CAP 6615 - Neural Networks - Programming Assignment 3 – Recurrent Neural Network

Zhounan Li, Huaiyue Peng, Tongjia Guo, Mingjun Yu, Zhiyun Ling

Spring Semester 2021
18 Mar 2021

1 Network parameters

In this project, we use RNN to process the time series data. During the whole experiment, we tried three kinds of model: CNN + LSTM + Dense, LSTM + Dense, and LSTM. After all the experiment that we have did, finally we chose the second model(LSTM + Dense). We use the LSTM layer extract the features in the time series. Then we use Dense layer with selu activation function to make the prediction. Before using selu, we use relu activation function because we don't want negative output. In this project, the negative output is meaningless. However, with 'relu', the result of the model before this activation func will not catch this kind of this error, because the gradient on the negative side is 0. So we use selu to replace relu to make a better prediction. The structure of our model is below.

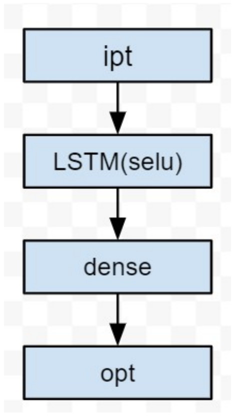


Figure 1: The structure of our model

2 Python code for models

2.1 Model with CNN, LSTM and Dense

```
def get_model1(input_length=180, normalized=False, activation=True):
    if activation:
        ipt = Input(shape=(input_length, 2))
        cnn1 = Conv1D(8, 3, 1, padding='same',
                        activation='relu')(ipt)
        if normalized:
            cnn_norm = LayerNormalization(axis=-1)(cnn1)
            lstm1 = LSTM(4, return_sequences=False,
                        activation='relu')(cnn_norm)
        else:
            lstm1 = LSTM(4, return_sequences=False,
                        activation='relu')(cnn1)
        opt = Dense(1, activation='relu')(lstm1)
        model = Model(ipt, opt)
        return model
    else:
        ipt = Input(shape=(input_length, 2))
        cnn1 = Conv1D(8, 3, 1, padding='same')(ipt)
        if normalized:
            cnn_norm = LayerNormalization(axis=-1)(cnn1)
            lstm1 = LSTM(4, return_sequences=False)(cnn_norm)
        else:
            lstm1 = LSTM(4, return_sequences=False)(cnn1)
        opt = Dense(1)(lstm1)
        model = Model(ipt, opt)
        return model
```

The reason we want to try CNN is that we assume CNN may help to reduce some noise by using convolution and also can extract some hidden features in some continuous days. However, the result is not as good as we thought. Not only the converging speed is slow, but also the predict results are not well enough. So we give up this model for the moment.

2.2 Model with LSTM and Dense

```
def get_model2(input_length=180, activation=True):
    ipt = Input(shape=(input_length, 2))
    lstm = LSTM(4, return_sequences=False,
    activation='relu')(ipt) if activation else LSTM(4,
    return_sequences=False)(ipt)
    opt = Dense(1, activation='relu')(lstm)
    model = Model(ipt, opt)
    return model
```

This is the model that seems to be the most appropriate one before we using SELU. Due to the mis-use of activation func, the prediction result is not stable.

2.3 Model with single LSTM

```
def get_model3(input_length=180):
    ipt = Input(shape=(input_length, 2))
    opt = LSTM(1, return_sequences=False)(ipt)
    model = Model(ipt, opt)
    return model
```

The reason we tried this model is that this is the simplest model, we wanted to know how it performs.

