Using a Financial Training Criterion Rather than a Prediction Criterion

Yoshua Bengio
bengioy@iro.umontreal.ca
Dept. IRO
Université de Montréal
Montréal, Qc, H3C 3J7
CANADA

To appear in the International Journal of Neural Systems, special issue on noisy time series

Abstract

The application of this work is to decision taking with financial time-series, using learning algorithms. The traditional approach is to train a model using a prediction criterion, such as minimizing the squared error between predictions and actual values of a dependent variable, or maximizing the likelihood of a conditional model of the dependent variable. We find here with noisy time-series that better results can be obtained when the model is directly trained in order to maximize the financial criterion of interest, here gains and losses (including those due to transactions) incurred during trading. Experiments were performed on portfolio selection with 35 Canadian stocks.

1 Introduction

Most applications of learning algorithms to financial time-series are based on predicting the value (either discrete or continuous) of output (dependent) variables given input (independent) variables. For example, the parameters of a multi-layer neural network are tuned in order to minimize a squared error loss. However, in many of these applications, the *ultimate goal* is not to make good predictions, but rather to use these often noisy predictions in order to *take some decisions*. In fact, the performance of these systems is usually measured in terms of financial profitability and risk criteria, after some heuristic decision taking rule has been applied to the trained model's outputs.

Because of the limited amount of training data, and because financial time-series are often very noisy, we argue here that better results can be obtained by choosing the model parameters

in order to *directly maximize the financial criterion of interest.* What we mean by *training criterion* in this paper is a scalar function of the training data and the model parameters. This scalar function is minimized (or maximized) with an optimization algorithm (such as gradient descent) by varying the parameters. In section 2, we present theoretical arguments justifying this direct optimization approach. In section 3, we present a particular cost function for optimizing the profits of a portfolio, *while taking into account losses due to transaction costs.* It should be noted that including transactions in the cost function makes it non-linear (and not quadratic) with respect to the trading decisions. When the decisions are taken in a way that depends on the current asset allocation (to minimize transactions), all the decisions during a trading sequence become dependent of each other. In section 4 we present a particular decision taking, i.e., trading, strategy, and a *differentiable* version of it, which can be used in the direct optimization of the model parameters with respect to the financial criteria. In section 5, we describe a series of experiments which compare the direct optimization approach with the prediction approach.

2 Optimizing the Correct Criterion

It has already been shown how artificial neural networks can be trained with various training criteria to perform a statistically meaningful task: for example, with the mean squared error criterion in order to estimate the expected value of output variables given input variables, or with cross-entropy or maximum likelihood, in order to build a model of the conditional distribution of discrete output variables, given input variables [Whi89, RL91].

However, in many applications of learning algorithms, the ultimate objective is not to build a model of the distribution or of the expected value of the output variables, but rather to use the trained system in order to take the best decisions, according to some criterion. The Bayesian approach is two-step: first, estimate a conditional model of the output distribution, given the input variables, second, *assuming this is the correct model*, take the optimal decisions, i.e, those which minimize a cost function.

For example, in *classification problems*, when the final objective is to minimize the number of classification errors, one picks the output class with the largest a-posteriori probability, given the input, and assuming the model is correct. However, this incorrect assumption may be hurtful, especially when the training data is not abundant (or non-stationary, for time-series), and noisy. In particular, it has been proven [HK92] for classification tasks that this strategy is less optimal than one based on training the model with respect to the decision surfaces, which may be determined by a *discriminant function* associated to each class (e.g., one output of a neural network for each class). The objective of training should be that the decision that is taken (e.g., picking the class whose corresponding discriminant function is the largest) has more chance of being the correct decision, without assuming a particular probabilistic interpretation for the discriminant functions (model outputs). Since the number of classification errors is a discrete function of the parameters, several training schemes have been proposed that are closer to that objective than a prediction or likelihood criterion: see for example the work on the Classification Figure of Merit [HW90], as well as the work on training neural networks through a post-processor based on dynamic programming for speech recognition [DBG91] (in which the

objective is to correctly recognize and segment sequences of phonemes, rather than individual phonemes).

The latter work is also related to several proposals to build modular systems that are trained cooperatively in order to optimize a common objective function (see [BG91] and [Ben96], Chapter 5). Consider the following situation. We have a composition of two models M_1, and M_2, with the output of M_1 feeding the input of M_2. Module M_1 computes $y(x, \theta_1)$, with input x and parameters θ_1. Module M_2 computes $w(y(x, \theta_1), \theta_2)$, with parameters θ_2. We have a prior idea of what M_1 should do, with pairs of input and desired outputs (x_p, d_p), but the ultimate measure of performance, $C(w)$, depends on the output w of M_2. In the context of this paper, as in Figure 1, M_1 represents a prediction model (for example of the future return of stocks), M_2 represents a trading module (which decides on portfolio weights w, i.e., when and how much to buy and sell), and C represents a financial criterion (such as the average return of the decision policy).

