CS230

Predicting Index Option Implied Volatility Surfaces

Final Report

James P Michels III
Department of Computer Science
Stanford University
kamujim@stanford.edu

Abstract

My goal is to train a network to predict the volatility surface of the QQQ options using the volatility surfaces of 10 of its most highly correlated components. This approach is an end to end approach and assumes the data understands the important relationships between the surfaces. It does not seek to understand what the surface should look like in a theoretical context or what a long term fair surface would look like. It seeks to understand the rules that the market is using to relate these surfaces in the recent past and into the near future. Additionally, the input set includes straddle information for all symbols in order to allow the model to focus on the shape of the surface without requiring it to also predict the expected future volatility of each stock or ETF. The model is able to predict a surface with very tight clustering around the expected value and with very few outliers.

1 Introduction

Market makers are the car dealers of the financial world. Their role is to buy inventory at wholesale prices and sell inventory at retail prices. When you trade in your car to buy a new one, the car dealer tries to offer you a price for your used car that will allow them to liquidate that inventory for a reasonable profit. They are not buying your used car because they expect its price to rise. They are looking to facilitate your desire to buy a new car without exposing themselves to unwanted losses when buying your old car. In the same way that understanding the used car market is important for car dealers, understanding the current rules that the market is using to value related options can help market makers avoid undesirable losses while they seek to liquidate the inventory they are acquiring through their market making activities.

2 Related work

In finance, models are typically built to generate alpha, which is a measure of excess return when compared to a benchmark. For example, the stock market's performance. The problem I am trying to solve is very niche. Instead of trying to create alpha, I am instead looking for a model to avoid negative alpha. As such, there is not a lot of literature related to previous efforts. However, "Gated neural networks for implied volatility surfaces"[1] and "Volatility Model Calibration With Convolutional Neural Networks"[2] both demonstrate the feasibility of applying neural network learning to the problem understanding implied volatilities. Gated neural networks for implied volatility surfaces shows that a neural network trained with 20 years of data for the S&P 500 can produce a volatility model that outperforms the widely used surface stochastic volatility inspired (SSVI). Volatility Model Calibration With Convolutional Neural Networks shows that a neural network can be used to replicate

CS230: Deep Learning, Spring 2020, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

the calibration of the Heston model which is then used to significantly speed up portfolio analysis tasks.

3 Dataset and Features

The raw dataset is a database of over 6 million implied volatility samples for the QQQ index and its 10 highest correlated components, AAPL, AMZN, CMCSA, CSCO, FB, GOOG, GOOGL, INTC, MSFT, and PEP. These were sampled every five minutes during regular trading hours. The data is all from the same distribution. The features used in the training/testing sets are "time to expiration", "moneyness", "at the forward price", "at the forward implied vol", and "implied vol". For this project, I wrote a "data cleaner" application. The raw dataset was first preprocessed by this data cleaner which verified that for each intended output sample there were at least 5 (a hyperparameter) different "time to expiration" sets each containing at least 8 (a hyperparameter) different "moneyness" samples. Samples that failed to meet these requirements were discarded at this stage. Sample units are collected by sampling all of the stocks at the same time. Sampling is repeated every 5 minutes throughout the normal trading session. The cleaned dataset was used to generate the dev (50,000 samples), test (50,000 samples), and training (remaining samples) sets. For each QQQ option sampled, we create our dataset by building a sample block for each input symbol. These sample blocks are stacked and reshaped into a vector of 2004 input features and 1 output.

Sample Unit (sampled at same time)
IV = Implied Volatility
timeExp = Time to expiration
stdDev = Moneyness
atfIV = At the forward implied vol
price = Forward price

Sample Block - 40 stacked sample units							
Sample Unit 1	Sample Unit 2	Sample Unit 3	Sample Unit 4	Sample Unit 5	Sample Unit 6	...	Sample Unit 40
IV	IV	IV	IV	IV	IV	...	IV
timeExp	timeExp	timeExp	timeExp	timeExp	timeExp	...	timeExp
stdDev	stdDev	stdDev	stdDev	stdDev	stdDev	...	stdDev
atfIV	atfIV	atfIV	atfIV	atfIV	atfIV	...	atfIV
price	price	price	price	price	price	...	price

IV <- The implied volatility feature of the QQQ sample becomes the output value.

