CUDA - Laboratory report No. 1

Bartłomiej Obrochta

March 2024

1 Introduction

In recent laboratory sessions and lectures, our exploration of machine learning fundamentals has been extensive. We've delved into various crucial topics, including the landscape of Artificial Intelligence, the significance of optimal weight distribution within the model, and the pivotal role of selecting an appropriate loss function to facilitate effective problem-solving by the network. The tasks tackled by the machine learning models, can be broadly categorized into two main types:

- Classification problems: These involve categorizing data points into predefined classes or categories. For example, determining whether an email is spam or distinguishing between images of cats, dogs and horses.
- Regression problems: These aim to predict values for new data based on the input data model learned from. For instance, forecasting weather conditions based on factors such as temperature, humidity, wind speed and direction.

This report focuses on experimenting with shallow network. The objective is to assess how each component of the model structure and the training methodology impacts the accuracy of its predictions.

2 Theory

Before engaging into the experimentation with the model, it is crucial to further explore the theoretical concepts. This exploration not only will enhance our understanding, but will also enable us to draw insightful conclusions.

2.1 Landscape of Artificial Intelligence

The field of Artificial Intelligence (AI) encompasses a vast array of concepts and applications, all centered around the idea of machines performing tasks without explicitly coded rules. Let's begin by examining this enormous domain.

- **Artificial Intelligence** - AI refers to systems capable of processing information and utilizing it to make decisions or achieve desired outcomes. Present-day applications typically exhibit Narrow Intelligence, meaning they specialize in specific task, or closely related task groups.
- **Machine Learning** - Machine learning involves the use of software to interpret data without the need for explicitly coded rules. It encompasses several subfields, including:

  - **Unsupervised Learning**: Algorithms analyze and interpret data without labeled responses.
  - **Supervised Learning**: Algorithms learn from labeled input-output pairs, aiming to generalize from the training data to predict outcomes for unseen data.
  - **Reinforcement Learning**: Agents learn to interact with an environment by taking actions and receiving feedback or rewards, aiming to maximize cumulative rewards over time.

2.2 Weights and Weight Distribution

In machine learning, neural networks consist of interconnected neurons organized into layers. A critical component is the concept of weights, representing the strength of connections between neurons. During training, these weights adjust to minimize differences between predicted and actual outputs. Proper initialization and distribution of weights are crucial for effective network functioning. An optimal weight distribution ensures each neuron receives appropriate input signals and contributes meaningfully to computations. Essentially, weights encapsulate the knowledge acquired by the model during the learning process. Adjusting weights facilitates learning, while unchanged weights preserve knowledge.
Choosing appropriate initial weight values is a strategic decision that significantly impacts model performance. Large initial values may imply prior knowledge within the model, but can pose challenges in unlearning biased information. Conversely, setting all weights to zero is not advisable, as it can detrimentally affect model performance.

2.3 Activation functions

Activation functions are essential components of neural networks, as they introduce non-linearity into the model's output, allowing it to learn complex patterns and relationships in the data. They are applied to the weighted sum of neuron's inputs, determining whether it should be activated or not. The choice of activation function can significantly impact the model's performance and training dynamics. Proper selection of activation functions can lead to faster convergence, better generalization, and improved model accuracy. However, inappropriate choices may result in training instabilities, vanishing or exploding gradients, or ineffective learning. The vanishing gradient problem occurs when gradients become extremely small during back propagation in deep neural networks, leading to slow or stagnant learning. Conversely, the exploding gradient problem arises when gradients grow exponentially, causing instability and numerical overflow issues during training. Let's cover some of the activation functions:

- **Sigmoid**
  - Range 0 to 1
  - Suitable for: Binary classification tasks
  - Reason to use: Sigmoid is useful when outputs need to be interpreted as probabilities or when dealing with binary classification problems. However, it may suffer from the vanishing gradient problem in deep networks due to saturation towards the extreme values of 0 and 1.
- **Hyperbolic Tangent (tanh)**
  - Range: -1 to 1

  - Suitable for: Centering the data around zero
  - Reason to use: Tanh function helps with gradient flow, making it suitable for deep networks. By centering the data around zero, tanh mitigates the vanishing gradient problem to some extent. However, it may still encounter saturation towards the extreme values of -1 and 1.

