Warsaw University of Technology

FACULTY OF
ELECTRONICS AND INFORMATION TECHNOLOGY

EARIN

Laboratory 5
Group 21
Variant 1
Artificial Neural Networks

Tymon Żarski 310992
Bartosz Peczyński 310703

WARSAW April 25, 2024

Contents

1. Introduction

1.1. Assigned task

Implement a multilayer perceptron for image classification. The neural network should be trained with the mini-batch gradient descent method. Remember to split the dataset into training and validation sets. The main point of this task is to evaluate how various components and hyperparameters of a neural network, and the training process affects the network's performance in terms of its ability to converge, the speed of convergence, and final accuracy on the training and validation sets.

Use the MNIST dataset. Evaluate at least 3 different numbers/values/types of:

- Learning rate,
- Mini-batch size (including a batch containing only 1 example),
- Number of hidden layers (including 0 hidden layers - a linear model),
- Width (number of neurons in hidden layers),
- Loss functions (e.g., Mean Squared Error, Mean Absolute Error, Cross Entropy).

1.2. Running of the program

To run the program, it is needed first to install all dependencies from *requirements.txt* using *pip3 install -r "requirements.txt"* and afterwards run *main.py* from CLI, using Python 3, for example: *python3 main.py*.

2. Network Archtecture

To perform all of the experiments and gather enough data, we created the class `NeuralNetwork` that extends `torch.nn.Module` and is structured as follows:

Listing 1. Definition of proposed simple ANN

```
class NeuralNetwork(nn.Module):
    def __init__(
        self,
        num_classes,
        num_hidden_neurons,
        num_hidden_layers=1,
        loss_finction_name="CrossEntropyLoss",
    ):
        super().__init__()

        self.encoder = nn.Sequential(
            nn.Flatten(),
            nn.Linear(28 * 28, num_hidden_neurons),
            nn.ReLU(),
        )

        self.hidden_layers = nn.ModuleList(
            [
                nn.Linear(num_hidden_neurons, num_hidden_neurons)
                for _ in range(num_hidden_layers)
            ]
        )

        potential_decoder = [
            nn.Linear(num_hidden_neurons, num_classes),
        ]

        if loss_finction_name != "CrossEntropyLoss":
            potential_decoder.append(nn.LogSoftmax(dim=1))

        self.decoder = nn.Sequential(*potential_decoder)

    def forward(self, x):
        """Forward pass"""

        x = self.encoder(x)
        for hidden_layer in self.hidden_layers:
            x = torch.relu(hidden_layer(x))
        x = self.decoder(x)
        return x
```

- **Input Layer:** The input layer includes a flattening operation, which transforms the 2D 28x28 pixel grayscale MNIST images into a 1D tensor of 784 elements (since $28 \times 28 = 784$). This is essential for processing the input in subsequent fully connected layers.
- **Hidden Layers:** The architecture supports an arbitrary number of hidden layers specified by the user (including 0). Each of these hidden layers is a fully connected layer followed by a ReLU activation. The number of neurons in each layer is consistent and matches the number specified for the first hidden layer. This uniformity in layer size simplifies the model's complexity and parameter tuning. In a perfect world scenario, subsequent hidden layers should follow the powers of 2, but in our case, when we aim to conduct the experiments on the constant width of the hidden layers, it is simplified.
- **Output Layer:** The output layer is a fully connected layer that maps the last hidden layer's output to the number of classes in the dataset (10 for MNIST). For classification tasks where the output is a class label, a softmax activation function is typically used to convert the logits to probabilities. However, when using the Cross-Entropy Loss function in PyTorch, the softmax operation is integrated into the

loss function for numerical stability. If a different loss function is used, a LogSoftmax layer is added to ensure the network's output can be interpreted as probabilities.

2.1. Loss Function

The network supports various loss functions that are to be specified during configuration. Depending on the choice of the loss function, the network configuration is adjusted dynamically. For example, when not using `CrossEntropyLoss`, a `LogSoftmax` layer is appended to the network's structure to handle the output probabilities appropriately. For our analysis, we used **CrossEntropyLoss**, **MultiMarginLoss** and **NLLLoss**.

