Cryptocurrency Predict using Recurrent Neural Network(RNN) and Convolutional Neural Network(CNN)

This code is from pythonprogramming.net tutorial.

Using Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN), this code will predict the price movement of Bitcoin, Ethereum, Litecoin and Bitcoin Cash cryptocurrency data. The target output will be a binary of **0** for price increase and **1** for price decrease. Also, this code will show the performace of RNN vs CNN.

In [1]:
```
import numpy as np
import pandas as pd
import random
import time
from collections import deque
from sklearn import preprocessing
```

In [2]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, CuDNNLSTM, BatchNormalizat
from tensorflow.keras.layers import Activation, Flatten, Conv1D, MaxPooling1D
from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint
```

Data

In [3]:
```
df = pd.read_csv("../input/LTC-USD.csv", names = ['time','low','high','open','close'

df.head()
```

Out[3]:

	time	low	high	open	close	volume
0	1528968660	96.580002	96.589996	96.589996	96.580002	9.647200
1	1528968720	96.449997	96.669998	96.589996	96.660004	314.387024
2	1528968780	96.470001	96.570000	96.570000	96.570000	77.129799
3	1528968840	96.449997	96.570000	96.570000	96.500000	7.216067
4	1528968900	96.279999	96.540001	96.500000	96.389999	524.539978

In [4]:
```
df.shape
```

Out[4]:
```
(101883, 6)
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101883 entries, 0 to 101882
Data columns (total 6 columns):
time      101883 non-null int64
low       101883 non-null float64
high      101883 non-null float64
open      101883 non-null float64
close     101883 non-null float64
```

```
volume    101883 non-null float64
dtypes: float64(5), int64(1)
memory usage: 4.7 MB
```

In [6]:
```
df.columns
```

Out[6]:
```
Index(['time', 'low', 'high', 'open', 'close', 'volume'], dtype='object')
```

Parameters

A sequence lenght of 60 minutes data will be collected to be feed into the model and will predict the futere in 3 minute time.

In [7]:
```
SEQ_LEN = 60
FUTURE_PERIOD_PREDICT = 3
RATIO_TO_PREDICT = 'LTC-USD'
```

Classify Target

In [8]:
```
def classify (current, future):
    if float(future) > float(current):
        return 1
    else:
        return 0
```

Preprocessing Data

In [9]:
```
def preprocess_df(df):
    df = df.drop('future', 1)

    for col in df.columns:
        if col != 'target':
            df[col] = df[col].pct_change()
            df.dropna(inplace=True)
            df[col] = preprocessing.scale(df[col].values)
    df.dropna(inplace=True)

    sequential_data = []
    prev_days = deque(maxlen=SEQ_LEN)

    for i in df.values:
        prev_days.append([n for n in i[:-1]])
        if len(prev_days) == SEQ_LEN:
            sequential_data.append([np.array(prev_days), i[-1]])
    random.shuffle(sequential_data)

    buys = []
    sells = []

    for seq, target in sequential_data:
        if target == 0:
            sells.append([seq, target])
        elif target == 1:
            buys.append([seq, target])
    random.shuffle(buys)
    random.shuffle(sells)

    lower = min(len(buys), len(sells))
    buys = buys[:lower]
```

```
    sells = sells[:lower]

    sequential_data = buys + sells
    random.shuffle(sequential_data)

    X = []
    y = []

    for seq, target in sequential_data:
        X.append(seq)
        y.append(target)

    return np.array(X), y
```

Build the Dataframe for data training and validation

In [10]:
```
main_df = pd.DataFrame()
ratios = ['LTC-USD', 'BCH-USD', 'BTC-USD', 'ETH-USD']

for ratio in ratios:
    ratio = ratio.split('.csv')[0]
    dataset = f'../input/{ratio}.csv'
    df = pd.read_csv(dataset, names=['time','low','high','open','close','volume'])
    df.rename(columns={'close':f'{ratio}_close','volume':f'{ratio}_volume'}, inplace
    df.set_index('time', inplace=True)
    df = df[[f'{ratio}_close',f'{ratio}_volume']]

    if len(main_df) == 0:
        main_df = df
    else:
        main_df = main_df.join(df)
```

In [11]:
```
main_df.fillna(method='ffill', inplace=True)
main_df.dropna(inplace=True)
print(main_df.head())
```

```
            LTC-USD_close       ...        ETH-USD_volume
time                            ...
1528968720      96.660004       ...             26.019083
1528968780      96.570000       ...              8.449400
1528968840      96.500000       ...             26.994646
1528968900      96.389999       ...             77.355759
1528968960      96.519997       ...              7.503300

[5 rows x 8 columns]
```

In [12]:
```
main_df['future'] = main_df[f'{RATIO_TO_PREDICT}_close'].shift(-FUTURE_PERIOD_PREDIC
main_df['target'] = list(map(classify, main_df[f'{RATIO_TO_PREDICT}_close'],main_df[
main_df.dropna(inplace=True)
print(main_df.head())
```

