TITLE: Machine Learning Algorithm for Personal Portfolio Wealth Creation

INVENTOR(S): Weimin Zhao, Lawrence Thoman, John C. Wang,

USPTO PATENT APPLICATION NUMBER: 62718943

BACKGROUND

In the 7x24 global trading market environment, it is critical for an investor to have access to the BIG DATA about the past, present and future cash flow value added or risk-adjusted prices and anticipated future changes in market ask vs bid transaction prices automatically. To this end, models are developed that attempt to predict the future. The problem is that the models need to be tuned with the right factors and factor values to accurately reflect what really happens. Machine learning Algorithms using a variety of tools including Deep Neural Networks (DNN) are used to tune the models. The problem is that for the large sets of data needed to get good results the DNN processing steps take a long time to run. A way to run the DNN or Deconvolution algorithms much faster is needed if one wants the model to reflect the real market conditions. To this end, we include a Field Programmable Gate Array (FPGA) based Internet of Things (IoT) Reconfigurable Matrix Engine (RME)

The goal of this Machine learning Algorithm and its implementation is to create personal wealth by making profitable trades in accordance with portfolio optimization.

BRIEF SUMMARY OF THE INVENTION

Our Solution to Portfolio Wealth Creation (PWC) is a combination of things:

- The system design
    - UBX™
        - Built on open standard , open source tools
        - scalable,
        - distributed,
        - parallel Linux platform
    - Virtual Pocket Sorting and Indexing

- A collection of algorithms for determining the risk reward of various trades.
    - Pattern Recognition Machine Learning Short Term Options Trading System
    - Deconvolution algorithms
    - Linear Virtual pocket sort
    - DNN model: Deep Neural Network Option Pricing Model
    - QED model: Quantum Electro Dynamic Field Effect Option Pricing Model.

- Specialized Hardware, the **Reconfigurable Matrix Engine (RME)** that make it possible to run the algorithms fast enough so that the results are meaningful in real time

Field Programmable Gate Array (FPGA) based Internet of Things (IoT) Reconfigurable Matrix Engine (RME).

Our solution to this configurable acceleration or speed up problem is to implement a variety of algorithms including the Markowitz portfolio optimization, DNN or Deconvolution algorithms in a pipelined manner in the Field Programmable Gate Array (FPGA) based Internet of Things (IoT) Reconfigurable Matrix Engine (RME).

In a similar fashion, the Deconvolution algorithm calculations for scoring on massive numbers of Mortgage Backed Security (MBS) loan-level data can be done in a timely manner.
This same approach can be used to speed up many other CPU intensive computation problems that involve doing the same type of complex calculation on a large data set, like the KDS patented linear time sorter. IE., Transformation of time & space tensor in Riemannian Manifold.

DETAILED DESCRIPTION AND BEST MODE OF IMPLEMENTATION

This Personal Portfolio Wealth Creation system is implemented in the ABC UBX^{TM} system architecture.
(Ai network engine Big data Cloud computing UBiquitous indeX matrix computing) architecture

The UBX™ Main server provides the data ticker and risk data to the Personal Portfolio Wealth Creation RME at the end users workplace

The (UBX^{TM}) parallel computational nodes

The UBX™ can perform computations in parallel across multiple computers, i.e., computational nodes. The UBX™ synchronizes among all nodes, combines the intermediate results, and handles user tasks in a concurrent fashion.

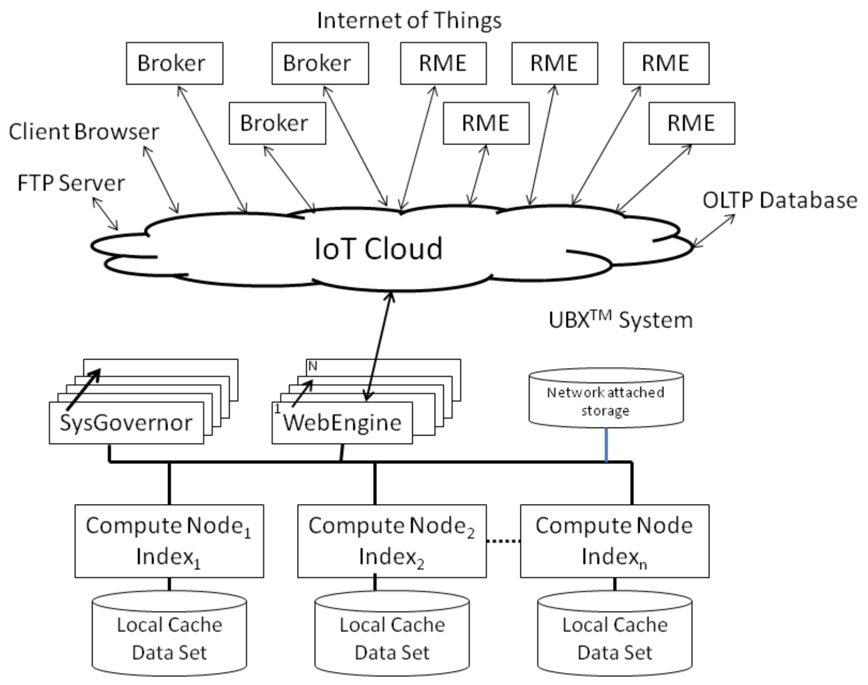


Figure 1. UBX™: user requests execute simultaneously on multiple nodes.

A logical view of UBX™ is shown in Figure 1. In the diagram, several computational nodes run separately on different computers. The data are distributed on each node. The preprocessor formulates and sends computational tasks to all nodes. After each node fulfill its tasks, the intermediate results are synchronized and added before sending to postprocessor. The postprocessor further manipulates the results and make a decision about the next step.

The (UBX^{TM}) key application concepts

The (UBX^{TM}) key application concepts can be represented as a layered pyramid of functions

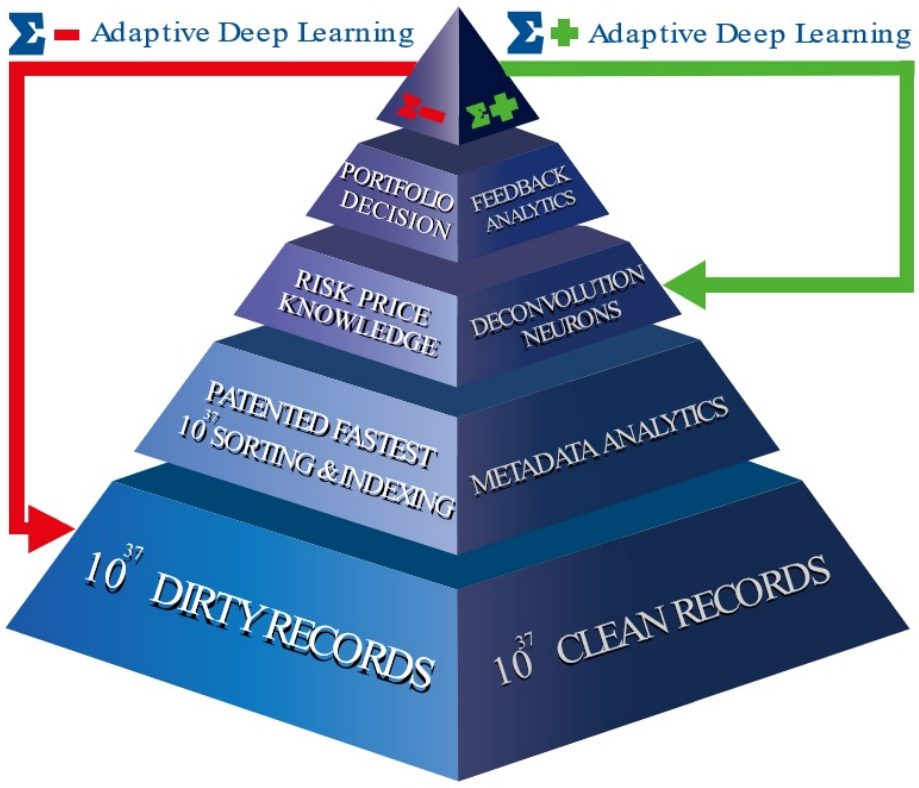


Figure 2.

The above conceptual illustration above shows how the problem is divided into the parts that interact directly to adjacent layers and with negative or positive feedback to other layers to achieve the solution, the information needed to make profitable trades.

The input to the system starts at the bottom of the pyramid (layer 1) and works up with data importing, cleaning, sorting, indexing, and generating metadata on the lower 2 layers.

The data is them transferred to layer 3 where the algorithms like the “Pattern Recognition Machine Learning Short Term Options Trading System” described below are executed.
The daily output of the models in layer 3 is used to the used to guide the trading decisions in layer 4 and the results of these decisions is be fed back into layer 3 to fine tune the process with adaptive Deep learning.

Basic trading strategy

QED trading strategies utilize holistically mathematical & statistical analysis, computational hardware, proprietary trading algorithms, artificial intelligence and availability of big data. The QED trading strategies are consisted of two champion-challenger models, QED model and DNN model, while consulting with the Black-Scholes model:

- QED model: Quantum Electro Dynamic Field Effect Option Pricing Model
- DNN model: Deep Neural Network Option Pricing Model
- Pattern Recognition Machine Learning

The QED and DNN models compute the probability of the range of price within which the underlying equity or index will fall, at the time option maturity. Such probability captures many factors such as market condition including current and historical spot prices, volume, option bid and ask premium, bid and ask sizes, time to expire, price volatility, etc. This probability is one of the most import metrics used in trading strategies. For example, in our short strangle trades, the price range of the given model predicated probability corresponds to the two strike prices we use.

Based on price range probability, profits from options trades in both bulls and bears market. For a given probability of spot price range at a future time, the fund constructs trades to make the most profit with the given level of risk. For instance, given a 3% of probability of a pair of out-of-money option strikes being exercised at maturity as computed by our models, the fund can place a short strangle trade and capture the premium at maturity, with 97% of confidence.

The trades with the acceptable risk reward will be relayed by the central UBX^{TM} to brokers to place the actual trade.

The Machine learning Algorithm for Personal Portfolio Wealth Creation

Pattern Recognition Machine Learning Short Term Options Trading System

In this patent application, we present an innovative trading system that uses multi-task machine learning to utilize underlying patterns implied in various technical analysis indicators for maximizing profit in short term options trading.

It is a common belief that the price of an actively traded stock reflects all the knowledge available in an efficient market and risk-free return is not attainable with public information about the underlying company and general market.

On the other hand, it is also a widely accepted practice for active stock traders to use a suite of technical analysis indicators to make their trading decisions. These indicators are obtained from historical recurring patterns.

However, the hundreds of such indicators and their variations lead difficulty in using them effectively for consistent profit. Successful trading using them is thus an art rather than science.

With this patent application, we propose a system that can make the best use of the available indicators, to achieve the maximal and consistent profit. It is done by using multi-task machine learning and by the proper selection of stock options.

In machine learning model, we assume the trades are done using multiple stocks, bought and sold on a daily basis. All the stocks are modeled jointly using the multi-task learning approach.

The prediction value is a linear model applied to a sigmoid function with the following form

$$\boldsymbol{p}(\boldsymbol{w},\boldsymbol{x}) = \tanh(\boldsymbol{w}^T\boldsymbol{x})\,,\#(1)$$

where $\boldsymbol{x}$ is the feature set, a vector of market data, such as daily high, low, open, close, plus a set of technical analysis indicators, such as MACD, RSI, Bollinger Band, trendline, various moving averages, Evening Star, Hanging Man, etc., and $\boldsymbol{w}$ is the weights to be trained. The type of trade, buy or sell, is determined by the sign of $\boldsymbol{p}$, e.g., positive value indicating a buy. Size of trade is decided by the magnitude of $\boldsymbol{p}$.

The relative daily price change is defined as,

$$\boldsymbol{y}(d) = \frac{C(d+1)-C(d)}{C(d)}\,,\#(2a)$$

where $C(d)$ is the daily close price of a stock at trading day d.

In addition to $\boldsymbol{y}(d)$ which is the relative price change in the following day, we also define a two-day forward relative daily price change,

$$\boldsymbol{z}(d) = \frac{C(d+2)-C(d+1)}{C(d+1)} = \boldsymbol{y}(d+1),\#(2b)$$

where $\boldsymbol{z}(d)$ is the forward stock return defined as the relative price movement two days beyond the current trading day, d. The following discussion on model training is based on $\boldsymbol{y}(d)$, but the same procedure applies equally well to $\boldsymbol{z}(d)$.

For a stock s and a giving trading day d, the profit is defined as our utility functions for optimization, defined as follows:

$$u(\boldsymbol{w},\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{p}(\boldsymbol{w},\boldsymbol{x})\boldsymbol{y} = \tanh(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x})\boldsymbol{y}\,.\#(3)$$

The overall utility function for all stocks, S over the training days D is expressed as

$$U(W) = \sum_{s=1}^{S}\sum_{d=1}^{D}\tanh\left(\boldsymbol{w}_s^T\boldsymbol{x}_{\boldsymbol{s},\boldsymbol{d}}\right)\boldsymbol{y}_{\boldsymbol{s},\boldsymbol{d}}\,.\#(4)$$

The optimization is to maximize $U(W)$, the total profit. To avoid overfitting, we add the following regularization term to bias toward smaller weights:

$$R_l = \lambda_l\sum_{f=1}^{F}\sum_{s=1}^{S}\sum_{d=1}^{D}\left\|W_{f,s,d}\right\|^2\,,\#(5)$$

where f represents the feature set of $\boldsymbol{x}$ of dimension F.

Multiple stocks are trained jointly, using the multi-tasking learning method. Each stock will have its own set of weights, $W_{.,s,.}$. The corresponding regularization term is defined as

$$R_s = \lambda_s \sum_{s=1}^{S} \left\| W_{.,s,.} - \text{avg}_s\left(W_{.,s,.}\right) \right\|^2, \#(6)$$

where the $\text{avg}_s()$ is the average done over stocks. It biases toward uniform weights among all stocks involved in the training.

The effectiveness of technical analysis indicators and the underlying dynamics of financial markets evolve with time. The model needs to maintain its stability against abrupt changes over time. To maintain temporal stationarity of the model, we use a time regularization term to minimize the variations of the weights $\boldsymbol{w}$ over different model training periods m:

$$R_m = \lambda_m \sum_{m=2}^{M} \| W_m - W_{m-1} \|^2 . \#(7)$$

With all of the above regularization terms considered, the optimization is to find the W that would minimize the $-U(W)$ plus the regularization terms:

$$\widehat{W} = \underset{W}{\text{argmin}} -U(W) + R_l(W) + R_s(W) + R_m(W) \#(8)$$

In terms of data preparation for the training, we collect the data at various time intervals, historical and current. Due to the large volume with the hundreds of technical indicators added, the computation resource requirement and efficiency would be of concern. We use distributed algorithms using multiple computing nodes to speed up the training.

The above optimization problem can be solved with LBFGS algorithm, which, for best computation efficiency, uses the closed form first derivatives of each term:

$$\frac{\partial U}{\partial w_{f,s}} = \sum_{d=1}^{D} \text{sech}\left(\boldsymbol{w}_s^T \boldsymbol{x}_{s,d}\right)^2 x_{f,s,d}\, y_{s,d} \#(9)$$

$$\frac{\partial R_l}{\partial w_{f,s}} = 2\lambda_l w_{f,s} \#(10)$$

$$\frac{\partial R_s}{\partial w_{f,s}} = 2\lambda_s \left(w_{f,s} - \text{avg}_s\left(W_{f,s}\right)\right) \#(11)$$

$$\frac{\partial R_m}{\partial w_{f,s,m}} = 2\lambda_m \left(w_{f,s,m} - w_{f,s,m-1}\right) \#(12)$$

The model training is performed over different historical periods, and optimal values of λ_l, λ_s, and λ_m are obtained using a grid search approach. Such optimal set of lambdas ensures the model do not overfit and remain stable over long historical periods.

The daily output of the models is used to guide the trading decisions. The computed weights for next day trades and forward day trades determines how the trades will be placed and whether each position will be held till the next day.

Since the trading strategy is to open and close the positions on a daily basis, the relative performance of the model can be measured against the daily price trends of the corresponding stocks with a simple daily buy-and-sell strategy. The following is a comparison of cumulative profits compared to the daily price change over the time from Aug 2001 to Jun 2018 for 20 sample stocks among the activated SP500 constituents:

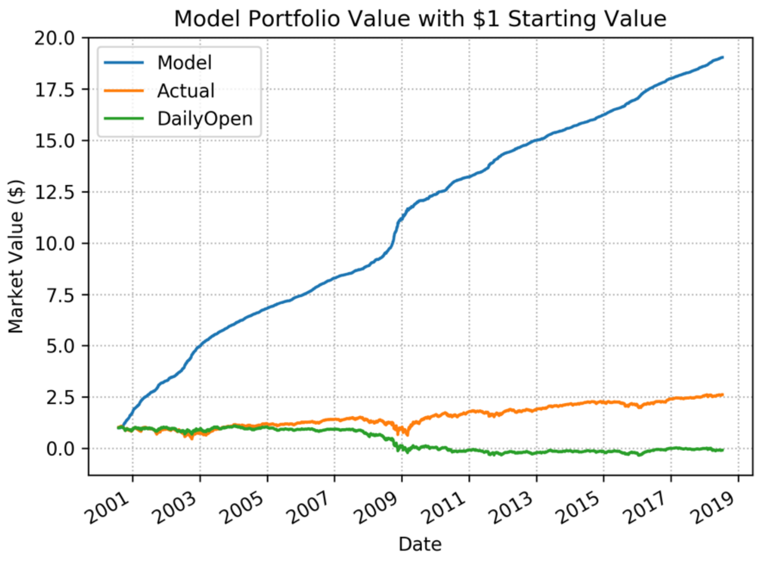


Figure 1. The Model value assumes buying or selling daily with equally weighted 20 stocks based on model predictions. The Actual value are the total $1 invested daily and equally in the 20 stocks since Aug 2001, no compounding, and hold for one day, with daily close prices. The Daily Open value simulates buying the 20 stocks with $1 at market open and selling their positions at market close. In all three cases, no trading cost is considered.

As shown in Fig. 1, the model performed well over an 18-year time span. However, active trading incurs large amount of trading cost. This cost can be reduced by holding the position longer by utilizing model prediction produced with the forward return rate, $\boldsymbol{z}(d)$, as defined in Eq. (2b). If the prediction based on $\boldsymbol{z}(d)$ has the same sign and large magnitude as that from $\boldsymbol{y}(d)$, we will hold the opened position for another day, to save on the trading cost and capture large favorable price move, until the relative predictions based on $\boldsymbol{z}(d)$ vs $\boldsymbol{y}(d)$ diverge far enough.

To further enhance the trading profit, we choose to use the stock options instead of the stocks themselves. We buy call options for positive $\boldsymbol{p}(\boldsymbol{w},\boldsymbol{x})$ or put options for negative $\boldsymbol{p}(\boldsymbol{w},\boldsymbol{x})$. Trading on options instead underlying stocks provides two benefits: 1) leverage, for the same amount of trading fund, larger shares can be bought. The effectiveness is mainly described by the Delta of options; 2) nonlinearity: the option price moves nonlinearly with the underlying stock price move, as quantified by the Gamma of the option.

The profitability depends on the values of Delta and Gamma, i.e., the first and second order derivative of option over stock prices. Ignoring the effect of risk-free interest rate and dividend yield for active trading, these Greeks are computed as follows:

$$Call\ delta\colon\ \Delta_C = N(v);\ v = \frac{\ln\left(\frac{S_0}{K}\right)}{\sigma\sqrt{t}} + \frac{\sigma\sqrt{t}}{2} \#(13a)$$

$$Put\ delta\colon\ \Delta_p = \Delta_C - 1\#(13b)$$

$$Gamma\colon \Gamma = \frac{\exp\left(-\frac{v^2}{2}\right)}{S_0\sigma\sqrt{2\pi t}}\ , \#(14)$$

where $N()$ is the standard normal probability density function. The larger the values of Delta and Gamma, the more potential is there for profitability.

According the above formula, the Delta ramps up when the spot price changes from out-of-money to in-the-money, and the Gamma is largest when the spot price, S_0, is near the strike, K. The larger the Gamma, the faster the option price accelerates when the underlying stock price moves in the right direction. The reverse is also beneficial when the stock price moves in the wrong direction for a model driven trade, the option prices drops slower when the underlying price moves farther away from the option strike. This property is used in our option trading system to the maximal benefit.

In summary, the proposed options trading system has the following special properties:

1. It utilizes multiple, if not all, available technical analysis indicators in access of a hundred, simultaneously.
2. It uses multi-task learning to model multiple stocks, multiple technical indicators, over multiple historical periods to calibrate the model
3. The strategy uses at least two independently calibrated models of maximizing the following-day and next-after-following-day price moves to reduce trading cost
4. It uses near-the-money stock options to further enhance profitability while using the trading capital efficiently.

Reconfigurable Matrix Engine (RME)

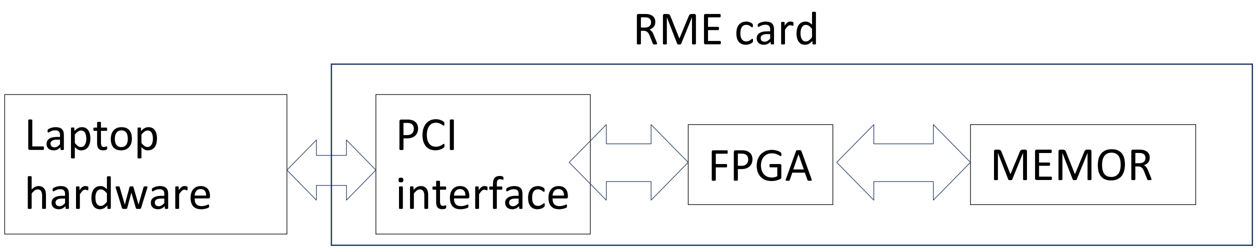
RME Hardware

The Reconfigurable Matrix Engine (RME) is used in several places in the Personal Portfolio Wealth Creation process:

It is used in the sorting and index part Level 1 of the pyramid
It is used to help implement the algorithms at Level 3 and 4 of the pyramid by doing the core parts of the DNN, Pattern Recognition, Deconvolution, QED algorithms

It is our intention to start with the RME initially implemented as much as practical with purchased hardware integrated by us into a RME.

Logical hardware description of a basic RME'



Application programming for Portfolio Wealth generator that can be executed using the RME

Sorting
DNN Tensor flow
Deconvolution
Block Manifold
User interface for use as a trading terminal

The RME is implemented as part of a Terminal used by the trader to do the calculations locally using data tickers and other data served up our UBX^{TM} central servers.

FPGA solution background

How does a FPGA solution provide faster computation to the same calculation for many different data groups situation?

The basic approach is to view the problem as a data flow representation with each calculation done on dedicated piece of the FPGA hardware configured for that calculation
The output for one step of the calculation feeds directly into the next step of the calculation instead of being copied to memory.

This makes it possible to be doing all steps on the complex calculation at the same time.
Another way to look at this data flow is as a pipeline.

Pipeline Configurations

Ideal for doing the same calculation on many data elements
Performance gains greater than 1,000 times are practical
Performance is the same for short or long calculations
One data element is completely processed for each pipeline time step.