Deep learning lab - Assignment 1

Christian Altrichter

October 23, 2022

1 Polynomial Regression [96/100 points]

1.1 Question 1.1

The corresponding code snippet is provided in the source file. For the visualization the given helper function "plot polynomial" was used. The corresponding coefficients are: [0,-5,2,1,0.05]
The outcome of the the completion of the code in a range from -3 to 3 on the x axis is as follows:

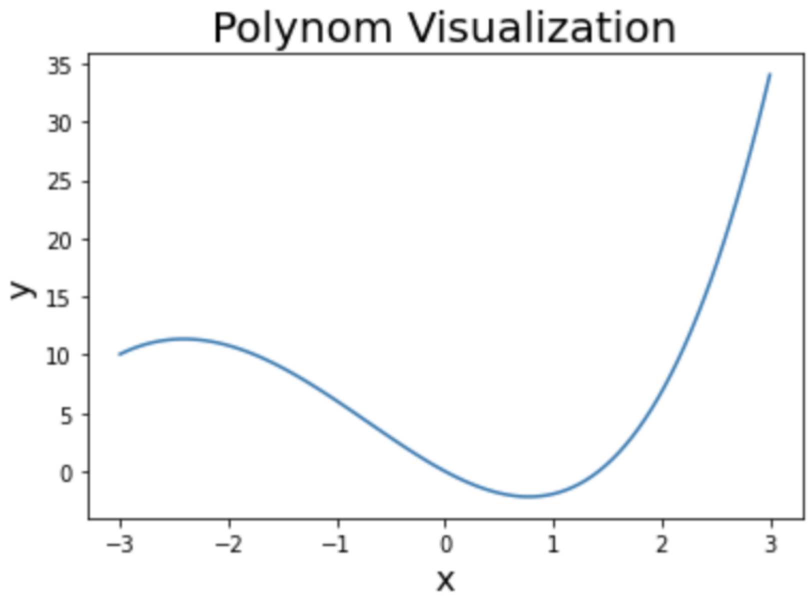

Figure 1: Graph for Question 1.1, plotting the given polynom

1.2 Question 1.2

The code has been completed and no special functions were required except for the function "create dataset" itself. Here, I work with the given information from the text that states that:

$$x_i = [z_i, z_i^2, z_i^3, z_i^4]^T$$

It is to be noted that the above tensor does need to include the 0-coefficient.

Thus, two loops have been used, the outer to represent the "i" index for each "x", which is equivalent to the sample size. The second loop represents the size of the coefficients which is used for the inner loop (index = j) to put each value to the power of "j". As mentioned above, as the range for the inner loop goes from 0 to 5 (for the shape of the weight), we take any

$$(z_i)^j$$

where j = 0 to include the zero coefficient.

1.3 Question 1.3

The only special function used was the one completed in Question 1.2, which is called "create dataset". Here, the two data sets only differ in the use of a different seed, where the training set used a seed of 0 and the validation set used a seed of 1. All other parameters (e.g. the weight "w", the sigma, the sample size and the interval) stayed the same.

This is needed so we can train our model on one set and verify our training on another set. Thus, assuring that our model works on any given randomized set and not only on our training set.

1.4 Question 1.4

In the following the two graphs are depicted based on the two data sets created from Question 1.3. Here, we plot the y values as comuted. To plot the x values on the x axis, one can use the index i = 1 (when starting at a 0 index array). Thus, I used the 1st coefficient, which is

$$z^1$$

to display the x values corresponding to the y values of the polynomial.