- **Rectified Linear Unit (ReLU)**
  - Range: 0 to infinity
  - Formula: $max(0, x)$
  - Suitable for: Most applications, especially deep neural networks
  - Reason to use: ReLU is the most widely used activation function, known for its simplicity and effectiveness. It replaces all negative input values with zero, providing a linear response for positive inputs. This characteristic helps alleviate the vanishing gradient problem by avoiding saturation for positive values.

- **Softmax**
  - Range: 0 to 1 (normalized probabilities)
  - Suitable for: Multi-class classification tasks.
  - Reason to use: Softmax function is commonly used in the output layer of neural networks for multi-class classification tasks. It transforms the raw outputs into probabilities, ensuring that the sum of the probabilities across all classes is equal to one.

2.4 Loss functions

Loss functions, also known as cost functions, are essential components in the training of machine learning models. Their primary purpose is to quantify the discrepancy between predicted and actual values, guiding the optimization process towards better model performance. The selection of an appropriate loss function hinges upon the specific characteristics of the problem under consideration. Fundamentally, loss functions aim to penalize the model for incorrect predictions while potentially rewarding desirable outcomes, depending on the specific function used. These functions are indispensable in the learning process, as they drive the adjustment of model weights to minimize prediction errors and improve overall accuracy. Let's cover some of the loss functions:

- **Mean Squared Error (MSE):**
  - Suitable for: Regression tasks
  - Measures: the average squared difference between predicted and actual values
  - Description: MSE is computed by taking the mean of the squared differences between each predicted value and its corresponding actual value. It provides a measure of the average discrepancy between predictions and ground truth. However, MSE is sensitive to outliers due to the squaring operation, which can disproportionately affect the loss.

- **Mean Absolute Error (MAE):**
  - Suitable for: Regression tasks
  - Measures: the average absolute difference between predicted and actual values

  - Description: MAE calculates the mean of the absolute differences between predicted and actual values, offering a more robust alternative to MSE in the presence of outliers. Unlike MSE, MAE does not square the differences, resulting in a more linear relationship between errors and the loss.

- **Cross-Entropy Loss (or Log Loss):**
  - Suitable for: Multi-class classification tasks
  - Measures: the dissimilarity between predicted probabilities and true labels
  - Description: Cross-Entropy Loss quantifies the difference between the predicted probability distribution and the true distribution of class labels. It penalizes models more heavily for incorrect predictions by assigning higher loss values to instances where the predicted probabilities diverge from the true labels. This encourages better calibration of predicted probabilities and is commonly used in multi-class classification tasks.

- **Categorical Cross-Entropy Loss:**
  - Suitable for: Multi-class classification tasks
  - Description: Categorical Cross-Entropy Loss is a variant of cross-entropy loss specifically designed for multi-class classification problems. It calculates the cross-entropy loss for each class separately and then sums them up, providing a more accurate measure of model performance in multi-class scenarios. This variant ensures that the loss is appropriately scaled across all classes, leading to more reliable optimization and evaluation in multi-class classification tasks.

3 Laboratory Task

During our laboratory sessions, we worked on a simple shallow model aimed at accurately classifying handwritten digits. The dataset comprised 60,000 labeled images of handwritten digits, with 50,000 images allocated for training and 10,000 for evaluating the model's performance. To prepare the data for our model, we flattened the 28 by 28 pixel images into vectors of size 784. To streamline the search for the most accurate model, we will train all models for 45 epochs, using batches of size 128.

3.1 Premade Model:

Initial model architecture :

- Layer 1: consisting of **64 neurons**, with Sigmoid activation function
- Layer 2: consisting of **10 neurons**, with SoftMax activation function
- The Loss Function: Mean Squared Error

The performance of this model is far from acceptable, with highest achieved accuracy being $0,5145$. This unsatisfactory performance can be readily explained, as we are using Mean Squared Error (MSE) as a loss function, which is completely incorrect for our task. MSE is typically used for regression problems, whereas we are dealing with a classification problem. Using MSE for a classification task leads to suboptimal results because it doesn't appropriately penalize classification errors.

