Application of Deep Convolutional networks applied to Portfolio Optimization.

Author: Roy Justus (roy@justushome.net)

Introduction:

In this paper, I demonstrate the use Deep Convolutional models to learn complex hidden features in a market, across equities with the ultimate objective of producing optimal portfolio allocations among multiple equities in order to inform trading strategies.

The underlying hypothesis behind this approach is:

Where correlations exist between prices of equities, the mechanisms of actions of some subset of these correlations are likely to take effect with some delay allowing some equities to serve as weak leading indicators of the behavior of others.

Prior work:

The applications of a reliable predictive and prescriptive models of stock prices and portfolios promise direct financial benefits and thus have been widely studied. While it can be assumed that the majority of the serious effort applied to this field is behind closed doors of institutional investors there is some published work that should be noted where Deep Learning architectures such as MLPs, CNN and LSTM models have been applied to the problem of predicting the future price of an equity at some point in the future.

K. Khare, O. Darekar, P. Gupta and V. Z. Attar provide a conceptual overview of how MLP, LSTM architectures could be applied and conducted their own research which concluded that an MLP architecture outperforms an LSTM based model for short term price prediction. Documented efforts to apply CNN models include the 2019 work on the Thai stock exchange by L. Sayavong, Z. Wu and S. Chalita. Their approach included application of one or two convolution + pooling layers and predicted output price.

Several common themes are seen in the literature is the use of absolute price as the prediction target and RMSE as the evaluation criteria. Also noteworthy is that each of these papers have been applied primarily to non-US stock markets using daily market data and look at one to 10 stocks independently, that is to say that the prediction of stock X's price is based only on data from stock X (daily open, close, high low and volume "bars" are the typical data sets.)

This project expands on the prior work through the incorporation of three novel components in the analysis:

1. I will attempt to factor in features of 346 stocks over sliding 60-minute windows using a novel architecture of 2D convolutional layers.
2. I will attempt to optimize a portfolio of multiple stocks simultaneously as a multi-task learning problem.
3. Finally, for purpose of evaluation I will use the ability to generate returns in excess of a balanced portfolio of the applicable equites as the evaluation metric rather than RMSE of price prediction.

Limitation of Scope:

This project has not taken on the very significant considerations of trading costs, market impact or ability to purchase shares at a quoted close price and leaves these tasks to future work, likely though construction of a Reinforcement Learning agent.

Selection of target equities:

The equities analyzed in this project were selected from among S&P 400 mid-cap stocks with a history going back to at least 2015. Mid-cap stocks are somewhat less actively traded, and I suspected that greater informational latency might exist in those markets due to less focused scrutiny from institutional investors.

Dataset and Features

Data Preparation:

I have requested and been granted access to Polygon.io datasets which provide equity, forex and other data streams both historically and in real time for minute by minute resolution. The retrieved data is for 346 stocks selected from US mid-cap stocks with a public trading history going back to at least 2015 and in the raw form consisted of standard Open, Close, High, Low, Volume for each minute period for each stock. (With gaps if a stock was not traded in that minute which have been filled by setting volume to 0 and filling forward the previous close for other values).

Features:

From this dataset I extracted 8 simple features (show in the table on the right) which were calculated as my model inputs. Given time constraints, tuning the feature inputs was not a significant focus and greater improvements are expected from greater refinement.

Table of Features

Name	Time Period
Absolute Change	1 minute
Percent Change	1 minute
Relative Rolling Mean (10)	10 minutes
Relative Rolling Mean (60)	60 minutes
Relative Rolling Low (10)	10 minutes
Relative Rolling High (10)	10 minutes

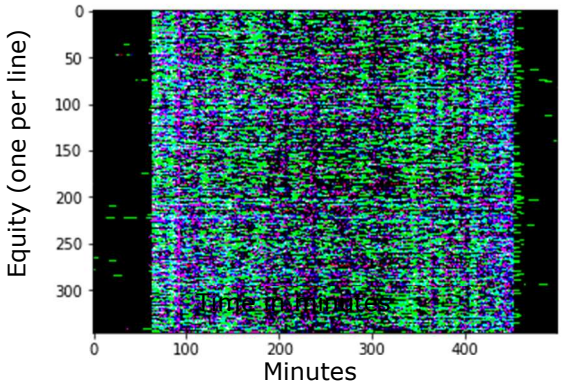
Standardization:

In X and in numerous blog articles online Y where the prediction target is the next equity price in absolute terms the input features are scaled to a range between 0-1 using a min-max scaler. Given that I had selected a set of near stationary features for my inputs I opted for a standard scaler instead (mean=0, variance=1) with each input feature scaled independently.

