Index

1. Web Integration Layer
   1. Market High Granularity Data Feed
   2. Market Open Close Prices Data Feed
   3. Twitter API Scraper
   4. News Web Scraper
2. Data Pre-Processing Layer
   1. Market Minute Granularity Data Cleanup Script
   2. Market Minute Granularity Self Encoder CNN
   3. Market Open Close Prices Data Cleanup Script
   4. Market Open Close Prices Self Encoder CNN
   5. Twitter Sentiment Classification RNN
   6. Twitter Sentiment Self Encoder VanillaNet
   7. News Sentiment Classification RNN
   8. News Sentiment Self Encoder VanillaNet
3. Decision Processing Layer
   1. Market Minute Granularity Buy/Sell classification RNN
   2. Market Open Close Prices Buy/Sell classification RNN
   3. Twitter Buy/Sell classification RNN
   4. News Buy/Sell classification RNN
   5. ReinforcementNet Input Cleanup Script
   6. Budget Constraint Calculator
   7. Strategy Picker ReinforcementNet
4. Platform Integration Layer
   1. Portfolio State Data Pull API
   2. Trade Execution Platform API
5. Web Option Spread Source
   1. Option Spread Data Cleanup Script
   2. Strategy Optimization Algorithm
6. Built-in Analytics Tool
   1. Portfolio Value Database
   2. Trade History Log

**Global Variables**

Need to account for special days via market operating hours

Add job flow control script.

@Market\_Open: 9:30am EST

@Market\_Close: 4:00pm EST

@Open\_Hours\_perday: 6.5

@Open\_Minutes\_perday: 390

@Open\_Seconds\_perday: 23,400

@Open\_Hours\_perweek: 32.5

@Open\_Minutes\_perweek: 1,950

@Open\_Seconds\_perweek: 117,000

@SP500 = list with objects for all companies listed under the S&P500 index

@Traded\_Assets: list with objects for the assets that the algorithm is allowed to trade. An example value for this variable would be [GOOGL, AAPL, TSLA]

@Traded\_Assets\_count: the number of assets in @Traded\_Assets. For the above example that would be equal to 3.

**System Architecture**

Could be interesting to include some algorithm that accounts for analysts ratings. Or also one that looks at economic calendar events.

1. **Web Integration Layer**
   1. **Market High Granularity Data Feed**

**Functionality:**

The goal of this system is to automate the process of gathering market data from a web source. The idea is that this data set will be an accurate representation of the high frequency short term fluctuations of the prices and will help the ReinforcementNet make an accurate decision by accounting for this information.

Is the high frequency price spread enough of an indicator of the traded volume? Could be interesting to add volume at each time stamp?

**Data Hyperparameters:**

The data specs are hyperparameters that are up for optimization. Here is a description of the boundaries of the data set that seem reasonable at a first glance. These should be optimized according to the data availability when the Neural Networks are built and ready for optimization.

* Data\_Granularity:

Lower bound: asset prices at each 30 sec interval from market open to market close

Upper bound: asset prices at each 5 min interval from market open to market close

* Data\_Time\_Range:

Lower bound: 3 days worth of data

Upper bound: 2 weeks worth of data

* Number\_of\_Assets:

Lower bound: @Traded\_Assets\_count, corresponding to the assets listed on @Traded\_Assets

Upper bound: 500, corresponding to the assets listed on @SP500

* Pull\_Frequency:

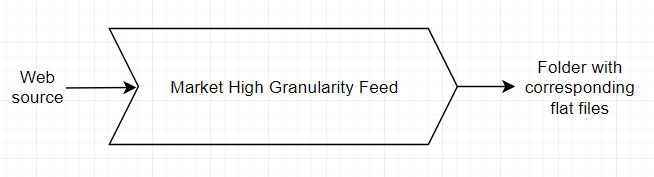
Lower bound: Daily

Upper bound: pull frequency = data granularity

This hyperparameter is somewhat dependent on the frequency at which the algorithm is trading as well as the processing power of the machine. Theoretically the system could run this once a day and trade at every 30 minutes but that might heavily harm the accuracy of this data set. On the other hand if this feed is running at every 30 second interval computing power might become an issue unless there is access to a powerful server.

Also, if frequency is not equal to granularity another parameter to be optimized is the time of the pull. My guess is that a pulls at close, open and midday could be a good approach.