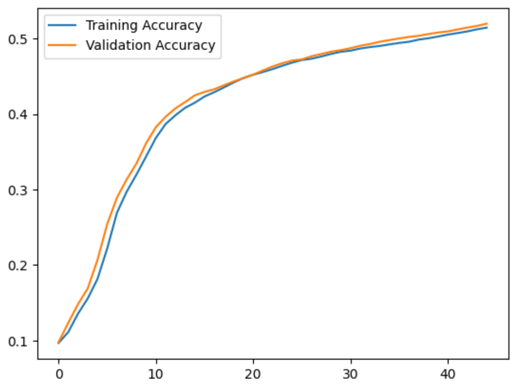

Figure 1: Model's accuracy plot - MSE loss

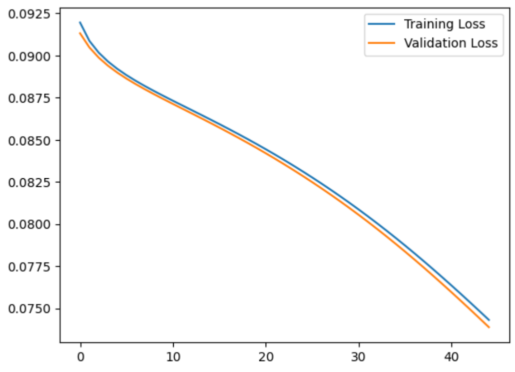

Figure 2: Model's loss plot - MSE loss

The analysis of the provided graphs reveals unpromising trajectory of model improvement, characterized by a gradual increase in accuracy and a corresponding decrease in loss with each epoch. However, it's evident that the current rate of learning might not be optimal, as the observed improvements are relatively slow, after 20th epoch.
With the accuracy reaching $0,7151$ after an additional 45 epochs, it's clear that the learning rate of the model is stagnating more and more.

3.2 Loss function change

The first improvement involves updating the loss function to better suit our task. Mean Squared Error, as mentioned in point 2.4, is suitable for regression tasks, but we are facing a classification problem. Therefore, let's change the loss function to Categorical Cross-Entropy.
Model architecture :

- Layer 1: consisting of **64 neurons**, with Sigmoid activation function
- Layer 2: consisting of **10 neurons**, with SoftMax activation function
- The Loss Function: Categorical Cross-Entropy

The adoption of a more suitable loss function has yielded promising outcomes. The model rapidly achieved an accuracy level of 0.9006 after just 25 epochs. However, subsequent progress stagnated, as evidenced by the following graphs, reaching $0,9117$ after 45 epochs.
There is not much room left for improvement regarding the loss functions. I've conducted additional training using Categorical Focal Cross-entropy and Mean Absolute Error (which is another loss function, that better suits regression problems), achieving accuracies of 0.8786 and 0.3803, respectively. It's evident that Categorical Cross Entropy is the most effective loss function for this task.

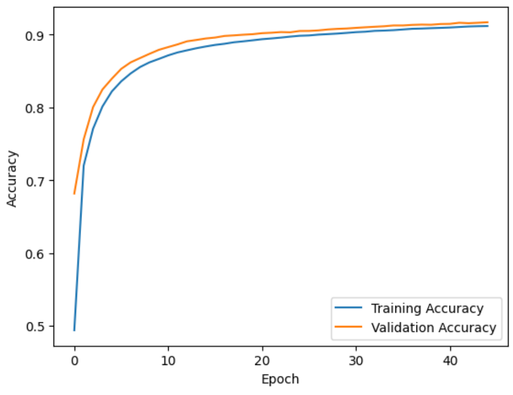

Figure 3: Model's accuracy plot- CCE loss

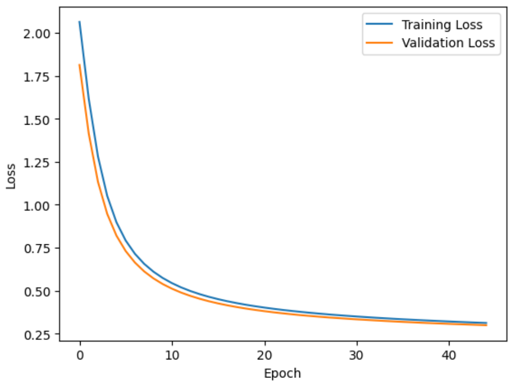

Figure 4: Model's loss plot - CCE loss

3.3 Activation Function change

As the model has reached its performance ceiling, our next focus will be experimentation with alternative activation functions. This exploration aims to identify whether different activation functions can further enhance the model's performance. Let's change the activation function of the first layer to Tanh.

