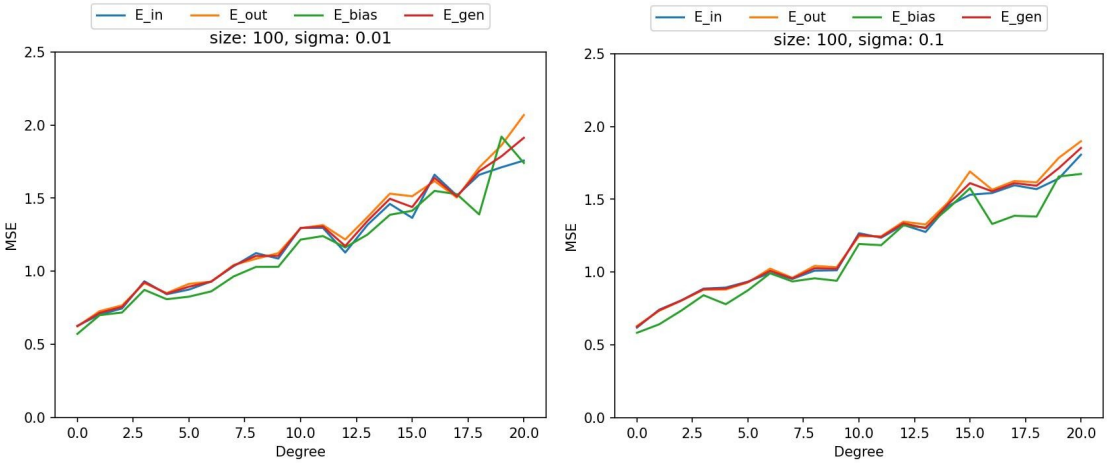

Figure 8. Error when size and variance are constants (size of 100)

First thing to note is that when the complexity of the model is increased, the batch size should also be increased. As shown in figure 7, when the batch size is low, the model tends to generalize poorly. This is shown by how the difference between training error and the testing error increases with the complexity of the model. This lack of complexity is causing the model to under fit the data. When looking at figure 8, it can been seen that the model is able to be more complex and better generalize when the batch size is much larger. That being said, it also shows that the model is performing poorly when the degree is augmented, as the error becomes much higher. This may be because the same amount of iterations to fit the model is being used. The model is therefore under fit to the data when a higher complexity is provided with not enough training. With more complexity, it is likely that the model would have to train for a longer amount of time to better fit the data, which we do not see in figure 7 and figure 8.

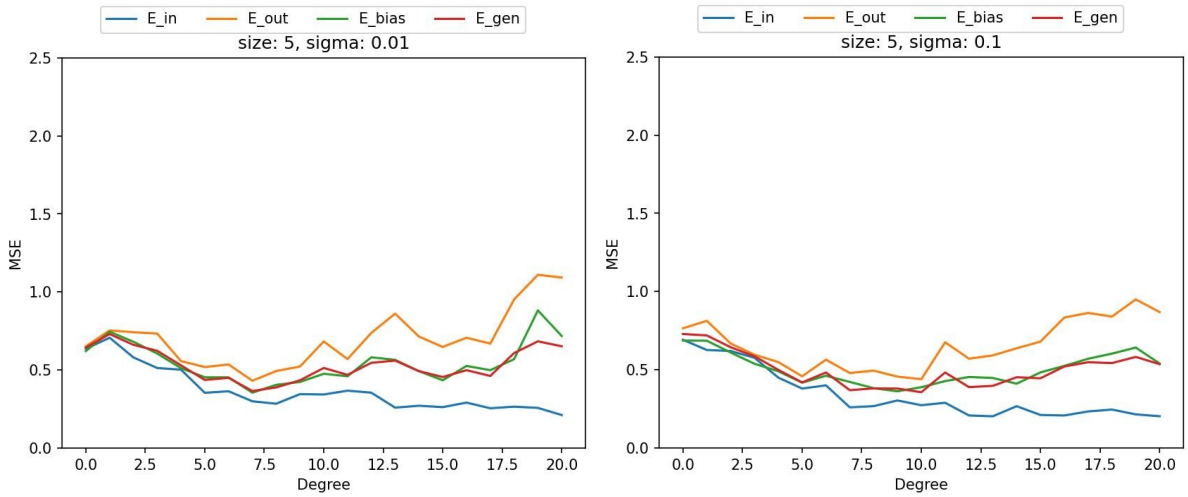

Figure 9. Augmenting iterations with complexity (size = 5, iterations = degree x 10)

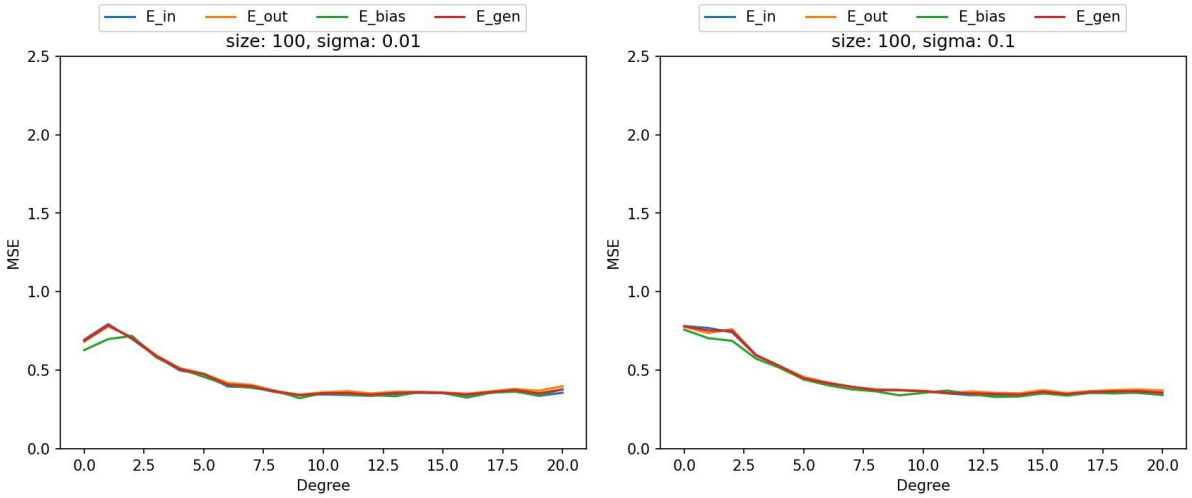

Figure 10. Augmenting iterations with complexity (size = 100, iterations = degree x 10)

When taking the relationship between complexity and iterations into account, and train the model longer with higher degrees, better results are obtained as shown in figure 10. This however might have more to do with the increase in training and less with the increase in complexity.

