ECE657, Spring 2020, Assignment 3

Zhijie Wang, [redacted]

July 14, 2020

1 Problem 1

1.1 Data Preprocessing

Step (1) Normalize each channel of each image into 0.5 mean and 0.5 standar variance.

Step (2) (Optional) Resize each image from 32*32*3 into 3702*1 for MLP.

1.2 Output layer settings

This is a classification problem, therefore, we choose the Cross Entropy Loss as loss function, and the output layer should be activated by softmax function. (However, in PyTorch, nn.CrossEntropyLoss() already contains the softmax function, therefore, we don't need to write it in the forward function of network class.)

1.3 Training settings

learning rate = 10^{-4}, optimizer = $Adam$, loss function = $CrossEntropyLoss$, epoch = 5

1.4 Training environments

GPU: NVIDIA GTX 1660 Super (6GB), CPU: AMD 3600 (4.0GHz), PyTorch 1.5.1

1.5 Change parameters of MLP

Here we test 4 diffrent settings of MLP, which have hidden nodes [512, 512] (original), [1024,1024], [1024, 512], [1024, 512, 512]. As we can see from the following figures, adding layers cannot increase the accuracy, however, adding nodes on a layer can increase it.

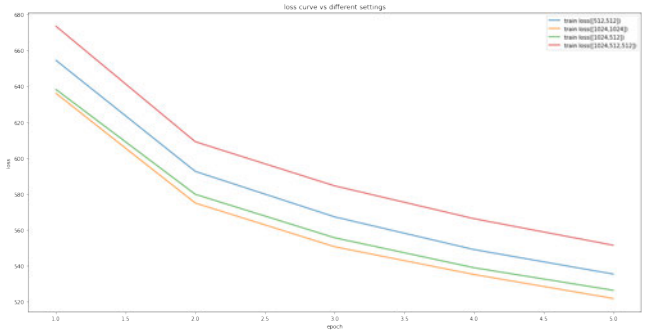


Figure 1

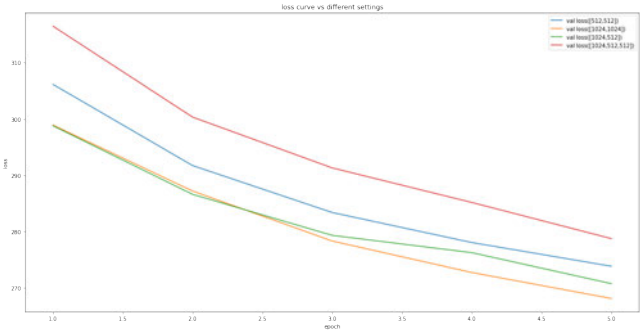


Figure 2

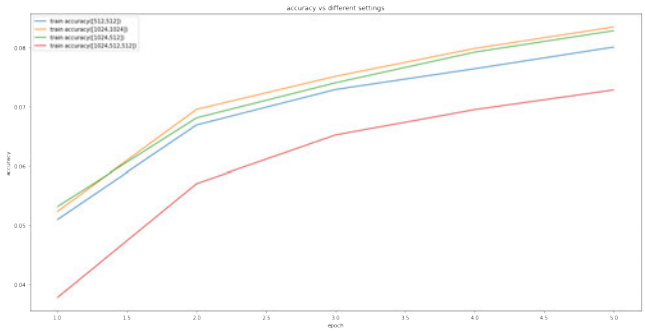


Figure 3

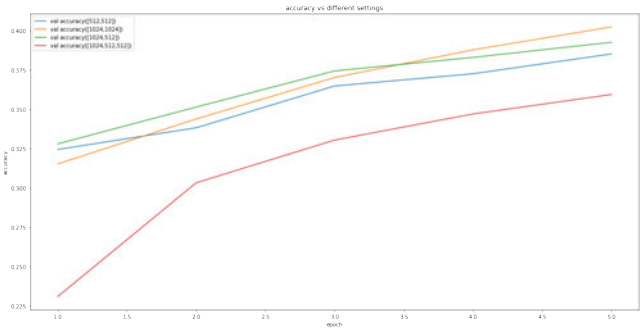


Figure 4

1.6 Train & test accuracy

	MLP	CNN1	CNN2
Train Accuracy	0.078420	0.118000	0.089480
Test Accuracy	0.379600	0.513800	0.455200

