

# Meta-Learning Generative Adversarial Networks for Extrapolating Nonlinear Dynamic Stochastic Systems.

Case Study: Price Forecasting of Volatile Assets for use in Algorithmic Trading.

Jashua Luna

2022

--	--	--

**Abstract**



For Catherine.

This is what I was doing when I missed your calls (I'm sorry please call me back).

Also, I love you.



## Table of Contents

## Introduction

Financial markets, in particular cryptocurrencies which are highly volatile, have random walk properties. Furthermore, the fundamental properties which affect an asset's price are many and attempts to measure and model them might result in changes escaping the scope of the measurement. Despite these random walk properties, the change in price of an asset observes a constant distribution so the price of an asset can be effectively modelled as a stochastic process. The famous Black-Scholes equation describes the price of an asset as one that “follows a geometric Brownian motion with constant drift and constant volatility” (Zhang 2) which satisfies this differential equation.

$$dS = \mu S dt + \sigma dW \quad (1)$$

(Zhang, 2)

Where  $S$  is price and  $W$  is a Wiener process of Brownian motion, and  $\mu$  and  $\sigma$  are constants denoting drift and volatility respectively. The Black-Scholes equation is known to have some limitations, however, notably its assumption of market volatility as a constant property.

In their 2000 paper, Cars H. Hommes models financial markets as a nonlinear adaptive belief system with multiple variables relating to types of investors and price vectors; stating, “evolutionary adaptive systems with heterogeneous agents using competing trading strategies [are] a natural nonlinear world full of homoclinic bifurcations and strange attractors” (Hommes 6).

$$X_{t+1} = F(X_t; n_{1t}, \dots, n_{Ht}; \lambda; \delta_t; \epsilon_t) \quad (2)$$

(Hommes, 6)

Where  $F$  is a nonlinear mapping,  $X_t$  is a vector of prices,  $n_{jt}$  is the fraction or weight of investors of type  $h$ ,  $\lambda$  is a vector of parameters and  $\delta_t$  and  $\epsilon_t$  are noise terms (Hommes 6).

A problem with these solutions is the assumption of some market structure. Particularly in an adaptive belief market, a structure which may exist at one time will likely not exist in another. Abstractions can be made through which an analytical solution might be reached but this could ultimately prove fruitless in an efficient market. The model must change as quickly if not faster than the market.

$$S_{t+1} = f(S_t, S_{t-1}, \dots, S_{t-n}) \quad (3)$$

$$S_{t+1} = \Phi(\Theta_k, X) \quad (4)$$

Where  $X$  is a vectorized sample of  $S$  and  $\Theta_k$  is a set of parameters.

Assuming the distribution of possible future states to be finite, the price of an asset can be described as a finite state system for which there is a function which precisely determines future states from information of current and past states. The function described by phi and theta can accurately model the system while the system dynamics theta trains on are constant.

Be it slippage or localized panic or euphoria, the price of a volatile asset often oscillates around a “true” price in what will be henceforth referred to as noise.

$$Price = IdealPrice + Noise \quad (5)$$

These high frequency undulations in price can not be profited from because of the long time it takes to enter and exit trades; primarily resulting from request-response lag and time to fill orders. This can be mitigated by placing only market orders, but this might result in slippage which in turn affects the accuracy of a model.

## Concept

As a solution to stochastic, deterministic system with unknown inputs, a neural network is developed which can learn how the expected distribution of future data relates to the known distribution of present and past data. This way, a likely prediction can be used to implement trades. This has the added benefit of not relying on precision as it is likely that the future price of an asset would immediately diverge from the predicted price the moment a trade was enacted on the predicted information. Generative adversarial networks learn to replicate the stochasticity of data (Isola 7) and as such are appropriate for this solution (7).

$$\Phi(\Theta_k, \mathcal{D}_k, \hat{X}_{t-1}) \rightarrow \hat{X}_t \quad (6)$$

$$\Theta_k = \arg \min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G) \quad (7)$$

$$\min_{\Theta} \sum_{k_o}^k \mathcal{L}(\Theta - \alpha \nabla_{\Theta} \mathcal{L}(\Theta, \mathcal{D}_k^S), \mathcal{D}_k^Q) = \min_{\Theta} \sum_{k_o}^k \mathcal{L}(\Theta', \mathcal{D}_k^Q) \quad (8)$$

(Eq. 7 Isola 3, Eq. 8 Finn 3)

Where  $\mathcal{D}_k$  is a windowed set of price data samples between  $t$  and  $t - w_s$  ( $w_s$  is window size) subsampled from the set containing all of the asset’s price data,  $\Theta_k$  is its corresponding set of parameters,  $G$  is a generator which attempts to produce “real” data, and  $D$  is a discriminator which trains to distinguish “real” data from that produced by the generator.

For temporal stability between predictions, the network’s previous prediction is also input. In accordance with the adaptive efficient market hypothesis, the network must be routinely actualized such that it keeps up with the changing system. This is facilitated by the meta learning model described in (8) which ensures the model can quickly and effectively train on new tasks as they arise; changing market conditions are analogous to new tasks. Finally, the proposed model takes the form of a nonlinear mapping function which maps a vector of parameters  $\Theta$ , a windowed set of price observations  $\mathcal{D}$ , and its previous prediction  $\hat{X}_{t-1}$  to a new price prediction  $\hat{X}_t$  (6).



## Proof of Concept

Imperative to the design of this neural network is that the change in price between candles can be reasonably described as a stochastic process, independent of market conditions. To validate this assertion, one hundred samples of two thousand candles were randomly selected from the price of Bitcoin between the years 2021 and 2022; these years were selected because they had constant and reasonable trading volume throughout. Probability distributions were then created for each of these samples using histograms, and events with a probability of occurrence below  $1e-3$  were discarded.

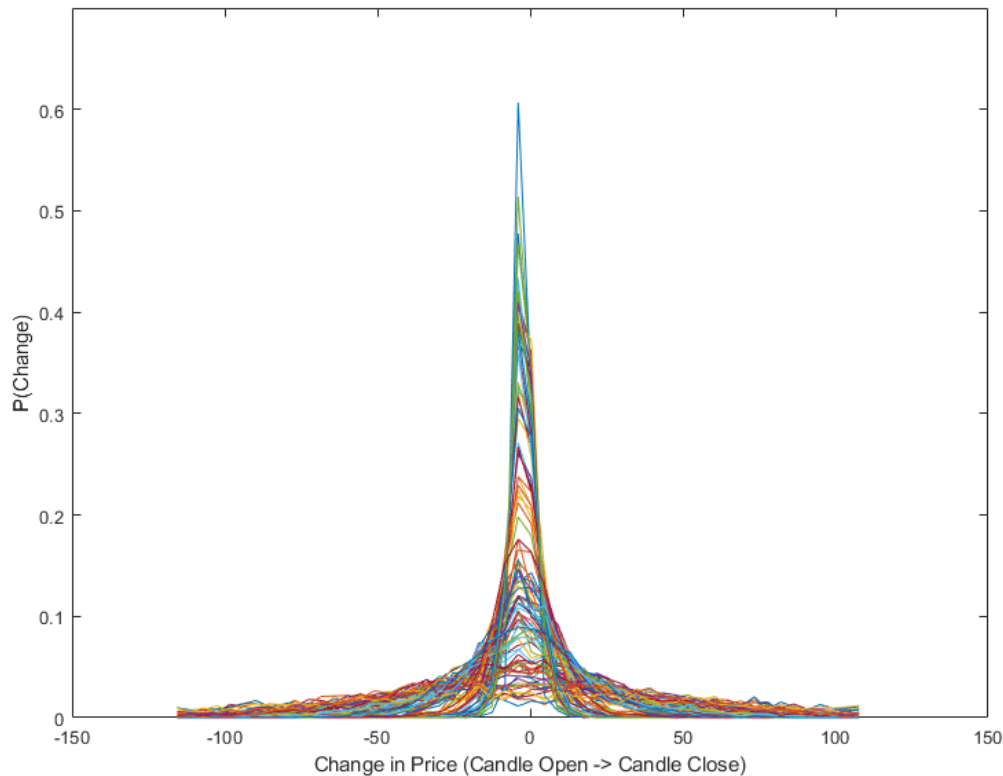


Figure 1 Change in Price per Candle vs The Probability it Occurs

Figure 1 shows the probability distributions for the one hundred samples. With ten thousand samples like the ones used for figure 1, subsample mean was determined to have mean 0.0107 and standard deviation 0.7823. Subsample standard deviation was found to have mean 30.7794 with standard deviation 24.4113. The extent to which the assertion holds true is ultimately determined by the model's performance, but from figure one, it can at least be concluded that the dataset does observe a relatively constant mean around zero with reasonable probability that change of price is less than twenty-five dollars, for any subsample.

Despite similarities in distribution, market conditions do influence changes in price, and this is evident when analyzing the spectral content of a sample. To illustrate the effect of market conditions on a sample's spectral content, one high energy sample, and one low energy sample were selected for comparison. The samples' energy was calculated as the sum of the sample's squared fast Fourier transform; from a set of a hundred samples of a hundred candles, the samples

with the highest and lowest energy readings were selected. The high-energy event corresponded to a high-volume trading scenario where within the hundred sampled candles, price changed by over one thousand USD. In the low-energy event, change in price across the entire sample was less than fifty USD.

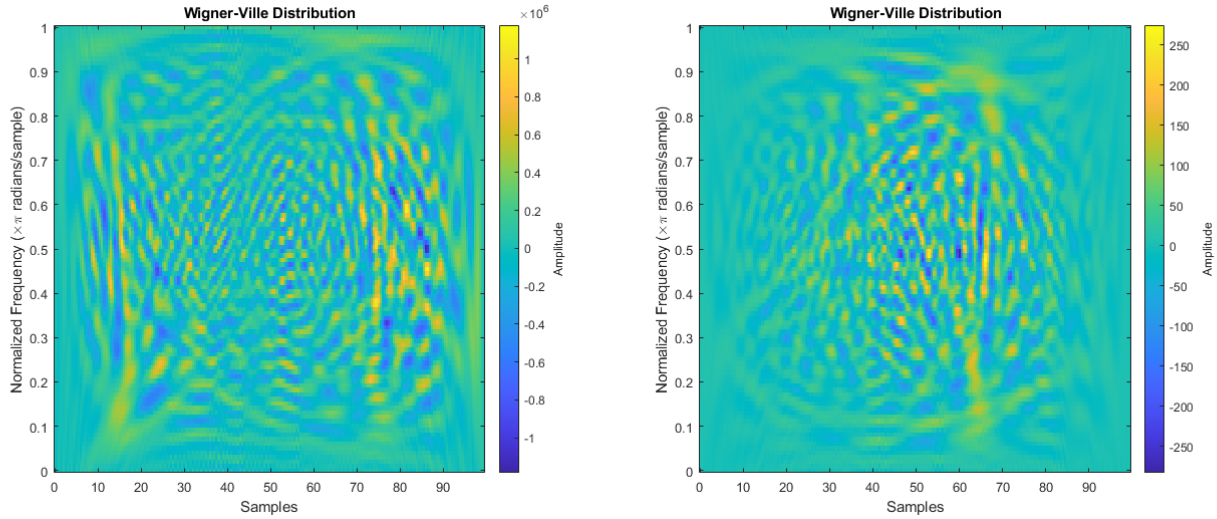


Figure 2 Wigner-Ville Distributions for High and Low Energy Events

Figure 2 compares the Wigner-Ville distributions for the selected high and low energy events. The one on the left is high-energy. There are more visible structures in the high-energy distribution than in the low-energy distribution, and the two have dissimilar shapes. Figure 3 shows the

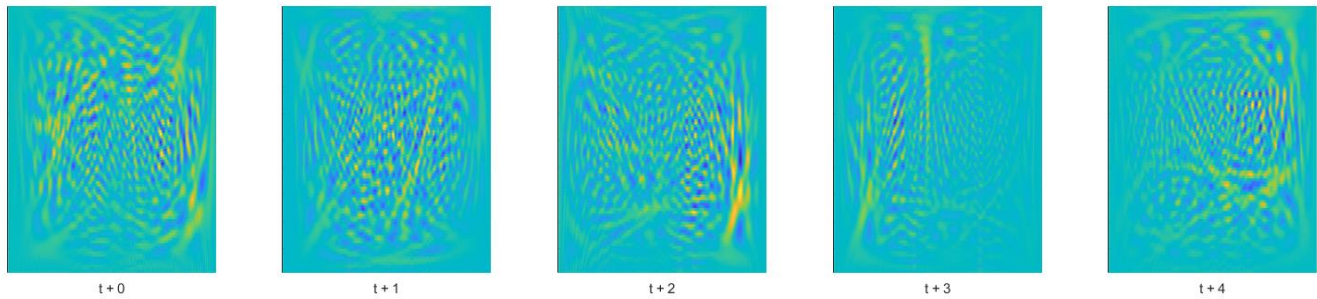


Figure 3 Wigner-Ville Distributions of Overlapping Consecutive Samples

Wigner-Ville distributions of five consecutive overlapping samples. Each sample contained data from one hundred candles and shared thirty candles with the following sample. Even while closely grouped in time the spectral content of each sample varies from the next. By grouping several overlapping samples, a set of samples is made from which the network – which will learn how changes in the structure of samples' spectral content relate to the system's future states – will use to predict. Because no two sets of samples are the same, the mathematical model for this network treats these sets as tasks; how the model learns the inter-sample relationships and their impact on future system states is where meta-learning takes place.

## Implementation

The price of an asset is observed at a regular frequency and samples having  $n$  observations are created. To reveal underlying trends, and encode inputs with frequency-information, samples are upsampled by the continuous wavelet transform (cwt).

$$X_{k,\psi,t} = \mathbb{Cwt}(X_{k,t}) \quad (10)$$

$$Y_{k,\psi} = \mathbb{Cwt}(X_{k,t+1}) \quad (11)$$

$$\mathcal{D}_k = \{X_{k,\psi,t}, X_{k,\psi,t-1}, \dots, X_{k,\psi,t-n}\} \quad (12)$$

For the network to learn system properties unique to the sample time, a set of wavelet-transformed signals is created by retaining the upsampled samples for a specified window. The meta learning model then learns features between different sample times (with varying market conditions) and determines which are relevant when faced with a “new” system state. Generative-Adversarial networks are implemented because they learn to replicate a dataset’s stochasticity which is desirable when the system has random-walk properties but maintains relatively uniform change-of-state distributions.

$$\begin{aligned} \Theta_G, \Theta_D = \arg \min_G \max_D \sum_{k_o}^k & (\alpha \mathcal{L}_{GAN}(G(\Theta_G, \mathcal{D}_k, Y_{k,\psi}), D(\Theta_D, Y_{k,\psi}))) \\ & + \lambda \mathcal{L}_{L1}(G(\Theta_G, \mathcal{D}_k, Y_{k,\psi})) \end{aligned} \quad (13)$$

$$\begin{aligned} \mathcal{L}_{GAN}(G(\Theta_G, \mathcal{D}_k, Y_{k,\psi}), D(\Theta_D, Y_{k,\psi})) = & E_Y[\log D(\Theta_D, Y_{k,\psi})] + \\ & E_{\mathcal{D}_z, z}[\log(1 - D(\Theta_D, G(\Theta_G, \mathcal{D}_k, Y_{k,\psi} + \zeta, z)))] \end{aligned} \quad (14)$$

$$\lambda \mathcal{L}_{L1}(G(\Theta_G, \mathcal{D}_k, Y_{k,\psi})) = \mathbb{E}_{X,Y,z} [\|Y_{k,\psi} - G(\Theta_G, \mathcal{D}_k, \mathcal{D}_z, Y_{k,\psi}, z)\|_1] \quad (15)$$

Instead of the network’s previous prediction, during training the expected output plus a noise vector  $\zeta$  is input. Additionally, a noise set  $\mathcal{D}_z$  and noise vector  $z$  is input so the adversarial network does not produce deterministic outputs (Isola 3). Equations 12-14 are the equations detailed in Isola et Al’s Image-To-Image Translation with Conditional Adversarial Networks modified to such that they accept sets of wavelets transformed data (as described in equation 12) as inputs. These provide a general overview of the network’s learning targets though the implemented architecture contains several sublayers some of which have their own learning targets. All the learning targets for the generator’s layers are compounded in the loss function which ultimately becomes that described in eq. 12. Where a generator sublayer has a unique learning target, its loss function is described in the section detailing that layer and the compounded loss function which governs the generator’s learning is described in equation 0.

## Generator Architecture

The implemented network is a generator-discriminator pair with several targets for learning. The generator is comprised of eight layers each with their own sublayers. In no order, the generator's layers are a scaling layer, a subsampling layer, a reweighting and detrending layer, a wavelet layer, a set-encoding layer, an estimate-encoding layer, a prediction layer, and an inverse-detrending layer. The generator accepts one dimensional inputs of the sampled change in price signal, and outputs one dimensional extrapolated predictions of that same price signal. For temporal stability, the generator's previous estimate is also input. While backpropagation is performed on all the layers simultaneously, loss is calculated for certain layers independently of the others before it is compounded for updating the gradients. This results in some changes to the loss functions detailed in equations 13 and 15.

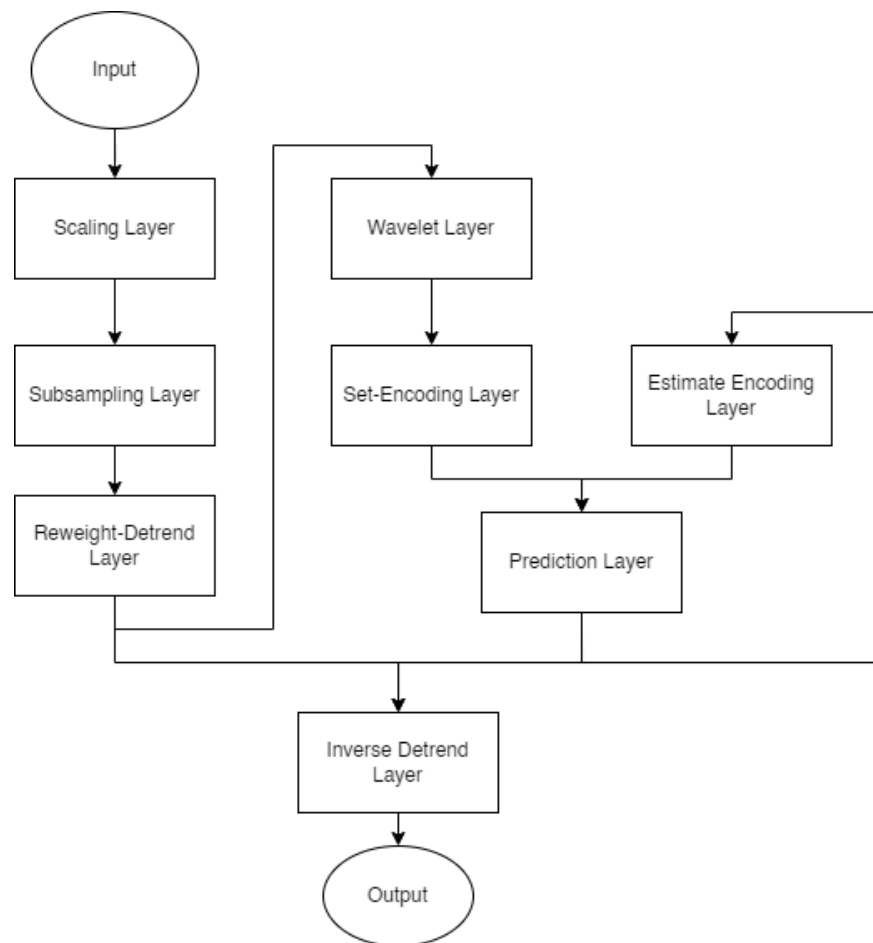


Figure 4 Generator Architecture Overview

The discriminator is essentially a condenser network trained with patch-Gan loss but since it accepts one dimensional samples of the price signal, its inputs undergo a transformation by the continuous wavelet transform before convolution. This will ensure that the samples the generator produce have similar frequency content to real samples.

## Scaling Layer

This layer scales a large sample of candles and scales the data such that all values are approximately in the range between -0.5 and 0.5. The number of candles this layer accepts as input depends on subsample window size, number of subsamples, and overlap between subsamples and is calculated with equation 16.

$$nCandles = WindowSize + (nSubsamples - 1) * (WindowSize - Overlap) \quad (16)$$

$$SF = \max|x| - \min|x| \quad (17)$$

$$X_{scaled} = X * SF^{-1} \quad (18)$$

Where  $x$  is a vector of candle data containing  $nCandles$ . At the time of writing, this layer scales samples by dividing each element by a scale factor; the range of the absolute values of the sample (equation 17). A scaling operation is useful for a deep network because it permits the network to learn patterns in the shape of the data, independent of the data's magnitude.

## Subsampling Layer

The subsampling layer rearranges the scaling layer's output into a two-dimensional array of dimensions  $[WindowSize, nSubsamples]$  whose rows each are a unique sample which overlaps the subsequent sample as dictated by the *Overlap* parameter. This layer is what converts a linear sample into a dataset sample and prepares the data for upscaling via the wavelet transform.

$$SSL(X_k) \rightarrow \mathcal{D}_k \quad (19)$$

## Reweight-Detrend Layer

Before data can be upsampled with the continuous wavelet transform, some steps must be taken to prevent loss of information and to maximize the accuracy of the final prediction. The continuous wavelet transform convolves a signal with a sliding wavelet at changing dilations to reveal the signal's frequency information; each dilation corresponds with a specific wavelength. To focus the wavelet, transform on higher frequency oscillations, the continuous wavelet transform algorithm applies a detrending operation to the data before convolving with wavelets. However, it does not return information about the detrend which would later affect the network's ability to predict future states.

Furthermore, the dataset in question has random-walk properties which skew sample statistics and severely reduce the network's accuracy in prediction. Figure 5 compares the short time Fourier transforms (STFT) of raw (left) and reweighted (right) price data. In the raw data's STFT, some high energy bands can be seen which indicate the presence of outlying points of data. Samples which contain outlying datapoints have vastly different shapes from regular samples, often

appearing as flat lines with one or two peaks. A reweighting operation is therefore necessary to remove or mitigate the effects of these outlying datapoints.

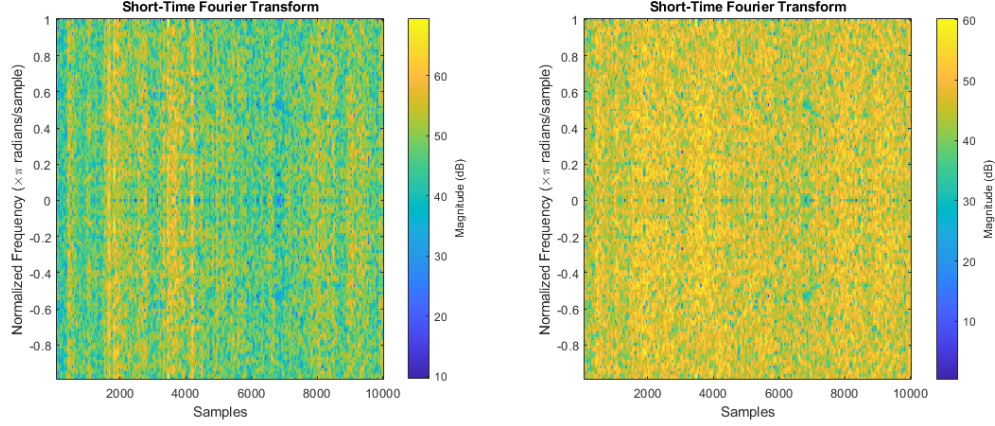


Figure 5 Comparison of Short Time Fourier Transform of Reweighted and Reference Samples.

To Produce this figure, the short time Fourier transform was taken on ten thousand raw and reweighted samples. The STFT was computed with an 80-point Blackman window, and an overlap of twenty datapoints between samples. A Blackman window was selected for its superior time localization when compared to other windows (wavelet design paper), which is good for isolating outlying events. Data was reweighted using Huber’s weight function in the steps outlined in 0.

This reweight-detrend layer seeks to identify and mitigate the effects of outlying datapoints, while applying its own detrend to the data, and returning information about these operations for use in other layers. Subsampled data is first passed through an encoder whose output is then used to generate reweighting and detrending information in parallel. The subsampled data is then reweighted and detrended.

As illustrated in figure 6, the encoder sublayer uses repeating blocks of two consecutive one dimensional convolutions with relu activations, followed by a max pooling and a batch normalization layer. The encoder sublayer’s output is stored for use in the inverse-detrend layer. The detrend-decoder and reweight-decoder sublayers have the same architecture, repeating blocks of transposed one-dimensional convolutions with tanh activations and batch normalization layers. The reweight-decoder’s output is then combined with the layer’s original input via element wise multiplication, then element-wise summed with the detrend-decoder’s output to produce this layer’s output.

$$\Theta_{RWDTL} = \operatorname{argmin} \mathcal{L}_{RWDTL}(\Theta_{RWDTL}, \mathcal{D}_k, \mathcal{D}_{ref}) \quad (20)$$

$$RWDTL(\Theta_{RWDTL}, \mathcal{D}_k) \rightarrow A, B \quad (21)$$

$$\mathcal{L}_{RWDTL} = \mathcal{L}_{Huber}(\mathcal{D}_k \odot A + B, \mathcal{D}_{ref}) \quad (22)$$

The reweight-detrend layer has its own learning target, designed to optimize its performance with the rest of the network. This layer trains to minimize Huber loss between this layer’s output and an ideal output. The ideal output is empirically determined by reweighting the sample with Huber’s weight function then in series applying the continuous and inverse continuous wavelet transforms (See equation 29).

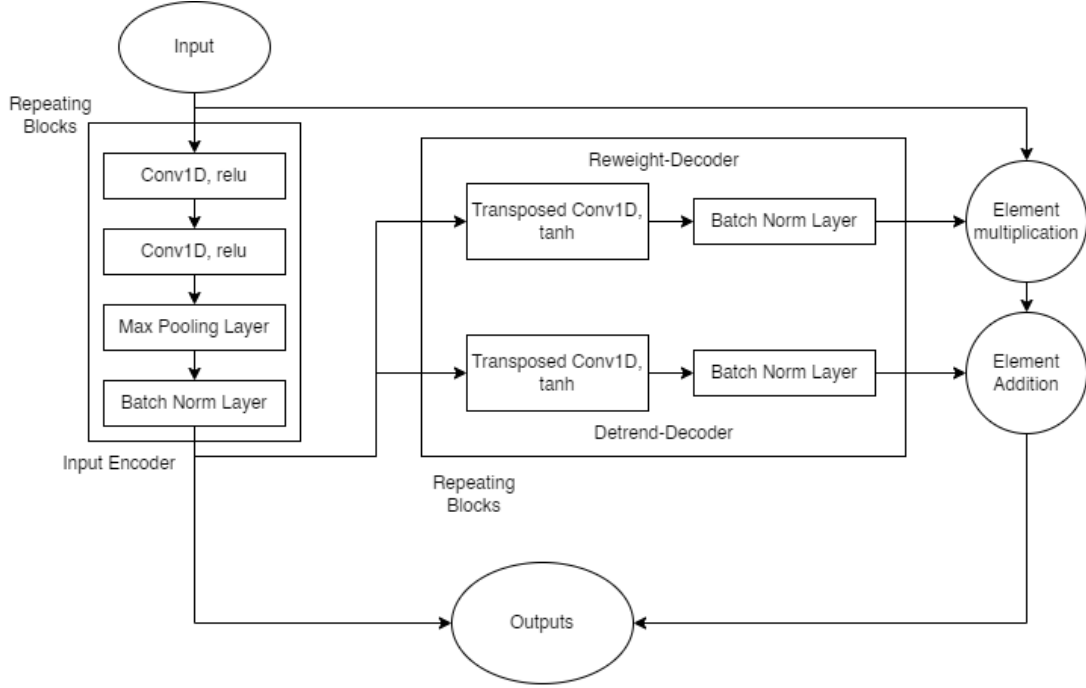


Figure 6 Reweight-Detrend Layer Overview

To generate ideal reference outputs for calculating loss, a scaled sample is reweighted with Huber's weight function using a scaled residual. The residual is the absolute difference between an element and the sample's median (eq 23), and the scale factor is the median absolute deviation of the sample, divided by 0.6745 (eq 24) which is the inverse of the coefficient used to compute Huber weight (eq 25). (rreikf people) found this method to have good performance/accuracy when identifying outlying datapoints.

$$SF = \text{median}(|X_i - \text{median}(|X_i|)|) * 0.6745^{-1} \quad (23)$$

$$R = |X_i - \text{median}(X)| * SF^{-1} \quad (24)$$

$$W_i = \begin{cases} 1, & \text{if } |X_i| \leq 1.547 \\ 1.547 * |X_i|^{-1}, & \text{otherwise} \end{cases} \quad (25)$$

$$\mathcal{L}_{\text{Huber}}(X, Y) = \begin{cases} 0.5 * (X_i - Y_i)^2, & \text{if } |X_i - Y_i| \leq \delta \\ \delta * |X_i - Y_i| - 0.5 * \delta^2, & \text{otherwise} \end{cases} \quad (26)$$

(Equation 23-25, (rreikf people). Equation 26, (matlab people))

The Huber loss function used computes loss according to (26) and defaults to epsilon 1. It is beyond the scope of this investigation to change the value of epsilon, so the default value is used throughout. This loss function also sums loss across batches which satisfies one of the criteria for meta-learning detailed in equation 8.



## Wavelet Layer

This layer upscales reweighted and detrended data by convolving it with a wavelet at different dilations, thus revealing some information about the signals' frequency content. This layer converts batches of two-dimensional arrays of signals into batches of three-dimensional sets of the samples' frequency information. The continuous wavelet transform algorithm is the one selected for its capacity to reveal lots of detail about a signal's frequency content, and a morse wavelet is used for the convolution. The exact specifications of the wavelet are determined by the algorithm and are dependent on the sample's window size.

## Set-Encoding Layer

The set encoding layer uses sliding three dimensional convolutions to extract features from sets of consecutive samples' spectral content. Sliding three dimensional convolutions are ideal for this because they permit the network to simultaneously learn features in the time-frequency domain, and the time domain. This layer also splits the wavelet-layer output into its real and complex components and concatenates them along the fourth dimension. This is so model gradients are not complex valued. After splitting and concatenating the data, the set-encoding layer uses repeating blocks - each having two consecutive three dimensional convolutions with relu activation, a three-dimensional max pooling layer and a batch normalization layer - to extract features from the set. For compatibility with other layers in the network which do not have as many dimensions to their data, the sliding convolutions and max pooling layers ensure that the output from the repeating blocks has one singleton dimension, which is then removed by squeezing the data.

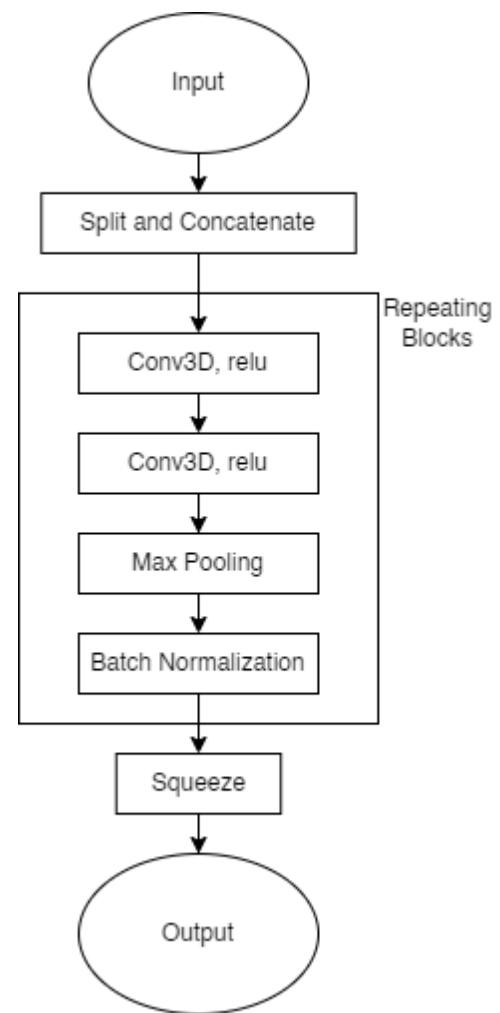


Figure 7 Set-Encoding Layer Overview

## Estimate-Encoding Layer

For stability between consecutive predictions, this network also uses as input, its previous prediction. The previous estimate is stored within the network object during use and in training, the last, scaled, detrended input sample plus a noise vector is used. This layer calls the wavelet function to upscale the previous prediction for encoding. Besides the wavelet layer, the estimate-encoder's architecture is like the set-encoding layer's but designed to handle fewer dimensions. See appendix for diagram.



## Prediction Layer

With the set and estimate encoders outputs as input, this layer predicts scaled and detrended future system states. In effect this layer is a transfer function which maps the encoders' outputs to scaled and detrended future states (equation). This layer has two decoders with the same convolution, batch norm architecture, operating with different dimensionalities.

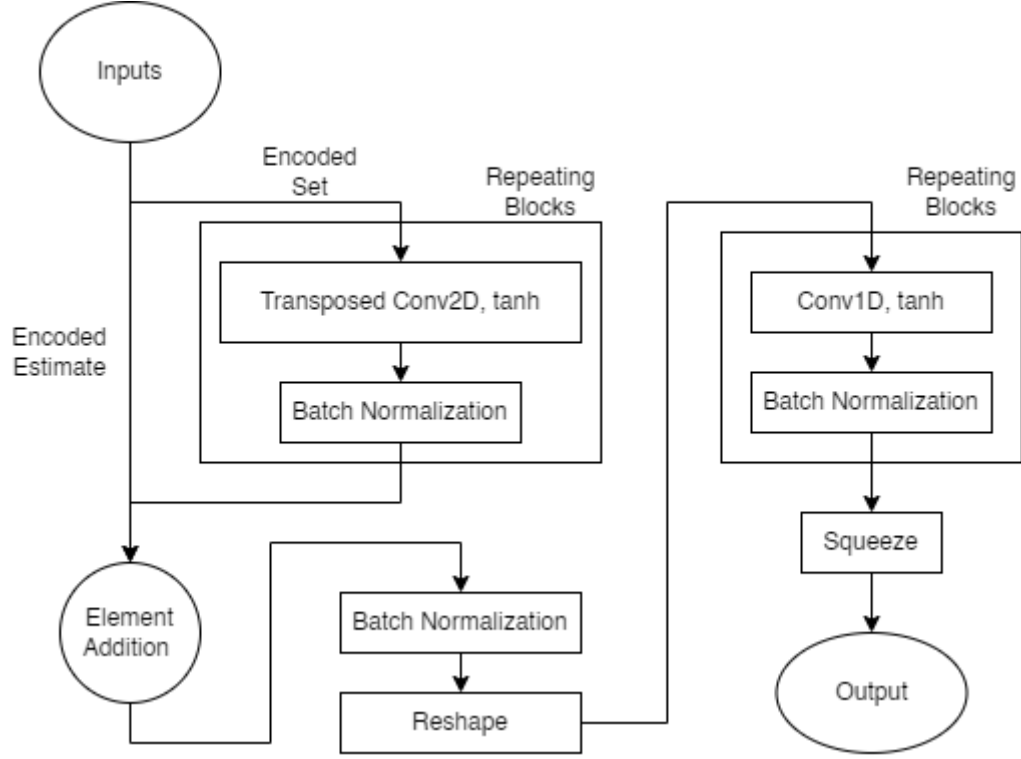


Figure 8 Prediction Layer Overview

This layer also has its own learning targets, which seek to ensure this layer's output matches the reweighted-detrended extrapolation. Huber loss is calculated according to equation twenty-five, and its reference outputs are generated in the same method as for the reweight-detrend layer. It is important to note that the output is scaled by the scaling layer before reweighting and detrending.

$$\mathcal{L}_{PL} = \mathcal{L}_{Huber}(\hat{X}_{RDT}, Y_{RDT}) \quad (27)$$

$$PL(\theta_{PL}, X_{set}, X_{estimate}) \rightarrow \hat{X}_{RDT} \quad (28)$$

$$Y_{RDT} = ICWT(CWT(W_{Huber} \odot Y)) \quad (29)$$

$$\theta_{PL} = \operatorname{argmin} \mathcal{L}_{PL}(\theta_{PL}, X_{set}, X_{estimate}, Y_{RDT}) \quad (30)$$

Where  $X_{set}$  and  $X_{estimate}$  are the set and estimate encoding layers' outputs respectively,  $ICWT$  and  $CWT$  are the inverse and continuous wavelet transforms,  $Y_{SDT}$  is the reweighted and detrended reference output as calculated by equation 29, and  $W_{Huber}$  are Huber weights as calculated with equations 23 through 25.

## Inverse-Detrend Layer

The final layer for this generator implementation, the inverse detrend layer uses information from the reweighting detrending layer to generate a vector which is then added to the prediction layer's output to recover the signal's underlying trend.

This layer uses repeating blocks of transposed one-dimensional convolutions, with hyperbolic tangent activations, and batch normalization layers to generate the trending vector. The vector is then added to the detrended prediction to recover the trended signal.

This layer also has its own learning targets which seek to minimize the error between this layer's output and the scaled reference output. Like other layers with their own learning targets in this network, this layer's loss is computed as the smooth L1 (Huber) loss between its output and a reference output. The reference output used for this layer's loss function is the windowed sample, time shifted forward by a fixed extrapolation length. This sample is scaled by dividing it by a scale factor (as determined in equation 17). When generating training samples this is achieved with the scaling layer.

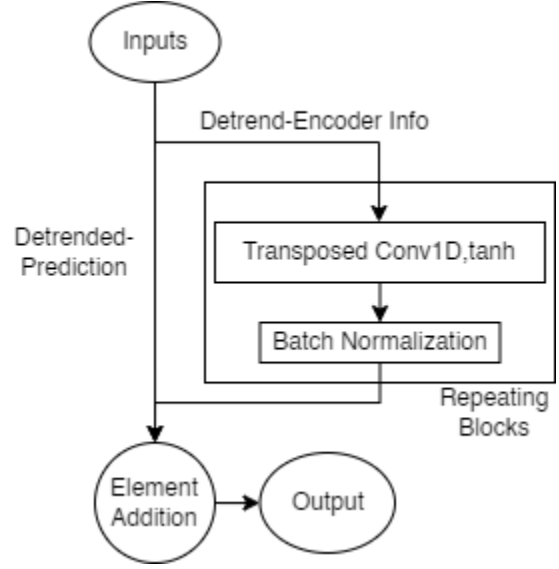


Figure 9 Inverse-Detrend Layer Overview

$$\mathcal{L}_{IDL} = \mathcal{L}_{Huber}(\hat{X}_{SC}, Y_{SC}) \quad (31)$$

$$IDL(\Theta_{IDL}, \hat{X}_{RDT}) \rightarrow \hat{X}_{SC} \quad (32)$$

$$Y_{SC} = Y * SF^{-1} \quad (33)$$

$$\Theta_{IDL} = \underset{\Theta_{IDL}}{\operatorname{argmin}} \mathcal{L}_{IDL}(\Theta_{IDL}, \hat{X}_{RDT}, Y_{SC}) \quad (34)$$

## Compound Loss Function

Since gradients will be computed for the entire network simultaneously, each sublayer's loss is compounded into a complete loss function which replaces equation 15.

$$\begin{aligned} \mathcal{L}_{Gen} = & \lambda_1 \mathcal{L}_{RWDTL}(\Theta_{RWDTL}, \mathcal{D}_k, \mathcal{D}_{ref}) \\ & + \lambda_2 \mathcal{L}_{PL}(\Theta_{PL}, X_{set}, X_{estimate}, Y_{RDT}) \\ & + \lambda_3 \mathcal{L}_{IDL}(\Theta_{IDL}, \hat{X}_{RDT}, Y_{SC}) \end{aligned} \quad (35)$$

## **Discriminator Architecture**

## Meta Architecture

## **Considerations for Training**

## Evaluation

Results in accuracy.

Consider showing results on data generated with known distributions as well as bitcoin data.

Ie :  $x_{t+1} = x_t + W$  where  $W$  is some value randomly generated from a known distribution.  
System can be initialized with a random point.

Show gan predictions have similar distributions to data







# Introduction

About the project.

# Background

About the task in hand.

# Concept

How the task is divided.

# Implementation

Alternatives explored for each part (why each alternative is considered and relative performance) and “final” design.

Possibly show code.

# Evaluation of Design.

# Implementation

# Evaluation

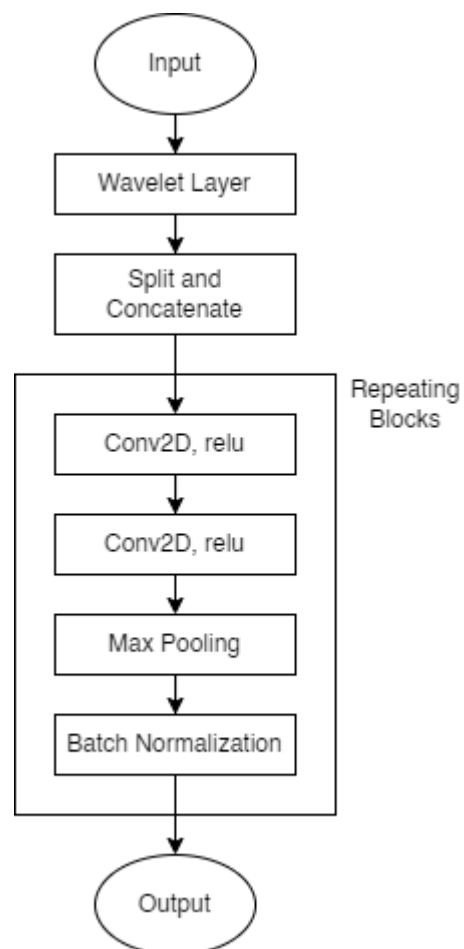
## Appendix

---

### Algorithm 1

Generating training samples from price data

1. Observe price data once per minute
2. Upscale signal with complex wavelet transforms
3. Store this in dataset



# Previous Version of Report

- Abstract
- Introduction
- Concept
  - Random Walk Characteristics
  - Finite States
  - Profit Maximization
- Implementation
- Input Layer
- Output Layer
  - Unideal Interface
- Prediction Layer
  - Outlier Detection and Removal
  - Spectral Densities and Wavelet Transform
  - Kalman Filter
  - Generative Adversarial Training Scheme
  - Considerations for Training
  - Network Architecture
  - Training Prediction Network
  - Wavelet Transform Layers
- Decision Layer
  - Redundant Weighted Predictions
  - Tuned Lowpass Filter
  - Binary Signal and Profit Optimization
  - Considerations for Training
  - Decision Network Architecture
- Discussion

# Quotes & Other Extracts

## A Quantum Like Approach to the Stock Market

*This contextual influence is responsible of the non-Kolmogorovian quantum-like behavior of the market at a statistical level. (Aerts, 1)*

*the stock price follows a geometric Brownian motion with constant drift and constant volatility. (Aerts, 2)*

$$dS = \mu S dt + \sigma S dW,$$

*(Aerts, 2) -> Black Scholes*

*This random walk hypothesis has characterized the financial derivatives modelling and valuation in the banks and financial institutions and is strictly linked with the so-called efficient market hypothesis, that is, the assumption that financial markets are 'informationally efficient' and prices of traded assets instantly change to reflect new public information. (Aerts, 3)*

*It is a basic doctrine in quantum mechanics that a property, e.g., the value of an observable, of a physical entity in each state cannot be considered as a pre-existing feature of the entity (Aerts, 4)*

*the property is actualized in a measurement process and can be different if a different measurement context is considered. (Aerts, 4)*

*probability cannot be interpreted as formalizing the subjective ignorance about the actual state of the entity, as it occurs for classical (Kolmogorovian) probability and statistical mechanics, but it is rather caused by the presence of a lack of knowledge (fluctuations) concerning how context interacts with the entity that is considered. (Aerts, 4)*

*A quantum-like potentiality structure occurs in an opinion poll because two types of questions are possible. (Aerts, 4)*

*Indeed, it is reasonable to suppose that prices of assets are not predetermined, but they depend on the simultaneous happenings of human decisions to buy and/or sell. (Aerts, 4)*

# Bibliography

Akram, M. A., Liu, P., Tahir, M. O., Ali, W., & Wang, Y. (2019). A state optimization model based on Kalman filtering and robust estimation theory for fusion of multi-source information in highly non-linear systems. *Sensors*, 19(7), 1687. <https://doi.org/10.3390/s19071687>

Beale, M. H., Hagain, M. T., & Demuth, H. B. (2022). *Deep Learning Toolbox Reference*. Mathworks.

Cai, W., Zhang, M., & Zhang, Y. (2017). Batch mode active learning for regression with expected model change. *IEEE Transactions on Neural Networks and Learning Systems*, 28(7), 1668–1681. <https://doi.org/10.1109/tnnls.2016.2542184>

Ceravolo, R. (2004). Use of instantaneous estimators for the evaluation of structural damping. *Journal of Sound and Vibration*, 274(1-2), 385–401. <https://doi.org/10.1016/j.jsv.2003.05.025>

Dahleh, M., Dahleh, M. A., & Verghese, G. (n.d.). *Lectures on Dynamic Systems and Control*. Massachusetts; Massachusetts Institute of Technology Department of Electrical Engineering and Computer Science.

Emzir, M. F., Woolley, M. J., & Petersen, I. R. (n.d.). *A Quantum Extended Kalman Filter*. School of Engineering and IT, University of New South Wales. Retrieved from <https://arxiv.org/abs/1603.01890>.

Geerts, N. C. P. J. (1996). *The Hilbert transform in complex Envelope Displacement Analysis (CEDA)*. (DCT rapporten; Vol. 1996.121). Eindhoven: Technische Universiteit Eindhoven.

Grover, B. R., & C., H. P. Y. (2012). *Introduction to random signals and applied kalman filtering: With Matlab exercises*. John Willey.

Hommes, C. H. (2001). Financial Markets as Nonlinear Adaptive Evolutionary Systems. *Quantitative Finance*, 1(1), 149–167. <https://doi.org/10.1080/713665542>

Huber, P. J., & Ronchetti, E. M. (2011). *Robust statistics*. Wiley.

Hull, J. C. (2022). *Options, futures, and other derivatives*. Pearson Education Limited.

Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2018). *Image-to-Image Translation with Conditional Adversarial Networks*. Arxiv. Retrieved from <https://arxiv.org/abs/1611.07004>.

Kingma, D. P., & Ba, J. L. (2015). *ADAM: A Method for Stochastic Optimization*. ICLR. Retrieved from <https://arxiv.org/abs/1412.6980>.

Lee, T. K., Baddar, W. J., Kim, S. T., & Ro, Y. M. (2018). Convolution with logarithmic filter groups for efficient shallow CNN. *MultiMedia Modeling*, 117–129. [https://doi.org/10.1007/978-3-319-73603-7\\_10](https://doi.org/10.1007/978-3-319-73603-7_10)

Li, H., Mao, C. X., & Ou, J. P. (2013). Identification of hysteretic dynamic systems by using Hybrid Extended Kalman filter and wavelet multiresolution analysis with limited observation. *Journal of Engineering Mechanics*, 139(5), 547–558. [https://doi.org/10.1061/\(asce\)em.1943-7889.0000510](https://doi.org/10.1061/(asce)em.1943-7889.0000510)

Liepins, V. (2013). *Extended Fourier Analysis of Signals (dissertation)*. arxiv. Retrieved from <https://arxiv.org/abs/1303.2033>.

Mili, L., Cheniae, M. G., Vichare, N. S., & Rousseeuw, P. J. (1996). *Robust state estimation based on projection statistics [of Power Systems]*. *IEEE Transactions on Power Systems*, 11(2), 1118–1127. <https://doi.org/10.1109/59.496203>

Misiti, M., Misiti, Y., Oppenheim, G., & Poggi, J.-M. (2015). *Wavelet Toolbox User's Guide*. Mathworks.

Peng, S. (2013). *Design and Analysis of Fir Filters Based on Matlab* (thesis).

Rankin, J. M. (1986). *Kalman filtering approach to market price forecasting*. Iowa State University Capstones, Theses and Dissertations. <https://doi.org/10.31274/rtd-180813-7911>

Vaswani, A., 123, 123, 123, 123, 132, 123, 123, 123, & 32, 123. (2017). *Attention is All You Need*. 31st Conference on Neural Information Processing Systems (NIPS). Retrieved from <https://arxiv.org/abs/1706.03762>.

Williams, M. B. (2016). *Fast Fourier Transform in Predicting Financial Securities Prices* (thesis).

Wu, N., Green, B., Ben, X., & O'Banion, S. (n.d.). *Deep Transformer Models for Time Series Forecasting The Influenza Prevalence Case*. Arxiv. Retrieved from <https://arxiv.org/abs/2001.08317>.

Zhang, C., & Huang, L. (n.d.). *A Quantum Model for the Stock Market* (thesis).

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). *Unpaired image-to-image translation using cycle-consistent adversarial networks*. 2017 IEEE International Conference on Computer Vision (ICCV). <https://doi.org/10.1109/iccv.2017.244>

---