Exercise Project X – Experimentation with NN Mathematics

This project was my first deep dive into understanding how a neural network works by actually building one from scratch. It was a really hands-on way to learn, eventhough I made some mistakes along the way! But, experimenting with the code and seeing how small changes affected the network helped me understand a lot more about things like weights, biases, the loss function, and gradient descent. Below, I'll talk about what I tried, where I messed up, and what I learned from it.

Starting weights and biases

One of the first things I played around with was changing the starting values of the weights and biases. At first, I set all the weights and biases to 1 because I thought, "Why not keep it simple?" But then the network completely failed to train properly. The loss barely changed, and I was super confused. After figuring out what was wrong (using my personal assistant, ChatGPT 😊), I realized this was because all the neurons were updating in the exact same way (symmetry problem), so the network couldn't learn properly.

To fix this, I added a randomizer to initialize the weights and biases. I tried small random values, and that worked much better! When I tried using very large random values (like weights around 100), the network went crazy – the loss kept going up, and I realized it was because the gradients became too big (exploding gradients). This taught me that starting values really matter, even for a very simple network.

Learning rate experiments

The learning rate was another thing I played around with a lot. I tried realy small values like 0.001, and it felt like the network was learning forever. The loss was going down, but it took way too long. Then I tried a bigger value like 0.2, but then the loss just started bouncing all over the place or even going up instead of down. I think I broke the training process a few times because I didn’t realize that too high a learning rate can make the network unstable.

Eventually, I found that something between 0.01 and 0.05 worked best for this specific neural network. It was a balance where the loss was going down steadily, and the network didn’t feel too slow or unstable. I learned here that the learning rate is super important, and finding the right value can be tricky.

Data distribution and dataset size

I also experimented with the data generator to see how different kinds of data would affect the network. At first, I used a small dataset of only 30 points, and the network didn’t perform well at all. The predictions were all over the place, and the loss didn’t go down much. Then I increased the dataset to 500 points, and suddenly everything worked much better. I guess the smaller dataset didn’t have enough information for the network to learn properly.

I also tried changing the data distribution. With uniform data, the network learned pretty consistentely, but when I skewed the data (like generating more points near one side), the predictions got worse. It made me realize how important it is to have good, balanced data for training.

Mistakes and learning

I made a lot of stupid mistakes while experimenting with this. One time, I forgot to update teh weights during training (I commented out the line by accident), and I couldn’t figure out why the loss wasn’t changing. Another time, I accidentally set the bias to a fixed value and didn’t realize it was supposed to update too. These mistakes were frustrating, but every time I fixed something, I felt like I understood the code and the math behind it a little better.

Manual neural networks vs frameworks

After doing all this manually, I can totally see why frameworks like TensorFlow and PyTorch are so popular. Writing everything from scratch was fun and helped me learn, but it’s really easy to mess something up, and debugging takes a lot of time. Plus, this is just a super simple network – I can’t even imagine building something more complex like a convolutional neural network without a framework.

That said, doing it this way gave me a much better idea of how things like gradient descent, backpropagation, and loss functions actually work under the hood. I don’t think I would’ve understood these concepts as well if I had just used TensorFlow from the start.

Final thoughts

In the end, this project taught me a lot about how neural networks learn. The weights and biases are adjusted through gradient descent to minimize the loss function, and the learning rate controls how quickly (or slowly) this happens. I also learned that small changes to the starting conditions or hyperparameters can have a huge impact on the training process.

This project wasn’t perfect – I made a bunch of mistakes, and sometimes I didn’t know why something wasn’t working right away. But I think that’s part of the learning process. Now I feel like I have a better understanding of what’s going on when I use tools like TensorFlow in the future, and I can apreciate how much they simplify things.