We compare two ways to train these two modules: either train them separately or train them jointly. When trained jointly, both θ_1 and θ_2 are chosen to minimize C, for example by back-propagating gradients through M_2 into M_1. When trained separately, M_1 is trained to minimize some intermediate training criterion, such as the Mean Squared Error (MSE) C_1 between the first module's output, $y(x_p, \theta_1)$, and the desired output d_p (here d_p could represent the actual future return of the stocks over some horizon for the p^{th} training example):

$$C_1(\theta_1) = \sum_p (d_p - y(x_p, \theta_1))^2 \tag{1}$$

Once M_1 is trained, the parameters of M_2 are then tuned (if it has any parameters) in order to minimize C. At the end of training, we can assume that local optima have been reached for C_1 (with respect to parameters θ_1) and C (with respect to parameters θ_2, assuming M_1 fixed), but that neither C_1 nor C have reached their best possible value:

$$\begin{aligned} \frac{\partial C_1}{\partial \theta_1} &= 0 \\ \frac{\partial C}{\partial \theta_2} &= 0 \end{aligned} \tag{2}$$

After this separate training, however, C could still be improved by changing y, i.e., $\frac{\partial C}{\partial y} \neq 0$, except in the trivially uninteresting case in which y does not influence w, or in the unlikely case in which the value of θ_1 which minimizes C_1 also minimizes C when θ_2 is chosen to minimize C (this is essentially the assumption made in the 2-step Bayes decision process).

Considering the influence of θ_1 on C over all the examples p, through y,

$$\frac{\partial C}{\partial \theta_1} = \sum_p \frac{\partial C}{\partial y(x_p, \theta_1)} \frac{\partial y(x_p, \theta_1)}{\partial \theta_1}, \tag{3}$$

so we have $\frac{\partial C}{\partial \theta_1} \neq 0$, except in the uninteresting case in which θ_1 does not influence y. Because of this inequality, one can *improve the global criterion* C by *further modifying* θ_1 along the direction of the gradient $\frac{\partial C}{\partial \theta_1}$. Hence separate training is generally suboptimal, because in general

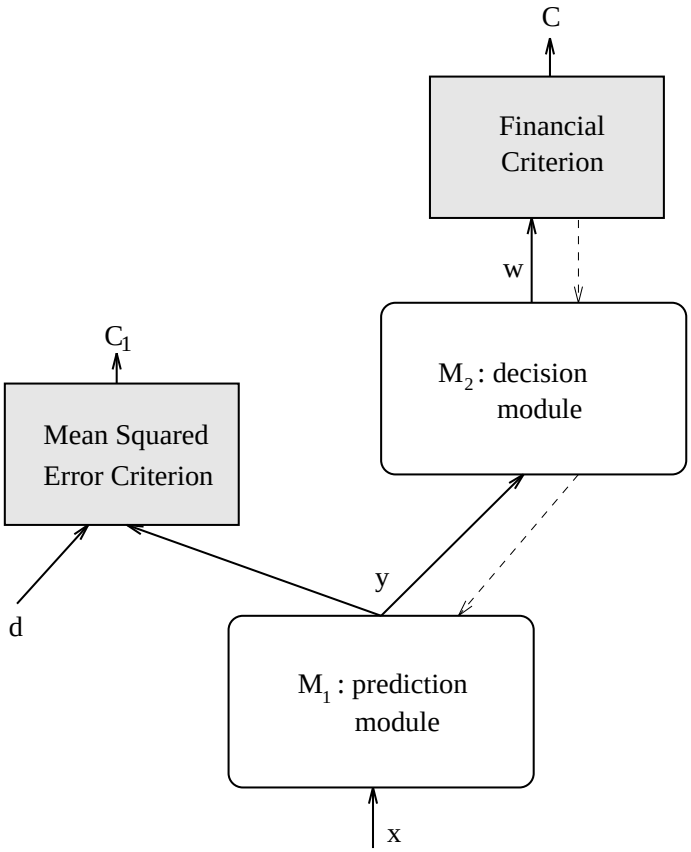


Figure 1: Task decomposition: a prediction module (M_1) with input x and output y, and a decision module (M_2) with output w. In the case of separate optimization, an intermediate criterion (e.g., mean squared error) is used to train M_1 (with desired outputs d). In the case of joint optimization of the decision module and the prediction module, gradients with respect to the financial criterion are back-propagated through both modules (dotted lines).

each module cannot perform *perfectly* the desired transformations from the preconceived task decomposition. For the same number of free parameters, joint training of M_1 and M_2 can reach a better value of C.

Therefore, if one wants to optimize on a given training set the global financial criterion C while having as few free parameters as possible in M_1, it is better to optimize M_1 with respect to C rather than with respect to an intermediate goal C_1.

3 A Training Criterion for Portfolio Management

In this paper, we consider the practical example of choosing a discrete sequence of portfolio weights in order to maximize profits, while taking into account losses due to transactions. We will simplify the representation of time by assuming a discrete series of events, at time indices $t = 1, 2, \ldots, T$. We assume that some decision strategy yields, at each time step t, the portfolio

weights $w_t = (w_{t,0}, w_{t,1}, \ldots, w_{t,n})$, for $n+1$ weights. In the experiments, we will apply this model to managing n stocks as well as a *cash* asset (which may earn short-term interest). We will assume that each transaction (buy or sell) of an amount v of asset i costs $c_i|v|$. This may be used to take into account the effect of differences in liquidity of the different assets. In the experiments, in the case of cash, the transaction cost is zero, whereas in the case of stocks, it is 1%, i.e., the overall cost of buying and later selling back a stock is about 2% of its value. A more realistic cost function should take into account the non-linear effects of the amount that is sold or bought: transaction fees may be higher for small transactions, transactions may only be allowed with a certain granularity, and slippage losses due to low relative liquidity may be higher for large transactions.

The training criterion is a function of the whole sequence of portfolio weights. At each time step t, we decompose the change in value of the assets in two categories: the return due to the changes in prices (and revenues from dividends), R_t, and the losses due to transactions, L_t. The overall return ratio is the product of R_t and L_t over all the time steps $t = 1, 2, \ldots, T$:

$$\text{overall return ratio} = \prod_t R_t L_t \tag{4}$$

This is the ratio of the final wealth to the initial wealth. Instead of maximizing this quantity, in this paper we maximize its logarithm (noting that the logarithm is a monotonic function):

$$C \stackrel{\text{def}}{=} \sum_t (\log R_t + \log L_t) \tag{5}$$

The yearly percent return is then given by $(e^{CP/T} - 1) \times 100\%$, where P is the number of time steps per year (12, in the experiments), and T is the number of time steps (number of months, in the experiments) over which the sum is taken. The return R_t due to price changes and dividends from time t to time $t+1$ is defined in terms of the portfolio weights $w_{t,i}$ and the multiplicative returns of each stock $r_{t,i}$,

$$r_{t,i} \stackrel{\text{def}}{=} \text{value}_{t+1,i} / \text{value}_{t,i}, \tag{6}$$

where $\text{value}_{t,i}$ represents the value of asset i at time t, assuming no transaction takes place: $r_{t,i}$ represents the relative change in value of asset i in the period t to $t+1$. Let $a_{t,i}$ be the actual worth of the i^{th} asset at time t in the portfolio, and let a_t be the combined value of all the assets at time t. Since the portfolio is weighted with weights $w_{t,i}$, we have

$$a_{t,i} \stackrel{\text{def}}{=} a_t w_{t,i} \tag{7}$$

and

$$a_t = \sum_i a_{t,i} = \sum_i a_t w_{t,i} \tag{8}$$

Because of the change in value of each one of the assets, their value becomes

$$a'_{t,i} \stackrel{\text{def}}{=} r_{t,i} a_{t,i}. \tag{9}$$

Therefore the total worth becomes

$$a'_t = \sum_i a'_{t,i} = \sum_i r_{t,i} a_{t,i} = a_t \sum_i r_{t,i} w_{t,i} \tag{10}$$

so the combined worth has increased by the ratio

$$R_t \stackrel{\text{def}}{=} \frac{a'_t}{a_t} \tag{11}$$

i.e.,

$$R_t = \sum_i w_{t,i} r_{t,i}. \tag{12}$$

After this change in asset value, the portfolio weights have changed as follows (since the different assets have different returns):

$$w'_{i,t} \stackrel{\text{def}}{=} \frac{a'_{t,i}}{a'_t} = \frac{w_{t,i} r_{t,i}}{R_t}. \tag{13}$$

At time $t+1$, we want to change the proportions of the assets to the new portfolio weights w_{t+1}, i.e, the worth of asset i will go from $a'_t w'_{t,i}$ to $a'_t w_{t+1,i}$. We then have to incur for each asset a transaction loss, which is assumed simply proportional to the amount of the transaction, with a proportional cost c_i for asset i. These losses include both transaction fees and slippage. This criterion could easily be generalized to take into account the fact that the slippage costs may vary with time (depending on the volume of offer and demand) and may also depend non-linearly on the actual amount of the transactions. After transaction losses, the worth at time $t+1$ becomes

$$\begin{aligned} a_{t+1} &= a'_t - \sum_i c_i |a'_t w_{t+1,i} - a'_t w'_{t,i}| \\ &= a'_t (1 - \sum_i c_i |w_{t+1,i} - w'_{t,i}|). \end{aligned} \tag{14}$$

The loss L_t due to transactions at time t is defined as the ratio

$$L_t \stackrel{\text{def}}{=} \frac{a_t}{a'_{t-1}}. \tag{15}$$

Therefore

$$L_t = 1 - \sum_i c_i |w_{t,i} - w'_{t-1,i}|. \tag{16}$$

To summarize, the overall profit criterion can be written as follows, in function of the portfolio weights sequence:

$$\begin{aligned} C = \sum_t & \log(\sum_i r_{t,i} w_{t,i}) + \\ & \log(1 - \sum_i c_i |w_{t,i} - w'_{t-1,i}|) \end{aligned} \tag{17}$$

where w' is defined as in equation 13. Therefore we can write C in terms of the return ratios $r_{t,i}$, the decisions $w_{t,i}$, and the relative transactions costs c_i as follows:

$$\begin{aligned} C = \sum_t & \log(\sum_i r_{t,i} w_{t,i}) + \\ & \log(1 - \sum_i c_i |w_{t,i} - \frac{w_{t-1,i} r_{t-1,i}}{\sum_i w_{t-1,i} r_{t-1,i}}|) \end{aligned} \tag{18}$$

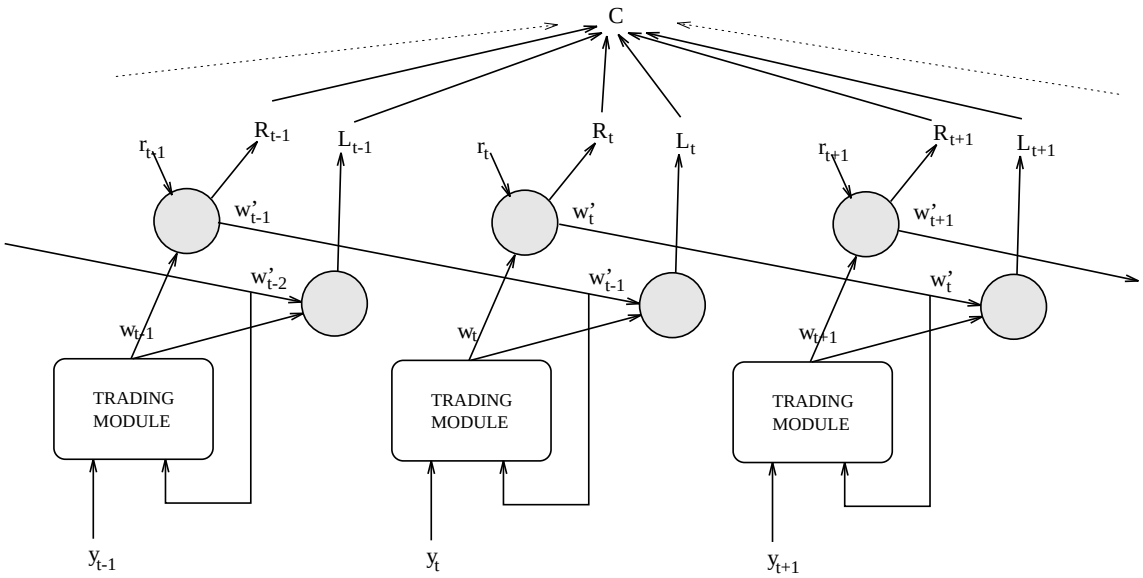


Figure 2: Operation of a trading module, unfolded in time, with inputs y_t (network output) and w'_{t-1} (previous portfolio weights after change in value), and with outputs w_t (next portfolio weights). R_t is the return of the portfolio due to changes in value, L_t is the loss due to transactions, and $r_{t,i}$ is the individual return of asset i.

At each time step, a trading module computes w_t, from w'_{t-1} and from the predictor output y_t, as illustrated (unfolded in time) in Figure 2. To backpropagate gradients with respect to the cost function through the trader from the above equation, one computes $\frac{\partial C}{\partial w_{t,i}}$, when given $\frac{\partial C}{\partial w'_t}$. The trading module can then compute $\frac{\partial C}{\partial w'_{t-1}}$ from $\frac{\partial C}{\partial w_t}$, and this process is iterated backwards in time. At each time step, the trading module also computes $\frac{\partial C}{\partial y_t}$ from $\frac{\partial C}{\partial w_t}$.

To conclude this section, it should be noted that the introduction of transaction losses in the training criterion makes it non-linear in the decisions (whereas the profit term is linear in the decisions). Note that it is not even differentiable everywhere (but it is differentiable almost everywhere, which is enough for gradient descent optimization). Furthermore, when the decision at time t is taken in function of the previous decision (to avoid unnecessary transactions), all the decisions are coupled together, i.e., the cost function can't be separated as a sum of independent terms associated to the network output at each time step. For this reason, an algorithm such as back-propagation through time has to be used to compute the gradients of the cost function.

4 The Trading Modules

We could directly train a module producing in output the portfolio weights $w_{t,i}$, but in this paper we use some financial a-priori knowledge in order to modularize this task in two subtasks:

1. with a "prediction" module (e.g., M_1 in figure 1), compute a "desirability" value $y_{t,i}$ for each asset on the basis of the current inputs,

2. with a *trading module*, allocate capital among the given set of assets (i.e., compute the weights $w_{t,i}$), on the basis of y_t and $w'_{t-1,i}$ (this is done with the decision module M_2 in figure 1).

In this section, we will describe two such trading modules, both based on the same a-priori knowledge. The first one is not differentiable and it has hand-tuned parameters, whereas the second one is differentiable and it has parameters learned by gradient ascent on the financial criterion C. The a-priori knowledge we have used in designing this trader can be summarized as follows:

- We mostly want to have in our portfolio those assets that are desirable according to the predictor (high $y_{t,i}$).
- More risky assets (e.g., stocks) should have a higher expected return than less risky assets (e.g., cash) to be worth keeping in the portfolio.
- The outputs of the predictor are very noisy and unreliable.
- We want our portfolio to be as diversified as possible, i.e., it is better to have two assets of similar expected returns than to invest all our capital in one that is slightly better.
- We want to minimize the amount of the transactions.

At each time step, the trading module takes as input the vectors y_t (predictor output) and w'_{t-1} (previous weights, after change in value due to multiplicative returns r_{t-1}). It then produces the portfolio weight vector w_t, as shown in Figure 2. Here we are assuming that the assets $0 \ldots n-1$ are stocks, and asset n represents cash (earning short-term interests). The portfolio weights $w_{t,i}$ are non-negative and sum to 1.

4.1 A Hard Decisions Trader

Our first experiments were done with a neural network trained to minimize the squared error between the predicted and actual asset returns. Based on advice from financial specialists, we designed the following trading algorithm, which takes hard decisions, according to the a-priori principles above. The algorithm described in figure 3 is executed at each time step t.

Statement 1 in figure 3 is to minimize transactions. Statement 2 assigns a discrete quality (`good`, `neutral`, or `bad`) to each stock in function of how the predicted return compares to the average predicted return and to the return of cash. Statement 3 computes the current total weight of `bad` stocks that are currently owned, and should therefore be sold. Statement 4 uses that money to buy the `good` stocks (if any), distributing the available money uniformly among the stocks (or if no stock is `good` increase the proportion of cash in the portfolio).

The parameters c_0, c_1, c_2, b_0, b_1, and b_2 are thresholds that determine whether a stock should be considered `good`, `neutral`, or `bad`. They should depend on the scale of y and on the relative risk of stocks versus cash. The parameter $0 < \tau < 1$ controls the "boldness" of the trader. A

1. By default, initialize $w_{t,i} \leftarrow w'_{t,i}$ for all $i = 0 \ldots n$.

2. Assign a $\text{quality}_{t,i}$ (equal to `good`, `neutral`, or `bad`) to each stock ($i = 0 \ldots n-1$):

   (a) Compute the average desirability $\bar{y}_t \leftarrow \frac{1}{n} \sum_{i=0}^{n-1} y_{t,i}$.

   (b) Let $\text{rank}_{t,i}$ be the rank of $y_{t,i}$ in the set $\{y_{t,0}, \ldots, y_{t,n-1}\}$.

   (c) **If** $y_{t,i} > c_0 \bar{y}_t$ and $y_{t,i} > c_1 y_{t,n}$ and $\text{rank}_{t,i} > c_2$
   Then
   $\text{quality}_{t,i} \leftarrow$ `good`,
   Else,
   If $y_{t,i} < b_0 \bar{y}_t$ or $y_{t,i} < b_1 y_{t,n}$ or $\text{rank}_{t,i} < b_2$
   Then, $\text{quality}_{t,i} \leftarrow$ `bad`,
   Else, $\text{quality}_{t,i} \leftarrow$ `neutral`.

3. Compute the total weight of `bad` stocks that should be sold:

   (a) Initialize $k_t \leftarrow 0$

   (b) **For** $i = 0 \ldots n-1$

   - **If** $\text{quality}_{t,i} =$ `bad` and $w'_{t-1,i} > 0$ (i.e., already owned), **Then**
     (SELL a fraction of the amount owned)
     $k_t \leftarrow k_t + \tau w'_{t-1,i}$
     $w_{t,i} \leftarrow w'_{t-1,i} - \tau w'_{t-1,i}$

4. **If** $k_t > 0$ **Then** (either distribute that money among `good` stocks, or keep it in cash):

   (a) Let $s_t \leftarrow$ number of `good` stocks *not owned.*

   (b) **If** $s_t > 0$
   Then
   - (also use some cash to buy `good` stocks)
     $k_t \leftarrow k_t + \tau w'_{t-1,n}$
     $w_{t,n} \leftarrow w'_{t-1,n}(1-\tau)$
   - **For** all `good` stocks not owned, BUY: $w_{t,i} \leftarrow k_t / s_t$.

   Else (i.e., no `good` stocks were not already owned)
   - Let $s'_t \leftarrow$ number of `good` stocks,
   - **If** $s'_t > 0$
     Then For all the `good` stocks, BUY: $w_{t,i} \leftarrow w'_{t-1,i} + k_t / s'_t$
     Else (put the money in cash) $w_{t,n} \leftarrow w'_{t-1,n} + k_t$.

Figure 3: Algorithm for the "hard" trading module. See text for more explanations.

small value prevents it from making too many transactions (a value of zero yields a buy-and-hold policy).

In the experiments, those parameters were chosen using basic judgment and a few trial and error experiments on the first training period. However, it is difficult to numerically optimize these parameters because of the discrete nature of the decisions taken. Furthermore, the predictor module might not give out numbers that are optimal for the trader module. This has motivated the following differentiable trading module.

4.2 A Soft Decisions Trader

This trading module has the same inputs and outputs as the hard decision trader, as in Figure 2, and executes algorithm described in 4 at each time step t.

Statement 1 of figure 4 defines two quantities ("goodness" and "badness"), to compare each asset with the other assets, indicating respectively a willingness to buy and a willingness to sell. "Goodness" compares the network output for a stock with the largest of the average network output over stocks and the promised cash return. "Badness" compares the network output for a stock with the smallest of the average network output over stocks and the promised cash return. Statement 2 computes the amount to sell based on the weighted sum of "badness" indices. Statement 3a then computes a quantity δ_t that compares the sum of the goodness and badness indices. Statement 3b uses that quantity to compute the change in cash (using a different formula depending on wether δ_t is positive or negative). Statement 3c uses that change in cash to compute the amount available for buying more stocks (or the amount of stocks that should be sold). Statement 4 computes the new proportions for each stock, by allocating the amount available to buy new stocks according to the relative goodness of each stock. In the first term of statement 4a the proportions are reduced proportionaly to the badness index, and in the second term they are increased proportionally to the goodness index. Again, a parameter τ controls the risks taken by the trading module (here when τ is very negative, the buy-and-hold strategy will result, whereas when it is large, more transactions will occur). Note that $\text{sigmoid}(x) = \frac{1}{1+\exp -x}$. The $\text{sigmoid}(\tau)$ rather than τ was used to constrain that number to be between 0 and 1. There are 9 parameters, $\theta_2 = \{ c_0, c_1, b_0, b_1, a_0, a_1, s_0, s_1, \tau \}$, five of which have a similar interpretation as in the hard trader. However, since we can compute the gradient of the training criterion with respect to these parameters, their value can be learned from the data. From the above algorithmic definition of the function $w_t(w'_{t-1}, y_t, \theta_2)$ one can easily write down the equations for $\frac{\partial C}{\partial y_{t,i}}$, $\frac{\partial C}{\partial w'_{t-1,i}}$ and $\frac{\partial C}{\partial \theta_2}$, when given the gradients $\frac{\partial C}{\partial w_{t,j}}$, using the chain rule.

5 Experiments

We have performed experiments in order to study the difference between training only a prediction module with the Mean Squared Error (MSE) and training both the prediction and decision modules to maximize the financial criterion defined in section 3 (equation 18).

1. (Assign a goodness value $g_{t,i}$ and a badness value $b_{t,i}$ between 0 and 1 for each stock)
   - (Compute the average desirability) $\bar{y}_t \leftarrow \frac{1}{n}\sum_{i=0}^{n-1} y_{t,i}$.
   - (goodness) $g_{t,i} \leftarrow \text{sigmoid}(s_0(y_{t,i} - \max(c_0\bar{y}_t, c_1 y_{t,n})))$
   - (badness) $b_{t,i} \leftarrow \text{sigmoid}(s_1(\min(b_0\bar{y}_t, b_1 y_{t,n}) - y_{t,i}))$
2. (Compute the amount to "sell", to be offset later by an amount to "buy")
   $k_t \leftarrow \sum_{i=0}^{n-1} \text{sigmoid}(\tau) b_{t,i} w'_{t-1}, i$
3. (Compute the change in cash)
   (a) $\delta_t \leftarrow \tanh(a_0 + a_1 \sum_{i=0}^{n-1}(b_{t,i} - g_{t,i}))$
   (b) **If** $\delta_t > 0$ (more bad than good, increase cash)
   Then $w_{t,n} \leftarrow w'_{t-1,n} + \delta_t k_t$
   Else (more good than bad, reduce cash)
   $w_{t,n} \leftarrow -w'_{t-1,n}\delta_t$
   (c) So the amount available to buy is:
   $a_t \leftarrow k_t - (w_{t,n} - w'_{t-1,n})$
4. (Compute amount to "buy", offset by previous "sell", and compute the new weights $w_{t,i}$ on the stocks)
   (a) $s_t \leftarrow \sum_{i=0}^{n-1} g_{t,i}$ (a normalization factor)
   (b) $w_{t,i} \leftarrow w'_{t-1,i}(1 - \text{sigmoide}(\tau) b_{t,i}) + \frac{g_{t,i}}{s_t} a_t$

Figure 4: Algorithm for the "soft" (differentiable) trading module. See text for more explanations.

5.1 Experimental Setup

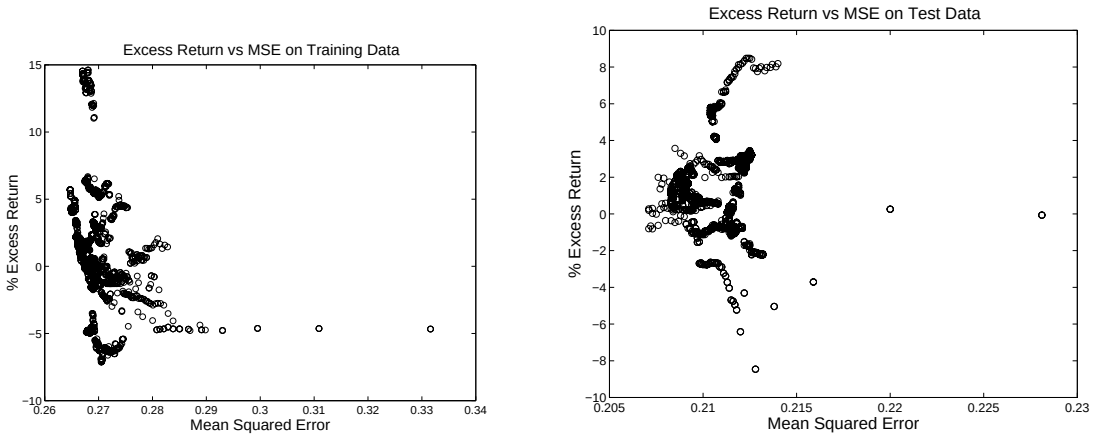
The task is one of managing a portfolio of 35 Canadian stocks, as well as allocate funds between those stocks and a cash asset ($n = 35$ in the above sections, the number of assets is $n + 1 = 36$). The 35 companies are major companies of the Toronto Stock Exchange (most of them in the TSE35 Index). The data is monthly and spans 10 years, from December 1984 to February 1995 (123 months). We have selected 5 input features (x_t is 5-dimensional), 2 of which represent macro-economic variables which are known to influence the business cycle, and 3 of which are micro-economic variables representing the profitability of the company and previous price changes of the stock.

We used ordinary fully connected multi-layered neural networks with a single hidden layer, trained by gradient descent. The same network was used for all 35 stocks, with a single output $y_{t,i}$ at each month t for stock i. Preliminary experiments with the network architecture suggested that using approximately 3 hidden units yielded better results than using no hidden layer or many more hidden units. Better results might be obtained by considering different sectors of the market (different types of companies) separately, but for the experiments reported here, we used a single neural network for all the stocks. When using a different model for each stock and sharing some of the parameters, significantly better results were obtained (using the same training strategy) on that data [GB97]. The parameters of the network are therefore shared across time and across the 35 stocks. The 36th output (for desirability of cash) was obtained from the current short-term interest rates (which are also used for the multiplicative return of cash, $r_{t,n}$).

To take into account the non-stationarity of the financial and economic time-series, and estimate performance over a variety of economic situations, multiple training experiments were performed on different training windows, each time testing on the following 18 months. For each experiment, the data is divided into three sets: one for training, one for validation (early stopping), and one for testing (estimating generalization performance). The latter two sets each span 18 months. Four training, validation, and test periods were considered, by increments of 18 months:

1. Training from first 33 months, validation with next 18 months, test with following 18 months.
2. Training from first 51 months, validation with next 18 months, test with following 18 months.
3. Training from first 69 months, validation with next 18 months, test with following 18 months.
4. Training from first 87 months, validation with next 18 months, test with following 18 months.

Training lasted between 10 and 200 iterations of the training set, with early stopping based on the performance on the validation set. The overall return was computed for the whole test period (of 4 consecutive sets of 18 months = 72 months = 6 years: March 89 - February 95).



(a) (b)

Figure 5: Scatter plots of MSE versus excess return of network, trained to minimize the MSE, (a) on training set, (b) on test set.

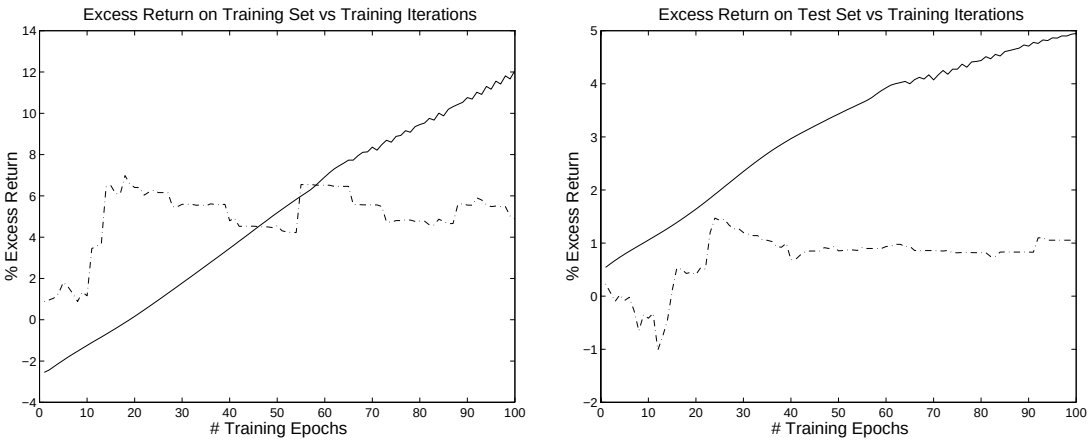
When comparing the two training algorithms (prediction criterion versus financial criterion), 10 experiments were performed with different initial weights, and the average and standard deviation of the financial criterion are reported.

A buy-and-hold *benchmark* was used to compare the results with a conservative policy. For this benchmark, the initial portfolio is distributed equally among all the stocks (and no cash). Then there are no transactions. The returns for the benchmark are computed in the same way as for the neural network (except that there are no transactions). The *excess return* is defined as the difference between the overall return obtained by a network and that of the buy-and-hold benchmark.

5.2 Results

In the first series of experiments, the neural network was trained with a mean squared error criterion in order to predict the return of each stock over a horizon of three months. We used the "hard decision trader" described in section 4.1 in order to measure the financial profitability of the system. We quickly realized that although the mean squared error was gradually improving during training, the profits made sometimes increased, sometimes decreased. This actually suggested that we were not optimizing the "right" criterion.

This problem can be visualized in Figures 5 and 6. The scatter plots were obtained by taking the values of excess return and mean squared error over 10 experiments with 200 training epochs (i.e, with 2000 points), both on a training and a test set. Although there is a tendency for returns to be larger for smaller MSE, many different values of return can be obtained for the same MSE. This constitutes an additional (and undesirable) source of *variance* in the generalization performance. Instead, when training the neural network with the financial criterion, the corresponding scatter plots of excess return against training criterion would put all the points on a single exponential curve, since the excess return is simply the value of the training criterion



(a) (b)

Figure 6: Evolution of excess return during training for network trained directly to maximize return (full line) and network trained to minimize MSE (dashed line), (a) on training set, (b) on test set.

normalized to obtain yearly returns (by dividing the log-returns by the number of years in the sequence, and taking the exponential), and from which the average return of the benchmark is substracted.

For the second series of experiments, we created the "soft" version of the trader described in section 4.2, and trained the parameters of the trader as well as the neural network in order to maximize the financial criterion defined in section 3 (which is equivalent to maximizing the overall excess return). A series of 10 training experiments (with different initial parameters) were performed (each with four training, validation and test periods) to compare the two approaches. Table 1 summarizes the results. During the whole 6-year test period (March 89 - February 95), the benchmark yielded returns of 6.8%, whereas the network trained with the prediction criterion and the one trained with the financial criterion yielded in average returns of 9.7% and 14.2% respectively (i.e, 2.9% and 7.4% in excess of the benchmark, respectively). The direct optimization approach, which uses a specialized criterion specialized for the financial task, clearly yields better performance on this task, both on the training and test data.

Following a suggestion of a reviewer, the experiments were replicated using artificially generated returns, and similar results were observed. The artificially generated returns were obtained from an artificial neural network with 10 hidden units (i.e., more than the 3 units used in the prediction module), and with additive noise on the return. Again we observed that decreases in mean squared error of the predictor were not very correlated with increases in excess return. When training with respect to the financial criterion instead, the average excess return on the test period increased from 4.6% to 6.9%. As in the experiments with real data, the financial performance on the training data was even more significantly superior when using the financial criterion (corroborating the hypothesis that as far as the financial criterion is concerned, the direct optimization approach offers more capacity than the indirect optimization approach).

Table 1: Comparative results: network trained with Mean Squared Error to predict future return vs network trained with financial criterion (to directly maximize return). The averages and standard deviations are over 10 experiments. The test set represents 6 years, 03/89-02/95.

	Average Excess Return on Training Sets	(Standard Deviation)	Average **Excess Return** on Test Sets	(Standard Deviation)
Net Trained with MSE Criterion	8.9%	(2.4%)	**2.9%**	(1.2%)
Net Trained with Financial Criterion	19.9%	(2.6%)	**7.4%**	(1.6%)

6 Conclusion

We consider decision-taking problems on financial time-series with learning algorithms. Theoretical arguments suggest that directly optimizing the financial criterion of interest should yield better performance, according to that same criterion, than optimizing an intermediate prediction criterion such as the often used mean squared error. However, this requires defining a differentiable decision module, and we have introduced a "soft" trading module for this purpose. Another theoretical advantage of such a decision module is that its parameters may be optimized numerically from the training data.

The inadequacy of the mean squared error criterion was suggested to us by the poor correlation between its value and the value of the financial criterion, both on training and test data.

Furthermore, we have shown with a portfolio management experiment on 35 Canadian stocks with 10 years of data that the more direct approach of optimizing the financial criterion of interest performs better than the indirect prediction approach.

In general, for other applications, one should carefully look at the ultimate goals of the system. Sometimes, as in our example, one can design a differentiable cost and decision policy, and obtain better results by optimizing the parameters with respect to an objective that is closer to the ultimate goal of the trained system.

6.0.1 Acknowledgements

The author would like to thank S. Gauthier and F. Gingras for their prior work on data preprocessing, E. Couture and J. Ghosn, for their useful comments, as well as, the NSERC, FCAR, IRIS Canadian funding agencies for support. We would also like to thank André Chabot from Boulton-Tremblay Inc. for the economic data series used in these experiments.

References

[Ben96] Y. Bengio. *Neural Networks for Speech and Sequence Recognition.* International Thompson Computer Press, London, UK, 1996.

[BG91] L. Bottou and P. Gallinari. A framework for the cooperation of learning algorithms. In R. P. Lippman, R. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 781–788, Denver, CO, 1991.

[DBG91] X. Driancourt, L. Bottou, and P. Gallinari. Learning vector quantization, multi-layer perceptron and dynamic programming: Comparison and cooperation. In *International Joint Conference on Neural Networks*, volume 2, pages 815–819, 1991.

[GB97] Joumana Ghosn and Yoshua Bengio. Multi-task learning for stock selection. In *Advances in Neural Information Processing Systems 9*, volume 9, Cambridge, MA, 1997. MIT Press.

[HK92] J. B. Hampshire and B. V. K. Vijaya Kumar. Shooting craps in search of an optimal strategy for training connectionist pattern classifiers. In J. Moody, S. Hanson, and R. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 1125–1132, Denver, CO, 1992. Morgan Kaufmann.

[HW90] John B. Hampshire and Alexander H. Waibel. A novel objective function for improved phoneme recognition using time-delay neural networks. *IEEE Transactions of Neural Networks*, 1(2):216–228, June 1990.

[RL91] Michael D. Richard and Richard P. Lippmann. Neural network classifiers estimate Bayesian a-posteriori probabilities. *Neural Computation*, 3:461–483, 1991.

[Whi89] H. White. Learning in artificial neural networks: A statistical perspective. *Neural Computation*, 1(4):425–464, 1989.