QQQ	AAPL	AMZN	CMCSA	CSCO	FB	GOOG	GOOGL	INTC	MSFT	PEP
	Sample Block	Sample Block	Sample Block	Sample Block	Sample Block	Sample Block	Sample Block	Sample Block	Sample Block	Sample Block
Partial Sample Unit	...	...	...	...	...	...	...	...	...	...
timeExp	...	...	...	...	...	...	...	...	...	...
stdDev	...	...	...	...	...	...	...	...	...	...
atfIV	...	...	...	...	...	...	...	...	...	...
price	...	...	...	...	...	...	...	...	...	...

Input features = 2004 = 4 QQQ partial features + (10 symbols * 40 sample units per block * 5 feature per sample unit)

Output features = 1

Figure 1: Data set construction

4 Methods

The network is a deep, fully connected residual network[9] using Relu activation $f(x) = max(x, 0)$[11] for all layers except the output layer. The output layer uses activation $f(x) = x$. The loss is computed using $root\ mean\ squared\ error = \sqrt{(\frac{1}{n})\sum_{i=1}^{n}(y_i - x_i)^2}$. Adam optimization[10] is used for gradient decent learning. Multiple residual blocks are stacked for a total depth of 92 hidden layers of which 28 are fully connected layers. Dropout[12] is used in the residual blocks to reduce overfitting. The network is implemented using the Keras framework[3].

The goal of the model is to implement an end to end network capable of learning how to produce a useful volatility surface. The residual network design is used because it enables the construction of deep neural nets that are easier to train and more accurate while also resolving the vanishing gradient problem. Identity blocks are constructed to allow chunks of data to skip over fully connected layers

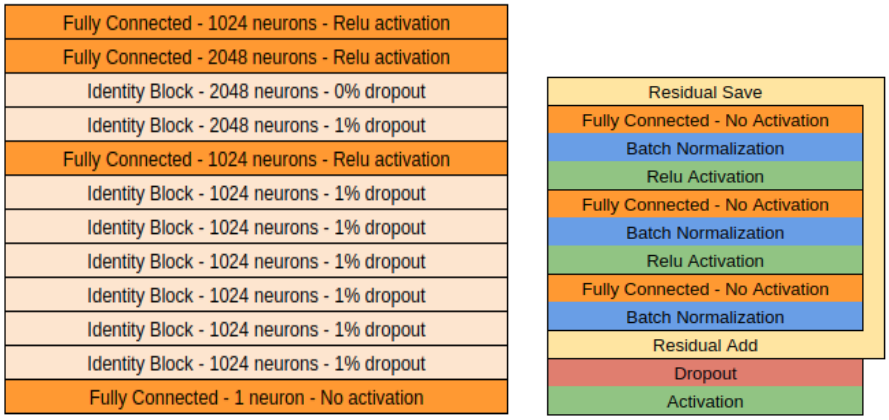

Figure 2: Network Diagram and Identity Block Diagram

while also allowing those layers to participate in learning. At the output layer, linear $f(x) = x$ activation is used to allow the learned features to directly pass forward their contribution through simple summation. In this way all of the inputs contribute to a proportion of the final value and that the network will have already learned how to weigh them. Adding a non-linear activation at the final node forces the accumulated learning to go through an unneeded transformation. I am using the idea of early stopping by evaluating the model against my dev set periodically and saving its parameters when the mean squared error for the dev set is lower than the previous best evaluation. I am also using an adaptive learning rate defined as...

$$learning\ rate = f(slope, learning\ rate) = \begin{cases} max_{\frac{learning\ rate}{1.25}, 0.000001}, & \text{if } slope \geq 0 \\ min_{(1.25\ learning\ rate), 0.001}, & \text{otherwise} \end{cases}$$

New samples are added daily. The adaptive learning rate allows the model to increase its learning rate when the new samples allow for faster forward progress. When the model absorbs the new information, the learning rate can the automatically decrease for fine tuning.

5 Experiments/Results/Discussion

I made a mistake early on by not recording results for all of my hyperparameter combinations. Since realizing this mistake, I have retrained some examples of different hyperparameter combinations that I thought were interesting, because they show why I stopped iterating with some hyperparameters. Since these models take several days to train, I was not able reproduce all of the hyperparameter combinations that I tried.

I experimented with different activations, specifically relu, sigmoid, and tanh. All of them worked to some degree, but when thing went bad, they went really bad with sigmoid and tanh. Relu seemed to fail more gracefully and intuitively made more sense to me. Because this is a regression problem, it didn't seem useful to me to have the activation outputs constrained the way sigmoid and tanh would. For these reasons, I focused on relu.

