Recognizer: the standalone application to recognize handwritten digits

Kyrylo Krysko
University of Michigan
coder@umich.edu

Thomas Steiner
University of Michigan
tsteiner@umich.edu

Abstract

The Recognizer project focuses on the development of a neural network model to recognize handwritten digits trained on labeled data set. This is a classic problem in the field of machine learning and serves as a great introduction to the concepts of neural networks.

The goal of the project is to familiarize myself with Neural Network concepts by building a standalone Python application. The Recognizer provides UI for managing and displaying source data, training models on different data sets with different parameters and recognizing digits from training data set as well as from custom images. The labeled dataset for training model available on kaggle.com.

Additional functionality includes ability to extend the original dataset with rotated images.

Keywords: Machine Learning, Artificial Intelligence, Neural Network, Python, Forward Propagation, Backward Propagation, Loss Function, Gradient Descent, Activation Function.

1. INTRODUCTION

The rapid advancement in artificial intelligence and machine learning technologies has opened up new frontiers in image recognition. This project, the Recognizer, designed as a practical exploration into the field of machine learning through the development of a neural network capable of recognizing handwritten digits.

The Recognizer serves not only as an educational tool to deepen understanding of neural networks but also as a standalone application capable of practical deployment. Users can interact with the application to upload images of handwritten digits and receive predicted numerals, making it applicable in scenarios requiring digit recognition from forms, educational tools, or any interactive learning environment.

The Recognizer project seeks to bridge the gap between theoretical knowledge and practical application, providing a robust platform for demonstrating the capabilities and challenges of implementing neural networks in real-world applications.

2. CONCEPT

2.1 Project Proposal

The application will allow users to upload images with hand written digits and it will recognize and output the numbers.

We can apply transformations (rotate and scale) to the source dataset to be able to recognize stretched and rotated numbers.

Application components:

User Interface: with functionality to train model, select images for recognition, execute recognition, output results

Functionality: using data set train model with specified parameters, save model on disk, recognize numbers in provided images using saved model.

The application can be implemented with Python3.10, using libraries such as NumPy for numerical computation, pandas for data manipulation, and PySide2 for the graphical user interface

2.2 Implementation Plan

I planned to implement application in two stages. First, step is to implement digits recognition using source dataset. Create application with UI, train model and test it.

Once we will be able to train model and recognize digit with source data set, we can move to the next stage, to create extended dataset, train second model and check how it is able to recognize rotated images.

