Exercise Project X – Experimentation with Neural Network (NN) Mathematics

In this report I am using the example code from the Jupyter Notebook nn_math_tool_2.ipynb. I am going to try out behaviour changes when adjusting weights and other experiments that interest me. Due to my previous experience with Neural Network Theory in my home university I am not going to experiment a lot but rather try out some stuff that I didn't have the chance yet to do.

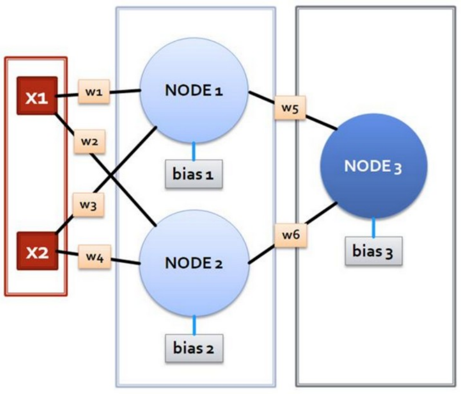
1. Baseline

First I need a baseline that I can compare to. So I am using the example weights and biases which are w1: 1.5, w2: 0.5, w3: -2, w4: -0.5, w5: 1.5, w6: 1.2, b1: 0.5, b2: -0.35, b3: 0.5. With those starting weights and biases I get the following result weights and biases after training (rounded): w1: -4.57, w2: -3.33, w3: -19.62, w4: -12.55, w5: -50.00, w6: -46.49, b1: -3.63, b2: -3.14, b3: 12.28

2. Adjusting w1 and w5

I am going to start by adjusting the weights w1 and w5. They are quite high initially with starting values of 1.5 compared to the rest of the starting values. I am going to lower the weight to 1.1 with the intention to improve NN performance. By lowering the values my idea is that the NN doesn't waste time trying to readjust those values to something lower and can use more Epochs on backpropagation at other places (afterthought: after looking at this again and at the starting result weights and biases, 1.5 doesn't seem that high of a weight in an absolute value sense and I was maybe on a bad approach here).
I got the following result (rounded): w1: -7.26, w2: 5.47, w3: -11.66, w4: -7.19, w5: -41.55, w6: -34.42 b1: -2.77, b2: -2.58, b3: 10.13

This increased the weights quite a bit into the positive. In the baseline values the weights for w1,w3,w5 are higher negative numbers than with 1.1 starting values. So by decreasing starting values I managed to decrease necessary adjustments to get the same result. This makes sense since w1,w3,w5 are all affecting or depending node1 as you can see in the image below. So by lowering the starting weight the node actually had more input than before which is quite an interesting effect since the bias also increased.



3. Increase w1 and w3 up to 10

Next I wanted to try out what happends when I use very high weights for the input affecting node1. So I am initializing w1,w3 with the value 10.
This results in the following weights and biases (rounded): w1: -0.08, w2: -7.57, w3: -5.13, w4: -12.60, w5: -168.30, w6: 1.2, b1: -4.54, b2: -4.38, b3: 12.28

As expected by increasing the input weight of Node1 very much this gets compensated by decreasing the output of Node1 very much. I expected this to happen but not into this extreme value. With w5 of -168 it barely gets taken into account compared to w6.

4. Identical weights and biases
Next I wanted to try out how the NN reacts to using all the same weights and biases. My prediction is that the result should be worse since the model needs more Epochs to adjust into the correct position. When using all identical weights this is the result (rounded): w1: 0.01, w2: 0.04, w3: -0.62, w4: -0.43, w5: -0.61, w6: -0.50, b1: 0.03, b2: -0.76, b3: 12.28
This is in my opinion a good result. It shows that with the same prerequisits the NN adjusts the weights properly and with a good result. So there are no unneccessarily big weights as a result due to different starting weights.

If you use starting all weights with value 2.0 instead of 1.0 this is the result:
Calculate factors also das verhältnis von den faktoren von 1 zu 2. The weights are different but the resulting biase of b3 is roughly the same. So the NN trained itself to the same result but with more adjusting needed due to higher weights complicating the training process.

5. Changing b1
Next I wanted to try out how changing the bias values impacts the NN. For this I am reverting the weights back to the starting point, so they have the following values: w1: 1.5, w2: 0.5, w3: -2, w4: -0.5, w5: 1.5, w6: 1.2
I am once again going to focus on Node1 so I am going to adjust b1 down to 0.1. This results in the following weights and biases: w1: -2.23, w2: -2.76, w3: -15.17, w4: -11.60, w5: -48.58, w6: -45.58, b1: -2.90, b2: -2.89, b3: 12.28
Compared to the baseline this increased the b1 up to -2.9 compared to -3.6. I would expect the output weight of the Node to decrease to compensate for the higher bias value inside the node. For this I compare w5 with the baseline value. In the baseline it resulted in a w5 of -50, here we now have w5 -48.58. So there is no compensation happening here, not what I was expecting. W6 stayed pretty much the same so no compensation happening here aswell. Since there is no compensation happening this means for my understanding that changing the bias increased NN performance and it got changed in the correct direction. I am not 100% sure if this is the correct conclusion to take out of this so I am trying it again with a different value.

This time I am going to change b1 down to -1.5, so quite a drastic change. This returned me the following final weights and biases: w1: 1.5, w2: 0.5, w3: -2.0, w4: -0.5, w5: 1.5, w6: 58.53, b1: -1.5, b2: -0.35, b3: 12.05. This resulted in a change in b3, the final Node.

6. Changing Range, Epochs and Learning Rate
When experimenting with a different range for the train data I recognized this:

I set the range from 100 up to 500 and the performance decreased significantly. The loss went from ~32 up to ~38. When decreasing it to 50 I got ~34, so also slightly worse. Down to 50 it gets significantly worse, ~37. So 100 sems to be quite a sweet spot for training data. When I increase epochs up to 2000 and range to 200 it is still up to ~37 even though the NN has more time to adjust to it. So it probably doesn’t scale linear. When giving the NN 5000 Epochs and range 200 it still doesn’t go down below 37, so the Epochs probably don’t have an effect on this. It just gets harder for the NN to predict it. Maybe adjusting learning rate accordingly helps it again. I increased it up to 0.05 but it didn’t help. It actually worsened it down to ~38. Maybe lowering the learning rate down to 0.0125 helps it by making it learn slower. Now I got ~36.5 as a result, so definitely an improvement. It`s also possible that the NN structure is too simple for that many Epochs / that big of an range.

7. Learning-Rate, Gradient-Descent and Loss-function

Backpropagation goes backwards through the NN and calculates the derivatives using the ReLu activation functions. The gradient descent then tries to calculate better weights by trying to minimize the loss for the NN using the following formula:
new_weight = old_weight – learning_rate *gradient (line 31 ipynb)
The gradient gets calculated for each weight of a node and is basically the derivative of the loss function which can be calculated using activation functions like activation_ReLu_partial_derivative in our case. As already observed in 6 the learning rate impacts how much the NN learns from each single data row. The loss function calculates the “wrongness” of the answer from the NN. This is an important metric to compare performance and improve upon it.

8. Conclusion

More or les regardless of the initial weights and biases the NN tends to strive towards a similar result. This is useful for a lot of applications in Data Science where a general solution for learning trends in data can be really useful. The NN seems like a decent solution and can be finetuned to each problem and dataset without much effort.