I found that larger mini batch sizes generally worked better, but I was limited on how large I could test. Because my model and dataset were large, Keras would crash due to memory exhaustion once the mini batch size became too large. In the end, I just used the largest mini batch size that I could without Keras crashing.

As mentioned above, I found that a learning rate 0.001 was good for training the model at first, but eventually, the slope of the loss over the last 10 epochs starts oscillating between positive and negative values. At this point, the only way I was able to create a negative slope for the loss was to decrease the learning rate. New data is added to the dataset daily. The adaptive learning rate algorithm described in the methods section automates these adjustments as new samples are integrated into the data sets.

When we learned about residual networks in the course, I realized that they would be extremely helpful with problem I was trying to solve. I scrapped my prior models and moved to the the residual architecture that I am using now. After a few iterations on the depth and number of hidden nodes, the model was producing errors (Figure 4) that were in an acceptable range.

Models with mean absolute error less than 0.008, negative Fisher kurtosis, and near zero skewness are useful for this problem. There are limits to how precisely implied volatility can be sampled due to issues such as underlyer market data jitter, bid ask spreads, and pricing model assumptions. Around 0.008 error, sampling noise starts taking over while the monetary value of the error becomes less important. A negative Fisher kurtosis indicates the errors are tightly clustered around mean. A near zero skewness suggests the errors are not overwhelmingly of the same sign. The model has been consistent at producing a surface (Figure 3, Figure 5, Figure 6) with tightly clustered errors and very few outliers.

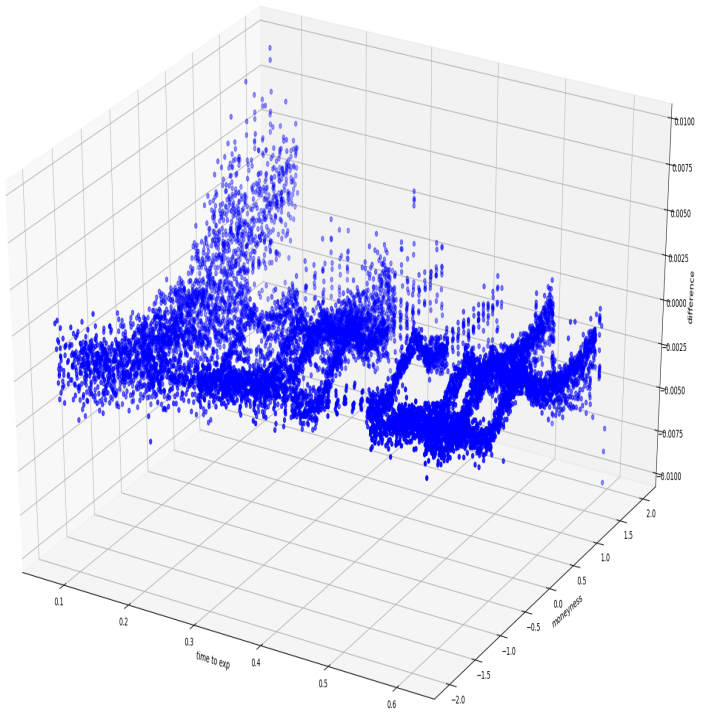

Figure 3: 3d view of the different between the predicted and expected volatility across the moneyness and time axes

As I iterated with methods for regularizing the model, I quickly discovered that weight decay regularization was preventing the model from learning. As you can see in Figure 4, L2 regularization with a penalty of just 0.000001 was enough to stop the model from learning. Dropout worked with very small drop rates of 1-2%, but the more commonly used drop rate of 20% (Figure 4) also stopped the model from learning.

Figure 4 shows that the results with 12 identity layers under performed the 8 identity layer model. In early attempts the 12 layer models did eventually achieve comparable results to the 8 layer model. However, as mentioned about, I had to retrain these models in order to present results. That said, the 8 identity layer model trains faster with acceptable errors.

Correlations in the stock market have a short shelf life. Every day introduces small changes to the relationships that have to be trained into the model. Data mismatch is the primary problem with this model. My goal in iterating on the number of identity blocks was to provide the network with enough depth to not only learn today's relationships, but also tomorrows. Having too many identity blocks does slow the model down, but the residual network design is still able to learn.