3.3.1 Tanh activation function

Model architecture :

- Layer 1: consisting of **64 neurons**, with Tanh activation function
- Layer 2: consisting of **10 neurons**, with SoftMax activation function
- The Loss Function: Categorical Cross-Entropy

The adjustment of the activation function in the first layer has significantly raised the performance ceiling, resulting in the model achieving an accuracy of 0.9458 and validation accuracy of 0,9437. However, additionally changing the activation function in the second layer to tanh has led to a decrease in accuracy to 0.0893, which is more than ten times worse compared to using softmax in the second layer. Given that we are classifying handwritten digits, it's no surprise that softmax performs the best in the second layer. Therefore, we will halt further experimentation with changing the activation function in the second layer for now.

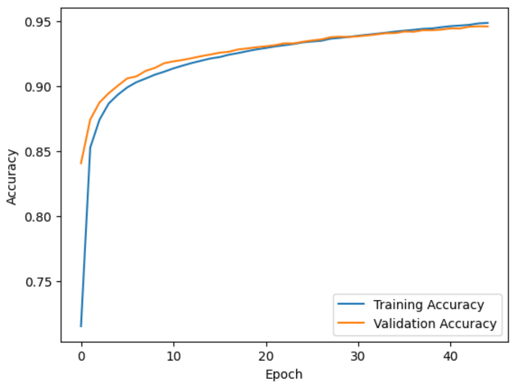

Figure 5: Model's accuracy plot - tanh activation

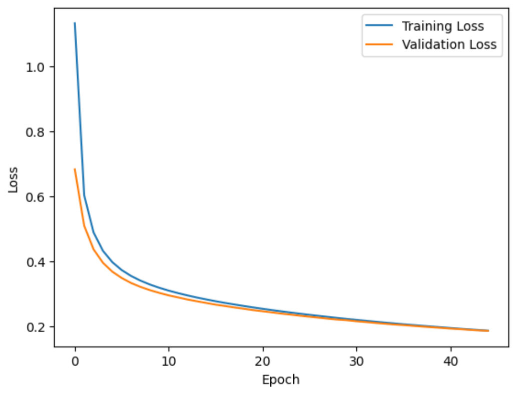

Figure 6: Model's loss plot - tanh activation

3.3.2 ReLU activation function

Model architecture :

- Layer 1: consisting of **64 neurons**, with ReLU activation function
- Layer 2: consisting of **10 neurons**, with SoftMax activation function
- The Loss Function: Categorical Cross-Entropy

The ReLU activation function performs slightly better than tanh, achieving an accuracy of 0.9538 and a validation accuracy of 0.9521. In this particular case, both tanh and ReLU activation functions yield good performance. Similarly, here, changing the second layer's activation function concludes with unacceptable performance of around 0.1513. Thus, we have yet another proof that softmax is the best-performing activation function on the second layer.

