Mitigating Overfitting on Financial Datasets with Generative Adversarial Networks

FERNANDO DE MEER PARDO AND RAFAEL COBO LÓPEZ

FERNANDO DE MEER PARDO
is a data scientist at ETS Asset Management Factory in Madrid, Spain.
fdemeer@ets.es

RAFAEL COBO LÓPEZ
is a quantitative research manager at ETS Asset Management Factory in Madrid, Spain.
rcobo@ets.es

KEY FINDINGS

- Generative adversarial networks can capture the complex dynamics that govern many financial assets and produce realistic synthetic scenarios based on historic data.
- Synthetic financial scenarios can be used to enlarge training datasets in order to improve the accuracy and robustness of other deep learning models.
- Synthetic financial series may also be applied in other tasks such as model backtesting, risk analysis, or option pricing.

ABSTRACT: *Overfitting is an inevitable phenomenon when applying deep learning techniques to financial data, given the relative scarcity of available historical data and the ever-changing nature of financial series. This seemingly unavoidable pitfall has heavily impaired the application of many machine learning techniques, such as deep or reinforcement learning, to financial settings. Data augmentation can be an option to circumvent this, specifically generative adversarial networks (GANs). GANs are a type of neural network architecture that focuses on sample generation. Through adversarial training, the GAN can implicitly learn the underlying structure inherent to the dynamics of financial series and acquire the capacity to generate scenarios that share many similarities to those seen in the historic time series. In this article, the authors propose a data augmentation technique using the Wasserstein GAN with gradient penalty and show how training deep models on synthetic data mitigates overfitting, improving their performance on test data when compared to models trained solely on real data.*

TOPICS: ***Statistical methods, simulations, big data/machine learning****

Machine learning techniques have been revolutionary in recent years due to their unparalleled proficiency at solving many tasks previously thought impossible to automatize. The list includes problems such as image recognition, ad recommendation, sample ranking, fraud detection, medical diagnosis, speech recognition, and so on.

However, applications to financial settings such as price forecasting, risk analysis, or portfolio construction have been impaired due to the unique characteristics of financial data. Financial time series are ever changing by nature and seemingly chaotic. In addition, available historical data are far too scarce for most machine learning applications and impossibly scarce for advanced approaches such as reinforcement learning.

*All articles are now categorized by topics and subtopics. **View at PM-Research.com.**

All of these hardships make overfitting an unavoidable pitfall when applying machine learning techniques to financial time series. Models are heavily fit to perform well on historic data but generalize poorly on unseen test data; see Lopez de Prado et al. (2016). This is where data augmentation comes into play.

There are many ways in which one may generate synthetic data. In the case of financial time series, stochastic models have historically been used to model asset prices, starting with the classic Black–Scholes model; numerous extensions have been developed to account for certain consistent characteristics observed in the market. Stochastic volatility models were devised to account for volatility clustering phenomena, Lèvy models were devised to account for jumps in price series and so on.

However, trying to model the market through hypothesis reflected in a stochastic equation in this way is certainly challenging. One must first recognize a consistent phenomenon in asset prices and then be able to reflect it on a stochastic equation in a suitable way (with additional changes then making pricing algorithms more and more involved). If we were to generate synthetic series in this way, we would be introducing bias in the generation process because we would be conditioning the generation to our set of hypothesis, which stems from our limited understanding of the price process.

Generation procedures based on historic data have been recently explored in order to reproduce more complex market behaviors because they do not require modeling asset prices via a set of stochastic equations. They consist of a series of feature extraction and feature reproduction routines. Franco-Pedroso et al. (2018a, b) provide examples of such procedures. This approach to modeling, however, is still vulnerable to bias because the choice feature extraction and feature reproduction routines radically condition the structure of the generated series. In the spirit of Da Silva and Shi (2019), we turn to a novel machine learning technique that completely automatizes the feature extraction and feature reproduction routines of our generation, generative adversarial networks (GANs).

GANs are a type of neural network architecture that focus on sample generation. Through adversarial training, the GAN can learn the underlying structure of input data and become able to generate samples very similar to those of the training dataset. This is especially useful in the case of high-dimensional objects in which the dimensions are heavily interdependent, such as images, music, and, in our case, financial time series.

The synthetic time series created this way can be used to enlarge training sets and improve the accuracy or robustness of other deep learning models. Presenting the model with a much larger number of realistic scenarios makes the model seek a more stable compromise because the parameter configurations sought during training have to accommodate a much bigger and diverse training set. In addition, being able to train a third deep learning model is in itself a proof of the quality of the generated time series. It assures us that they may be useful in other applications.

In this article, the authors introduce the GANs and the modifications needed to generate synthetic financial time series and illustrate how they may be used to improve the behavior of another classification/forecasting deep learning models.

GENERATIVE MODELING

A GAN is an example of a generative model. The term refers to any model that takes a training set, consisting of samples that are considered to be drawn from a distribution $\mathbb{P}_{data}$, and learns to represent an estimate of that distribution somehow. The result is a probability distribution $\mathbb{P}_{model}$. GANs do not explicitly estimate $\mathbb{P}_{model}$ but rather learn how to produce samples from it; see Exhibit 1.