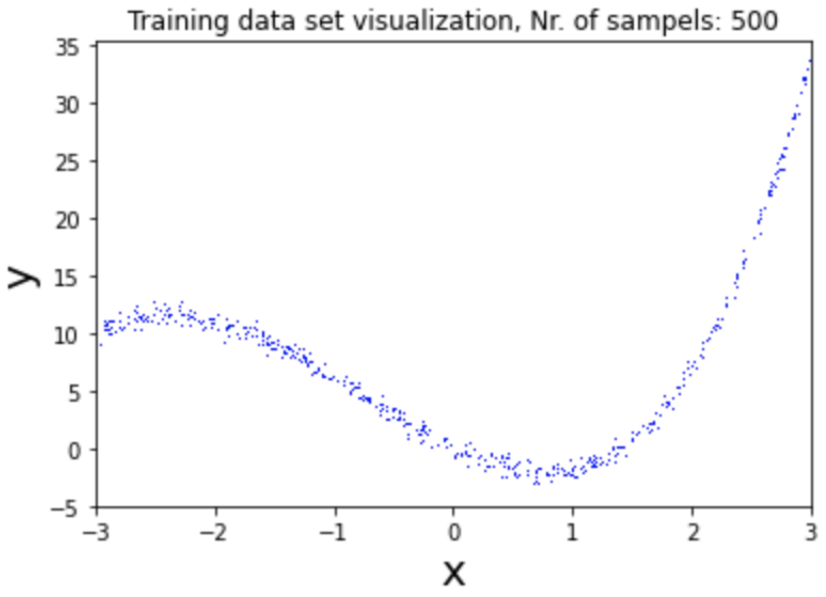

Figure 2: Graph for Question 1.4, scatter plot of the test data

1.5 Question 1.5

For nn.Linear, if the bias is set to True, a randomly generated bias is added to the output where the bias is in dependency to the input. Here, bias is calculated by

$$k = \frac{1}{'in_features'}$$

and then randomized between

$$[-\sqrt{k} \rightarrow \sqrt{k}]$$

. However, if it is set to false, then the bias is set to 0. The function

$$y = x * W^T + b$$

where x is our input, W is the weight and b the bias.

Due to the fact that the weight of the model for the 0-coefficient is non-existent (thus, equal to zero), the model does not need to learn an additional bias. Therefore, it can be set to false in our application.

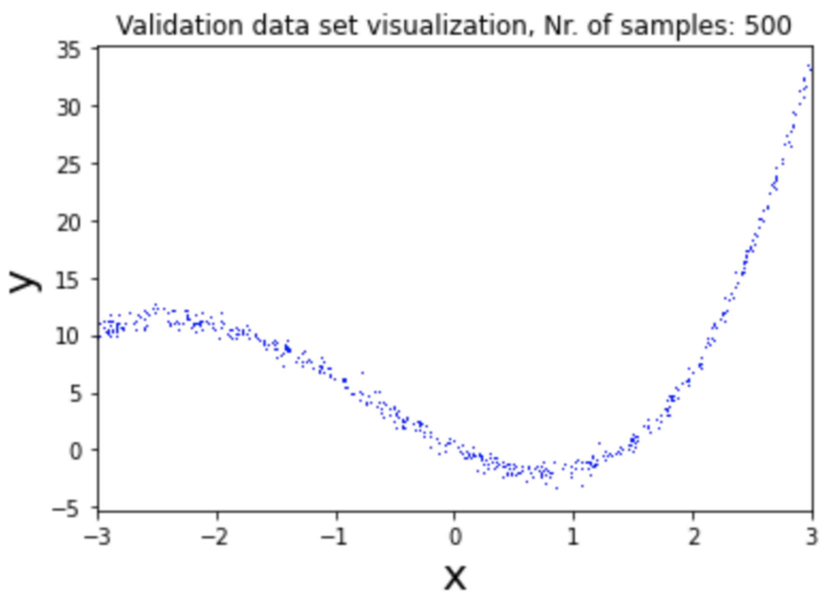

Figure 3: Graph for Question 1.4, scatter plot of the validation data

1.6 Question 1.6

The adapted code for Question 1.6 can be found in the source file.
Amongst others, the code has been adapted by:

- Adapting the reshaping of the tensors.
- Including the breaking validation and training loss if under 0.6.

1.7 Question 1.7

NOTE: I have created a function around the data set creation and the model training. This function called "dl gradient descent" takes as parameters the following:

- **learning rate input**
- **iterations input**
- **train size**
- **val size**

This enables the program to be easily tested with various parameters. Furthermore, the function returns the estimated weight of the model which subsequently can then be used for further analysis.

I have started with adjusting the hyper parameters as in the table below:

Learning rate	Iterations	Estimated 'w'
0.004	5000	'naan'
0.0001	5000	[0.8266806, -2.0675335, 1.1985728, 0.5462172, 0.14852887]
0.0005	2000	[0.28982046, -4.5170016, 1.8167112, 0.92569387, 0.06966818]
0.0008	2000	[0.43103147, -3.980467, 1.7431525, 0.84302974, 0.07765391]
0.001	2000	[0.3188451, -4.201298, 1.8114821, 0.87715226, 0.07015042]

A well trained model in our scenario leading to a low training and validation loss has been achieved with the following hyper parameters:

Learning rate	Iterations	Initial ’w’
0.0012	2000	[0.2864882, 0.2919198, 0.37758976, 0.3272866, -0.21655442]

Learning rate	Iterations	Estimated ’w’
0.0012	2000	[0.31423962, -4.472779, 1.8031044, 0.91886634, 0.07111725]