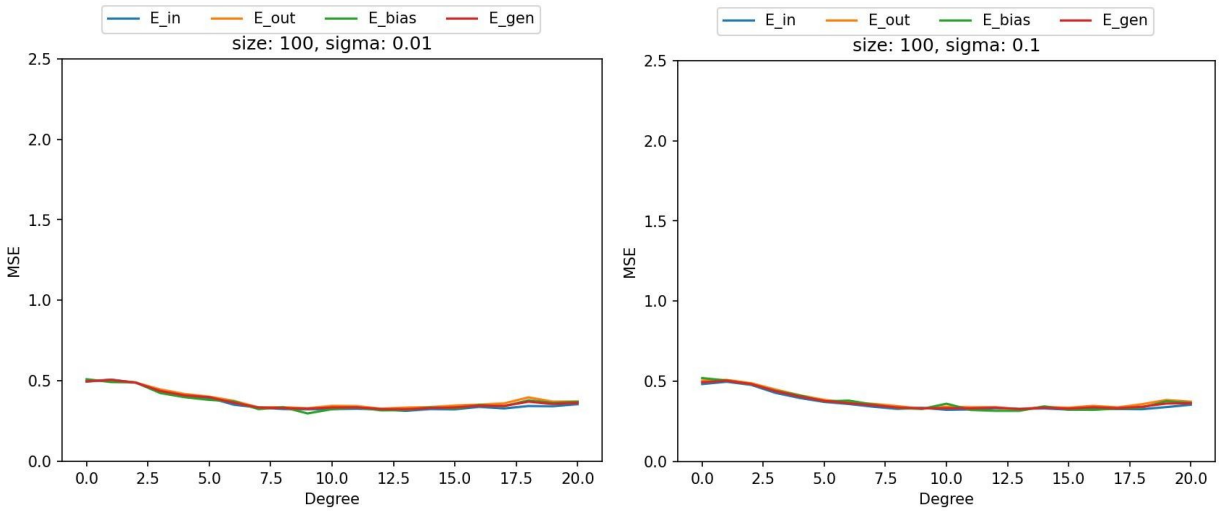

Figure 11. Error with consistent iterations (iterations = 200)

When using a consistent value for the iterations, the error looks similar to that of when the iterations are dependant on the complexity of the model, as seen in figure 11. Thus, when the model has a lower complexity it does not benefit as much from more training as when it has a higher complexity.

With a higher amount of iterations, one would expect for the model to over fit. At the end of the graphs in figure 11, you can see that the training errors do start to separate, which may be due to either over fitting or under fitting. With a higher variance, this becomes more apparent through the entire graph, as seen in figure 12.

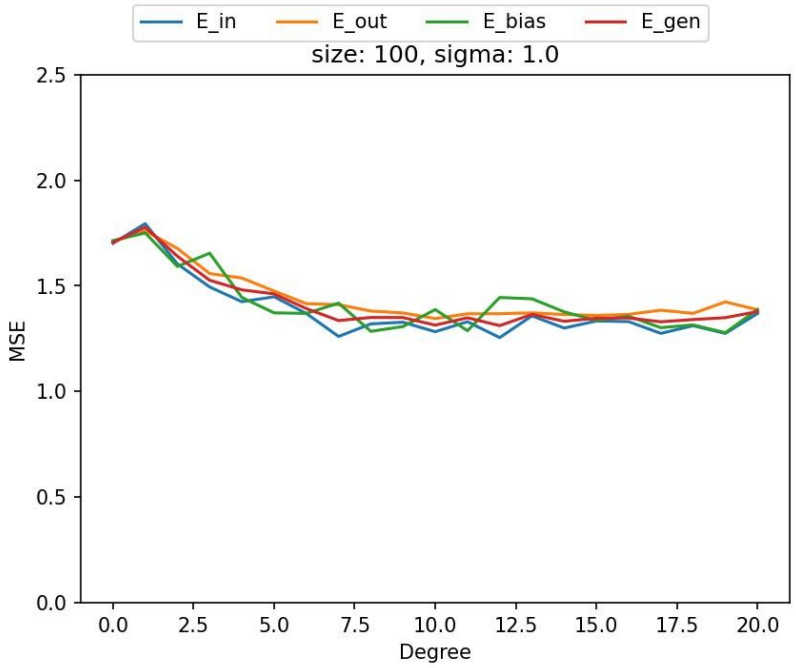

Figure 12. Error with consistent iterations (iterations = 200), variance = 1

With all this, the key notes are that the a more complex model will need to train for a longer amount of time, and it will also need more training data if it wants to generalize well. That said, the model

otherwise will risk either over fitting (low complexity, not enough training) or under fitting (large complexity, not enough data).

When the model is trained on a small batch size, for many iterations (1000 iterations), the bellow is obtained.