**Design:**



* Inputs:

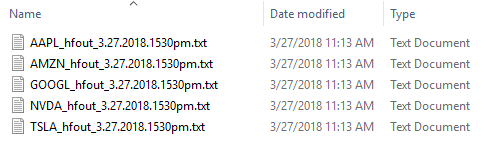
URL with the data source or data source API -> TBD.

Are there any free sources that provide data with this granularity?

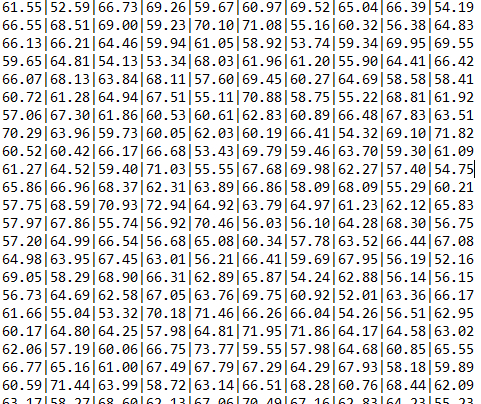
* Outputs:

Set of flat text files containing ordered data for each of the assets accounted for in Number\_of\_Assets

An example output folder where the list of monitored stocks = [AAPL, AMZN, GOOGL, NVDA, TSLA] would look like:



Where each file contains a character delimited data set with dimensions @Open\_Seconds\_perday/Data\_Granularity, Number of operating days within Data\_Time\_Range. Each row represents the price at a specific timestamp starting at @Market\_Open and ending at @Market\_Close with intervals determined by Data\_Granularity. Each column represents an operating day within the Data\_Time\_Range closest to the extract date. It should look something like this:



A python script should work well for this job. In case of available API from source check which language is recommended.

For the final design I probably dont need to be passing a ton of files around. Check out cube extension for SQL to learn more about how to store high dimension data into DBs

<https://www.postgresql.org/docs/current/static/cube.html>

* 1. **Market Open Close Prices Data Feed**

**Functionality:**

The goal of system is to automate the process of gathering market data from a web source. The idea is that this data set will be an accurate representation of the medium frequency long term fluctuations of the market prices and will help the ReinforcementNet make an accurate decision by accounting for this information.

Again, same concern how can I account for volume? With a daily frequency it sounds like just the price fluctuations won’t be enough to accurately account for the volume fluctuations. Could be a good idea to have another feed with volume and other market indicators, maybe momentum indexes and other higher level features? What other “raw” attributes are relevant besides price and volume? Might be interesting to stay away from indexes and let the nets figure out those by itself.

**Data Hyperparameters:**

The data specs are hyperparameters that are up for optimization. Here is a description of the boundaries of the data set that seem reasonable at a first glance. These should be optimized according to the data availability when the Neural Networks are built and ready for optimization.

* Data\_Granularity:

Lower bound: Four snapshots of the price for each asset per day.

Upper bound: Open and close prices per day for each asset.

In case the design choice includes snapshots that are not on open and close it could be interesting to have a non-even distribution of those. Maybe concentrating the snapshots at the first and last hours of the day could be more informative than an evenly distributing them.

* Data\_Time\_Range:

Lower bound: one month worth of data

Upper bound: six months worth of data

* Number of Assets:

Lower bound: @Traded\_Assets\_count, corresponding to the assets listed on @Traded\_Assets

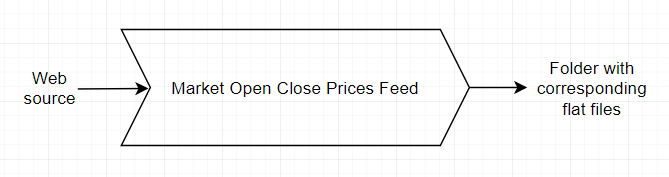
Upper bound: 500, corresponding to the assets listed on @SP500

* Pull Frequency:

Lower bound: Daily

Upper bound: One pull at each price snapshot

**Design:**

****

* Inputs:

URL with the data source or data source API -> TBD.

There probably are free sources that provide data with this granularity. Investigate for those with no policies regarding automatic scrapping.

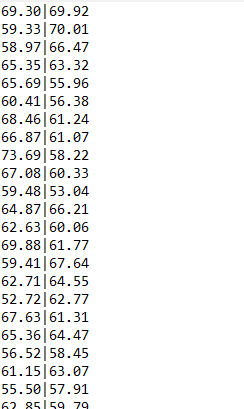
* Outputs:

Set of flat text files containing ordered data for each of the assets accounted for in Number\_of\_Assets

An example output folder where the list of monitored stocks = [AAPL, AMZN, GOOGL, NVDA, TSLA] would look like:



Where each file contains a character delimited data set with dimensions Data\_Granularity, Number of operating days within Data\_Time\_Range. Each row represents the prices at a specific operating day within the Data\_Time\_Range closest to the extract date. Each column is the price at a timestamp (in the example below price at open, price at close). It should look something like this:



A python script should work well for this job. In case of available API from source check which language is recommended.

