00_build_dataset

September 29, 2021

1 Train a Deep NN to predict Asset Price movements

1.1 Setup Docker for GPU acceleration

docker run -it -p 8889:8888 -v /path/to/machine-learning-for-trading/16_convolutions_neural_ne
--name tensorflow tensorflow/tensorflow:latest-gpu-py3 bash

1.2 Imports & Settings

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
import os
from pathlib import Path
from importlib import reload
from joblib import dump, load

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪StratifiedKFold
from sklearn.metrics import roc_auc_score

import tensorflow as tf
from keras.models import Sequential
from keras import backend as K
from keras.wrappers.scikit_learn import KerasClassifier
from keras.layers import Dense, Dropout, Activation
from keras.models import load_model
from keras.callbacks import Callback, EarlyStopping, TensorBoard,␣
↪ModelCheckpoint
```

```
Using TensorFlow backend.
```

[3]:
```
np.random.seed(42)
```

1.3 Build Dataset

We load the Quandl adjusted stock price data:

[4]:
```
prices = (pd.read_hdf('../data/assets.h5', 'quandl/wiki/prices')
          .adj_close
          .unstack().loc['2007':])
prices.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2896 entries, 2007-01-01 to 2018-03-27
Columns: 3199 entries, A to ZUMZ
dtypes: float64(3199)
memory usage: 70.7 MB
```

1.3.1 Resample to weekly frequency

We start by generating weekly returns for close to 2,500 stocks without missing data for the 2008-17 period, as follows:

[5]:
```
returns = (prices
           .resample('W')
           .last()
           .pct_change()
           .loc['2008': '2017']
           .dropna(axis=1)
           .sort_index(ascending=False))
returns.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 522 entries, 2017-12-31 to 2008-01-06
Freq: -1W-SUN
Columns: 2489 entries, A to ZUMZ
dtypes: float64(2489)
memory usage: 9.9 MB
```

[6]:
```
returns.head().append(returns.tail())
```

[6]:
```
ticker             A       AAL       AAN      AAON       AAP      AAPL  \
date
2017-12-31 -0.005642 -0.010648 -0.010184 -0.001361 -0.008553 -0.033027
2017-12-24 -0.003846  0.029965  0.090171  0.044034 -0.001490  0.006557
2017-12-17  0.003413  0.000784 -0.052591 -0.014006  0.003888  0.026569
2017-12-10 -0.019071  0.041012 -0.005359 -0.017882  0.010375 -0.009822
2017-12-03 -0.009660  0.009267  0.105501  0.013947  0.112630 -0.022404
2008-02-03  0.038265  0.252238  0.002941  0.095182  0.097833  0.028767
2008-01-27 -0.013963 -0.048762  0.191310  0.071788  0.043997 -0.194286
2008-01-20 -0.065000  0.086627 -0.080541 -0.054762 -0.007176 -0.065609
2008-01-13  0.035375 -0.041902 -0.037818 -0.046538 -0.101486 -0.040878
```

```
2008-01-06 -0.072553 -0.156356 -0.068707 -0.133301 -0.065496 -0.098984

ticker          AAWW      ABAX       ABC      ABCB  …      ZEUS      ZIGO  \
date                                                …
2017-12-31 -0.024938 -0.001814 -0.006922 -0.019329  … -0.029797  0.000000
2017-12-24  0.046087  0.032681 -0.007620  0.017598  …  0.032153  0.000000
2017-12-17  0.004367  0.008396  0.074625  0.026567  …  0.036715  0.000000
2017-12-10 -0.028014 -0.010386  0.020600 -0.054271  … -0.002410  0.000000
2017-12-03  0.073838 -0.028456  0.045796  0.024717  …  0.065742  0.000000
2008-02-03  0.006245 -0.078058  0.036913  0.083217  …  0.137066  0.127561
2008-01-27 -0.008984 -0.090807 -0.034771  0.054572  …  0.018349 -0.026292
2008-01-20  0.015818 -0.019721 -0.015219 -0.044397  …  0.040573  0.010999
2008-01-13 -0.052095  0.097385  0.080137 -0.017313  … -0.054176 -0.047993
2008-01-06 -0.029478 -0.098374 -0.037363 -0.132733  … -0.027290 -0.075806

ticker          ZINC      ZION      ZIOP      ZIXI       ZLC       ZMH  \
date
2017-12-31  0.000000 -0.009741  0.022222 -0.015730  0.000000  0.000000
2017-12-24  0.000000  0.026395 -0.068966 -0.024123  0.000000  0.000000
2017-12-17  0.000000 -0.018064 -0.018059  0.075472  0.000000  0.000000
2017-12-10  0.000000  0.016973 -0.015556 -0.055679  0.000000  0.000000
2017-12-03  0.000000  0.080475  0.014656 -0.006637  0.000000  0.000000
2008-02-03  0.286550  0.167722 -0.087879  0.069364  0.171949  0.193189
2008-01-27 -0.046975  0.136418 -0.003021  0.145695  0.042164 -0.014553
2008-01-20 -0.167109 -0.051614 -0.054286 -0.124638  0.037172 -0.037312
2008-01-13 -0.102381  0.037264 -0.022346 -0.172662  0.011799  0.051880
2008-01-06 -0.004739 -0.081058  0.101538 -0.143737 -0.134100  0.000752

ticker           ZQK      ZUMZ
date
2017-12-31  0.000000 -0.029138
2017-12-24  0.000000  0.067164
2017-12-17  0.000000 -0.051887
2017-12-10  0.000000  0.062657
2017-12-03  0.000000  0.047244
2008-02-03  0.127811  0.149083
2008-01-27  0.141892  0.118666
2008-01-20 -0.030144 -0.076969
2008-01-13  0.018692 -0.094249
2008-01-06 -0.133102 -0.269012

[10 rows x 2489 columns]
```

1.3.2 Create & stack 52-week sequences

We'll use 52-week sequences, which we'll create in a stacked format:

[7]:
```
n = len(returns)
T = 52 # weeks
tcols = list(range(T))
```

[8]:
```
data = pd.DataFrame()
for i in range(n-T-1):
    if i % 50 == 0:
        print(i, end=' ', flush=True)
    df = returns.iloc[i:i+T+1]
    data = pd.concat([data, (df
                             .reset_index(drop=True)
                             .transpose()
                             .reset_index()
                             .assign(year=df.index[0].year,
                                     month=df.index[0].month))],
                     ignore_index=True)
data.info()
```

```
0 50 100 150 200 250 300 350 400 450 <class 'pandas.core.frame.DataFrame'>
RangeIndex: 1244500 entries, 0 to 1244499
Data columns (total 25 columns):
ticker    1244500 non-null object
0         1244500 non-null float64
1         1244500 non-null float64
2         1244500 non-null float64
3         1244500 non-null float64
4         1244500 non-null float64
5         1244500 non-null float64
6         1244500 non-null float64
7         1244500 non-null float64
8         1244500 non-null float64
9         1244500 non-null float64
10        1244500 non-null float64
11        1244500 non-null float64
12        1244500 non-null float64
13        1244500 non-null float64
14        1244500 non-null float64
15        1244500 non-null float64
16        1244500 non-null float64
17        1244500 non-null float64
18        1244500 non-null float64
19        1244500 non-null float64
20        1244500 non-null float64
21        1244500 non-null float64
year      1244500 non-null int64
month     1244500 non-null int64
dtypes: float64(22), int64(2), object(1)
```

```
memory usage: 237.4+ MB
```

1.3.3 Create categorical variables

We create dummy variables for different time periods, namely months and years:

[9]:
```
data[tcols] = (data[tcols].apply(lambda x: x.clip(lower=x.quantile(.01),
                                                  upper=x.quantile(.99))))
data.ticker = pd.factorize(data.ticker)[0]
data['label'] = (data[0] > 0).astype(int)
data['date'] = pd.to_datetime(data.assign(day=1)[['year', 'month', 'day']])
data = pd.get_dummies((data.drop(0, axis=1)
                       .set_index('date')
                       .apply(pd.to_numeric)),
                      columns=['year', 'month']).sort_index()
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1244500 entries, 2008-06-01 to 2017-12-01
Data columns (total 45 columns):
ticker       1244500 non-null int64
1            1244500 non-null float64
2            1244500 non-null float64
3            1244500 non-null float64
4            1244500 non-null float64
5            1244500 non-null float64
6            1244500 non-null float64
7            1244500 non-null float64
8            1244500 non-null float64
9            1244500 non-null float64
10           1244500 non-null float64
11           1244500 non-null float64
12           1244500 non-null float64
13           1244500 non-null float64
14           1244500 non-null float64
15           1244500 non-null float64
16           1244500 non-null float64
17           1244500 non-null float64
18           1244500 non-null float64
19           1244500 non-null float64
20           1244500 non-null float64
21           1244500 non-null float64
label        1244500 non-null int64
year_2008    1244500 non-null uint8
year_2009    1244500 non-null uint8
year_2010    1244500 non-null uint8
year_2011    1244500 non-null uint8
year_2012    1244500 non-null uint8
```

```
year_2013    1244500 non-null uint8
year_2014    1244500 non-null uint8
year_2015    1244500 non-null uint8
year_2016    1244500 non-null uint8
year_2017    1244500 non-null uint8
month_1      1244500 non-null uint8
month_2      1244500 non-null uint8
month_3      1244500 non-null uint8
month_4      1244500 non-null uint8
month_5      1244500 non-null uint8
month_6      1244500 non-null uint8
month_7      1244500 non-null uint8
month_8      1244500 non-null uint8
month_9      1244500 non-null uint8
month_10     1244500 non-null uint8
month_11     1244500 non-null uint8
month_12     1244500 non-null uint8
dtypes: float64(21), int64(2), uint8(22)
memory usage: 254.0 MB
```

[10]:
```
data.to_hdf('data.h5', 'returns_daily')
```

[11]:
```
data.shape
```

[11]:
```
(1244500, 45)
```