The result of an optimal training result, where validation and training loss are below 0.6 can be visualized as follows:

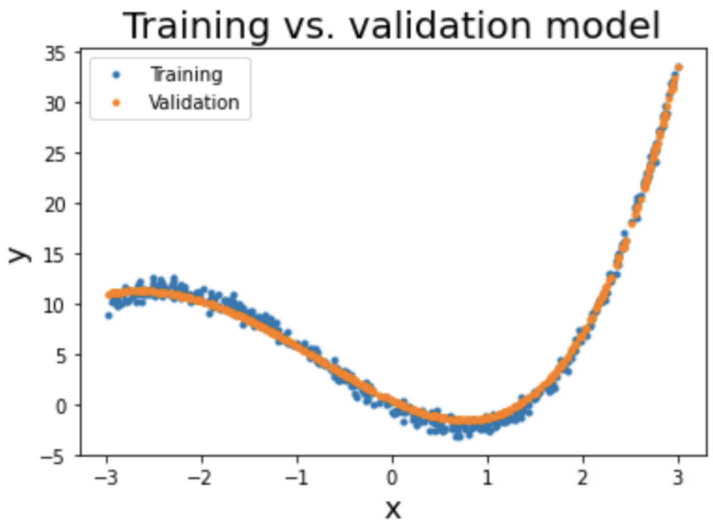

Figure 4: Graph for Question 1.7, validation and training model

1.8 Question 1.8

The validation and training losses from the above trained model are depicted as follows: Through gradient descent they converge both to the loss required of:

$$<= 0.6$$

.

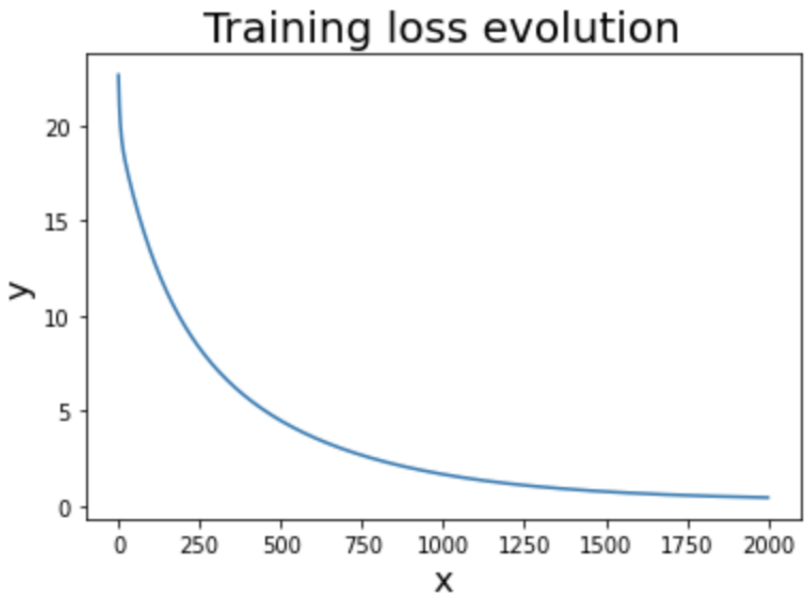

Figure 5: Graph for Question 1.8, training losses

And respectively:

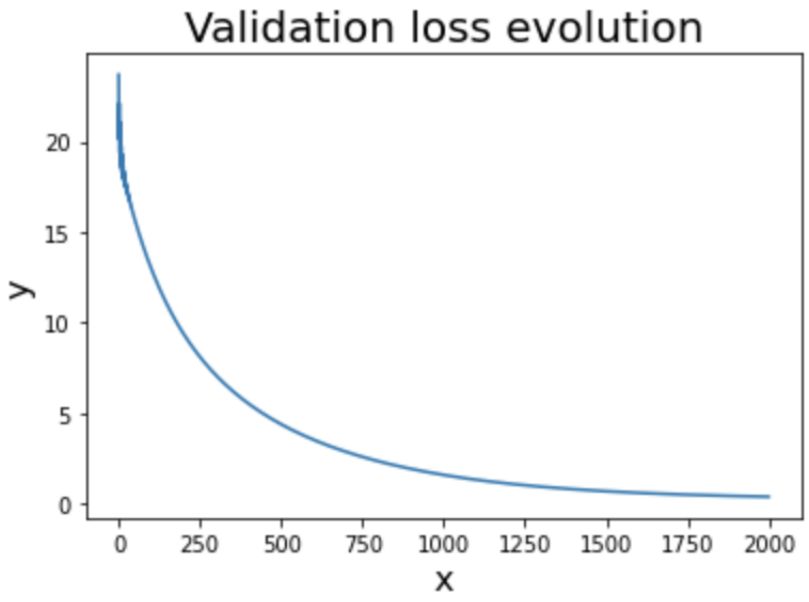

Figure 6: Graph for Question 1.8, validation losses

1.9 Question 1.9

The evolution of w based on the results obtained in question 1.7 can be visualized as follows:

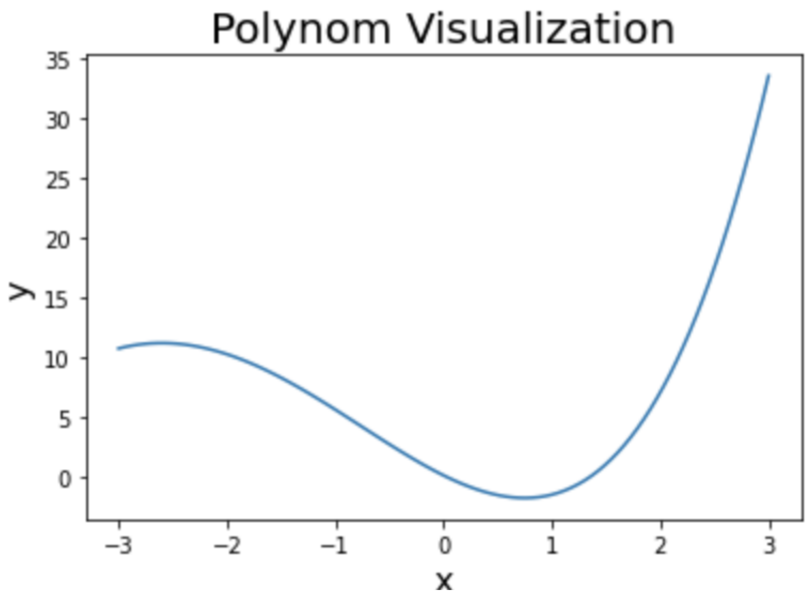

Figure 7: Graph for Question 1.9, polynomial defined by the estimate of w (obtained by question 1.7)

1.10 Question 1.10

Here, the training data set was reduced to 10. Thus, the trained model is based on the 10 data points. Now, when comparing the learned model to the validation data set, it first needs to recognize the underlying data structure from the validation model which takes longer due to the difference in size between training and validation. The larger the iteration size is, the more accurate the model will become with respect to then also the validation data set. Due to the fact that the training data set is quite small, the training loss is reduced comparatively quickly as well. This could however give the risk of initial underfitting the model just based on the training data set.

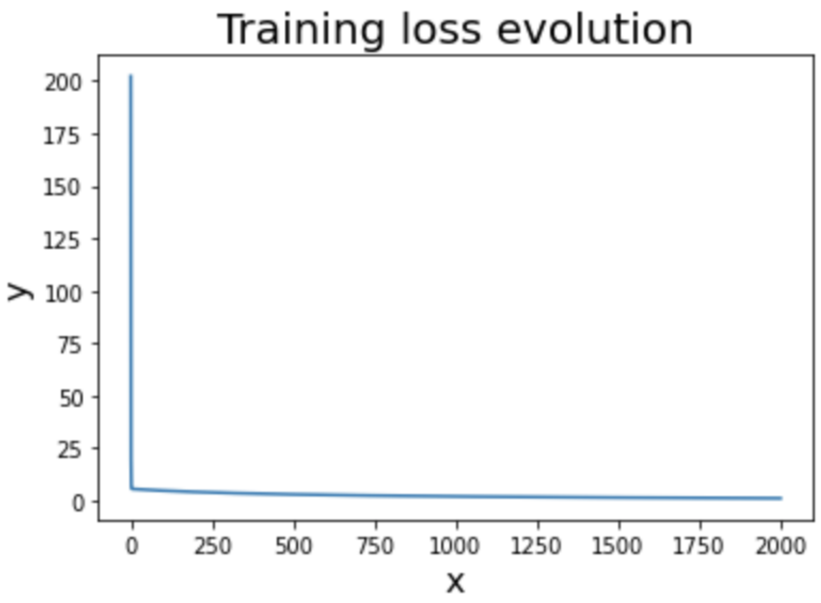

Figure 8: Graph for Question 1.10, training losses with reduced observations (10)