For the final design I probably dont need to be passing a ton of files around. Check out cube extension for SQL to learn more about how to store high dimension data into DBs

<https://www.postgresql.org/docs/current/static/cube.html>

* 1. **Twitter API Scraper**

**Functionality:**

The job of this feed is to automatically pull data from [twitter.com](https://twitter.com/). The goal is use a set of keywords, hashtags and twitter users as a source to pull large amount of tweets related to the companies that underlie the assets being traded.

Tracking companies that we are not trading could improve performance?

Can potentially be improved by creating a strong algorithm to dynamic select the set of keywords, hashtags and users for the scrapper.

**Data Hyperparameters:**

The data specs are hyperparameters that are up for optimization. Here is a description of the boundaries of the data set that seem reasonable at a first glance. These should be optimized according to the data availability when the Neural Networks are built and ready for optimization.

* Data\_Time\_Range:

Lower bound: 7 days

Upper bound: 1 month

* No\_of\_Tweets:

Lower bound: 1,000

Upper bound: 50,000

The idea here is to have a lower and upper bound amount of tweets that can be collected within the Data\_Time\_Range. This will avoid an overload of tweets whenever there is some sort of special event as well as an underload that could generate an invalid data set.

Looking at the volume concentration of tweets could be an interesting indicator (e.g. how many tweets within the last hour, a big spike could indicate some special event)

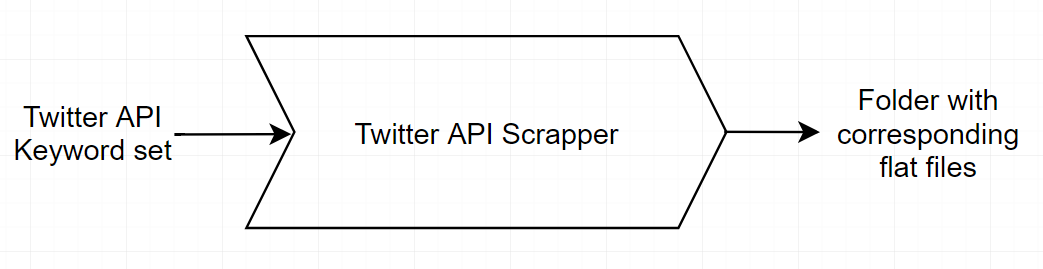
* Pull\_Frequency

Lower bound: Daily

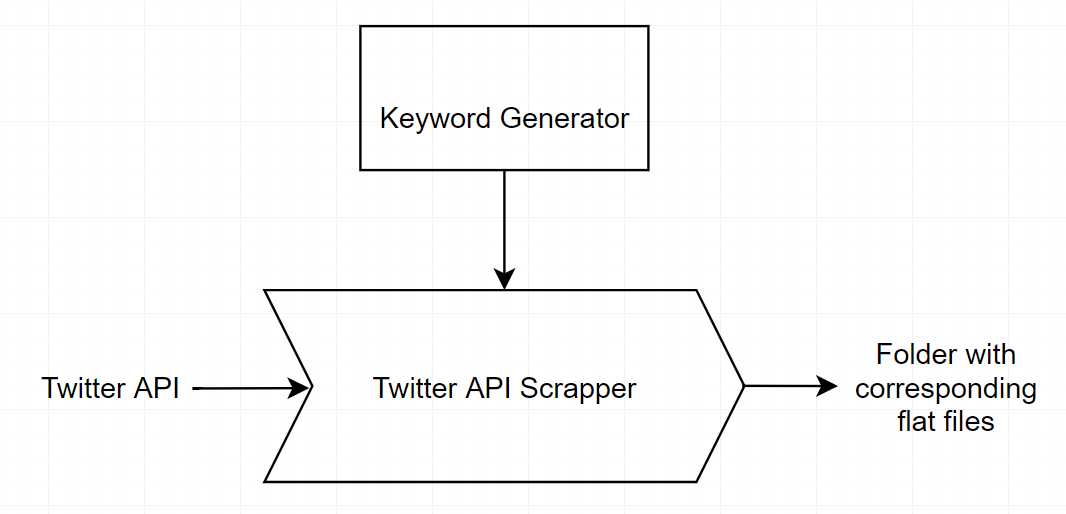
Upper bound: Weekly

This pull frequency can become tricky depending on how trending the underlying company is. Maybe it could be interesting to select @Traded\_Assets based on a minimum/maximum twitter requirement otherwise this data feed might bias the performance of the algorithm.

