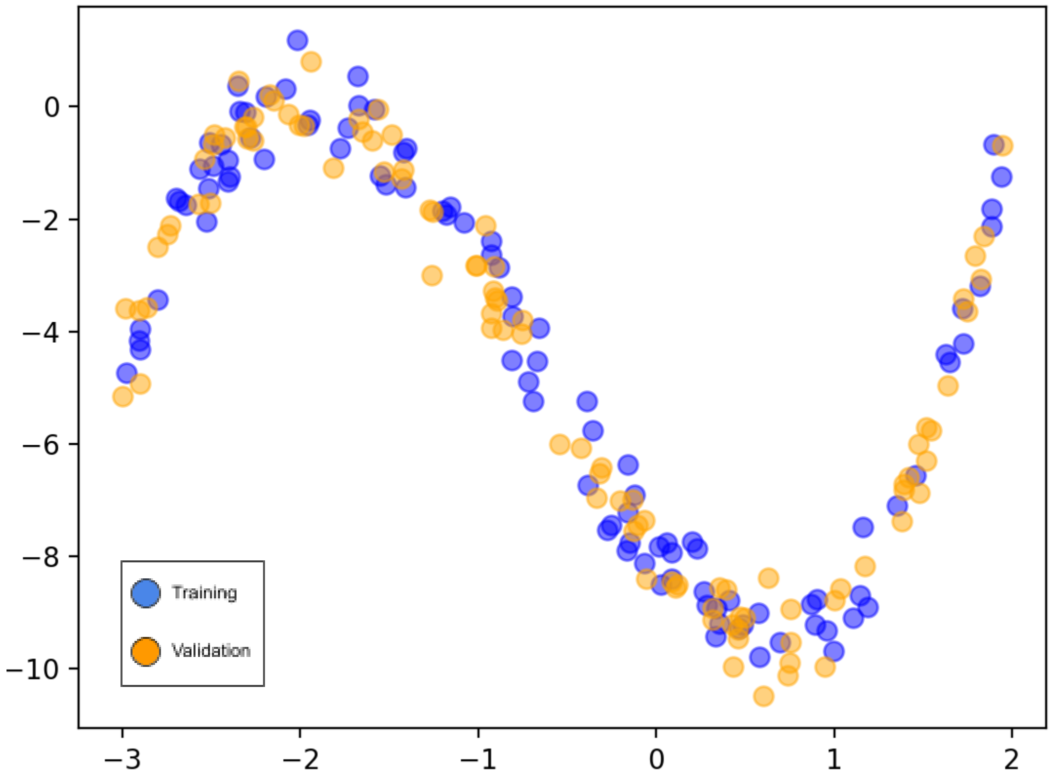
Alessandro De Grandi

Deep Learning Lab Assignment 1: Part1:

1. Just needed to implement $x[i]^j$ where j is the index of the polynomial term (j=1,..,degree)
2. Called the function create_dataset with the parameter values given.
3. Plot x,y:



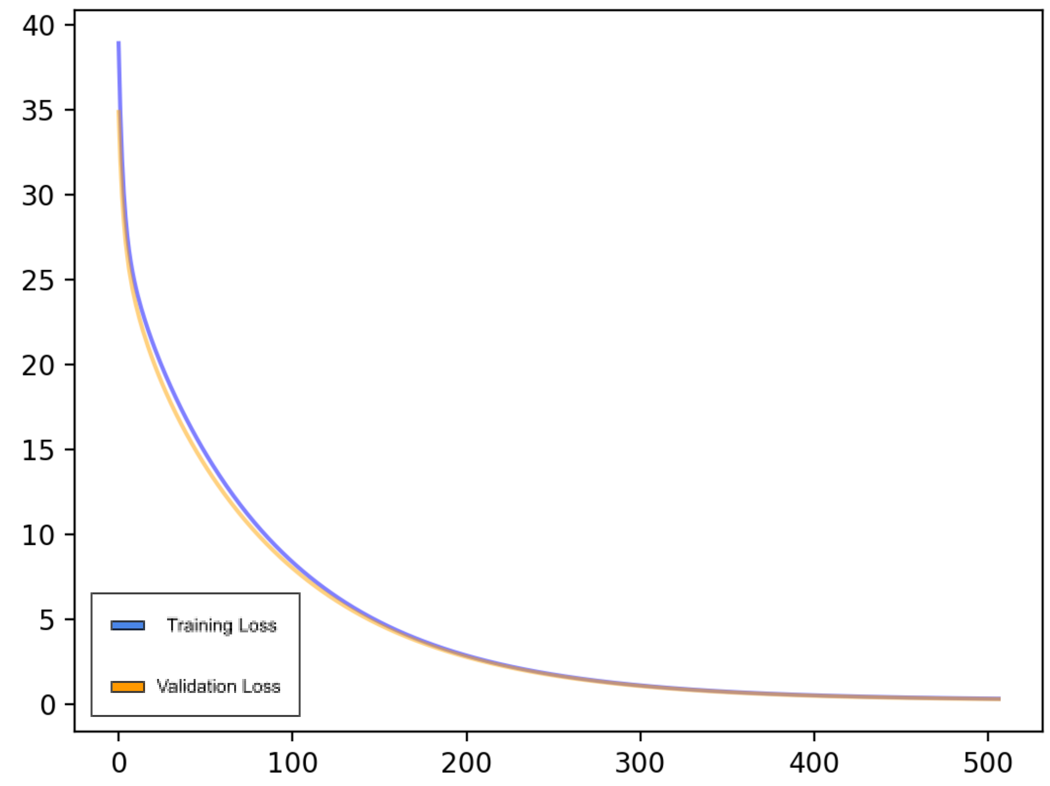
4. From https://pytorch.org/docs/stable/generated/torch.nn.Linear.html
   “bias – If set to `False`, the layer will not learn an additive bias. Default: `True`”

   output of a neuron = sum (weights * inputs) + bias

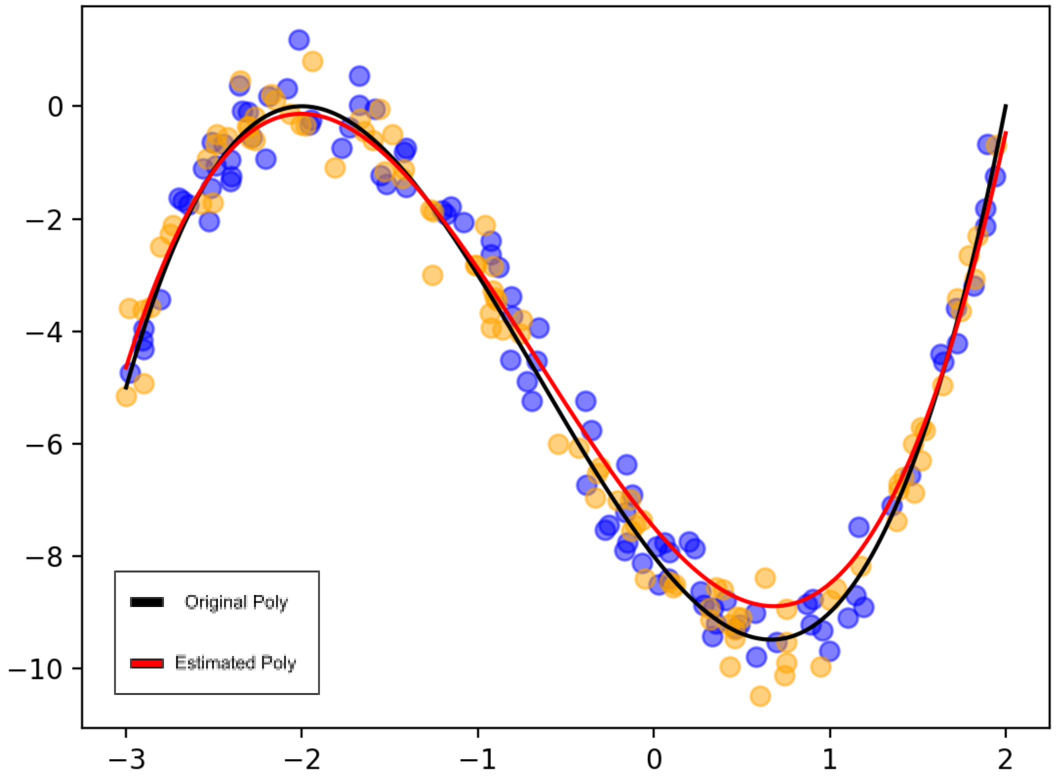
   The bias is like the additive value (intercept) in a linear equation. In this case we already have a bias because of the polynomial term of degree 0, so the flag should be explicitly set to False.

5. Implemented using model = nn.Linear(4,1,False). And following the guidelines and code provided in Lecture slides. Training the model to learn the polynomial coefficients. So after a training procedure, the model weights are, eg: w=[-7.7428, -3.9595, 1.9206, 0.9765] which is a pretty good estimate of w* = [−8, −4, 2, 1].

6. Starting from learning rate = 0.1, and after a number of attempts (that can be seen as comments in the code) of increasing the value (loss diverges) and decreasing it (it gets better until it is too small) , I settled for learning rate = 0.01.
   For the number of iterations I implemented a rudimental form of early stopping, so when the difference between the last validation loss and the current one is less than a certain value delta, that I set for example at delta=0.001, the training stops. Usually around 500 with a max of 1000 iterations, with that delta and learning rate,.
7. Plot Loss over Iterations:

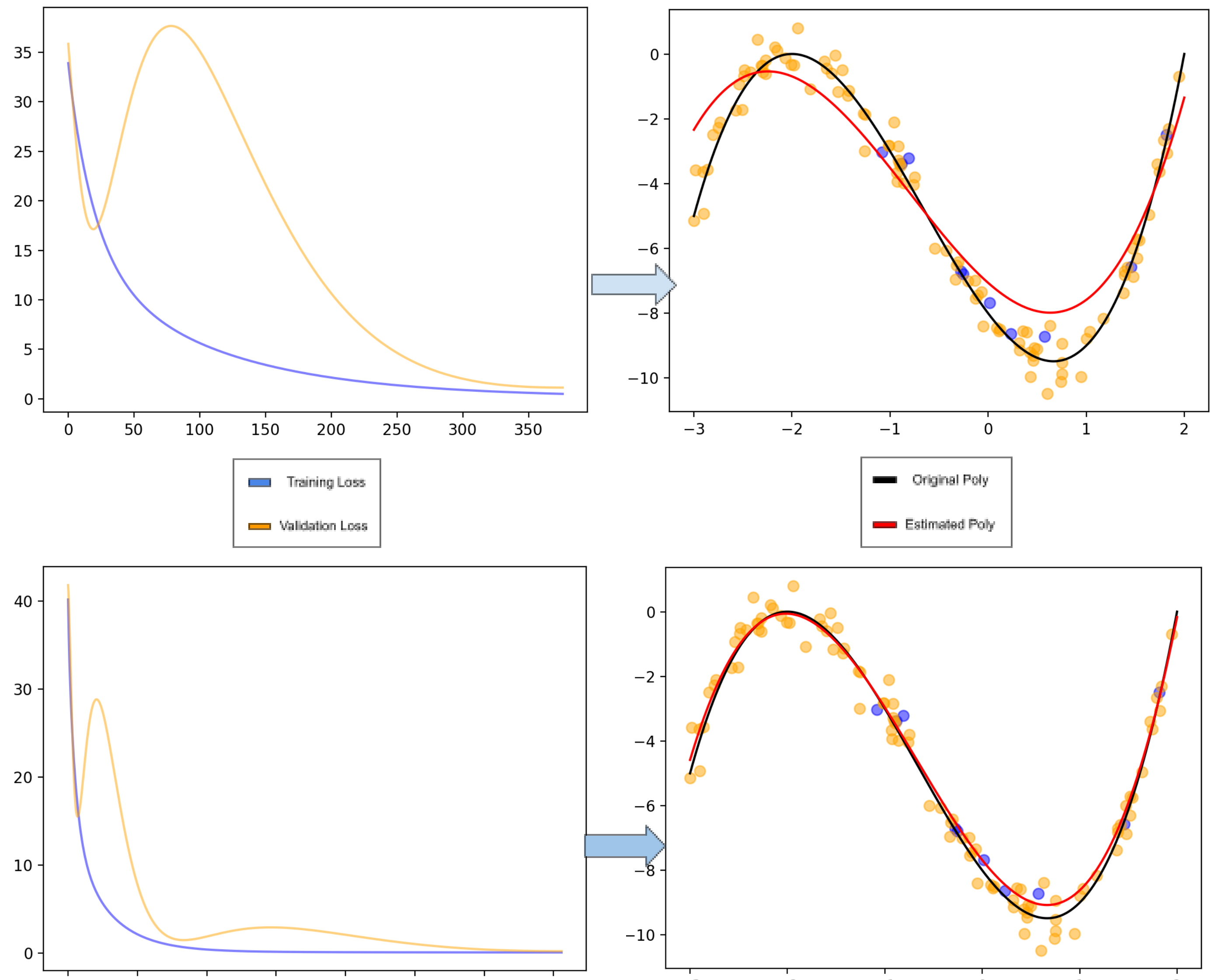


8.



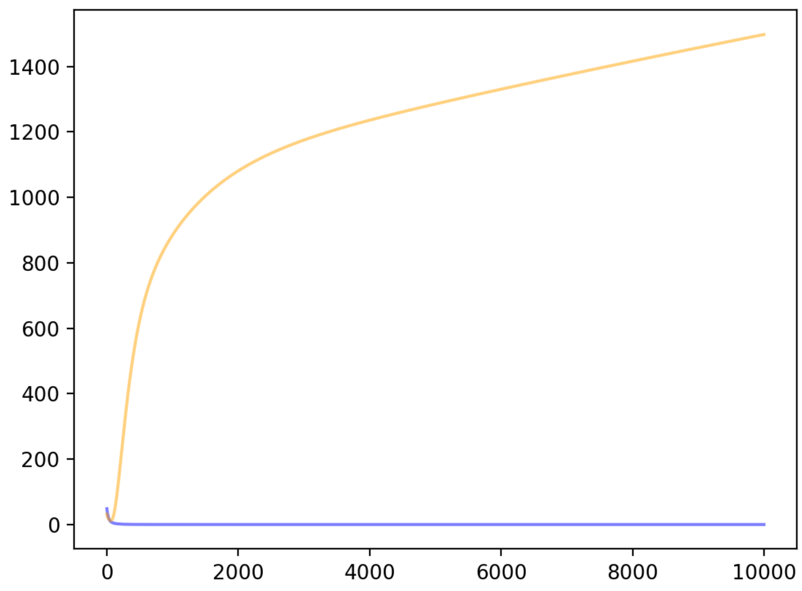
9. With size = 50: There is not much difference, the model learns fine.

