Assignment 3

Multi-Layer Artificial Neural Network Implementation and Comparison

Noa Magrisso – 206934978
Shaked Tayouri – 323866749

1. Overview

This assignment focuses on implementing and comparing artificial neural networks (ANNs) with one and two hidden layers for classifying handwritten digits from the MNIST dataset. The project extends the implementation from Chapter 11 of *Machine Learning with PyTorch and Scikit-Learn* by Raschka et al. (2022) and evaluates model performance in comparison to a fully connected ANN implemented in Keras/TensorFlow.

2. Model Implementation and Training Pipeline

2.1 Model Architectures

2.1.1 Original Single-Hidden-Layer ANN

The original implementation follows the Chapter 11 framework, utilizing a single hidden layer. It is trained using gradient descent and evaluates its performance using accuracy metrics.

2.1.2 Two-Hidden-Layer ANN

The extended version includes two fully connected hidden layers, each with adjustable sizes, to model complex patterns in the data. This required updating the forward and backward propagation logic to incorporate the additional layer and modifying the training process to ensure smooth and efficient learning.

2.1.3 Fully Connected Keras ANN

The fully connected artificial neural network (ANN) was implemented using Keras, utilizing a two-hidden-layer architecture. The model leverages Keras' optimized framework for efficient training.

2.2 Initial Training Process: Forward and Backward Propagation *– Notebook 1*

2.2.1 Custom Models (Initial Version: Sigmoid + MSE)

The initial implementation followed a fully connected artificial neural network (ANN) with either one or two hidden layers, both using sigmoid activation and trained with mean squared error (MSE) loss.

Forward Propagation:

- The architecture initially included one hidden layer, and we extended it to two hidden layers, both using sigmoid activation to model complex patterns in the data – consistent with the original notebook.
- The output activation function remained sigmoid, computing final predictions.
- The network computed activations sequentially through one or two hidden layers (correspondingly) before reaching the output layer.

Backward Propagation:

- Weight and bias updates were calculated using the chain rule to ensure proper gradient propagation.
- The loss function was MSE, guiding weight adjustments through gradient descent.
- For the single-hidden-layer model, gradients were computed for one hidden layer.
- For the two-hidden-layer model, an additional set of gradients and weight updates were required to accommodate the second hidden layer.

2.2.2 Keras Model (Initial Version: Sigmoid + MSE)

Unlike the custom models, Keras abstracts forward and backward propagation, handling these steps internally.

- Forward Propagation: Keras automatically performs matrix multiplications and applies activations, following the defined architecture (initially used sigmoid activation in all layers).
- Backward Propagation: Uses automatic differentiation to compute gradients for weight updates.
- Weight Updates: The model was trained using the Stochastic Gradient Descent (SGD) optimizer with a learning rate of 0.1, following the same approach as the custom models with MSE loss.

  In the second step, the loss function was changed from MSE to Cross-Entropy to align with best practices for multi-class classification. Additionally, Softmax replaced Sigmoid in the output layer to produce class probabilities.

2.3 Improved Training Process: Forward and Backward Propagation *– Notebook 2*

To improve convergence speed, stability, and classification accuracy, all three models were updated to use Cross-Entropy loss, Softmax in the output layer, and ReLU in hidden layers instead of MSE and Sigmoid.

2.3.1 Custom Models (Improved Version: ReLU + Cross-Entropy + Softmax)

Forward Propagation Improvements:

- **Single-hidden-layer model**:
  - Sigmoid activation in the hidden layer was replaced with ReLU to prevent vanishing gradients and accelerate learning.
  - Softmax replaced sigmoid in the output layer to produce a probability distribution across classes.
- **Two-hidden-layer model**:
  - Both hidden layers switched from Sigmoid to ReLU to improve convergence and gradient flow.
  - Softmax replaced Sigmoid in the output layer for better classification performance.

Backward Propagation Adjustments:

- Loss function changed from MSE to Cross-Entropy, which is more suitable for multi-class classification.
- Weight updates were modified to accommodate the use of ReLU activations in hidden layers.
- Gradient updates adjusted to ensure smooth learning with Cross-Entropy loss.