2.2. Flexibility

This architecture is designed flexibly, allowing for easy adjustments of the network's depth, width, and choice of loss function based on experimental needs or specific performance goals. This makes it adaptable for exploring how different network configurations affect learning effectiveness, convergence speed, and overall model accuracy.

3. Experiments setup

The experiments are configured to automatically tune the hyperparameters of a neural network trained on the MNIST dataset. The hyperparameters under consideration are:

- **Epochs:** The maximum number of epochs for training, denoted as `max_num_epochs`.
- **Number of Classes:** Fixed at 10, corresponding to the 10 digits in MNIST.
- **Number of Hidden Neurons:** Set to powers of two, ranging from 2^6 to 2^8 (i.e., 64 to 256 neurons).
- **Number of Hidden Layers:** Three choices are tested—0, 1, and 5 layers, to examine the effect of depth in network architecture.
- **Learning Rate (lr):** Log-uniformly distributed between 10^{-4} and 10^{-1}, allowing for a wide exploration of learning dynamics.
- **Batch Size:** Configured to select among 1, 16, or 32, exploring the impact of batch size on the training process.
- **Criterion:** The loss function choices include CrossEntropyLoss, MultiMarginLoss, and NLLLoss, each appropriate for classification tasks but influencing convergence and performance differently.

Resource Allocation and Scheduler

Each trial in the hyperparameter tuning process is allocated 1.5 CPUs and, optionally, GPUs based on availability. The `ASHAScheduler` is employed to manage and prune the trials:

- **Metric:** The scheduler optimizes for the "loss" metric.
- **Mode:** Set to "min", which aims to minimize the loss.
- **Max_t:** Corresponds to the maximum number of epochs per trial, `max_num_epochs`.
- **Grace Period:** Trials are allowed a minimum of 3 epochs before they can be stopped if they are not promising.
- **Reduction Factor:** Specifies that the number of trials is halved in each round of pruning, focusing resources on the most promising configurations.

The configuration and scheduler setup ensure a systematic exploration of the parameter space, enhancing the chances of identifying optimal network settings while efficiently using computational resources.

4. Hyperparameter impact

4.1. Reproducibility

In PyTorch, ensuring reproducibility involves setting a fixed seed for the random number generators used in the library. This controls various random elements that can influence the outcomes of training a neural network, such as parameter initialization and data shuffling. To achieve reproducibility in your experiments, we set the seed to a specific value, in our case **0**, using *torch.manual_seed(0)* and *raytorch.enable_reproducibility(0)*. Those functions set the seed for generating random numbers and can help obtain the same results every time the experiment is conducted. This practice is vital for validating experimental results and comparisons in scientific research where consistency is critical.

4.2. Detecting best parameters

The best trial configuration and its outcomes generated by our script that uses scheduler from **ray** library derived a result after studying 25 trials and max epoch of 10.

Parameter	**Value**
Epochs	10
Number of Classes	10
Number of Hidden Neurons	256
Number of Hidden Layers	1
Learning Rate	0.00223616911307715
Batch Size	1
Loss Function	MultiMarginLoss

Table 4.1. Best trial configuration

- Final Validation Loss: **0.007310299732959208**
- Final Validation Accuracy: **96.79%**

The following figure describes how the tuner managed the training of the Trials. Data is plotted as model accuracy against the epoch for each trial. To take further insides, we can check the parameters of each trial by finding suitable for in the Table 4.2.