1.7 Train & validation curve for two CNNs

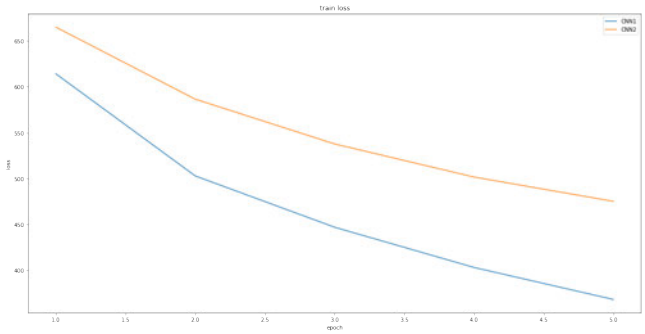


Figure 5

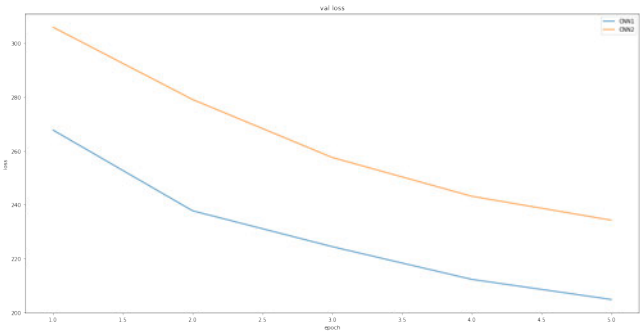


Figure 6

As we can see from the above figure, CNN1 has higher train and validation accuracy. For the training time, CNN1 runs around 8 seconds for an epoch, CNN2 needs around 4 seconds. The pooling layer reduce the output dimension, therefore reduce the training time. If training more epochs, I believe both two CNN can achieve a

high accuracy comparing to MLP.

1.8 Recommendation for improving

To improve the performance, I think we can add more convolutional layer, so that the network can extract features better.

2 Problem 2

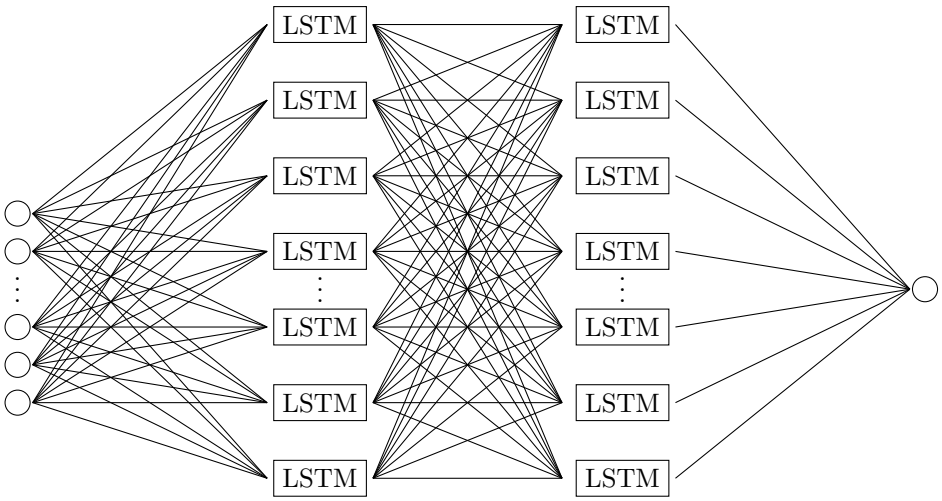
2.1 Creating Dataset

```
def create_dataset(path, n_steps=3):
    df = pd.read_csv(path)
    features_column = [" Volume", " Open", " High", " Low"]
    dates = df["Date"].values
    sequence = df[features_column].values # read 4 features from original dataset
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps # n_stpes corresponding to n dates as features
        # if the remained data size < n_steps
        if end_ix > len(sequence) - 1:
            break
        # first pattern as target, another n_steps patterns as features
        seq_x, seq_y = sequence[i+1:end_ix+1], sequence[i][1]
        X.append(seq_x.reshape(12))
        # clean date format
        date = dates[i] if len(dates[i]) == 8 else (dates[i][:6] + dates[i][8:])
        y.append([seq_y, date])
    X, y = np.array(X), np.array(y)
    n = len(X)
    indices = np.random.permutation(n)
    X, y = X[indices],  y[indices]
    split = int(n * 0.7)
    X_train, X_test, y_train, y_test = X[:split], X[split:], y[:split], y[split:]
    # n_step * 4 features
    columns = ["f" + str(i) for i in range(n_steps*4)]
    columns += ["target", "date"]
    train_df = pd.DataFrame(data=np.concatenate((X_train, y_train), axis=1),
                            columns=columns)
    train_df.to_csv('./data/train_data_RNN.csv')
    test_df = pd.DataFrame(data=np.concatenate((X_test, y_test), axis=1),
                           columns=columns)
    test_df.to_csv('./data/test_data_RNN.csv')
```

