Neural Network for recognizing hand-drawn images

[2000185] MACHINE LEARNING

Silas Ueberschaer
Università degli Studi di Siena

Benjamin Pöhlmann
Università degli Studi di Siena

Abstract

This project revolves around the implementation of a neural network for image classification, focusing on handwritten drawings that we defined ourselves. The neural network is designed using Python and leverages the NumPy library for numerical operations. The network architecture is modular, with the configurable layers defined in a separate file. The project also includes a user interface developed with Pygame, enabling users to draw images for real-time predictions and training. Users can also view drawings from the dataset and decide to delete them. Furthermore we visualized the network where for each input we can see the activation of the neurons and how much they influenced the decision.

Keywords

neural network, classification, on-line learning, handwritten recognition

1. Introduction

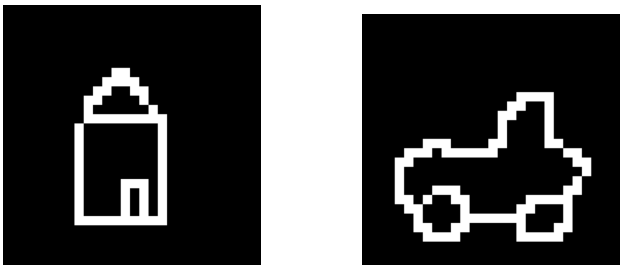
In our current era of rapid development in the fields of artificial intelligence and machine learning, the application of neural networks have become important in addressing complex tasks across various domains. One such application is image classification, a field that has taken a big leap in the past few years, especially with the advancements of deep learning. In this project we focus on the creation of a neural network for image classification of hand-drawn 28x28 images from predefined classes. Each pixel in the image can either be 0 (white) or 1 (black). To implement this neural network we used Python and in specific the numpy library for numerical operations.

The motivation behind this project stems from diverging from the MNIST handwritten digit dataset and create our own dataset of own classes to fully experience the difficulties in creating a network end-to-end. This encapsulates data creation, labeling and then training the model until it can be used to predict new objects and enable online-learning. Handwritten drawings pose an interesting challenge due to their inherent variability, making it less sterile and more exciting for exploration. The project's scope extends beyond only the development of a neural network: We included the integration of a user interface using Pygame, providing users with the possibility to contribute to the model's training dataset by drawing new images and see real-time predictions as we draw each single pixel.

The neural networks' architecture is designed with modularity in mind, facilitating easy extension and experimentation. Leveraging object-oriented approach, each layer of the network is an isolated instance of our own created "Layer"-class, promoting a clear and organized structure. Therefore we ensure that we can easily adapt input-size, output-size or the activation function. The project focuses on the stochastic gradient descent training mechanism and mini-batch processing. This allows the model to iteratively adjust its parameters, enhancing its ability to classify hand-drawn images accurately. Therefore the network is able to adjust its parameters for each new image it sees.

To initially create and then augment the dataset and engage users in the training process, a user interface is implemented using Pygame. This interactive canvas allows users to draw images directly into the system, offering a direct impact on dataset creation. Users can assign specific classes to their drawings, ensuring a diverse and well-labeled dataset for training. The classes are predefined and can not be extended without further modification.

In the subsequent sections we will look into the technical details of the network architecture, the training process, experimental results, and the user interfaces' role in dataset creation and augmentation. Through this exploration, we aim to showcase the potential of the developed system in recognizing and classifying hand-drawn images. The following Figure 1 shows an excerpt of examples to give an impression of how examples were drawn.



(a) House **(b)** Car

Figure 1: Examples of own drawings with the UI

2. Neural Network Architecture and Implementation

The neural network at the core of this project is designed to handle image classification tasks, specifically focusing on recognizing hand-drawn images. The architecture is structured as a fully connected feedforward neural network with two hidden layers that have 16 neurons each. Every one of these is responsible for extracting relevant features from the input data. The network implementation is realized using Python and NumPy.

The modular design of the neural network architecture makes the network flexible and easy to extend in case of experimentation. For the layers we wrote our own class "Layer" promoting clean code principles. The layers consist of weights and biases, initialized with random values, and an activation function to introduce non-linearity into the model. The choice of activation function, in this case, is by default the sigmoid function. But this can be changed when instantiating a new Layer and introducing the implementation in the `activation.py` file.

Forward propagation through the network involves passing the input data through each layer, computing the net activation, and applying the activation function to generate the output used in subsequent layers. This process is encapsulated in the `forward` function within the network.py file. During training, the network's ability to learn and generalize is enhanced through the iterative adjustment of weights using our backpropagation algorithm.

The backpropagation algorithm, implemented in the `backward` function, computes the gradients of the loss of each example with respect to the weights and biases. These gradients guide the update of parameters during the optimization step, enhancing the model's performance over time. The training process, orchestrated by the train function, involves multiple epochs where the entire dataset is processed, and the models' parameters are updated to minimize the empirical risk.