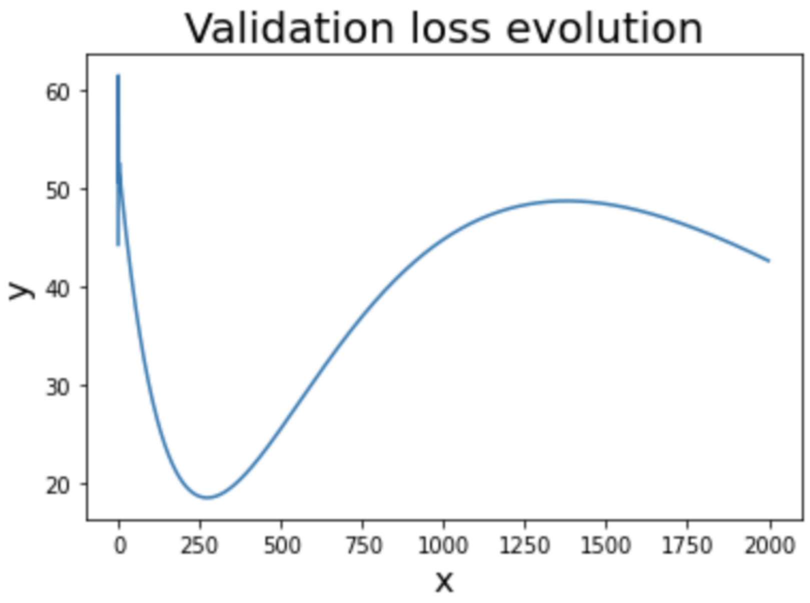

Figure 9: Graph for Question 1.10, validation losses with reduced observations (10)

1.11 Question 1.11 - Bonus

The following is the visualization of the polynomial defined by the estimate of w obtained by question 1.7.

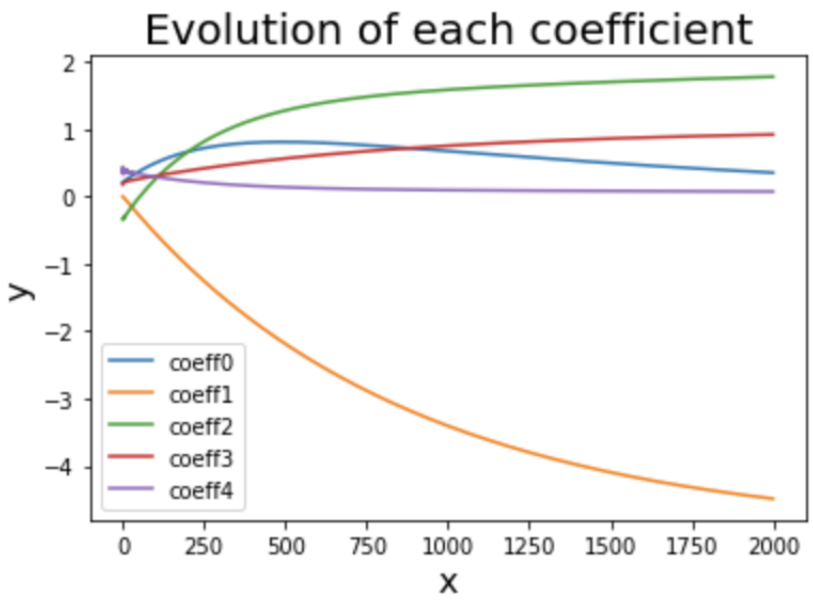

Figure 10: Graph for Question 1.11, visualization of the polynomial with obtained coefficient of Question 1.7

2 Question 2 [4/100 points]

2.1 Question 2.1

Overfitting is when the model perfectly recognizes the structure of the training data set. Here, the trained model fits slightly too well to the training data set. This means that the learned model cannot be applied to newer instances because the model learned beyond the underlying structure (e.g. also the outliers/noise (not representative data points) of the training data set). Overfitting can be detected if the loss for the validation and training data set differs greatly.

2.2 Question 2.2

Overfitting can be reduced by the following suggestions and/or techniques:

- Increase the training data set
- Data augmentation (meaning the training data ever so slightly changes during the training of the model)
- Controlling the iteration to tell the model s threshold when to stop (e.g. in terms of training data loss).
- Cross Validation meaning we can split up the training data into various parts and therefore iterative train and test on the split parts.
- Simplify the model by e.g. adapting the parameters in a neural network etc.

Note: for simplicity reasons we will not elaborate on all individual techniques to reduce over fitting.

Sources used:

- https://www.v7labs.com/blog/overfitting
- https://www.analyticssteps.com/blogs/5-machine-learning-techniques-solve-overfitting