Assignment 1

Student: Stefano Andreotti
Student's email: andres@usi.ch

Polynomial regression

2. The plot for the function $p(z) : \sum_{k=0}^{4} z^k w_k$ is reported in figure 1, using $w = [1, -1.5, -0.1, \frac{1}{30}]$. The functions is calulated using a linspace for the values on the z axis and using numpy polyval function for the values of p(z) as described in the definition. The function is plotted in the range -4, 4.

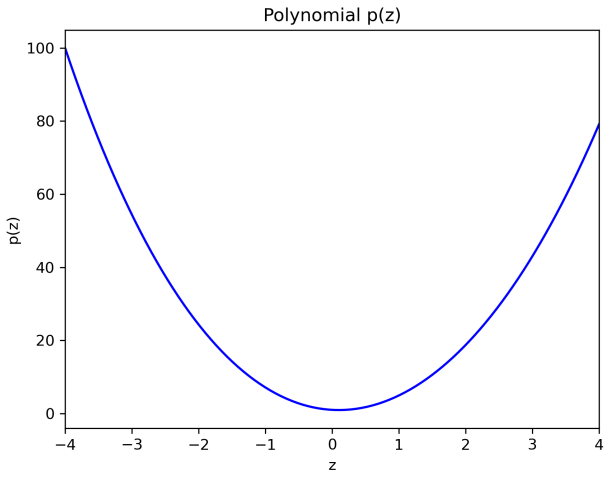


Figure 1: Function p(z)

3. The function create dataset is used to create training and validation points from a normal distribution, below it is reported the creation of z range with torch.rand and the scaling in the desidered interval using z range, where the element 0 is the minimum and the element 1 is the maximum.

```
z = torch.rand(sample_size)*(z_range[1]-z_range[0])+z_range[0]
```

After have created the z tensor, i have used it and a stack function to create the X dataset. The stack function concatenate the inner list of zs to the dimension 1.

```
X = torch.stack([torch.ones(sample_size), z, z**2, z**3, z**4],
 dim=1)
```

The last step is to calculate the real y function, in this case we have to flip the coeffs because they are stored from the coefficient of the term with the higher grade to the lower. Sum function will sum all the five terms calculated as coefficient multiplied by z to the power of i. The y tensor for the generated data is created from the real one adding noise with a normal distribution and a sigma factor.

```
y_hat = sum(coeff * z**i for i, coeff in enumerate(torch.flip
(coeffs, [0])))
y = y_hat + torch.normal( torch.zeros( sample_size ), sigma *
 torch.ones(sample_size))
```

4. In this task i have declared some variables as in the assignment track and using the function described at point 3 i have generated the training and validation data.

5. The function visualize data plot the generated training and validation data as points and the p(z) function. In figure 2 are plotted the points generated for the training and in the plot 3 there is the data for the validation. I stored the coefficients w going from the coefficient for the term 4 to the term 0 because numpy polyval function start to calculate from the greatest degree to the zero degree, so doing this way result more efficient as i must flip the coefficients only when showing the weights and in the creation of the dataset.

