

Article

Optimizing Portfolio in the Evolutional Portfolio Optimization System (EPOS) [†]

Nikolaos Loukeris ^{1,*} , Yiannis Boutalis ², Iordanis Eleftheriadis ³ and Gregorios Gikas ¹ 

¹ Department of Business Administration, University of West Attica, Petrou Ralli & Thivon Avenue, 12241 Athens, Greece

² Department of Electrical and Computer Engineers, Democritus University of Thrace, 67100 Xanthi, Greece

³ Department of Business Administration, University of Macedonia, Egnatias 156, 54636 Thessaloniki, Greece

* Correspondence: nloukeris@uniwa.gr

[†] This paper is a modified version of our paper: Loukeris, N.; Boutalis, Y.; Eleftheriadis, I. The Evolutional Portfolio Optimization System—(EPOS). In Proceedings of the 17th International Conference in Artificial Intelligence, WORLDCOMP 17—ICAI 17, Las Vegas, NV, USA, 27–30 July 2017.

Abstract: A novel method of portfolio selection is provided with further higher moments, filtering with fundamentals in intelligent computing resources. The Evolutional Portfolio Optimization System (EPOS) evaluates unobtrusive relations from a vast amount of accounting and financial data, excluding hoax and noise, to select the optimal portfolio. The fundamental question of Free Will, limited in investment selection, is answered through a new philosophical approach.

Keywords: higher moments; Isoelastic Utility Function; Jordan–Elman networks; recurrent networks; time-lag recurrent networks; MLP; neural networks; genetic algorithms; hybrids; portfolio optimization; Epicurus; Aristotle; logic; Free Will; eudaimonia

MSC: 91G10



Citation: Loukeris, N.; Boutalis, Y.; Eleftheriadis, I.; Gikas, G. Optimizing Portfolio in the Evolutional Portfolio Optimization System (EPOS).

Mathematics **2024**, *12*, 2729. <https://doi.org/10.3390/math12172729>

Academic Editor: Chuangyin Dang

Received: 23 June 2024

Revised: 8 August 2024

Accepted: 13 August 2024

Published: 31 August 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Two-phase processing during portfolio selection (i) initially evaluates the feasible portfolios, then determines an efficient frontier and excellent portfolio combination, considering the behavior of the investor; then, (ii) the minimum risk (variance, hyper-kurtosis, and ultra-kurtosis) is selected. The history of the portfolio optimization problem ranges from the mean-variance approach [1] on the randomness of stock prices [2] advanced to the stochastic forms on martingales [3]. In the modern theory of contingent claim valuation, and the Black and Scholes option pricing, [4] developed credit risk in Markov chains, and [5] developed mean-variance hedging (MVH) and mean-variance portfolio selection (MVPS) with limited information, resolving it explicitly in a Lévy triplet. The authors of [6] studied optimization on the Conditional VaR, [7] looked at dynamic mean-variance portfolio optimization with VaR constraints in continuous time, and [8] assessed the chaotic dynamics in portfolio optimization.

This research evaluates the initial step of optimization in a generic resolution, and in the next step, provides an EPOS-integrated system to choose portfolios in advanced approaches of finance and computational intelligence. The model of a single period is evaluated in various AI models of the recurrent networks family—recurrent networks, time-lag recurrent networks, Jordan–Elman networks,—and MLP in neural or hybrid neural–genetic forms of alternative topologies, with 40 in the recurrent networks, 44 in the TLRNs, and 66 in the Jordan–Elman networks, concluding on the highest Sharpe portfolio, aiming:

- (I) to examine the Free Will of investors to select the best investments logically, reflecting their behavior in further higher moments;

- (II) to establish the form of the Isoelastic Utility Function that optimally implements higher moments;
- (III) to extend the portfolio theory, filtering the fundamentals in the price and any available information, minimizing noise, hoax, manipulation, and defining healthy assets;
- (IV) to test the quality of the recurrent network family on neural or hybrid neural-genetic networks to define the optimal classifier for portfolio selection;
- (V) To provide the Evolutional Portfolio Optimization System (EPOS) to academia.

2. The Moments of Higher Order

The distributions of returns are not normal, independent, and identically distributed (n.i.i.d.) in reality, and the Efficient Market Hypothesis (EMH) fails significantly in the markets, as multiple effects impact the prices of stocks. The work in [9] noted that investors pay more attention to their potential losses. We will model these preferences, including the subconscious trends. Investors balance fears and expectations to allocate their wealth-expected utility. If they are logical, their expectations will be an average return based on historical values plus extra events, as the loss hazard will constrain their decisions. As most investors follow risk-averse patterns, fear manipulates them. During bearish times, the fear of maximizing losses encourages irrational herding behaviors, and in bullish times, the fear of loss on excess profits dictates a conservative trend from a specific point onward.

The hidden norms of investors' decision making are expressed in advanced higher moments. The authors of [10,11] remarked that the implied utility function of the HARA family (Hyperbolic Absolute Risk Aversion) in the fifth moment of hyperskewness and the sixth of hyper-kurtosis can be applied, as follows:

$$U_t(R_{t+1}) = aE_t(R_{t+1}) - bVar_t(R_{t+1}) + cSkew_t(R_{t+1}) - dKurt_t(R_{t+1}) + eHypSkew_t(R_{t+1}) - fHypKurt_t(R_{t+1}) + gUltraSkew_t(R_{t+1}) - hUltraKurt_t(R_{t+1}), \quad (1)$$

where:

$$Kurt_t(R_{t+1}) = Var_t^2(R_{t+1}) \quad (2)$$

$$HypKurt_t(R_{t+1}) = Var_t(R_{t+1})Kurt_t(R_{t+1}) = Var_t^3(R_{t+1}) \quad (3)$$

$$UltraKurt_t(R_{t+1}) = Kurt_t^2(R_{t+1}) = Var_t^4(R_{t+1}) \quad (4)$$

Equation (2) is a series of higher moments emphasized on the expected gain and risk detail. Thus, the utility function becomes, in standardized terms:

$$U_t(R_{t+1}) = \sum_{\lambda_v=1}^{\omega} (-1)^{\lambda_v+1} \frac{a_{\lambda_v}}{n} \sum_{i=1}^n \left(\frac{r_i - \sum \frac{r_i}{n}}{\sigma} \right)^n \quad (5)$$

where λ_v is the accuracy of utility to risk, ω is a very accurate utility representation parameter of the investor, n is the degree of higher moments we want to work until, a_{λ_v} is a constant in the investor profile, where $a_{\lambda_v} = 1$ for rational risk-averse individuals and $a_{\lambda_v} \neq 1$ for the non-rational, r_i is the value of return i at time t , and v is an identifier for the λ_v accuracy. The distributions of returns are not simulated, e.g., by the log-normal distribution, but rather calculated as they are. Equation (6) is produced by Equation (2) after calculations on the odd and even moments that represent return and risk in higher complexity. The Isoelastic Utility Function (a HARA derived from CRRA) to risk aversion becomes:

$$U = \begin{cases} \frac{W^{1-\lambda}-1}{1-\lambda}, & \lambda \in (0,1) \cup (1,+\infty] \\ \log(x), & \lambda = 1 \end{cases} \quad (6)$$

where W is wealth, and λ is the risk aversion counter. The authors of [10,11] noted that the Makowitz substratum can relax its elementary hypothesis to a normal distribution of prices. The Isoelastic Utility is an ideal function because it can support the calculation of higher-order moments, e.g., kurtosis, hyper-skewness, etc., without the limitations of the Quadratic or Power Utility that, because of its form, could not calculate more complicated derivatives, e.g., kurtosis

On the convex problem of quadratic utility maximization [1], the objective function is:

$$\min_x Uf(x) = \text{Var}(r_p) \quad (7)$$

where f , the risk function of the portfolio, is inadequate in real markets. The work in [12] incorporated higher-order moments in risk and return:

$$\min_x f(x) = \lambda \text{Var}(r_p) - (1 - \lambda)E(r_p) \quad (8)$$

$$r_p = \sum_i x_i r_i \quad (9)$$

$$x_i \geq 0 \quad (10)$$

$$\sum_i x_i = 1 \quad (11)$$

where r_p is the portfolio return, x_i is the weight of asset i , r_i is the return of the i^{th} asset, μ is the mean, and σ^2 is the variance.

3. A New Approach on the Problem

The authors of [10,11] indicated the necessity of further higher moments in the model, to optimally describe the preferences of investors:

$$\min_x f(x) = \lambda v_\gamma [b \text{Var}_t(r_p) + d \text{Kurt}_t(r_p) + f \text{HypKurt}_t(r_p) - h \text{UltraKurt}_t(r_p)] - (1 - \lambda) v_\gamma [a E_t(r_p) + c \text{Skew}_t(r_p) + e \text{HypSkew}_t(r_p) + g \text{UltraSkew}_t(r_p)] \quad (12)$$

$$v_\gamma = 1 - \varepsilon_\tau \quad (13)$$

$$r_p = \sum_i x_i r_i^* \quad (14)$$

where f is the portfolio function, v_γ is the financial health of the company (0 towards bankruptcy, 1 healthy), γ is an identifier of the financial health v_γ , ε_τ is the output of the heuristic model (0 healthy, 1 distressed), r_i^* is the return of stock i of the Efficient Frontier superior to the others, and x_i is the weights. The superiority of stocks in the portfolio is $i^* \sup j$ if and only if $R_t(i^*) > R_t(j)$, in the same time period t , analyzed as follows:

$$\text{Var } R_t(i^*) < \text{Var } R_t(j) \quad (15)$$

$$\text{Kur } R_t(i^*) < \text{Kur } R_t(j) \quad (16)$$

$$\text{HypKur } R_t(i^*) < \text{HypKur } R_t(j) \quad (17)$$

$$\text{UltraKur } R_t(i^*) < \text{UltraKur } R_t(j) \quad (18)$$

Stocks that do not meet all of the superiority terms are removed as sub-optimal [10,11]:

$$U_t(R_t(i)) = \sum_{\lambda_v=1}^{\omega} (-1)^{\lambda_v+1} \frac{a_{\lambda_v}}{n} \sum_{i=1}^n \left(r_i - \sum \frac{r_i}{n} \right)^n \quad (19)$$

Then, in the same time period t ,

$$U_t(R_t(i^*)) > U_t(R_t(x_j)) \quad (20)$$

Hence,

$$U_t(R_p) = U_t(R_t(i^*)) \quad (21)$$

The previous is identical to

$$\max_x E(U_p(w, \lambda)) \quad (22)$$

in

$$E(U_p(w, \lambda)) = \max \left\{ \frac{\sum_i [1 + \exp(r_i x_i)]^{1 - \frac{v_\gamma}{\lambda}}}{1 - \frac{v_\gamma}{\lambda}} \right\} / N \quad (23)$$

Let

$$\text{Var}_t^2(r_p) = z \quad (24)$$

and

$$\text{Var}_t(r_p) = y \quad (25)$$

as

$$z = y^2 = \sigma^4 \quad (26)$$

then

$$\min_x Uf(x) = \lambda v_\gamma \text{Var}_t(r_p) [b + dy + fz + hzy + kz^4] \quad (27)$$

The non-convexity of the problem needs robust heuristics. The novel contribution is that we extract hidden accounting and financial prototypes on a more precise asset evaluation, excluding hoax and manipulation. I filter the distressed companies with no significant potentials from portfolios. The evaluation v_γ is more important than the investor's risk behavior, as they have a reverse influence in v_γ/λ . The $\min_x f(x)$ equality defines a categorical, objective influence of an asset as more influential than subjective investors' behavior. The flow chart of processes is described in Figure 1.

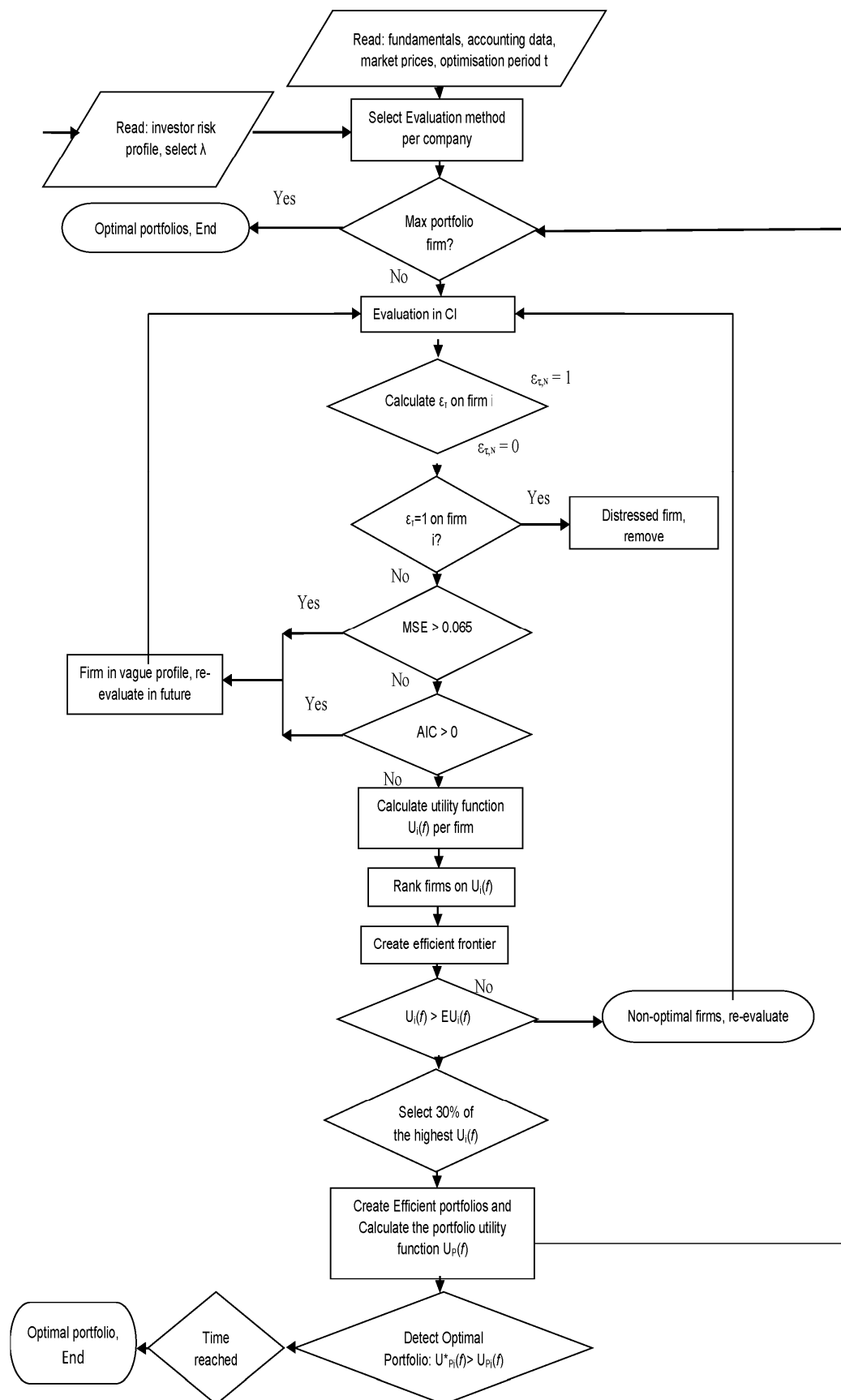


Figure 1. The EPOS model's flow chart.

4. The Evolutional Portfolio Optimization System—EPOS

The integrated Evolutional Portfolio Optimization System (EPOS) first reads the market prices, the accounting data, the fundamentals, and the time period.

Then, it proceeds by selecting the initial method to evaluate the companies whose stocks are candidates in the portfolio. At this step, the individual investor's risk profile is given and λ is selected for the Isoelastic Utility.

Next, it checks if this is the last stock, and if the optimal portfolio condition is fulfilled.

Else, it moves on to evaluate initially within a Computational Intelligence model, to form two subsets: subset A, representing healthy stocks, and subset B, representing distressed stocks. Currently, we choose the best network of the examined nets. The ε_τ value is calculated (0 healthy and 1 distressed).

If $\varepsilon_\tau = 1$, then the firm is distressed and it is removed; else, if $\varepsilon_\tau = 0$, the firm is healthy and thus a candidate for the optimal efficient portfolio.

In the next step, $U_t(R_t(i))$, the utility function, is calculated per firm.

Next, firms are ranked according to their utility score.

Then, the Efficient Frontier is calculated.

Next, the firms with the higher utility score are selected for the efficient portfolio.

The sub-optimal firms as well as the non-optimal firms are revaluated with potential new data in step 4 of the Neural Net evaluation, following all the steps.

Next, after the efficient portfolio is created, its Utility Function is calculated $U_{P_j}(f)$.

Then, the optimal overall portfolio $U^*_P(f)$, whose utility is the maximum available, is detected, if possible, by all the available efficient portfolio utilities $U_{P_j}(f)$ recorded in $U^*_P(f) > U_{P_j}(f)$.

The process stops when the time limit is reached and the EPOS has the optimal portfolio. The key idea is to filter fraud and speculative noise that interfere with the price and disorient investors. Thus, examining recent accounting entries and using their financial indexes, we can define the real financial health of the firm. After the real healthy firms are selected, then their returns are considered in the model and we proceed on the main core of the Markowitz initial approach, the detection of the efficient frontier and the creation of the efficient portfolio.

5. Intelligent Computing

The research emphasizes the classifiers of the Recurrent family: (i) recurrent, (ii) time-lag recurrent, (iii) Jordan–Elman, and MLP as benchmarks, all in neural net and hybrid neural–genetic systems, with various topologies.

5.1. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) form their connections in a directed cycle, creating a network of neurons with feedback connections. Recurrent Neural Networks operate differently from feedforward neural networks, during their computational behavior and training (Figures 2–7). Recurrent Neural Networks may behave chaotically; thus, dynamical systems theory is able to model and analyze them. The novel feedback network of Long Short-Term Memory (LSTM) [13] overcomes the fundamental problems of traditional RNNs, and efficiently learns to solve many previously unlearnable tasks involving [14–16]:

- I. Recognition of temporally extended patterns in noisy input sequences;
- II. Recognition of the temporal order of widely separated events in noisy input streams;
- III. Extraction of information conveyed by the temporal distance between events;
- IV. Stable generation of precisely timed rhythms, smooth and non-smooth periodic trajectories.

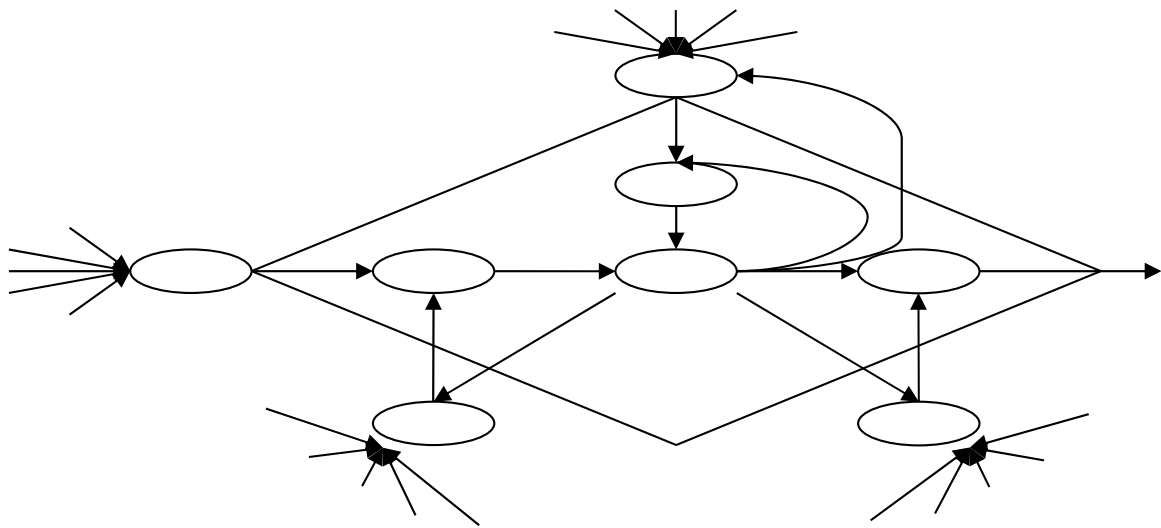


Figure 2. Recurrent Neural Network of 1 hidden layer.

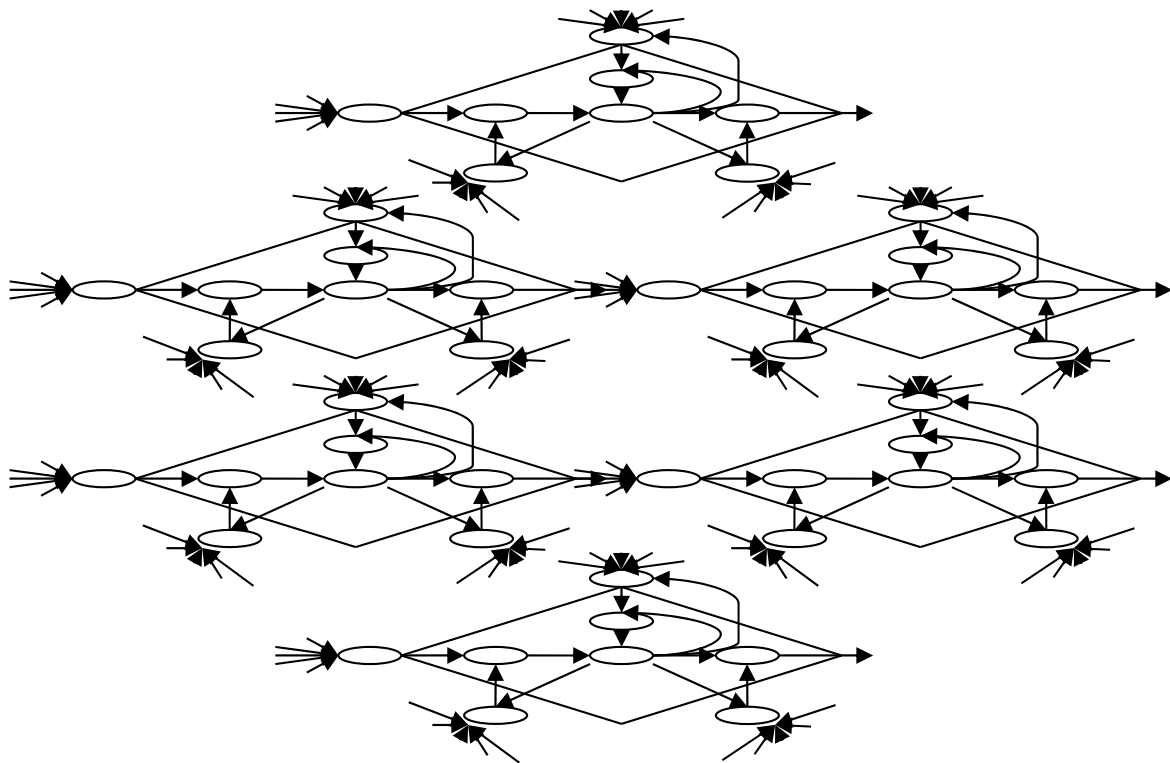


Figure 3. Recurrent Neural Network of multiple layers.

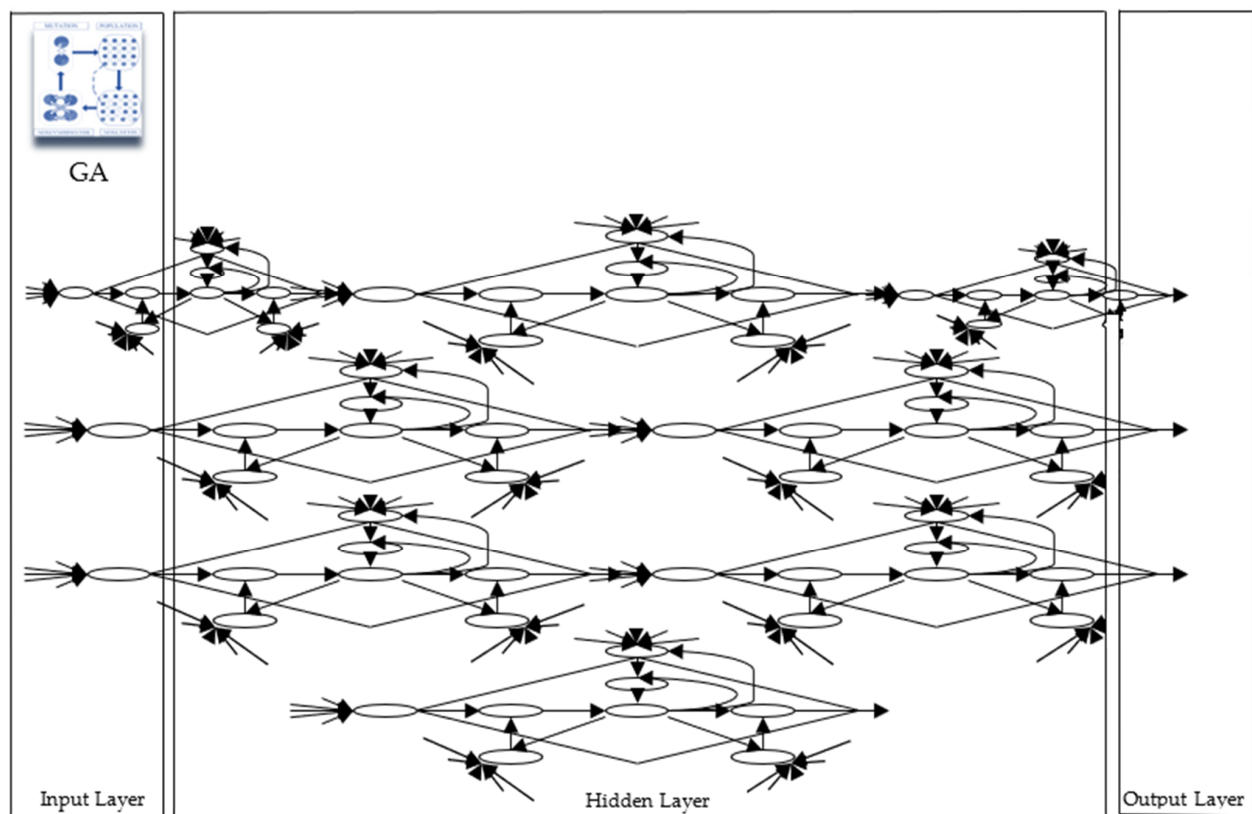


Figure 4. Hybrid Recurrent Net on input Genetic Algorithms—GAs.

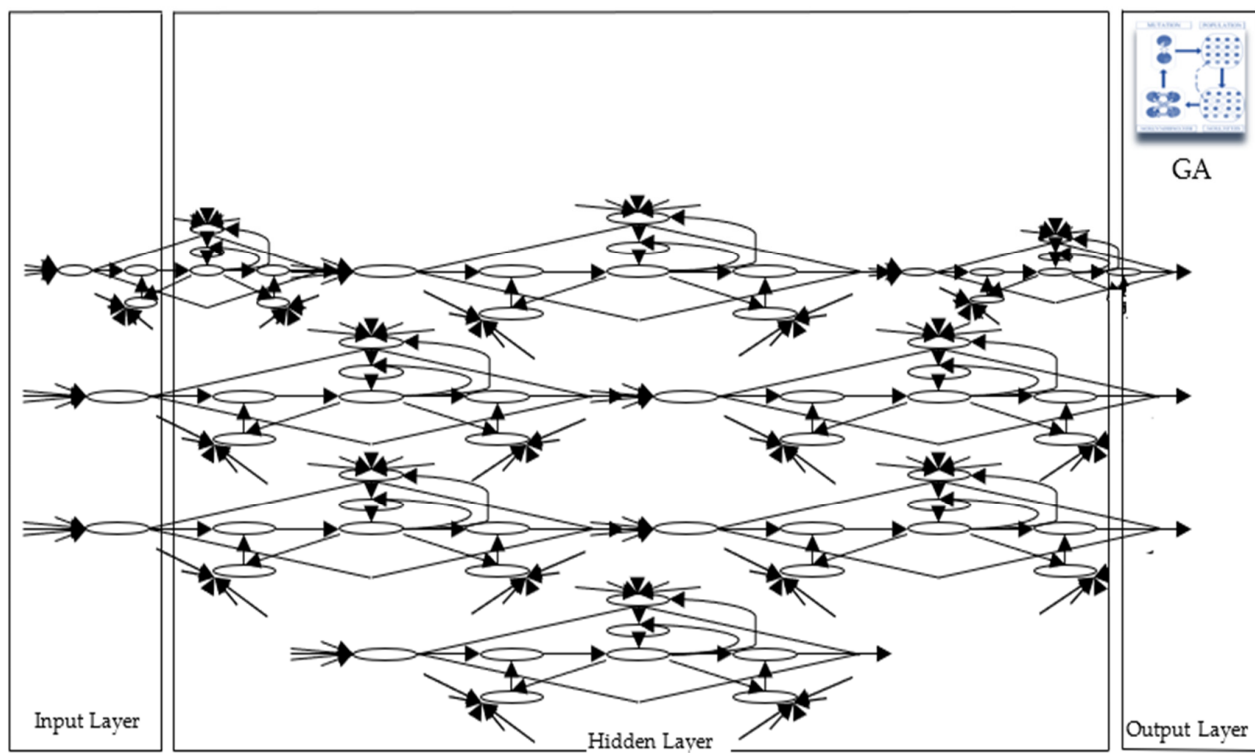


Figure 5. Hybrid Recurrent Net on output GAs.

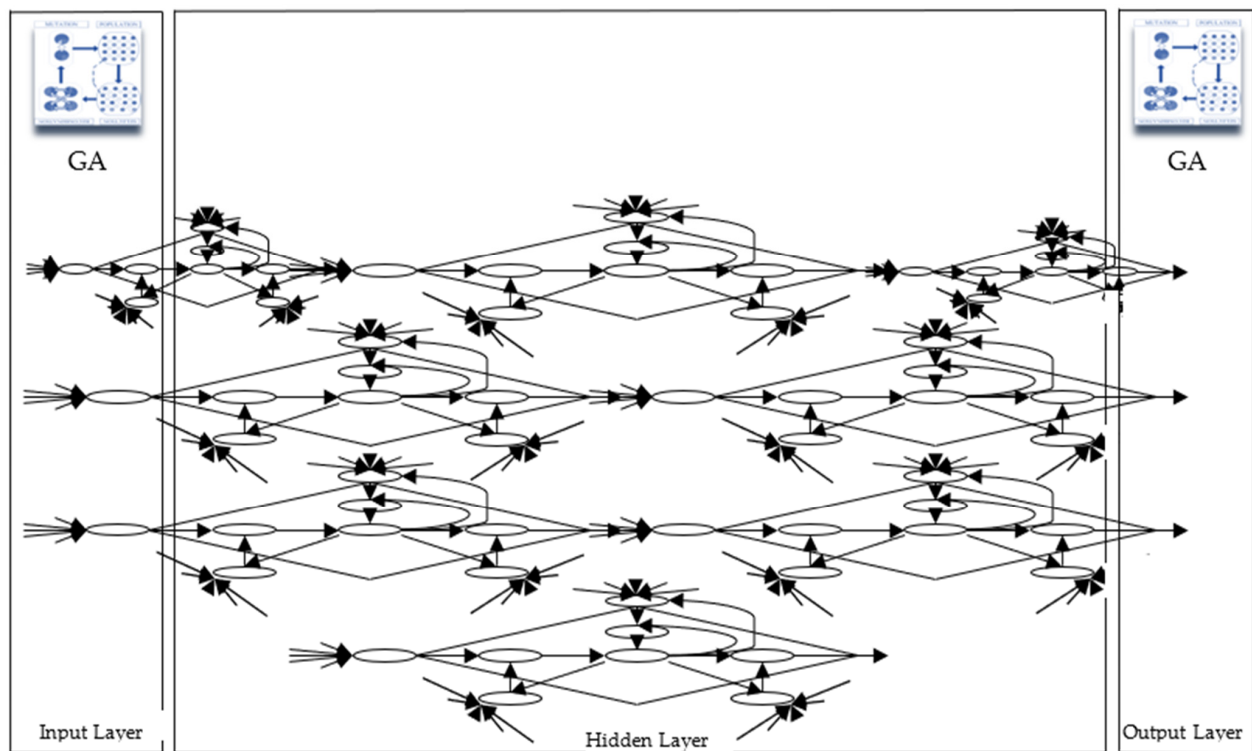


Figure 6. Hybrid Recurrent Net on input and output GAs only.

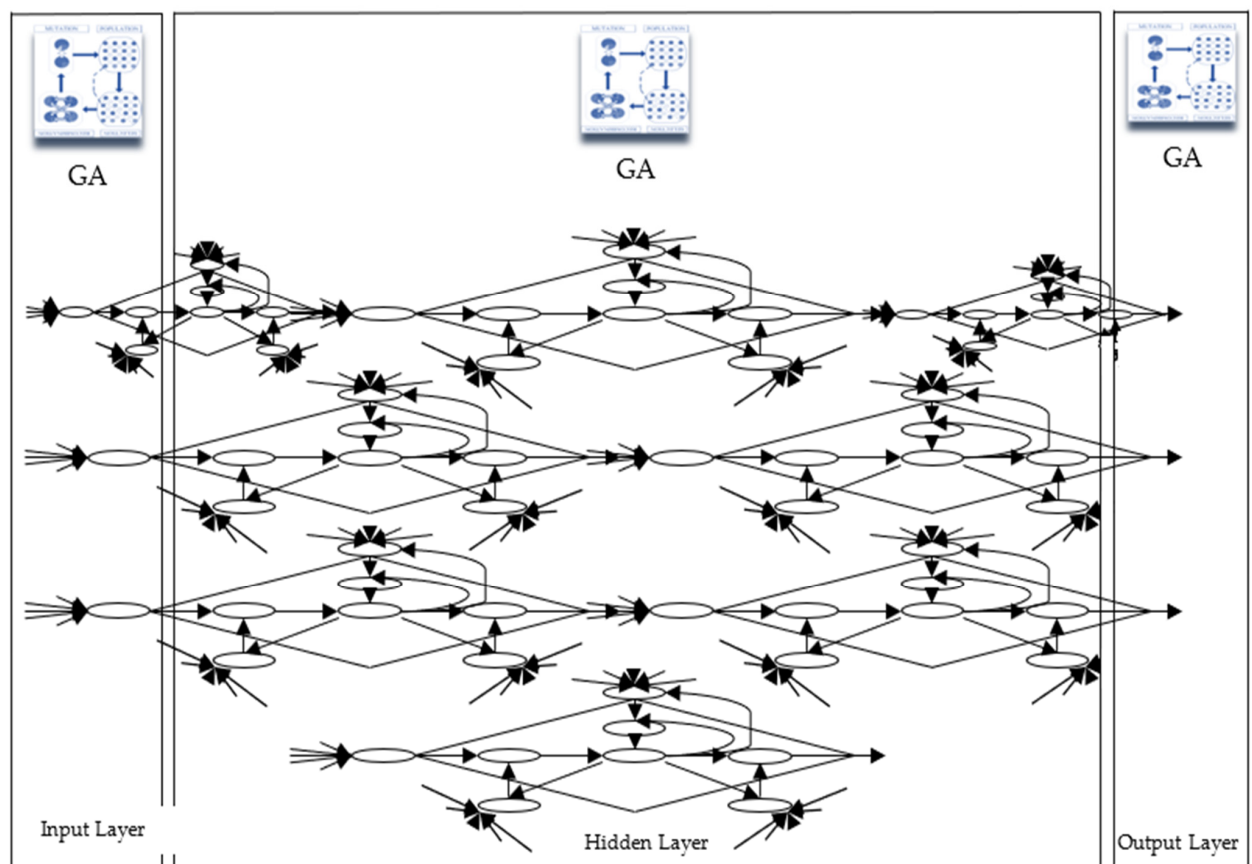


Figure 7. Hybrid Recurrent Net in GAs on all layers and/or cross-validation.

The Recurrent Neural Networks were optimized by Genetic Algorithms [17,18], implementing four different topologies (Figure 1).

Hybrid Recurrent Neuro-Genetic Networks

The Recurrent Networks were optimized with Genetic Algorithms that determined the inputs, and split the neuron performance in every layer into different topologies. Online learning was preferred to update the weights of the hybrid neuro-genetic Recurrent Networks, optimizing the (a) quantity of artificial neurons, (b) step size, and (c) momentum rate, whilst the output layer had optimized step size and momentum of the genetics. The significance of each financial index out of the 16 inputs in hybrids is unknown in the system; thus, GA chose them. Each model is trained for numerous periods, optimizing the necessary inputs in the lowest MSE. The Genetic Algorithms were elaborated on alternative forms: (i) on the input layer only, (ii) on the input and output layers only, (iii) on all layers, (iv) on all layers and with cross-validation, in different topologies. The competitive rule was the Conscience Full function in the Euclidean metric, as the conscience mechanism counts the frequency with which a neuron wins the competition, thus enforcing a constant winning rate across the neurons. The TanhAxon transfer function is used on the momentum learning rule. The hybrid nets require multiple training sessions to obtain the lowest error. The output layer elaborated Genetic Algorithms in some hybrids, optimizing the step size and the momentum, and it receives discrete values of $\{0, 1\}$, where a_i is the inputs of the neuron and w_i is the weights of the neuron. The non-linear activation function φ specifies the activation level of the neuron. This model is implemented here as a basis for further comparison to more advanced systems.

5.2. Time-Lag Recurrent Networks

Time-lag recurrent networks (TLRNs) are Multi-Layer Perceptrons (MLPs) that have been extended with short-term memory formations with local recurrent synapses (Figures 8–12). TLRNs are suitable for processing temporal information such as temporal pattern recognition, time series prediction, etc. TLRNs implement the Backpropagation Through Time (BPTT) training algorithm, a superior algorithm to the standard backpropagation. Recurrent Neural Networks use a modification of the feed-forward networks of multiple layers, adding a context layer that retains data between observations, and hence, each time step offers new inputs to the network. The previous outcomes of the hidden layer are given to the context layer and re-feed the hidden layer in the next time step. The context layer contains no data initially, and the output from the hidden layer after the first input into the network will be identical, as if there is no context layer. The weights are processed in the same manner for the new connections from and to the context layer from the hidden layer.

Time-lag recurrent networks require a smaller network to learn temporal problems than Multi-Layer Perceptron (MLP), which requires additional inputs to represent the past samples. TLRN is biologically more plausible and computationally more vigorous than relevant adaptive models. Noise influence on TLRNs is low, and the repetition of the TLRN grants the benefit of an adaptive memory depth. TLRNs implement nonlinear moving average (NMA) models. With their global feedback from the output to the hidden layer, TLRNs can be extended to nonlinear autoregressive moving average (ARMA) models, into applications of optimal control, surpassing the performance of their linear equivalents. TLRNs preprocess inputs in case the memory layer is restricted to the input. Time-lag recurrent networks are an exceptional alternative to brute force approaches, with long time periods that resolve the problem of creating very large networks by brute force that are difficult or impossible to train. The TLRNs that are implemented currently use the Gamma function in the neuron memory in the form of flowing integrators. The Gamma TLRN network is a very popular model among the three memory structures of these neural nets: (i) TDNN memory, which deploys the flow of ideal delays, as a delay of one sample; (ii) gamma memory; and (iii) Laguerre memory, which is slightly more advanced than gamma memory, as it orthogonalizes the memory space, which is useful in large memory

kernels. The Focused topology was selected, aiming to incorporate the memory kernels connected to the input layer. Thus, only the past of the input is remembered. If the Focused switch is not set, the hidden layers' PEs will also be equipped with memory kernels. The Depth in Samples parameter (D) is used to compute the number of taps (T) contained within the memory structure(s) of the network. The number of taps within the input memory layer is dependent on the type of memory structure used. For the Time Delay Neural Net Axon (TDNNAxon), the number of input taps T_0 is equal to the depth D . The formula for the other two memory types is

$$T_0 = 2D/3 \quad (28)$$

The number of taps for the memory structures at hidden layer n is computed (for all memory types) by the formula

$$T_n = T_0/2n \quad (29)$$

This is only used as a foundation level for the memory depth, since the depth will be modified by the network. The Trajectory Length communicates to the Samples (Exemplar) setting inside the Dynamic Controller and thus determines the manner of reading on numerous samples, before the backpropagation occurs. Alternatively, it can be considered as a complete pattern of samples. The default setting is generally two times the memory depth, but it can be adjusted based on the characteristics of the underlying data [14–16]. The signal at the taps of the gamma memory can be represented by

$$x_0(n) = u(n) \quad (30)$$

$$x_k(n) = (1 - \mu)x_k(n-1) + \mu x_{k-1}(n)x_{k-1}(n-1) \quad (31)$$

$$k = 1, \dots, K$$

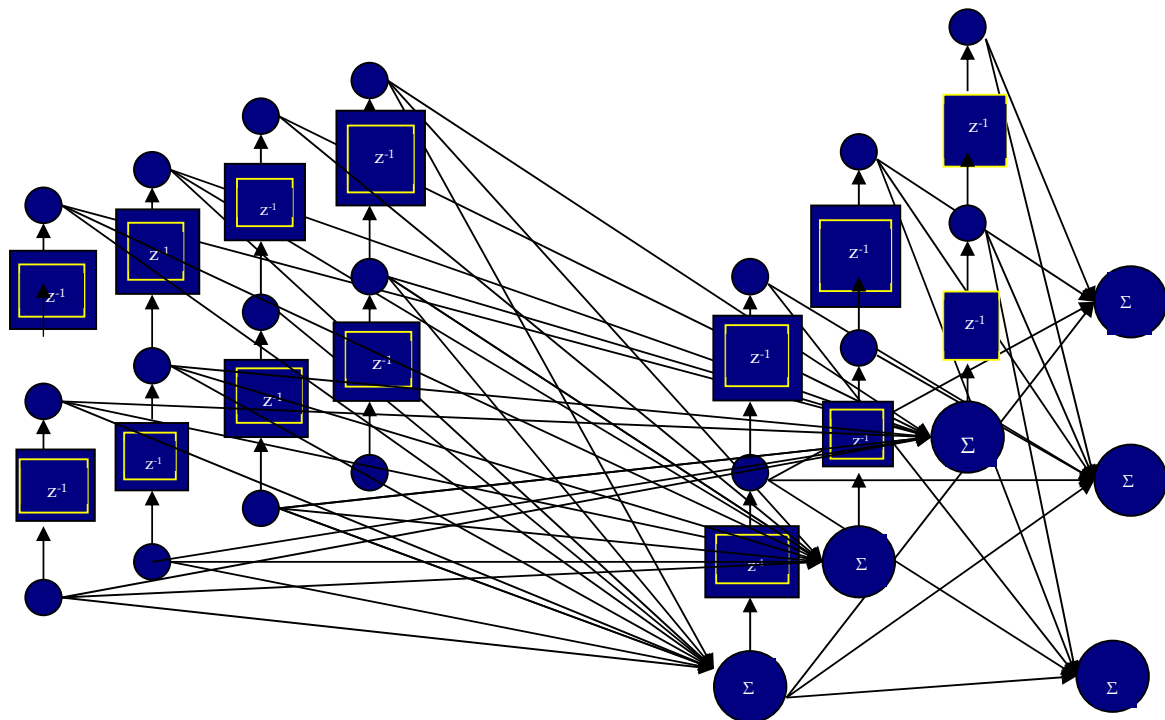


Figure 8. A representation of time-lag recurrent neural networks.

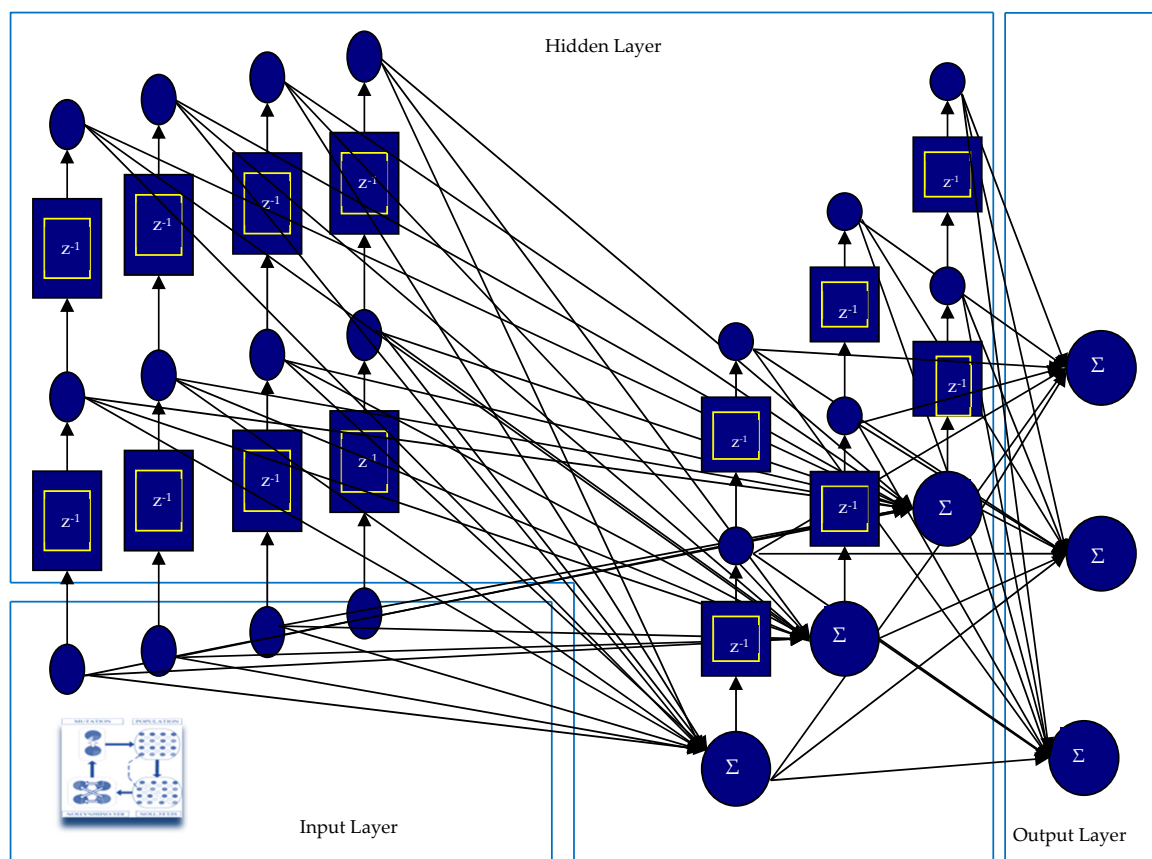


Figure 9. Hybrid time-lag recurrent networks in input Genetic Algorithms—GAs.

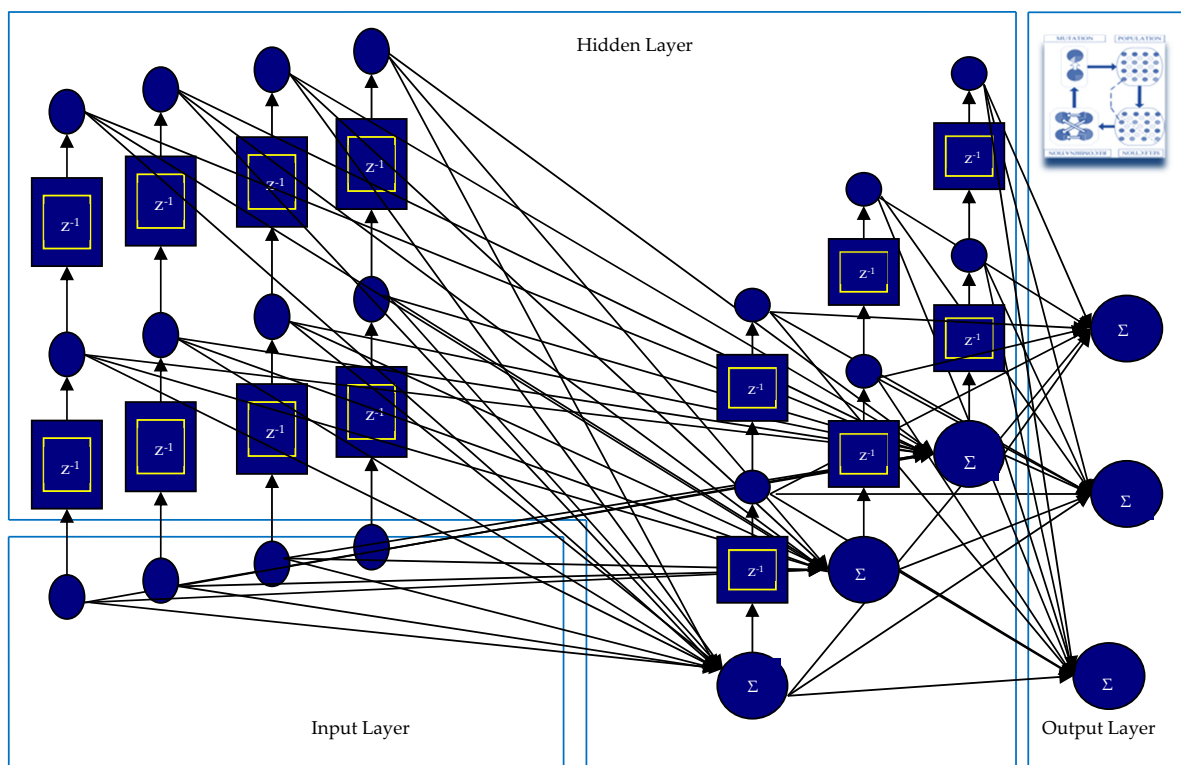


Figure 10. Hybrid time-lag recurrent networks in output GAs.

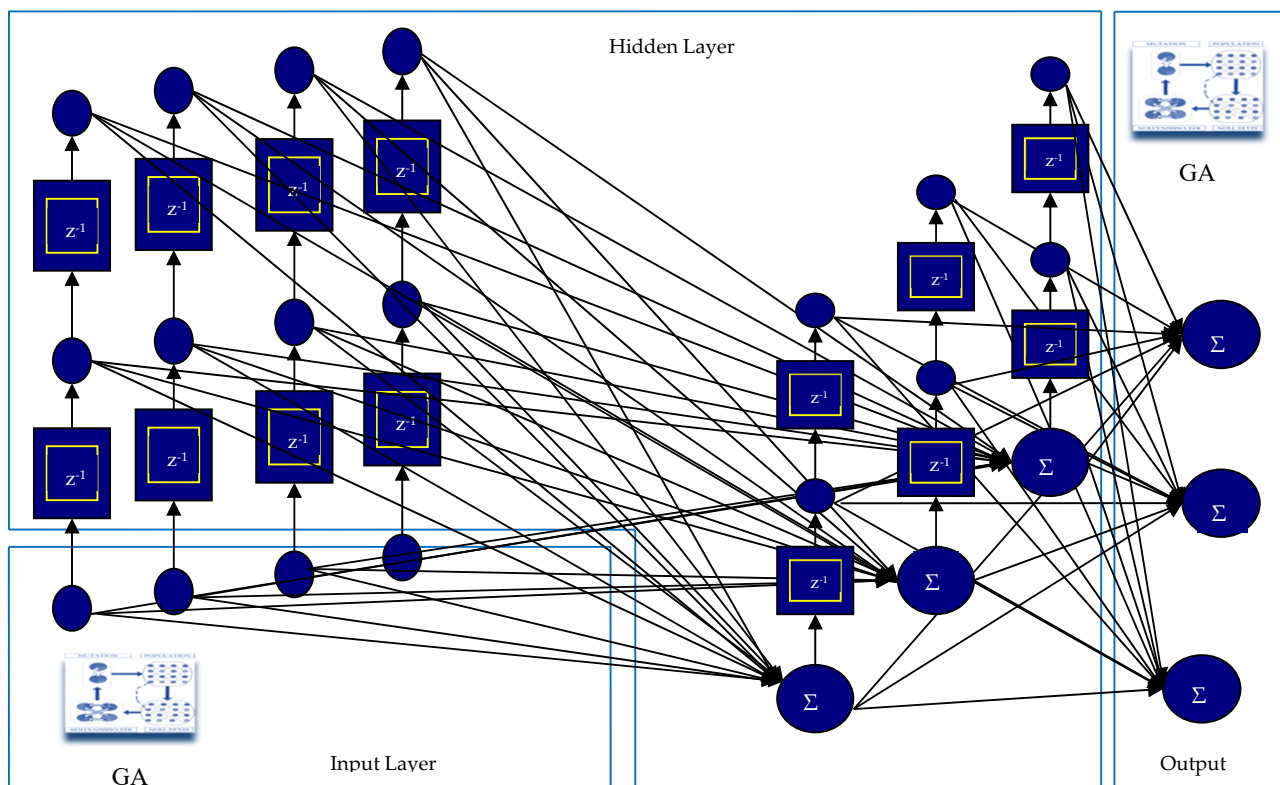


Figure 11. Hybrid time-lag recurrent networks in input and output GAs.

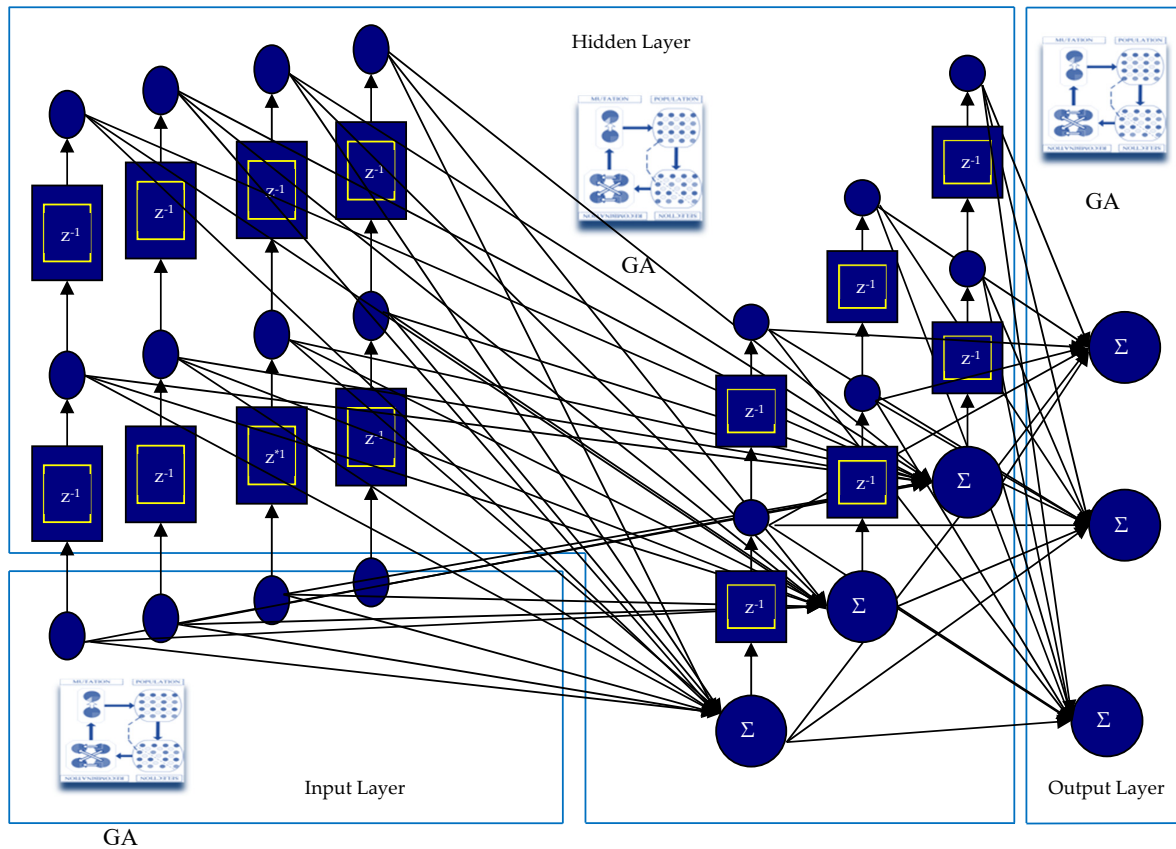


Figure 12. Hybrid TLRN in all layers—GAs and/or cross-validation.

The signal at tap k is a smoothed version of the input, which holds the voltage of a past event, creating a memory. The point in time where the response has a peak is approximately given by k/m , where m is the feedback parameter, indicating that the neural net can control the depth of the memory, adjusting the value of the feedback parameter instead of the input number. The parameter m is recursive and can be adapted using gradient descent procedures just like the other parameters in the neural network, demanding a more powerful learning rule to be applied, which is provided by Backpropagation Through Time (BPTT). Several different memory structures have recently been applied with some advantages, such as Laguerre memory, based on the Laguerre functions. The Laguerre functions are an orthogonal set of functions, built from a low-pass filter followed by a cascade of all-pass functions. This family of functions constitutes an orthogonal span of the gamma space, possessing the same properties as the gamma memories, but they may display faster convergence in some cases. Laguerre functions have the following form:

$$L_i(z, \mu) = [z^{-1} - (1 - \mu)]^{i-1} / [1 - ((1 - \mu)z^{-1})^i] \sqrt{1 - (1 - \mu)^2}, \quad (32)$$

where $i = 1, 2, \dots$

providing a recursion equation of the form

$$x_0(n) = (1 - \mu)x_0(n - 1) + \sqrt{1 - (1 - \mu)^2} u(n) \quad (33)$$

$$x_k(n) = (1 - \mu)x_k(n - 1) + x_{k-1}(n - 1) - (1 - \mu)x_{k-1}(n) \quad (34)$$

where $u(n)$ is the input signal. Memories can be appended to any layer in the network, producing very sophisticated neural topologies capable of complex analyses (Figures 13–15).

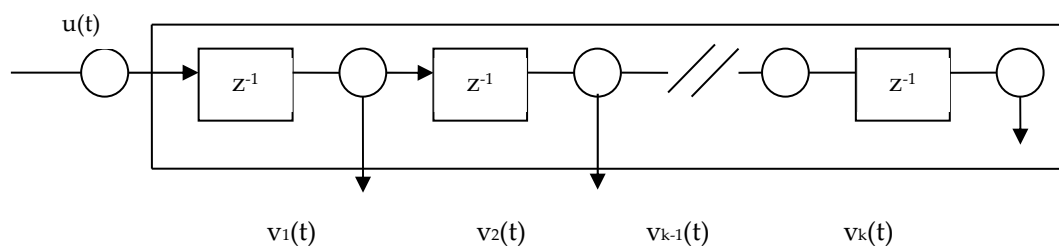
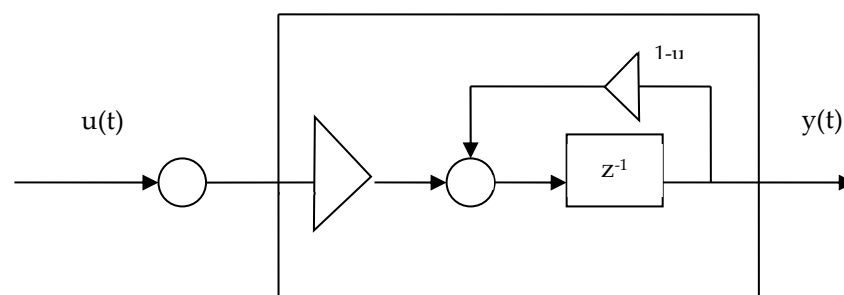
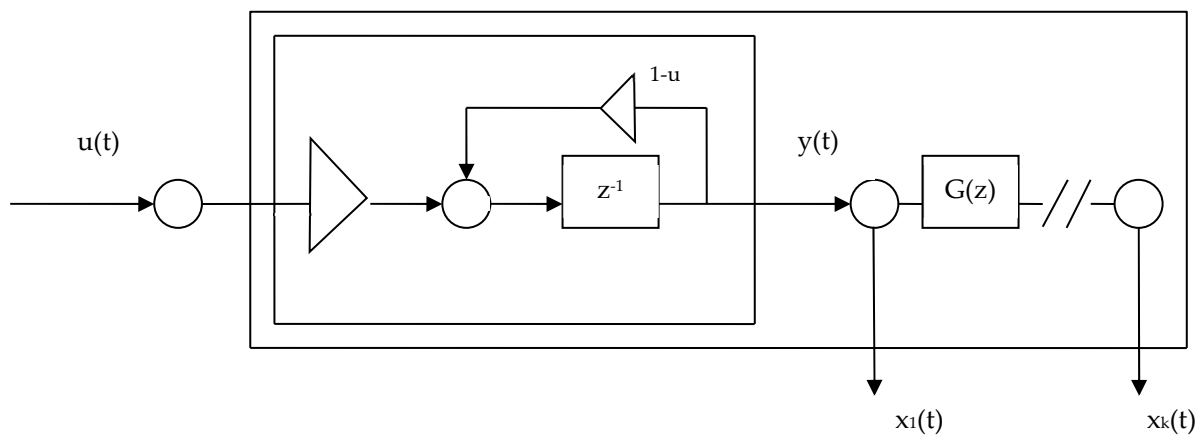


Figure 13. Tape relay line. Source: NeuroDimensions Inc., Sandwich, Massachusetts, USA.



Delay operator: $\mu/[z - (1 - \mu)]$, memory depth: $1/\mu$, memory resolution.

Figure 14. Context unit. Source: NeuroDimensions Inc., Sandwich, Massachusetts, USA.



Delay operator: $\mu/[z-(1-\mu)]$, memory depth: k/μ , memory resolution: μ .

Figure 15. Gamma memory. Source: NeuroDimensions Inc.

Time-Lag Recurrent Networks with Genetic Algorithms

Time-lag recurrent networks (TLRNs) were implemented in a hybrid form with Genetic Algorithm optimization. The significance of each one of the 16 financial indices, which are the inputs of the TLRN, is not predefined; thus, we used Genetic Algorithms to select or de-select the important inputs. This form of optimization requires that the network be trained multiple times in order to find the combination of inputs that produces the lowest error. Genetic Algorithms were used in each hidden layer implemented in TLRN with different topologies. The weights of hybrid neuro-genetic TLRN were chosen to be updated through online learning, after the presentation of each exemplar. In contrast, batch learning updates the weights after the presentation of the entire training set, and it was rejected. Accumulation of the gradient contributions for all data points in the training set before updating the weights is referred to as batch learning [19]. In online learning, the weights are updated immediately after seeing each data point. The gradient for a single data point can be considered a noisy approximation to the overall gradient G , called stochastic (noisy) gradient descent. Online learning has a number of advantages [19]:

- It is often much faster, especially when the training set is redundant.
- It can be used when there is no fixed training set.
- It is better at tracking non-stationary environments.
- The noise in the gradient can help to escape from local minima.

Many powerful optimization techniques (conjugate and second-order gradient methods, support vector machines, Bayesian methods, etc.) are batch methods that cannot be used online [19].

A Genetic Algorithm was used in (a) neurons, (b) step size, and (c) momentum rate, solving the sub-problem of the optimal values for these three parameters. This form of optimization requires that the network be trained multiple times in order to find the settings that produce the lowest error. The output layer was chosen to implement Genetic Algorithms, optimizing the value of the step size and the momentum.

5.3. Partially Recurrent Neural Networks

Partially Recurrent Networks are MLP nets where a few recurrent connections are introduced. The input layer of Partially Recurrent Networks includes two types of neurons: the neurons that behave as inputs, receiving external signals, and the context neurons, or neurons of state, that remember past actions and take output values from one of the layers delayed by one step. Internal states, which function as a short-term memory, of the partially recurrent neural nets, can predict time series, as they can represent information about the preceding inputs [20]. The Partial Recurrent Networks are (i) the Jordan network, (ii) the Elman network, and (iii) the Multi-Step Recurrent network.

5.4. The Jordan Network

The authors of [21,22] created the Jordan neural nets, where the context neurons receive a copy from the output neurons and from themselves; thus, many context neurons are the outputs. The recurrent connections from the output layer to the context neurons have an associated parameter of constant value: $m \in (0, 1)$ (Figures 16 and 17).

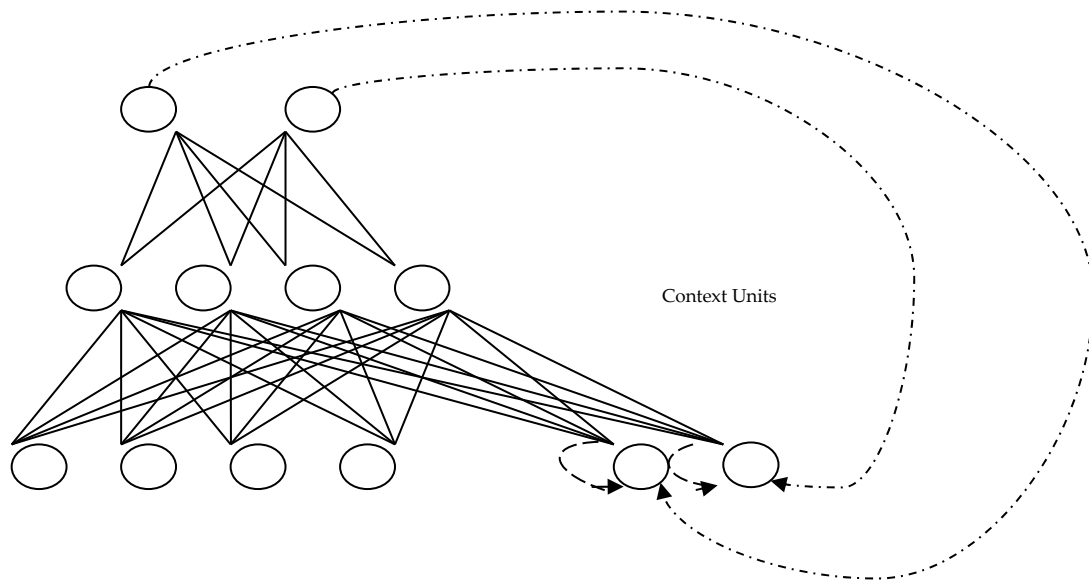


Figure 16. The Jordan network of a single hidden layer (1986), $\lambda \in [0, 1]$.

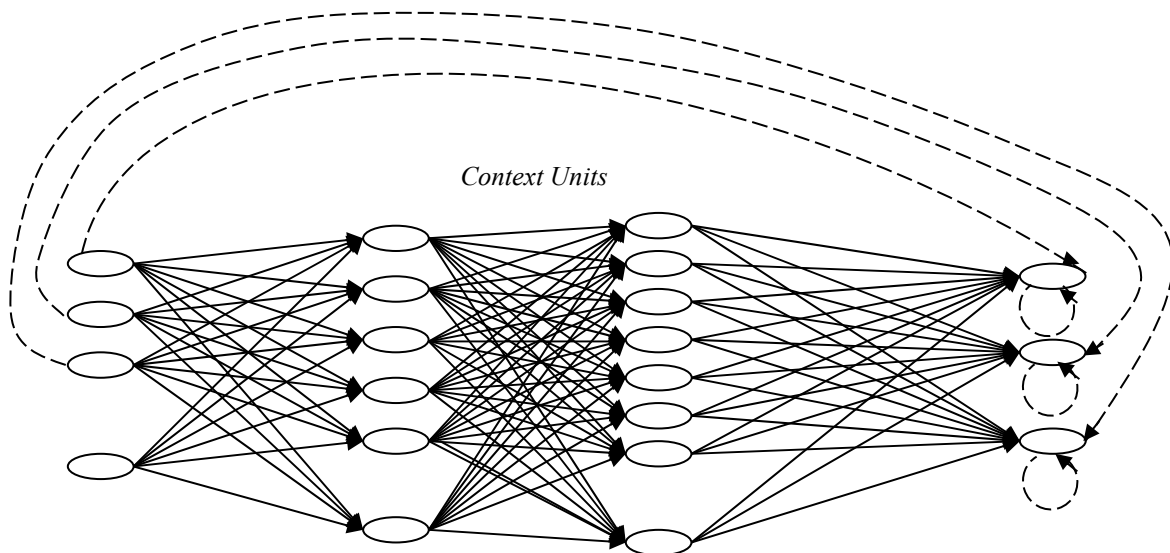


Figure 17. The Jordan network of multiple layers (1986), $\lambda \in [0, 1]$.

5.5. The Elman Network

The work in [23] created the Elman nets, where the context neurons receive a copy of the networks' hidden neurons, and these connections do not need to associate any parameter. Thus, the number of the context neurons is identical to the number of hidden neurons in the network. The remaining activations are calculated similarly as in a Multi-Layer Perceptron, considering the sequence of external inputs and context neurons as the vector input to the network (Figures 18 and 19).

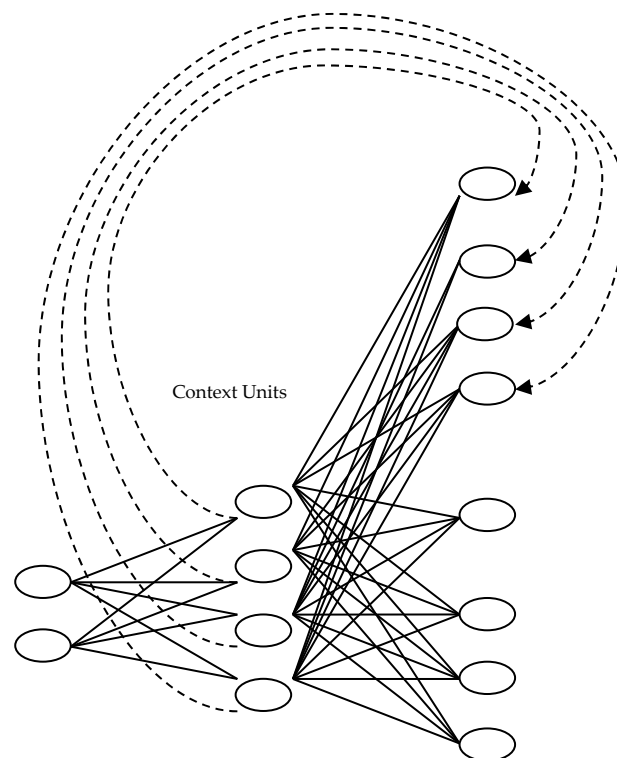


Figure 18. The Elman network (1990) of a single layer.

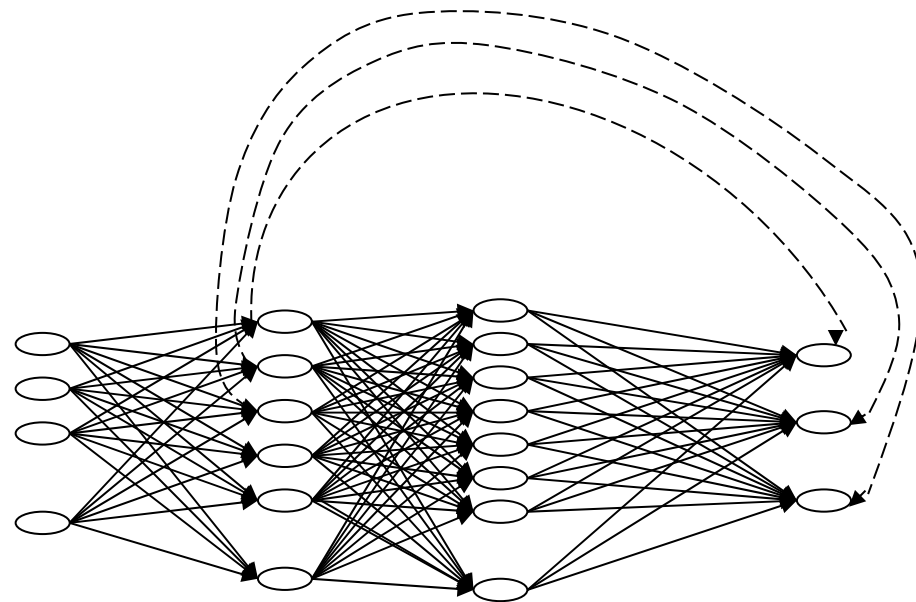


Figure 19. The Elman network of multiple layers.

5.6. The Multi-Step Recurrent Network

In the Multi-Step recurrent network [24], the feedback connections are directed from the output neuron to the input layer. The context neurons memorize previous outputs of the network. The number of input and context neurons is replaced in every sampling time period by other neurons in cases of prediction.

5.7. The Jordan–Elman Networks

The Jordan and Elman networks extend the MLP, implementing neurons that remember past activity, the context units. These context units offer JE models the ability to extract

temporal information from the data. There are available four essential topologies that are altered by the layers that feed the context units. Topology I provides the context units with the inputs, and builds a robust past substratum of the input by its memory traces. Topology II follows Elman's method and builds memory traces from the initial hidden layer. Topology III elaborates the past of the last hidden layer outputs as input to the context units. Finally, topology IV works with Jordan's method, taking the past of the output layer to create the memory traces. In this research, we elaborate the initial topology I, which takes account of the inputs' past.

The context neuron memorizes the past of its inputs in the recency gradient, which uses oblivion in an exponential decay. Thus, the memory of recent data is better than the memory of those in the more distant past. The neurons of the context unit manage the forgetting factor in the time constant, which takes values between 0 and 1. In the case of 0, only the present time of data is considered, indicating that there is no self-recurrent connection, whilst for a value of 1, all of the past is factored in. The closer the value is to 1, the longer the memory depth and the slower the forgetting factor.

The pull-down menu within the Context Units box is used to select the transfer function of the context units. There are linear and nonlinear context units and linear and nonlinear integrators. The integrators are the same as the context units, except that they normalize the input based on the time constant.

The number of neurons in the context layer is defined by the number of neurons in the layer that feeds the context layer (i.e., the network will assign one context unit per input connection). As with the other neural models, the number of hidden layers must be defined. Note that if there are zero hidden layers, then the first and second topologies are equivalent, as are the third and the fourth. If there is one hidden layer, then the second and third topologies are equivalent. With two or more hidden layers, all four topologies are unique.

The configuration of all of the Jordan–Elman nets in the current research is selected to feed the context units with the input samples, providing an integrated past of the input. The context unit remembers the past of its inputs using a recency gradient, forgetting the past with an exponential decay, and controls the forgetting factor through the time constant that here is selected to be the IntegratorAxon function of 0.8 s time, with a longer memory depth and a slower forgetting factor. There were a standard of four neurons per hidden layer using as the transfer function TanhAxon; the learning rule was the Momentum function, with a value of 0.7 as the momentum and changing step size per hidden layer on a scale of 0.1.

The Genetic Algorithms in the Jordan–Elman Hybrids

The significance of each one of the sixteen financial inputs in all of the Jordan–Elman networks is calculated through the Genetic Algorithms, based on the hybrid models only. These models are trained multiple times to detect the input combination that produces the lowest error. The Genetic Algorithms are elaborated in four different hybrid models of different topologies:

- (i) on the input layer only;
- (ii) on the input and output layers only;
- (iii) into all the layers;
- (iv) into all the layers with cross-validation.

Batch learning was preferred to update the weights of hybrid neuro-genetic JE, after the presentation of the entire training set. The Genetic Algorithms also resolved the problem of optimal values in all the hidden layers and the output in (Figure 20):

- (a) the step size;
- (b) the momentum rate.

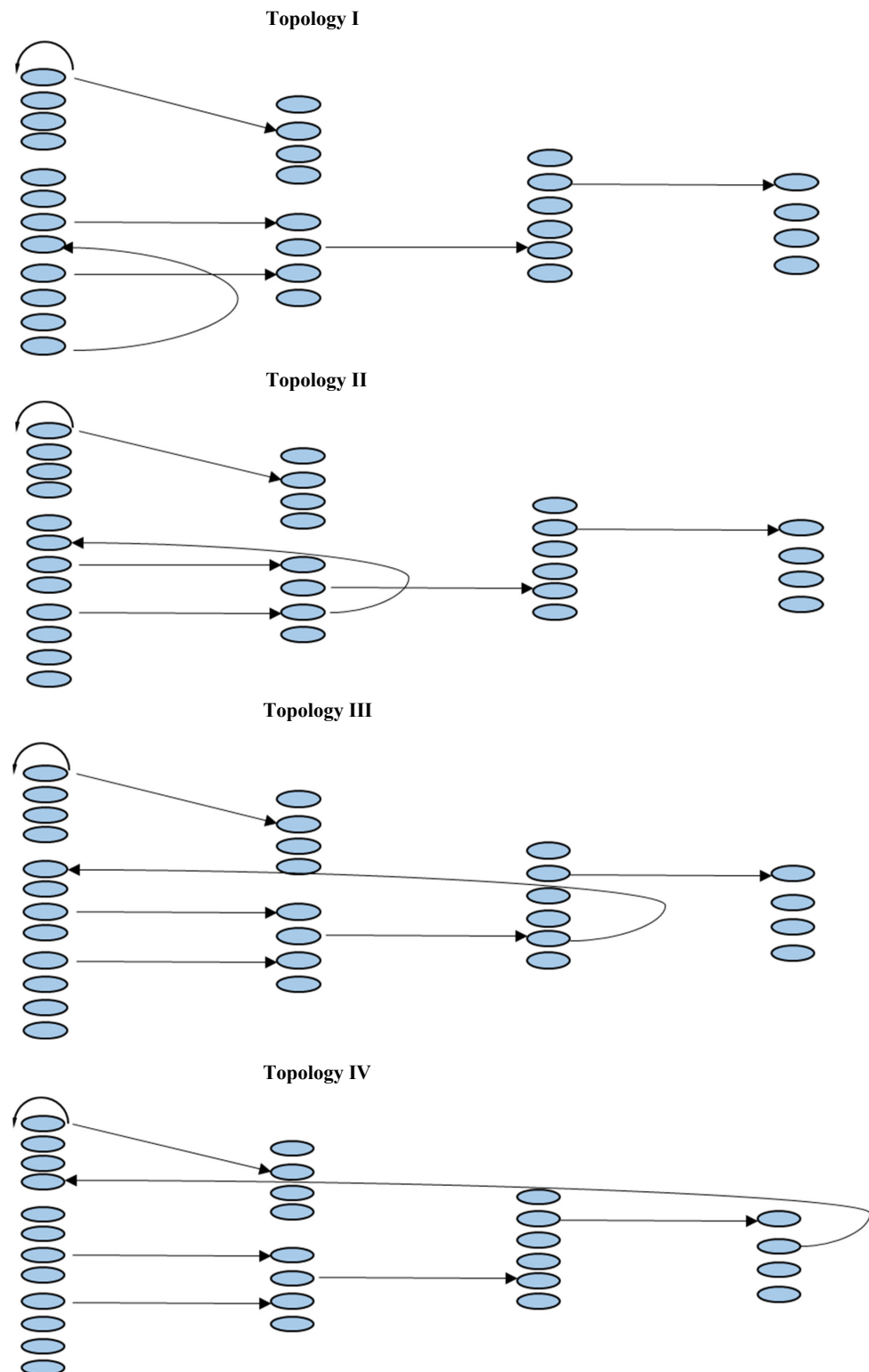


Figure 20. The processing in Jordan and Elman networks throughout the 4 different topologies.

JE nets require multiple training session to achieve the lowest error.

5.8. Multi-Layer Perceptrons

The Multi-Layer Perceptron (MLP) is a widely used neural network [25], where input signals are computed in a number of layers [26] that contain artificial neurons of the network. The number of PEs in the input layer is identical to the set of variables, and

at the output layer neurons, equals the desired number of classes, whilst the number of sets of nonlinear neurons [27] constitutes the hidden layers. Neurons except those of the input layer produce a linear combination of the outputs given by previous layers plus a bias. When the synaptic weights between different neurons are normalized with the output classes to 0–1, the MLP achieves the optimal performance of the maximum ex-post receiver in classifications [28]. The next neurons in the hidden layer process a non-linear sigmoid function of their input:

$$\varphi(x) = 1/(1 + \exp(-x)) \quad (35)$$

The output neurons produce a result equal to the linear combination. Multi-Layer Perceptrons with one hidden layer process a linear combination of the inputs' sigmoid functions. The linear sigmoid function approximates continuous functions of one or more variables, obtaining a continuous function fitting a finite set of points when no underlying model is available, and when the approximated function is trained with a desired answer 1 for signal and 0 for background, knowing the input values on probability, allowing classifications. MLPs can approximate arbitrary functions when they are trained with the backpropagation algorithm [29], whilst the LMS learning algorithm [30] cannot be extended to hidden neurons. Backpropagation begins with an initial value for each weight and proceeds until a stopping criterion is met, such as (i) to cap the number of iterations, (ii) to threshold the output mean square error, or (iii) to use cross-validation, which is the most powerful, since it stops the training at the point of best generalization.

Errors in the network are minimized through a backpropagation rule, permitting adaptation of the hidden neurons. Multi-Layer Perceptrons with nonlinear neurons have a smooth nonlinearity as the logistic function and the hyperbolic tangent, whilst their massive interconnectivity permits the computation of non-linear functions. The Multi-Layer Perceptron is trained with error correction learning, where the desired response for the system must be known [14,16]. In case of implementing point estimates, the problem of overfitting may appear to be supported by the flexibility of the MLP network. To alleviate overfitting, many of the signal transformation (ST) approaches need to restrict the structure of the MLP network. Training is implemented either by presenting a pattern and adapting the weights online, or by presenting all of the patterns in the input file (an epoch), accumulating the weight updates, and then updating the weights with the average weight update batch learning. Learning is controlled in any iterative training procedure, as the learning curve. Bias affects biological neurons in extreme weather conditions or in physiological disorders; thus, a bias input is given to each one of the artificial neurons, as shown in Figure 21:

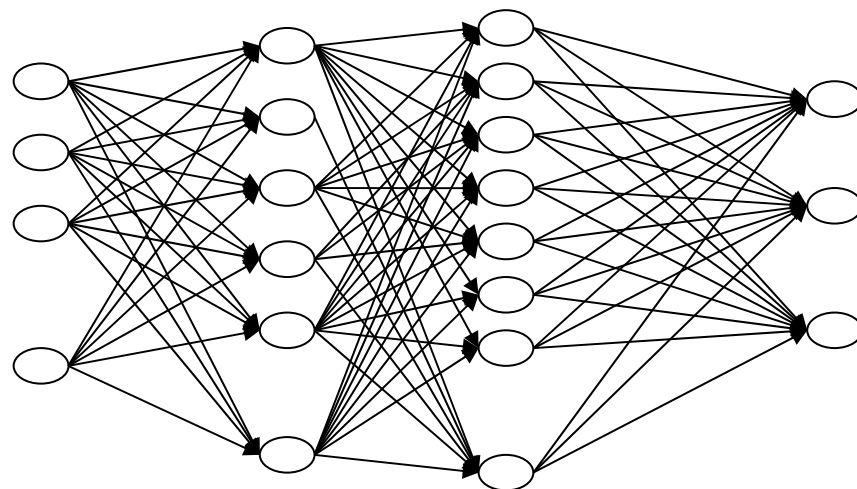


Figure 21. Multi-Layer Perceptron biased, with n hidden nonlinear layers.

6. Data of Neural Computation

Data were obtained from 1411 companies from the loan department of a Greek commercial bank, with the following 16 financial indices:

- (1) EBIT/Total Assets;
- (2) Net Income/Net Worth;
- (3) Sales/Total Assets;
- (4) Gross Profit/Total Assets;
- (5) Net Income/Working Capital;
- (6) Net Worth/Total Liabilities;
- (7) Total Liabilities/Total assets;
- (8) Long Term Liabilities / (Long Term Liabilities + Net Worth);
- (9) Quick Assets/Current Liabilities;
- (10) (Quick Assets-Inventories)/Current Liabilities;
- (11) Floating Assets/Current Liabilities;
- (12) Current Liabilities/Net Worth;
- (13) Cash Flow/Total Assets;
- (14) Total Liabilities/Working Capital;
- (15) Working Capital/Total Assets;
- (16) Inventories/Quick Assets.

and a 17th index of banker classification [31]. The test set was 50% overall, and the training set 50%. All of the stocks are unique, every index has the 3-year average [31], and the dependent value ε_τ is 0: healthy, 1: distressed. The observations have discrete frequency in three different values. The computational resources included an AMD Athlon II, with four cores, 2.61 GHz, Win XP SP2, 32 bits, 4GB Ram for all of the models except the Jordan–Elman model, tested on Intel i7-4935K, with 6/12 cores, 3.40 GHz, Win 8.1Pro, X64 bits, 64GB Ram.

7. Results

The classification process of the EPOS model is boosted by the appropriate model. In Table 1, the Active Confusion Matrix, which includes the classifications of the healthy companies (0), the distressed companies (1), and their ill-judged cases, e.g., healthy classified as distressed: $0 \rightarrow 1$, or distressed as healthy $1 \rightarrow 0$, gives the results of the neural and hybrid networks on the definition of the efficient set of stocks (0: healthy) and their statistics, to be elaborated at the next stage by the EPOS model. The recurrent models and their variations offered very competitive results that can claim the role of an efficient classifier of the EPOS. Specifically, the hybrid TLRN of GA optimization in all layers and cross-validation, with no hidden layers, was the best classifier of 99.57% and 96.6% successful classifications for the healthy and the distressed companies, respectively; the highest fitness of the model to the data in r was 0.991; there was a very low error, with MSE of 0.043, NMSE of 0.106, 5.61% error, and a very low partiality risk of Akaike Information Criterion (AIC), at -2093.61 , with quite similar results on the CV, and a processing time of 3 h 22 min 35 s.

The second rank was given to the hybrid TLRN of one layer and GA optimization in all layers, where the healthy and the distressed companies were classified correctly at 99.66% and 98.05%, respectively, the correlation coefficient r was 0.986, the error was the lowest, with MSE of 0.022, NMSE of 0.056, 3.78% error, and AIC pf -828.46 , requiring 3 h 34'11" to converge, exposed though to overfitting.

The third rank was taken by the Hybrid Recurrent net of one layer, GAs on the inputs only, with 98.99% and 96.6% correct classifications for the healthy and the distressed companies, very high fitness of the model to the data at 0.986, 0.038 MSE, 0.093 NMSE, 2.86% error, very high impartiality integrity on the Akaike Information Criterion (AIC) at -2148.5 , and a fast time of 54 min 40 s.

Table 1. Optimal recurrent models.

Model	Active Confusion Matrix						Performance				Time	
	Layers	0 → 0	0 → 1	1 → 0	1 → 1	MSE	NMSE	r	%Error	AIC	MDL	
Hybrid TLRN, GA all, CV	0	99.57	0.41	3.39	96.6	0.043	0.106	0.991	5.61795	−2093.61	−1985.16	3 h 22'35"
CV		99.32	0.66	6.87	93.11	0.055	0.132	0.994	5.95513	−325.95	−1826.39	
Hybrid TLRN, GA all	1	99.66	0.33	1.93	98.05	0.022	0.056	0.986	3.78903	−828.46	323.994	3 h 34'11"
Hybrid RNN Inp.GA	1	98.99	1	3.39	96.6	0.038	0.093	0.986	2.86	−2148.5	−2062.4	54'40"
Jordan– Elman input–output GA	1	99.83	0.16	3.20	96.78	0.022	0.052	0.983	3836	−2481.7	−2355.07	55'18"
Jordan– Elman input–output GA	0	99.91	0.08	3.66	96.32	0.031	0.075	0.978	4955.5	−2416.6	−2398.1	57'29"
Hybrid TLRN, GA all, CV	1	99.07	0.91	2.90	97.08	0.035	0.865	0.973	3.96943	−1122.04	−329.291	7 h 29'23"
CV		99.32	0.67	1.83	98.16	0.029	0.068	0.973	3.80316	−1284.93	−489.975	
Hybrid TLRN, GA all, CV	2	99.15	0.83	2.42	97.57	0.031	0.035	0.973	3.62095	−48.60	1495.71	11 h 30'21"
CV		99.24	0.75	0.91	99.08	0.025	0.059	0.968	3.35872	−178.70	2783.77	
Jordan– Elman NN	1	99.91	0.08	3.20	96.78	0.022	0.053	0.972	37.603	−2407.8	−2212.1	4"
Jordan– Elman GA all, CV	2	99.66	0.33	5.50	94.49	0.023	0.055	0.972	1572.26	−2439.5	−2287.3	2 h 35'29"
CV		99.83	0.16	0.91	99.08	0.023	0.056	0.971	28.511	−2425.7	−2273.5	
Hybrid TLRN, GA all, CV	3	99.24	0.75	1.93	98.06	0.031	0.075	0.973	3.39348	852.32	2958.24	10 h 34'35"
CV		99.15	0.83	1.83	98.16	0.032	0.074	0.971	3.50027	858.45	2970.24	
TLRN N. N.	2	98.90	1.08	3.39	96.6	0.043	0.055	0.971	5.00901	−1366.1	−818.945	17.5"
Jordan– Elman GA all	1	99.83	0.16	5.50	94.49	0.026	0.062	0.970	4127.5	−2378.5	−2263.3	1 h 38'53"
Jordan– Elman NN, CV	2	100	0	6.42	93.57	0.028	0.067	0.966	37.174	−2201.8	−1980.5	8"
CV		100	0	6.42	93.57	0.028	0.067	0.966	37.174	−2201.8	−1980.5	
Jordan– Elman GA inputs	1	100	0	8.25	91.74	0.027	0.065	0.966	40.46	−2352.8	−2226.1	20'01"
Jordan– Elman NN	2	99.91	0.08	4.12	95.86	0.035	0.084	0.960	45.335	−2006.4	−1785.1	5"
TLRN N. N.	1	99.66	0.33	4.85	95.14	0.045	0.112	0.958	5.86383	−1600.60	−1244.73	2 h 13'35"
Jordan– Elman GA inputs	2	99.83	0.16	7.33	92.66	0.039	0.092	0.956	47.15	−2006.0	−1824.9	54'24"
MLP NN, GA all, CV	1	98.56	1.92	21.55	78.43	0.132	0.312	0.917	42.3573	−1305.0	−1224.4	2 h 20'08"
Recurrent NN	1	99.15	0.83	25.51	74.48	0.117	0.287	0.874	8.83	−1429.1	−1295.1	9"

The fourth rank was given to the Hybrid Jordan–Elman model of one layer, GAs in inputs and outputs only, fine classification, 0.983 r, low error, low Akaike Information Criterion (AIC). The fifth rank was taken by the Hybrid Jordan–Elman of no layers, GAs in inputs and outputs only in excellent classification, 0.978 r, low error, low AIC. The sixth rank was awarded to the Hybrid TLRN of one layer, GAs in all layers and cross-validation, in fine classification, very high r, low error, low AIC. The seventh rank was taken by the Hybrid TLRN of two layers, GAs in all layers and cross-validation, in fine classification, very high r, low error, low AIC. The eighth rank was given to the Jordan–Elman Neural Net of one layer in excellent classification, very high r, low error, low AIC. The ninth rank was given to the Hybrid Jordan–Elman Neural Net of two layers, GAs in all layers and cross-validation, in excellent classification, very high r, low error, low AIC. The tenth rank

was taken by the Hybrid TLRN of three layers, GAs in all layers and cross-validation, in fine classification, very high r , low error, high AIC, indicating partiality. The eleventh rank was given to the TLRN of two layers, in fine classification, very high r , low error, low AIC. The twelfth rank was offered to the Hybrid Jordan–Elman of one layer, GAs in all layers, in fine classification, very high r , low error, low AIC. The thirteenth rank was taken by the Jordan–Elman model of two layers, GAs in all layers, in fine classification, very high r , low error, low AIC. The fourteenth rank was offered to the TLRN of one layer in a nice classification, very high r , low error, low AIC. The fifteenth rank was given to the Hybrid Jordan–Elman of two layers, GAs in input layers, in an excellent classification, very high r , low error, low AIC. The sixteenth rank was given to the Hybrid MLP of one layer, GAs in all layers, in a fine classification, very high r , medium error, low AIC. Finally, the last rank was given to the Recurrent Neural Network of one layer, with quite good classification, high r , medium error, low AIC.

8. Concluding Remarks

The EPOS model offers an integrated approach on the optimal portfolio selection problem. Its module-based architecture ensures a flexible environment that can support demanding environments. The consideration of the fundamentals as a significant parameter of the problem, alongside the introduction of a more complex isoelastic utility function, can provide more customer-tailored solutions with higher efficiency. Finally, in terms of the classifier within the EPOS model, the Hybrid TLRN of GA optimization in all layers and cross-validation, with no hidden layers, was the most optimal hybrid system that can support the decision-making process.

This approach can have multiple applications in the industry: (1) portfolio optimization, in more efficient filtering of stocks, (2) fraud detection, with a core analysis of the semi-strong efficient information that is publicly available, (3) optimization of classifiers in many industries. The principal problem of logic restricted here in the portfolio optimization problem is answered. The main philosophical question, ambiguous since the dawn of civilization, is answered here, showing that logic is dynamically flexible in a linear process, but adjusts by overriding on new challenging ideas with temporally short memory requirements that offer higher potential than the usual series of events. Thus, the time-consuming and complicated process is effectively simplified. It is non-linear, but is consistent with the maximization of utility and investors' economic welfare. Future work will numerically examine utilities and the wealth effect in the new trends of the bubble correlation.

Author Contributions: Conceptualization, N.L.; methodology, N.L.; software, N.L.; validation, N.L.; formal analysis N.L.; investigation, N.L.; resources, N.L.; data curation, N.L.; writing—original draft preparation, N.L.; visualization, N.L.; supervision, I.E., Y.B., G.G. and N.L.; project administration, N.L.; funding acquisition, N.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received external funding from the ELKE Fund of the Universities of Macedonia, Democritus University of Thrace and West Attica.

Data Availability Statement: Data are unavailable due to privacy restrictions.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Markowitz, H.M. Portfolio selection. *J. Financ.* **1952**, *7*, 77–91.
2. Cootner, P.H. *The Random Character of Stock Market Prices*; MIT Press: Cambridge, MA, USA, 1964.
3. Harrison, J.M.; Pliska, S.R. Martingales and Stochastic Integrals in the Theory of Continuous Trading. *Stoch. Process. Their Appl.* **1981**, *11*, 215–260. [[CrossRef](#)]
4. Bielecki, T.; Jeanblanc, M.; Rutkowski, M. *Credit Risk Modeling, Center for the Study of Finance and Insurance*; Osaka University Press: Osaka, Japan, 2009.
5. Schweizer, M.; Zivov, D.; Sikic, M. Dynamic Mean-Variance Optimisation Problems with Deterministic Information. *Int. J. Theor. Appl. Financ.* **2018**, *21*, 1850011. [[CrossRef](#)]

6. Rockafellar, R.T.; Uryasev, S. Optimization of Conditional Value-At-Risk. *J. Risk* **2000**, *2*, 21–42. [[CrossRef](#)]
7. Blăjină, O.; Ghionea, I.G. On Solving Stochastic Optimization Problems. *Mathematics* **2023**, *11*, 4451. [[CrossRef](#)]
8. Loukeris, N.; Eleftheriadis, I. The Portfolio Yield Reactive (PYR) model. *J. Risk Financ. Manag.* **2024**, *17*, 376. [[CrossRef](#)]
9. Subrahmanyam, A. Behavioral Finance: A Review and Synthesis. *Eur. Financ. Manag.* **2007**, *14*, 12–29. [[CrossRef](#)]
10. Loukeris, N.; Eleftheriadis, I.; Livanis, S. Optimal Asset Allocation in Radial Basis Functions Networks, and hybrid neuro-genetic RBFNs to TLRNs, MLPs and Bayesian Logistic Regression. In Proceedings of the World Finance Conference, Venice, Italy, 2–4 July 2014.
11. Loukeris, N.; Eleftheriadis, I.; Livanis, E. Portfolio Selection into Radial Basis Functions Networks and neuro-genetic RBFN Hybrids. In Proceedings of the IEEE 5th International Conference IISA, Chania, Greece, 7–9 July 2014.
12. Maringer, D.; Parpas, P. Global Optimization of Higher Order Moments in Portfolio Selection. *J. Glob. Optim.* **2009**, *43*, 2–3. [[CrossRef](#)]
13. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [[CrossRef](#)] [[PubMed](#)]
14. Principe, J.; de Vries, B.; Kuo, J.; Oliveira, P. Modeling applications with the focused gamma network. In *Advances in Neural Information Processing Systems—NIPS1991*, 4; Moody, J., Hanson, S., Lippmann, R.P., Eds.; Morgan Kaufmann: Burlington, MA, USA, 1992; pp. 121–126.
15. de Vries, B.; Principe, J.C. The gamma model: A new neural model for temporal processing. *Neural Netw.* **1992**, *5*, 565–576. [[CrossRef](#)]
16. Principe, J.; Euliano, N.; Lefebvre, W. *Neural and Adaptive Systems: Fundamentals through Simulations*; John Wiley & Sons, Inc.: New York, NY, USA, 2000.
17. Davis, L. *Handbook of Genetic Algorithms*; Van Nostrand Reinhold: New York, NY, USA, 1991.
18. Holland, J.H. *Adaptation in Natural and Artificial Systems*, 2nd ed.; MIT Press: Cambridge, MA, USA, 1992.
19. Orr, G. *Neural Networks*; Willamette University: Salem, OR, USA, 1999.
20. Stagge, P.; Senho, B. An extended elman net for modelling time series. In Proceedings of the International Conference on ANN, Lausanne, Switzerland, 8–10 October 1997.
21. Jordan, M.I. Attractor dynamics and parallelism in a connectionist sequential machine. In Proceedings of the Eighth Annual Conference of the Cognitive Science Society, Amherst, MA, USA, 15–17 August 1986; Erlbaum: Hillsdale, NJ, USA, 1986; pp. 531–546.
22. Jordan, M.I. *Serial Order: A Parallel Distributed Processing Approach*; Technical Report; Institute for Cognitive Science, University of California: Berkeley, CA, USA, 1986.
23. Elman, J. Finding structure in time. *Cogn. Sci.* **1990**, *14*, 179–211. [[CrossRef](#)]
24. Galvan, I.M.; Isasi, P. Multi-step learning rule for recurrent neural models: An application to time series forecasting. *Neural Process. Lett.* **2001**, *13*, 115–133. [[CrossRef](#)]
25. Lippman, R. An introduction to computing with neural nets. *IEEE Trans.* **1987**, *4*, 4–22. [[CrossRef](#)]
26. Hornik, K.; Stinchcombe, M.; White, H. Multilayer feedforward networks are universal approximators. *Neural Netw.* **1989**, *2*, 359–366. [[CrossRef](#)]
27. Lapedes, A.; Farber, R. Nonlinear signal processing using neural networks: Prediction, and system modelling. In Proceedings of the IEEE International Conference on Neural Networks, San Diego, CA, USA, 21 June 1987.
28. Makhoul, J. Pattern recognition properties of neural networks. In Proceedings of the 1991 IEEE Workshop on Neural Networks for Signal Processing, Princeton, NJ, USA, 30 September–1 October 1991; pp. 173–187.
29. Rumelhart, D.; Hinton, G.; Williams, R. Learning internal representations by error back-propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*; Rumelhart, D., McClelland, J., Eds.; MIT Press: Cambridge, MA, USA, 1986.
30. Widrow, B.; Lehr, M. 30 years of adaptive neural networks: Perceptron, madaline and backpropagation. *Proc. IEEE* **1990**, *78*, 1415–1442. [[CrossRef](#)]
31. Courtis, J. Modeling a Financial Ratios Categorical Framework. *J. Bus. Financ. Account.* **1978**, *5*, 371–386. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.