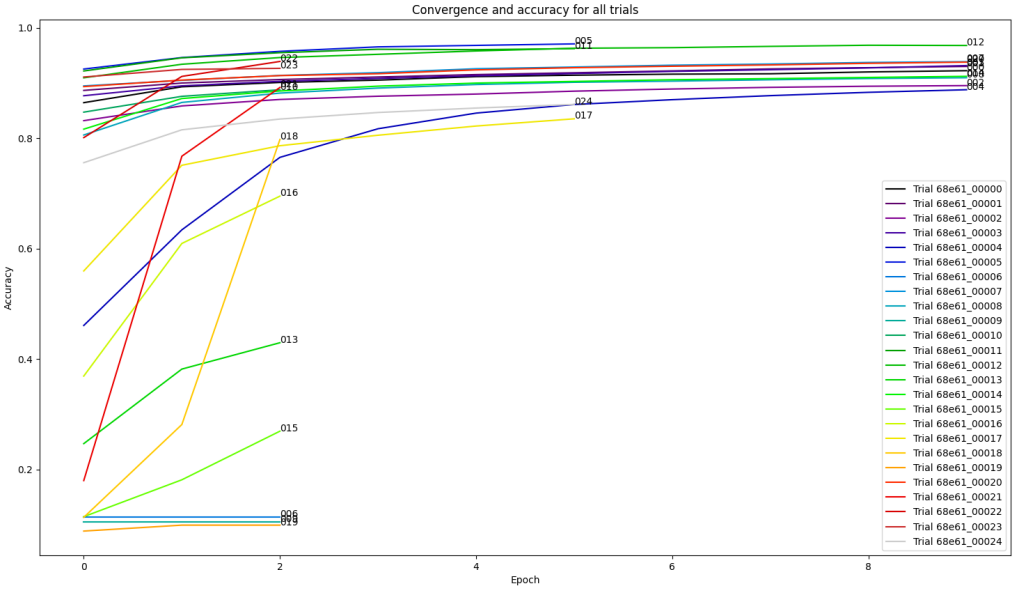

Figure 4.1. Trial results for best hyperparameter search

The analysis of multiple neural network training trials on the MNIST dataset highlights significant influences of hyperparameters on model performance. Notably, higher numbers of neurons (128-256) in a single hidden layer configuration generally yield better results, suggesting a sweet spot between model complexity and learning effectiveness. Batch sizes of 1, although computationally demanding, sometimes achieve the highest accuracies, indicating a trade-off between performance and training stability. Moreover, the choice of learning rate is crucial, with lower rates sometimes leading to underfitting and higher rates causing instability. The optimal setup for this dataset likely involves a moderate learning rate and a balanced approach to batch size and loss function to ensure both computational efficiency and robust model performance.

4.3. Convergence

From the Figure 4.1, we can see that trials 19, 6, 9, 15, and 13 have the worst convergence; some would not converge. All those trials have a low Learning Rate (lower than 0,0004), whereas most trials that achieve the best results and converge faster have a Learning Rate 5 times larger. Additionally, those trials suffer from problems caused by many hidden layers or high batch size. The used criteria do not seem to greatly influence the convergence or accuracy.

4.4. Loss value across steps

We made a graph visualizing the loss value for every step in the best trial we found and for every epoch value in this trial. From the graph Figure 4.2, we can see that loss varies across steps, but each consecutive epoch has a smaller loss value and is also more stable. It can be best seen in the last epoch (grey colour). Also, most often, the highest loss values are from the first epochs: epoch 0 (black) and epoch 1 (purple).