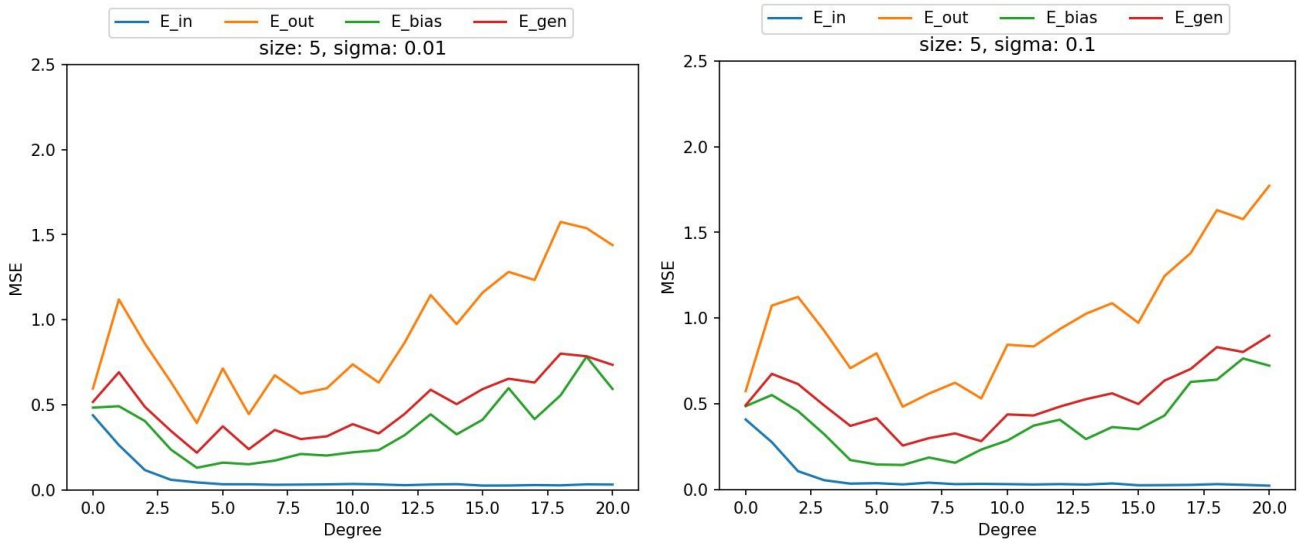

Figure 13. Error when size and variance are constants (size of 5) – 1000 training iterations

In this figure, it is clear that the model is being over fit. The models results on the training data improve significantly as the complexity augments, however it performs continuously worse on the the testing data. In the figure bellow is shown an example of a model that was trained using these parameters, after 900 total iterations. It is clear to see that the model is not fitting properly to the data distribution that were set in PART A.

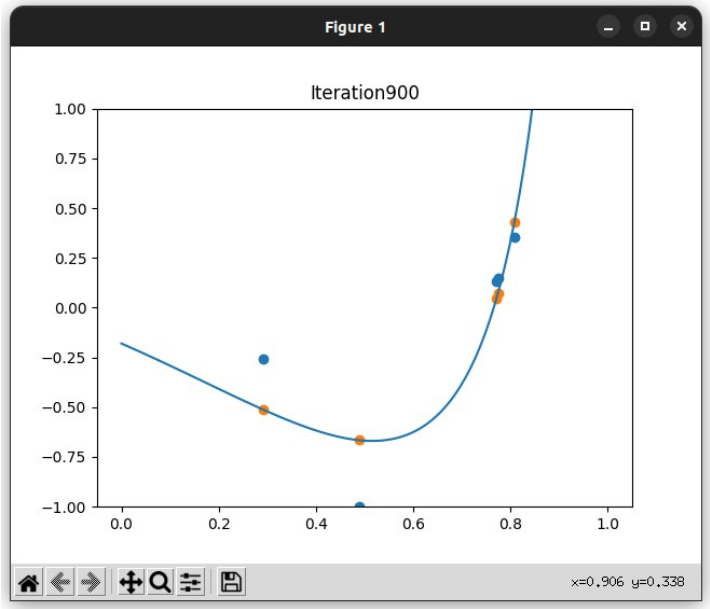

Figure 14. Poorly trained model (blue line)

Next the effect of the batch size on the error rates is briefly analyzed.