2.3.2 Keras Model (Improved Version: ReLU + Cross-Entropy + Softmax)

After evaluating performance, the Keras model was updated to match the improvements introduced in the custom models:

- Hidden Layers: Sigmoid was replaced with ReLU activation.
- Output Layer: Softmax replaced Sigmoid, producing a probability distribution over class labels.
- Loss Function: Cross-Entropy replaced MSE, aligning with best practices for multi-class classification.
- Weight Updates: The model was trained using Stochastic Gradient Descent (SGD) with a learning rate of 0.1, ensuring consistency with the custom models.

3. Experimental Setup

To maintain consistency with the original notebook, we unified all available data. However, for this implementation, we followed the required partitioning of 70% training and 30% testing, as specified in the assignment instructions. Within the training set, 10% was allocated for validation, ensuring a standardized evaluation process across models.

Additionally, we conducted experiments with various hyperparameters, including validation set size, learning rate, number of epochs, etc.

However, these variations did not yield significant performance differences. Therefore, we present the following configuration as the representative setup for our analysis.

- **Dataset:** MNIST (28×28 grayscale images of handwritten digits, 10 classes: digits 0-9).
- **Data Split:** 70% Training, 30% Testing. (Validation – 10% from training).
- **Batch Size:** 100.
- **Epochs:** 50.
- **Learning Rate:** 0.1.

4. Evaluation Metrics

- **Accuracy:** Measures the percentage of correct predictions.
- **Macro AUC:** Evaluates the model's ability to distinguish between classes, considering all classes equally.
- **Loss Function:**
  - **Initial Version (Notebook 1):** Mean Squared Error (MSE) was used to measure the difference between predicted probabilities and true labels.
  - **Improved Version (Notebook 2):** Cross-Entropy replaced MSE, providing a more appropriate loss function for multi-class classification.
- **Comparison of Activation Functions:**
  - The impact of **Sigmoid vs. ReLU** in hidden layers was analysed.
  - **Softmax** was introduced in the output layer for probability distribution in multi-class classification.

5. Results & Comparisons

This section compares the performance of different architectures under two training configurations:

1. Initial Version – using Mean Squared Error (MSE) loss with Sigmoid activation.
2. Improved Version – using Cross-Entropy loss with ReLU activation.

We analysed the impact of activation functions and loss functions on accuracy, AUC, and overall performance. The results demonstrate how these design choices influence the network's ability to learn and classify data effectively.

Initial Version

Seed 42, Validation – 10% from training set, Loss Function – **MSE**, Activation Function – **Sigmoid** (without SoftMax):

Model	***Accuracy***	***Macro AUC***	***MSE***
One-Hidden-Layer ANN	*94.43%*	*99.18%*	0.0097
Two-Hidden-Layer ANN	*94.55%*	*99.33%*	0.0091
Fully Connected Keras ANN	*84.46%*	*96.6%*	0.035
Fully Connected Keras ANN	**With SoftMax** *96.79%*	*99.89%*	**Cross-Entropy** 0.1112

Improved Version

Seed 42, Validation – 10% from training set, Loss Function – **Cross Entropy**, Activation Function – **ReLU** (with SoftMax):

Model	***Accuracy***	***Macro AUC***	***Cross-Entropy***
One-Hidden-Layer ANN	*96.09%*	*99.85%*	0.1489
Two-Hidden-Layer ANN	*96.2%*	*99.88%*	0.1565
Fully Connected Keras ANN	*96.85%*	*99.91%*	0.1431

Observations:

According to our initial version –

- **The two-hidden-layer ANN slightly outperformed the one-hidden-layer ANN**, achieving marginally higher **accuracy (94.55% vs. 94.43%)**, better **Macro AUC (99.33% vs. 99.18%)**, and a **lower MSE (0.0091 vs. 0.0097)**. This suggests that adding an extra hidden layer provides better feature extraction and learning capacity, resulting in improved classification performance**.**

- **The fully connected Keras ANN (without SoftMax) performed significantly worse** compared to the custom models, with **only 84.46% accuracy** and a **higher MSE (0.035 vs. 0.009X)**. This aligns with our expectations, as **MSE is generally not the optimal loss function for multi-class classification problems**.

- **Switching the Keras model to use SoftMax in the output layer and Cross-Entropy loss significantly improved its performance**, boosting **accuracy to 96.79% and Macro AUC to 99.89%**. This further confirms that **MSE loss is suboptimal for classification tasks and that using SoftMax with Cross-Entropy is more appropriate**.

According to the improved version –

- **Switching from Sigmoid to ReLU in hidden layers improved model performance across all architectures**, likely due to ReLU's ability to mitigate the vanishing gradient problem and accelerate convergence.

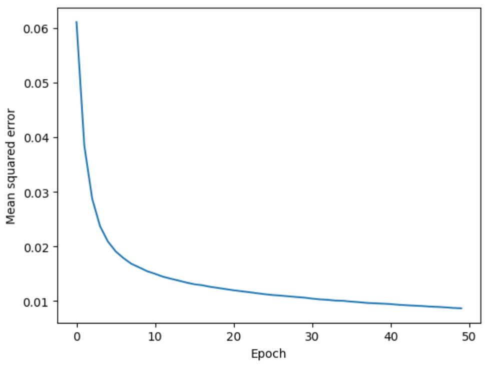
- **The two-hidden-layer ANN outperformed the one-hidden-layer ANN** (**96.2% vs. 96.09% accuracy, 99.88% vs. 99.85% Macro AUC**), reinforcing the benefit of deeper architectures for feature learning. However, the difference is relatively small, suggesting that additional depth may provide diminishing returns beyond a certain point.

- **The fully connected Keras ANN achieved the highest accuracy (96.85%) and best Macro AUC (99.91%)**, showing that leveraging Keras' optimized framework, combined with appropriate activation functions and loss functions, led to **better generalization and convergence**.

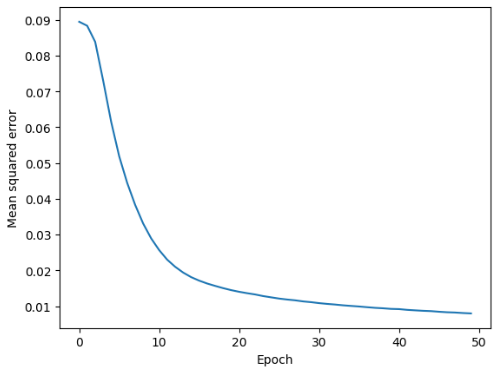
- **Cross-Entropy loss values decreased compared to MSE**, further confirming that Cross-Entropy is **better suited for multi-class classification**.

1st Notebook – Initial Training Process Results

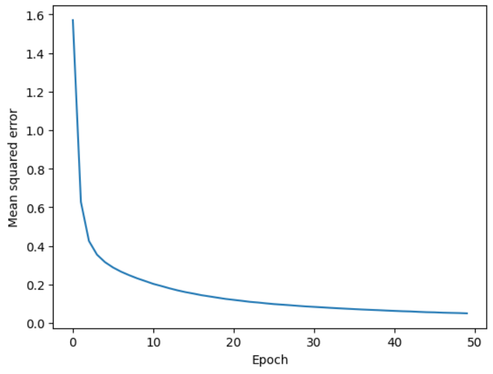
Training loss (MSE) per epoch:



(a) ANN with a single hidden layer

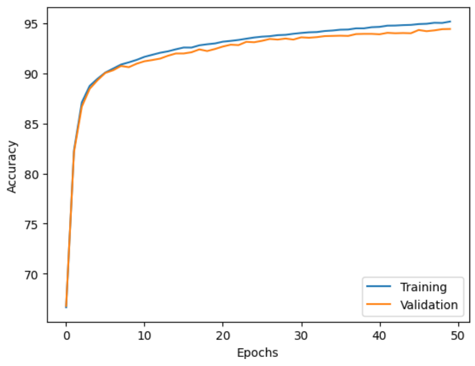


(b) ANN with two hidden layers

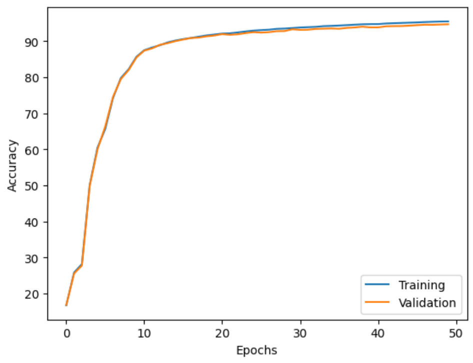


(c) TensorFlow/Keras fully connected ANN with two hidden layers - CE

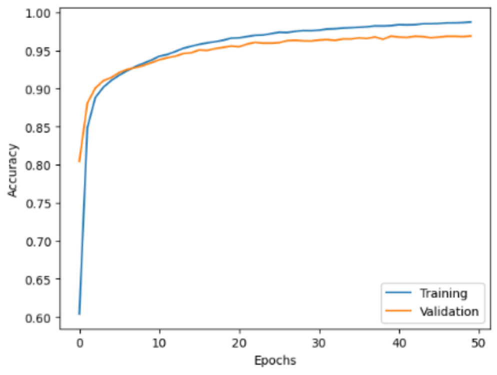
Accuracy of training and validation sets per epoch:



(d) ANN with a single hidden layer

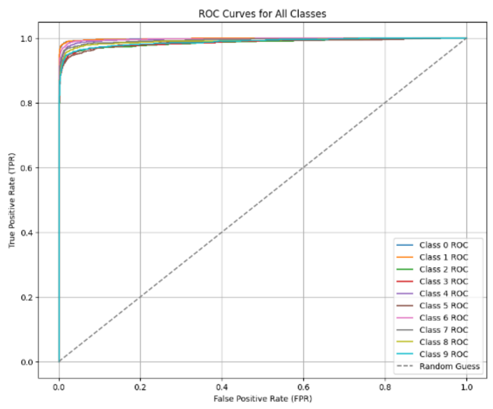


(e) ANN with two hidden layers

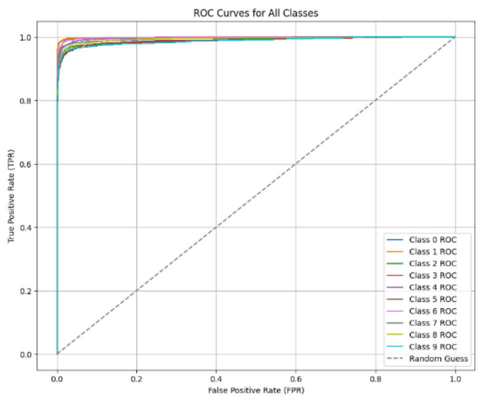


(f) TensorFlow/Keras fully connected ANN with two hidden layers - CE

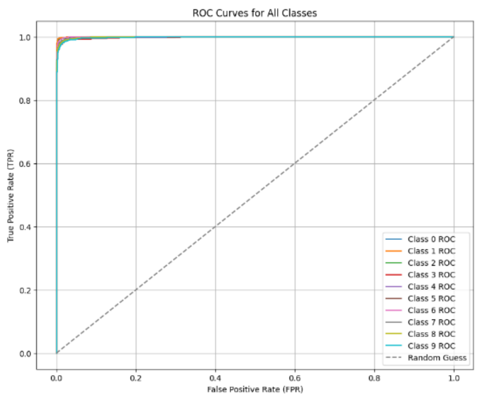
AUC of training and validation sets per epoch:



(g) ANN with a single hidden layer



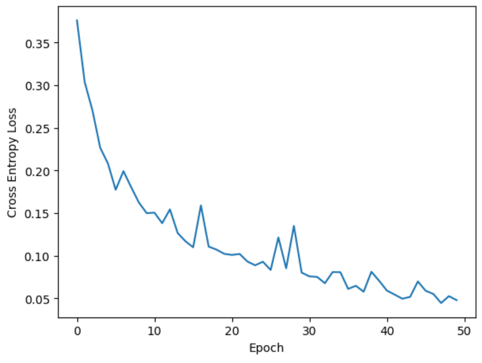
(h) ANN with two hidden layers



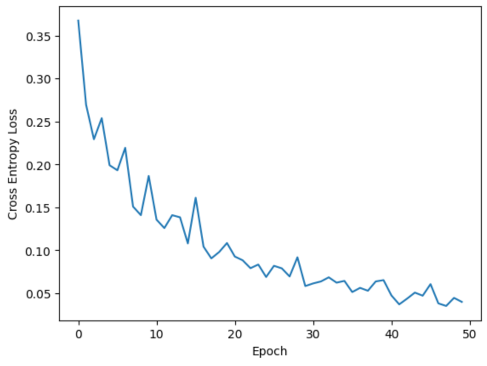
(i) TensorFlow/Keras fully connected ANN with two hidden layers - CE

2nd Notebook – Improved Training Process Results – Cross-Entropy; ReLU

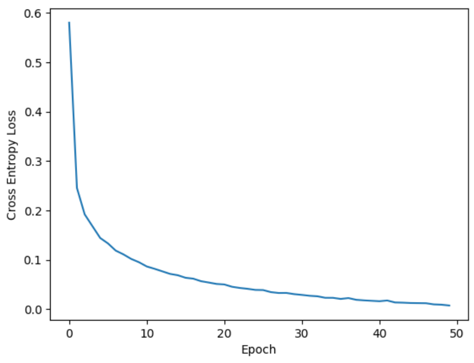
<u>Training loss (Cross-Entropy) per epoch:</u>



(a) ANN with a single hidden layer

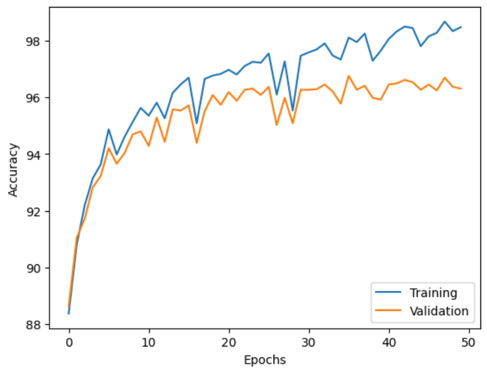


(b) ANN with two hidden layers

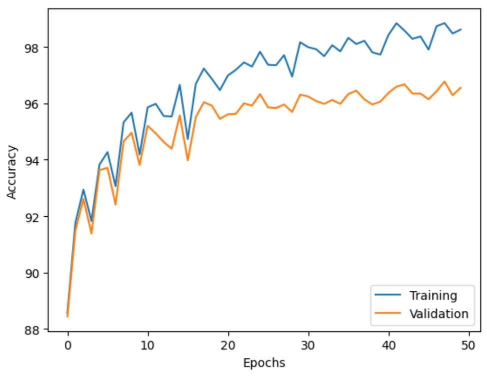


(c) TensorFlow/Keras fully connected ANN with two hidden layers

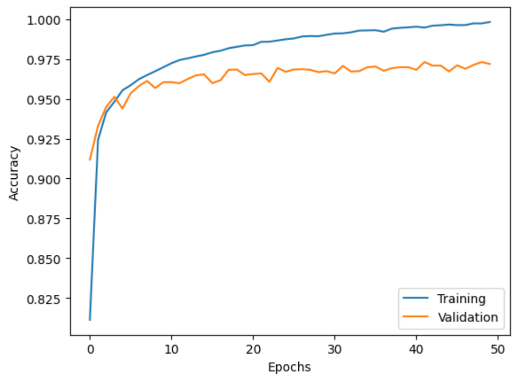
<u>Accuracy of training and validation sets per epoch:</u>