Identity Layers	Block 1-2 Nodes	Block 3+ Nodes	Dropout Rate	L2 Regularization	Batch Size	Mean Sq Error	Mean Abs Error	Kurtosis	Skewness
8	2048	1024	1%	None	32768	0.00003	0.007	-0.83	0.071
8	2048	1024	20%	None	32768	0.0166	0.108	0.78	0.027
12	2048	1024	1%	None	4096	0.000157	0.0094	1.57	0.76
8	2048	1024	0%	0.000001	16384	6.2662	2.4065	61.75	-7.219

Figure 4: Results of important hyperparameter combinations

6 Conclusion/Future Work

The model is able to learn the relationships between volatility surfaces. Each day new data is added to the dataset, while older data is removed. Transfer learning allows the model to quickly retrain to the new dataset. This rapid retraining allows it to evolve in near real-time with the underlying relationships. This allows the model to help answer the question, "What is normal now, given what we know about the current state of the market." This can help differentiate between changes in the market which are novel and changes that are fully explainable by relationships which have already been observed and learned.

I think the residual network concept played a significant role in the success of this model. This approach is an end to end approach. It expects the data to know what is important. Residual networks allow for a model that is overly deep to still effectively learn from the data. This also allows the model some degree of future proofing in case future data needs the extra depth.

Going forward, I'd like to study the activations to understand what the model thinks is important. However, as this is an evolving model, it would not be enough to study what is important today. I think it would be really important to study these activation over time. Are the same things always important? Are they cyclic?

I would also like to iterate further on the depth and number of hidden nodes. I was able to identify hyperparameters that did and didn't work, but I was not able to identify a gradient that could be use to guide hyperparameter iteration. Considering how slowly these networks train from random weights, it would useful if the network could be made smaller without sacrificing accuracy.

References

[1] Zheng, Yu & Yang, Yongxin & Chen, Bowei. (2019). Gated neural networks for implied volatility surfaces.

[2] Dimitroff, Georgi and Röder, Dirk and Fries, Christian P. (September 20, 2018). Volatility Model Calibration With Convolutional Neural Networks.

[3] 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. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems, Software available from tensorflow.org.

[4] J. D. Hunter. (2007). "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95

[5] Travis E. Oliphant. (2006). A guide to NumPy, USA: Trelgol Publishing

[6] Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. (2011). The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30

[7] John C. Hull, (2018). Options, Futures, and Other Derivatives

[8] Lawrence G. McMillan. (1993). Options as a Strategic Investment, Third Edition

[9] Christian Szegedy Wei Liu Yangqing Jia Pierre Sermanet Scott Reed Dragomir Anguelov Dumitru Erhan Vincent Vanhoucke Andrew Rabinovich. (2015). Going Deeper with Convolutions. Computer Vision and Pattern Recognition

[10] Diederik P. Kingma, Jimmy Ba. (2014). Adam: A Method for Stochastic Optimization

[11] Abien Fred Agarap. (2018). Deep Learning using Rectified Linear Units (ReLU)

[12] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. (1929-1958, 2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting

7 Appendix

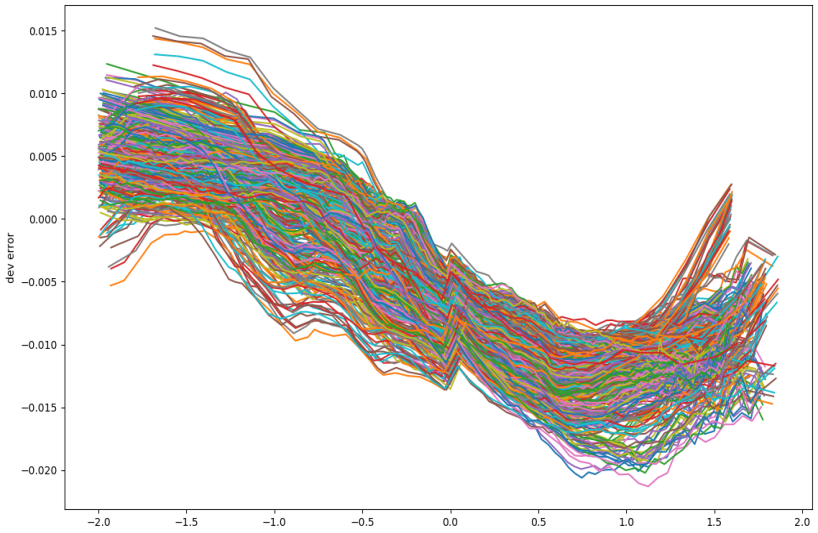

Figure 5: Multicolored view of the different between the predicted and expected volatility. The x-axis is the "moneyness" of the options. Each line is a formed using options sampled concurrently and with matching "time to expiration" values.

