Train MLP With GA Report

1. Introduction:

MLP is used for solving a handwritten digit classification problem with MNIST dataset. The weights and bias are not trained with gradient descent methods but genetic algorithm.

2. Multilayer Perceptron Trained by Genetic Algorithm:

2.1. Short description:

MLP is generated with **scikit-learn MLPClassifier**. Without training the model directly with prebuilt fit() function, the initial weights and bias are trained with genetic algorithm.

Dataset used in this task is MNIST dataset. This is a dataset with 60000 train data and 10000 test data of handwritten digits. The purpose of the model is to classify the digits precisely. The input is a handwritten picture with 28 x 28 pixels while the output is an identified number with label [0 1 2 3 4 5 6 7 8 9].

The dataset can be downloaded from <http://yann.lecun.com/exdb/mnist/>. Training set and testing set are already separated.

2.2. Multilayer Perceptron:

Multilayer perceptron is formed by multi fully connected layers. Its architecture is simple and direct. It can be used for relatively simple tasks easily with good performance. However, with increasing layer, the number of parameters become extremely large, causing redundancy in dimensions. The efficiency is thus low, especially in the case of many hidden layers [1].

On the other hand, Convolutional Neural Network (CNN) plays a more important role in nowadays computer vision tasks. It allows local connectivity of image capture at parts of the image with different kernels. The weights in layers are smaller and shared. With this advantage, much deeper model can be constructed for higher performance. In a result, CNN is very effective and efficient.

Diagram

Description automatically generated

In this project, MLP is applied to MNIST datasets, which is a relatively simpler dataset. As the layers in MLP are fully connected. It is expected that the training time would be quite long, especially we train the model using genetic algorithm instead of gradient descent approach.

2.3. Challenges:

Trial training is carried out with simple model with only 1 hidden layer of 10 perception. Raw data is used for training. The performance of training is very bad as the capture as below.



Even after 100 generation, the training accuracy is still only around 33% and the rate of convergence even decreases. This may be caused by the reason below.

The dimension of raw data is very high with 784(28 x 28 pixels). Assume model with only 1 hidden layer of 10 perceptron is used, together with the output layer, total number of weights of the model is 784 x 10 + 10 x 10 while the total number of biases is 10 + 10. Then, the length of chromosome in genetic algorithm is 7960 which is very large. It is very hard for the genetic algorithm to cover such a large search space. Every time crossover and mutation cannot help converging the variables. Then, it results in bad performance.

To tackle the problem, some data preprocessing methods, PCA and normalization of data are introduced. Details will be mentioned under Methodology part.

2.4. Genetic algorithm (GA):

Genetic algorithm is used to train the MLP in this project. Below are the functions in GA in details.

**2.4.1. Population generation:**

First, a population is initiated with population size given. The gene is to be in range [-1, 1]. The length of chromosome is the total number of variables in the model as mentioned in part 2.2. For example, data without PCA with a single hidden layer with 10 weights contributes length of 7960 in a chromosome.

**2.4.2. Evaluation function:**

The score of chromosome is calculated by the score function from **skleran.neural\_network.MLPClassifier**. It compares the predicted outputs with labels for the accuracy of model.

**2.4.3. Selection function:**

Each chromosome is given a score by evaluation function. The chromosomes are ranked by their scores. Then, they are assigned different probabilities to be selected. The higher the score, the higher the rank, the larger the probability.

**2.4.4. Crossover function:**

Each time, two chromosomes are selected by selection function for crossover to generate two child chromosomes. A random number in [0, 1] is generated. If the number is larger than crossover rate, the two chromosomes are cut at a random position and crossed over to form the child chromosomes.

**2.4.5. Mutation function:**

Each gene in the input chromosome is viewed one by one. Each time a random number again in range [0, 1] is generated. If the number is larger than mutation rate, a new gene with random number in range [-1, 1] replaces the existing gene.

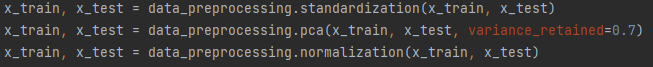
**2.4.6. Elitism function:**

Best chromosomes in parent population replaces worst chromosomes in child population with elitism size. For example, with 0.1 elitism size and 10 population size. One best chromosome in parent population replaces one worst chromosome in child population.

2.5. Methodology:

**2.5.1. Data preprocessing:**

As mentioned above, data preprocessing is adopted for more efficient training.



Standardization:

Before PCA, standardization is used on data as it is a prerequisite for PCA. Data is standardized with mean 0 and s.d. 1. **sklearn.preprocessing.StandardScaler** is used.

Principal component analysis (PCA):

PCA is a way to reduce dimensions of raw data. Normally, not all attributes in data have much contribution to the prediction. Some attributes with low contribution can be omitted for simpler training. **sklearn.decomposition.PCA** is used for PCA on raw data.

When the dimensionality is lower, the number of variables for training can be reduced. The search space of genetic algorithm is smaller, improving the performance. Different values of retained variance are used for training.

Normalization:

Normalization is another way help training. This can make all the attributes equally important. This facilitates the training performance. Data is normalized with range [0, 1]. **sklearn.preprocessing.normalize** is used

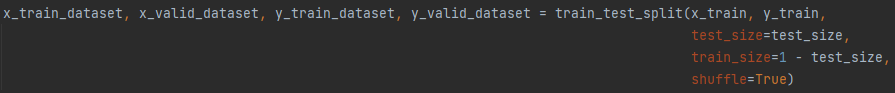
**2.5.2. Construction of MLP model:**

MLP model is constructed by **skleran.neural\_network.MLPClassifier** with various architectures. The inputs of the MLP are 60000 images with 28 x 28 pixels. The data will pass through a set of weights and biases in the model and give a prediction on the handwritten digits. The outputs are integers with label 0-9.

The weights and biases are the key of getting a good prediction. In this project, genetic algorithm is used for training. All variables in the model are flattened as a chromosome with long 1D array.

**2.5.3. Training using GA:**

In each epoch, train dataset is first split into training set and validation set with validation ratio of 0.2 with shuffle. This can ensure cross validation of data and enhance training efficiency at the same time.



Genetic algorithm is initialized with following parameters. Generation limit 10 here means each batch of data will be experienced 10 generation. Other parameters are quite straight forward. Population is initialized at the first epoch.

Text

Description automatically generated

Then, the model is trained with 15 epochs. In each epoch, training dataset is split into train loader with batch size 10000. This can train the model more efficiently.



Below is the core of training algorithm.

Text

Description automatically generated

In each generation under each epoch, a new population is generated by some GA functions. Selection function is used to find two parent chromosomes. Crossover function is used to produce child chromosomes from the parent chromosomes. Mutation function is used for any potential mutation on the child chromosomes. New chromosomes are kept generating until the population is full. Elitism is then used to replace worst chromosome in new population by best chromosome in old population.

For each 10 generations, next data loader is used for training. This can let each data participate in training the model, avoiding overfitting. After all datasets being trained, the model is passed to next epoch for training again. The training will be stopped after 10 epochs. Below is a screen capture of log during training.

Table

Description automatically generated

2.6. Results and Analysis:

Testing the model with test dataset, the accuracy is 0.5232. The performance of the model is not that good.

Below is a plot of accuracy vs epoch, displaying the trend of train accuracy and validation accuracy.

Chart, line chart

Description automatically generated

From the graph, no overfit problem occurs and the accuracy increasing smoothly and gradually. The training algorithm works fine. However, the rate of increase drops after accuracy of 0.45. The learning rate of the model becomes very slow, but there is still progress in each epoch. To further improve the accuracy, despite long computation time, it is believed that more generation for each batch data and more epochs can further improve the accuracy.

Using GA, it depends on certain randomness. It does not act like gradient descent, which can find a clear orientation for optimization. GA needs to generate new population compare it with previous population for update. When accuracy is getting higher, probability of finding better chromosome becomes lower, causing a drop of learning rate.

Moreover, training time of GA is relatively long. The above test with 15 epochs has spent 6 hours. On the other hand, mlp.fit function which uses gradient descent only spends several minutes for training the same model with over 90% accuracy. Thus, GA is not an efficient way to train complicated model.

2.7. Further studies:

**2.7.1. Different hidden layer**

1 hidden layer with 10 perceptron is used in previous training. A model with 2 hidden layers with 10 perceptron in each layer is trained for around 100 generations for comparison. Below is the learning curve of the two models.

Chart, line chart

Description automatically generated

Generally, the accuracy of 1-layer model increases faster than that of 2-layer model. This can be explained by the nature of GA. Number of variables in 1-layer model is 1100 while number of variables in 2-layer model is 1210. GA replies in crossover and mutation for generating new population and then evolves. Although more variables may lead to a more accurate model, with larger search base, GA cannot find a better solution efficiently. It takes much longer time for the model to be trained well. Given a generation limit, model with fewer layers performs better.

**2.7.2. Different number of hidden units**

The number of hidden units in the trained model is 10. With the same layer, different number of hidden units is tested. Below is the learning curve of the models with different number of hidden units.

Chart, line chart

Description automatically generated

From the graph, no obvious difference in accuracy with different number of hidden units. Their learning curves are similar in the first 50 generations. More hidden units mean longer chromosome. This may result in larger search space which has negative impact on GA training. However, the learning curves here do not differ much. Even the number of hidden units increase, the number of hidden layers remains 1. The structure of the model remains simple. This may be the reason of similar learning curve despite different chromosome lengths.

|  |  |  |
| --- | --- | --- |
| Number of hidden units | Computation time (s) | Chromosome length |
| 5 | 1114 | 555 |
| 10 | 1443 | 1100 |
| 100 | 4498 | 10910 |

Despite the similar learning curves, computation time increases with number of hidden units of the model. Some functions in GA need to evaluate the chromosome one by one, they take longer time when chromosome length is larger. For example, mutation function needs to go through each gene in a chromosome. The computation directly increases with the length of chromosome.

Thus, within small number of generation (around 50), small number of hidden units in model can achieve same performance with shorter training time.

3. References:

[1] Uniqtech, “Multilayer perceptron (MLP) vs Convolutional Neural Network in deep learning,” *Medium*, 13-Jun-2019. [Online]. Available: https://medium.com/data-science-bootcamp/multilayer-perceptron-mlp-vs-convolutional-neural-network-in-deep-learning-c890f487a8f1. [Accessed: 01-Dec-2021].