Bahri Batuhan BİLECEN
22201372
12.11.2023

EEE443-543 Neural Networks Mini-Project 1 Report

Task: Hyperparameter optimization of a 2-layer perceptron trained on MNIST dataset. The results of all 10 steps are presented in the **Results** section, and the observations are listed on **Comments and Observations**. For implementation, see **Implementation** and **the well-commented .py** file.

Implementation

The implementation is made from scratch; hence the matrix notations are compliant with the course content. I utilized `torch` library to migrate the data to GPU and perform the training sessions faster with `CUDA`, but I did not utilize any of its automatic gradient functionalities. In other words, using `torch` is identical to using `numpy` but with GPU in this scenario; and in fact, most of the matrix functions are the same in `torch` & `numpy` in terms of valid input arguments and operation behavior.

See the attached `.py` file for more comments on my backpropagation implementation.

Results

The quantitative results are given in Tab. 1. The best-performing model achieves **98.63%** accuracy on the MNIST test set.

Table 1. 2-layer perceptron training hyperparameter configurations and test results for MNIST dataset. BS is batch size, Act is activation type, R+S and T+T are ReLU+Sigmoid and Tanh+Tanh respectively, N is the number of neurons in the hidden layer, and LR is the learning rate. The best score and the associated model are given in bold.

Run ID	Epoch	L2 reg	BS	Act	N	LR	Accuracy
1	50	0	1	R+S	300	0.01	98.32
2	50	0	1	R+S	300	0.05	98.54
3	50	0	1	R+S	300	0.09	98.46
4	50	0	1	R+S	500	0.01	98.39
5	50	0	1	R+S	500	0.05	98.60
6	50	0	1	R+S	500	0.09	98.56
7	50	0	1	R+S	1000	0.01	98.44
8	50	0	1	R+S	1000	0.05	98.60
9	**50**	**0**	**1**	**R+S**	**1000**	**0.09**	**98.63**
10	50	0	1	T+T	300	0.01	98.11
11	50	0	1	T+T	300	0.05	67.03
12	50	0	1	T+T	300	0.09	27.94
13	50	0	1	T+T	500	0.01	98.00
14	50	0	1	T+T	500	0.05	59.21
15	50	0	1	T+T	500	0.09	30.38
16	50	0	1	T+T	1000	0.01	98.01
17	50	0	1	T+T	1000	0.05	39.30

18	50	0	1	T+T	1000	0.09	9.83

9_1	50	0	10	R+S	1000	0.09	98.42
9_2	50	0	50	R+S	1000	0.09	97.92
9_3	50	0	100	R+S	1000	0.09	96.99

9_4	50	0.01	1	R+S	1000	0.09	nan
9_5	50	0.001	1	R+S	1000	0.09	85.7

In addition, some comparative training loss plots are given in Fig. 1-4. Since the visualization of the loss per sample is very noisy, outlier removal and running average smoothing is applied beforehand:

```python
"""
Simple outlier removal implementation using the mean and standard deviation.
"""
def replace_outliers_with_neighbors(data, m=1.5):
    mean_val = np.mean(data)
    std_val = np.std(data)
    # Find the indices of outliers
    outliers_indices = np.where(abs(data - mean_val) > m * std_val)[0]
    # Replace outliers with neighboring values
    for idx in outliers_indices:
        if idx == 0:
            data[idx] = data[idx + 1]
        elif idx == len(data) - 1:
            data[idx] = data[idx - 1]
        else:
            data[idx] = (data[idx - 1] + data[idx + 1]) / 2
    return data
"""
EMA implementation according to
https://github.com/tensorflow/tensorboard/blob/34877f15153e1a2087316b9952c931807a122aa
7/tensorboard/components/vz_line_chart2/line-chart.ts#L699
"""
def smooth(scalars: list[float], weight: float = 0.9) -> list[float]:
    last = 0
    smoothed = []
    num_acc = 0
    for next_val in scalars:
        last = last * weight + (1 - weight) * next_val
        num_acc += 1
        # de-bias
        debias_weight = 1
        if weight != 1:
            debias_weight = 1 - math.pow(weight, num_acc)
        smoothed_val = last / debias_weight
        smoothed.append(smoothed_val)
    return smoothed
```

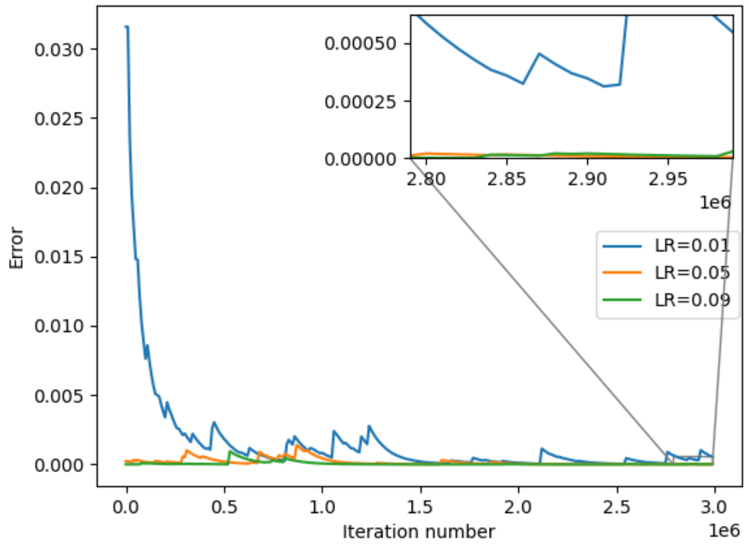

Figure 1. Training losses for run ID’s 1, 2, and 3 (R+S, 300 neurons, different LR).