2.2 Date Preprocessing

Step (1) Applying MinMaxScaler to dataset

2.3 Network Structure



2.4 Training

2.4.1 Training settings

learning rate = 10 ^{2}, optimizer = $Adam$, loss function = $MSELoss$, epoch = 300

2.4.2 Training environments

GPU: NVIDIA GTX 1660 Super (6GB), CPU: AMD 3600 (4.0GHz), PyTorch 1.5.1

2.4.3 Training process and output

First we show the plot of training loss as the output of training. Then, we show the prediction result on training set comparing to ground truth (for better visualization, we sort the value according to dates)

2.5 Testing result

Testing loss: 0.000231777157750912. Then we show the prediction result on testing set comparing to ground truth (for better visualization, we sort the value according to dates) As we can see, the training loss decrease

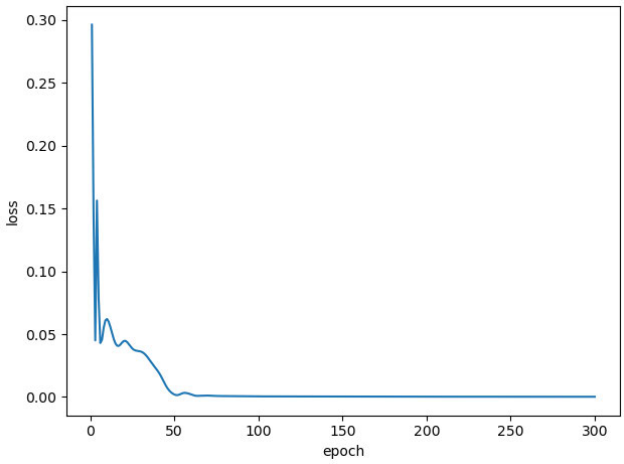


Figure 7: Training Loss plot

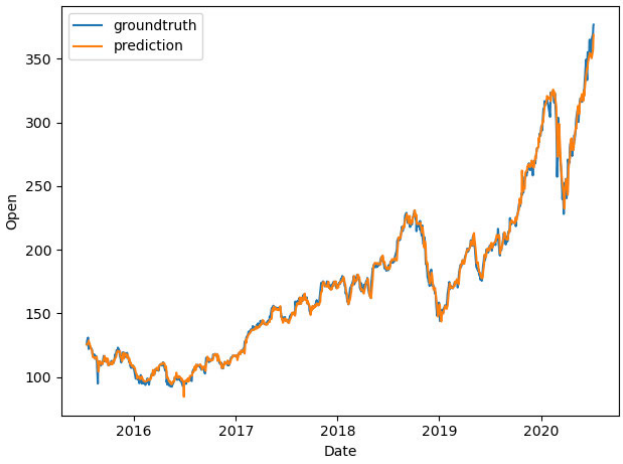


Figure 8: Prediction on training set

normally, it means that our network can work for this problem.

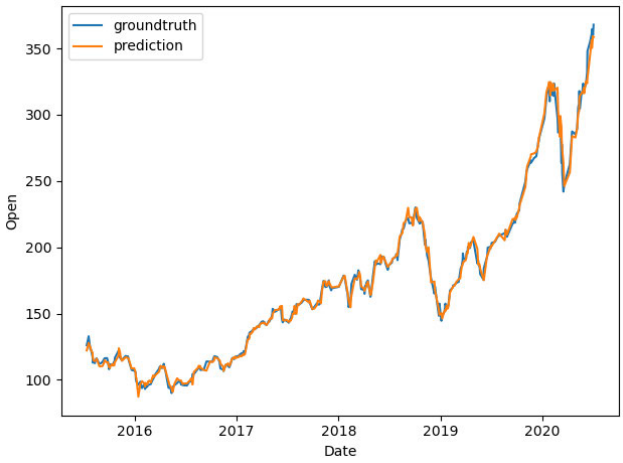


Figure 9: Prediction on testing set

If using more dates, I don't think the result would be better since the MSE is already small with the current settings. We tried using 5 days as features (didn't implement in the submission), the MSE for testing is 0.0002304534864379093, only a little bit lower.

3 Problem 3

3.1 Data Preprocessing

Step (1) Reading data.

