Application of Hybrid Adaptive filters for Stock-market prediction

Abhishek Mittal Ashuthosh Bharadwaj

23rd November 2020

Abstract

Stock market prediction is a very active research area. Many algorithms have been desgined to extract information about the future values of the stock market from its past values. Machine Learning has been used extensively nowadays to observe the past data and predict the future outcomes. But creating these artificial neural networks involves thousand and thousands of parameters like weights and biases and can be very time consuming both in computation and modifications. Signal processing offers elegant solutions which we will discuss below. The most conventional filters are the moving average filter which do a decent job, but we aim to achieve a higher accuracy. We thereby shift our focus to adaptive filters which are very old filters but have been recently used in predicting the markets, they give us a higher accuracy due to their variable filter coefficients. After analysing the adaptive filters , we feel that we can do better by introducing the concept of hybrid adaptive filters which take in input as the discrete cosine transform of the input signal and then give the outputs.

1 Conventional Filters

1.1 Moving Average Filter

The moving average filter is the most basic filter to make predictions. All it does is take the average of the past data inputs. It is useful in removing the high frequency noise from the data. Removing the market noise from the stock market data is really helpful as it gives you a more clear picture of when the stock market is truly rising and falling. Daily rises and falls don't give much information about a stock's performance. This is the property that moving average filters exploit, they give a low output only if the input data has been low for quite some time in the past and same for high values. But there is a problem with this, they have a huge delay in the output, which is not feasible for trading atleast, because trading is about making predictions each day and then taking decisions. So the moving average filter is not very reliable and more accurate filters have to be constructed.

1.2 Kalman Filter

If we have a lot of noisy measurements, then how do we determine the actual value or the true value based on these measurements. We do this with a Kalman filter, which takes a weighted combination of these measurements and then estimates the true value. But to assign the weights to each of the measurements we need more information like how our the measurements related, what are the statistical properties of the noise in the measurements etc. Apart from this it also requires a mathematical model which would be used to calculate the mathematical estimate based on the previous values. The basic Kalman filter works only on linear dynamic systems which is what we have worked on. But there are modifications of this filter which also work on non linear systems.

This recursive filter has a variety of applications in control theory. Many of the real life applications have many states that need to be measured but a very few observable parameters, but by combining multiple measurements we can calculate the internal state of the system. For example we want to estimate the position of the car travelling on the road. The measurement instruments that we have in hand are the GPS, car's odometer and speedometer. In an ideal situation the car's motion can be characterised by the basic kinematics equations. But the world is a non ideal place and here there are multiple parameters that have to be considered to be able to exactly model the car's movement. When in motion, the readings might be different based on what we calculate from our mathematical model in a given interval of time. This could be because of bad road, low air pressure in tires etc. This is called process noise. Now when we take the measurements, they might not accurately measure the amount of distance the car has actually travelled. This could be because of bad measurement devices, least error, etc. This is called measurement noise. So we have 2 types of noise that we will have to take into account when designing the Kalman filter. We model this noise as a normal distribution where the mean is the measured value and the covariance is based on the instrument error or environmental disturbances.

$$x_k = Ax_{k-1} + Bu_k + w_k$$

where $w_k \sim N(0, Q)$

$$y_k = Cx_k + v_k$$

where $v_k \sim N(0, R)$

The algorithm is broken down into 2 main parts: PREDICT and UPDATE. We first predict the values using the previous values and our mathematical model. We are not only predicting the estimated state value but we are also going to estimate how much noise is our estimated state going to have, which would be dependent on both the process noise's covariance and the measurement noise's covariance. During the update stage we would be updating our predicted state and predicted covariance using the measured values. During this update we define an interesting quantity, called the Kalman gain which tells us how much are the measured values close to the predicted values. We aim to find the optimal value from the linear combination of the estimated value and the predicted value. We then use these optimal values in the future as a reference.

The most interesting part of the Kalman filter is that it only depends on the previous state, whereas all the other filters that we will be talking about in this project are going to depend on a huge amount of past values.

Prediction

$$\hat{x}_k^- = A\hat{x}_{k-1}^- + Bu_k$$

$$P_k^- = AP_{k-1}A^T + Q$$

Update

$$K_k = \frac{P_k^- C^T}{CP_k^- C^T + R}$$

$$x_k = x_k^- + K_k(y_k - Cx_k^-)$$

$$P_k = (I - K_k C)P_k^-$$

2 Adaptive filters

2.1 Need

The role of a filter in signal processing is to transform an input signal into another output signal so that we extract useful information about it. We have most commonly studied causal linear time-invariant filters. But these filters are only good for the deterministic signals and not the stock market signals. The first reason they are not good is because they are time invariant whereas the stock market signals are a highly dynamic and time varying signal. The first approach is to study the statistical properties of the stock market signal and assume it is constant over all time.In this case rather than analysing the stock market signal, we can analyse its statistics and get the statistics of the predicted signal as output. But we would be making really wrong assumptions, because practically the stock market signals are non stationary and their statistical properties change with time. So the second approach is to have time varying filter coefficients, which leads us to adaptive filters.

2.2 Working of Adaptive filters

Adaptive filters are basically linear filters but their coefficients are not hardcoded beforehand, rather they are determined by an adaptive algorithm which takes 2 inputs, the error signal and the desired signal. There are 2 main algorithms LMS and RLS, many versions of these algorithms exist. In general all algorithms have one thing in common: drive the error signal to 0. But we want a common performance index irrespective of any input signal which we will model as a random process. Now to minimise the error we would need a desired signal, based on which we will determine the coefficients. Unless the desired signal is known, the algorithm would not know if the current filter coefficients are correct or not. The differentiating factor between algorithms is the performance index, which is high when the error is low and vice versa. The most common application of the adaptive filters is to estimate the filter coefficients of a plant which is given to us but its filter coefficients are unknown, so based on the various input and desired signal pairs we can use them to train the algorithm. Here also we will do the same, and treat the stock market as a plant whose filter coefficients are unknown but we wish to estimate them. Using the coefficients we estimated for this unknown plant we will make an assumption that the system will remain almost the same for a few days and try to apply this filter to predict the stock market returns for the next 10 days. The input signal to the filter would be the 1:N days returns of the stock market data and the desired signal will be the next days returns that is 2:N+1 returns. We can improve the performance of these algorithms, by continuously training the filter using a set of latest N data points. The stock market is very dynamic so just using a filter to predict for the long term is very unfruitful. The more we increase the number of data points, that is the window size, the more accuracy we will achieve. But such huge amount of data increases the storage space hugely. Instead we can take the DCT of the input signals, so that we can eliminate the high frequency terms from the signal. The high frequency terms are market noise and dont represent the intrinsic nature of the market. We choose DCT instead of DFT because DCT is purely real and for our application thats exactly what we need. This then leads us to the study of hybrid adaptive filters which we will discuss later in this report.

2.3 LMS Algorithm

The performance index of this algorithm is the mean of the square error. Since our input and the desired signals are unknown to us, we must devise an algorithm which works for any signal. Hence we model the signal as a random process and define its statistical properties like the auto correlation of the signal and the cross correlation between the input and desired signal. Now obviously our error signal would also be a random process, so we cant guarantee that the error signal after the algorithm is executed would be absolutely zero. So now the question is how do we prove that the algorithm would minimise the error in general for any signal. Rather than proving that the error would be minimum for any signal, we show that the statistical properties like the mean sqaure error would be minimum. And this is our performance index. There are many reasons why we take the mean square error as our performance index apart from the one explained above. The square term introduces convexity to the error graph when plotted against the filter coefficients. For any convex function we know that there exists only 1 minima. So if we find a local minima we can be sure that this minima is the global minima and we have thus found the optimal solution.

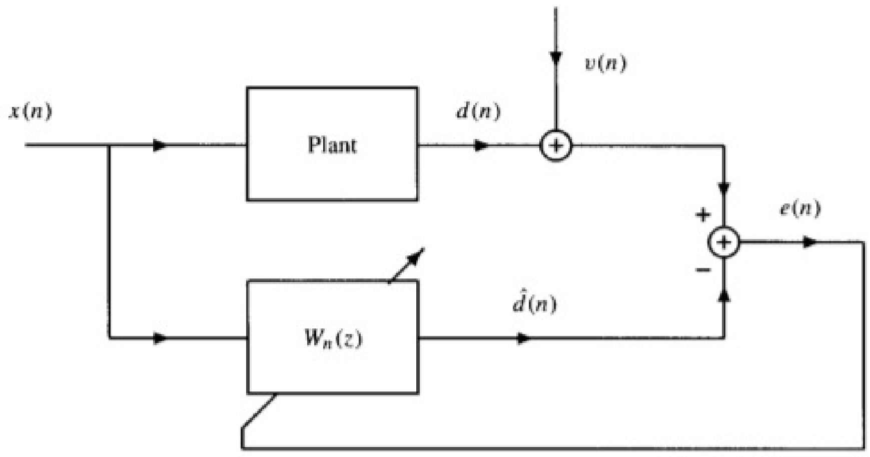

Figure 1: System identification

$$\xi(n) = E\{|e(n)|^2\}$$

$$e(n) = d(n) - \hat{d}(n) = d(n) - \mathbf{w}_n^T \mathbf{x}(n)$$

$$\hat{d}(n) = \sum_{k=0}^{p} w_n(k)x(n-k) = \mathbf{w}_n^T \mathbf{x}$$

$$E\{e^2(n)\} = E\{v^2(n)\} + E\{[d(n) - y(n)]^2\}$$

We want to minimise the error with respect to the filter coefficients, so we partially differentiate it with respect to all the coefficients individually and then equate their gradient to 0. After partial differentiation we get these following equations

$$E\{e(n)x^*(n-k)\} = 0, \forall k = 0, 1, \ldots p$$

They can be rewritten in the matrix form where R is the auto correlation matrix and is Toeplitz.

$$\mathbf{R}_x(n)\mathbf{w}_n = \mathbf{r}_{dx}(n)$$

$$\mathbf{R}_x(n) = \begin{pmatrix} E\{x(n)x^*(n)\} & \cdots & E\{x(n-p)x^*(n)\} \\ E\{x(n)x^*(n-1)\} & \cdots & E\{x(n-p)x^*(n-1)\} \\ \vdots & \ddots & \vdots \\ E\{x(n)x^*(n-p)\} & \cdots & E\{x(n-p)x^*(n-p)\} \end{pmatrix}$$

$$\mathbf{r}_{dx}(n) = [E\{d(n)x^*(n)\}, E\{d(n)x^*(n-1)\}, \ldots, E\{d(n)x^*(n-p)\}]^T$$

$$\hat{E}\{|x(n)|^2\} = \frac{1}{N}\sum_{k=0}^{N-1} |x(n-k)|^2$$

Direct solutions and iterative solution
The direct solution involves solving the above set of linear equations at all points of time, but that is computationally expensive, so we will stick to the iterative solution throughout this paper.

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \mu\nabla\xi(n)$$

$$\nabla\xi(n) = \nabla E\{|e(n)|^2\} = E\{\nabla|e(n)|^2\} = E\{e(n)\nabla e^*(n)\}$$

$$\nabla e^*(n) = -\mathbf{x}^*(n)$$

$$\nabla\xi(n) = -E\{e(n)\mathbf{x}^*(n)\}$$

$$\hat{E}\{e(n)\mathbf{x}^*(n)\} = \frac{1}{L}\sum_{l=0}^{L-1} e(n-l)\mathbf{x}^*(n-l)$$

On substituting the following values in the iterative solution we get

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \frac{\mu}{L}\sum_{l=0}^{L-1} e(n-l)\mathbf{x}^*(n-l)$$

$$\hat{E}\{e(n)\mathbf{x}^*(n)\} = e(n)\mathbf{x}^*(n)$$

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu e(n)\mathbf{x}^*(n)$$

If we assume the signal is stationary then we have to solve only a set of linear equations and we would get our filter coefficients, but we want our algorithm to handle non stationary and stationary solutions both. Hence instead of the direct solution, we will focus on the iterative solution. The iterative algorithm is as follows:
1. Initialise the filter coefficient vector with any arbitrary value, preferably the zero vector
2. Find the error between the desired signal and the output signal
3. update your coefficients based on the error.
4. Go to step 2

In this algorithm we will find our way to the bottom of the error graph in each step, but we have to also decide how large a step we want to take in each iteration. This is also called the learning rate. Choosing this learning rate is very crucial, too big can lead us to never reach the bottom point, rather we would be jumping around the bottom point and too small may take us forever to reach the bottom point. We can derive an upper bound on the learning rate and its given by this formula below.

$$0 < \mu < \frac{2}{\lambda_{max}}$$

$$\lambda_{max} \leq \sum_{k=0}^{p} \lambda_k = tr(\mathbf{R}_x)$$

$$tr(\mathbf{R}_x) = (p+1)r_X(0) = (p+1)E\{|x(n)|^2\}$$

$$0 < \mu < \frac{2}{(p+1)E\{|x(n)|^2\}}$$

A major drawback of the LMS algorithm is that the correction in the filter coefficients vector is dependent on the input that is being given to it. If the inputs are too small, this could be a problem as our coefficients are hardly getting updated in the previous iteration. So to solve this problem we move to the normalised LMS.

2.4 Normalised LMS

This is a variation of the LMS algorithm where we keep updating the learning rate in each iteration. This gives us better results than the normal LMS algorithm, as we reach the bottom point more often in this algorithm. The reason being that variable learning rates can decide whether to take huge or small steps depending on how far they are from the bottom point. The algorithm remains the same as that of the LMS algorithm but with an extra line in the iteration part where we calculate the new learning rate. The reason it is called normalised LMS is because the correction term is normalised by the magnitude of the input signal, so even if the input is too small it doesnt matter as the input has been normalised.

$$w_{N+1} = w_n + \beta \frac{x^*(n)}{||x(n)||^2} e(n)$$

$$w_{N+1} = w_n + \beta \frac{x^*(n)}{\|x(n)\|^2 + \epsilon} e(n)$$

$$\|\mathbf{x}(n+1)\|^2 = \|\mathbf{x}\|^2 + x(n+1)^2 - x(n-p)^2$$

2.5 RLS Algorithm

One thing to notice in the LMS algorithm is that the performace index of the algorithm didnt directly depend on the input signal, rather it depended on the statistical properties of the signal. There can be multiple signals which have the same statistical properties, so the LMS algorithm would give the same output for all those signals whereas in reality the output of the signals is not going to be the same. This is solved by the RLS algorithm whose performance index is dependent on the input signal and not just the statistical properties of the signal. The performance index is the summation of the square error between the signals. The contrast is that the LMS uses a mean square error, more like a global optimal algorithm whereas the RLS uses a least square error pointwise, where local optimality is achieved at every point. We had mentioned about choosing a window of data to train our filter. So till now we have chosen a window where every data point's contribution in affecting the distribution is equal, but we want to change this by giving the most recent data points more weightage than the distant past values. This is also called the exponential weighted RLS algorithm

$$\varepsilon(n) = \sum_{i=0}^{n} |e_i|^2$$

2.6 Exponential Weighted RLS

$$\mathbf{w}_n = [w_n(0), w_n(1), \ldots, w_n(p)]^T$$

$$\mathbf{x}(i) = [x(i), x(i-1), \ldots, x(i-p)]^T$$

$$\varepsilon(n) = \sum_{i=0}^{n} \lambda^{n-i} |e_i|^2$$

$$e(i) = d(i) - y(i) = d(i) - \mathbf{w}_n^T \mathbf{x}(i)$$

Derivatives are zero

$$\frac{\partial \varepsilon(n)}{\partial w_n(k)} = \sum_{i=0}^{n} \lambda^{n-i} e(i) \frac{\partial e^*(i)}{\partial w_n^*(k)} = -\sum_{i=0}^{n} \lambda^{n-i} e(i) x^*(i-k) = 0$$

We have done the same thing as in the case of LMS Algorithm. We differentiated the error with respect to the filter coefficients. And we obtain the following matrix form

$$\mathbf{R}_x(n)\mathbf{w}_n = \mathbf{r}_{dx}(n)$$

$$\mathbf{R}_x(n) = \sum_{i=0}^{n} \lambda^{n-i} \mathbf{x}^*(i) \mathbf{x}^T(i)$$

$$\mathbf{r}_{dx}(n) = \sum_{i=0}^{n} \lambda^{n-i} d(i) \mathbf{x}^*(i)$$

We now define our iterative equation

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \Delta\mathbf{w}_{n-1}$$

$$\mathbf{w}_n = \mathbf{R}_x^{-1}(n)\mathbf{r}_{dx}(n)$$

$$\mathbf{r}_{dx}(n) = \lambda\mathbf{R}_x(n-1) + \mathbf{x}^*(n)\mathbf{x}^T(n)$$

We will use the Householder's equation stated below to simplify the expression given above.

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^H)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^H\mathbf{A}^{-1}}{1 + \mathbf{v}^H\mathbf{A}^{-1}\mathbf{u}}$$

$$\mathbf{R}_x^{-1}(n) = \lambda^{-1}\mathbf{R}_x^{-1}(n-1) - \frac{\lambda^{-2}\mathbf{R}_x^{-1}(n-1)\mathbf{x}^*(n)\mathbf{x}^T(n)\mathbf{R}_x^{-1}(n-1)}{1 + \lambda^{-1}\mathbf{x}^T(n)\mathbf{R}_x^{-1}(n-1)\mathbf{x}^*(n)}$$

$$\mathbf{P}(n) = \mathbf{R}_x^{-1}(n)$$

$$\mathbf{g}(n) = \frac{\lambda^{-1}\mathbf{P}(n-1)\mathbf{x}^*(n)}{1 + \lambda^{-1}\mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{x}^*(n)}$$

$$\mathbf{P}(n) = \lambda^{-1}[\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{x}^T\mathbf{P}(n-1)]$$

$$\mathbf{g}(n) + \lambda^{-1}\mathbf{g}(n)\mathbf{x}^T\mathbf{P}(n-1)\mathbf{x}^*(n) = \lambda^{-1}\mathbf{P}(n-1)\mathbf{x}^*(n)$$

$$\mathbf{g}(n) = \lambda^{-1}[\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{x}^T\mathbf{P}(n-1)]\mathbf{x}^*(n)$$

$$\mathbf{g}(n) = \mathbf{P}(n)\mathbf{x}^*(n)$$

$$\mathbf{R}_x(n)\mathbf{g}(n) = \mathbf{x}^*(n)$$

$$\mathbf{w}_n = \mathbf{P}(n)\mathbf{r}_{dx}(n)$$

$$\mathbf{w}_n = \lambda\mathbf{P}(n)\mathbf{r}_{dx}(n-1) + d(n)\mathbf{P}(n)\mathbf{x}^*(n)$$

$$\mathbf{w}_n = [\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{x}^T\mathbf{P}(n-1)]\mathbf{r}_{dx}(n-1) + d(n)\mathbf{g}(n)$$

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \mathbf{g}(n)[d(n) - \mathbf{w}_{n-1}^T\mathbf{x}(n)]$$

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \alpha(n)\mathbf{g}(n)$$

$$\alpha(n) = d(n) - \mathbf{w}_{n-1}^T\mathbf{x}(n)$$

$$e(n) = d(n) - \mathbf{w}_{n-1}^T\mathbf{x}(n)$$

$$z(n) = \mathbf{P}(n-1)\mathbf{x}^*(n)$$

Initial Conditions

$$\mathbf{P}(0) = [\sum_{i=-p}^{0} \lambda^{-i}\mathbf{x}^*(i)\mathbf{x}^T(i)]^{-1}$$

$$\mathbf{r}_{dx}(0) = \sum_{i=-p}^{0} \lambda^{-i}d(i)\mathbf{x}^*(i)$$

$$\mathbf{R}_x(0) = \delta\mathbf{I}$$

$$\mathbf{P}(0) = \delta^{-1}\mathbf{I}$$

$$\mathbf{w}_0 = \mathbf{0}$$

There are 3 parts to the algorithm

1. Defining the parameters are the p(filter order), λ(the exponential weighing factor) and δ which is used to initialise $P(0)$

2. Initialising the weight vector and the P matrix.

3. $z(n) = \mathbf{P}(n-1)\mathbf{x}^*(n)$

4. $g(n) = \frac{1}{\lambda+\mathbf{x}^T(n)\mathbf{z}(n)}\mathbf{z}(n)$

5. $\alpha(n) = d(n) - \mathbf{w}_{n-1}^T\mathbf{x}(n)$

6.$\mathbf{w}_n = \mathbf{w}_{n-1} + \alpha(n)\mathbf{g}(n)$

7. $\mathbf{P}(n) = \lambda^{-1}[\mathbf{P}(n-1) - \mathbf{g}(n)z^H(n)]$

The only drawback of the RLS algorithm is the high computational complexity, which can lead to stability problems.

3 DCT

The discrete cosine transform is similar to the DFT of a discrete time signal, with the only difference being that the DCT is purely real and hence finds many applications in the real world due to most of the real life signals being real in nature. Just like we multiply the signal with complex exponential in the DFT, here we would multiply them with cosine terms.

To cancel the imaginary terms from the DFT that we obtain, we just take an even extension of the original signal and then calculate its DFT. There are many ways to take this even extension. We will describe one method called the half sample symmetry where we just reverse the signal and append it to the end of the original signal.

On doing so we find that the DCT coefficients look like as follows $C(k) = \alpha(k) \sum_{n=0}^{N-1} x(n) cos[\frac{\pi(2n+1)k}{2N}$ where k lies between 0 and $N-1$
$\alpha(0) = \sqrt{\frac{1}{N}}$ and $\alpha(k) = \sqrt{\frac{2}{N}}$ where k lies between 1 and $N-1$.
We can represent this in $N \times N$ matrix form C_N where its elements are given as

$c_{kn} = \sqrt{\frac{1}{N}}$ when $k = 0$ and $0 \leq n \leq N-1$

$c_{kn} = \sqrt{\frac{2}{N}} cos[\frac{\pi(2n+1)k}{2N}]$ when $k = 0$ and $0 \leq n \leq N-1$

Hybrid Adaptive Filters

We have seen the power of adaptive filters, but let us try to experiment with this filter and try to improve our existing results. We have given our algorithms the time domain signals, but now let us give them frequency domain signals and see if that improves our prediction accuracy. The algorithm is as follows: 1. We first take the input signal and find its equivalent transformation in whichever domain we want to (z transform, discrete foruier transform or discrete cosine transform). Here we choose DCT because the DCT is specifically constructed to avoid running into the complex numbers, which are not needed as our signals are real valued. 2. After this we normalise the transformed signals with the square root of the power 3. Now we feed them into our Adaptive filter which is controlled by either LMS or RLS Algorithm. Challenges faced Giving the adaptive filter the DCT signal, but couldnt understand the output Tried creating the weight plots but were unable to find out why the minimum error was so high Finding the right auto correlation and cross correlation.

4 Results

The results can be seen from the codes

5 Conclusions

Out of all these filters the Hybrid Adaptive filters have shown very high accuracy due to their adaptive nature. The other filters like the adaptive filters show medium level accuracy whereas the Moving average and random walk process models show very low accuracy. There are various modifications that can be done with these adaptive algorithms like the fast RLS algorithm which reduces the complexity of the algorithm allowing for more data to be used for training the filter.