Input Data structure:

It's important to note that my input data eventually takes the shape of (**m, t**, **e, f)** where the variables are respectively sa**m**ples, **t**imesteps, **e**quities and **f**eatures. This can be thought of as a 60 by 346 image with 8 channels. This intuition is important and I've plotted an image to the right using a subset of features to give you a sense of what it might look like over one trading day.

2D visualization of Input Data



R: Percent Change
G: Relative Rolling Mean
B: Range

Methodology:

While the ultimate objective is to recommend an optimal portfolio allocation policy, this project separated the objective into two stages. The first phase was intended to confirm that the model architecture had a feature extractor capable able to make meaningful sense of the raw input data and then in the second stage the objective function was modified to directly optimize profitability of the portfolio based on input data.

The two problems shared a substantial portion of the network architecture with only a few layers at the output side being changed to allow for the appropriate output format. (section C in the *Model Architecture Summary* on the following page)

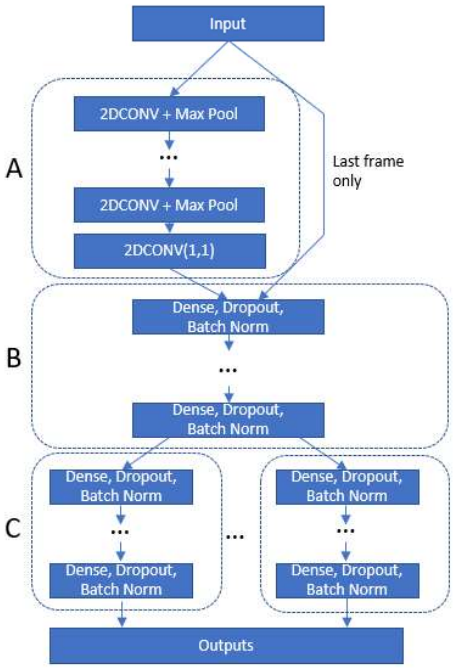
Section A is a convolutional network which performs two stages of feature extraction. In the first stage (3, 1) and (2, 1) convolutions [1]are used with average pooling to analyses the features of each equity independently, summarizing to more abstract features based on increases and decreases in the various feature channels. Then in the second stage of convolutions a (2, 346) convolution scans the entire market one timestep at a time with 256 filters intended to map interdependencies between the high-level features of multiple equities. This can be thought of as first looking for structure in the X axis of my 2d visualization and then followed by scanning for noteworthy interrelations in those structures over the entire Y axis.

[1] The unusual choice of such small filters is inspired by the work of J. Eapen, D. Bein and A. Verma who combined 1D convolutions with LTSTM models to produce promising results.

Section B is a simple sequence of Dense layers whose exact parameters are determined in my hyperparameter search.

Section C is the output mapping layers, in the Prediction problem this consists of small dense layers for each equity while in the Allocation problem this consists of dense layers terminating in a SoftMax to provide a vector of fractional portfolio allocations summing to 1.

Model Architecture Summary



Hyper Parameters

For each of my experiments I have constructed a hyperparameter specification config file which allowed me to quickly vary the following quantities (and some others that I ultimately found it not advantageous to tune)

My search methodology initially was to hand select likely combinations based on intuition, quickly this became time consuming so I set up an asynchronous hypothesis generation and testing pipeline by deploying my learning model to parallel cloud instances in AWS and feeding in a central config file queue for worker nodes to test. This allowed me to scale up to 6 GPU instances to simultaneously test hyperparameters.

Encoder layers – A full specification of Encoder Layers for Module B. This consisted of 4-8 Dense layers with 150-2250 units in each. Here I sampled uniformly in 150 unit increments throughout the range.

Dropout Regularization – Most dense layers in the model have dropout applied. For practical reasons only a single dropout value was used throughout and this was sampled uniformly from between 0-0.5. with 50% probability no dropout is applied.

L2 Weight Decay Regularization – Most layers in the model have an L2 regularization penalty applied. Only a single value was used throughout. A value between 0-1 was sampled and raised to the power of eight, resulting in a trend towards smaller values with a very few larger exceptions.

Other Parameters

Parameters which were originally varied but were not tuned in the final hyperparameter search include convolutional filters, Learning Rate and Optimizer (ADAM with default parameters was finally used), Loss function (for the Regression problem), numerical precision, batch size, and number of stocks to predict for. (The latter three ended up being bounded primarily by available GPU memory.)