Generative Adversarial Networks

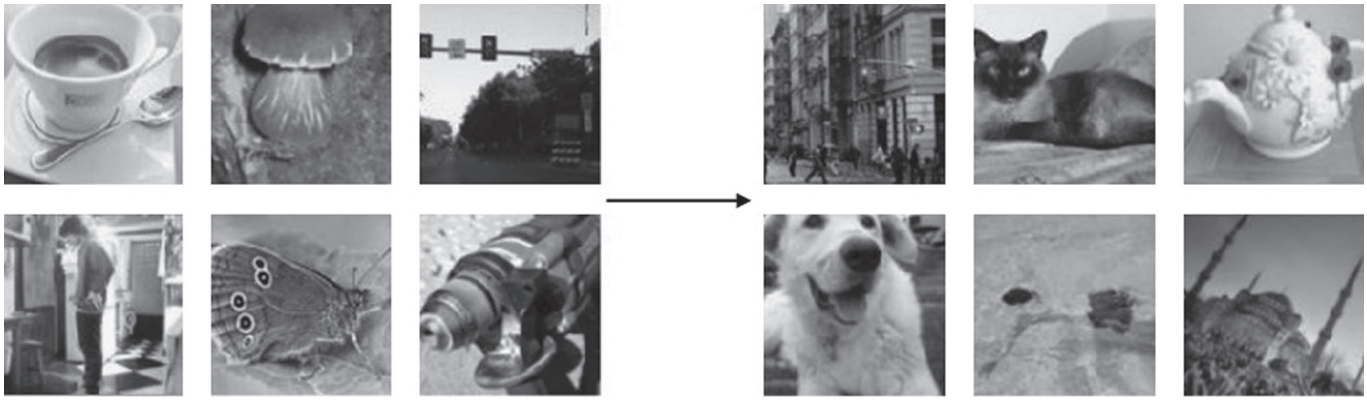
The basic idea behind GANs is to train two neural networks in an adversarial game. One of the neural networks is called the *generator.* The generator creates samples that are intended to come from the same distribution as the training data. The other neural network is the *discriminator.* The discriminator examines samples to determine whether they are real or fake.

The generator takes $\mathbf{z}$, a random normal noise vector, as input and produces fake data samples. Afterwards, the discriminator is fed a batch of real samples and another of fake ones and must learn to classify them correctly as real or fake.

Both neural networks have loss functions that are defined in terms of both networks' parameters. The discriminator aims to minimize $L^{(D)}(\mathbf{w}^{(D)}, \mathbf{w}^{(G)})$ and must do

EXHIBIT 1

An Ideal Generative Model Would Be Trained on Examples as Shown on the Left and Then Be Able to Create More Examples from the Same Distribution as Shown on the Right



Source: ImageNet

so while controlling only its own parameters,[1] $\mathbf{w}^{(D)}$. Equivalently, the generator aims to minimize $L^{(G)}(\mathbf{w}^{(D)}, \mathbf{w}^{(G)})$ and must do so while controlling only $\mathbf{w}^{(G)}$. Because each network's loss depends also on the other network's parameters, which it cannot control, this scenario is most straightforwardly described as a game rather than as an optimization problem.

GAN Training

The GAN training process consists of iterative stochastic gradient descent updates. On each step, two minibatches are sampled: a minibatch of x values from the dataset and a minibatch of $\mathbf{z}$ values that are run through the generator. Then, two gradient steps are made: one updating $\mathbf{w}^{(D)}$ to reduce $L^{(D)}$ and one updating $\mathbf{w}^{(G)}$ to reduce $L^{(G)}$. In both cases, it is possible to use the gradient-based optimization algorithm of choice (such as Adam or RMSProp; see Kingma and Ba (2014) and Tieleman and Hinton (2012), respectively).

Training a GAN, however, is far from trivial. The actions of two different networks influence the training process and as a result, nonconvergence is a very likely scenario. Mode collapse, also known as the *Helvetica scenario*, is a phenomenon that arises when training results in failure. It may manifest as a generator that maps several or even all input $\mathbf{z}$ values to the same output point or output region or as a generator that misses certain regions of the target distribution $\mathbb{P}_{data}$.

Wasserstein's GAN with Gradient Penalty

Many of the challenges faced by GAN training are the result of the topologies induced in the space the dataset lies in by the loss functions $L^{(G)}$ and $L^{(D)}$ used in the original GAN setup, which are too coarse and cause the gradients of the losses to vanish. Wasserstein GAN (WGAN) gradient penalty (GP) is a GAN setup proposed in Gulrajani et al. (2017) that addresses these problems. It uses a loss function based on the Wasserstein distance that induces a much weaker topology that allows for a more robust training process.

Wasserstein's Distance

Let $\mathcal{X}$ be a compact metric set. Let Prob($\mathcal{X}$) denote the space of probability measures defined on $\mathbf{x}$. The earth-mover (EM) distance or Wasserstein-1 is then defined as

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\left[\left\|x - y\right\|\right] \quad (1)$$

in which $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are, respectively, $\mathbb{P}_r$ and $\mathbb{P}_g$, and $||\cdot||$ is a metric.