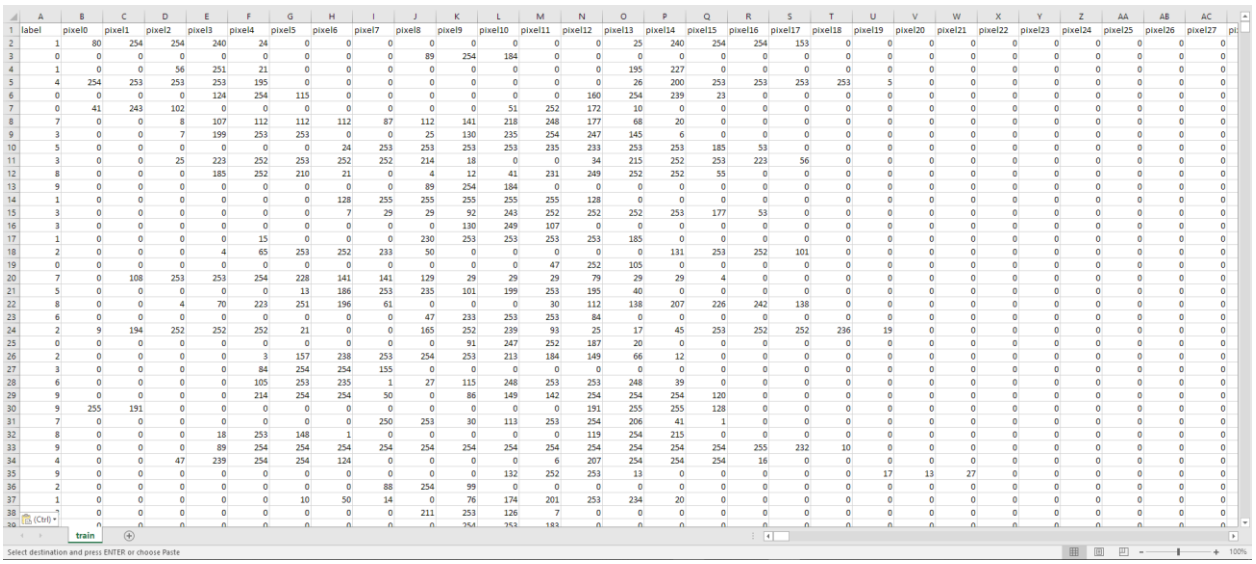
3. IMPLEMENTATION

3.1 Dataset Description

I am using the MNIST dataset, which contains 42,000 training images. The data files train.csv and test.csv contains gray-scale images of hand-drawn digits, from zero through nine.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels. Each pixel has a single value between 0 and 255, indicating the brightness of that pixel (higher numbers meaning darker).

The train.csv columns are one "label" column which represents the digit value and 784 columns for each pixel value. The test.csv file contains similar data but without the label.



3.2 System Design

Recognizer uses a two-layer neural network to classify digits from 0 to 9. The system is designed to be trained on the MNIST dataset, a large database of handwritten digits that is commonly used for training and testing in the field of machine learning.

The source dataset will be extended with a new data, I will rotate each image 90, 180, 270 degrees, and then train another model on extended dataset. The UI will allow switching between two models, trained on original and extended datasets.

The system is implemented in Python 3.7.10, using libraries such as NumPy for numerical computation, pandas for data manipulation, and PySide2 for the graphical user interface. The system also uses the PIL library for image processing and matplotlib for displaying images.

The system is divided into two classes:

MatplotlibWidget: This class is responsible for displaying images in the user interface. It uses matplotlib to create a plot and draw the image.

Recognizer: This is the main class of the system. It is responsible for loading the data, training the neural network, recognizing digits, and interacting with the user interface.

The Recognizer class uses a two-layer neural network for digit recognition. The network is trained using gradient descent, a common optimization algorithm in machine learning. The network uses the **ReLU** activation function in the first layer and the **softmax** function in the output layer.

The system also includes a **user interface** that allows the user to load a custom image, train the model, and recognize digits. The user interface displays the recognized digit and the original image.

The Recognizer class is the main class in the system that encapsulates the functionality of the neural network model. It is responsible for training the model, making predictions, and saving and loading the model parameters. Here are the main functions of the Recognizer class:

__init__: This is the constructor of the Recognizer class. It initializes the paths for source and trained models, model parameters variables and define UI widgets behavior.

forward_propagation: This function performs forward propagation through the network. It takes the input matrix X and calculates the output of the network by applying the weights, biases, and activation functions of each layer. The output of this function is the final prediction of the network.

backward_propagation: This function performs backward propagation through the network. It takes the input matrix

X, the true labels Y, and the activations of the hidden layer A1 and output layer A2. It calculates the gradients of the loss function with respect to the weights and biases of the network.

update_parameters: This function updates the weights and biases of the network using the gradients calculated in the backward propagation step. The **alfa** parameter controls the step size of the updates.

gradient_descent: Train neural network model by updating its parameters (weights and biases) iteratively to minimize the loss function (difference between the model's predictions and the actual values).

make_predictions: This function makes predictions on the input data X using the trained network. It performs forward propagation and returns the predicted labels.

train_model: This function trains the network on the input data X and true labels Y for a specified number of iterations. It performs forward propagation, backward propagation, and parameter updates in a loop for each iteration. We have separate functions for training source and extended datasets.