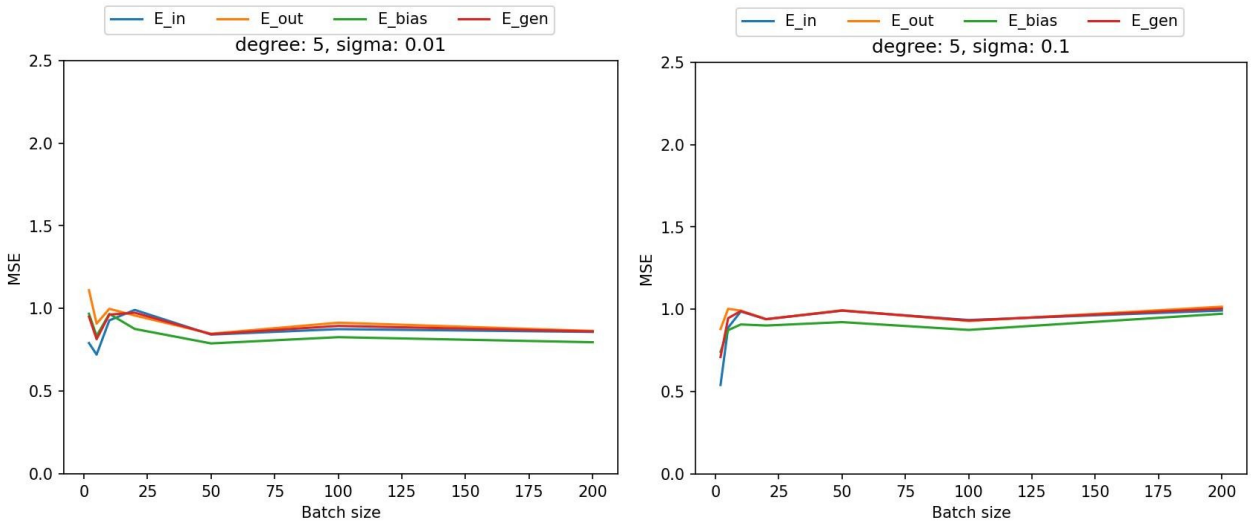

Figure 15. Error when complexity and variance are constants (degree of 5)

When the complexity of the model is low, there does not seem to be any over fitting, and the errors stay fairly consistent with respect to each other, keeping a close distance. Changing the variance (between 0.01 and 0.1) has little effect on the error rates. If anything, a lower variance keeps the model from fluctuating as much as when a higher variance is used.

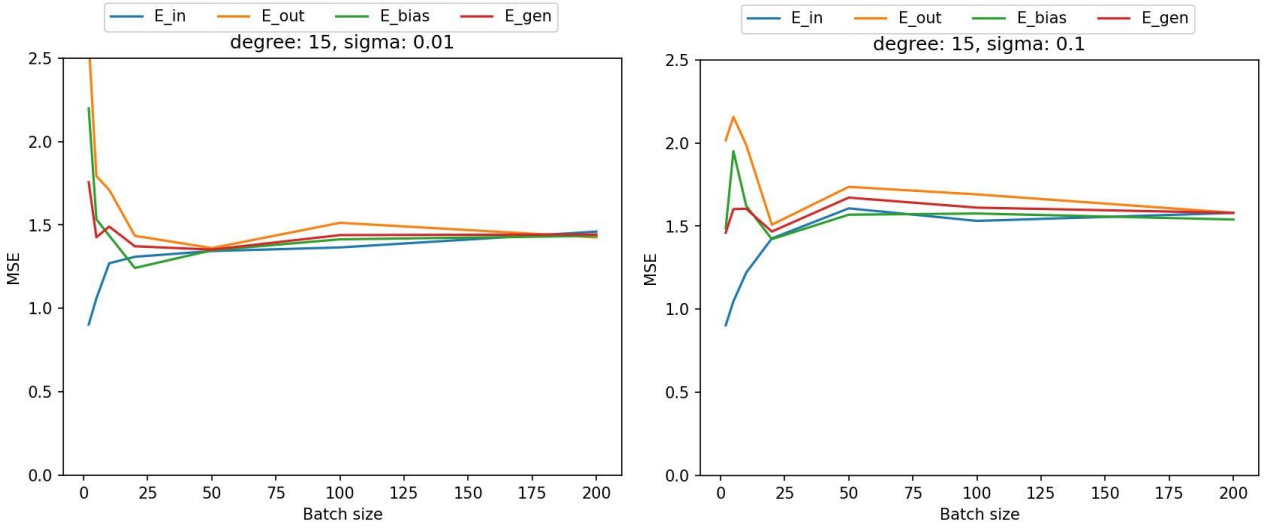

Figure 16. Error when complexity and variance are constants (degree of 15)

When the complexity of the model is high, a larger batch size is needed to better fit the data. We can therefore derive the same conclusions that we did when analyzing the error with respect to the complexity of the model.