**Design:**



Possible improvement:



* Input: Twitter API <https://developer.twitter.com/en/docs>

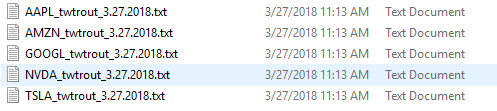
Set of keywords, hashtags and twitter users

how to avoid pulling several tweets from the same user?

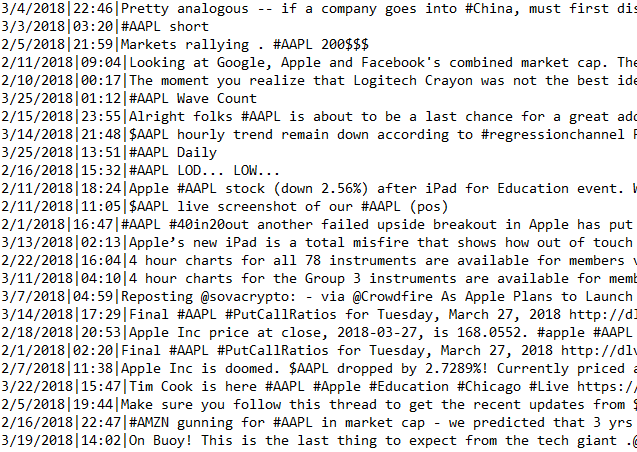
How to create a relevant keyword generator for @Traded\_Assets

* Output: Set of flat text files containing time stamped strings for each of the underlying companies related to the assets accounted for in Number\_of\_Assets.

An example output folder where the list of monitored stocks = [AAPL, AMZN, GOOGL, NVDA, TSLA] would look like:



Each file contains a set of tweets selected according to the set of keywords. They should look something like the following:



For the final design I probably dont need to be passing a ton of files around. Check out cube extension for SQL to learn more about how to store high dimension data into DBs

<https://www.postgresql.org/docs/current/static/cube.html>

* 1. **News Web Scraper**

**Functionality:**

The job of this feed is to automatically pull data from news sources. The goal is use a set of keywords as a source to pull large lists of news articles related to the companies that underlie the assets being traded.

Which one would yield best results: Pulling exclusively from financial new sources (might induct bias) or pulling from general search engines like google (might pull loosely related data)?

Is it better to pull only the article titles (easier to create a classification algorithm but might be classification can be deceiving if its only based on the title) or is it better to pull the full article and try to classify it (probably very challenging to classify a full article)?

**Data Hyperparameters:**

The data specs are hyperparameters that are up for optimization. Here is a description of the boundaries of the data set that seem reasonable at a first glance. These should be optimized according to the data availability when the Neural Networks are built and ready for optimization.

* Data\_Time\_Range:

Lower bound: 7 days

Upper bound: 1 month

* No\_of\_Articles:

Lower bound: 500

Upper bound: 5,000

The idea here is to have a lower and upper bound amount of articles that can be collected within the Data\_Time\_Range. This will avoid an overload whenever there is some sort of special event as well as an underload that could generate an invalid data set.

Looking at the volume concentration of news could be an interesting indicator (e.g. how many news within the last hour, a big spike could indicate some special event)