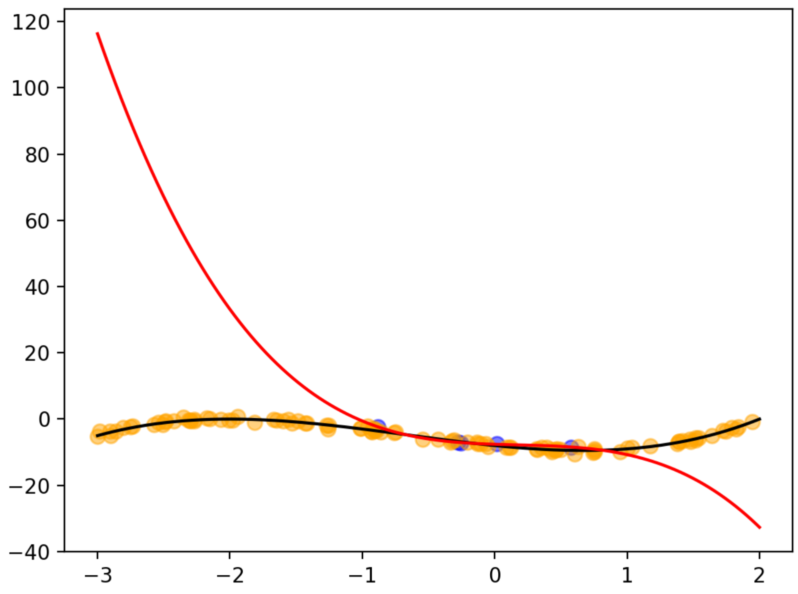
With size = 10:



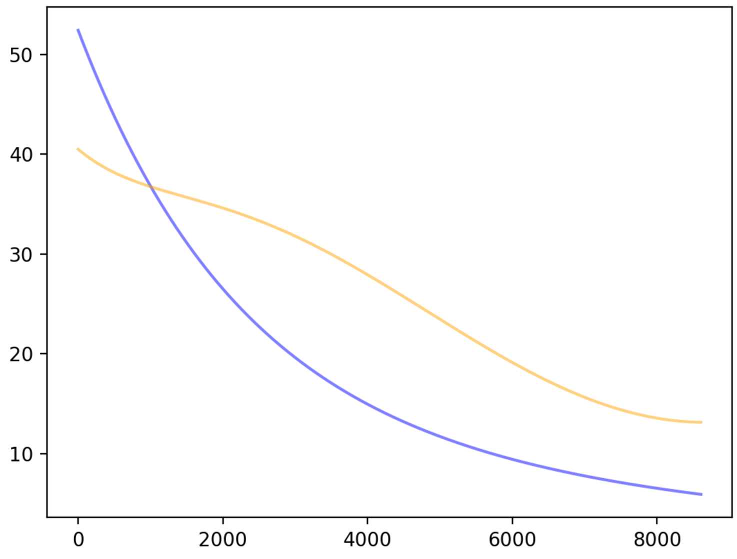
It looks like the model is struggling to learn at first, as we can see from the validation curves in the plots above, it is not performing well on the validation set, but since I am running more iterations, it will eventually reach the same predictions as with 100 samples.

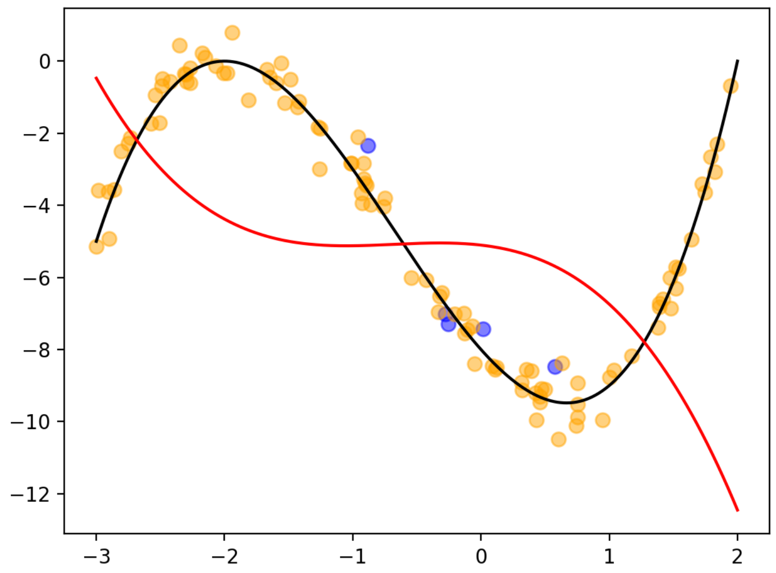
With size = 5:
In this case the amount of data is not enough to represent the underlying structure of the polynomial, so the model cannot learn it; In the following two graphs we can see the big difference in losses between training and validation. An example of overfitting.