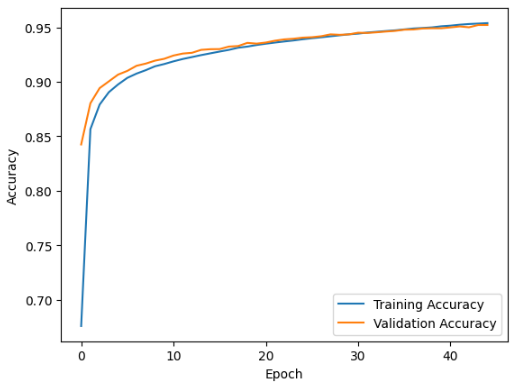

Figure 7: Model's accuracy plot - ReLU activation

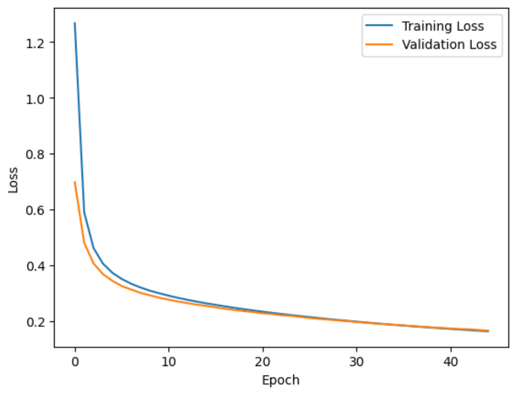

Figure 8: Model's loss plot - ReLU activation

3.4 Weight initialization

As there is little to no room left for improvement regarding activation and loss functions, we'll attempt to enhance performance by initializing weights using the Glorot uniform method.
However, the results are not significantly different, as the accuracy and validation accuracy both oscillate around the 0.9500 mark. It might be due to the fact that we are using a shallow model, and the weight initialization barely affects our system. The shallow nature of the model, with its limited capacity to capture complex relationships in the data, could be a contributing factor to the lack of substantial improvement. It appears that we have reached the limit of improvement with the current model configuration.

3.5 Batch size

In the beginning of the chapter, we decided to streamline the experimentation process by settling for a batch size of 128 and training for 45 epochs. While it's evident from the plots that training for more epochs leads to only slight performance increases due to graph stagnation, let's explore the effect of reducing the batch size.
We'll employ the best-performing model configuration identified thus far:

- Layer 1: consisting of **64 neurons**, with ReLU activation function
- Layer 2: consisting of **10 neurons**, with SoftMax activation function
- The Loss Function: Categorical Cross-Entropy

For the batch size of 64, which is half of the previous batch size, the graphs remain mostly unchanged from the ones above (Figure 7 and Figure 8). However, the model has reached a performance of 0.9700 accuracy and validation accuracy of 0.9659. Let's further reduce the batch size to 32. The graphs of accuracy and validation accuracy start to visibly diverge from one another, and the model achieves an accuracy of 0.9826 and validation accuracy of 0.9742.

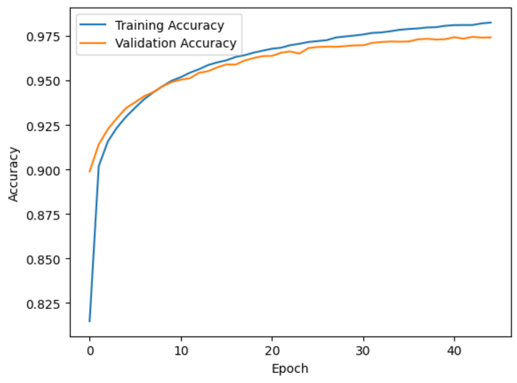

Figure 9: Model's accuracy plot - Batch Size 32

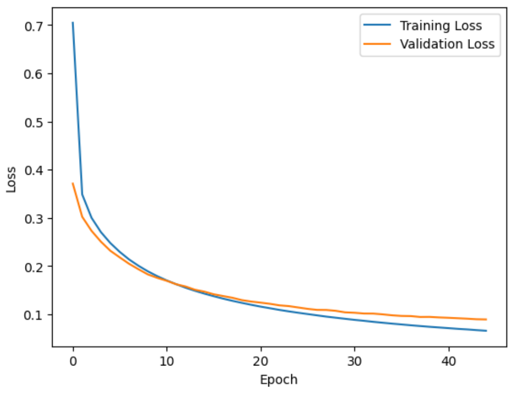

Figure 10: Model's loss plot - Batch Size 32

Further reduction in batch size fails to produce significant improvements in performance. Consequently, we can assert that the model has reached its peak performance.

4 Summmary

This laboratory report explores shallow neural networks in the context of machine learning, focusing on optimizing a model for handwritten digit classification. It delves into fundamental concepts like classification vs. regression problems, weight distribution, activation functions, and loss functions, providing a solid theoretical foundation for empirical investigations.

Initial experiments revealed the significance of aligning model components, such as activation and loss functions, with the problem's nature. Adjustments, like transitioning from Mean Squared Error (MSE) to Categorical Cross-Entropy as the loss function and exploring different activation functions, notably improved the model's performance.

The iterative optimization process highlighted the intricate balance needed for optimal results, showcasing the impact of each component on accuracy. Structured modifications, including changes in activation functions, weight initialization, and batch size, significantly enhanced performance.
In conclusion, this exploration underscores the multi-dimensional nature of model optimization in machine learning. While the journey from a basic to a high-accuracy model showcases the iterative learning process, it's essential to recognize that in more complex models, careful planning before training is crucial. The insights gained contribute to our understanding of neural networks and offer a blueprint for future research, emphasizing the synergy between theory and practice.