Exercise Project X

Student: Áron Harmadás

Course: Deep Learning

The purpose of this exercise project was to experiment with a very simple neural network that was built completely from scratch during the lectures. Instead of using high-level libraries such as TensorFlow or PyTorch, all parts of the neural network were implemented manually using basic Python code. This includes the forward pass, loss calculation, and backpropagation with gradient descent.

The goal of this work was not to achieve high performance or build a realistic machine learning model, but to better understand how neural networks work internally. By manually calculating outputs, loss values, and partial derivatives, I was able to gain a much deeper understanding of how learning actually happens inside a neural network.

This document describes my personal experiments, observations, and learning experiences while working with the provided neural network code.

2. Description of the Neural Network

The implemented neural network is a very small feed-forward neural network with the following structure:

- Two input values (input1 and input2)
- One hidden layer with two neurons
- One output neuron
- ReLU activation function used in all neurons
- Mean Squared Error (MSE) as the loss function
- Gradient descent as the optimization method

Each neuron computes a weighted sum of its inputs, adds a bias, and applies the ReLU activation function. The output of the final neuron is compared to the true target value, and the error is calculated using the MSE formula:

$$Loss = (predicted\_value - true\_value)^2$$

After the forward pass and loss calculation, the network performs backpropagation. During backpropagation, partial derivatives of the loss function with respect to each weight and bias are calculated manually using the chain rule. These derivatives are then used to update the parameters using gradient descent.

3. Forward Pass Observations

During the forward pass, I noticed how each layer builds on the previous one. The first hidden layer already combines the raw input values with weights and biases, and the ReLU activation function removes all negative values.

One important observation was how sensitive the output is to small changes in weights and biases. Even though the network is extremely small, changing a single weight slightly can already cause a noticeable difference in the final prediction.

This helped me understand that neural networks are essentially large chains of mathematical operations, where small numerical changes can propagate and grow through the network.

4. Experiments With Learning Rate

One of the most interesting parameters to experiment with was the learning rate (LR). I tested several different learning rate values, including:

- 0.005
- 0.01 (original value)
- 0.1
- 0.25

With a very small learning rate (0.005), the updates to the weights and biases were very small. This made the learning process stable, but also slow. The loss decreased only slightly after one update step.

With a moderate learning rate (0.01), the network behaved reasonably. The loss decreased and the weight updates were noticeable but not extreme.

When I increased the learning rate to higher values such as 0.1 or 0.25, the learning process became unstable. The weight updates were too large, and in some cases the network overshot the optimal values. This sometimes caused the loss to increase instead of decrease.

From this experiment, I learned that the learning rate directly controls how fast and how safely the neural network learns. A learning rate that is too small makes learning inefficient, while a learning rate that is too large can break the training process entirely.