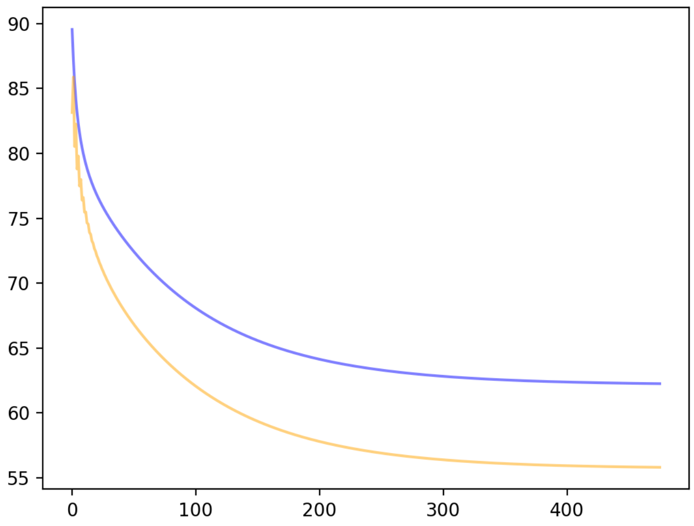
Adjusting the learning rate and early stopping help against overfitting but is not enough. The model needs a better dataset.

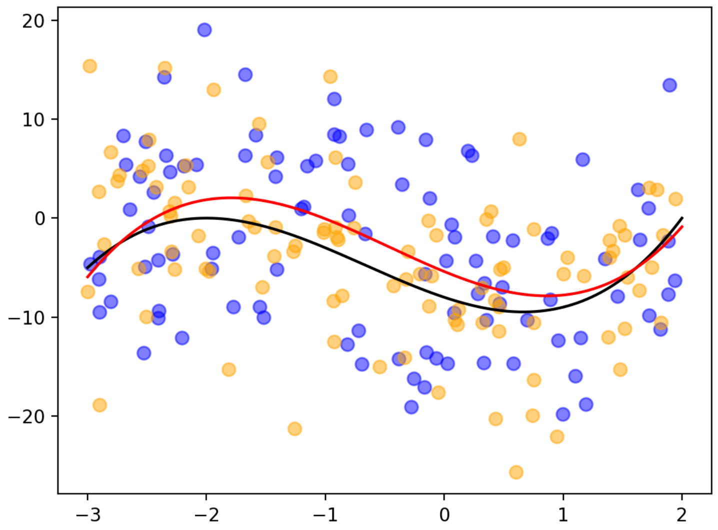




10. σ=2, σ=4,σ=8: increasing the standard deviation adds more “noise” to the dataset, so it is possible that with overfitting, the model would “fit” the noise better than the structure we are interested in. This is not the case, but what is interesting is that the evaluation losses end up smaller than the training losses, the model does better with new data than with data it has seen!

Eg: sigma = 8





This is probably explainable by knowing how the loss is calculated, MSE or mean squared error uses the metric of distance between prediction and data, so if the points generated with seed 1(validation) happen to be closer to each other it is easier to get smaller distance values. This can be noticed by looking at the graph on the right; the blue dots(training) happen to be more spread out on the y axis, and the orange dots(validation) are closer to each other.

So it's a random effect and is due to the structure of the dataset.

Part 2:

1.A local minimum is like a low spot on a mountain, looking at your surroundings it might look like you have descended, because everything around you is higher. But roaming over the higher surroundings can reveal more pathways downhill that can lead you to the global minimum. That is the absolute lower spot at the base of the mountain.

2.Overfitting is when your model has learned the training data very well, too well in fact(like memorization), but it has not learned the underlying structures in the data, so it cannot adapt well to new information. Underfitting is when the model has not learned enough.

3.Early stopping;splitting dataset into training,validation,test;finding more data; are examples of techniques used to mitigate overfitting. To mitigate underfitting you could also try tweaking the hyperparameters or/and using a more complex model.