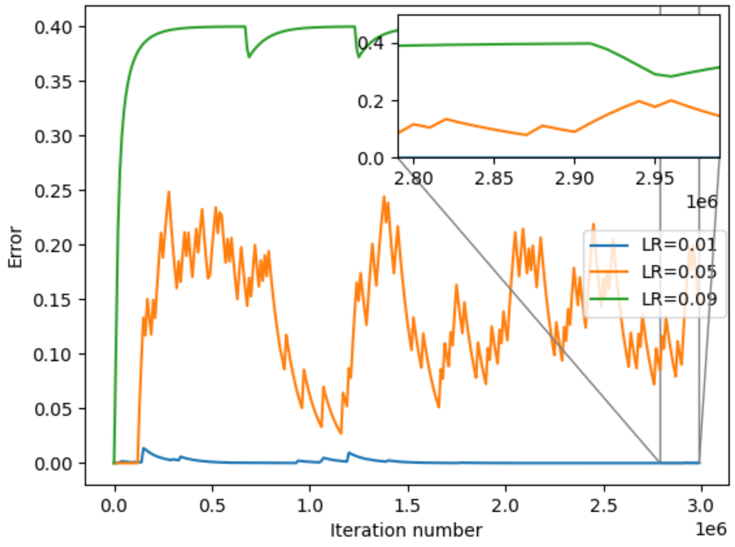

Figure 2. Training losses for run ID’s 10, 11, and 12 (T+T, 300 neurons, different LR).

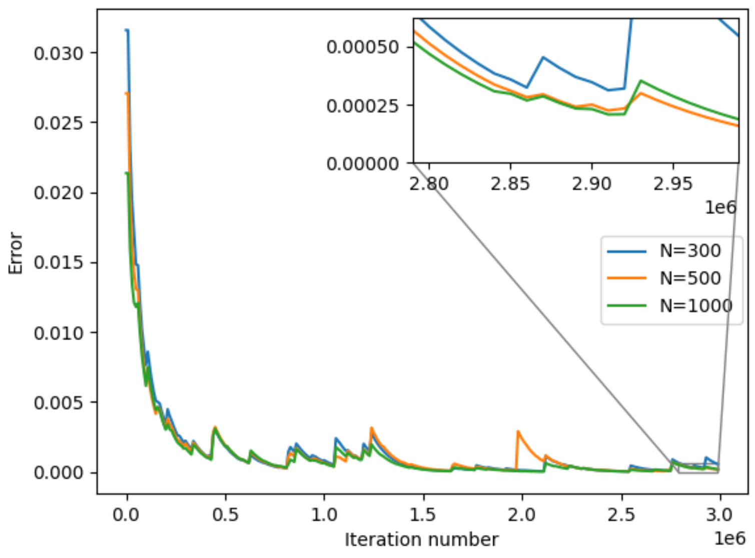

Figure 3. Training losses for run ID’s 1, 5, and 7 (R+S, LR=0.01, different N).

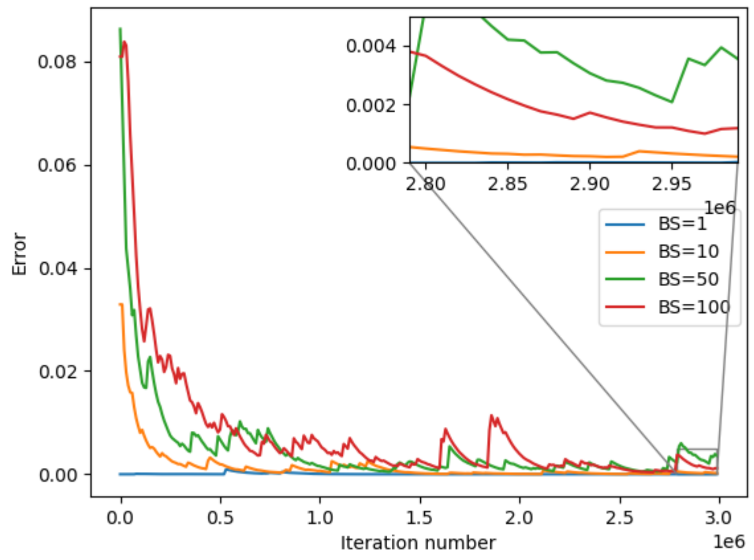

Figure 4. Training losses for run ID’s 9, 9_1, 9_2, and 9_3 (R+S, LR=0.09, N=1000, different BS).

Comments and Observations

I briefly list my observations below:

- Fig. 1-4 along with Tab. 1 reveal that lower training loss indicates better test performance (unless there is an overfitting problem-and there probably is not, see L2 regularization part).

- Increasing the learning rate benefits the models with ReLU+sigmoid but hinders the models with tanh. tanh appears to be more sensitive to learning rate. In addition, to fully benefit from tanh, we should have normalized our input data between [-1,1] instead of [0,1].
- Increasing batch size did not help the training, neither in training loss nor test score.
- L2 regularization also did not facilitate the training, either. L2 regularization is mostly utilized for the elimination of overfitting and can be seen as if we inject noise to our training data. We do not have a validation set, so we may not be able to fully conclude whether our model overfits or underfits. However, as we do not get better results with the L2 regularization, I am more inclined to say that our model is probably not overfitted.