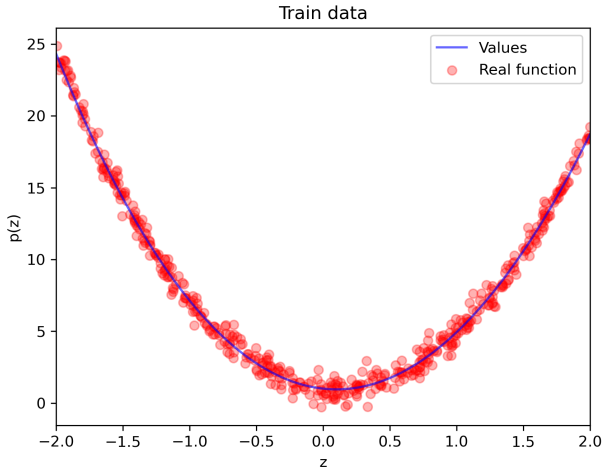


Figure 2: Train data

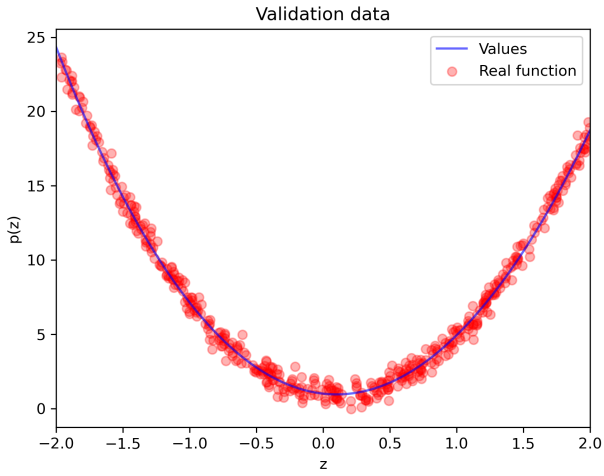


Figure 3: Validation data

6. The training of the model is done in the *regression_model* function, first of all we retrive the device to use (in my case mps), while it's not needed for the model i've done it as a good pratcice. Next the model it's initialized with 5 features in, because we have 5 coefficients as input and 1 output as the result of the function is just one value. Then is setted the loss function as the min square error and the optimizer as stochastic gradient descent. After the initialization phase the train and validation data are passed to the right device and the training loop is started. For each step the model is trained, setted the gradient to zero and calculated the loss then, in the same iteration start an evaluation phase where the values of training loss, validation loss anf the weights values are stored to make the plots. After the two phases of the iteration (training and evaluation) a new iteration start until the number if step is reached. In the code i left a part commented that i have used to test when loss decreasing rate is too low to allow the training phase to stop earlier and do not reach the number of step.

   In this section i've used learning rate equal to 0.02 becuse doing some trials this is the value where i've seen best results, going down from this value will led to too much steps needed for a good approximation, going up with the learning rate will make the weights change too much and too fast leading to the model to have a infinite loss caused by the weigths having a value too high.

   The bias value in this case should be setted as false because the function doesn't have any constant so the model has no bias to be aware of. I've used 600 steps, i think they are enough because doing more step will not make a significant difference in term of loss value, using 600 steps and learning rate euqal to 0.02 i have archived a final loss value of 0.0287.
   In figure 5 can be observed that the model approximate very well the

function so there is no need to go further using more steps

7. In figure 4 are reported the training and validation loss over the steps, as expected the function is decreasing over time showing that the model is every step more accurate.

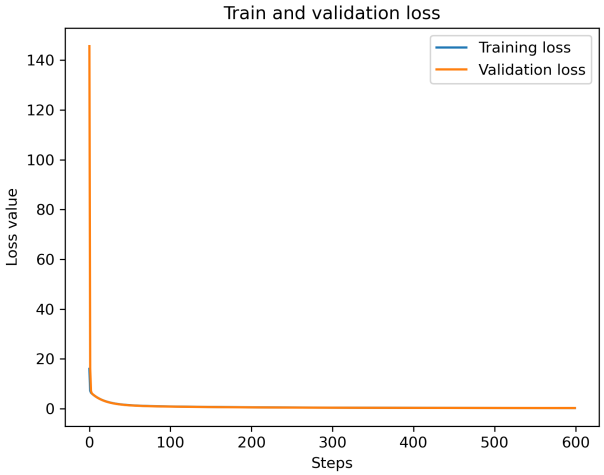


Figure 4: Train and validation loss

8. The function resulted from the model is compared with the real function in figure 5, as said before the two functions are very close meaning that the model has extimated a function near the real one.

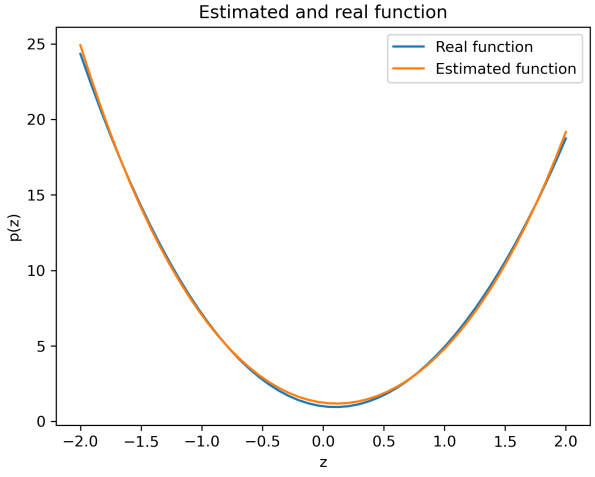


Figure 5: Estimated and real function

9. The model has extimated the weights near the real one, this is showed by figure 6 where the dashed lines are the real values of the coefficients while

the solid lines show the evolution of the weights over time, each coefficient is colored same as the corresponding estimated weight. As we can expect more step the model has done more precisely can approximate the real coefficients.