(d) ANN with a single hidden layer

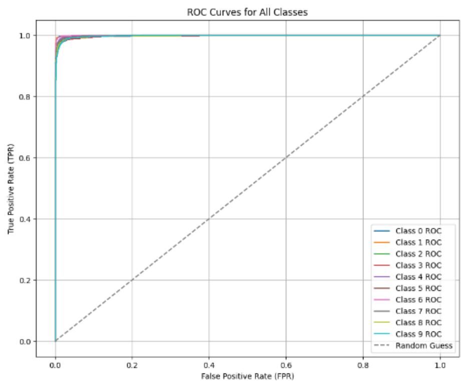


(e) ANN with two hidden layers

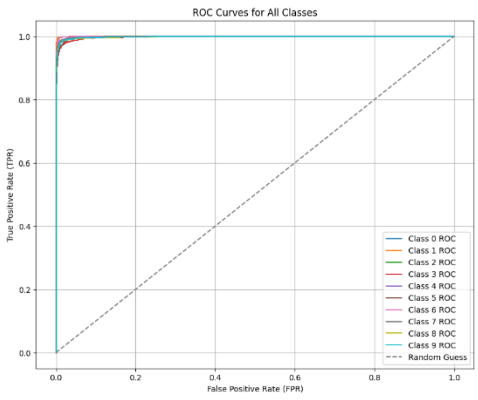


(f) TensorFlow/Keras fully connected ANN with two hidden layers

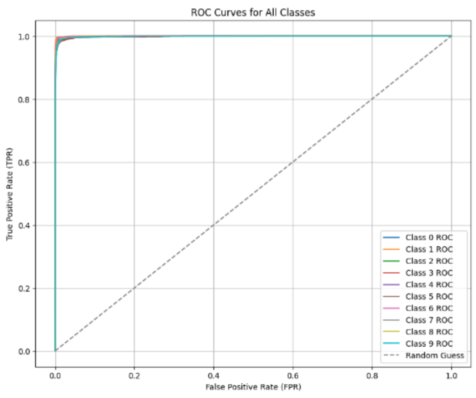
<u>AUC of training and validation sets per epoch:</u>



(g) ANN with a single hidden layer



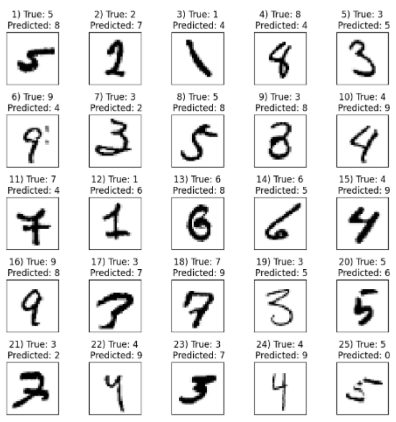
(h) ANN with two hidden layers



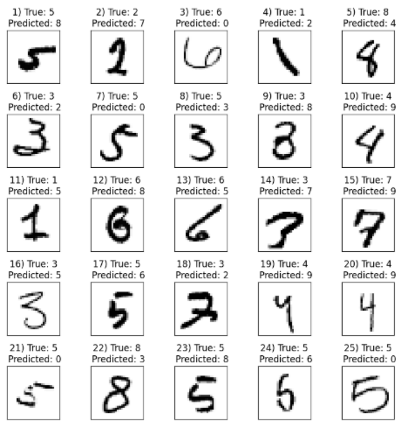
(i) TensorFlow/Keras fully connected ANN with two hidden layers

Plot failure cases:

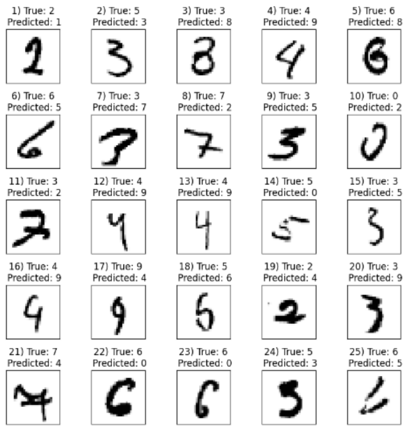
1st Notebook – Initial Training Process Results