PART F: Regularization

Using regularization is a good way to keep the complexity of a model lower by making certain weights very small. This in turn help to not over fit the model.

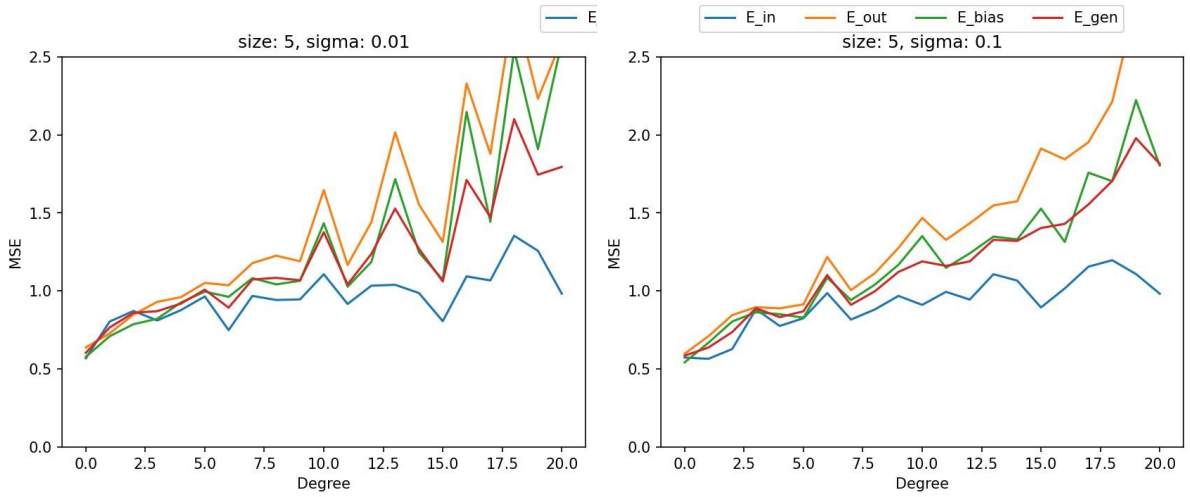

Figure 17. Error when size and variance are constants (size of 5) - Regularized (rate of 0.01)

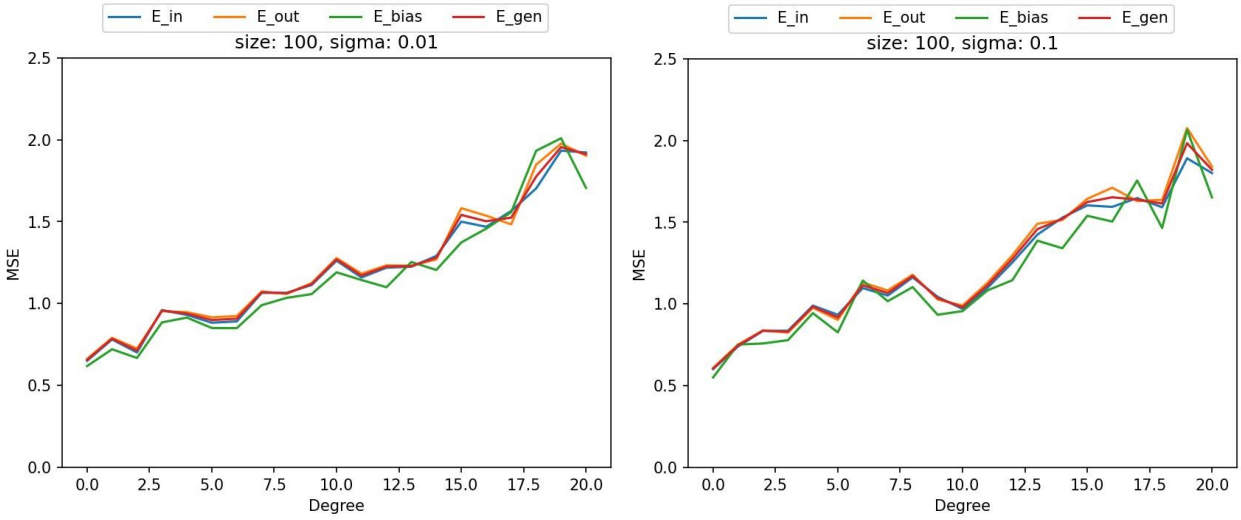

Figure 18. Error when size and variance are constants (size of 100) - Regularized (rate of 0.01)