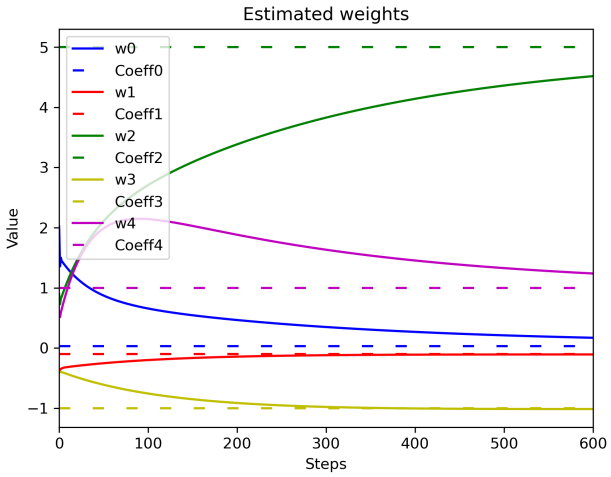


Figure 6: Weights value over steps

Question

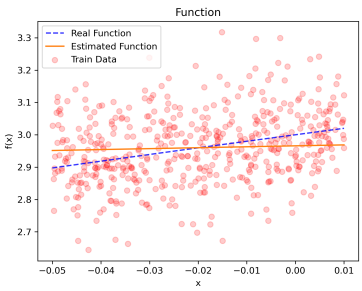
The manual method of changing manually the learning rate and do comparisons between wich use is a manual process, i've used this method in the project. An Adaptive method change the learning rate for each parameter based on how much they will change for example Adagrad maintains a running sum of squared gradients for each parameter, the learning rate is inversely proportional to the square root of the sum so a parameter that has a big gradient will have a small learning rate because it will change very fast, more over a parameter with small gradient will have a bigger learning rate.

Bonus question

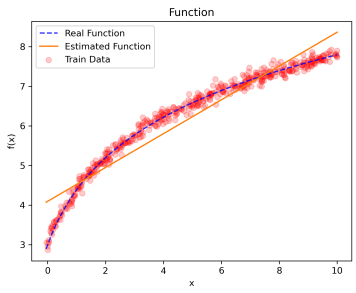
Plotting the function $f(x) = 2log(x+1)+3$ in the interval $[-0.05, 0.01]$ it looks almost as a straight line (figure 7a) so we can expect a very good approximation using the linear regression shown as the orange line, while the same function in the bigger interval $[-0.05, 10]$ clearly show a curve (figure 7b) so in this case we expect a worse approximation compared to the first interval also shown in orange. The data for training and validation are created in the same way as the data before, this points can be seen in the figures 7a 7b with the functions. The training process was done by the same function that is used for the polynomial regression but using different settings, first of all i setted the in features of the model to 1 instead of 5 because there is only one feature as input, than i setted 700 steps, a smaller learning rate of 0.01 and most importantly now the bias

must be setted to True because there is a constant of +3 in the function.

After some trials i got a loss value of 0.01 for the first interval while for the bigger interval i got a bigger loss value equal to 0.124 that is really higher compared to the first interval but not too much high overall. In figure 8 is reported the comparison between the loss of the function with the two different intervals.



(a) f(x) on $[-0.05, 0.01]$



(b) f(x) on $[-0.05, 10]$

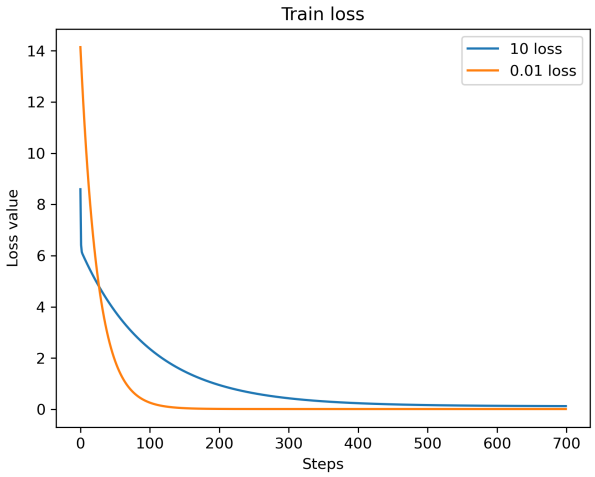


Figure 8: Loss on training with different intervals