```
            LTC-USD_close  LTC-USD_volume   ...       future  target
time                                        ...
1528968720      96.660004      314.387024   ...    96.389999       0
1528968780      96.570000       77.129799   ...    96.519997       0
1528968840      96.500000        7.216067   ...    96.440002       0
1528968900      96.389999      524.539978   ...    96.470001       1
1528968960      96.519997       16.991997   ...    96.400002       0

[5 rows x 10 columns]
```

In [13]:
```
times = sorted(main_df.index.values)
last_5pct = sorted(main_df.index.values)[-int(0.05*len(times))]
```

```
print(time)
print(last_5pct)
```

```
<module 'time' (built-in)>
1534902300
```

In [14]:
```
validation_main_df = main_df[(main_df.index >= last_5pct)]
train_main_df = main_df[(main_df.index < last_5pct)]

print(validation_main_df.head())
print(train_main_df.head())
```

```
            LTC-USD_close  LTC-USD_volume   ...       future  target
time                                        ...
1534902300      58.180000      380.746002   ...    58.279999       1
1534902360      58.259998       56.717766   ...    58.259998       0
1534902420      58.400002      702.115906   ...    58.150002       0
1534902480      58.279999      247.673599   ...    58.160000       0
1534902540      58.259998      243.763382   ...    58.189999       0

[5 rows x 10 columns]
            LTC-USD_close  LTC-USD_volume   ...       future  target
time                                        ...
1528968720      96.660004      314.387024   ...    96.389999       0
1528968780      96.570000       77.129799   ...    96.519997       0
1528968840      96.500000        7.216067   ...    96.440002       0
1528968900      96.389999      524.539978   ...    96.470001       1
1528968960      96.519997       16.991997   ...    96.400002       0

[5 rows x 10 columns]
```

In [15]:
```
train_x, train_y = preprocess_df(train_main_df)
validation_x, validation_y = preprocess_df(validation_main_df)
```

In [16]:
```
print(f"train data: {len(train_x)} validation: {len(validation_x)}")
print(f"Dont buys: {train_y.count(0)}, buys: {train_y.count(1)}")
print(f"VALIDATION Dont buys: {validation_y.count(0)}, buys: {validation_y.count(1)}
```

```
train data: 81638 validation: 4022
Dont buys: 40819, buys: 40819
VALIDATION Dont buys: 2011, buys: 2011
```

In [17]:
```
print(train_x.shape[1:])
```

```
(60, 8)
```

RNN Model

In [18]:
```
EPOCHS = 10
BATCH_SIZE = 64
NAME = f'{SEQ_LEN}-SEQ-{FUTURE_PERIOD_PREDICT}-PRED-{int(time.time())}'
```

In [19]:
```
rnn_model = Sequential()

rnn_model.add(CuDNNLSTM(128, input_shape=(train_x.shape[1:]),return_sequences=True))
rnn_model.add(Dropout(0.2))
rnn_model.add(BatchNormalization())

rnn_model.add(CuDNNLSTM(128,return_sequences=True))
rnn_model.add(Dropout(0.1))
rnn_model.add(BatchNormalization())

rnn_model.add(CuDNNLSTM(128))
rnn_model.add(Dropout(0.2))
```

```
rnn_model.add(BatchNormalization())

rnn_model.add(Dense(32, activation='relu'))
rnn_model.add(Dropout(0.2))

rnn_model.add(Dense(2, activation='softmax'))
```

In [20]:
```
opt = tf.keras.optimizers.Adam(lr=0.001, decay=1e-6)
```

In [21]:
```
rnn_model.compile(loss='sparse_categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

In [22]:
```
tensorboard = TensorBoard(log_dir='../{}'.format(NAME))

filepath = 'RNN_Final-{epoch:02d}-{val_acc:.3f}'
checkpoint = ModelCheckpoint('../{}.model'.format(filepath, monitor='val-acc', verbo
```