save_model: This function saves the weights and biases of the network to a file specified by filename. This allows the trained model to be saved and loaded later without having to retrain it.

load_model: This function loads the weights and biases of the network from a file specified by filename. This allows a previously trained model to be loaded and used for making predictions.

recognize_custom: Recognize number in JPEG files provided by users.

recognize_mnist: Recognize number form dev part of a MNIST set.

set_source_model: Set model trained on source data as current model for recognition.

set_extended_model: Set model trained on extended data as current model for recognition.

extend_source_data: Run the procedure of rotating each image in the train.csv file and save it as train_extended.csv file.

3.3 AI Concepts Utilized

The project utilizes several key concepts in artificial intelligence and machine learning:

Neural Networks: The system uses a two-layer neural network for digit recognition. Neural networks are a type of machine learning model that is inspired by the human brain. They consist of layers of nodes (or "neurons") that are connected by "synapses". Each neuron computes a weighted sum of its inputs and applies an activation function to the result.

Gradient Descent: The system uses gradient descent to train the neural network. Gradient descent is an optimization algorithm that iteratively adjusts the parameters of the model to minimize the loss function.

ReLU and **Softmax** functions: The system uses the ReLU (Rectified Linear Unit) function as the activation function in the first layer of the network, and the softmax function in the output layer. The ReLU function introduces non-linearity into the model, allowing it to learn more complex patterns. The softmax function is used to produce a probability distribution over the output classes, making it suitable for multi-class classification problems.

One-Hot Encoding: The system uses one-hot encoding to represent the labels of the digits. One-hot encoding is a way of representing categorical variables as binary vectors.

Forward and Backward Propagation: The system uses forward propagation to compute the output of the neural network given the input data and the current parameters, and backward propagation to compute the gradients of the loss function with respect to the parameters.

The **mathematical concepts** behind the neural network model can be broken down into a few key areas:

Forward Propagation: This is the process of taking an input and running it through the neural network to get a prediction. The input layer takes the pixel values of the image, normalizes them, and then passes them to the hidden layer. The hidden layer applies weights and biases to these values, and then applies a Rectified Linear Unit (ReLU) activation function. The output layer then applies another set of weights and biases, followed by a **softmax** activation function to get the final output.

Backward Propagation: This is the process of taking the prediction made by the network, calculating the error of the prediction, and then running this error backwards through the network to adjust the weights and biases. This is done using a method called gradient descent, which iteratively adjusts the parameters to minimize the error.

Loss Function: The loss function used in this model is the cross-entropy loss function. This function takes the predicted probabilities for each class and the actual class, and calculates a value that represents how well the model's predictions match the actual values. The goal of training the model is to minimize this loss.

Gradient Descent: This is the method used to adjust the weights and biases of the network during training. The idea is to calculate the gradient of the loss function with respect to each parameter, and then adjust the parameter in the opposite direction of the gradient. This is done iteratively until the model converges to a set of parameters that minimize the loss.

Activation Functions: These are functions applied to the outputs of each layer in the network. The ReLU activation function is used for the hidden layer, and the softmax activation function is used for the output layer. The ReLU function introduces non-linearity into the model, allowing it to learn more complex patterns. The softmax function ensures that the output values can be interpreted as probabilities, as it squashes the outputs to be between 0 and 1 and ensures they sum to 1.

Vectorized Implementation: This refers to the use of matrix and vector operations to perform calculations on all training examples at once, rather than looping over them one by one. This is a more efficient way to implement the model, especially when dealing with large datasets.

Model Parameters: These are the weights and biases that the model learns during training. The weights are the coefficients that determine how much each input contributes to the output, and the biases are constants that are added to the output. These parameters are initialized randomly and then updated through gradient descent during training.

3.4 Lessons Learned

This project provided valuable experience in the development of a machine learning application, from the initial design and implementation to the testing and evaluation of the system. It also provided a practical understanding of key concepts in artificial intelligence and machine learning, including neural networks, gradient descent, activation functions, and data preprocessing.

Data Preprocessing: The input images were preprocessed by normalizing the pixel values to be between 0 and 1. This is a common practice in machine learning to ensure that all input features have a similar scale, which can help the model to converge faster during training.

Model Training and Evaluation: The model was trained using the MNIST dataset and evaluated on a separate test set to assess its performance. This highlighted the importance of splitting the data into a training set and a test set to avoid overfitting and to get an unbiased estimate of the model's performance on new data.

Model Saving and Loading: The ability to save and load trained models is a useful feature in machine learning applications. In this project, the trained weights and biases of the neural network were saved to CSV files, allowing the model to be reloaded and used without having to retrain it.

User Interface Design: Designing a user-friendly interface is important for making machine learning applications accessible to non-expert users. In this project, a graphical user interface was created using PySide2, allowing the user to interact with the system in a simple and intuitive way.

Debugging and Error Handling: During the development process, various bugs and errors were encountered. These were addressed through careful debugging and error handling, which is a crucial aspect of any software development project. This included handling potential issues with data input, ensuring the correct dimensions for matrix operations, and checking for numerical stability during computations.

Performance Optimization: The project also highlighted the importance of performance optimization in machine learning. For example, vectorized implementations were used for the forward and backward propagation steps, which are significantly faster than using for loops. This is particularly important when dealing with large datasets or complex models.

Understanding of AI Concepts: Implementing a neural network from scratch provided a deep understanding of how neural networks work, including the process of forward propagation, the calculation of the loss function, and the update of weights and biases through backward propagation. It also reinforced the understanding of key concepts such as gradient descent, activation functions, and one-hot encoding.

3.5 Challenges and Detailed Results

During the project, several challenges were encountered, particularly in data handling and model training. The rotation of images introduced significant complexity to the model. Initially, the model's performance decreased, indicating difficulty in generalizing from the augmented data. This was particularly evident when the model trained on the extended dataset showed a marked decrease in accuracy, dropping to around 0.4 despite adjustments in learning parameters.

3.6 Efficiency of Training and Testing

The training process was initially set up with a fixed number of iterations and a learning rate that was not optimized for the augmented dataset. This led to inefficient training sessions where either the model would overfit or underfit drastically. The use of batch processing and iterative training with validation checkpoints helped in optimizing the training phase, but the process still required significant computational resources due to the increased dataset size from image rotations.

4. RESULTS

User Interface

Once user launches application the Recognizer main window rise:

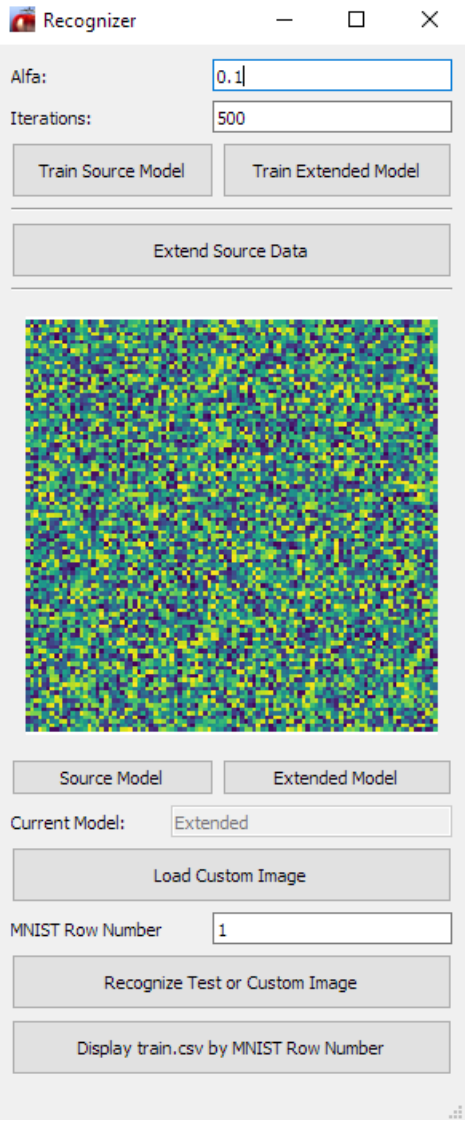


Image 1. The Recognizer main window

The UI explanation:

Alfa: Set learning rate parameter when train the model

Iterations: Set number of iterations for model training

Train Source Model: Run the process of training model based on source data set and save results to CSV files.