[1] Neural network parameters are also referred to as weights, hence the $\mathbf{w}$ notation.

Intuitively, $W(\mathbb{P}_r, \mathbb{P}_g)$ indicates how much mass must be transported from x to y in order to transform the distributions $\mathbb{P}_r$ into the distribution $\mathbb{P}_g$, thinking of the distributions as *piles of earth*.

It is intractable to exhaust all the possible joint distributions in $\Pi(\mathbb{P}_{data}, \mathbb{P}_{model})$[2] to compute $inf_{\gamma \sim \Pi(pdata, pmodel)}$ in Equation 1, thus the authors in Arjovsky, Chintala, and Bottou (2017) proposed a transformation of the formula based on the Kantorovich–Rubinstein duality:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \frac{1}{K} \sup_{||f||_L \leq K} (\mathbb{E}_{\mathbf{x} \sim \mathbb{P}_r}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_g}[f(\mathbf{x})]) \quad (2)$$

The function f in the new form of the Wasserstein metric is demanded to satisfy $||f||_L \leq K$, meaning it should be K-Lipschitz continuous.

In the WGAN, the discriminator model is used to learn a set of parameters $\mathbf{w}$ to find a good $f_{\mathbf{w}}$, and the loss function for both networks is configured as measuring the Wasserstein distance between $\mathbb{P}_{data}$ and $\mathbb{P}_{model}$. The gradient penalty is added so that it forces the discriminator toward being a 1-Lipschitz function.

$$L = \mathbb{E}_{x \sim pdata}[D_{w(D)}(\mathbf{x})] - \mathbb{E}_{x \sim pdata(\mathbf{z})}[D_{w(D)}(G_{w(G)}(\mathbf{z}))] + \lambda \mathbb{E}_{\hat{\mathbf{x}} \sim \mathbb{P}_{\hat{\mathbf{x}}}}\left[\left(\left\|\nabla_{\hat{\mathbf{x}}} D_{w(D)}(\hat{\mathbf{x}})\right\|_2 - 1\right)^2\right]$$

During training the discriminator aims to maximize the loss function above and, hence, approximates the Wasserstein distance. The generator on the other hand tries to minimize it by modifying the fake data distribution. As the loss function decreases during training, the Wasserstein distance gets smaller and the generator model's output grows closer to the real data distribution. For a further theoretical analysis of GAN training, see De Meer Pardo (2019). The main takeaway for the work presented here is that a WGAN-GP equipped with neural networks able to handle vector-shaped data can efficiently be trained to generate realistic financial time series as we will see shortly.

Conditional GAN

As we have already mentioned, in a standard GAN setup the input to the generator is always defined as $\{\mathbf{z}^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_{\mathbf{z}}$ ($\mathbb{P}_{\mathbf{z}}$ being an easy-to-sample distribution, usually $\mathcal{N}(0, 1)$) per batch in which m is the batch size. This way, we simply ask a well-trained G to generate a random sample from $\mathbb{P}_{data}$. It is possible, however, to condition this generation on additional information that is also added as an input to G.

In the conditional setup, training is conducted in the same way, only adding the conditional information as input to both G and D as part of each sample. There are few restrictions on the way this conditional information may be added as input to the networks; we choose to concatenate it along with the noise input to the generator. We will employ a WGAN-GP in a conditional setup to generate a Chicago Board of Options Exchange Volatility Index (VIX) series based on the starting price, which will be added to the input.

Synthetic Time-Series Generation

In order to illustrate how to generate financial time series with WGAN-GP, we choose a financial time series of interest, the daily closing prices of the VIX for the period February 1, 2004–April 4, 2019, pictured in Exhibit 2. February 1, 2004 is chosen as the start of the dataset because it was when the VIX started being calculated with its current methodology.

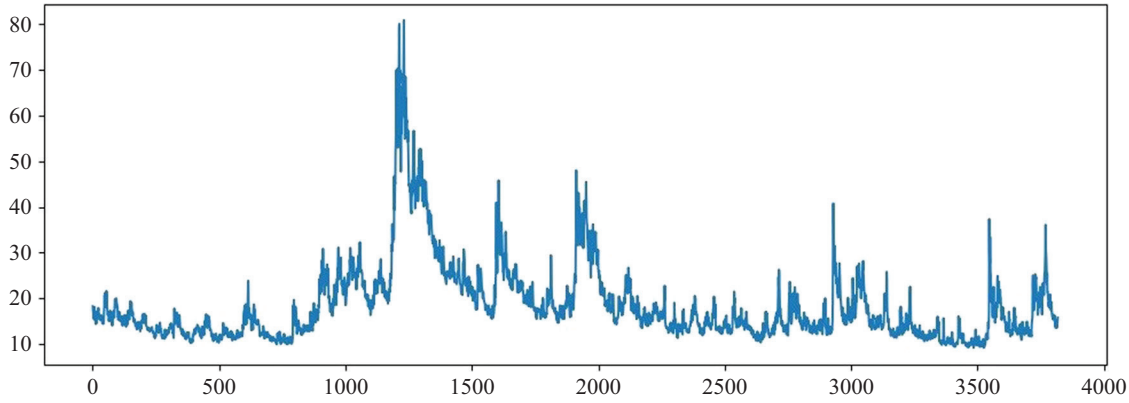
As explained by the Chicago Board of Options Exchange, intraday VIX values are based on snapshots of S&P 500 Index option bid–ask quotes every 15 seconds and are intended to provide an indication of the fair market price of expected volatility at particular points in time. There are many ways in which investors may choose to participate in the VIX such as VIX-based options, futures, or exchange-traded funds, and it is considered one of the most important ways in which investors may interact with market volatility. Because of this, the behavior of the VIX has been widely studied, and many volatility models have been proposed to model its dynamics.