Table 4.2. Experiment Results

Trial name	Neurons	Layers	LR	Batch	Criterion	Iter	Time (s)	Loss	Acc
68e61_00000	64	1	0.00488	16	MarginLoss	10	260.54	0.0343	0.922
68e61_00001	256	0	0.00846	32	MarginLoss	10	236.67	0.0327	0.93
68e61_00002	256	0	0.00061	16	MarginLoss	10	272.88	0.0588	0.895
68e61_00003	64	0	0.00463	32	CrossEntropyLoss	10	225.11	0.2406	0.9316
68e61_00004	64	1	0.00084	32	CrossEntropyLoss	10	208.21	0.3968	0.8878
68e61_00005	128	0	0.00111	1	NLLLoss	6	753.46	0.0965	**0.9708**
68e61_00006	64	5	0.00019	1	MarginLoss	3	393.02	0.8877	0.1131
68e61_00007	64	0	0.00059	1	MarginLoss	10	650.79	0.0249	0.9386
68e61_00008	64	1	0.00248	16	MarginLoss	10	176.20	0.0421	0.9095
68e61_00009	256	5	0.00018	32	MarginLoss	3	57.48	0.9005	0.1052
68e61_00010	256	0	0.00186	32	CrossEntropyLoss	3	49.74	0.4485	0.8878
68e61_00011	64	0	0.00202	1	NLLLoss	6	569.23	0.0935	**0.9623**
68e61_00012	256	1	0.00224	1	MarginLoss	10	975.35	0.0073	**0.9679**
68e61_00013	64	1	0.00011	16	NLLLoss	3	52.52	2.1550	0.4294
68e61_00014	64	1	0.00571	32	MarginLoss	10	155.67	0.0405	0.9121
68e61_00015	128	1	0.00011	32	NLLLoss	3	52.15	2.2300	0.2692
68e61_00016	256	1	0.00043	32	NLLLoss	3	48.38	1.7835	0.6950
68e61_00017	64	1	0.00104	32	MarginLoss	6	89.55	0.1000	0.8351
68e61_00018	64	5	0.00036	1	NLLLoss	3	311.26	1.2904	0.7977
68e61_00019	64	5	0.00038	32	NLLLoss	3	42.63	2.3049	0.0989
68e61_00020	64	0	0.00055	1	MarginLoss	10	532.23	0.0259	0.9378
68e61_00021	256	5	0.00038	1	CrossEntropyLoss	3	379.85	0.5045	0.8919
68e61_00022	64	5	0.00139	1	CrossEntropyLoss	3	269.53	0.2403	0.9390
68e61_00023	64	0	0.00906	1	NLLLoss	3	225.76	0.2348	0.9265
68e61_00024	128	0	0.00032	16	MarginLoss	6	90.87	0.0937	0.8612

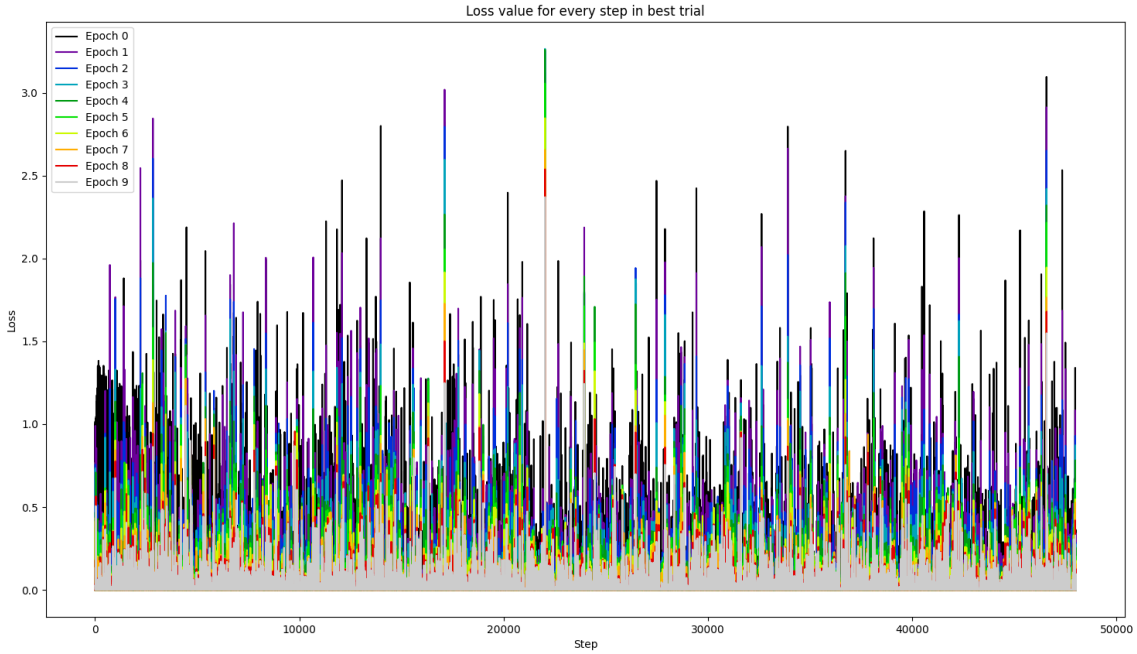

Figure 4.2. Loss value for every step

5. Summary

From this laboratory, we learned that using Neural Networks for image recognition is a very good solution. Moreover, the neural network is able to correctly recognise numbers that might be misinterpreted by a human. We also learned that the correct parameters and design of the network are crucial for obtaining good results.