INF 552 Programming Assignment 5

Neural Network

4472-6221-33 Yo Shuan Liu

I. Introduction

I complete this homework in Python3. My code can be executed from terminal, and there is no absolute path for input files. User only needs to follow below instructions to execute this script:

1) Put all the input file/folder needed (downgesture_train.list.txt, downgesture_test.list.txt, /gestures) and this code **under the same directory**.
2) Change directory to this folder and execute below line in terminal:
   python NNet.py
3) After printing out all the output, a message will show up in the terminal and ask user to **press [enter]** in order to end the execution and close all the output graph windows.

In this report, I will introduce my implementations of neural network using mini-batch gradient descent and vanilla gradient descent, and compare the difference among them. I prefer using the vanilla gradient descent for training neural network in this assignment, because it has the same good result as the mini-batch one and the result is reproducible.

II. Data Structure

As for neural network structure, I use 961 input neurons (960 for the image grids and 1 for the bias), 100 neurons for hidden layer (included 1 bias neuron) and 1 output neuron. I take advantages of numpy array, and store the input data and weights in 2D arrays. I also store the calculated loss of each epoch in a list for evaluation purpose.

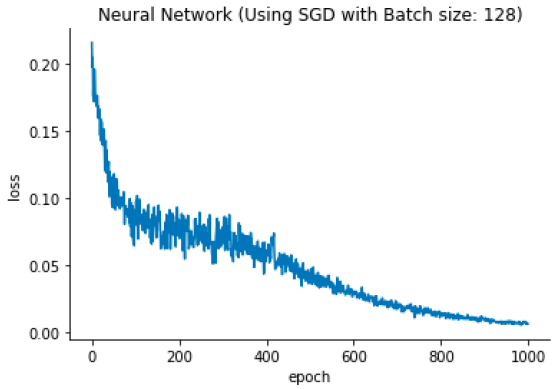
III. Implementation of Neural Network Using Mini-batch Gradient Descent

Why choosing mini-batch gradient descent over stochastic gradient descent?

I didn't use the stochastic gradient descent (which batch size is 1), because the loss would be so fluctuating. And also because our training dataset isn't a really large one, it is feasible to implement mini-batch and take advantages of vectorized implementation for faster computations.

Since we have 184 training samples, I choose 128 as my mini-batch size for efficiency reason. It is a good idea to choose a batch size which is power of 2, taking that the page size of our CPU is also power of 2. And smaller batch size makes the loss more fluctuating, therefore I end up choosing 128 as my mini-batch size.

Attached loss over 1000 epochs of neural network using mini-batch gradient descent with batch size 128 as below. Accuracy of 95.18% is the best result I have seen so far. However, the accuracy falls between 86.75% and 87.95% in most cases. This is another reason why I prefer using the vanilla gradient descent (which I will introduce later).

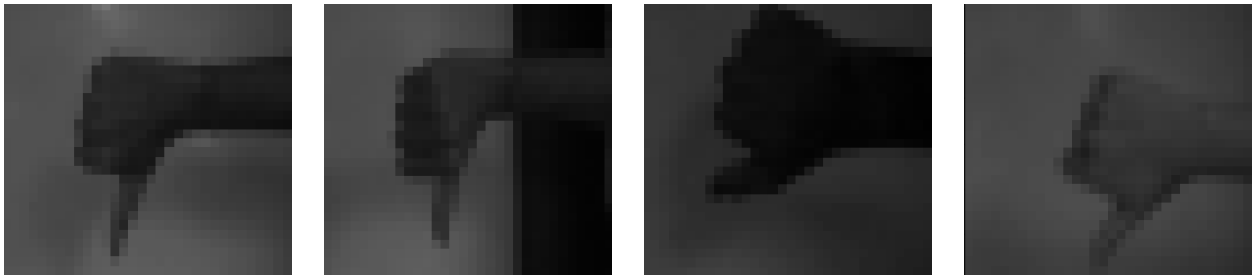



```
Accuracy on Test Data: 95.18%
```

Sample with replacement.
And the stochastic process of my mini-batch is done with replacement. I realize that the mini-batch gradient descent is drawn without replacement normally, and I will explain my specific reason in the challenges I face session.

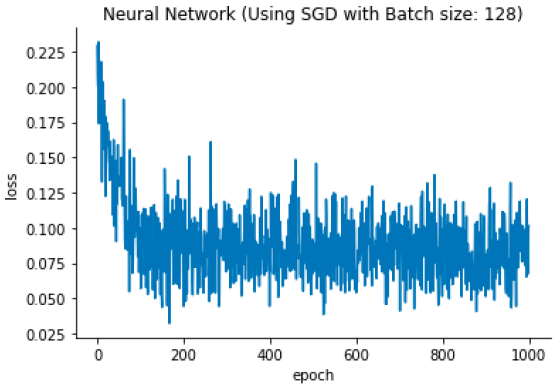
Result
Accuracy: 95.18%
Prediction on Test Data:
[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0]

Wrong classification on test case 8, 9, 16, 24.



Challenges I Face: With replacement or without replacement?
I realize that the in most cases mini-batch gradient descent is drawn without replacement. That is, each data in the training dataset is scanned exactly once in an epoch. However, I realize that for our dataset, we get a less fluctuated loss over epochs if we sample our training data for mini-batch with replacement. That is, one training sample may show up several times in different batch in the same training epoch.

Attached the loss over 1000 epochs if we drawn without replacement as below.



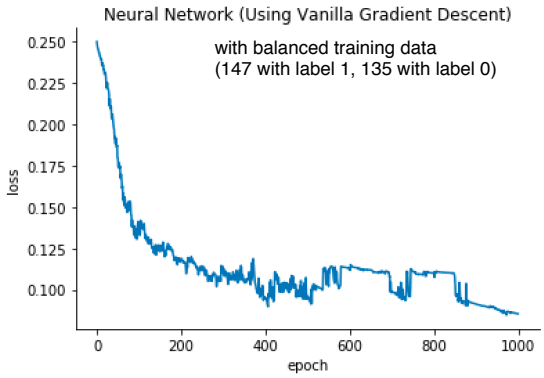
IV. Implementation of Neural Network Using Vanilla Gradient Descent

Implementation
Vanilla gradient descent is the one taking all the training sample into account in every epoch. Since our training data isn't really huge, vanilla gradient descent is also feasible in our case. The accuracy is 87.95% on test data without any data preprocessing. However, if we balance our data by tripling the

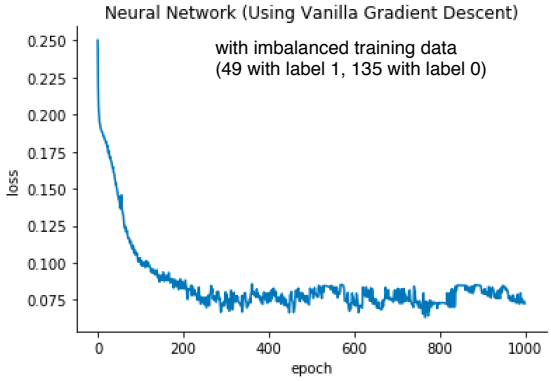
amount of training data with label 1 (down gesture), we can increase our prediction accuracy to 95.18% on the same test dataset.

It also has a less fluctuated loss than the stochastic gradient descent and mini-batch gradient descent.

Moreover, unlike the mini-batch or stochastic gradient descent, we can easily reproduce the result of neural network using vanilla gradient descent by assigning a random seed.



Accuracy on Test Data: 95.18%



Accuracy on Test Data: 87.95%

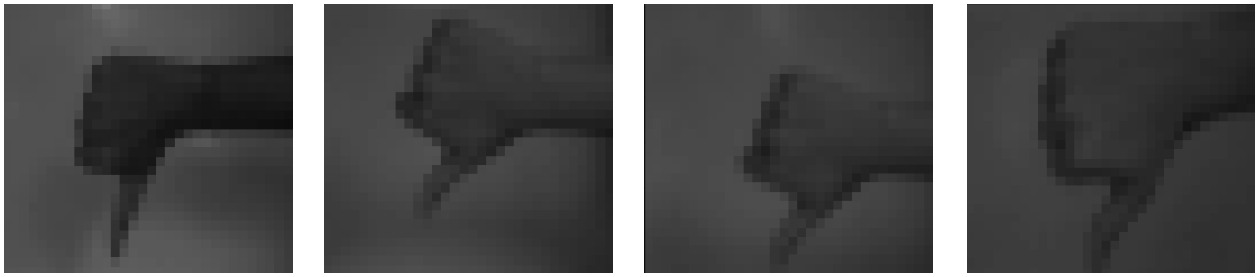
Result

Accuracy: 95.18%

Prediction on Test Data:

[1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0]

Wrong classification on test case 8, 23, 24, 46.



V. Software Familiarization

sklearn.neural_network.MLPClassifier

MLP classifier allows users to choose between four activation functions (‘identity’, ‘logistic’, ‘tanh’, and ‘relu’), and also the solver for optimizing weights ('lbfgs’, ‘sgd’, ‘adam’). In our assignment, we used ‘logistic’ as activation function and ‘sgd’ as the solver. In addition, users can also specify the learning rate and if an adaptive learning rate is needed. However, it supports only the Cross-Entropy loss function.

VI. Code Level Optimization

1. I write this code in an object-oriented manner. There is a function for inputing training and testing data, and a class containing all functions needed for training.

2. For the vanilla gradient descent, I improved the performance by applying a data preprocessing technique to balance the training dataset. Initially, training dataset has 49 data with label 1 and 135 data with label 0. I tripled data with label 1 and finally obtained a better classification result.

VII. Applications

There are lots of interesting applications of neural network, and also many different kind of neural network. One of them is convolutional neural network, known as CNN. Convolutional neural network is widely used for speech recognition, text classification and also image classification (just like what we have done in this assignment) and labeling.

Speech recognition has many applications, such as home automation, mobile telephony, virtual assistance, hands-free computing, video games, and so on. Text classification is also an essential part in many applications, such as web searching, information filtering, language identification, readability assessment, and sentiment analysis.

Reference:
https://medium.com/@datamonsters/artificial-neural-networks-in-natural-language-processing-bcf62aa9151a