We build a dataset by taking segments of 1,000 days from the time series of closing prices, rolling forward 10 days at a time. From these closing prices, we calculate daily returns. In this way, we get a dataset consisting of snapshots with different behaviors of the VIX over time. We employ our WGAN-GP in a conditional setup; the VIX tends to be mean reverting; hence, the price level at which the series starts is important. If at the start of the series the VIX is valued at 60 points, it is more likely

[2] Keep in mind that there is an infinite amount of joint distributions with the same marginals.

EXHIBIT 2

VIX Daily Closing Prices for the Period February 1, 2004–April 4, 2019



to descend in the following days, given its past behavior. In order to make our WGAN-GP conditional, as explained in the last section, we simply concatenate to each vector of 999 returns its corresponding starting price level so that they may be fed together as inputs to the discriminator. On the other hand, we generate returns given a starting price level by concatenating said price to the noise vector and feed them together as inputs to the generator.

We build the generator and discriminator networks by concatenating convolutional layers inspired by the deep convolutional GAN architecture (see Radford, Metz, and Chintala 2015) but adapted to one-dimensional vectors rather than matrices.

After training, the model can produce samples of $\mathbb{P}_{model}$ by sampling from a vector $\mathbf{z} \sim \mathbb{P}_{\mathbf{z}}$ ($\mathcal{N}(0,1)$ in our case) and by randomly sampling starting prices from the dataset. We get a series of returns as the generator output and calculate prices evolving from the starting price. Two such of the synthetic series produced can be visualized in Exhibits 3 and 4.

Perhaps a more visual way to see how the WGAN-GP produces synthetic time series is an output interpolation. As the trained generator of a GAN is a continuous function, by going along a continuous path in the input space (of vectors $\mathbf{z}$ or initial prices) we can generate samples in a continuous manner. A video illustrating the outputs resulting from an interpolation of the initial price from 40 to 15 and a fixed noise seed $\mathbf{z}$ can be seen at https://vimeo.com/354844686.

The results are visually promising. The generated time series seem to share the behavior of the VIX; they have sudden spikes as well as curve-shaped downtrends. Quantifying their quality is, however, complex because there is no clear consensus on the way to quantitatively score samples. GANs are harder to evaluate than other generative models because it is not possible to estimate the likelihood of generated samples because we do not explicitly know $\mathbb{P}_{data}$ (we only have samples from it), and estimating it is not a feasible task because it lies in a high-dimensional space, and its dimensions are highly dependent. Many of the difficulties involved in evaluating generative models are described in Theis, van den Oord, and Bethge (2015), Lucic et al. (2017), and Borji (2018).

In our case, we want to mitigate the overfitting of other deep models, so instead of comparing $\mathbb{P}_{model}$ and $\mathbb{P}_{data}$, we will simply take a successful training of a third model on synthetic data as a measure of performance. If the GAN can generate data able to train another model, then we consider the synthetic series to be representative of $\mathbb{P}_{data}$ and our generation process successful. This approach was first proposed in the case of time series in Esteban, Hyland, and Rätsch (2017).

Mitigating Overfitting

In this section, we show how to mitigate the overfitting of a deep learning model applied to financial data on a classification/forecasting task. We first present the model employed, the residual network (ResNet), and then the task along with the results.