Here we use glob to read the original dataset (*.txt file), then save the content of each *.txt file into a list, and their corresponding label (0 for negative and 1 for positive) for another list.

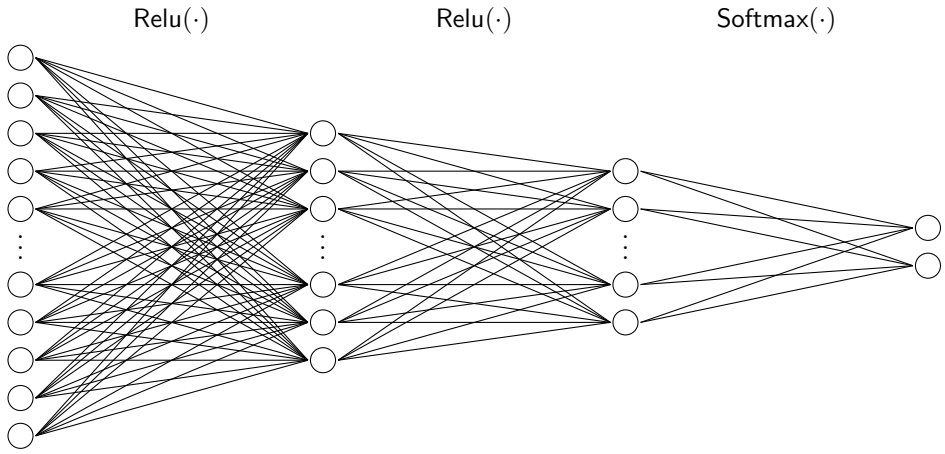
Step (2) Removing useless word and symbol in data.

For each corpus in the dataset, first separate them to words, then the symbols and stopwords from nltk package were removed, finally the remained words were combined into corpus again which separated by blank space.

Step (3) Use tf-idf for vectorization.

After above two steps, use TfidfVectorizer from sklearn to vectorizing each corpus into a (1*74047) vector.

3.2 Network Structure



Input layer $\in \mathbb{R}^{74047}$ Hidden layer $\in \mathbb{R}^{1024}$ Hidden layer $\in \mathbb{R}^{512}$ Output layer $\in \mathbb{R}^{2}$

3.3 Training

3.3.1 Training settings

learning rate = 10^{-4}, batch size = 1024, optimizer = $Adam$, loss function = $CrossEntropyLoss$

3.3.2 Training environments

GPU: NVIDIA GTX 1660 Super (6GB), CPU: AMD 3600 (4.0GHz), PyTorch 1.5.1

3.3.3 Training process

First we split the dataset into 20000 training samples and 5000 validation sample to see how many epoch needed for training. As we can see from the figure below, after 7 epochs, the validation accuracy begin increasing, therefore, 7 epochs are enough for this problem.

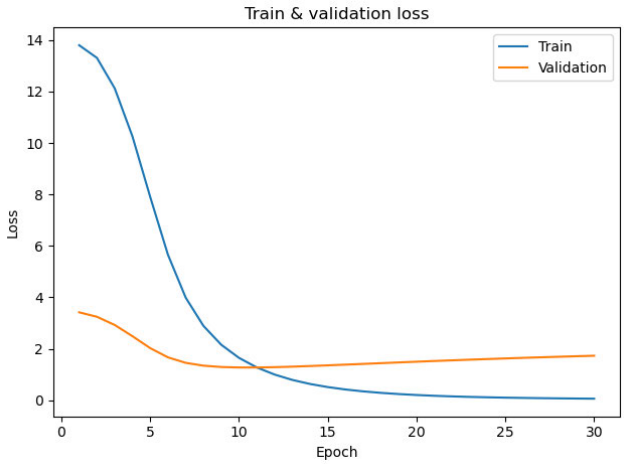


Figure 10

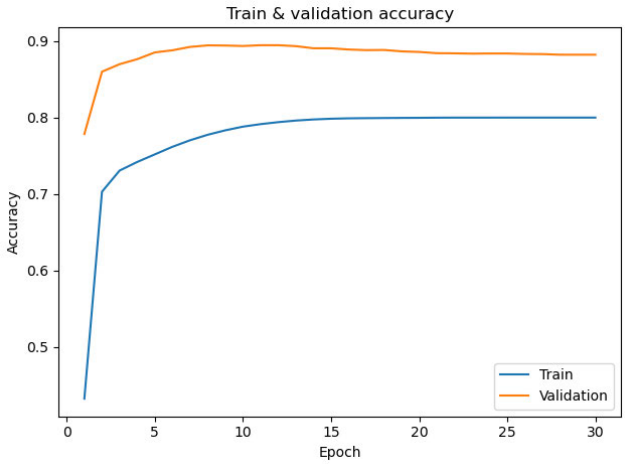


Figure 11

Then, we use the training set and set to 7 epchos for training.

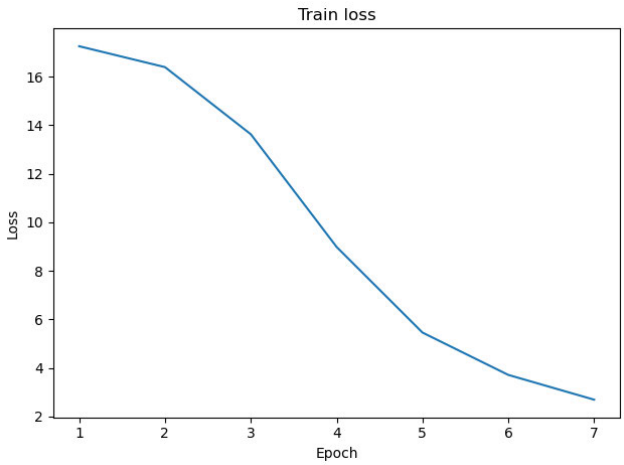


Figure 12

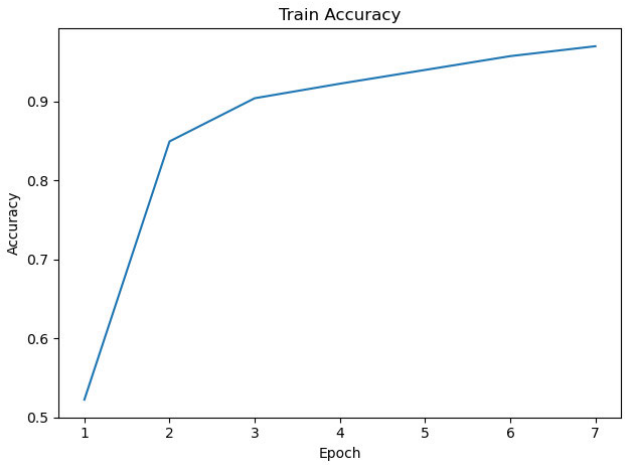


Figure 13

3.4 Testing result

Test accuracy: 0.874120, F1 score: 0.870414, Recall rate: 0.845520

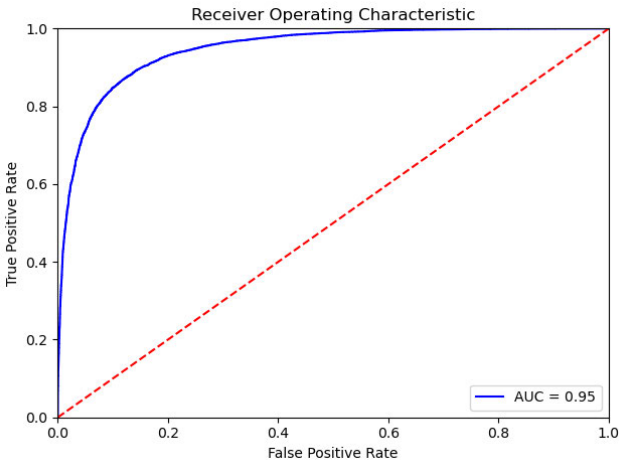


Figure 14: ROC Curve

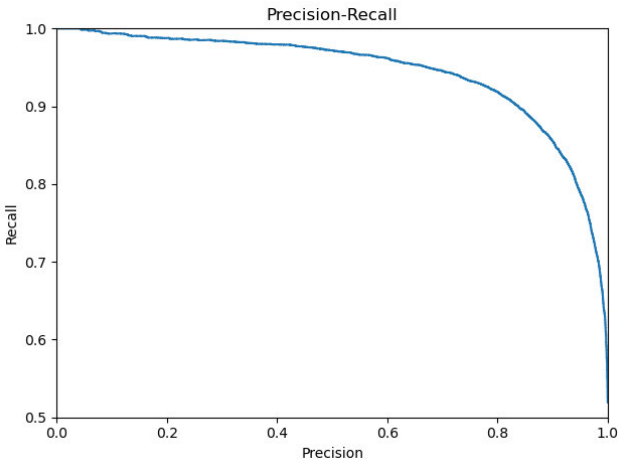


Figure 15: Precision-Recall Curve