Discussion of Experiments and Results

I've evaluated two objectives; one is a regression objective to predict the absolute change in value of the equity and the other is to distribute funds in a portfolio:

Regression Results

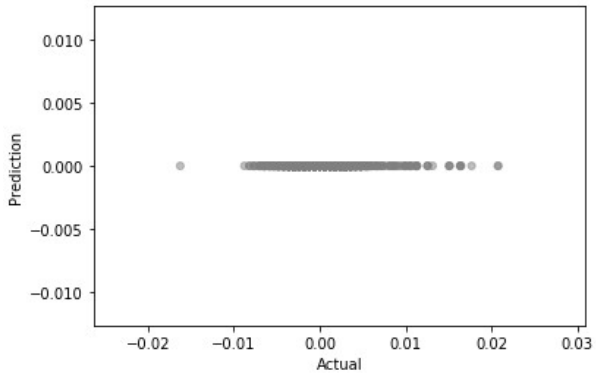
In the first phase I aimed to find a mechanism to extract useful features and generate an encoding of the market that could be relied on to predict price movement. While working on the regression problem I explored multiple loss functions including both standard and non-standard options. Mean squared error was ultimately selected.

Analysis of the data showed three important insights:

1. Stocks would often go a whole month without a single buying signal being detected. In fact in January and September 2018 not a single signal was detected in the entire month.
2. The resulting predictions were also not easy to translate to a recommendation on trading strategy and left unanswered questions on how to allocate a fixed sum of money and which of several stocks has the best probability of profit.

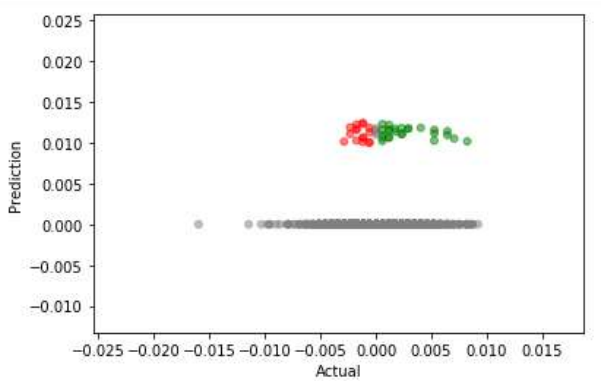
Overall, I found that I was able to make slightly useful predictions about a small number of samples and I achieved validation accuracy of 52.4% on average (over 12,950 predictions in 2018 (validation). Note that predictions approximately equal to 0 are considered no signal and thus not counted.

January 2018 (FULT), No useful predictions



	month	stock	TP	FP	TN	FN	X	Accuracy
10	1	JCOM	0	0	0	0	8125	NaN

April 2018 (FULT), 52 of 82 predictions correct



	month	stock	TP	FP	TN	FN	X	Accuracy
38	4	FULT	18	12	34	18	8108	0.634146

The purpose of this phase was simply to confirm that I had some useful information being extracted from the input data. The initial results indicate positive confirmation that the model is in fact learning something that is useful in prediction. The results are not directly usable for portfolio optimization which is why we need the next stage of project, a portfolio allocation model.

Allocation Problem

Modifications to Model Architecture:

In order to generate an output of a portfolio allocation I've replaced the final layers of the previously mentioned model with a new output layer with a SoftMax Activation resulting in an output vector summing to 1 with an element for each equity (this can be thought of as a weighting of each equity in proportion to how profitable it is expected to be).

The following 11 stocks with good (>50%) 2018 validation accuracy were selected to provide a representative portfolio:

MANH, CLI, FULT, CMC, GGG, KMT, ZBRA, OI, TCBI, GDOT, JCOM

Loss Function

The Loss function designed for portfolio allocation is designed to reward profitable portfolio allocation while severely punishing losses. As a result, the allocations will be slightly risk adverse and the degree of risk aversion is possible to tune through adjustment of the Beta parameter below.

$$L = -\sum_{n=0}^{s}\{(y_n * \hat{a}_n) - \beta min(0, y_n * \hat{a}_n)\}$$

n = index of an equity in the list of portfolio stocks(range: 0 to $s-1$)
$\mathbf{y}$ = column vector (length n) of stock price increases in next minute
y_n = Scalar true increase in stock price of equity n in next minute
$\hat{\mathbf{a}}$ = column vector (length n) of recommended fractional allocations.
$\hat{a}_n$ = Scalar recommended fractional allocation of portfolio funds to equity n
β = parameter for additional risk aversion.

The loss function in this case is the negative profit earned in a timestep minus a negative value of losses if any equities lost money in this step (resulting in an increase to loss). For my experiments Beta is set to 1 but could be adjusted upwards to make the resulting model less risk adverse or downwards to encourage profit maximization.

Results

For validation purposes the model was trained on three years of data from 2015 through 2017 and tested on 2018 data. Finally, when the results looked positive the model was re-trained on 2015 through 2018 and tested on 2019.