EXHIBIT 3
Synthetic VIX Sample 1

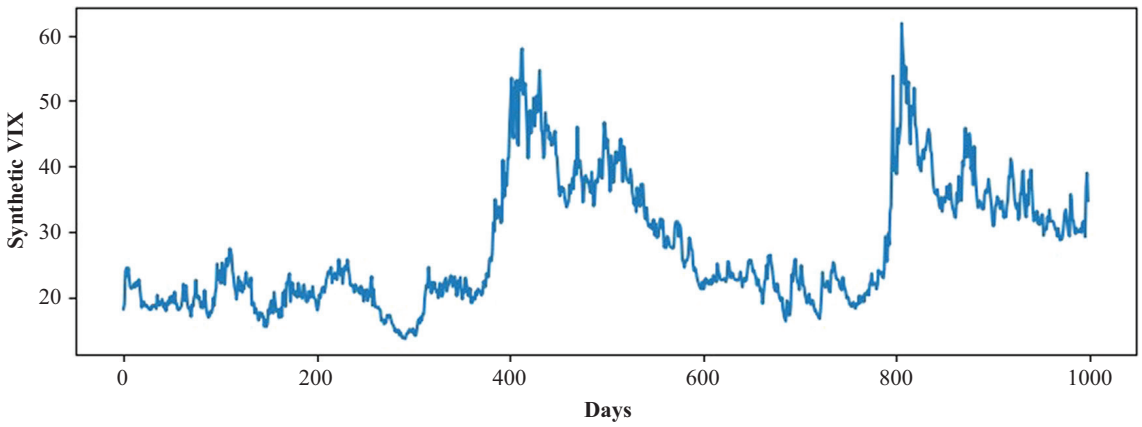
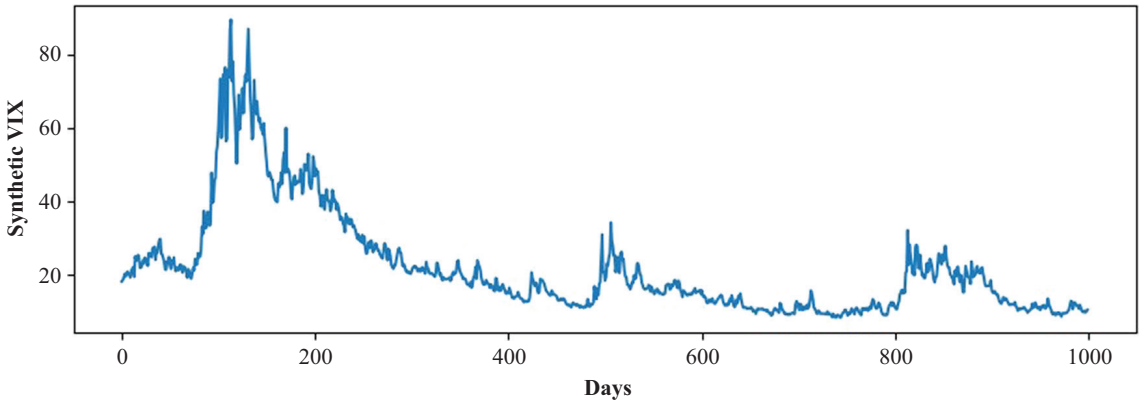


EXHIBIT 4
Synthetic VIX Sample 2



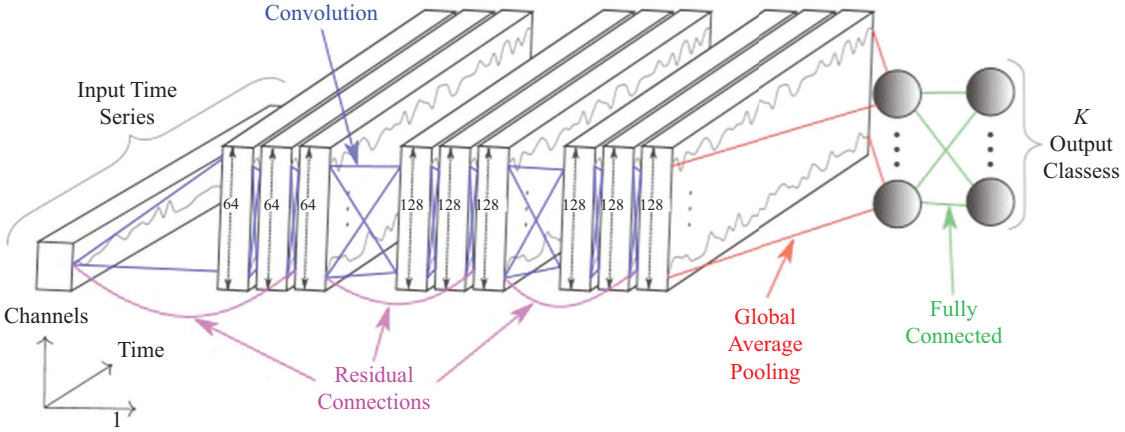
ResNet

Presented in Wang, Yan, and Oates 2016, the ResNet architecture is a special type of neural network designed for time-series classification (TSC). The main characteristic of ResNets is the shortcut residual connection between consecutive convolutional layers. A linear shortcut is added to link the output of a residual block to its input, thus enabling the flow of the gradient directly through these connections. This makes training a ResNet much easier by reducing the vanishing gradient effect.

The network is composed of three residual blocks followed by a global average pooling layer and a final softmax classifier, whose number of neurons is equal to the number of classes in the dataset. Each residual block is first composed of three convolutions whose output is added to the residual block's input and then fed to the next layer. The number of filters for all convolutions is fixed to 64, with the rectified linear unit activation function that is preceded by a batch normalization operation. In each residual block, the filter's length is set to 8, 5, and 3, respectively, for the first, second, and third convolution. The structure is illustrated in Exhibit 5.

In a thorough empirical comparison of many TSC models (see Ismail Fawaz et al. 2018) the ResNet was found to be the best performer in a wide variety of datasets.

EXHIBIT 5
ResNet Architecture



Forecasting VIX Peaks

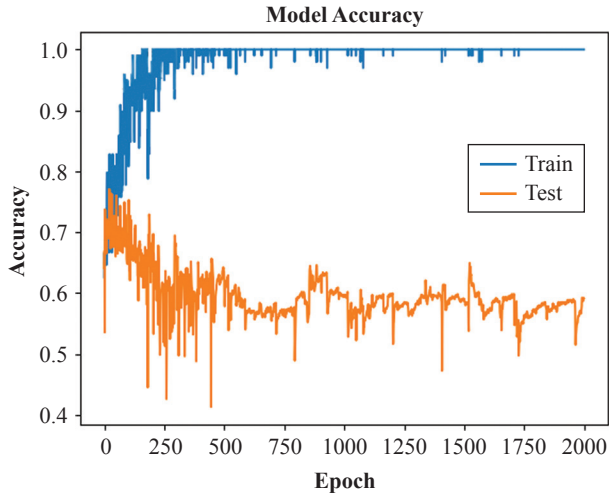
The VIX tends to be a mean-reverting time series and stays in the 10–15 point range for long periods of time. Hence, the options with strike prices in this range are very popular. Call options in particular are very competitively priced due to the mean-reverting behavior the VIX presents; being able to predict VIX peaks could be very valuable. Given the last 30 prices of the VIX, can the ResNet predict at what level the VIX will be 20 trading days from today?

First, we perform training in the classic way, splitting the dataset in a training set from January 2, 2004 to October 20, 2015 and a test set from October 21, 2015 to April 4, 2019. We split each dataset in periods of 30 days (nonoverlapping in the training set and overlapping in the test set) and then divide the samples into two classes:

1. **Class 0:** If the VIX price 20 days after the last day of each period was below 15.
2. **Class 1:** If the VIX price 20 days after the last day of each period was above 15.

We conduct training for 2,000 iterations. The accuracy curves and confusion matrix can be seen in Exhibits 6 and 7. The accuracy stabilizes close to 60%, and the curve suggests some overfitting to the training data. These results are coherent. Financial data are ever changing by nature, and for this reason, overfitting

EXHIBIT 6
Original Accuracy



is to be expected. The sudden drops visible in the accuracy plot on the test set are, however, particularly worrisome, especially those that do not correlate with a drop in performance on the training set because they hint that our model may not generalize well on unseen data.

In order to mitigate this overfitting, we now want to generate synthetic VIX price series so that the ResNet may be presented with a bigger number of scenarios during training. To do this, we train the WGAN-GP from the previous section using training

EXHIBIT 7
Original Confusion Matrix

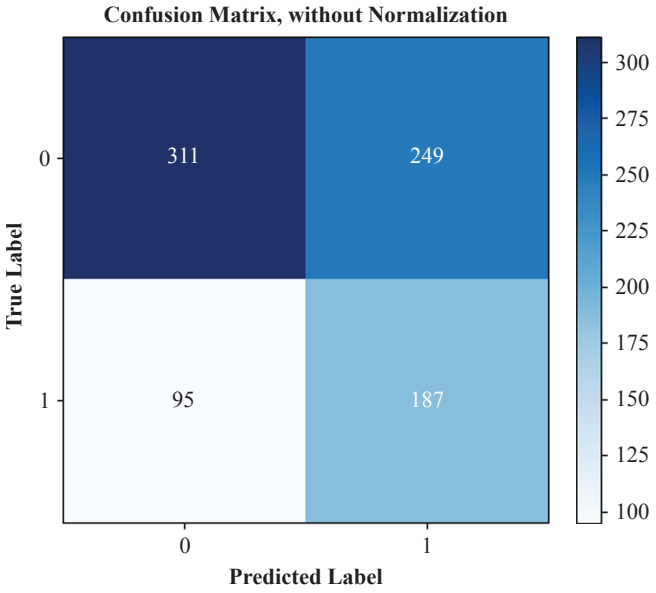
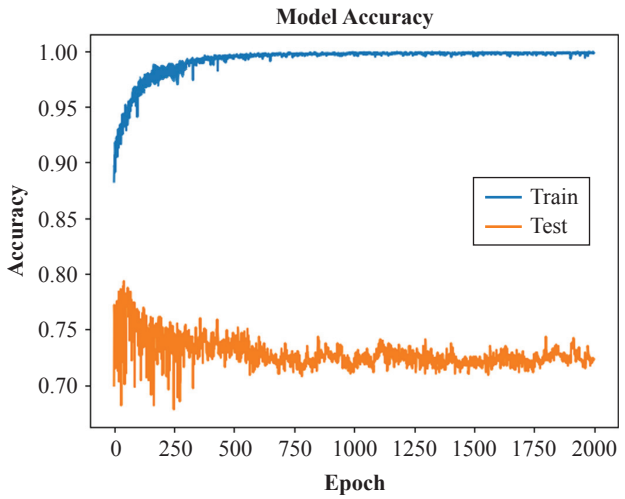
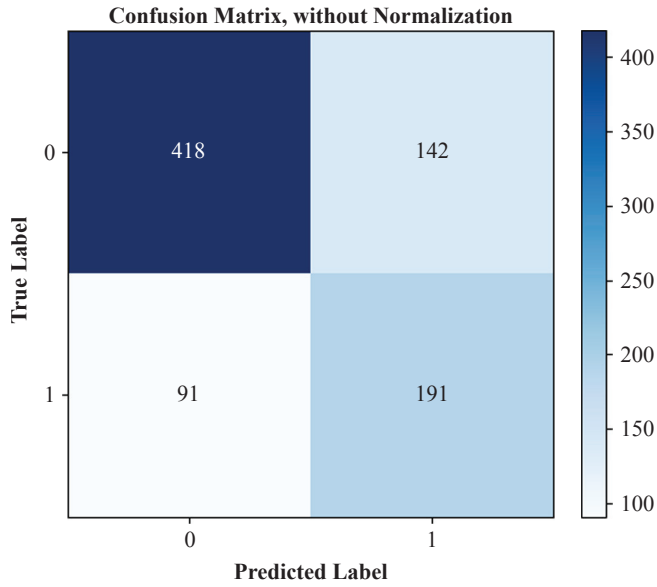