Train Extended Model: Run the process of training model based on data set extended with rotated images and save results to CSV files.

Extend Source Data: Rotate each number in train.csv file on 90, 180, 270 degrees and save this data as a train_extended.csv. This file will be used to train extended model.

Image Display: Shows the image of digit that we recognizing with “Recognize Test or Custom Image” function.

Source Model: Set model trained on source data set as current model for digit recognition

Extended Model: Set model trained on extended data set as current model for digit recognition

Current Model: shows which model currently used for recognition.

Load Custom Image: Load and show in UI custom JPEG image of handwritten digit to recognize it.

MNIST Row Number: Set row number from test dataset (test.csv) for digit recognition.

Recognize Test or Custom Image: Run recognition process and display results in the Status Line: “Custom Number 2 recognized as 2”

Display train.csv by MNIST Row Number: Debugging function, shows the image from train.csv file in the UI by row index set in **MNIST Row Number** field.

User Workflow

Once loaded the first time, user can start extending the source data set and training. First user suppose to train model on source data set and check results with that model, then the dataset can be extended and new model can be produced. The user can switch between two models and check their abilities to recognize digits including rotated ones.

5. FUTURE IMPROVEMENTS

While the current system is able to recognize handwritten digits with a high degree of accuracy, there are several potential improvements that could be made:

Extend source dataset further with scaled images: Currently only rotation is implemented, it would nice to scale source images non proportionally to extend recognition abilities of the application.

Adding More Hidden Layers: The current model is a two-layer neural network with one hidden layer. Adding more hidden layers could potentially improve the model's ability to recognize more complex patterns in the data.

Using Different Activation Functions: The current model uses the **ReLU** activation function in the hidden layer and the **softmax** function in the output layer. Experimenting with different activation functions could potentially improve the model's performance.

Hyperparameter Tuning: The learning rate and the number of training iterations are important hyperparameters that can significantly affect the model's performance. These hyperparameters could be tuned using techniques such as grid search or random search to find the optimal values.

Implementing Regularization: Regularization techniques such as L1 or L2 regularization could be implemented to prevent overfitting and improve the model's generalization ability.

Improving the User Interface: The user interface could be improved by adding more features, such as the ability to adjust the learning rate and the number of training iterations, or to visualize the training process.

Data Augmentation Control: Introducing a control mechanism to selectively augment the dataset could prevent overwhelming the model with complex variations, focusing on incremental learning.

Enhanced Model Architecture: Considering a deeper or more complex network might be necessary to handle the increased complexity of rotated images. Incorporating convolutional layers could also be explored to better capture spatial hierarchies in image data.

6. CONCLUSION

In the Project Proposal I expected to implement the non-proportional scale of all digits but decided to skip this part due to the time constraints. Missing this feature seems acceptable for the learning purposes of the project.

The implementation of Recognizer trained on source data set is successful, the model trained with high accuracy and able to recognize handwritten digits from test data set and my custom digits as well.

The extension of the dataset with rotated images lead to recognizing images with much less accuracy than expected. I tried to adjust Learning Rate (Alpha) and number of iterations. I get quite high accuracy number of 0.76 using 5000 iteration with 0.8 alpha, but the model ability is quite poor.

There are some strategies that can be utilized to achieve better results with extended data set. I might need to increase number of layers, try different activation functions, increase epoch and adjust batch size.