A Neural Evolutionary Approach to Financial Modeling

Antonia Azzini
Università degli Studi di Milano
Dipartimento di Tecnologie dell'Informazione
via Bramante, 65
26013, Crema, Italy
azzini@dti.unimi.it

Andrea G.B. Tettamanzi
Università degli Studi di Milano
Dipartimento di Tecnologie dell'Informazione
via Bramante, 65
26013, Crema, Italy
tettamanzi@dti.unimi.it

ABSTRACT

This paper presents an approach to the joint optimization of neural network structure and weights which can take advantage of backpropagation as a specialized decoder. The approach has been applied to a financial problem, whereby a factor model capturing the mutual relationships among several financial instruments is sought for. A sample application of such a model to statistical arbitrage is also presented.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*connectionism and neural nets*; I.6.5 [**Simulation and Modeling**]: Model Development—*modeling methodologies*

General Terms

Algorithms

Keywords

Evolutionary Algorithms, Neural Networks, Financial Modeling, Statistical Arbitrage

1. INTRODUCTION

The evolutionary approach that implements the conjunction of evolutionary algorithms (EAs) with neural networks (NNs) is a more integrated way of designing artificial neural networks (ANNs) since it allows all aspects of NN design to be taken into account at once and does not require expert knowledge of the problem. Much research has been undertaken on the combination of EAs and NNs. Through the use of EAs, the problem of designing a NN is regarded as an optimization problem.

Some EAs have implemented a search over the topology space, or a search for the optimal learning parameters. Some others concentrate just on weight optimization: these can be regarded as alternative training algorithms, and in this case the evolution of weights assumes that the architecture of the network must be static.

The primary motivation for using evolutionary techniques to establish the weighting values rather than traditional gradient descent techniques such as backpropagation (BP) [7], lies in the trapping in local minima and in the non-differentiability of the function. For this reason, rather than adapting weights based on local improvement only, EAs evolve weights based on the whole network fitness. Several works in this direction have been carried out by Montana and Davis [6] and by Whitley and colleagues [9]; in [10], they also implemented a purely evolutionary approach using binary codings of weights. In other cases an EA and a gradient descent algorithm have been combined [3].

The design of an optimal NN architecture can be formulated as a search problem in the architecture space, where each point represents an architecture. As pointed out by Yao [13, 14, 12], given some performance (optimality) criteria, e.g., minimun error, fastest learning, lower complexity, etc., about architectures, the performance level of all these forms a surface in the design space. Determining the optimal architecture design is equivalent to finding the highest point on this surface. There are several arguments which make the case for using EAs for searching for the best network topology [5, 8].

An interesting area of evolutionary NNs is the simultaneous evolution of different aspects of a NN. One of the most important is the combination of architecture and weight evolution. The advantage of combining these two basic elements of a NN is that a completely functioning network can be evolved without any intervention by an expert.

2. NEURO-GENETIC APPROACH

This new approach concerns the design of NNs based on EAs, and its aim is both to find an optimal network architecture and to train the network on a given data set. The approach is designed to be able to take advantage of the backpropagation (BP) algorithm if that is possible and beneficial; however, it can also do without it. The basic idea is to exploit the ability of the EA to find a solution close enough to the global optimum, together with the ability of the BP algorithm to finely tune a solution and reach the nearest local minimum.

A peculiar aspect is that BP is not used as some genetic

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

operator, as it is the case in some related work [1]. Instead, the EA optimizes both the topology and the weights of the networks; BP is optionally used to decode a *genotype* into a *phenotype* NN. Accordingly, it is the genotype which undergoes the genetic operators and which reproduces itself, whereas the phenotype is used *only* for calculating the genotype's fitness.

2.1 Algorithm Overview

In the neuro-genetic algorithm only a specific subset of NN architectures, named MLP, is considered for neural encoding. MLPs are feedforward NNs with a layer of input neurons, a layer of one or more output neurons and zero or more 'hidden' (i.e., internal) layers of neurons in between; neurons in a layer can take inputs from the previous layer only.

The overall evolutionary process can be described by the following pseudo-code:

1. Initialize the population, either by generating new random individuals or by loading a previously saved population.
2. Create for each genotype the corresponding MLP, and calculate its mean square error (mse), its cost and its fitness values.
3. Save the best individual as the best-so-far individual.
4. While not termination condition do
   (a) Apply the genetic operators to each network.
   (b) Decode each new genotype into the corresponding network.
   (c) Compute the fitness value for each network.
   (d) Save statistics.

The application of the genetic operators to each network is described by the following pseudo-code:

1. Select from the population (of size n) $\lfloor n/2 \rfloor$ individuals by truncation and create a new population of size n with copies of the selected individuals.
2. For all individuals in the population:
   (a) Perform crossover.
   (b) Mutate the topology and the weights of the offspring.
   (c) Train the resulting network using the training and validation sets if $bp = 1$.
   (d) Calculate f and $\hat{f}$ (see Section 2.3).
   (e) Save the individual with lowest $\hat{f}$ as the best-so-far individual if the $\hat{f}$ of the previously saved best-so-far individual is higher (worse).
3. Save statistics.

If a new population is to be generated, the corresponding networks will be initialized with different hidden layer sizes, using two exponential distributions to determine the number of hidden layers and neurons for each individual, and a normal distribution to determine the weights and bias values. Variance matrices will be also defined for all weights and bias matrices, that will be applied in conjunction with evolutionary strategies in order to perturbe network weights and bias. Variance matrices will be initialized with matrices of all ones.

In both cases, unlike other approaches like [14], the maximum size and number of the hidden layers is not determined in advance, nor bounded, even though the fitness function may penalize large networks.

Table 1: Individual Representation.

Element	Description
l	Lenght of the topology string, corrisponding to the number of layers.
topology	String of integer values that represent the number of neurons in each layer.
$\mathbf{W}^{(0)}$	Weights matrix of the input layer neurons of the network.
$\mathbf{Var}^{(0)}$	Variance matrix of the input layer neurons of the network.
$\mathbf{W}^{(i)}$	Weights matrix for the ith layer, $i = 1, \ldots, l$.
$\mathbf{Var}^{(i)}$	Variance matrix for the ith layer, $i = 1, \ldots, l$.
b_{ij}	Bias of the jth neuron in the ith layer.
$Var(b_{ij})$	Variance of the bias of the jth neuron in the ith layer.

2.2 Encoding

Each individual is encoded in a structure in which basic information are maintained as illustrated in Table 1. The values of all these parameters are affected by the genetic operators during evolution, in order to perform incremental (adding hidden neurons or hidden layers) and decremental (pruning hidden neurons or hidden layers) learning.

Note that the use of the *bp* parameter defines two different types of genetic encoding: if no BP-based network training is employed, we have a *direct encoding*, in which the network structure is directly translated into the corresponding phenotype; otherwise, we have an *indirect encoding* of networks, where the phenotype is obtained by the training of an initial (embryonic) network using BP.

2.3 Fitness

The fitness of an individual depends both on its accuracy (i.e., its mse) and on its cost. Although it is customary in EAs to assume that better individuals have higher fitness, we adopt the convention that a lower fitness means a better NN. This maps directly to the objective function of our problem, which is a cost minimization problem.Therefore, the fitness is proportional to the value of the mse and to the cost of the considered network. It is defined as

$$f = \lambda kc + (1 - \lambda)\text{mse}, \tag{1}$$

where $\lambda \in [0, 1]$ is a parameter which specifies the desired trade-off between network cost and accuracy, k is a constant for scaling the cost and the mse of the network to a comparable scale, and c is the overall cost of the considered network, defined as

$$c = \alpha N_{hn} + \beta N_{syn}, \tag{2}$$

where N_{hn} is the number of hidden neurons, and N_{syn} is the number of synapses.

The mse depends on the *Activation Function*, that calculates all the output values for each single layer of the neural network. In this work the *tangent sigmoid* transfer function

$$y = \frac{2}{1 + e^{-2x}} - 1 \tag{3}$$

is implemented. The rationale behind introducing a cost term in the objective function is that we seek for networks which use a reasonable amount of resources (neurons and synapses), which makes sense in particular when a hardware implementation is envisaged.

To be more precise, two fitness values are actually calculated for each individual: the fitness f, used by the selection operator, and a test fitness $\hat{f}$. Following the commonly accepted practice of machine learning, the problem data are partitioned into three sets:

- *training set*, used to train the network;
- *test set*, used to decide when to stop the training and avoid overfitting;
- *validation set*, used to test the generalization capabilities of a network.

It is important to stress that no thikness is given to these dataset definitions in the literature. Now, $\hat{f}$ is calculated according to Equation 1 by using the mse over the test set. When BP is used, i.e., if $bp = 1$, $f = \hat{f}$; otherwise ($bp = 0$), f is calculated according to Equation 1 by using the mse over the training and test sets together.

2.4 Selection

The selection method implemented in this work follows the breeder genetic algorithm [11], which differs from natural probabilistic selection in that only the best adapted individuals in the population are selected for reproduction. In particular, the selection strategy used by the algorithm is truncation: starting from a population of n individuals, the worst $\lfloor n/2 \rfloor$ (with respect to f) are eliminated. The remaining individuals are duplicated in order to replace those eliminated. Finally, the population is randomly permuted.

2.5 Mutation

Two types of *mutation* operators are used: a general random perturbation of weights, applied before the BP learning rule, and three *mutation* operators which affect the network architecture. The weight mutation is applied first, followed by the topology mutations, as follows:

1. Weight mutation: all the weight matrices $\mathbf{W}^{(i)}$, $i = 0, \ldots, l$ and the biases are perturbed by using variance matrices and evolutionary strategies applied to the number of synapses of the entire neural network N_{syn}. This mutation is implemented by the following equation:

$$\begin{aligned} W_j^{(i)} &\leftarrow W_j^{(i)} + N(0,1) \cdot \mathrm{Var}_j^{(i)} \\ \mathrm{Var}_j^{(i)} &\leftarrow \mathrm{Var}_j^{(i)} \cdot e^{\tau' N(0,1) + \tau N(0,1)} \end{aligned}$$

with

$$\begin{aligned} \tau' &= \frac{1}{\sqrt{2N_{syn}}} \\ \tau &= \frac{1}{\sqrt{2\sqrt{N_{syn}}}} \end{aligned}$$

After this perturbation has been applied, neurons whose contribution to the network output is negligible are eliminated: a variable threshold is defined, depending on a norm (in this case L_∞) of the weight vector for each node, and the relevant average and standard deviation of the norms of the considered layer. This task is carried out according to the following pseudo-code:

for $i = 1$ **to** $l - 1$ **do**
if $N_i > 1$
  for $j = 1$ **to** N_i **do**
    if $||W_j^{(i)}|| < (\mathrm{avg}_k(||W_k^{(i)}||) - r \cdot \mathrm{stdev}_k(||W_k^{(i)}||))$
      delete the jth neuron

where N_i is the number of neurons in the ith layer, $W_j^{(i)}$ is the jth column of matrix $\mathbf{W}^{(i)}$, and r is a parameter which allows the user to tune how many standard deviations below the layer average the contribution of a neuron must be before it is deleted.

2. Topology mutations: these operators affect the network structure (i.e., the number of neurons in each layer and the number of hidden layers). In particular, three mutations can occur:
   (a) Insertion of one hidden layer: with probability p^+_{layer}, a hidden layer i is randomly selected and a new hidden layer $i - 1$ with the same number of neurons is inserted before it, with $\mathbf{W}^{(i-1)} = \mathbf{I}(N_i)$ and $b_{i-1,j} = b_{ij}$, with $j = 1, \ldots, N_i = N_{i-1}$, where $\mathbf{I}(N_i)$ is the $N_i \times N_i$ identity matrix.
   (b) Deletion of one hidden layer: with probability p^-_{layer}, a hidden layer i is randomly selected; if the network has at least two layers and layer i has exactly one neuron, layer i is removed and the connections between the $(i-1)$th layer and the $(i+1)$th layer (to become the ith layer) are rewired as follows:
   $$\mathbf{W}'^{(i-1)} \leftarrow \mathbf{W}'^{(i-1)} \cdot \mathbf{W}'^{(i)}.$$
   Since $\mathbf{W}^{(i-1)}$ is a row vector and $\mathbf{W}^{(i)}$ is a column vector, the result of the product of their transposes is a $N_{i+1} \times N_{i-1}$ matrix.
   (c) Insertion of a neuron: with probability p^+_{neuron}, the jth neuron in the hidden layer i is randomly selected for duplication. A copy of it is inserted into the same layer i as the $(N_i + 1)$th neuron; the weight matrices are then updated as follows:
      i. a new row is appended to $\mathbf{W}^{(i-1)}$, which is a copy of its jth row;
      ii. a new column $W^{(i)}_{N_i+1}$ is appended to $\mathbf{W}^{(i)}$, where
      $$W_j^{(i)} \leftarrow \frac{1}{2} W_j^{(i)},$$
      $$W^{(i)}_{N_i+1} \leftarrow W_j^{(i)}.$$

The rationale for halving the output weights from both the jth neuron and its copy is that, by doing so, the overall network behavior remains unchanged, i.e., this kind of mutation is neutral.

All three topology mutation operators are designed so as to minimize their impact on the behavior of the network; in other words, they are designed to be as little disruptive (and as much neutral) as possible.

2.6 Recombination

As indicated in [4] there has been some debate in the literature about the opportunity of applying crossover to ANN evolution, based on disruptive effects that it could make into neural model. In this approach the idea is to implement a kind of vertical crossover, defining a *merge-operator* between the topologies and weights matrices of two parents in order to create the offsprings.

The new crossover operator is implemented as shown in Figure 1.

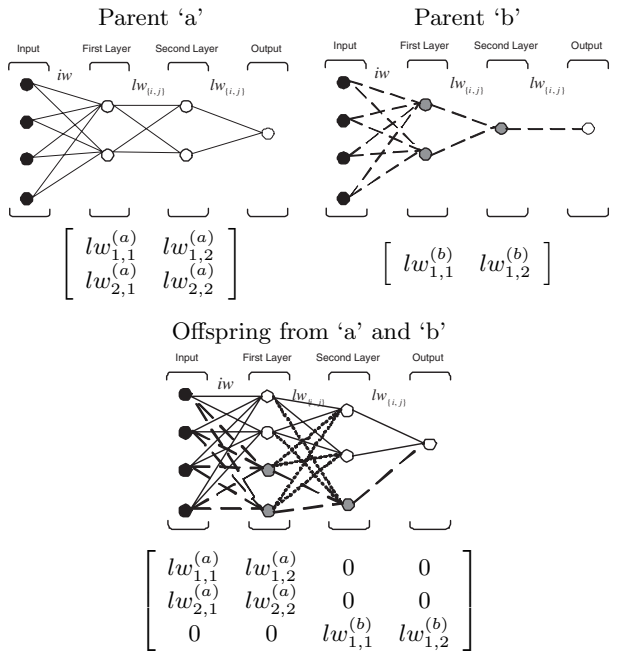

Figure 1: Merge-Crossover Representation.

Once the new population has been created by the *selection* operator described in Section 2.4, two individual are chosen for coupling and their neural structures are compared. If there are some differences in the topology length l, the hidden layer insertion mutation operator will be applied to the shortest neural topology in order to obtain individuals with the same number of layers.

Then it will be created a new individual, the offspring of the two parents selected. The neural structure of the new individual is created by adding the number of neurons in any hidden layer of each parent, excepted for input and output layer (they are the same for each neural network).

The new input-weights matrix $\mathbf{W}^{(0)}$ and the relative variance matrix $\mathbf{Var}^{(0)}$ are respectively obtained by appending the matrix of the second parent to the matrix of the first parent. Then, the new weight matrix $\mathbf{W}^{(i)}$ and the corresponding variance matrix $\mathbf{W}^{(i)}$ for each hidden layer of the offspring are respectively defined as the block diagonal matrix of the matrix of the first parent and the matrix of the second parent. Bias values and corresponding variance matrices of two parents are concatenated in order to obatin the new values for the new biases b_{ij} and variances $Var(b_{ij})$.

The weights of the inputs to the new output layer will be all set to the half of the corresponding weights in the parents. The rationale of this choice is that, if both parents were 'good' networks, they would both supply the appropriate input to the output layer; without halving it, the contribution from the two subnetworks would add and yield an approximately double input to the output layer. Therefore, halving the weights helps to make the operator as little disruptive as possible.

Table 2 lists all the parameters of the algorithm, and specifies the default values that they assume in this work.

Table 2: Parameters of the Algorithm.

Symbol	Meaning	Default Value
n	Population size	60
seed	Previously saved population	none
bp	Backpropagation selection	0/1
p^{+}_{layer}	Probability to insert a hidden layer	0.05
p^{-}_{layer}	Probability to delete a hidden layer	0.05
p^{+}_{neuron}	Probability to insert a neuron in a hidden layer	0.05
p_{cross}	Probability to crossover	0.2
r	Parameter for use in weight mutation for neuron elimination	1.5
h	Mean for the exponential distribution	3
N_{in}	Number of network inputs	*
N_{out}	Number of network outputs	*
α	Cost of single neuron	2
β	Cost of single synapsis	4
λ	Desired tradeoff between network cost and accuracy	0.5
k	Constant for scaling cost and mse in the same range	10^{-5}

*) depends on the problem.

3. AN APPLICATION TO FINANCIAL MODELING

3.1 Problem Description

The application of the neural evolutionary approach, implemented in this work, regards the building of factor models of financial instruments. Factor models are statistical models (in this case ANNs) that represent the returns of a financial instrument as a function of the returns of other financial instruments [2]. Factor models are used primarily for statistical arbitrage. A statistical arbitrageur builds a hedge portfolio consisting of one or more long positions and one or more short positions in various correlated instruments. When the price of one of the instruments diverges from the value predicted by the model, the arbitrageur puts on the arbitrage, by going long that instrument and short the others, if the price is lower than predicted, or short that

Table 3: Input Market Indices.

Class	Ticker	Description
Foreign Exhange Rates	EURGBP	1EUR = xGBP
	GBPEUR	1GBP = xEUR
	EURJPY	1EUR = xJPY
	JPYEUR	1JPY = xEUR
	GBPJPY	1GBP = xJPY
	JPYGBP	1JPY = xGBP
	USDEUR	1USD = xEUR
	EURUSD	1EUR = xUSD
	USDGBP	1USD = xGBP
	GBPUSD	1GBP = xUSD
	USDJPY	1USD = xJPY
	JPYUSD	1JPY = xUSD
Industry Representatives*	DNA	Biotecnologies
	TM	Motors
	DOW	Chemicals
	NOK	Communications
	JNJ	Drug Manufactures
	UN	Food
	BAB	Airlines
	XOM	Oil & Gas
	BHP	Metal & Mineral
	AIG	Insurance
	INTC	Semiconductors
	VZ	Telecom
	GE	Conglomerates
Commodities	OIL	Crude Oil $/barrel
	AU	Gold, $/Troy ounce
	AG	Silver, $/Troy ounce
US Treasury Bonds	TYX	30-year bond
	TNX	10-year note
	FVX	5-year note
	IRX	13-week bill

*) Representatives are, as a rule, the companies with largest market capitalization for their sector.

instrument and long the others, if the price is higher. If the model is correct, the price will tend to revert to the value predicted by the model, and the arbitrageur will profit.

To study the capabilities of our approach, we have tried it on a factor modeling problem whereby the Dow Jones Industrial Average (DJIA) is modeled against a number of other market indices, including foreign exchange rates, stock of individual companies taken as representatives of entire market segments, and commodity prices as shown in Table 3.

3.2 Experiments

In this application the training and test sets are created by considering daily closing prices for the period since the 2nd of January, 2001 until the 30th of November, 2005. All data are divided in two different datasets, rispectively with 1000 cases for training set and 250 cases for test set. The validation set consists of the daily closing prices for the period since the 1st of December, 2005 until the 13th of January, 2006.

All time series of three data sets are preprocessed by deleting the long term components. This step is carried out with the deletion of the 20 days moving average from that components.

All the parameters are set to the default values shown in Table 2. Several runs of this approach have been carried out in order to find out optimal settings of the genetic parameters p^+_{layer}, p^-_{layer}, and p^+_{neuron}. For each run of the evolutionary algorithm, up to 100,000 network evaluations (i.e., simulations of the network on the whole training set) have been allowed, including those performed by the backpropagation algorithm.

The results obtained are presented in Table 4: here are reported data about the average and the standard deviation of the test fitness values about the best solutions found for each parameter settings over 10 runs.

The first observation is that all these simulations consider backpropagation algorithm ($BP = 1$), while not all cases without backpropagation ($BP = 0$) have been considered. This is essentialy due to the fact that there is a striking superiority of the version of the algorithm which uses backpropagation. This is probably due to fact that whereas evolutionary algorithms are known to be quite effective in exploring the search space, they are in general quite poor at closing into a local optimum; backpropagation, which is essentially a local optimization algorithm, appears to complement well the evolutionary approach.

Another observation is that the approach is substantially robust with respect to the setting of parameters other than bp.

The best solutions, on average, have been found with $p^+_{\text{layer}} = 0.2$, $p^-_{\text{layer}} = 0.2$, and $p^+_{\text{neuron}} = 0.2$, although they do not differ significantly from other solutions found with $bp = 1$.

3.3 Results

The best model over all runs performed has been found by the algorithm using backpropagation. The best model is a multi-layer perceptron with a phenotype of type $[2, 1]$, which obtained a mean square error of 0.39 on the test set. Figure 2 shows a satisfactory agreement between the output of the best model with the actual closing values of the DJIA on the validation set.

The weights of the inputs to the neurons of the hidden layer are shown in Table 5. Those neurons have two biases of -1.8462 and -2.1212, and their output is connected with two respectively weights of -1.8637 and 1.0041 to the output neuron, whose bias is -2.0586.

In order to assess the quality of the model, we performed a comparison with simple linear regression on the same data. The linear regression yields a linear model

$$y = \sum_{i=1}^{32} w_i x_i, \tag{4}$$

where the w_i are those reported in Table 5 in the linear regression column. The prediction obtained by the linear regression model are compared with our best solution found, as shown in Figure 3. The neuro-genetic solution obtained with our approach has a mse of 1291.7, a better result compared to the mse of 1320.5 of the prediction based on linear regression on the same validation dataset.

Table 4: Financial Modeling Experimental Results.

Setting	Parameter Setting			BP=1	
	p^+_{layer}	p^-_{layer}	p^+_{neuron}	avg	stdev
1	0.05	0.05	0.05	0.2988	0.0464
2	0.05	0.05	0.1	0.2980	0.0362
3	0.05	0.05	0.2	0.3013	0.0330
4	0.05	0.1	0.05	0.2865	0.0368
5	0.05	0.1	0.1	0.2813	0.0435
6	0.05	0.1	0.2	0.3040	0.0232
7	0.05	0.2	0.05	0.2845	0.0321
8	0.05	0.2	0.1	0.2908	0.0252
9	0.05	0.2	0.2	0.3059	0.0208
10	0.1	0.05	0.05	0.2987	0.0290
11	0.1	0.05	0.1	0.3039	0.0341
12	0.1	0.05	0.2	0.3155	0.0396
13	0.1	0.1	0.05	0.3011	0.0395
14	0.1	0.1	0.1	0.2957	0.0201
15	0.1	0.1	0.2	0.3083	0.0354
16	0.1	0.2	0.05	0.2785	0.0325
17	0.1	0.2	0.1	0.2911	0.0340
18	0.1	0.2	0.2	0.2835	0.0219
19	0.2	0.05	0.05	0.2852	0.0292
20	0.2	0.05	0.1	0.2983	0.0309
21	0.2	0.05	0.2	0.2892	0.0374
22	0.2	0.1	0.05	0.3006	0.0322
23	0.2	0.1	0.1	0.2791	0.0261
24	0.2	0.1	0.2	0.2894	0.0260
25	0.2	0.2	0.05	0.2892	0.0230
26	0.2	0.2	0.1	0.2797	0.0360
27	0.2	0.2	0.2	0.2783	0.0369

Table 5: Input weights in the best neural network (second and third column), compared with the relevant coefficients of a linear model obtained by means of linear regression (fourth column).

Input	Weight		Linear
	1st Neuron	2nd Neuron	Regression
USDEUR	0.4953	0.4202	-16773
EURUSD	0.4214	0.3745	26979
USDGBP	0.1879	0.2300	139950
GBPUSD	-0.0095	0.1078	-56971
USDJPY	0.4642	0.4009	-630.66
JPYUSD	0.3441	0.3267	7131000
EURGBP	0.2991	0.2988	-239430
GBPEUR	0.2063	0.2414	112840
EURJPY	0.4246	0.3752	1120.3
JPYEUR	0.2048	0.2405	-17876000
GBPJPY	-0.1838	-0.0001	-438.21
JPYGBP	-0.1341	0.0307	12651000
AIG	-0.1914	-0.0048	15.963
BAB	0.2319	0.2572	14.326
BHP	0.2848	0.2900	12.095
C	0.3638	0.3388	22.405
DNA	0.2079	0.2424	3.5997
GE	0.4324	0.3813	45.966
INTC	0.0195	0.1258	23.786
JNJ	0.2774	0.2854	3.4063
NOK	0.3907	0.3555	-13.171
TM	0.8933	0.6665	3.1901
UN	0.3734	0.3448	10.95
VZ	0.4558	0.3958	25.732
XOM	0.2932	0.2952	21.554
OIL	0.3634	0.3386	-5.4066
AU	0.4418	0.3871	1.0003
AG	0.0969	0.1737	-31.656
TNX	0.3375	0.3225	316.7
TYX	0.2596	0.2744	-243.85
FVX	0.1404	0.2006	41.866
IRX	0.2127	0.2453	-93.432

3.4 An Application to Statistical Arbitrage

To evaluate the usefulness of such a model, a paper simulation of a very simple statistic arbitrage strategy has been carried out starting on December 1, 2005 until Friday, January 13, 2006.

The strategy is as follows: on each day, we consider the closing actual value of the DJIA index, as well as the closing values of the 32 instruments of Table 3; a model estimate y of the 'fair' value of the DJIA index is obtained by applying the neural network described above to those values. Three cases may occur:

- $\ln \frac{y}{\text{DJIA}} > \frac{\epsilon}{100}$: this means the DJIA is 'cheaper' than expected, and the strategy buys \$320,000 worth of it, while at the same time (short)-selling \$10,000 worth of each of the 32 instruments;
- $\frac{-\epsilon}{100} < \ln \frac{y}{\text{DJIA}} < \frac{\epsilon}{100}$: no action is taken;
- $\ln \frac{y}{\text{DJIA}} < \frac{-\epsilon}{100}$: this means the DJIA is overvalued, and the strategy (short)-sells \$320,000 worth of it, while at the same time buying \$10,000 worth of each of the 32 instruments.

Actually, buying (or selling) the same amount of the 12 exchange rates would have a zero net effect, because all pairs of transactions like 'buy USDEUR', 'buy EURUSD' would cancel. Therefore, no sensible trader would make those transactions. However, to simplify the exposition, we overlook such detail.

As a consequence of this strategy, a hedge portfolio is created and reallocated each day. For the sake of simplicity, we assume all transactions are carried out at the closing price and without cost. Of course a more realistic simulation should take transaction costs into account. On the other hand, most brokers pay a short interest rebate, usually close to the federal funds rate or the LIBOR, on the deposit they require to collateralize the short selling of securities, and this factor should be taken into account as well.

The net asset value (i.e., the total theoretical amount remaining after selling all long positions and covering all short positions at market value) of the hedge portfolio constructed by the above strategy during the validation market days of the simulation is shown in Table 6. The net asset value on Friday, January 13, 2006 can be taken as a (perhaps slightly optimistic) estimate of the profits of the arbitrage.

An inspection of Table 6 reveals that a starting capital of at least \$2,562,000 would have been required to implement the strategy, and probably more, under the assumption that our broker demanded a cash deposit (plus about 2% more) as collateral for the short positions taken by the strategy, based on the peak exposure recorded on January 13.

A profit of \$17,536 would represent almost a 0.67% return on that capital.

That is more than a 5.75% return on an annual basis, with a very moderate risk. This is not to suggest that the reader should take the model and put all of his or her savings on arbitrage in Wall Street; however, one can take such a result of an evidence that the model obtained actually contains some significant insight on the relationships between the financial instruments considered.

Table 6: Simulation of a statistical arbitrage strategy based on the best model. The action in the Action column refers to the (short)-selling or buying of the DJIA, counterbalanced by a buying or (short)selling of the other instruments, as explained in the text. The NAV column shows the net asset value of the hedge portfolio at the end of each day.

Date	Action	NAV
December 1st, 2005	Sell DJIA	\$0
December 2nd, 2005	Sell DJIA	\$1,373
December 5th, 2005	Buy DJIA	\$5,028
December 6th, 2005	Sell DJIA	\$3,806
December 7th, 2005	Sell DJIA	\$6,182
December 8th, 2005	Sell DJIA	\$10,598
December 9th, 2005	No Action	\$9,976
December 12th, 2005	Sell DJIA	\$16,910
December 13th, 2005	Buy DJIA	\$7,670
December 14th, 2005	Sell DJIA	\$(4,921)
December 15th, 2005	Sell DJIA	\$(7,339)
December 16th, 2005	Sell DJIA	\$(7,119)
December 19th, 2005	Buy DJIA	\$(5,446)
December 20th, 2005	Buy DJIA	\$1,393
December 21th, 2005	No Action	\$4,289
December 22th, 2005	Sell DJIA	\$(7,704)
December 23th, 2005	Sell DJIA	\$(6,613)
December 27th, 2005	Buy DJIA	\$3,526
December 28th, 2005	Buy DJIA	\$8,602
December 29th, 2005	No Action	\$12,256
December 30th, 2005	Sell DJIA	\$21,463
January 3rd, 2006	Buy DJIA	\$21,430
January 4th, 2006	Buy DJIA	\$22,490
January 5th, 2006	Buy DJIA	\$20,881
January 6th, 2006	No Action	\$17,637
January 9th, 2006	Sell DJIA	\$14,646
January 10th, 2006	Sell DJIA	\$13,535
January 11th, 2006	Sell DJIA	\$12,220
January 12th, 2006	Sell DJIA	\$17,160
January 13th, 2006	Sell DJIA	\$17,536

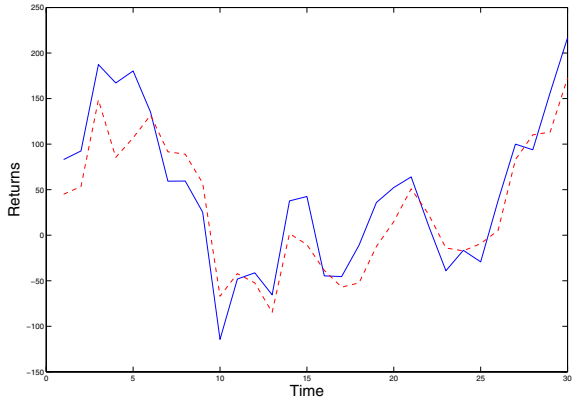

Figure 2: Comparison between the daily closing prices predicted by the best model (dashed line) and actual daily closing prices (solid line) of the DJIA on the validation set.

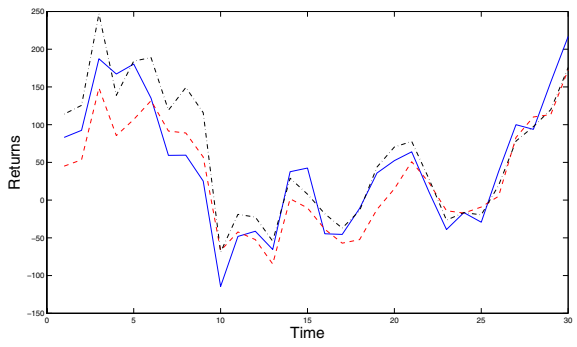

Figure 3: Comparison between the daily closing prices predicted by the best model (dashed line), those predicted by the linear regression (dash-dotted line), and the actual daily closing prices (solid line) of the DJIA on the validation set.

4. CONCLUSIONS

The work described in this paper demonstrates an approach to the joint optimization of neural network weights and structure which takes advantage of both evolutionary algorithms and the backpropagation algorithm.

Its effectiveness has been validated with an application to financial modeling. The results of a paper simulation of an arbitrage strategy which depends on the accuracy of the model show that the information given by a neural network obtained by the approach would enable an arbitrageur to gain significant profits.

5. REFERENCES

[1] P. A. Castillo, J. Carpio, J. J. Merelo, A. Prieto, V. Rivas, and G. Romero. Evolving multilayer perceptrons. *Neural Processing Letters*, 12(2):115–127, 2000.

[2] L. Harris. *Trading and exchanges: market microstructure for practitioners*. Oxford University Press, Oxford, 2003.

[3] R. Keesing and D. Stork. Evolution and learning in neural networks: the number and distribution of learning trials affect the rate of evolution. *Advanced in Neural Information Processing Systems*, 3:805–810, 1991.

[4] V. Maniezzo. Genetic evolution fo the topology and weight distribution of neural networks. *IEEE Transactions on Neural Networks*, 5(1):39–53, January 1994.

[5] G. Miller, P. Todd, and S. Hegde. Designing neural networks using genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 379–384. J.D. Schaffer, 1989.

[6] D. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the 11th International Conference on Artificial Intelligence*, pages 762–767. Morgan Kaufmann, 1989.

[7] D. E. Rumelhart, J. L. McClelland, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[8] K. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. Technical report, University of Texas at Austin, Department of Computer Science, Austin, Texas, 2001.

[9] D. Whitley and T. Hanson. Optimizing neural networks using faster, more accurate genetic search. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 391–396. J. D. Schaffer, 1989.

[10] D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel computing*, 14:347–361, 1993.

[11] H. Muhlenbein and D. Schlierkamp-Voosen. The science of breeding and its application to the breeder genetic algorithm (bga). *Evolutionary Computation*, 1(4):335–360, 1993.

[12] X. Yao. Evolving artificial neural networks. In *Proceedings on IEEE*, volume 87, pages 1423–1447, 1999.

[13] X. Yao and Y.Liu. A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8(3):694–713, May 1997.

[14] X. Yao and Y.Liu. Towards designing artificial neural networks by evolution. *Applied Mathematics and Computation*, 91(1):83–90, 1998.