EXHIBIT 8
Accuracy Plot Resulting from the Enlarged Training Set



data only (meaning the GAN is never in contact with test data) and then sample 1,000 synthetic scenarios from $\mathbb{P}_{model}$ so that we can enlarge the training set. Finally, we train the ResNet on the enlarged training set. The accuracy plot and confusion matrix resulting from training on the enlarged training set can be seen in Exhibits 8 and 9.

EXHIBIT 9
Confusion Matrix Resulting from the Enlarged Training Set



We can see that the enlarged training set improves overall accuracy because it now stabilizes around 70% and the robustness of the model because the accuracy on the test set no longer presents any drops. The performance on class **1** is significantly improved as well, jumping from 42% to 53%. The synthetic scenarios produced by our GAN mitigate overfitting because they force the model to compromise during training in order to accommodate a much bigger set of scenarios, which makes the model generalize better. This behavior is consistent among training runs; see Appendix A for additional training instances.

This result is reassuring in multiple ways because it proves both the quality of our synthetic series and one of their many applications.[3]

[3] The necessary code to reproduce this experiment can be found in the following GitHub Repository: https://github.com/FernandoDeMeer/Mitigating-Overfitting-Experiment.

CONCLUSIONS

In this work we have shown how GANs may be used to create realistic financial scenarios that mitigate the overfitting suffered by other deep learning models.

With the same test set, enriching the training set by means of a GAN improves the performance of the presented model when compared to training the model solely on real data. In addition, it makes the parameter configurations found during training more robust, improving the generalization capabilities of the presented model.

Finally, many potential applications of synthetic financial series are worth mentioning. Their use could be explored regarding model backtesting, risk analysis, or option pricing. Such applications could open up further research paths.

APPENDIX A

ADDITIONAL TRAINING RUNS FOR THE VIX EXPERIMENT

EXHIBIT A1

Additional Accuracy Plot Resulting from Training on the Original Training Set

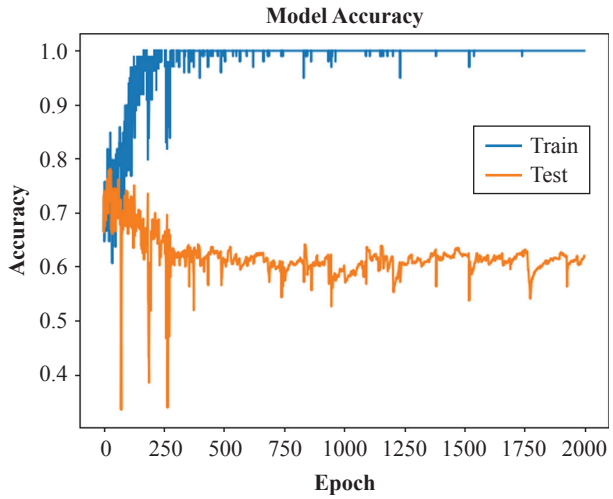
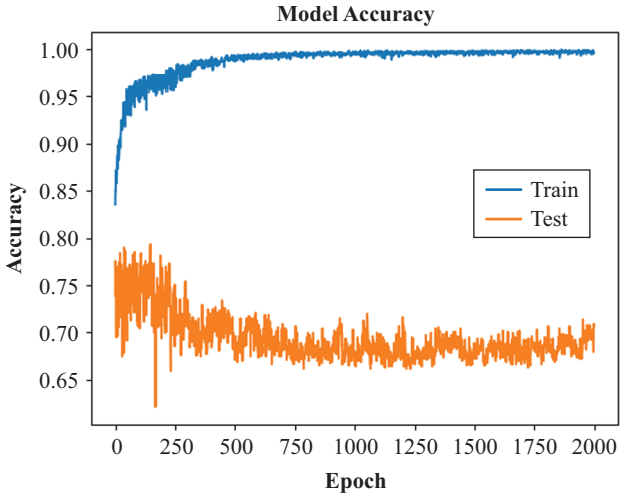


EXHIBIT A2

Additional Accuracy Plot Resulting from the Enlarged Training Set



REFERENCES

Arjovsky, M., S. Chintala, and L. Bottou. 2017. "Wasserstein GAN." *arXiv e-prints* arXiv:1701.07875.

Borji, A. 2018. "Pros and Cons of GAN Evaluation Measures." *arXiv e-prints* arXiv:1802.03446.

Da Silva, B., and S. S. Shi. 2019. "Towards Improved Generalization in Financial Markets with Synthetic Data Generation." *arXiv e-prints* arXiv:1906.03232.

De Meer Pardo, F. "Enriching Financial Datasets with Generative Adversarial Networks." Master's thesis, Delft University of Technology, the Netherlands, 2019.

Esteban, C., S. L. Hyland, and G. Rätsch. 2017. "Real-Valued (Medical) Time Series Generation with Recurrent Conditional GANs." *arXiv e-prints* arXiv:1706.02633.

Franco-Pedroso, J., J. Gonzalez-Rodriguez, J. Cubero, M. Planas, R. Cobo, and F. Pablos. 2018a. "Generating Virtual Scenarios of Multivariate Financial Data for Quantitative Trading Applications." *arXiv e-prints* arXiv:1802.01861.

——. 2018b. "The ETS Challenges: A Machine Learning Approach to the Evaluation of Simulated Financial Time Series for Improving Generation Processes." *arXiv e-prints* arXiv:1811.07792.

Gulrajani, I., F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. 2017. "Improved Training of Wasserstein GANs." *arXiv e-prints* arXiv:1704.00028.

Ismail Fawaz, H., G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. 2018. "Deep Learning for Time Series Classification: A Review." *arXiv e-prints* arXiv:1809.04356.

Kingma, D. P., and J. Ba. 2014. "Adam: A Method for Stochastic Optimization." *arXiv e-prints* arXiv:1412.6980.

Lopez de Prado, M., D. Bailey, J. Borwein, A. Salehipour, and Q. Zhu. 2016. "Backtest Overfitting in Financial Markets." *Automated Trader* 39.

Lucic, M., K. Kurach, M. Michalski, S. Gelly, and O. Bousquet. 2017. "Are GANs Created Equal? A Large-Scale Study." *arXiv e-prints* arXiv:1711.10337.

Radford, A., L. Metz, and S. Chintala. 2015. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks." *arXiv e-prints* arXiv:1511.06434.

Theis, L., A. van den Oord, and M. Bethge. 2015. "A Note on the Evaluation of Generative Models." *arXiv e-prints* arXiv:1511.01844.

Tieleman, T., and G. Hinton. "Lecture 6.5—RmsProp: Divide the Gradient by a Running Average of Its Recent Magnitude." In *Neural Networks for Machine Learning* COURSERA, 2012.

Wang, Z., W. Yan, and T. Oates. 2016. "Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline." *arXiv e-prints* arXiv:1611.06455.

To order reprints of this article, please contact David Rowe at d.rowe@pageantmedia.com or 646-891-2157.

ADDITIONAL READING

Neural Networks in Finance: *Design and Performance*

Irene Aldridge and Marco Avellaneda

The Journal of Financial Data Science
https://jfds.pm-research.com/content/1/4/39

ABSTRACT: *Neural networks have piqued the interest of many financial modelers, but the concrete applications and implementation have remained elusive. This article discusses a step-by-step technique for building a potentially profitable financial neural network. The authors also demonstrate a successful application of the neural network to investing based on daily and monthly financial data. The article discusses various components of neural networks and compares popular neural network activation functions and their applicability to financial time series. Specifically, use of the tanh activation function is shown to closely mimic financial returns and produce the best results. Incorporating additional inputs, such as the S&P 500 prices, also helps improve neural networks' forecasting performance. Longer training periods deliver strategies that closely mimic common technical analysis strategies, such as moving-average crossovers, whereas shorter training periods deliver significant forecasting power. The resulting neural network–based daily trading strategies on major US stocks significantly and consistently outperform the buy-and-hold positions in the same stocks.*

Enhancing Time-Series Momentum Strategies Using Deep Neural Networks

Bryan Lim, Stefan Zohren, and Stephen Roberts

The Journal of Financial Data Science
https://jfds.pm-research.com/content/1/4/19

ABSTRACT: *Although time-series momentum is a well-studied phenomenon in finance, common strategies require the explicit definition of both a trend estimator and a position sizing rule. In this article, the authors introduce deep momentum networks—a hybrid approach that injects deep learning–based trading rules into the volatility scaling framework of time-series momentum. The model also simultaneously learns both trend estimation and position sizing in a data-driven manner, with networks directly trained by optimizing the Sharpe ratio of the signal. Backtesting on a portfolio of 88 continuous futures contracts, the authors demonstrate that the Sharpe-optimized long short-term memory improved traditional methods by more than two times in the absence of transactions costs and continued outperforming when considering transaction costs up to 2–3 bps. To account for more illiquid assets, the authors also propose a turnover regularization term that trains the network to factor in costs at run-time.*