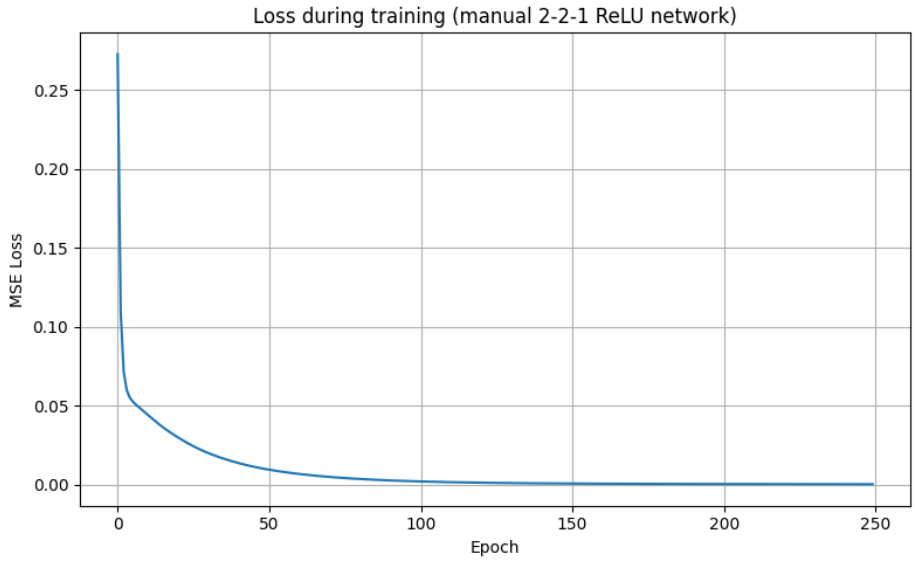

Figure 1. *Loss (MSE) over training epochs while experimenting with learning rate and manual gradient descent in the 2–2–1 ReLU network.*

5. Experiments With Initial Weights and Biases

Another important experiment was changing the initial values of weights and biases. I tested different scenarios, such as:

- All weights initialized with the same small value
- Mixed positive and negative values
- Larger absolute values for weights and biases

I noticed that the starting values can have a significant effect on the behavior of the network. In some cases, the network produced very large outputs, which increased the loss. In other cases, the ReLU activation function caused some neurons to output zero consistently.

This experiment helped me understand why neural network frameworks usually use random initialization. If all weights start with the same value, the neurons may behave too similarly, which reduces the learning capability of the network.

6. ReLU Activation and Dead Neurons

While experimenting with biases, I discovered the concept of "dead ReLU neurons". If the input to a ReLU neuron is always negative, the output of that neuron is always zero. Since the derivative of ReLU is also zero in this case, the neuron stops learning completely.

For example, setting a very negative bias value caused the neuron to never activate. As a result, its gradient became zero and its weights were no longer updated during backpropagation.

This was an important realization, because it showed a real limitation of the ReLU activation function. Even though ReLU is simple and efficient, it can completely disable neurons under certain conditions.

7. Understanding Backpropagation and Gradient Descent

Manually implementing backpropagation was the most educational part of this project. By calculating the partial derivatives step by step, I finally understood what gradient descent actually does.

Each derivative tells how much the loss would change if a specific weight or bias were increased slightly. Gradient descent then updates the parameter in the opposite direction of the gradient, which reduces the loss.

Seeing this process numerically helped me understand that learning is not magic. The neural network is simply adjusting numbers in small steps to reduce an error value.

This experiment also made it clear why backpropagation becomes very complex in larger networks. Even in this small example, the number of derivative calculations is already quite high.

8. Sensitivity of a Small Network

Because the neural network is very small, it is extremely sensitive to changes. A small modification in one weight can lead to a large change in the output and loss.

This sensitivity made it easier to observe how learning works, but it also showed why training real neural networks can be difficult. In larger networks, this sensitivity can lead to problems such as exploding or vanishing gradients.

9. Comparison With TensorFlow and Other Frameworks

After working with this manual implementation, I gained a much stronger appreciation for machine learning frameworks like TensorFlow and PyTorch.

While it is possible to build and train a neural network manually, it quickly becomes impractical for anything beyond simple experiments. Frameworks handle weight initialization, backpropagation, optimization, and numerical stability automatically.

However, this manual approach was extremely useful for learning purposes. It helped me understand what happens “under the hood” when using high-level libraries, instead of treating them as black boxes.

10. Limitations of This Neural Network

This neural network has several limitations:

- It uses only one training example
- It has no batching or epochs
- It has no regularization
- It uses only ReLU activation
- It is not scalable

Despite these limitations, the network is perfect for educational purposes. Its simplicity makes it possible to clearly observe how learning works step by step.

11. Personal Learning Experience and Conclusion

This exercise significantly improved my understanding of neural networks. Before this project, concepts such as loss functions, gradient descent, and backpropagation felt abstract. By implementing everything manually, these concepts became concrete and intuitive.

I learned that neural networks do not “understand” data in a human sense. They simply adjust numerical parameters to minimize an error function. I also realized how important design choices such as learning rate, activation functions, and weight initialization are.

Overall, this experiment was challenging but very rewarding. It gave me a solid foundation for understanding more advanced neural network models and frameworks in the future.