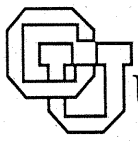
CLEARNING

Andreas S. Weigend, Hans Zimmerman, & Ralph Neuneier

CU-CS-772-95



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

ANY OPINIONS, FINDINGS, AND CONCLUSIONS OR RECOMMENDATIONS EXPRESSED IN THIS PUBLICATION ARE THOSE OF THE AUTHOR(S) AND DO NOT NECESSARILY REFLECT THE VIEWS OF THE AGENCIES NAMED IN THE ACKNOWLEDGMENTS SECTION.

University of Colorado Computer Science Technical Report CU-CS-772-95
May 1995. ftp://ftp.cs.colorado.edu/pub/Time-Series/MyPapers/clearning.ps

Clearning

Andreas S. Weigend
Department of Computer Science
and Institute of Cognitive Science
University of Colorado
Boulder, CO 80309-0430, USA
andreas@cs.colorado.edu*

Hans Georg Zimmermann
Ralph Neuneier
Siemens AG, ZFE T SN 4
Otto Hahn Ring 6
D–81739 München, Germany
georg.zimmermann@zfe.siemens.de

Most connectionist modeling assumes noise-free inputs. This assumption is often violated. This paper introduces the idea of *clearning,* of simultaneously *cleaning* the data and *learning* the underlying structure. The cleaning step can be viewed as top-down processing (where the model modifies the data), and the learning step can be viewed as bottom-up processing (where the data modifies the model). Clearning is used in conjunction with standard pruning.
This paper discusses the statistical foundation of clearning, gives an interpretation in terms of a mechanical model, describes how to obtain both point predictions and conditional densities for the output, and shows how the resulting model can be used to discover properties of the data otherwise not accessible (such as the signal-to-noise ratio of the inputs).
This paper uses clearning to predict foreign exchange rates, a noisy time series problem with well-known benchmark performances. On the out-of-sample 1993-1994 test period, clearning obtains an annualized return on investment above 30%, significantly better than an otherwise identical network (with the same inputs, size and pruning, but without clearning). The final ultra-sparse network with 36 remaining non-zero input-to-hidden weights (of the 1035 initial weights between 69 inputs and 15 hidden units) is very robust against overfitting. This small network also lends itself to interpretation.

1 Introduction

Most connectionist modeling assumes noise-free inputs: the data are used to build the model in a bottom-up approach. In contrast, the basic assumption here is that the data is noisy and the data set size is limited—conditions fulfilled for example on most financial problems with daily data.[1] This paper introduces a formalism for building numerical models using top-down information: the emerging model is allowed to modify data if the cost of changing the data is smaller than the benefit associated with lower output errors. Conceptually, the method consists of two steps:

1. **learning:** use the data to modify the model (structure);
2. **cleaning:** use the structure to modify the data (observations).

We thus use the term *clearning* to describe the simultaneous application of both steps in model building. The trade-off between the belief in the data and the belief in the model can also be viewed as an *observer-observation dilemma.*

In this paper, we apply clearning to forecasting, in essence a regression problem.[2] So far, all connectionist models for regression assume noise-free inputs and try to find a regression surface that approximates the targets (or desired

*http://www.cs.colorado.edu/~andreas/Home.html

[1]Noise in financial data can come from several source. One source is the entering of the data (such as wrong numbers), or unreliable timing in quotes (no longer tradable). Even if entered correctly, the available data might be poor indicators of the underlying economic processes (e.g., industrial production and unemployment). Furthermore, assessing quantities such as the GNP or the tax revenue is difficult and often needs revision. Finally, there always are external influences that are not captured by the inputs into the model; they also show up as noise.

[2]The idea of clearning can also be applied to other areas, such as classification: Let us consider the case of two classes (e.g., input patterns of the first class belong to a trending market, input patterns of the second class represent a side market). On the one hand, if these clusters are well separated, it is easy to find a decision boundary. On the other hand, if the clusters overlap, a flexible model will be able to find a complicated boundary that will not generalize well to new patterns. Cleaning the data corresponds to moving patterns on the training set closer to their centers, reducing the overlap on the training set, and allows a simpler decision structure to match the cleaned data.

values) associated with the inputs. This standard approach breaks the potential symmetry between inputs and outputs. Particularly in univariate time series prediction where the inputs are simply lagged values of the output, assuming noise-free inputs is clearly inconsistent.

Having very noisy data and very flexible methods (such as neural networks) is a potentially dangerous combination: if the model is too flexible, it will not only model the signal but also the noise, yielding poor out-of-sample performance. Moreover, outliers absorb resources: as learning proceeds, the hidden units shift to the location of the outliers since outliers cause the largest error signals. Consequently, there are not enough resources left to approximate the true structure in the lower-noise regions.

In order to obtain good generalization (or out-of-sample performance) on problems with finite, noisy data sets, such flexible models require regularization. We briefly describe some of the methods that are useful on financial data. One of them, pruning, will be discussed in detail in Section 3 since it plays a crucial role in combination with clearning (discussed in the Section 2).

- **Stop early.** The complexity of the model, expressed as an effective number of hidden units, gradually increases with training time (iterations of backpropagation) [Weigend, 1994]. Starting training with small weights and stopping early introduces a preference for linear models, a serious problem when the nonlinear signal is masked by noise. We always monitor the error on a validation set as function of training time. When it starts going up, it is time to bring in some of the following techniques.
- **Penalize network complexity.** The simplest regularization *weight-decay*, known as ridge regression in the statistics community, penalizes weights proportionally to their squared value. It also represents a bias towards linear models. Adding a complexity term to the cost function that in essence counts the number of significantly sized weights is known as *weight-elimination* [Weigend et al., 1990].
- **Prune weights.** The size of a weight compared to the standard deviation of its fluctuations during a training epoch indicates the reliability of the information coded in the weight. (The fluctuations occur in response to the training inputs on a pattern-by-pattern basis.) As explained in detail in Section 3, we start with an oversized network, rank all weights in terms of a test statistic (Eq. 8), and remove those of low significance in order to obtain a sparse network topology. Pruning limits the ability of the network to memorize the training data without introducing a bias towards linear models [Finnoff et al., 1993, Cottrell et al., 1995].
- **Add tasks.** In order to reduce overfitting and stabilize learning on very noisy data, we bestow supplementary tasks upon the network in the form of additional output units with their own targets [Weigend et al., 1992]. Such multi-task learning can be viewed from several perspectives: (1) the extra outputs provide additional information about the task to the network, (2) they restrict the possible solution space to that subset consistent with all tasks, thus acting as a regularizer, (3) they change the gradient-descent search path through weight space, hopefully visiting good areas on the way. The relative weight given to the output units can be adjusted during learning. A principled way of scheduling additional tasks has been developed it the following framework of hints.
- **Hints.** [Abu-Mostafa, 1995] gives the example of the symmetry-hint for predicting foreign exchange data. This hint corresponds to viewing exchange rate returns first from one country, then from the other country. The hint suggests that the dynamics should be the same. During training, the cost functions switches back and forth between descent on the performance cost function (i.e., learning to predict the returns), and descent on the hint (i.e., learning to minimize the difference in response to a pattern and to the flipped version of the pattern).
- **Pseudo-data.** The idea of adding noise to the inputs in order to prevent overfitting has been used in [Weigend et al., 1991] and compared to weight-elimination. Adding independent Gaussian noise to each input in each presentation can be seen as an on-line implementation of smearing the data by kernels.The difference between hints and pseudo-data is that the input vectors for descending on the hints are drawn randomly, whereas pseudo-data stay close to the actual data, possibly leading to overfitting in the vicinity of the training points.

Like pseudo-data, the clearning approach proposed in this paper also focuses on noise in the inputs.[3] Let us contrast the two methods: Rather than adding different random noise to the inputs at each training iteration in pseudo-data, clearning

[3]Clearning can be related to the method of *total least squares* [Huffel and Vanderwalle, 1991] (in the context of linear models), to *error in variables* [Seber and Wild, 1989], to methods dealing with *missing data* [Buntine and Weigend, 1991, Tresp et al., 1994], and to *bounded influence* models in econometrics.

prevents the network from overfitting by moving each input pattern to a more likely location, based on information from the corresponding target value and the model. Note that during training, the data is used to adjust the model, and the model is used to adjust the data: clea(r)ning is only possible through a model.

2 Clearning = Cleaning and Learning

Clearning makes two basic assumptions. First, that there actually is a cleaner input value, carrying the hope for a better model, i.e., a model with less stochasticity. The second assumption is that cleaning moves the inputs closer to that true value; our task now is to show how this is achieved with gradient descent in a backpropagation framework.

Suppressing pattern indices, the total per-pattern **cost function** is given by the sum of two terms,

$$E = \frac{1}{2}\eta \left(y - y^d\right)^2 + \frac{1}{2}\kappa \left(x - x^d\right)^2 . \tag{1}$$

The first term, $E^y = \frac{1}{2}\eta \left(y - y^d\right)^2$, is the usual squared error term between network output $y = y(x, w)$ and the data output, y^d. (The symbol w denotes the vector of model parameters.) The second term, $E^x = \frac{1}{2}\kappa \left(x - x^d\right)^2$, is the squared deviation between the cleaned input x and the data input, x^d.

There are two sets of **update rules**, the update rules for the weights, and the update rules for the input values. The gradient descent update rule for the weights is identical to standard backpropagation

$$w_{i+1} = w_i - \frac{\partial E}{\partial w} = w_i - \eta \left(y - y^d\right) \frac{\partial y}{\partial w} \quad . \tag{2}$$

The update rule for the cleaned input x_i at iteration i is given by

$$x_{i+1} = x_i - \frac{\partial E}{\partial x} \quad . \tag{3}$$

Rewriting x_i as a sum of the original data point x^d and a correction term Δ_i,

$$x_i = x^d + \Delta_i \quad , \tag{4}$$

the cleaning update rule can be expressed most easily as

$$\Delta_{i+1} = (1 - \kappa)\Delta_i - \eta \left(y - y^d\right) \frac{\partial y}{\partial x} \quad . \tag{5}$$

The elements of the update rule for the input correction term are:

- **Exponential decay of the correction term Δ.** Without new "impulses," i.e., the second term, Δ exponentially decays to zero, with a decay constant of $1 - \kappa$. $(0 < \kappa < 1)$.
- **Proportionality to the output error $\left(y - y^d\right)$.** This is the same proportionality as in standard error backpropagation: the larger the deviation, the larger its effect on the update (in this case on the cleaning). Note that the learning rate η enters here (rather than the cleaning rate κ) since it describes the scaling of the output error.
- **Proportionality to the sensitivity of the output with respect to the input, $\partial y/\partial x$.** This quantity, already computed in the backpropagation step, describes the slope (gradient) of the surface at the present operating point x (the cleaned value; the network no longer sees x^d!). If the slope is small, not much can be gained by moving the input; if the slope is large, moving the input has a big effect on the output error.

The cost function Eq. 1 contains two parameters: the learning rate η, and the cleaning rate κ. These parameters can be interpreted from the perspective of classical mechanics as spring constants (E = $\frac{1}{2}k\Delta^2$); from the perspective of statistics as the inverse of a noise variance.

A **mechanical interpretation** of the cost function and the (relative) learning and cleaning rates is given in Fig. 1. Data points are placed in the joint (input $\times$ output) space. The regression output (network response) can be viewed as a surface above the input space. The data points are attached to the surface with vertical springs; these springs store the (internal) energy. There are two "stiffnesses:" the stiffness of the regression surface and the stiffness η of the springs.

At the one extreme, an infinitely flexible model would just go through all of the data points. At the other extreme, infinitely weak springs would not modify the model from its prior value (e.g., from a hyperplane).

For cleaning, we add springs in the input space, connecting each input data point x^d with its cleaned value x. The energy stored is $\frac{1}{2}\kappa\Delta^2$, where κ is the spring constant, and $\Delta = x^d - x$ is the amount the input spring is stretched. Minimizing the total cost function (Eq. 1) corresponds to minimizing the total energy stored in the input springs and the output springs (averaged over all patterns). The ratio between η and κ describes the tradeoff between the stiffnesses (or importances) between the output errors and input errors. In an autoregressive model, where outputs and inputs are the same series, we set $\eta = \kappa$.

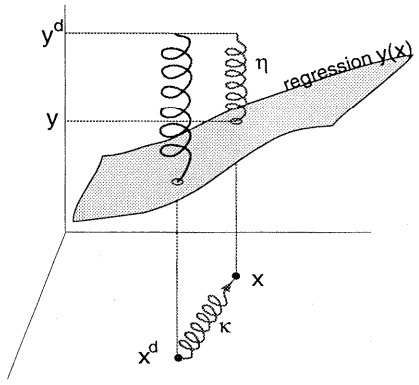


Figure 1: Mechanical analogy of the trade-off between learning and cleaning. In contrast to error-free input modeling, we here attach a spring to the data point x^d and allow that spring to be stretched to x for a price: the energy stored in that spring is $\kappa \left(x^d - x\right)^2/2$, where κ is the spring constant. The ground state is reached when the sum of this input energy and the energy stored in the output spring (drawn next to η in the figure) is minimal.

A **statistical interpretation** of the cost function can be given in a maximum likelihood framework [Rumelhart et al., 1995]. We assume that each pattern was generated by a "true" input (estimated by x) and a "true" output (estimated by y). We then assume that all input and output components were independently corrupted by additive Gaussian noise. The ratio of the noise levels (variances) of output to input was κ/η.

Reversing this argument: the statistical interpretation, in conjunction with the model that we have obtained, allows us to *characterize the inputs* by their noise levels. The total variances of the inputs (i.e., sum of the noise and the signal) are easily computed (and indeed routinely used to scale the inputs). The present method allows us to estimate the two parts–the noise level and the signal level–separately! We record the deviations $\Delta(t)$ of each input and of each output as functions of the pattern index t. (t is the time when each prediction is made, plotted along the horizontal axis.) This plot (not shown here) allows us to discover properties about the data:

- **Detect outliers.** Plotting the squared errors of all the inputs and the outputs for each pattern as a function of time (i.e., the element-by-element squared $\Delta(t)$ matrix; plotted in a similar way as a spectrogram) allows us to extract three signatures: Individual spikes indicate a data-entry type error. Horizontal strips (or bars) indicate an outlier in a raw input that was smeared out during preprocessing in smoothing operation (e.g., a moving average training signal). Vertical strips indicate outliers in the output: several inputs try to compensate for the output outlier, and the output itself also has a large error.
- **Characterize input variables by their stochasticity.** Taking the mean of the squared errors across time (i.e., computing the mean squared errors of each of the inputs) allows us to characterize the signal-to-noise ratio of each input feature. Note that this information cannot be obtained without a model.
- **Estimate error covariance matrix.** Computing the covariance matrix of the errors allows us to investigate the validity of the assumption of statistical independence of the noise of the inputs. If there are significant non-zero off-diagonal contribution, the modeling can be improved by transforming the data by the inverse of the noise covariance matrix. Last, but not least, histogramming the individual entries of the $\Delta(t)$ matrix and comparing them to a Gaussian is a check of the underlying noise assumptions.

Before turning to the other ingredient (pruning) and the simulations, we describe how to obtain predictions. The first

case of point predictions, i.e., estimating the conditional expectation of the next value, is straightforward. The second case of predicting the conditional probability distribution over the outputs exploits the noise structure in the inputs, revealed by clearning.

- **Predicting the conditional expectation of the next value (point predictions).** Once we have built the model, point predictions are obtained by a feed-forward pass through the network. In principle, we use cleaned data whenever available: e.g., any lagged variable in the input should be replaced by its cleaned value.[4] However, in cases where most variables are not simply past values of the outputs but more complicated indicators, we simply use the raw data at all the inputs in the final feed-forward step.
- **Predicting the conditional probability density of the next value.** Elsewhere, we have emphasized the importance of knowing the accuracy of a prediction.[5] The clearning algorithm puts us in the fortunate position of being able to estimate the error in the outputs due to the uncertainty in the inputs. We start by computing the matrix of empirical input errors $\Delta(t)$; this matrix consists of one vector (across inputs) for each time step (or pattern). In order to forecast the probability density of the next value, we use today's input vector (as in the case of point predictions), randomly pick one of the empirical noise vectors, add it to today's input, and record the resulting output. We then draw (with replacement) a second vector from the set of empirical noise vectors, add it to today's (original) input, and generate and record the corresponding output. We repeat this bootstrapping procedure several hundred or a few thousand times, and present tomorrow's probability distribution as a histogram of these predictions.[6]

This section has introduced clearning. Its strength shines when used in conjunction with pruning: simpler data (obtained through clearning) allow for simpler models (obtained through pruning [Finnoff et al., 1993, Cottrell et al., 1995]).

3 Pruning

The previous section discussed how to use the evolving model to modify the input data. In the introductory section, we had already discussed some standard methods of regularization in neural networks. The present section gives the details of the pruning algorithm. It is the combination of clearning and pruning that allows us arrive at very small networks that are very robust against overfitting: simpler data allow for a simpler model.

In more detail, we clearn until we observe overfitting, indicated by an increase of $\sum(y - y^d)^2$ on the validation set. We want to remove the weights that respond most to the noise. Consider a specific weight w in the network. The key idea is the following: we present one epoch (iteration) of input patterns $t = 1, \ldots, N$. We then compare the size of the weight (at the end of the epoch) to its fluctuations in response to the inputs during that epoch.

Let ξ_t denote the weight change in response to pattern t. (In gradient descent: $\xi_t \propto -\partial E_t / \partial w$.) The mean and standard deviation of the weight updates over the epoch are given by

$$\text{mean}(\xi) = \frac{1}{N} \sum_{t=1}^{N} \xi_t \qquad \text{mean weight change (over epoch)} \quad , \tag{6}$$

and

$$\text{std}(\xi) = \sqrt{\text{var}(\xi)} = \sqrt{\frac{1}{N} \sum_{t=1}^{N} \left[\xi_t - \text{mean}(\xi)\right]^2} \qquad \text{rms weight change (fluctuations)} \quad . \tag{7}$$

[4]This also applies to variables derived from cleaned inputs; e.g., a moving average should be replace by its cleaned value.

[5]Examples are (1) estimating uncertainties due to the splitting of the data [Weigend and LeBaron, 1994]–crucial when validation sets are set aside to determine meta-parameters, (2) estimating confidence intervals for unimodal distributions [Nix and Weigend, 1995]–a connectionist implementation of ARCH models, and also useful for problems that consider Sharpe ratios, (3) obtaining model-free distributions with the method of fractional binning [Weigend and Srivastava, 1995]–important for multi-modal processes, e.g., when we expect a big move that could go either up or down, and (4) finding trading days where we can trust our model to a higher than average degree, using the method of gated experts [Weigend and Mangeas, 1995]–important for very noisy processes where additional information both about the certainty and the noise level of the regime can be crucial.

[6]If we want more resamplings than we have input patterns, the individual errors to be added to the inputs can be drawn independently, as opposed to drawing an entire vector at each time). This is a good approximation if the off-diagonal elements of the correlation matrix of the input noise are small, and if higher-order correlations are not important.

We combine these two quantities to the following test statistic:

$$\textbf{test value} \; : \; \frac{|w + \text{mean}\,(\xi)\,|}{\text{std}\,(\xi)} = \frac{|\text{weight at end of epoch}|}{\text{fluctuations during epoch}} \quad . \tag{8}$$

If this test value is large, we keep the weight since it is well determined (the fluctuations are small compared to the size of the weight). If the test value is small, we prune the weight since it is not well determined (the size of the weight is small compared to its fluctuations). We are primarily interested in pruning connections from the inputs to the hidden units.All the weights of the input-to-hidden layer are ranked according to their values of this test statistic. (We are primarily interested in thinning out the first layer of weights.)

One further decision is needed: the test value can be evaluated either on the raw or on the cleaned inputs. We chose the raw (uncleaned) data for two reasons:

1. The fluctuations of the weights mirror the noise present in the data. Removing the weights that fluctuate most can be viewed as a noise filter.
2. In prediction mode, the most recent inputs are only available in raw form. Using in pruning the raw inputs everywhere is thus closer to the final task particularly when the predictions are made with raw inputs only (as in the example in this paper; see the discussion at the end of Section 2).

We clearn until overfitting sets on. We then do one pruning epoch. We then return to clearning, until the next indication of overfitting. At overall early stages of the entire procedure, we reinitialize the now smaller network with a new set of random weights (of the order of 10^{-4}) and reset the cleaning correction vector Δ to zero (defined in Eq. 4). This restart of the smaller architecture corresponds to a search in a reduced subspace. At the later stages of the entire procedure, we omit the re-initialization part. Re-initialization presents, just like early stopping and weight-decay, a bias towards linear models that we want to avoid for the final model.

When does the entire process stop? The test statistic is evaluated on all weights, not only the "survivors." A large test value of supposedly "dead" weights indicates overpruning. In the final stage, we resurrect these weights, and train to a local minimum. The final, ultra-sparse structures are very robust against overfitting. We evaluated the clearning-and-pruning procedure both on computer generated data and on real-world data. The following section picks one particularly interesting example where benchmarks are well known: the very noisy problem of forecasting daily foreign exchange rates between the US Dollar and the German Mark.

4 Example: Exchange rate predictions

We now demonstrate the proposed method on the problem of predicting daily foreign exchange rates between the US Dollar and the German Mark. We used data from January 15, 1985 through January 27, 1994. We first set aside the last 216 days (starting April 1, 1993) for the test period. From the remaining data, we set aside every fourth day as validation set (or tuning set) used to estimate the generalization performance.

The architecture is a simple feed-forward network with 15 tanh hidden units.[7] There are 69 inputs. 12 inputs reflect information derived from the series itself (relative strength index, skewness, point and figure chart indicators, ...). 57 inputs reflect fundamental information beyond the series itself (indicators depending on exchange rates between different countries, interest rates, stock indices, currency futures, ...). These inputs contain information from six countries (France, Germany, Japan, Switzerland, UK, and USA).

The network has 3 outputs. The first output predicts the *return.* We use a normalized version of the return: we divide the logarithm of the ratio of tomorrow's price over today's price by the standard deviation computed over the last 10 trading days. A network solely trained on the one-day has a hard time capturing long-term dynamics of the market. We want to focus the network on information about the next turning point, defined as the next maximum or minimum of the daily series, whichever comes next. Specifically, we give it two additional outputs, one for the *number of days to next turning point,* the other for the *return between today and the next turning point* (divided by the standard deviation of the data).

Fig. 2 shows the resulting network. As baseline for comparison, pruning alone (without clearning) allows us to reduce the 1035 potential weights between inputs and hidden units to 60 weights. Applying the clearning-and-pruning

[7] We use the standard tanh architecture to facilitate the comparison with other modeling techniques. However, faster convergence and more stable results are often obtained with a hidden layer of normalized sigmoids.

procedure, only 36 weights survive. These remaining weights encode nonlinearities; their values are between -1.6 and $+1.7$ with a standard deviation of 0.8.[8] The proposed clearning-and-pruning procedure acts also as feature selection: of the 69 inputs, only 20 inputs survive.

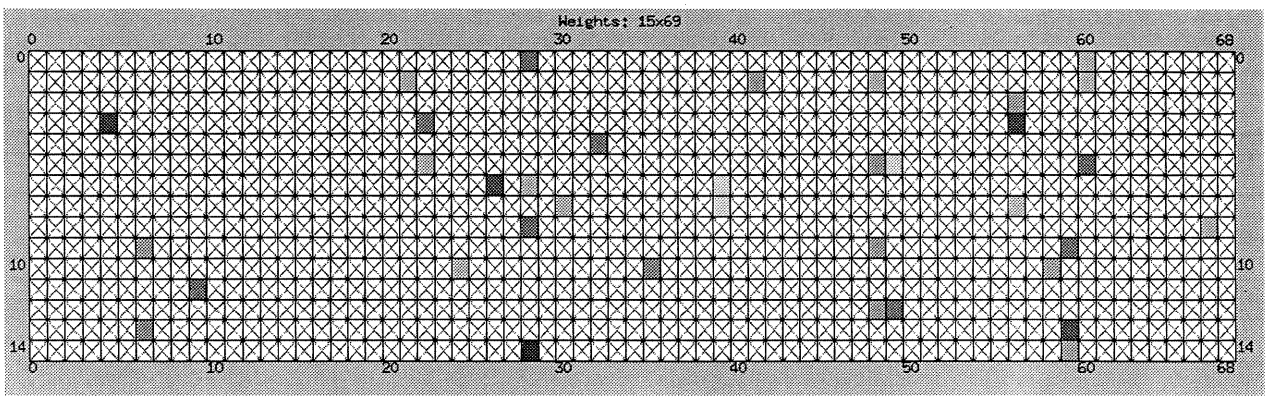


Figure 2: The weight diagram of the weights between inputs and hidden units for the exchange rate example. The larger the weight, the darker the field. (The darkest gray corresponds to an absolute value of 1.7; in gray-scale rendering, we lose the information about the sign of the weight.) An "X" indicates that the weight has been pruned away.

When computing the return on investment on the test set, we take full position size as given by the sign of the first output (trained to predict the return). The profit and loss curves shown in Fig. 3 include a transaction cost of 0.001. The annualized return on investment is above 30% in the out-of-sample period with small maximum draw down.

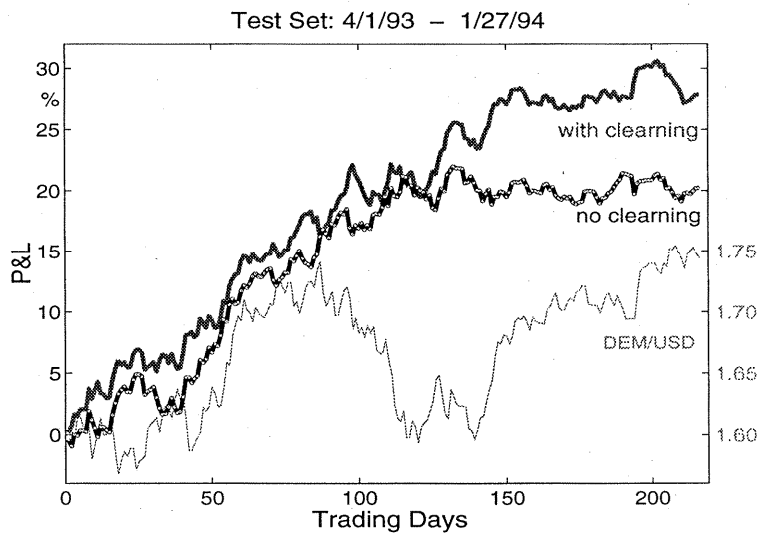


Figure 3: Performance on the held-out test set. The top curve uses clearning and pruning. The curve below only pruning. The scale on the left is for these profit and loss curves. We also give the exchange rate during this time period (bottom curve); its scale is on the right.

Acknowledgments

We thank Art Owen for suggesting to bootstrap the input errors to obtain the distribution of the forecast, and Barak Pearlmutter for sharing his intuitions about the cost function. The simulations were carried out with SENN (Simulation Environment for Neural Networks) at the University of Colorado at Boulder. Andreas Weigend acknowledges support by the National Science Foundation (Grant No. RIA ECS-9309786).

[8]The standard deviation of the offsets or biases of the hidden units is 0.4. We also have direct connections between the inputs and the outputs, but their weight values are negligible.

References

[Abu-Mostafa, 1995] Abu-Mostafa, Y. (1995). Hints. *Neural Computation*, 7:(in press).

[Buntine and Weigend, 1991] Buntine, W. L. and Weigend, A. S. (1991). Bayesian back-propagation. *Complex Systems*, 5:603–643.

[Cottrell et al., 1995] Cottrell, M., Girard, B., Girard, Y., Mangeas, M., and Muller, C. (1995). Neural modeling for time series: a statistical stepwise method for weight elimination. *IEEE Transaction on Neural Networks*, (in press).

[Finnoff et al., 1993] Finnoff, W., Hergert, F., and Zimmermann, H. G. (1993). Improving generalization performance by nonconvergent model selection methods. *Neural Networks*, 6:771–783.

[Huffel and Vanderwalle, 1991] Huffel, S. V. and Vanderwalle, J. (1991). The total least squares problem: Computational aspects and analysis. In *Frontiers in Applied Mathematics*, volume 9. SIAM.

[Nix and Weigend, 1995] Nix, D. A. and Weigend, A. S. (1995). Local error bars for nonlinear regression and time series prediction. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7 (NIPS*94)*. MIT Press, Cambridge, MA.

[Rumelhart et al., 1995] Rumelhart, D. E., Durbin, R., Golden, R., and Chauvin, Y. (1995). Backpropagation: The basic theory. In Chauvin, Y. and Rumelhart, D. E., editors, *Backpropagation: Theory, Architectures, and Applications*, pages 1–34, Hillsdale, NJ. Lawrence Erlbaum Associates.

[Seber and Wild, 1989] Seber, G. A. F. and Wild, C. J. (1989). *Nonlinear Regression*. Wiley, New York.

[Tresp et al., 1994] Tresp, V., Ahmad, S., and Neuneier, R. (1994). Training neural networks with deficient data. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6 (NIPS*93)*, pages 128–135, San Francisco, CA. Morgan Kaufmann.

[Weigend, 1994] Weigend, A. S. (1994). On overfitting and the effective number of hidden units. In Mozer, M. C., Smolensky, P., Touretzky, D. S., Elman, J. L., and Weigend, A. S., editors, *Proceedings of the 1993 Connectionist Models Summer School*, pages 335–342, Hillsdale, NJ. Lawrence Erlbaum Associates.

[Weigend et al., 1990] Weigend, A. S., Huberman, B. A., and Rumelhart, D. E. (1990). Predicting the future: A connectionist approach. *International Journal of Neural Systems*, 1:193–209.

[Weigend et al., 1992] Weigend, A. S., Huberman, B. A., and Rumelhart, D. E. (1992). Predicting sunspots and exchange rates with connectionist networks. In Casdagli, M. and Eubank, S., editors, *Nonlinear Modeling and Forecasting*, pages 395–432. Addison-Wesley.

[Weigend and LeBaron, 1994] Weigend, A. S. and LeBaron, B. (1994). Evaluating neural network predictors by bootstrapping. In *Proceedings of International Conference on Neural Information Processing (ICONIP'94)*, pages 1207–1212. Technical Report CU-CS-725-94, Computer Science Department, University of Colorado at Boulder, ftp://ftp.cs.colorado.edu/pub/Time-Series/MyPapers/bootstrap.ps.

[Weigend and Mangeas, 1995] Weigend, A. S. and Mangeas, M. (1995). Analysis and predictions of multi-stationary time series using nonlinear gated experts. Technical Report CU-CS-764-95, University of Colorado at Boulder, ftp://ftp.cs.colorado.edu/pub/Time-Series/MyPapers/experts.ps.

[Weigend et al., 1991] Weigend, A. S., Rumelhart, D. E., and Huberman, B. A. (1991). Generalization by weight-elimination with application to forecasting. In Lippmann, R. P., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 3 (NIPS*90)*, pages 875–882. Morgan Kaufmann.

[Weigend and Srivastava, 1995] Weigend, A. S. and Srivastava, A. N. (1995). Predicting conditional probability distributions: A connectionist approach. *International Journal of Neural Systems*, 6.