When the batch size is very large, the regularization does a really good job at closing the gap between the training and testing error. The regularization rate used was consistent for both figure 17 and figure 18, at 0.01. In figure 17, the rate does not seem to be helping very much, and this may be due to it being too small.

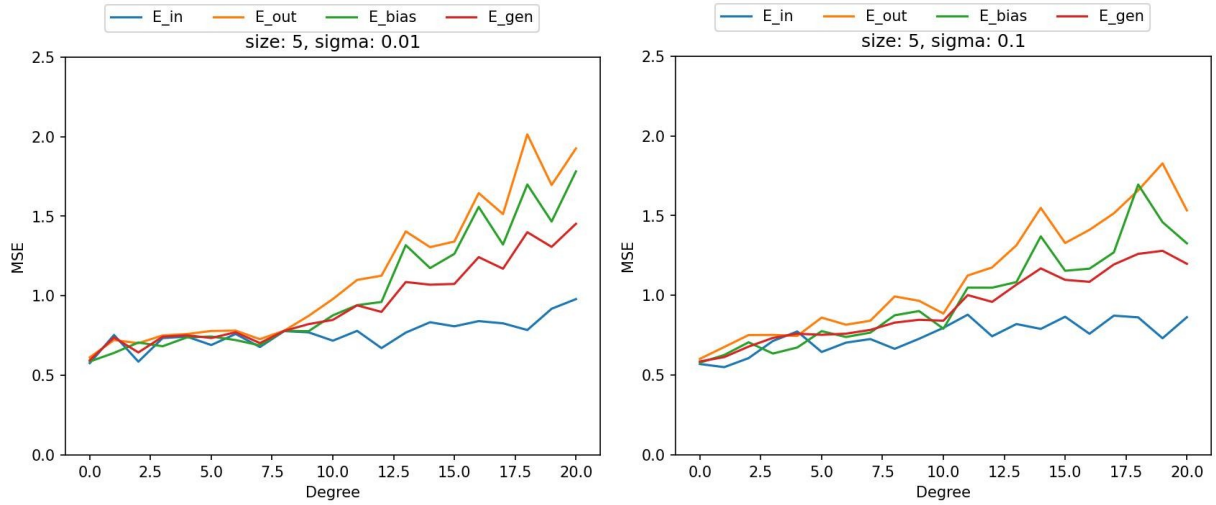

Figure 19. Error when size and variance are constants (size of 5) - Regularized (rate of 0.99)

When the rate is augmented to 0.99, it has a much stronger effect on the model when a smaller batch size is used, as seen in figure 19 vs figure 17.