(a) ANN with a single hidden layer

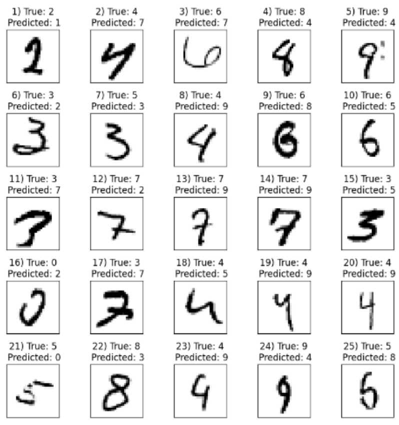


(b) ANN with two hidden layers

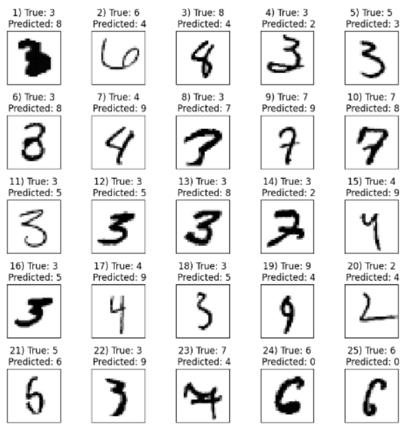


(c) TensorFlow/Keras fully connected ANN with two hidden layers

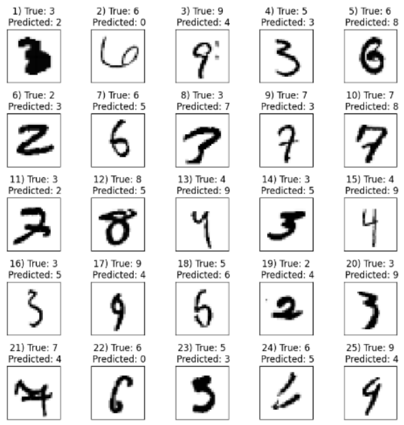
2nd Notebook – Improved Training Process Results – Cross-Entropy; ReLU



(a) ANN with a single hidden layer



(b) ANN with two hidden layers



(c) TensorFlow/Keras fully connected ANN with two hidden layers

These visualizations compare failure cases from the **initial training (MSE + Sigmoid)** and **improved training (Cross-Entropy + ReLU)**. The **single-hidden-layer model (a)** had trouble distinguishing similar-looking digits like **4 and 9** or **3 and 5**, showing its limitations in feature extraction. The **two-hidden-layer model (b)** did slightly better but still struggled with tricky cases like **7 and 2**, proving that adding more layers helps but isn't a perfect solution. The **Keras ANN (c)** initially misclassified digits with overlapping strokes, like **5 and 3**, but improved a lot after switching to **Cross-Entropy and ReLU**.

Even with these improvements, all models still find it hard to classify messy or unclear digits. To make them even better, we could try **data augmentation** or use **more advanced architectures like CNNs**, which are designed to handle handwritten digits more effectively.

6. Conclusion

1. **The two-hidden-layer ANN consistently outperformed the one-hidden-layer model**, demonstrating that deeper networks provide better feature extraction and classification performance for MNIST.

2. The transition **from Mean Squared Error (MSE) to Cross-Entropy loss**, along with the switch **from Sigmoid to ReLU** in hidden layers, improved accuracy and convergence speed.

    a) Sigmoid activation in deeper networks tends to suffer from vanishing gradients, slowing down learning and leading to suboptimal weight updates.

    b) By using ReLU in hidden layers, we observed faster convergence, more efficient gradient propagation, and improved weight updates, allowing the network to better learn complex representations.

    c) Switching from MSE to Cross-Entropy optimized the learning process for classification tasks, ensuring better class separation and aligning training with **SoftMax in the output layer**.

3. **The Keras ANN served as an additional benchmark**, confirming that leveraging pre-optimized deep learning libraries can yield competitive results with minimal manual implementation effort.

4. **Future improvements** could include further hyperparameter tuning, adding dropout regularization, exploring convolutional neural networks (CNNs), or experimenting with more advanced optimizers like Adam or RMSprop.

7. References

1. **Machine Learning with PyTorch and Scikit-Learn**:
    - Raschka, S., Liu, Y., & Mirjalili, V. (2022). "Implementing a Multi-layer Artificial Neural Network from Scratch."
    - Chapter 11 on GitHub
2. **MNIST Dataset**:
    - Yann LeCun and Corinna Cortes. "The MNIST Database of Handwritten Digits."
    - Website Link
3. **Keras Documentation**:
    - Keras: "Building Fully Connected Neural Networks."
    - Keras Documentation

The full implementation can be found in the GitHub repository.