In [23]:
```
history = rnn_model.fit(train_x, train_y,
                    batch_size=BATCH_SIZE,
                    epochs=EPOCHS,
                    validation_data=(validation_x, validation_y),
                    callbacks = [tensorboard, checkpoint])
```

```
Train on 81638 samples, validate on 4022 samples
Epoch 1/10
81638/81638 [==============================] - 25s 308us/step - loss: 0.7055 - acc:
0.5261 - val_loss: 0.6857 - val_acc: 0.5587
Epoch 2/10
81638/81638 [==============================] - 22s 273us/step - loss: 0.6853 - acc:
0.5504 - val_loss: 0.6767 - val_acc: 0.5743
Epoch 3/10
81638/81638 [==============================] - 22s 273us/step - loss: 0.6810 - acc:
0.5638 - val_loss: 0.6776 - val_acc: 0.5845
Epoch 4/10
81638/81638 [==============================] - 22s 275us/step - loss: 0.6798 - acc:
0.5668 - val_loss: 0.6777 - val_acc: 0.5748
Epoch 5/10
81638/81638 [==============================] - 23s 281us/step - loss: 0.6781 - acc:
0.5716 - val_loss: 0.6739 - val_acc: 0.5801
Epoch 6/10
81638/81638 [==============================] - 22s 275us/step - loss: 0.6756 - acc:
0.5771 - val_loss: 0.6789 - val_acc: 0.5671
Epoch 7/10
81638/81638 [==============================] - 22s 274us/step - loss: 0.6719 - acc:
0.5853 - val_loss: 0.6773 - val_acc: 0.5636
Epoch 8/10
81638/81638 [==============================] - 23s 287us/step - loss: 0.6665 - acc:
0.5927 - val_loss: 0.6797 - val_acc: 0.5709
Epoch 9/10
81638/81638 [==============================] - 23s 284us/step - loss: 0.6606 - acc:
0.6036 - val_loss: 0.6852 - val_acc: 0.5656
Epoch 10/10
81638/81638 [==============================] - 23s 276us/step - loss: 0.6508 - acc:
0.6173 - val_loss: 0.6795 - val_acc: 0.5781
```

In [24]:
```
rnn_score = rnn_model.evaluate(validation_x, validation_y, verbose=0)
print('Test loss:', rnn_score[0])
print('Test accuracy:', rnn_score[1])
```

```
Test loss: 0.6795175345841479
Test accuracy: 0.5780706117249199
```

CNN Model

In [25]:
```
cnn_model = Sequential()

cnn_model.add(Conv1D(128,3,input_shape=(train_x.shape[1:])))
cnn_model.add(Activation('relu'))
cnn_model.add(Dropout(0.2))
cnn_model.add(MaxPooling1D(pool_size=2))

cnn_model.add(Conv1D(128,3))
cnn_model.add(Activation('relu'))
cnn_model.add(Dropout(0.2))
cnn_model.add(MaxPooling1D(pool_size=2))

cnn_model.add(Conv1D(128,3))
cnn_model.add(Activation('relu'))
cnn_model.add(Dropout(0.2))
cnn_model.add(MaxPooling1D(pool_size=2))

cnn_model.add(Flatten())
cnn_model.add(Dense(32))

cnn_model.add(Dense(2, activation='softmax'))

cnn_model.compile(loss='sparse_categorical_crossentropy',
                  optimizer=opt,
                  metrics=['accuracy'])

cnn_history = cnn_model.fit(train_x, train_y,
                    batch_size=BATCH_SIZE,
                    epochs=EPOCHS,
                    validation_data=(validation_x, validation_y))
```

```
Train on 81638 samples, validate on 4022 samples
Epoch 1/10
81638/81638 [==============================] - 7s 82us/step - loss: 0.6979 - acc: 0.
5159 - val_loss: 0.6930 - val_acc: 0.5109
Epoch 2/10
81638/81638 [==============================] - 6s 74us/step - loss: 0.6918 - acc: 0.
5214 - val_loss: 0.6926 - val_acc: 0.5139
Epoch 3/10
81638/81638 [==============================] - 6s 74us/step - loss: 0.6913 - acc: 0.
5266 - val_loss: 0.6932 - val_acc: 0.5102
Epoch 4/10
81638/81638 [==============================] - 7s 83us/step - loss: 0.6912 - acc: 0.
5260 - val_loss: 0.6925 - val_acc: 0.5206
Epoch 5/10
81638/81638 [==============================] - 6s 78us/step - loss: 0.6910 - acc: 0.
5264 - val_loss: 0.6933 - val_acc: 0.5045
Epoch 6/10
81638/81638 [==============================] - 6s 74us/step - loss: 0.6908 - acc: 0.
5276 - val_loss: 0.6939 - val_acc: 0.5035
Epoch 7/10
81638/81638 [==============================] - 6s 73us/step - loss: 0.6905 - acc: 0.
5302 - val_loss: 0.6933 - val_acc: 0.5042
Epoch 8/10
81638/81638 [==============================] - 6s 74us/step - loss: 0.6901 - acc: 0.
5293 - val_loss: 0.6943 - val_acc: 0.5015
Epoch 9/10
81638/81638 [==============================] - 6s 74us/step - loss: 0.6897 - acc: 0.
5316 - val_loss: 0.6968 - val_acc: 0.4978
Epoch 10/10
81638/81638 [==============================] - 6s 74us/step - loss: 0.6891 - acc: 0.
5360 - val_loss: 0.6952 - val_acc: 0.5050
```

In [26]:
```
cnn_score = cnn_model.evaluate(validation_x, validation_y, verbose=0)
print('Test loss:', cnn_score[0])
print('Test accuracy:', cnn_score[1])
```

```
Test loss: 0.6952022758364855
Test accuracy: 0.5049726504226753
```

In [27]: