Stocks ML

Release 0.1b

Ryan Raba

Feb 20, 2021

CONTENTS

Apply artificial intelligence to the stock market. StocksML can build an entire trading strategy from scratch and simulate the performance of that strategy against any specified benchmark.

CHAPTER

ONE

API

StocksML modules

1.1 stocksml.data

FetchData(*symbols*, *apikey*, *start=None*, *stop=None*, *path=None*, *append=True*)

Download symbol data from iex using provided api key, counts against quota

Parameters

- **symbols** (*list of str*) – list of ticker symbols to retrieve
- **apikey** (*str*) – api token of the iex account to use
- **start** (*str*) – start date of historical prices to retrieve. Format is yyyy-mm-dd. Default None uses current date
- **stop** (*str*) – stop date of historical prices to retrieve. Format is yyyy-mm-dd. Default None uses current date
- **path** (*str*) – path of folder to place downloaded data. Default None uses current directory
- **append** (*bool*) – append new data to existing file or create if missing. Duplicate dates ignored. False will overwrite file. Default True

LoadData(*symbols=None*, *path=None*)

Load price data from CSV files

Parameters

- **symbols** (*list of str*) – list of ticker symbol files to load. Files should be in the form of symbol.csv. Default None loads all files in provided directory.
- **path** (*str*) – path to symbol data files. Default None uses included demonstration data folder location

Returns symbol dataframe

Return type pandas.DataFrame

BuildData(*sdf*)

Transform price data from symbol dataframe to training feature set

Parameters **sdf** (*pandas.DataFrame*) – symbol dataframe

Returns feature dataframe

Return type pandas.DataFrame

1.2 `stocksml.model`

`BuildModel` (*fdf*, *choices*, *layers=[('rnn', 32), ('dnn', 64), ('dnn', 32)]*, *depth=5*, *count=2*)
Build a model with the given structure

Parameters

- **`fdf`** (*`pandas.DataFrame`*) – feature dataframe
- **`choices`** (*`int`*) – number of ticker symbols model can choose between
- **`layers`** (*`list of tuples`*) – list of tuples defining structure of model. Each tuple is (layer, size) where layer can be 'dnn', 'cnn', 'lstm', 'rnn', or 'drop'. Default is a 3-layer model with [('rnn',32),('dnn',64),('dnn',32)]
- **`depth`** (*`int`*) – depth of time dimension for recurrent and convolutional networks (rnn, cnn, lstm). Ignored if using dnn only. Default is 5.
- **`count`** (*`int`*) – number of models to build. Default is 2

Returns list of keras Models built, compiled and ready for training along with the appropriate data array for training

Return type list of keras.Model, numpy.ndarray

`LearnStrategy` (*models*, *sdf*, *dx*, *symbols*, *baseline=None*, *days=5*, *maxiter=1000*, *notebook=False*)
Learn a trading strategy by training models against provided data

Parameters

- **`models`** (*`list of keras.Model`*) – list of prebuilt models to train
- **`sdf`** (*`pandas.DataFrame`*) – symbol dataframe with price information
- **`dx`** (*`numpy.array`*) – vectorized training data
- **`symbols`** (*`list of str`*) – list of ticker symbols available to the trading strategy. Must all be contained in sdf
- **`baseline`** (*`str`*) – ticker symbol to use for baselining of trading strategy. Default None performs no baseline
- **`days`** (*`int`*) – number of days to use for trading strategy. Default is 5
- **`maxiter`** (*`int`*) – maximum number of training iterations. Default is 1000
- **`notebook`** (*`bool`*) – configures live plots for running in a Jupyter notebook. Default is False

`ExamineStrategy` (*model*, *sdf*, *dx*, *symbols*, *start_date*, *days=5*, *baseline=None*)
Explore a strategy learned by a model

Parameters

- **`model`** (*`keras.Model`*) – trained model to execute strategy with
- **`sdf`** (*`pandas.DataFrame`*) – symbol dataframe with price information
- **`dx`** (*`numpy.array`*) – vectorized training data
- **`symbols`** (*`list of str`*) – list of ticker symbols available to the trading strategy. Must all be contained in sdf
- **`start_date`** (*`str`*) – date to start trading strategy on. yyyy-mm-dd format
- **`days`** (*`int`*) – number of days to run strategy for. Default is 5

- **baseline** (*str*) – ticker symbol to use for baselining of trading strategy. Default None performs no baseline

Demo (*notebook=False*)
Demonstration of how to use this package

Parameters **notebook** (*bool*) – set live plots for running properly in Jupyter notebooks. Default is False

1.3 stocksml.trade

EvaluateChoices (*sdf*, *symbols*, *dates*, *choices*, *baseline=None*)
Evaluate trading strategy choices

Parameters

- **sdf** (*pandas.DataFrame*) – symbol dataframe with price information
- **symbols** (*list of str*) – list of symbol tickers corresponding to the symbol enum in choices
- **dates** (*list of str*) – dates corresponding to choices, should match subset of pdf index values
- **choices** (*list of tuples*) – tuple of (action, symbol enum, limit) for each day. action is an enum of range 0-4 where [buy_limit, buy_sell, hold, sell_limit, sell_buy]. limit is the percent over/under open price (range -1 to 1)
- **baseline** (*str*) – ticker symbol to use for baseline buy-hold strategy. Default None will not compute a baseline (returns 0)

Returns performance of choices and baseline as a fraction of initial cash and ledger log of trades

Return type float, float, str

Open in Colab: https://colab.research.google.com/github/ryanraba/stocksml/blob/master/docs/quick_start.ipynb

CHAPTER

TWO

INSTALLATION

Normal installation from the command line (assuming you have Python3 installed)

```
$ python3 -m venv venv
$ source venv/bin/activate
(venv) $ pip install stocksml
```

From a Jupyter notebook environment such as Google Colab

```
!pip install stocksml
```

CHAPTER

THREE

QUICK START

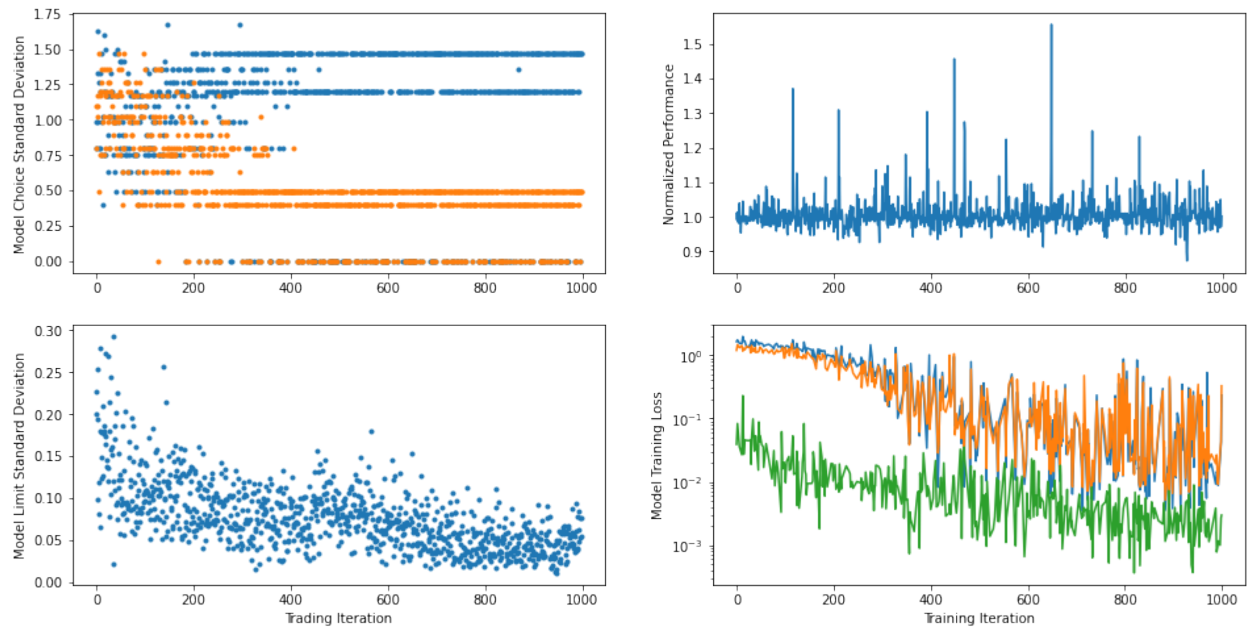
Quick demonstration using included sample data sources.

[1]:
```
import os
os.system("pip install stocksml")
print('installed stocksml')
```

```
installed stocksml
```

[2]:
```
from stocksml import Demo

Demo(notebook=True)
```




```
2021-02-01   buy market order for   24 shares of vixm at    0.0  ->  bought   24␣
↪shares at   41.6  ($0.9, $1000.0,   43.2,   41.6,   41.6,   41.9 )
2021-02-01  sell  limit order for   24 shares of vixm at   40.9  ->    sold   24␣
↪shares at   41.6  ($1000.0, $1000.0,   43.2,   41.6,   41.6,   41.9 )
2021-02-02   buy market order for   24 shares of vixm at    0.0  ->  bought   24␣
↪shares at   41.2  ($11.4, $1000.0,   41.3,   40.6,   41.2,   40.7 )
2021-02-02  sell  limit order for   24 shares of vixm at   42.6
2021-02-03  sell market order for   24 shares of vixm at    0.0  ->    sold   24␣
↪shares at   40.2  ($977.4, $977.4,   40.5,   39.9,   40.2,   39.9 )
```

(continues on next page)

(continued from previous page)

```
2021-02-03   buy market order for   24 shares of vixm at    0.0  -> bought   24␣
↪shares at   40.2  ($11.4, $977.4,   40.5,   39.9,   40.2,   39.9 )
2021-02-03  sell  limit order for   24 shares of vixm at   43.4
2021-02-04  sell market order for   24 shares of vixm at    0.0  ->   sold   24␣
↪shares at   39.5  ($959.2, $959.2,   39.8,   39.2,   39.5,   39.7 )
2021-02-04   buy market order for   24 shares of vixm at    0.0  -> bought   24␣
↪shares at   39.5  ($11.4, $959.2,   39.8,   39.2,   39.5,   39.7 )
2021-02-04  sell  limit order for   24 shares of vixm at   40.4
2021-02-05  sell market order for   24 shares of vixm at    0.0  ->   sold   24␣
↪shares at   39.4  ($958.0, $958.0,   40.0,   39.4,   39.4,   39.9 )
2021-02-05   buy market order for   24 shares of vixm at    0.0  -> bought   24␣
↪shares at   39.4  ($11.4, $958.0,   40.0,   39.4,   39.4,   39.9 )
2021-02-05  sell  limit order for   24 shares of vixm at   40.6
---------- liquidate ----------
2021-02-05  sell market order for   24 shares of vixm at    0.0  ->   sold   24␣
↪shares at   39.9  ($968.1, $968.1,   40.0,   39.4,   39.4,   39.9 )
---------- result = $968.1 at 0.932 of baseline ----------
```

3.1 Load Data

[3]:

```
from stocksml import LoadData, BuildData

# load symbols and build a symbol dataframe
sdf, symbols = LoadData(symbols=['SPY','BND'])

# convert symbol dataframe to a feature dataframe
fdf = BuildData(sdf)

fdf.head()
```

```
building BND data...
building SPY data...
```

[3]:

```
                bnd0      bnd1      bnd2  ...      spy2      spy3      spy4
date                                      ...
2017-01-03 -0.001654 -0.000511 -0.001190  ... -0.003938 -0.003082  0.018743
2017-01-04  0.009508  0.018398  0.053818  ...  0.013076  0.026660  0.391944
2017-01-05  0.109609  0.010800  0.025270  ...  0.015081 -0.007054  0.026528
2017-01-06 -0.043183  0.003252  0.021451  ...  0.003648  0.014804  0.152411
2017-01-09  0.012214  0.010019  0.011997  ...  0.007136 -0.019585 -0.380552

[5 rows x 10 columns]
```

3.2 Build a Model

Define a model and create a set of 2 or more with corresponding training data.

[4]:
```
from stocksml import BuildModel

models, dx = BuildModel(fdf, len(symbols), layers=[('rnn',32),('dnn',64),('dnn',32)],␣
↪count=2)

models[0].summary()
```

```
Model: "model"
__________________________________________________________________________________________
↪______________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================================================
input (InputLayer)              [(None, 5, 10)]      0
__________________________________________________________________________________________
↪______________
rnn_0 (SimpleRNN)               (None, 32)           1376        input[0][0]
__________________________________________________________________________________________
↪______________
dnn_1 (Dense)                   (None, 64)           2112        rnn_0[0][0]
__________________________________________________________________________________________
↪______________
dnn_2 (Dense)                   (None, 32)           2080        dnn_1[0][0]
__________________________________________________________________________________________
↪______________
action (Dense)                  (None, 5)            165         dnn_2[0][0]
__________________________________________________________________________________________
↪______________
symbol (Dense)                  (None, 2)            66          dnn_2[0][0]
__________________________________________________________________________________________
↪______________
limit (Dense)                   (None, 1)            33          dnn_2[0][0]
==========================================================================================================
Total params: 5,832
Trainable params: 5,832
Non-trainable params: 0
__________________________________________________________________________________________
↪______________
```

3.3 Learn a Strategy

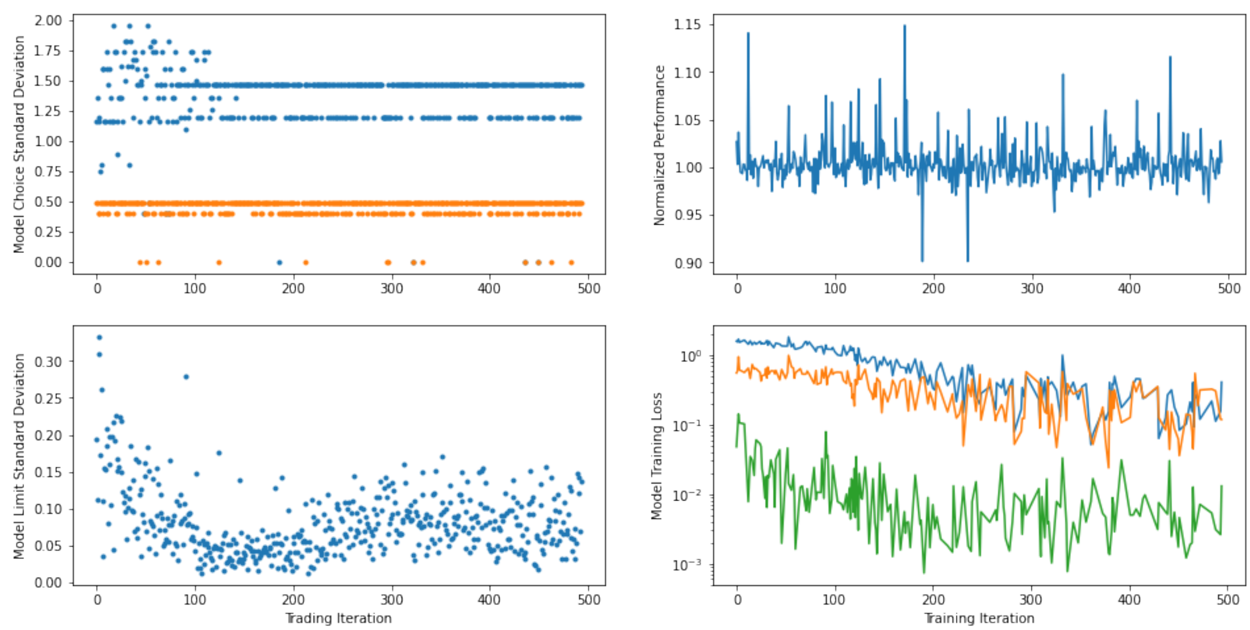
After creating a set of adversarial models and the corresponding training data formatted for them, StocksML is ready to learn a new trading strategy. This is done in an unsupervised manner, meaning no truth data is provided.

The algorithm begins with each model making random guesses. When one model successfully guesses a sequence of trades that results in superior performance (i.e. makes money or beats a benchmark), that model's strategy is "learned" by the unsuccessful model. This continues for a set period of iterations or until it appears that the models are no longer learning anything useful.

The `LearnStrategy` function displays a live plot of various metrics to illustrate the learning process and help inform when a good stopping point might be.

[5]:
```
from stocksml import LearnStrategy

LearnStrategy(models, sdf, dx, symbols, 'SPY', 5, 500, True)
```



3.4 Examine the Strategy

Once a trading strategy has been learned, it can be applied to different points in time across the available market data to see what it does and how it performs.

To avoid overfitting, it would be wise to examine strategy performance on data that wasn't used for training.

[6]:
```
from stocksml import ExamineStrategy

ExamineStrategy(models[0], sdf, dx, symbols, '2021-02-01', days=5, baseline='SPY')
```

```
2021-02-01   buy market order for    2 shares of  spy at    0.0  ->  bought    2␣
↪shares at  376.2  ($247.5, $1000.0,  377.3,  370.4,  373.7,  376.2 )
2021-02-02  sell  limit order for    2 shares of  spy at  397.7
2021-02-03  sell market order for    2 shares of  spy at    0.0  ->    sold    2␣
↪shares at  382.4  ($1012.4, $1012.4,  383.7,  380.5,  382.4,  381.9 )
2021-02-03   buy market order for    2 shares of  spy at    0.0  ->  bought    2␣
↪shares at  382.4  ($247.5, $1012.4,  383.7,  380.5,  382.4,  381.9 )
2021-02-03  sell  limit order for    2 shares of  spy at  390.0
2021-02-04  sell market order for    2 shares of  spy at    0.0  ->    sold    2␣
↪shares at  383.0  ($1013.5, $1013.5,  386.2,  382.0,  383.0,  386.2 )
2021-02-04   buy market order for    2 shares of  spy at    0.0  ->  bought    2␣
↪shares at  383.0  ($247.5, $1013.5,  386.2,  382.0,  383.0,  386.2 )
2021-02-04  sell  limit order for    2 shares of  spy at  389.9
2021-02-05  sell  limit order for    2 shares of  spy at  346.2  ->    sold    2␣
↪shares at  388.2  ($1023.9, $1023.9,  388.5,  386.1,  388.2,  387.7 )
2021-02-05   buy market order for   11 shares of  bnd at    0.0  ->  bought   11␣
↪shares at   87.0  ($67.4, $1023.9,   87.1,   87.0,   87.1,   87.0 )
```

(continues on next page)

(continued from previous page)

```
---------- liquidate ----------
2021-02-05  sell market order for   11 shares of  bnd at    0.0  ->    sold   11␣
↪shares at   87.0  ($1023.9, $1023.9,   87.1,   87.0,   87.1,   87.0 )
---------- result = $1023.9 at 0.986 of baseline ----------
```

Open in Colab: https://colab.research.google.com/github/ryanraba/stocksml/blob/master/docs/data.ipynb

CHAPTER FOUR

MARKET DATA

StocksML uses stock market price data as the basis for training models to learn market trading strategies. A small set of demonstration data is included in the StocksML package, but generally users will need to download or otherwise supply their own price data.

4.1 Download from IEX Cloud

The `FetchData` function in StocksML can be used to download data from IEX Cloud. An account is needed (free or paid tier) on IEX to retrieve an API token from the console screen. Copy the token and paste it in to the `apikey` parameter. A list of desired ticker symbols and a start/end date range should be supplied. These will be stored as CSV files in the specified location.

Note that this will count towards your monthly quota on IEX.

Here we download a small sample of Google and Exxon price data.

[11]:
```
!pip install stocksml >/dev/null
!mkdir data >/dev/null
from stocksml import FetchData

FetchData(['GOOG', 'XOM'], apikey='xxxxxxxxxxxxxxxx', start='2020-08-01', stop='2020-
↪12-31', path='./data')
```

```
fetching GOOG data... 106 days
fetching XOM data... 106 days
```

Each ticker symbol is stored in a separate CSV file containing daily high, low, open, close and volume columns with a date column in yyyy-mm-dd format.

[16]:
```
!ls data/
```

```
GOOG.csv  XOM.csv
```

[17]:
```
!head data/GOOG.csv
```

```
date,open,high,low,close,volume
2020-08-03,1486.64,1490.47,1465.64,1474.45,2331514
2020-08-04,1476.57,1485.56,1458.65,1464.97,1903489
2020-08-05,1469.3,1482.41,1463.46,1473.61,1979957
2020-08-06,1471.75,1502.39,1466.0,1500.1,1995368
2020-08-07,1500.0,1516.845,1481.64,1494.49,1577826
2020-08-10,1487.18,1504.075,1473.08,1496.1,1289530
2020-08-11,1492.44,1510.0,1478.0,1480.32,1454365
```

(continues on next page)

(continued from previous page)

```
2020-08-12,1485.58,1512.3859,1485.25,1506.62,1437655
2020-08-13,1510.34,1537.25,1508.005,1518.45,1455208
```

Data from any other source may be used instead of IEX cloud if it can be represented in this same format.

4.2 Load Symbol DataFrame

Appropriately named and formatted CSV files can be loaded in to a single Symbol DataFrame (sdf) using `LoadData`. The sdf provides a convenient single location for all market data needed later on for model training and trading strategy simulation.

All files in the specified directory can be loaded by leaving the `symbols` parameter as None.

[18]:
```
from stocksml import LoadData

sdf, symbols = LoadData(symbols=None, path='./data')

sdf.head()
```

[18]:
```
            xom_open  xom_high  xom_low  ...  goog_low  goog_close  goog_volume
date                                     ...
2020-08-03     42.05     42.50    41.47  ...   1465.64     1474.45      2331514
2020-08-04     42.34     43.60    42.24  ...   1458.65     1464.97      1903489
2020-08-05     44.15     44.31    43.53  ...   1463.46     1473.61      1979957
2020-08-06     43.40     43.90    43.25  ...   1466.00     1500.10      1995368
2020-08-07     43.23     43.52    42.81  ...   1481.64     1494.49      1577826

[5 rows x 10 columns]
```

4.3 Build Feature DataFrame

The raw price data is not used directly by the models to learn a market strategy. Instead a set of training features must first be created to represent the data in a way that is more conducive to model learning. These are held in a feature dataframe (fdf).

These features are currently fixed within the `BuildData` function and are a work in progress, likely to be expanded in the future. They may potentially be made user configurable at a later date.

For now, all that is required to build an fdf is to pass the sdf to `BuildData`.

[19]:
```
fdf = BuildData(sdf)

fdf.head()
```

```
building GOOG data...
building XOM data...
```

[19]:
```
               goog0     goog1     goog2  ...      xom2      xom3      xom4
date                                      ...
2020-08-03 -0.014814 -0.017526 -0.010784  ... -0.001423 -0.000581  0.159583
2020-08-04 -0.043365 -0.066009 -0.054701  ...  0.042882  0.161771  0.510642
2020-08-05 -0.033192  0.015996 -0.042706  ...  0.273211  0.048568 -0.159542
2020-08-06  0.101999  0.000117  0.000026  ... -0.110557 -0.027508  0.256103
2020-08-07  0.068573  0.090926  0.113664  ... -0.026588 -0.026349  0.215602
```

(continues on next page)

(continued from previous page)

```
[5 rows x 10 columns]
```

Now we are ready to build a model that can learn a market strategy from this data.

Open in Colab: https://colab.research.google.com/github/ryanraba/stocksml/blob/master/docs/modeling.ipynb

---

CHAPTER

FIVE

DEFINING MODELS

Models can be created through a simple structure that defines each hidden layer. Keras and Tensorflow are used under the covers so many of the common layer types available in Keras are passed through including: - Dense Neural Network - Recurrent Neural Network - Long Short-Term Memory Network - Convolutional Neural Network - Dropout

The desired output size of each layer must also be defined. Activations and other settings are fixed. StocksML will attempt to fit together layers correctly and align with the training data, but some care must be taken to define things in a way that makes sense.

StocksML uses an unsupervised adversarial algorithm for learning new trading strategies. This requires at least two models to learn from each other. Additional models (specified by the `count` parameter) are created by copying the first model and re-initializing the initial weights. The `BuildModel` function returns a list of Keras models and a numpy array of training data appropriately shaped for the model set.

First lets create a dense neural network with three hidden layers. Dropout layers are typically inserted to help the model generalize and prevent overfitting.

[1]:
```
!pip install stocksml >/dev/null
from stocksml import LoadData, BuildData, BuildModel

sdf, symbols = LoadData(symbols=['SPY','BND', 'VNQI', 'VIXM'])
fdf = BuildData(sdf)
```

```
building BND data...
building SPY data...
building VIXM data...
building VNQI data...
```

[2]:
```
models, dx = BuildModel(fdf, len(symbols), count=2, layers=[('dnn',128),
                                                            ('drop', 0.25),
                                                            ('dnn',64),
                                                            ('drop', 0.25),
                                                            ('dnn',32)])
print('training data shape', dx.shape)
models[0].summary()
```

```
training data shape (1036, 20)
Model: "model"
__________________________________________________________________________________________________
↪______________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input (InputLayer)              [(None, 20)]         0
__________________________________________________________________________________________________
↪______________
dnn_0 (Dense)                   (None, 128)          2688        input[0][0]
```

(continues on next page)

(continued from previous page)

```
________________________________________________________________________________________
↪______________
drop_1 (Dropout)                (None, 128)          0           dnn_0[0][0]
________________________________________________________________________________________
↪______________
dnn_2 (Dense)                   (None, 64)           8256        drop_1[0][0]
________________________________________________________________________________________
↪______________
drop_3 (Dropout)                (None, 64)           0           dnn_2[0][0]
________________________________________________________________________________________
↪______________
dnn_4 (Dense)                   (None, 32)           2080        drop_3[0][0]
________________________________________________________________________________________
↪______________
action (Dense)                  (None, 5)            165         dnn_4[0][0]
________________________________________________________________________________________
↪______________
symbol (Dense)                  (None, 4)            132         dnn_4[0][0]
________________________________________________________________________________________
↪______________
limit (Dense)                   (None, 1)            33          dnn_4[0][0]
==================================================================================================
Total params: 13,354
Trainable params: 13,354
Non-trainable params: 0
________________________________________________________________________________________
↪______________
```

The dense and dropout layers we specified are created in the middle of the model (the 'hidden' portion) with the output sizes we provided. An input layer is added at the start and shaped to fit our provided feature dataframe (`fdf`). The 2-D numpy array `dx` is built from the feature dataframe returned for use in training later on.

Every model must end with three output layers: action, symbol, and limit. These output layers represent the "trading strategy" that is learned, including what action to take in the market (i.e. buy, sell, hold), what ticker symbol to use, and what limit price to set.

5.1 Recurrent Neural Networks

When a recurrent neural network (rnn or lstm) a third dimension is needed in the training data. This third dimension represents time and is created by stacking previous days of data. Use the `depth` parameter to control the size of the time stacking.

The recurrent layers can pass through the third dimension to each other, but this must be dropped when passing to a dense layer or the final output layers. This is handled automatically by StocksML.

[3]:
```
models, dx = BuildModel(fdf, len(symbols), count=2,
                        depth=5, layers=[('rnn',64),
                                         ('drop',0.25),
                                         ('rnn',32),
                                         ('drop',0.25),
                                         ('dnn',32)])
print('training data shape', dx.shape)
models[0].summary()
```

```
training data shape (1036, 5, 20)
Model: "model"
__________________________________________________________________________________________
↪______________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================================================
input (InputLayer)              [(None, 5, 20)]      0
__________________________________________________________________________________________
↪______________
rnn_0 (SimpleRNN)               (None, 5, 64)        5440        input[0][0]
__________________________________________________________________________________________
↪______________
drop_1 (Dropout)                (None, 5, 64)        0           rnn_0[0][0]
__________________________________________________________________________________________
↪______________
rnn_2 (SimpleRNN)               (None, 32)           3104        drop_1[0][0]
__________________________________________________________________________________________
↪______________
drop_3 (Dropout)                (None, 32)           0           rnn_2[0][0]
__________________________________________________________________________________________
↪______________
dnn_4 (Dense)                   (None, 32)           1056        drop_3[0][0]
__________________________________________________________________________________________
↪______________
action (Dense)                  (None, 5)            165         dnn_4[0][0]
__________________________________________________________________________________________
↪______________
symbol (Dense)                  (None, 4)            132         dnn_4[0][0]
__________________________________________________________________________________________
↪______________
limit (Dense)                   (None, 1)            33          dnn_4[0][0]
==========================================================================================================
Total params: 9,930
Trainable params: 9,930
Non-trainable params: 0
__________________________________________________________________________________________
↪______________
```

We see that the input and rnn_0 layers have an extra dimension in the output shape. This is gone in the output of rnn_2 passed to dnn_4. The shape of the training data returned in `dx` is now 3 dimensional.

5.2 Convolutional Neural Network

As with recurrent neural networks, convolutional neural networks also need a third time dimension. When using a CNN, the third dimension is suppressed with an extra Flatten layer inserted afterwards.

[4]:
```
models, dx = BuildModel(fdf, len(symbols), count=2,
                        depth=5, layers=[('cnn',32),
                                         ('drop',0.25),
                                         ('cnn',16),
                                         ('drop',0.25),
                                         ('dnn',32)])
print('training data shape', dx.shape)
models[0].summary()
```

```
training data shape (1036, 5, 20)
Model: "model"
________________________________________________________________________________________
↪______________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input (InputLayer)              [(None, 5, 20)]      0
________________________________________________________________________________________
↪______________
cnn_0 (Conv1D)                  (None, 3, 32)        1952        input[0][0]
________________________________________________________________________________________
↪______________
drop_1 (Dropout)                (None, 3, 32)        0           cnn_0[0][0]
________________________________________________________________________________________
↪______________
cnn_2 (Conv1D)                  (None, 1, 16)        1552        drop_1[0][0]
________________________________________________________________________________________
↪______________
flatten (Flatten)               (None, 16)           0           cnn_2[0][0]
________________________________________________________________________________________
↪______________
drop_3 (Dropout)                (None, 16)           0           flatten[0][0]
________________________________________________________________________________________
↪______________
dnn_4 (Dense)                   (None, 32)           544         drop_3[0][0]
________________________________________________________________________________________
↪______________
action (Dense)                  (None, 5)            165         dnn_4[0][0]
________________________________________________________________________________________
↪______________
symbol (Dense)                  (None, 4)            132         dnn_4[0][0]
________________________________________________________________________________________
↪______________
limit (Dense)                   (None, 1)            33          dnn_4[0][0]
==================================================================================================
Total params: 4,378
Trainable params: 4,378
Non-trainable params: 0
________________________________________________________________________________________
↪______________
```

Here we see that the cnn_0 layer passed 3-D data to the next cnn_2 layer, but then a flatten layer is automatically inserted before passing to the dense layers. As with the recurrent models, the training data in `dx` is now 3-D.

5.3 Limiting Symbol Choices

One of the three output layers (symbol) decides which ticker symbol to use in trading for the corresponding action and limit. This symbol must be present in the feature dataframe (fdf), but the models don't actually care about that. They simply need to know what the maximum number of symbols is that they are going to be choosing from.

Sometimes it is desireable to restrict the ticker symbols used for actual trading to just a subset of what is in the training data. In this case, the `choices` parameter can be reduced to the desired value. Later on during training, this must be remembered and preserved for accurate strategy learning.

```
[5]: models, dx = BuildModel(fdf, 2, count=2, layers=[('dnn',128),('dnn',64),('dnn',32)])
models[0].summary()
```

```
Model: "model"
__________________________________________________________________________________________________
↪_____________
Layer (type)                    Output Shape          Param #     Connected to
==================================================================================================
input (InputLayer)              [(None, 20)]          0
__________________________________________________________________________________________________
↪_____________
dnn_0 (Dense)                   (None, 128)           2688        input[0][0]
__________________________________________________________________________________________________
↪_____________
dnn_1 (Dense)                   (None, 64)            8256        dnn_0[0][0]
__________________________________________________________________________________________________
↪_____________
dnn_2 (Dense)                   (None, 32)            2080        dnn_1[0][0]
__________________________________________________________________________________________________
↪_____________
action (Dense)                  (None, 5)             165         dnn_2[0][0]
__________________________________________________________________________________________________
↪_____________
symbol (Dense)                  (None, 2)             66          dnn_2[0][0]
__________________________________________________________________________________________________
↪_____________
limit (Dense)                   (None, 1)             33          dnn_2[0][0]
==================================================================================================
Total params: 13,288
Trainable params: 13,288
Non-trainable params: 0
__________________________________________________________________________________________________
↪_____________
```

The size of the symbol output layer tracks to the value passed in to the `choices` parameter.

5.4 Advanced Models

If you are comfortable using Keras directly, you can certainly build your own models with whatever advanced features you desire. The only constraint is that they must have one input layer and three output layers corresponding to action, symbol and limit as demonstrated above. It is likely easiest to continue to use the `BuildModel` function to construct the training data array `dx` even if ignoring the model list returned. The other option is augmenting the model list with additional advanced models of your own, they need not all be the same.

PYTHON MODULE INDEX

S

INDEX