Returns of a $1000 initial investment over the validation and test years are plotted below:

Model trained 2015-2017, Results on 2018 Data

Model 2018 Return: 27.9%, Balanced Return: -9.1%

Model trained 2015-2018, Results on 2019 Data

Model 2018 Return: 73.6%, Balanced Return: 16.9%

Over the course of 2018 my model was able to theoretically outperform a balanced portfolio of the stocks being optimized before accounting for trading costs market impact and cost of crossing the bid-ask spread. In order to capture a fraction of the value from these recommended allocations an agent should be designed to handle execution in a more optimal way however the design and implementation of such an agent was out of scope for this project.

Error Analysis:

The model performs poorly in a couple of important cases and my analysis provides insight into the causes as well as potential optimization. During September-November 2018 for example a large loss is incurred. Analyzing the time series of underlying equities, one sees that each of the stocks in the sample portfolio decreased over that time making it impossible to prevent losses other than by removing money from the market, something the model does not currently allow.

More interesting is the findings in early 2019 (Jan-May) that the model's recommendations underperformed the balanced portfolio. Analyzing that period shows that the growth of the balanced portfolio primarily came from one equity, ZBRA which increased 53% in value from Jan 01 to April 8 and which our model allocated on average only 8.86e-04 % of the resources to. The likely root cause is that the ZBRA equity had not previously seen such rapid growth and as such no predictive signals had been learned to indicate that it may be a beneficial time to invest there, similar scenarios where a buying opportunity occur in

Conclusion

This project has demonstrated evidence that a deep learning approach to stock market portfolio allocation may theoretically provide returns in excess of a balanced portfolio across equities before accounting for trading costs, market impact and the cost of crossing the spread to take positions. The documented approach of applying a 2D convolutional network to an input of features of many hundreds of equities over time allows the model to generalize very well into the future with relatively strong performance up to 12 months after the training period.

The next logical step on this line of research is to incorporate the model into a Reinforcement Learning Agent which can learn when it may be profitable to place orders based on the information that the model can extract from the recent states of the market. Eventually even greater performance may be achieved by searching to optimize the architecture of the convolutional layers of the model as well as investigating the impact of a wider and more diverse set of features. Finally, the ability to hold a portion of the portfolio in cash promises an opportunity to improve overall results and mitigate the large losses we see during down markets.

It seems that although past performance is no guarantee of future results, if you look both deeply and widely it does offer some very useful and surprisingly robust information.

Contributions and References

All work for this project was done by Roy Justus with the guidance of Chris Waites.

Academic References

K. Khare, O. Darekar, P. Gupta and V. Z. Attar, "Short term stock price prediction using deep learning," *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, Bangalore, 2017, pp. 482-486.

L. Sayavong, Z. Wu and S. Chalita, "Research on Stock Price Prediction Method Based on Convolutional Neural Network," *2019 International Conference on Virtual Reality and Intelligent Systems (ICVRIS)*, Jishou, China, 2019, pp. 173-176.

A. Ariyo, A. O. Adewumi and C. K. Ayo, "Stock Price Prediction Using the ARIMA Model," *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*, Cambridge, 2014, pp. 106-112.

Y. Hu and S. Lin, "Deep Reinforcement Learning for Optimizing Finance Portfolio Management," *2019 Amity International Conference on Artificial Intelligence (AICAI)*, Dubai, United Arab Emirates, 2019, pp. 14-20.

T. Sanboon, K. Keatruangkamala and S. Jaiyen, "A Deep Learning Model for Predicting Buy and Sell Recommendations in Stock Exchange of Thailand using Long Short-Term Memory," *2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*, Singapore, 2019, pp. 757-760.

J. Eapen, D. Bein and A. Verma, "Novel Deep Learning Model with CNN and Bi-Directional LSTM for Improved Stock Market Index Prediction," *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, 2019, pp. 0264-0270.

Non-Academic References

Thushan Ganegedara, “Market Predictions with LSTM in Python” January 1st, 2020. Retrieved from the internet at: *https://www.datacamp.com/community/tutorials/lstm-python-stock-market*

Yacoub Ahmed, “Predicting stock prices using deep learning” Oct 11, 2019. Retrieved from the internet at: *https://towardsdatascience.com/getting-rich-quick-with-machine-learning-and-stock-market-predictions-696802da94fe*

Frameworks and APIs

TensorFlow 2.0:

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng.

TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

Keras:

François Chollet, Keras, 2015 available at: https://github.com/fchollet/keras

Alpaca Python API, available at: https://github.com/alpacahq/alpaca-trade-api-python

Polygon.io Market Data, available at: https://polygon.io/