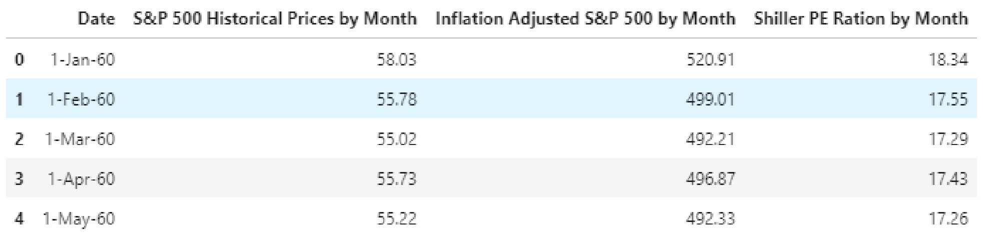
2.4 Final Model

```
ipt = Input(shape=(180, 2))
lstm = LSTM(4, return_sequences=False, activation='selu')(ipt)
opt = Dense(1)(lstm)
model_new = Model(ipt, opt)
```

This is the final model. Based on the model2, we replace the activation func on LSTM layer with 'selu'. Also we drop the activation function on the final layer. The reason we did that is we want to keep the gradient on whole range, and we don't want the prediction cut by activation func too sharply. And the result shows that this is the best model our team built.

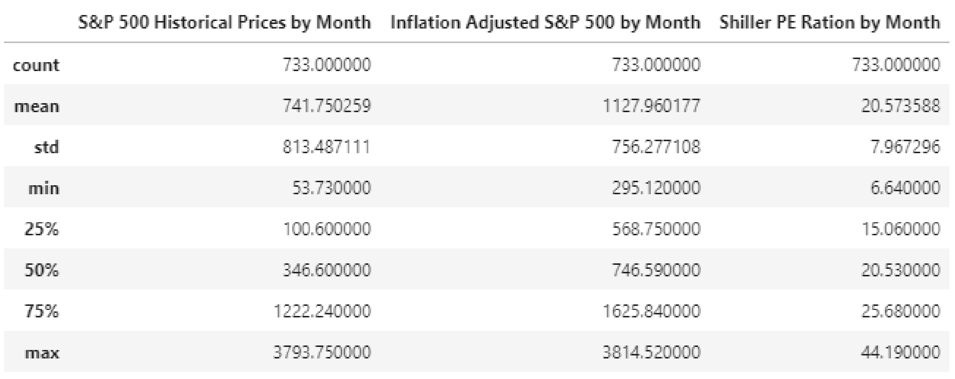
3 Training set configuration

In this project, the size of train set is 470 and the size of test set is 83.



	Date	S&P 500 Historical Prices by Month	Inflation Adjusted S&P 500 by Month	Shiller PE Ration by Month
0	1-Jan-60	58.03	520.91	18.34
1	1-Feb-60	55.78	499.01	17.55
2	1-Mar-60	55.02	492.21	17.29
3	1-Apr-60	55.73	496.87	17.43
4	1-May-60	55.22	492.33	17.26

Figure 2: The part of dataset



	S&P 500 Historical Prices by Month	Inflation Adjusted S&P 500 by Month	Shiller PE Ration by Month
count	733.000000	733.000000	733.000000
mean	741.750259	1127.960177	20.573588
std	813.487111	756.277108	7.967296
min	53.730000	295.120000	6.640000
25%	100.600000	568.750000	15.060000
50%	346.600000	746.590000	20.530000
75%	1222.240000	1625.840000	25.680000
max	3793.750000	3814.520000	44.190000

Figure 3: The summary of dataset

Our code for building dataset:

```
data_X = np.array([features[i:i+180] for i in
range(len(features)-180)])
data_Y = np.array(labels)
assert len(data_X) == len(data_Y)
data_X.shape, data_Y.shape
# split
def train_test_split(dataset_X, dataset_Y, split=0.3):
    train_len = int(len(dataset_X) * (1-split))
    train_X = dataset_X[0:train_len].reshape((-1, 180, 2))
    train_Y = dataset_Y[0:train_len]
    test_X = dataset_X[train_len:].reshape((-1, 180, 2))
    test_Y = dataset_Y[train_len:]
    return train_X, train_Y, test_X, test_Y
```

4 Model Prediction Results

4.1 Unoptimized output results

We use single LSTM layer to train our dataset and do prediction. The results are below.

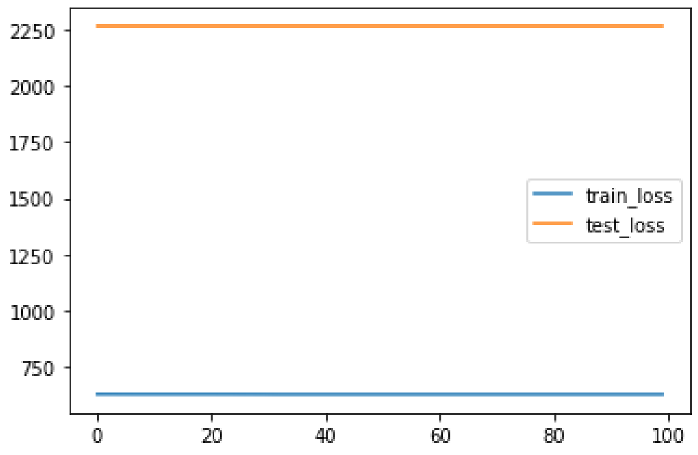


Figure 4: The loss of LSTM model

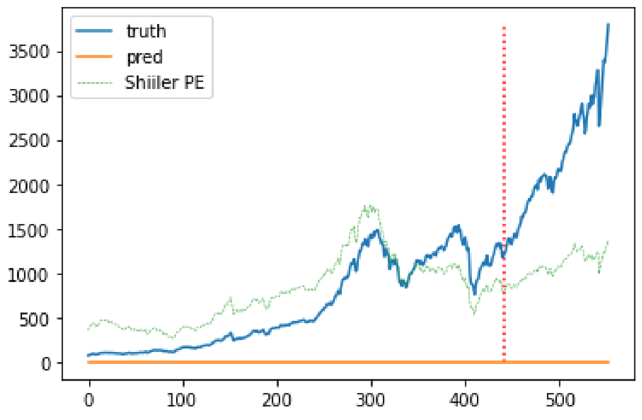


Figure 5: The prediction result of LSTM model

From the results, we can see that the single LSTM model doesn't work well on our dataset. The model is a bit simple that it doesn't learn any useful things from datasets.

4.2 Model-2 output results

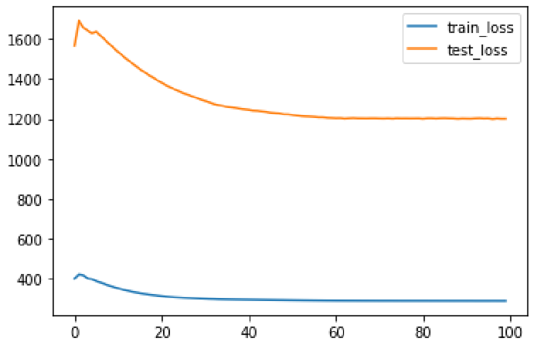


Figure 6: The loss of LSTM + Dense model

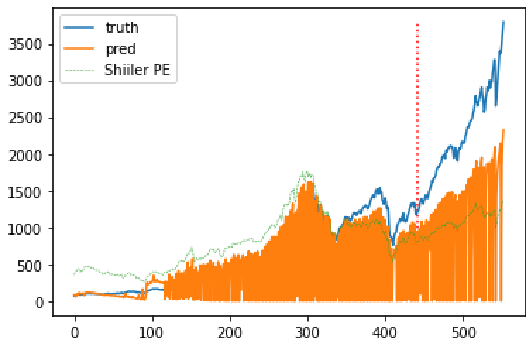


Figure 7: The prediction result of LSTM + Dense model

From results, we can see that this model performs much better than single LSTM model. The reason is if we use the model above, with 1 as hidden size, the LTSM cell can not extract enough info, and will forget lots of info during the 180 time steps. So we set the hidden layer size as 4 to extract more info, and add a dense layer to make prediction using those info. But there is still a problem: the prediction is unstable although the model has alreadt converged. The reason is about the relu activation func. It force the prediction to be larger than 0, and make those wrong prediction cannot not be trained due to the 0-gradient. So we replace relu with selu to tolerate negative value before final layer. Also we drop relu on the final layer to make the model stronger.

$$\begin{aligned} selu(x) &= x \qquad\qquad x > 0 \\ &= ae^x - a \qquad x <= 0 \end{aligned} \tag{1}$$

4.3 Final Model results

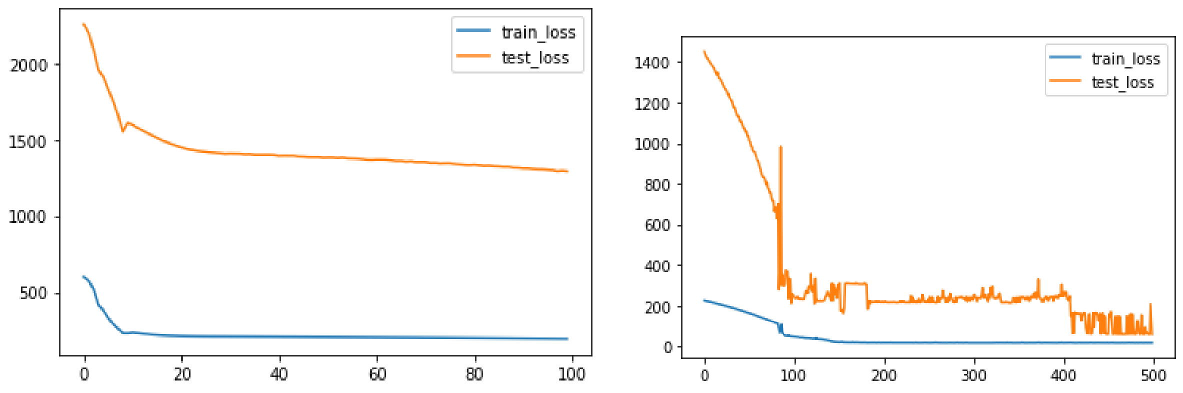


Figure 8: The training process of the final model

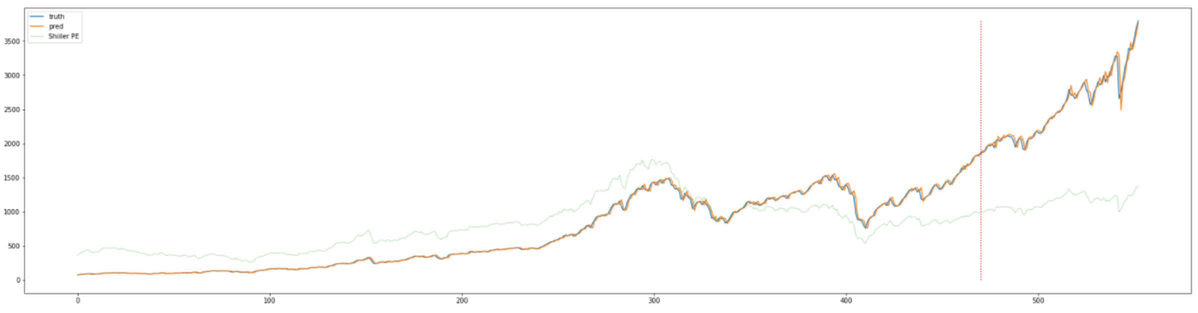


Figure 9: The prediction result of LSTM + Dense model on whole dataset

5 Code for computing noisy data and prediction error

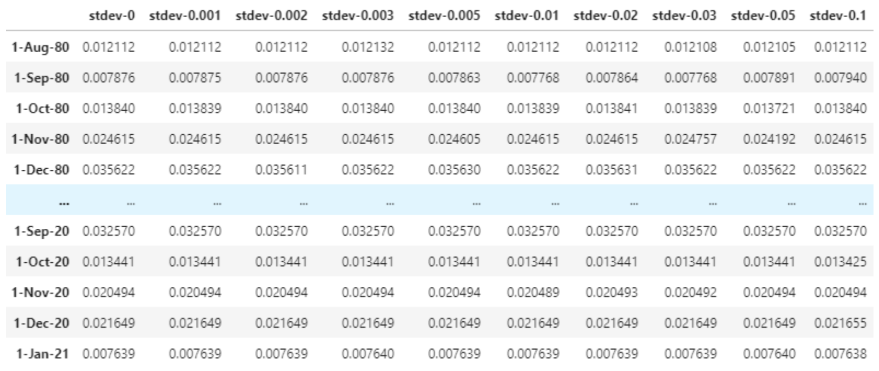
Codes of computing noisy data.

```
def add_noise(dev):
    test_data_noise_x = new_test_data_X.copy()
    noise_0 = np.random.normal(0, dev, 18)
    noise_1 = np.random.normal(0, dev, 18)
    for i in range(len(test_data_noise_x)):
        x_idx = np.arange(180)
        np.random.shuffle(x_idx)
        x_idx = x_idx[0:18]
        for j, idx in enumerate(x_idx):
            test_data_noise_x[i,idx,0] += noise_0[j]
            test_data_noise_x[i,idx,1] += noise_1[j]
    return test_data_noise_x
```

Codes of computing prediction error

```
for dev in [0, 0.001, 0.002, 0.003, 0.005, 0.01, 0.02, 0.03,
0.05, 0.1]:
    test_data_noise_x = add_noise(dev)
    y_pred = model_new.predict(test_data_noise_x).reshape(-1, )
    y_truth = new_test_data_Y
    error = abs(y_truth - y_pred)
    error_rate = [e / y for e, y in zip(error, y_truth)]
    res['stdev-'+str(dev)] = error_rate
```

The results of noise-corrupted inputs, the entire chart is too big. So we show some parts of it.



	stdev-0	stdev-0.001	stdev-0.002	stdev-0.003	stdev-0.005	stdev-0.01	stdev-0.02	stdev-0.03	stdev-0.05	stdev-0.1
1-Aug-80	0.012112	0.012112	0.012112	0.012132	0.012112	0.012112	0.012112	0.012108	0.012105	0.012112
1-Sep-80	0.007876	0.007875	0.007876	0.007876	0.007863	0.007768	0.007864	0.007768	0.007891	0.007940
1-Oct-80	0.013840	0.013839	0.013840	0.013840	0.013840	0.013839	0.013841	0.013839	0.013721	0.013840
1-Nov-80	0.024615	0.024615	0.024615	0.024615	0.024605	0.024615	0.024615	0.024757	0.024192	0.024615
1-Dec-80	0.035622	0.035622	0.035611	0.035622	0.035630	0.035622	0.035631	0.035622	0.035622	0.035622
...	...	...	...	...	...	...	...	...	...	...
1-Sep-20	0.032570	0.032570	0.032570	0.032570	0.032570	0.032570	0.032570	0.032570	0.032570	0.032570
1-Oct-20	0.013441	0.013441	0.013441	0.013441	0.013441	0.013441	0.013441	0.013441	0.013441	0.013425
1-Nov-20	0.020494	0.020494	0.020494	0.020494	0.020494	0.020489	0.020493	0.020492	0.020494	0.020494
1-Dec-20	0.021649	0.021649	0.021649	0.021649	0.021649	0.021649	0.021649	0.021649	0.021649	0.021655
1-Jan-21	0.007639	0.007639	0.007639	0.007640	0.007639	0.007639	0.007639	0.007639	0.007640	0.007638

Figure 10: The results of noise-corrupted inputs

From results, we can see most prediction-error is below 0.01, which means our model performs well on noise-corrupted data.

6 Discussion

From the above results, we know that our model performs very well on train and test dataset and even on noise-corrupted dataset. We increase the output size of LSTM layer and make more use of the former price. But there is a problem. First, we just use one LSTM layer to do predictions. Though our model performs well in this project. When the data become more complex, when the data size becomes bigger, our model will not perform well. Because it is a bit simple and can't learn enough useful things to do accurate prediction. Then we also add a CNN layer to pre-process data. But there are some problems. The CNN doesn't have any effect on the dataset in this project. This may be because CNN is good at dealing images. But it can't deal with time data very well. The train loss and test loss remains unchanged. If CNN works, our model may become more complex and cause over-fitting problem. So adding CNN layer is not a good idea.