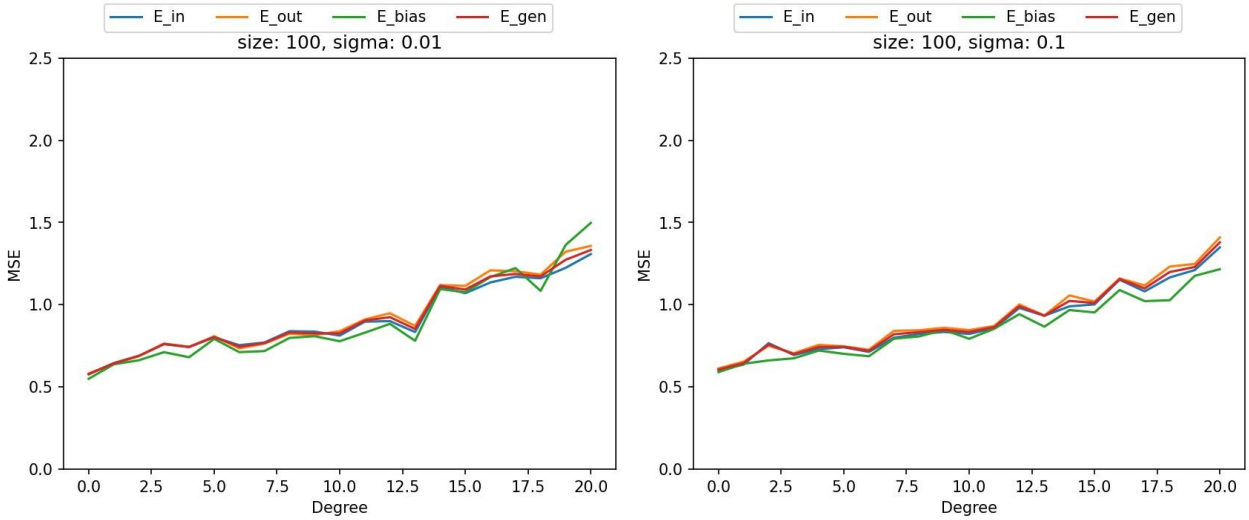

Figure 20. Error when size and variance are constants (size of 100) - Regularized (rate of 0.99)

However, when the same rate is used with a larger batch size, it can lead to the opposite effect as seen (slightly) in figure 20. The error is augmenting in each case above because of the lack of training with the higher degrees (As seen in PART E). If an augmenting iteration is used, the following results are obtained.

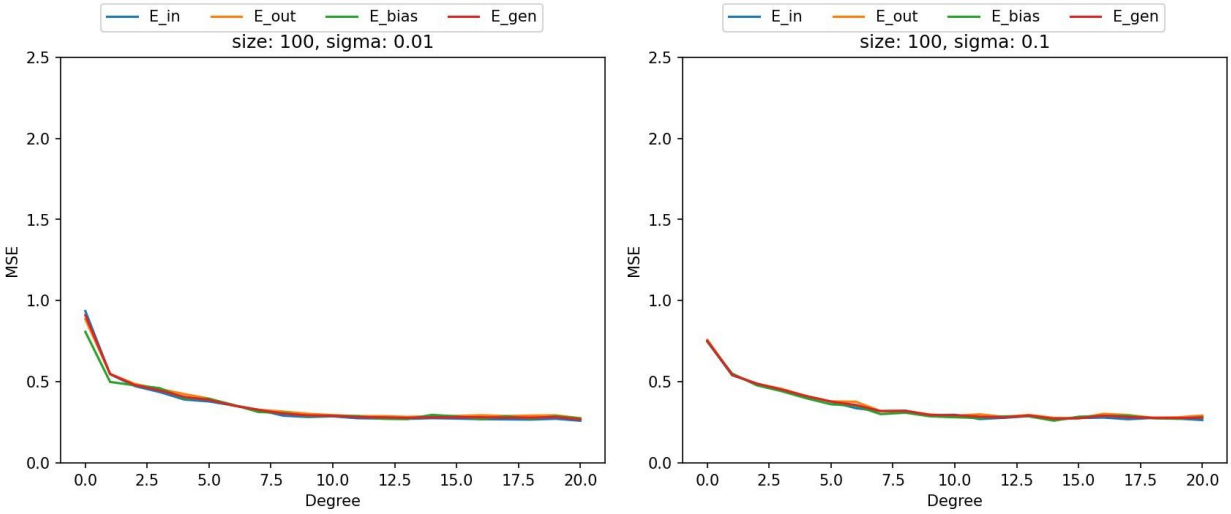

Figure 21. Error when size and variance are constants (size of 100) - Regularized (rate of 0.01) – Augmenting iterations with complexity (iterations = degree x 5)

Which seem to be the best results seen so far in terms of generalizing and error rates. We do see that after a certain complexity (when degree = 10), there is no loss in error. However, thanks to the regularization, the model does not seem to over fit either. What should be taken from this analysis is that a more complex model will be able to fit training data better, but more data and training is required. It is also import to use regularization so the model does not over fit the training data.

(More figures were generated, only a select few were used in this report)