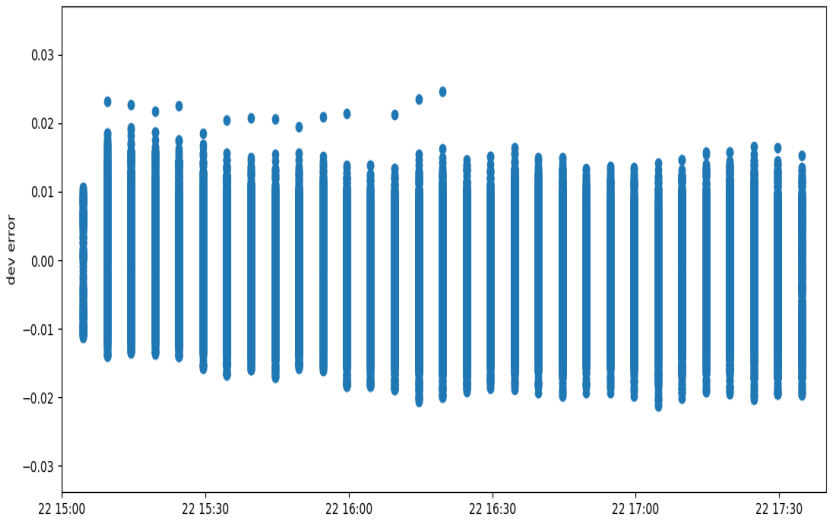

Figure 6: View of the different between the predicted and expected volatility. The x-axis is the time at which the samples were taken and reflect a 2 and a half hour period.

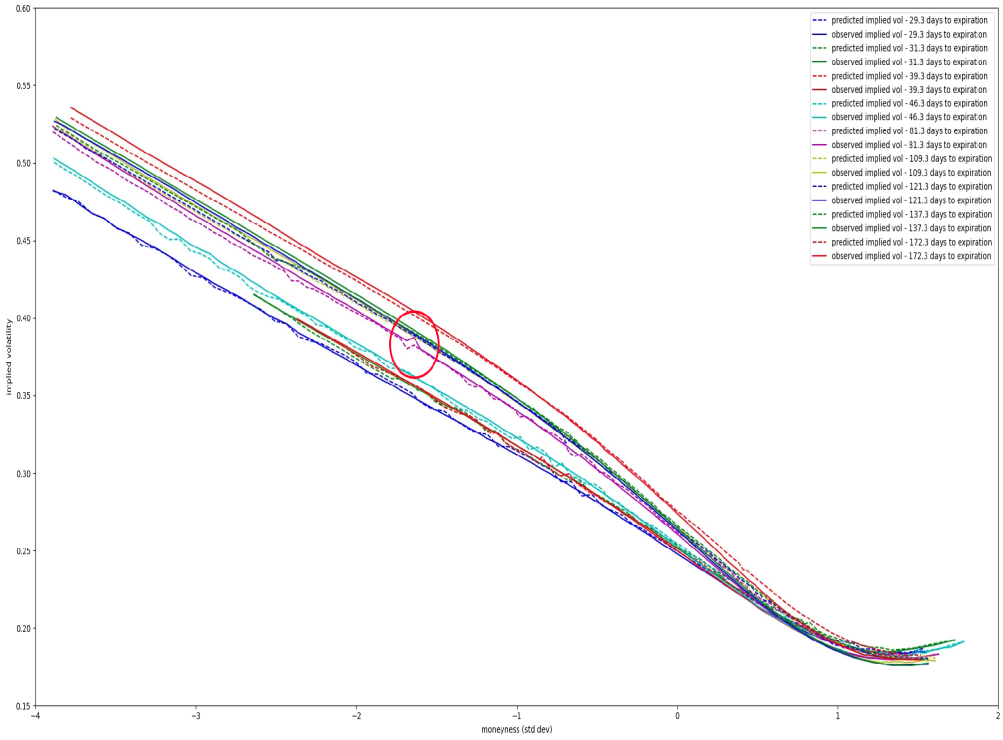

Figure 7: View of a deflection in the implied volatility surface cause by an order, combined with the model's prediction.