Neural Networks

1st Assignment, Multilayer Perceptron, April 2025

Michal Švec

1 Introduction

The goal of this project was to train a neural network to classify 2D points into three classes: A, B, and C. We implemented a neural network with two hidden layers containing 12 and 8 neurons, respectively. A grid search was used to evaluate 216 different hyperparameter combinations. The best-performing model achieved a test accuracy of 98.40%.

2 Dataset

We use the provided dataset of 2D points, which can be classified into three classes: A, B, and C. The train dataset contains 8000 samples. From this, we created validation dataset whose total size is 20% of the original training dataset.

The original dataset is not balanced, however, our split is stratified, meaning we select 20% of the samples from each class to preserve the class distribution. In addition, all labels were one-hot encoded.

Split	**Samples**
Train	6400
Validation	1600
Test	2000

Table 1: Dataset split

3 Architecture

In this project, I decided to use a neural network with two hidden layers consisting of 12 and 8 neurons, respectively. Even a single hidden layer yielded solid results, so I extended the architecture to two layers. ReLU was used as the activation function in all hidden layers, while the output layer uses the Softmax activation function. I chose ReLU because I read it is commonly used in practice. These parameters were fixed, and I experimented with other hyperparameters, which will be discussed later.

The neural network was trained using mini-batch gradient descent, with the batch size fixed at 32. The solution also uses gradient clipping. If the norm is greater than 1, the gradient multiplied by $\frac{1}{norm}$. I implemented this because I often encountered strange behavior during training, and this fixed it. During training, the samples are randomly shuffled in each epoch and that means the training process is not completely deterministic.

4 Experiments

In my implementation, there are six parameters that can be tuned, so I chose the best parameters using a standard grid search.

Data normalization

For the normalization of the input data, there are two options, either z-score normalization or min-max normalization.

$$\begin{aligned} x_i' &= \frac{x_i - \mu}{\sigma} && \text{(z-score)} \\ x_i' &= \frac{x_i - x_{\min}}{x_{\max} - x_{\min}} && \text{(min-max)} \end{aligned}$$

Weights initialization

Since all layers use the ReLU activation function and the output layer uses Softmax, I read that the most appropriate weights initialization method is known as He initialization (also known as Kaiming). I use two variants, namely He-normal and He-uniform, which are defined as follows:

$$\begin{aligned} W_{ij} &\sim \mathcal{N}\Big(0,\ \frac{2}{fan_{in}}\Big), \\ W_{ij} &\sim \mathcal{U}\Big(-\sqrt{\frac{6}{fan_{in}}},\ \sqrt{\frac{6}{\text{fan}_{\text{in}}}}\Big), \end{aligned}$$

where fan_{in} represents how many inputs each neuron in the layer is receiving.

Leaning rate

There are three hyperparameters related to the learning rate. The first one is learning rate decay:

$$\begin{aligned} \eta_t &= \eta_0 \cdot e^{-kt} && \text{(exponential decay)} \\ \eta &= k \cdot \eta && \text{(step decay)} \end{aligned}$$

where t is the epoch, and both formulas are parameterized by another hyperparameter k, which determines the decay factor. The third hyperparameter is the initial learning rate. The values of k and the initial learning rate in our experiments took the following values:

$$\begin{aligned} k &\in \{0.5, 0.25, 0.125\} \\ \text{init\_lr} &\in \{0.1, 0.05, 0.025\} \end{aligned}$$

5 Results

In total, we evaluated 216 configurations. Among all the configurations, the model with the following parameters achieved the best performance on the validation dataset:

$$\begin{aligned} \text{init\_lr} &= 0.1 \\ \text{lr\_decay} &= \text{step} \\ \text{decay\_k} &= 0.5 \\ \text{weight\_init} &= \text{he-uniform} \\ \text{epochs} &= 75 \\ \text{normalization} &= \text{z-score} \end{aligned}$$

Its accuracy on the validation and test datasets reached 98.50% and 98.40%, respectively. Due to space limitations, we include only the 10 best and 10 worst performing configurations below. The complete CSV file with all results is provided in the project files.

init_lr	**lr_decay**	**decay_k**	**weight_init**	**epochs**	**normalization**	**accuracy**
0.1	step	0.5	he-uniform	75	z-score	0.9850
0.1	exponential	0.125	he-uniform	50	z-score	0.9837
0.1	exponential	0.125	he-normal	50	z-score	0.9819
0.1	step	0.5	he-normal	75	z-score	0.9819
0.1	step	0.5	he-normal	25	z-score	0.9806
0.1	exponential	0.125	he-uniform	75	z-score	0.9794
0.1	exponential	0.25	he-uniform	50	z-score	0.9787
0.1	step	0.5	he-uniform	25	z-score	0.9781
0.1	exponential	0.125	he-uniform	25	z-score	0.9775
0.1	exponential	0.25	he-normal	25	z-score	0.9769
⋮						
0.025	exponential	0.5	he-normal	50	min-max	0.5716
0.025	step	0.25	he-normal	75	min-max	0.5685
0.05	exponential	0.5	he-uniform	50	min-max	0.5666
0.05	exponential	0.25	he-normal	25	min-max	0.5597
0.025	exponential	0.25	he-normal	25	min-max	0.5528
0.025	step	0.125	he-normal	50	min-max	0.5478
0.025	step	0.125	he-normal	25	min-max	0.5403
0.05	step	0.125	he-normal	50	min-max	0.5285
0.025	step	0.125	he-normal	75	min-max	0.5228
0.025	step	0.125	he-uniform	25	min-max	0.5078

Table 2: The 10 best and 10 worst configurations based on validation accuracy

It is very difficult to infer any strong patterns from these results, as the values in the table do not reveal significant trends. However, one observation is that an initial learning rate of 0.1 appears in every top-performing model. Additionally, z-score normalization seems to outperform min-max normalization in this case. The worst-performing models use min-max normalization and a low initial learning rate. This likely contributed to the poor accuracy, as the models may have failed to learn effectively due to the very low learning rate.

Additional results for the best performing model

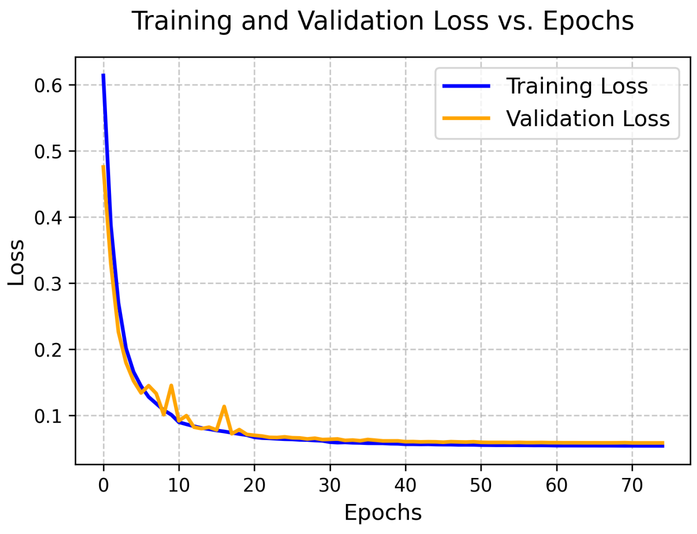

Figure 1: Train and validation loss during training

In the picture, we can see that the training and validation losses are nearly the same, however, there are some spikes in the validation loss. These spikes are likely caused by the higher learning rate at the beginning. The best-performing model uses the largest initial learning rate of 0.1. When using a lower learning rate, the validation loss curve is smoother. Overall, it seems that the model is learning and generalizing well.

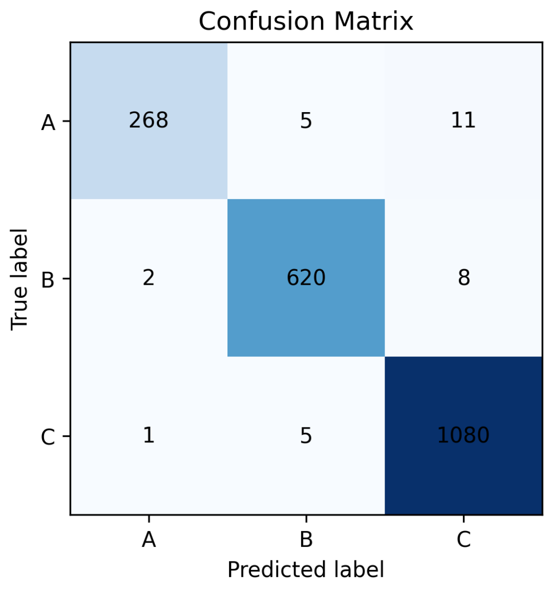

Figure 2: Confusion matrix of test dataset

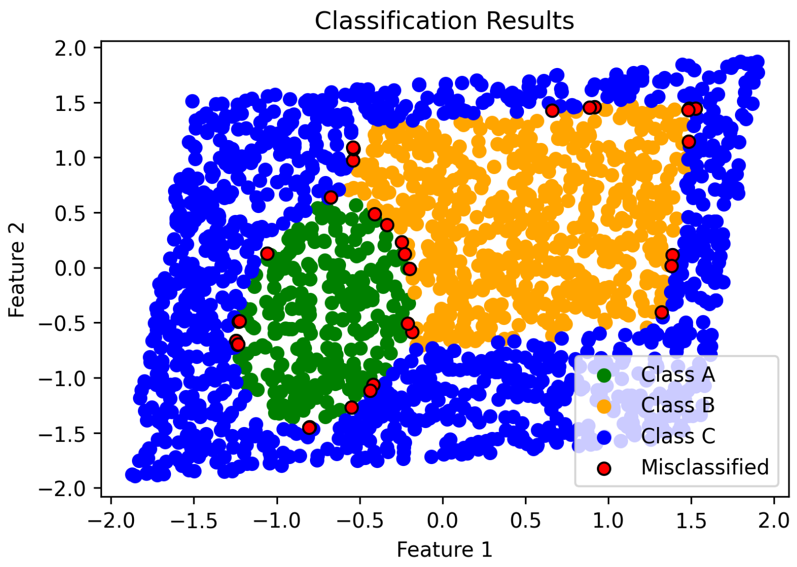

Figure 3: Classification results of test dataset

Based on the confusion matrix and the classification results plot, we can see that only 32 elements were misclassified. These misclassifications occur at the borders of the clusters, which is entirely reasonable.