To further ease experimentation, the project includes a user interface developed with Pygame. This interface allows users to draw images directly onto a canvas, which are then incorporated into the training dataset if desired. The training process is initiated through user interactions, providing a dynamic approach to dataset creation.

In summary, this section outlines the neural network architecture and its implementation, emphasizing modularity and clarity.

3. Training Process and Hyperparameter Tuning

The effectiveness of the neural network crucially depends on its training process and the fine-tuning of hyperparameters like the learning rate. This section describes the training methodology used in this project, along with experiments conducted to find the optimal hyperparameters and enhance the model's accuracy.

The training process is initiated by iterating over multiple epochs, with each epoch representing a complete pass through the training dataset. The dataset is divided into mini-batches, and stochastic gradient descent is employed for optimization. The use of mini-batches helps to introduce randomness into the parameter updates, preventing the model from getting stuck in local minima and improving convergence.

Hyperparameters play a pivotal role in determining the performance of the neural network. Key hyperparameters include the learning rate, batch size, and the number of epochs. The learning rate influences the size of parameter updates during optimization, while the batch size determines the number of samples used in each iteration. Additionally, the number of epochs defines how many times the entire dataset is processed during training.

Several experiments were conducted to understand the impact of different hyperparameter configurations on the model's accuracy. The learning rate was varied to observe its effect on convergence speed and final accuracy. Similarly, batch sizes ranging from small to large were tested to identify the optimal size for efficient training.

Results from these experiments, outlined in the provided `experiments.txt`, highlight the trade-offs associated with different hyperparameter choices. For instance, higher learning rates might accelerate convergence but risk overshoot-

ing the optimal values, while larger batch sizes may offer computational efficiency but might lead to slower convergence.

In conclusion, this section sheds light on the training process and the critical role played by hyperparameters in shaping the neural network's performance. The provided experimental results offer valuable insights into optimal configurations, guiding users in selecting appropriate hyperparameters for their specific use cases. The subsequent section will delve into the experimental results, providing a comprehensive analysis of the network's accuracy under varying conditions.

4. Results

Noteworthy experimental results include achieving an accuracy of 93% with a batch size of 10 over 50,000 epochs, demonstrating the network's capacity to learn complex patterns from hand-drawn images. Additionally, comparisons between different learning rates and batch sizes provide insights into the trade-offs between training speed and accuracy.

When using such a trained model in online mode by drawing new images we can instantly see what parts of the image correspond to increasing the predicted probability each class. To better understand the network and grasp the impact of each individual neuron and connection we decided to visualize the network in `ui_network.py` which can be seen in Figure 2. Through this visualization we can easily see the importance of specific neurons for the classes.

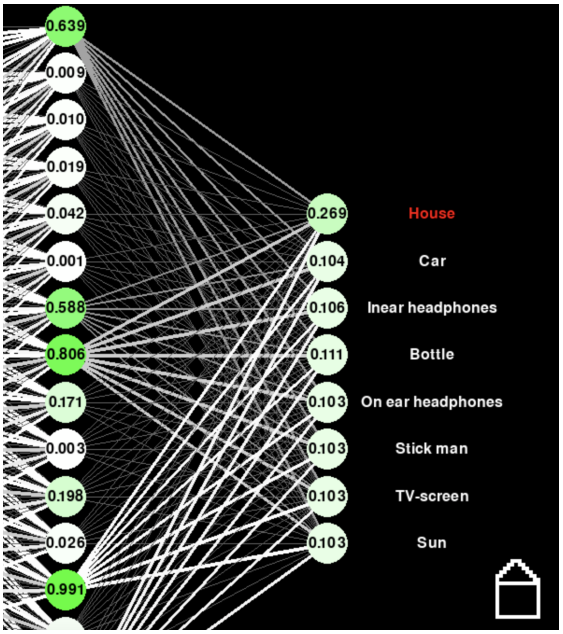


Figure 2: Visualization of the Network

These results can serve as benchmark for other projects trying to achieve similar results.

5. Limitations

While the project shows promising results already, there are still a lot of limiting factors at play. The dataset consists only of around 500 images which means that each class has only 50-80 images. This means that we couldn't capture the variability in data that will inevitably result from collecting more drawings. Also as people have different intuitions of how to draw a class such as a car in this limited canvas space, this can result in very inaccurate predictions because the difference for the reference patterns can be large. Spatial dimensions can also not be accounted for. If we draw the exact same object twice but in very different parts of the canvas they will be seen as very different objects. To mitigate this risk, use of a convolutional layers might be used.

Due to this being a student project we only had limited resources for creating and training this model from scratch.