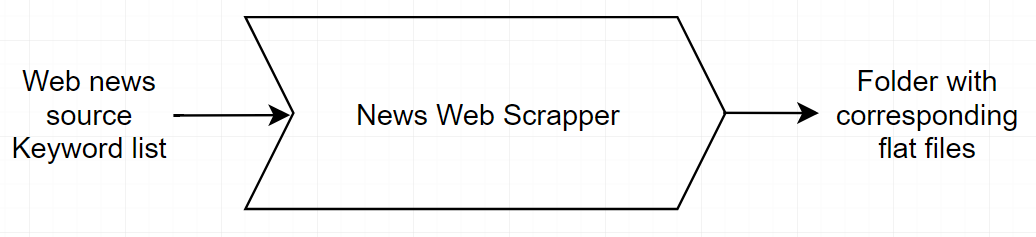
* Pull\_Frequency

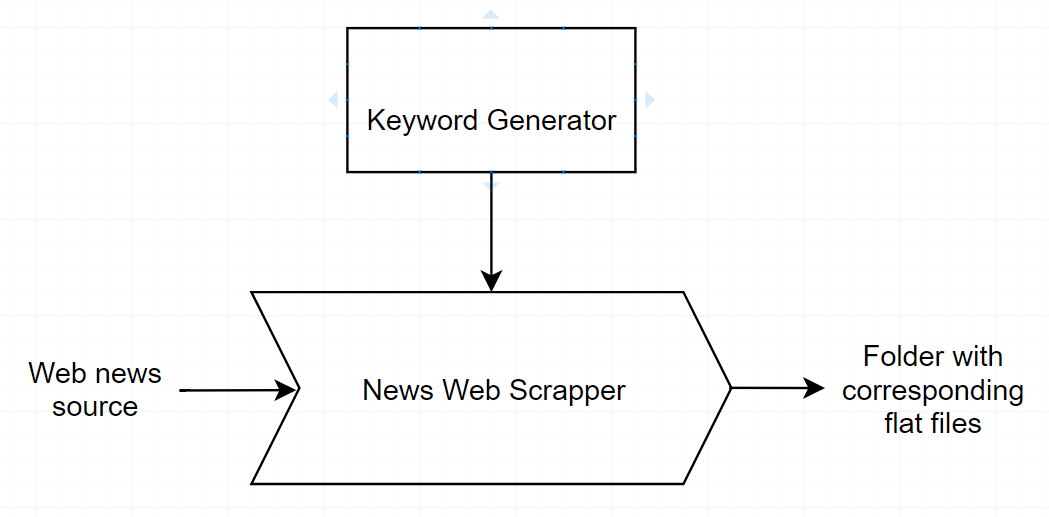
Lower bound: Daily

Upper bound: Weekly

Is the data collected by this algorithm significantly different from the twitter API data? Could be interesting to investigate if having both isn’t some sort of overkill.

**Design:**

****

****

* Inputs:

News source list -> TBD

Keywords list

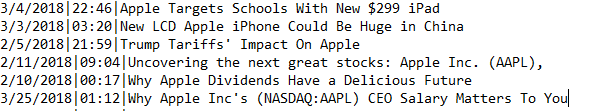
* Outputs:

Set of flat text files containing time stamped strings for each of the assets accounted for in @Traded\_Assets.

An example output folder where the list of monitored stocks = [AAPL, AMZN, GOOGL, NVDA, TSLA] would look like:



Where each file contains strings corresponding to the title of news articles that were matched by the keyword list.



Python will probably be a good choice for this script. Investigate web scraping tools

For the final design I probably dont need to be passing a ton of files around. Check out cube extension for SQL to learn more about how to store high dimension data into DBs

<https://www.postgresql.org/docs/current/static/cube.html>

1. **Data Pre-Processing Layer**
   1. **Market High Granularity Data Cleanup Script**

**Functionality:**

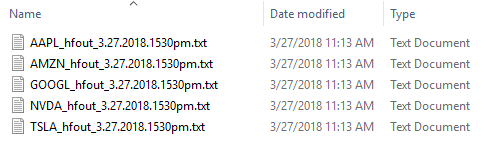
The goal of this script is to transform the flat files outputted by the Market High Granularity Data Feed into a tensor that will be encoded and eventually used as an input for the Market High Granularity Buy/Sell classification RNN.

**Design:**

* Input:

Folder with flat files extracted by the Market High Granularity Data Feed

An example input folder where the list of monitored stocks = [AAPL, AMZN, GOOGL, NVDA, TSLA] would look like:



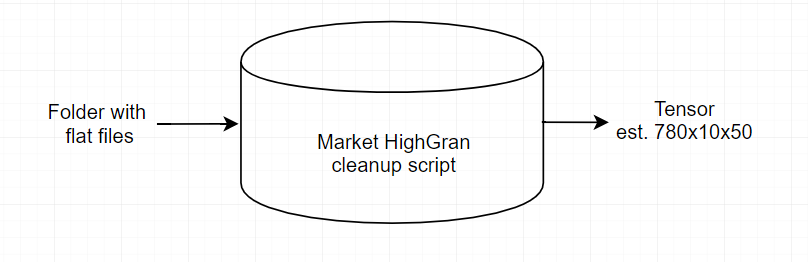
* Output:

A tensor containing all the information from the flat files. This tensor will serve as an input to the Market High Granularity Self Encoder CNN. The expected dimensions of the tensor are:

Lower bound: 78x3x@Traded\_Assets

Upper bound: 780x10x500

Dimension sizes correspond to (@Open\_Seconds\_perday/Data\_Granularity), Number of operating days within Data\_Time\_Range, Number of Assets being analyzed by the algorithm (>= @Number\_of\_Assets).



A simple SSIS script should be enough to do this job. How to create and store tensors with SSIS? There must be some sort of ML package that allows that.

* 1. **Market High Granularity Self Encoder CNN**

**Functionality:**

The goal of this neural network is to reduce the dimensions of the tensor outputted by the Market High Granularity Cleanup Script while preserving the highest amount of information. The output of this encoder will then be used as an input for the Market High Granularity Buy/Sell classification RNN.

What are the implications of using a self encoder on a time series? Is there a specific type of self-encoder that should be used? For now I’m exemplifying with a regular CNN self-encoder

Check Sequence-to-Sequence Self encoders (<https://machinelearningmastery.com/develop-encoder-decoder-model-sequence-sequence-prediction-keras/>)

Doing PCA can be more effective than using a CNN. If it is can it be automated? PCA requires hyperparameter optimization that is case specific?

**Design:**

* Input:

A tensor containing all the information from the high granularity flat files outputted by the Market High Granularity Cleanup Script. The expected dimensions of the tensor are:

Lower bound: 78x3x@Traded\_Assets

Upper bound: 780x10x500

Dimension sizes correspond to (@Open\_Seconds\_perday/Data\_Granularity), Number of operating days within Data\_Time\_Range, Number of Assets being analyzed by the algorithm (>= @Number\_of\_Assets).

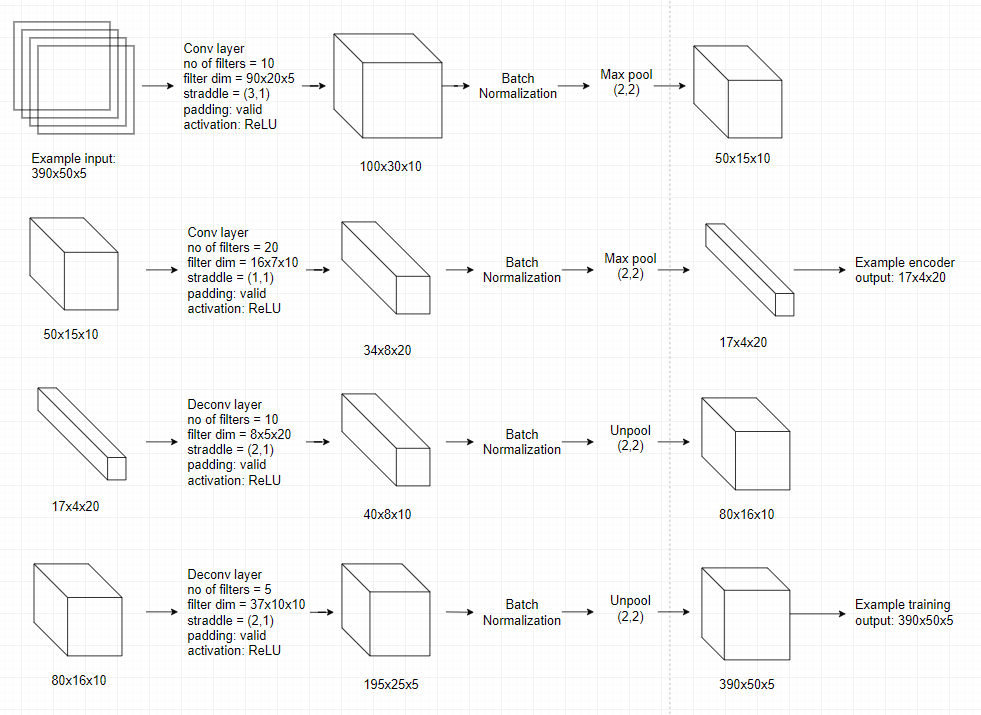
* Output:

A lower dimensional tensor derived from the input that preserves the probabilistic features of the data.

Lower bound: 8x3x10

Upper bound: 260x10x50

Here is a base architecture that can work as a starting point for hyperparameter optimization.



* 1. **Market Open Close Prices Data Cleanup Script**

**Functionality:**

The goal of this script is to transform the flat files outputted by the Market Open Close Prices Data Feed into a tensor that will be encoded and eventually used as an input for the Market Open Close Prices Buy/Sell classification RNN.

**Design:**

* Input:

Folder with flat files extracted by the Market Open Close Prices Data Feed

An example input folder where the list of monitored stocks = [AAPL, AMZN, GOOGL, NVDA, TSLA] would look like:



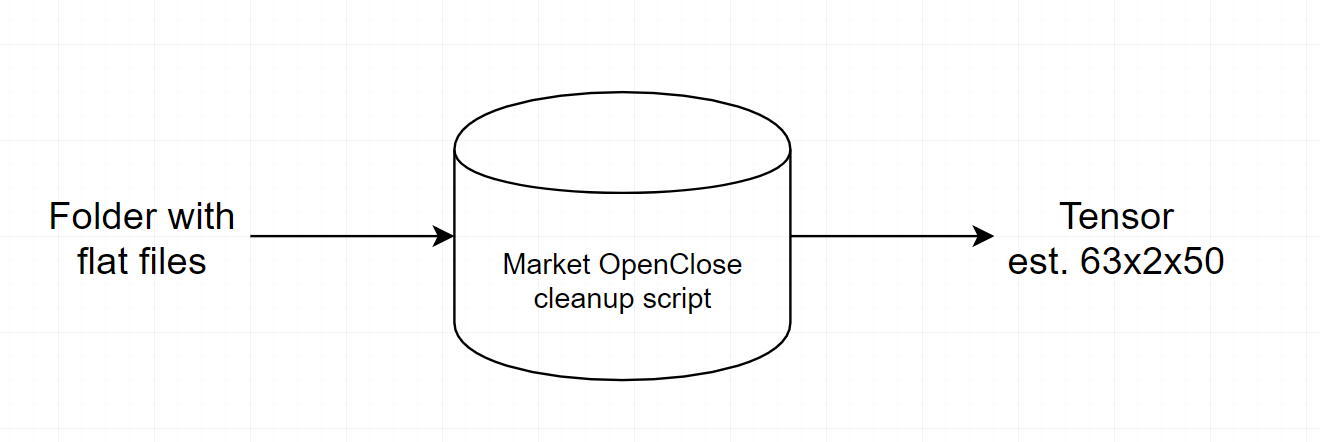
* Output:

A tensor containing all the information from the flat files. This tensor will serve as an input to the Market Open Close Prices Self Encoder CNN. The expected dimensions of the tensor are:

Lower bound: 21x2x@Traded\_Assets

Upper bound: 126x4x500

Dimension sizes correspond to Number of operating days in Data\_Time\_Range, Number of price snapshots per day, Number of Assets being analyzed by the algorithm (>= @Number\_of\_Assets).



A simple SSIS script should be enough to do this job. How to create and store tensors with SSIS? There must be some sort of ML package that allows that.

* 1. **Market Open Close Prices Self Encoder CNN**

**Functionality:**

The goal of this neural network is to reduce the dimensions of the tensor outputted by the Market Open Close Price Cleanup Script while preserving the highest amount of information. The output of this encoder will then be used as an input for the Market Open Close Price Buy/Sell classification RNN.

What are the implications of using a self encoder on a time series? Is there a specific type of self-encoder that should be used? For now I’m exemplifying with a regular CNN self-encoder

Check Sequence-to-Sequence Self encoders (<https://machinelearningmastery.com/develop-encoder-decoder-model-sequence-sequence-prediction-keras/>)

Doing PCA can be more effective than using a CNN. If it is can it be automated? PCA requires hyperparameter optimization that is case specific?

**Design:**

* Input:

A tensor containing all the information from the high granularity flat files outputted by the Market Open Close Price Cleanup Script. The expected dimensions of the tensor are:

Lower bound: 21x2x@Traded\_Assets

Upper bound: 126x4x500

Dimension sizes correspond to Number of operating days in Data\_Time\_Range, Number of price snapshots per day, Number of Assets being analyzed by the algorithm (>= @Number\_of\_Assets)

* Output:

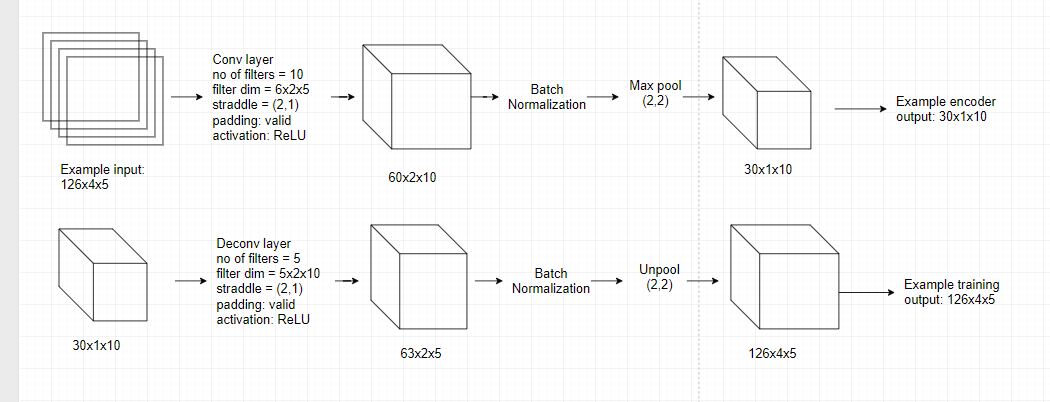
A lower dimensional tensor derived from the input that preserves the probabilistic features of the data.

Lower bound: 8x2x10

Upper bound: 62x4x50

These output dimensions are arbitrary estimates. Would the algorithm perform better if all self encoders outputted a tensor with similar size?

Here is a base architecture that can work as a starting point for hyperparameter optimization.



* 1. **Twitter Sentiment Classification RNN**

Functionality:

The goal of this Neural Network is to take as an input the time stamped data extracted by the Twitter API Scraper function and classify each data point individually into one of three categories:

1. Buy signal
2. Sell signal
3. Neutral

Then it will order the classified output according to its time stamp into a time series (Possibly aggregate tweets that happened on the same day as a single time stamp for the series so that each time stamp is a sequence of classified tweets that happened on that day). This ordered output will serve as an input to the Twitter Sentiment Sequence to Sequence Self Encoder.

Design: TBD. Check for already existing algorithms for text classification. Shouldn’t be too hard to find a generic LSTM of the type each character = 1 sequence input as a starting point for this algorithm.

* 1. **Twitter Sentiment Sequence to Sequence Self Encoder**

Functionality:

The goal of this self encoder/decoder is to reduce the dimensionality of the Twitter Sentiment Classification RNN output. This reduced sequence will serve as an input to the Twitter buy/sell classification RNN.

Design: TBD. Need to study more about sequence to sequence encoder decoders

<https://machinelearningmastery.com/develop-encoder-decoder-model-sequence-sequence-prediction-keras/>

* 1. **News Sentiment Classification RNN**

Functionality:

The goal of this Neural Network is to take as an input the time stamped data extracted by the News Web Scraper function and classify each data point individually into one of three categories:

1. Buy signal
2. Sell signal
3. Neutral

Then it will order the classified output according to its time stamp into a time series (Possibly aggregate news that came out on the same day as a single time stamp for the series so that each time stamp is a sequence of classified news that happened on that day). This ordered output will serve as an input to the News Sentiment Sequence to Sequence Self Encoder.

Design: TBD. Check already existing algorithms for text classification. Shouldn’t be too hard to find a generic LSTM of the type each character = 1 sequence input as a starting point for this algorithm.

Should this algorithm look at just the headlines of the news or the entire texts on the articles? Different approach for different sources? What are the computational implications for this?

Architecture: TBD

* 1. **News Sentiment Sequence to Sequence Self Encoder**

Functionality:

The goal of this self encoder/decoder is to reduce the dimensionality of the News Sentiment Classification RNN output. This reduced sequence will serve as an input to the News buy/sell classification RNN.

Design: TBD. Need to study more about sequence to sequence encoder decoders

<https://machinelearningmastery.com/develop-encoder-decoder-model-sequence-sequence-prediction-keras/>

1. Decision Processing Layer
   1. Market High Granularity Buy/Sell classification RNN

Functionality:

* 1. Market Open Close Prices Buy/Sell classification RNN

Functionality:

* 1. Twitter Buy/Sell classification RNN

Functionality:

* 1. News Buy/Sell classification RNN

Functionality:

* 1. ReinforcementNet Input Cleanup Script

Functionality:

* 1. Budget Constraint Calculator

Functionality:

* 1. Strategy Picker ReinforcementNet

Functionality:

1. Platform Integration Layer
   1. Portfolio State Data Pull API

Functionality:

* 1. Trade Execution Platform API

Functionality:

1. Web Option Spread Source
   1. Option Spread Data Cleanup Script

Functionality:

* 1. Strategy Optimization Algorithm

Functionality:

1. Built-in Analytics Tool
   1. Portfolio Value Database

Functionality:

* 1. Trade History Log

Functionality: