Working Paper IS-97-007, Leonard N. Stern School of Business, New York University.
In: *Journal of Computational Intelligence in Finance* **6** (1998) 14–23.
(Special Issue on "Improving Generalization of Nonlinear Financial Forecasting Models".)
http://www.stern.nyu.edu/~aweigend/Research/Papers/InteractionLayer

EXPLOITING LOCAL RELATIONS AS SOFT CONSTRAINTS TO IMPROVE FORECASTING

ANDREAS S. WEIGEND
Department of Information Systems
Leonard N. Stern School of Business
New York University
44 West Fourth Street, MEC 9-74
New York, NY 10012
aweigend@stern.nyu.edu
www.stern.nyu.edu/~aweigend

HANS GEORG ZIMMERMANN
Siemens AG, Corporate Research
Otto-Hahn-Ring 6
D-81730 München, Germany
georg.zimmermann@mchp.siemens.de

Abstract. Predictive models for financial data are often based on a large number of plausible inputs that are potentially nonlinearly combined to yield the conditional expectation of a target, such as a daily return of an asset. This paper introduces a new architecture for this task: On the output side, we predict dynamical variables such as first derivatives and curvatures on different time spans. These are subsequently combined in an *interaction output layer* to form several estimates of the variable of interest. Those estimates are then averaged to yield the final prediction. Independently from this idea, on the input side, we propose a new *internal preprocessing layer* connected with a diagonal matrix of positive weights to a layer of squashing functions. These weights adapt for each input individually and learn to squash outliers in the input. We apply these two ideas to the real world example of the daily predictions of the German stock index DAX (Deutscher Aktien Index), and compare the results to a network with a single output. The new six layer architecture is more stable in training due to two facts: (1) More information is flowing back from the outputs to the input in the backward pass; (2) The constraint of predicting first and second derivatives focuses the learning on the relevant variables for the dynamics. The architectures are compared from both the training perspective (squared errors, robust errors), and from the trading perspective (annualized returns, percent correct, Sharpe ratio).

1 Introduction

Historically, the problem of time series prediction has often been reduced to mere regression or pattern recognition.[a] This popular approach ignores any explicit relation between adjacent patterns in time. In fact, the presentation order of the patterns is often randomized during training in order to avoid local minima and to improve the training speed.

On the other extreme, fully recurrent networks allow for a perfect representation of temporal dependencies. Unfortunately, despite a lot of effort in the last decade, fully recurrent networks have proven difficult to train, in particular for long-term dependencies.

Recently, two new classes of approaches have been applied to financial markets that are located between these two extremes. The first class employs the idea of a hidden state that cannot be inferred from a finite window in time. This state can be continuous (see Timmer & Weigend (1997) for the application of state space models to volatility prediction), or it can describe the switching between several discrete models (see Shi & Weigend (1997) for the application of Hidden Markov experts for predictions of returns). These examples of the first class rely on heavy statistical machinery to estimate the hidden states.

The second class, described in this paper, focuses on the variables used for the targets of the network. It incorporates dynamics as architectural constraints, such as the first and second derivatives of the dynamical system. It turns out that this architecture improves learning behavior since it uses several outputs that are variations of the quantity of interest. Apart from providing perspectives of the problem on different time scales, this approach enables an information flow in the backward pass of an amount similar to that in the forward pass. In contrast to the first class of approaches mentioned in the previous paragraph, this second class maintains the training simplicity of a feed-forward network and does not introduce the notion of hidden states.

We begin by discussing the first point, the use of variables that formulate the problem in terms of adequate dynamic quantities. A common approach to time series prediction is using (potentially nonlinear) ARMA models. Their main goal is to obtain a point prediction of a future value by regressing the point to be predicted onto past values of the same observable, as well as on external variables.

[a] A pattern typically consists of a target and several inputs, such as lagged values of the time series variable.

When such simple point predictions are carried out with a very rich non-linear model class, such as a neural network or connectionist model, they often lead to the serious problem of overfitting. For noisy data in particular, the small amount of information contained in a single point does not suffice in determining the structure of the model and the values of all the parameters. Because of high flexibility and usually insufficient constraints, the model tends to memorize the training points well, but the out-of-sample performance does not improve beyond chance.

In this paper we exploit the additional structure that originates from the nature of a dynamical system. This is in contrast to the assumption of arbitrary mappings between random inputs and random outputs, typically made in machine learning. Specifically, we use the following two features:

- Inspired by physics and economics, we use variables, both for the inputs and the targets, that characterize *dynamic behavior* (such as curvatures, corresponding to accelerations that are proportional to external forces driving the system).

- It is well known that the information provided by the input needs to capture the state of the system. We take the idea of state seriously also for the output side, by providing a *sufficient number of outputs* to describe the subsequent state of the systems, as opposed to almost all other prediction models that only use information from a single target (e.g., the value to be predicted).

Thus far, one might think that it would suffice to just predict several such dynamical system quantities. This more symmetrical approach between inputs and outputs would indeed have the added advantage to accelerate iterative learning procedures such as error backpropagation, since more information is "flowing back" from the target side. However, networks trained on many such outputs tend to learn the individual mappings quite well, ignoring the fact that they are related in time. In many cases, just sharing hidden units does not stipulate sufficient interactions between these outputs, particularly when the complexity of the problem requires many hidden units.

We thus add a third feature to the specification of the network architecture for time series prediction: an additional layer that we call the *interaction layer*. The interaction layer encodes the relationship between its neighbors. Examples include derivatives and curvatures on different scales between predicted points. This feature is missing in standard multivariate regression where the different

outputs typically do not have a relation to one another.

The general ideas are made concrete in Section 2 through two specific architectures: (1) point predictions followed by an interaction layer, and (2) an indicator layer of dynamical system variables followed by an interaction layer combining the dynamical system variables. Section 3 introduces another useful ingredient for modeling that is inserted between the inputs and the hidden units. This internal preprocessing layer connects each input with one preprocessing hidden unit with a hyperbolical tangens (tanh) transfer function. Adapting the corresponding weights allows for a problem-driven suppression of outliers. These architectural elements are combined in Section 4 and applied to daily forecasts of the German equity index DAX (Deutscher Aktien Index). The learning performance is shown to be more stable compared to a standard network with a single output unit, and the performance is comparable to a state-of-the-art neuro-fuzzy model (Zimmermann *et al.*, 1996). Section 5 discusses why the observed stability cannot be reached by simply reducing the learning rate.

2 Taking relations to neighbors into account: the interaction layer

This section presents two new architectures that take relations between neighboring values in time into account. The first architecture adds an interaction layer on top of a layer with individual point predictions. The second architecture (which will be used for the DAX example) first computes dynamical quantities on several time scales such as first derivatives and curvatures, and subsequently combines them to predictions of the desired value.

2.1 Architecture 1: Point-predictions followed by the interaction layer

In most applications of neural nets in financial engineering, the number of inputs is huge (of the order of a hundred), but only a single output is used. This can be viewed as a large "inverted" funnel; the hard problem is that the only information provided in learning is that of the single output. This is one of the reasons for the "data-hungryness" of single-output neural networks, i.e., a large set of training data is often required in order to distinguish nonlinearities from noise.[b]

[b] Too small data sets also imply a bias towards linear models. This is not to be confused with the bias towards linear models that is a consequence of some overfitting techniques such as early stopping or weight decay.

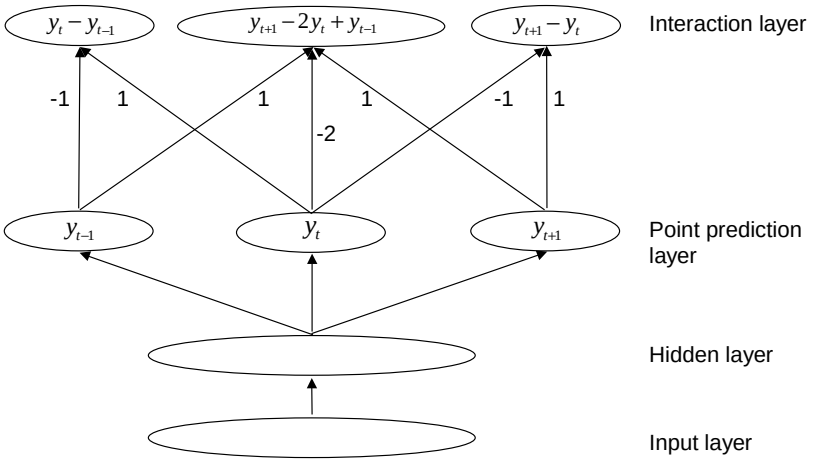

Figure 1: The first architecture, where point predictions are followed by the interaction layer.

One approach to increase the information flow from the output side to the input side is to increase the number of output units. In the simplest case, two outputs can be used, one to predict the return, and the other one to predict the sign of the return, see Weigend *et al.* (1991), as well as the experiments described by Caruana (1994) and Caruana (1997).

While the idea of increasing the information flow through multiple outputs can generally be used, the specific task of time series allows us to exploit yet another source of information. We would like to provide enough information to characterize the state of the autonomous part of the dynamics on the output side, similarly to Takens' theorem for the notion of state on the input side, see e.g., Gershenfeld (1989). In this time series context, this embedding of the output can be done analogously to the input side of a tapped "delay" line, indicated in Fig. 1 as *point prediction layer*.

The forecast we are interested in is y_t, the t-step ahead forecast of variable y. Additionally, we here also predict y_{t-1} and y_{t+1}. However, the experiments with this architecture were disappointing; contrary to our original hopes, sharing hidden units does not promote significant interactions between the outputs. This prompted us to add an explicit second output layer, the *interaction layer*. It computes the next-neighbor derivatives $(y_t - y_{t-1})$ and $(y_{t+1} - y_t)$, as well as the curvature $(y_{t+1} - 2y_t + y_{t-1})$.

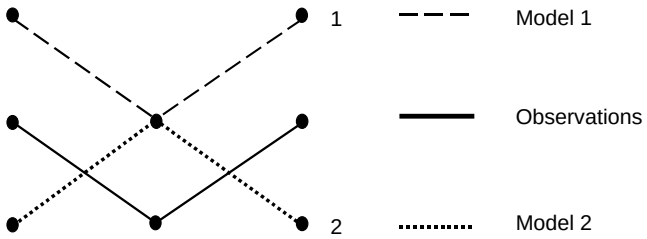

Figure 2: Geometric interpretation of the effect of the interaction layer on the cost function (schematic). Given are three curves connecting points at three adjacent steps in time. The solid line connects the target points. Model 1 (dashed line) is an example of a model that takes relationships between neighbors into account, as proposed in this paper. Model 2 (dotted line) only focuses on the quality of the point predictions. Note that we constructed this fictitious example such that both models have the same point-by-point errors, but when derivatives and curvatures are taken into account, Model 1 is favored.

Since the differences between neighbors are encoded as fixed weights, they do not have to be estimated. Therefore they do not increase the number of free parameters that have to be determined from the data. The overall cost function is the sum of all six contributions. Here we weigh all individual contributions evenly. An alternative is to give equal contribution to the error and down-weigh each error output by the average error of that output unit.

If the point forecasts were perfect, the interaction layer would have no effect at all. To explain the effect of non-zero errors, consider Fig. 2. Both predictions of the three points have the same pointwise errors at each of the three neighboring points. However, both the slopes and the curvature are correct in Model 1; they do not contribute to the error. In Model 2, however, they are nonzero and thus increase the error.[c]

Despite all of these desirable features, this architecture has a disadvantage when viewed from the input side: In order to be able to make predictions on several time horizons on the output side, we have to use the preprocessing transformations (such as moving averages) on all corresponding time scales. This unfortunately increases the number of inputs dramatically, yielding once again a strong asymmetry between the information flow in the forward and in the backward pass, defying our original intention. Furthermore, the additional parameters increase the problem of overfitting.

[c] In the case of a quadratic error function, the interaction layer can be substituted by a single output layer of point predictions that are combined with a positive definite non-diagonal quadratic form to create an equivalent target function.

2.2 Architecture 2: An indicator layer recombined by the interaction layer

The second architecture keeps the advantages of the model given above but avoids its disadvantages. Let us assume that the target of ultimate interest has the form:

$$\text{TARGET} \;=\; \frac{y_t - y_0}{y_0} = \frac{y_t}{y_0} - 1 \quad . \tag{1}$$

y denotes a price, y_0 is the value at the present time, and y_t is the price for t days into the future. This target thus corresponds to the relative return for t-day trading. We now define an indicator layer that adds a series of additional pairs of targets to this true goal:

$$\text{TARGET} \;=\; \frac{y_{t+n} + y_t + y_{t-n}}{3y_0} \;-\; 1 \tag{2}$$

$$\text{TARGET} \;=\; \frac{-y_{t+n} + 2y_t - y_{t-n}}{3y_0} \quad . \tag{3}$$

where n defines the *span* of the embedding. Eq. (2) describes a "smoothed" return centered at time t, and Eq. (3) is proportional to the curvature, normalized similarly to the average by the current price y_0. The curvature, as a second derivative, can be seen to reflect an acceleration, revealing an underlying force, as well as drawing attention to the turning points. Note that these pairs are chosen such that they add up to the ultimate target, Eq. (1). All of the targets are centered around the same midpoint t. This indicator layer is followed by the interaction layer. It recombines the N pairs of smoothed returns and curvatures, yielding N estimates of the ultimate target.

The number N of such pairs used is one of the modeling choices. In the task of one-day forecasts discussed in this paper we use three pairs, covering in their union the forces of one week. In another problem of six-month forecasts, we find six spans useful; they can be seen as attempting to cover the forces of one year.

In the final step, all of the individual forecasts in the interaction layer will be averaged in a final output layer. Combining forecasts is beneficial to the degree to which the individual forecasts are uncorrelated, see Bates & Granger (1969) and Granger (1989), as well as the corresponding ideas in the mean variance framework of modern portfolio theory, e.g., Elton & Gruber (1995).

Experimenting with this architecture, we find that forecasts of the individual averages and the individual curvatures resemble each other more than we had hoped for. If the degree of correlation between the forecasts should be

reduced, a remedy consists of including additional targets, such as combinations of dynamic variables, indicating to the learning algorithm that it should also consider the differences between the individual forecasts, not just their commonalities. In other works we use the pairwise differences of the point averages of the spans, as well as the pointwise differences of the curvatures of the spans. This corresponds to $2(N-1)$ additional targets for the interaction layer.

3 Internal preprocessing layer

In order to have potentially very different inputs on a comparable scale, we follow the common practice and standardize the input data to zero mean and unit variance. A problem with this approach is that outliers in the input can have a large impact. This is particularly serious for data in finance and economics that contain large shocks.[d] To deal with the shocks, we inserted an additional preprocessing layer between the inputs and the first hidden layer. The complete network architecture is shown in Fig. 3.

This additional preprocessing layer has the same number of hidden units as the input layer and uses standard tanh squashing functions. But the weight matrix between the input layer and the preprocessing layer is only a square diagonal matrix. The weights have initial values of 0.1, ensuring that the tanh is initially in its linear range, letting the external inputs pass through essentially unchanged.

The diagonal weights to the preprocessing layer are constrained to be positive. Furthermore, no bias term is included for the preprocessing layer to avoid numerical ambiguities. The weights are adaptive and updated in the same way as all the other weights in the network. In practice, we observe both growing and shrinking values for the weights. The growing values cause a larger proportion of the inputs to be compressed by the squashing function of the tanh. Shrinking diagonal elements indicate that the corresponding input might be a candidate to be pruned away. So as to not mix too many features in this analysis, we do not use pruning in the simple example presented in this paper.

[d] The same reasoning about shocks and outliers in the data lead us to use a robust error function on the output side. Rather than using squared errors between target and outputs, we minimize in all the examples in this paper $\frac{1}{2}\log(\cosh(2\,(\text{target - output})))$. This function approximates the parabola of the squared errors for small differences (Gaussian noise model), and is proportional to the absolute value of the difference for larger values of the difference (Laplacian noise model).

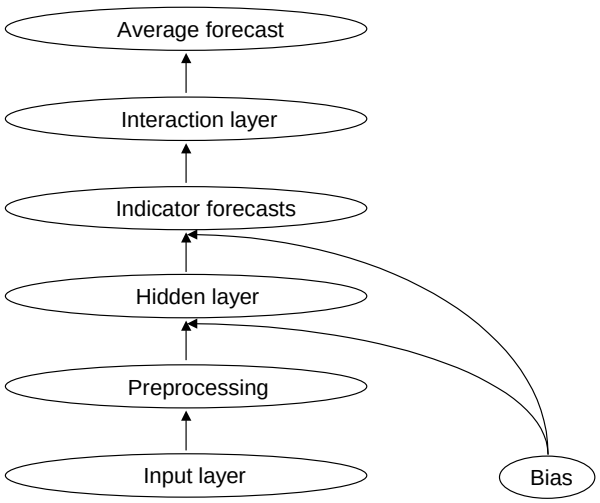

Figure 3: The second architecture, the six layer network. The novel features are the preprocessing layer that learns to scale the importance of the individual inputs, and the indicator forecasts of differences and curvatures followed by the interaction layer of individual predictions of the target that are averaged to the final forecast.

4 Real-world application: One-day ahead predictions of DAX

We now apply the ideas about the interaction layer and the preprocessing layer, introduced in the previous sections, to a real world example. We chose the benchmark problem of predicting the German stock index (DAX, Deutscher Aktien Index). Section 4.1 describes the task and the data, and Section 4.2 presents the results, and compares the new six-layer architecture to a neural network with a single output.

4.1 Data

The task is a DAX one-day ahead return forecast based on twelve input time series that include various stock indices, bond indices, and inflation indicators.[e]

[e] In full detail, the variables used are:

1. DAX Price Index (30 firms);
2. DAX Price-Earnings Ratio;
3. DAX Composite (several hundred firms);

We present information from these time series in a variety of ways to the network. The "momentum" in the market is represented by the first difference of the prices. If we only chose such first differences as inputs and nothing else, the forecast model will very likely be trending. In order to avoid the bias towards a trending model. Additional inputs include the underlying "forces" expressed as second derivatives, as well as a simple description of the "distances" to the estimated equilibria.

In the experiments reported here, we use the time period from the beginning of January 1986 up to the end of February 1993 as training set, and the period from March 1993 up to mid-August 1996 as the test set. We trained for 300 epochs without any early stopping nor pruning. (This justifies the omission of a validation set.) For performance comparison, a simple buy-and-hold strategy on this period gives an annualized return of 12 percent.

4.2 Results and Comparisons

We compare the six-layer network with a standard three-layer network using the same 44 inputs in both cases. Both networks have 40 hidden units. The resulting behavior during the first 300 epochs of learning is shown in Fig. 4 for the log(cosh(.)) target function used to train the networks, and in Fig. 5, for the normalized mean squared error.

Fig. 4 compares a standard network with a single output unit (left panel) with the six-layer architecture (right panel). The level is not normalized but reflects the different volatilities of the training and test sets. The relative shape of each curve matters, not their absolute values.

Fig. 5 shows the normalized mean squared error, E_{NMS}, as a function of the training time. Note the stabilizing effect of the six layer architecture, in

4. Dow Jones Industrial Average;
5. Nikkei 225;
6. 3-month German interest rate (reflecting decisions by the Bundesbank);
7. Yield of German circulating bonds (average over all maturities);
8. U.S. 30-year bond;
9. Morgan Stanley Price Index Germany (inflation indicator);
10. Morgan Stanley Price Index Europe (inflation indicator);
11. Gold price (inflation indicator);
12. DEM/USD exchange rate (Frankfurt closing);

comparison to a single output on the left. Furthermore, the out-of-sample behavior measured by squared errors is better for a network trained on the robust log(cosh(.)) than for a network trained directly on squared error minimization.

The next three figures, Figs. 6–8, evaluate the network from a trading perspective. The forecast needs to be translated into a strategy in order to compute financial objectives. We use a simple strategy based on the sign of the forecasting model: We buy when the predicted return is positive, and we sell short when the predicted return is negative, both to the full position size.

Using this specification, Fig. 6 shows annualized return as a function of training iterations, Fig. 7 shows the percentage of correct predictions, and Fig. 8 shows the Sharpe ratio of the six layer architecture followed by the simple strategy. Here, we compute the Sharpe ratio simply by dividing the returns by the standard deviation of the returns, and annualizing it. In contrast to usual practice, no risk-free interest rate has been subtracted from the returns. For more details on the Sharpe ratio, as well as its explicit optimization with variable position sizes, see Choey & Weigend (1997). Furthermore, no transaction costs have been applied. The key point here is the comparison between the new and the standard architecture.

Using the forecast model as part of a decision model, especially in a trading application, it is important to interpret the major input influences at a given day. The input-output sensitivities can vary over time in a nonlinear model, depending on the current operating point. Fig. 9 shows as an example the sensitivities for the 20 days leading up to December 17, 1992. The rightmost column represents the values for the day of the evaluation. Variations over time indicate changes in the importance of the input indicators.

Summarizing the results, the six layer architecture stabilizes the learning process. Given the simple sign-based strategy of translating the forecast into a trading signal, this behavior carries over to the decision process.

5 Discussion and implications

This paper introduced a new architecture for time series prediction. The most important result is that this six layer architecture stabilizes learning in the sense that the network does not overfit on a learning time scale where a standard network has already overfitted strongly. This is mainly caused by the richer error evaluations in the interaction layer. We showed that this can be understood as a consequence of the penalty function effect of the interaction layer. Instead of this approach, we included features of the dynamics in the

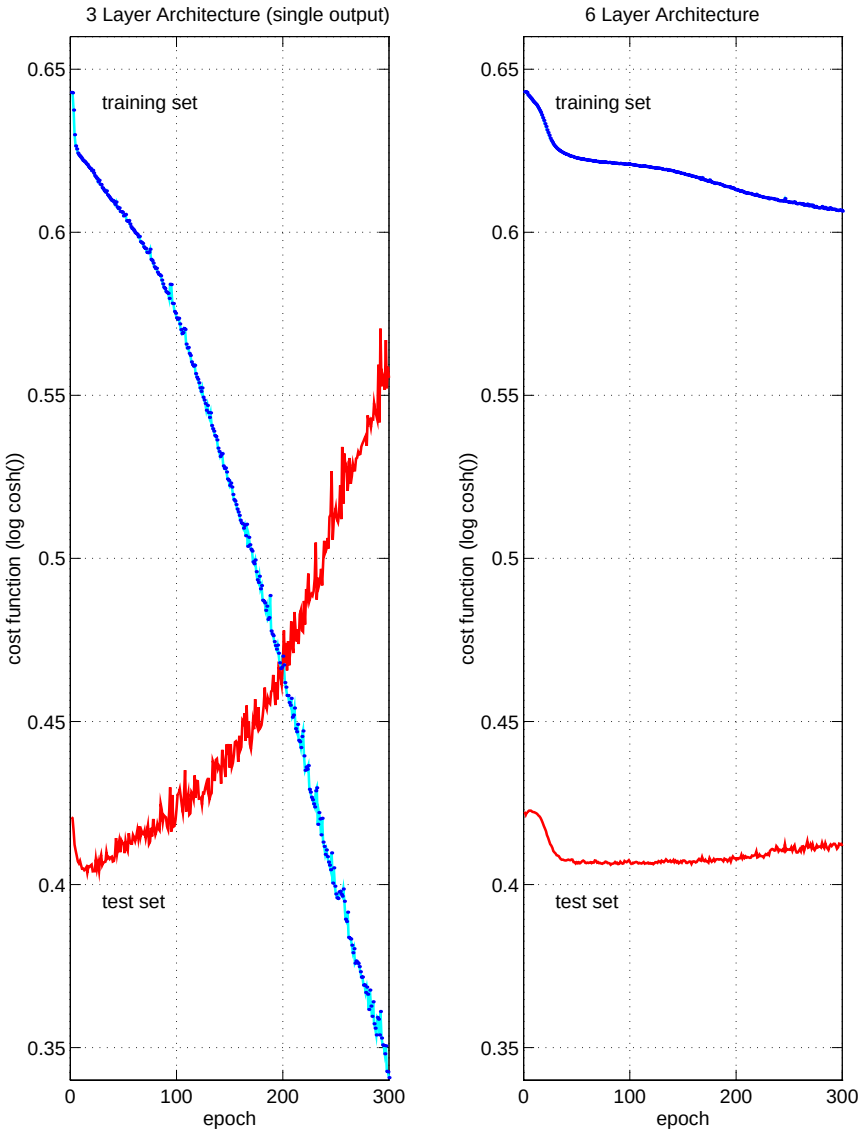

Figure 4: Learning curves for the DAX example. The left panel describes a standard network with a single output unit; the right panel gives the training and testing for the six-layer architecture. Each panel has the performance for both the training and test set. Note the clear overfitting on the network without interaction layer.

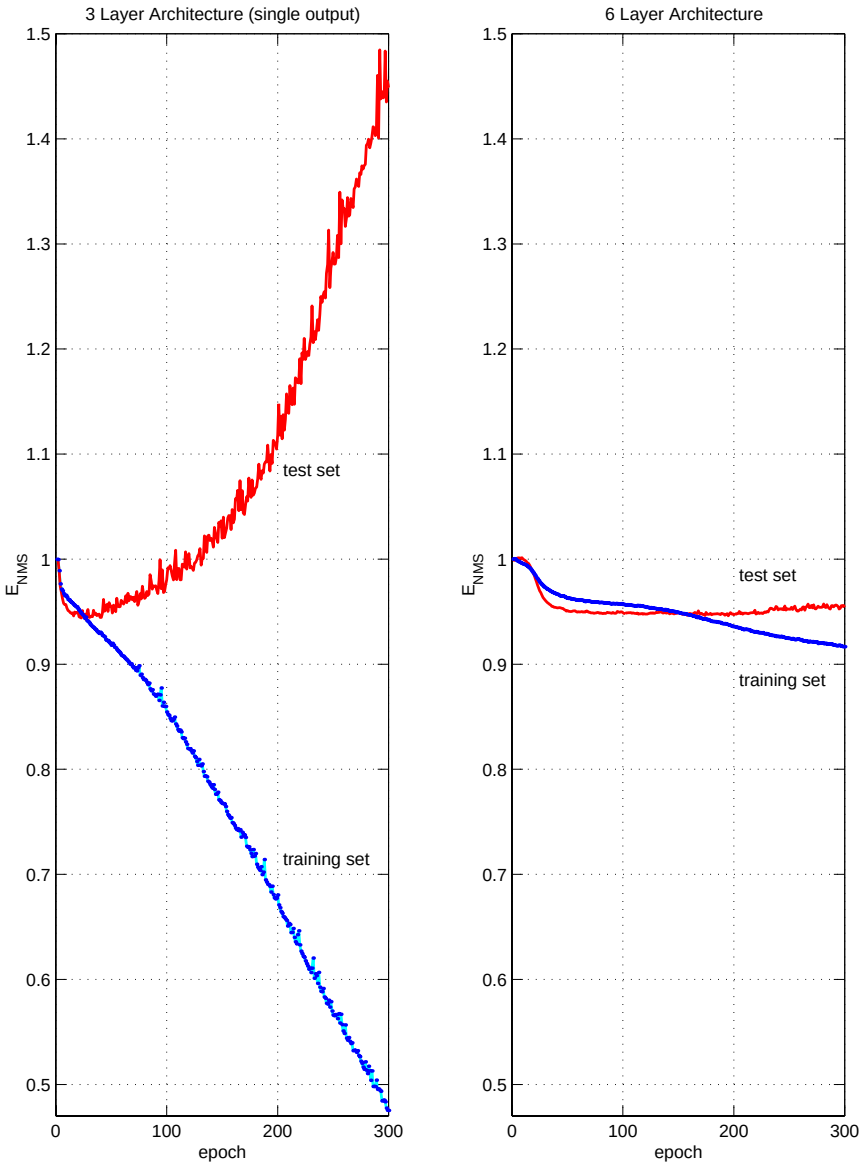

Figure 5: Normalized mean squared error as a function of the training time. The picture shows the stabilizing effect of the six layer architecture on the right, in comparison to a single output on the left.

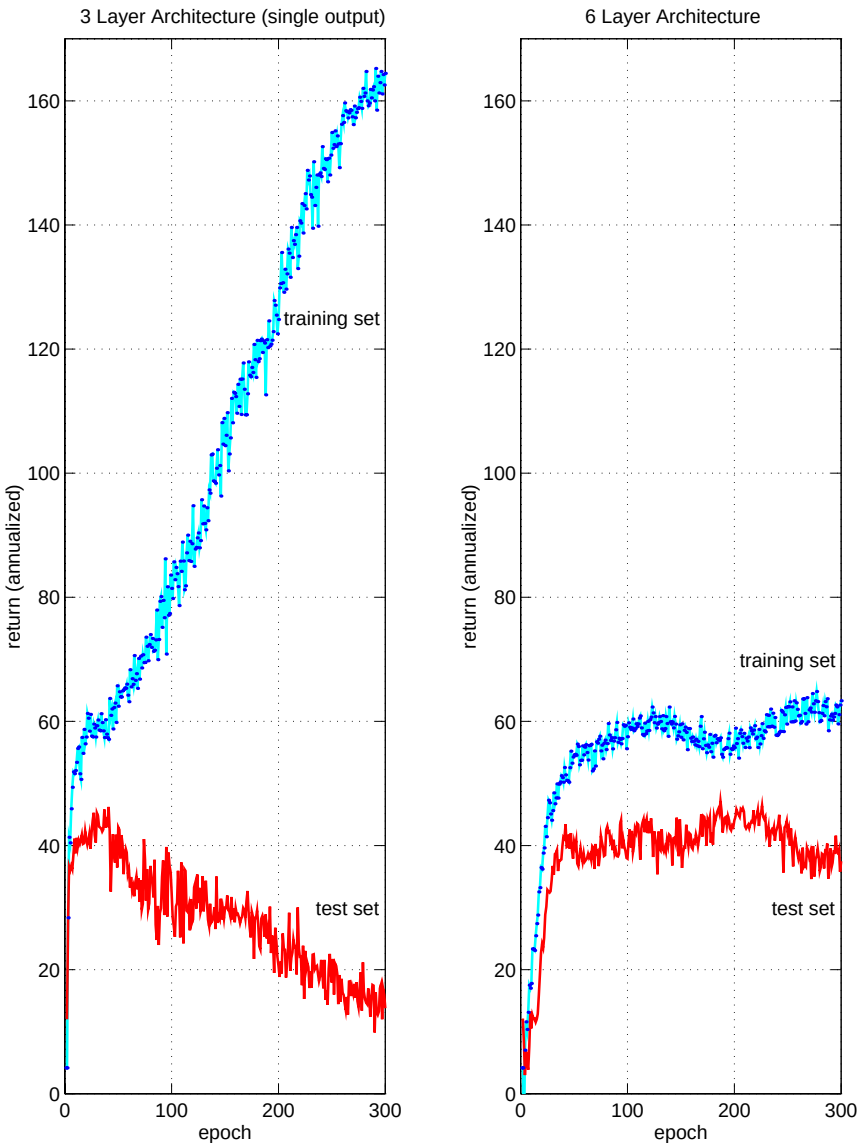

Figure 6: Annualized return as a function of the training time. Whereas the network with a single output (left) overfits dramatically, there is no strong deterioration of the out-of-sample performance for the six layer architecture on the right. (No transaction costs are taken into account.)

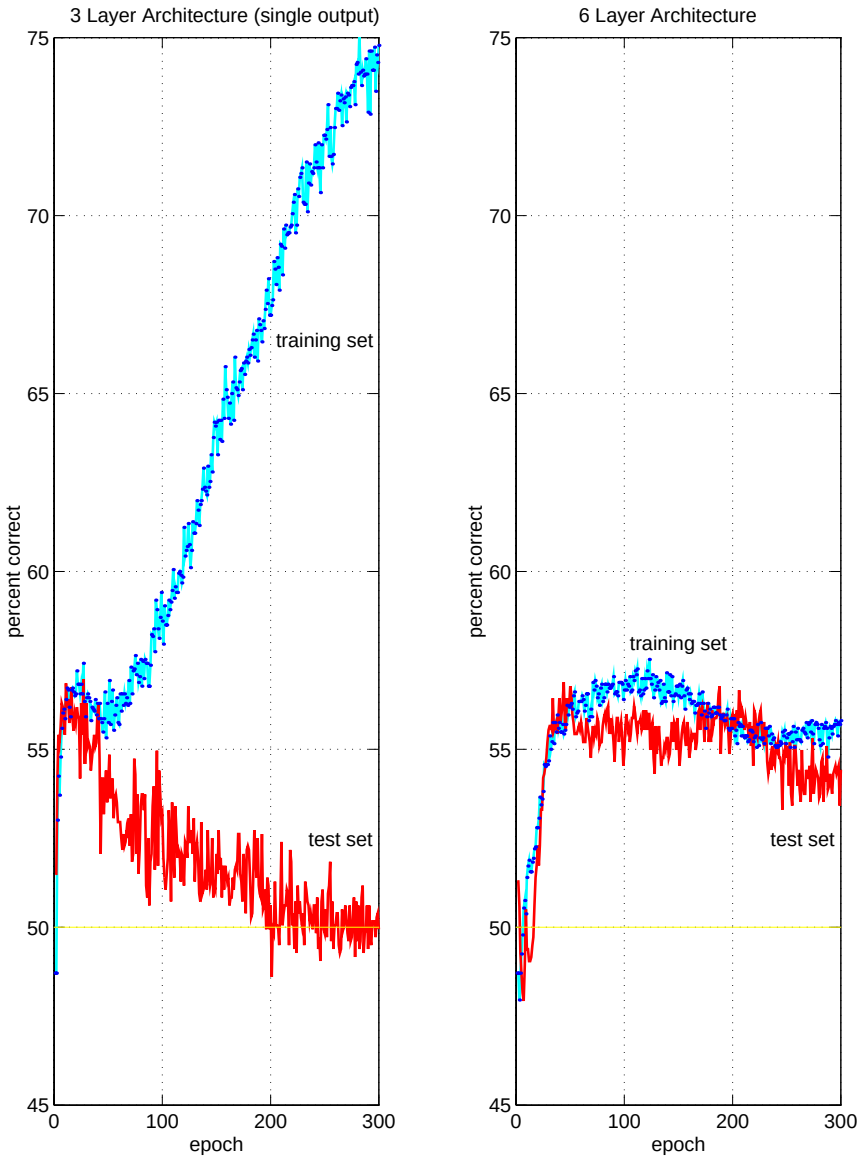

Figure 7: Probability of correct sign ("hit rate") as a function of the training time. Note the close proximity of training and test set values for the six layer architecture on the right.

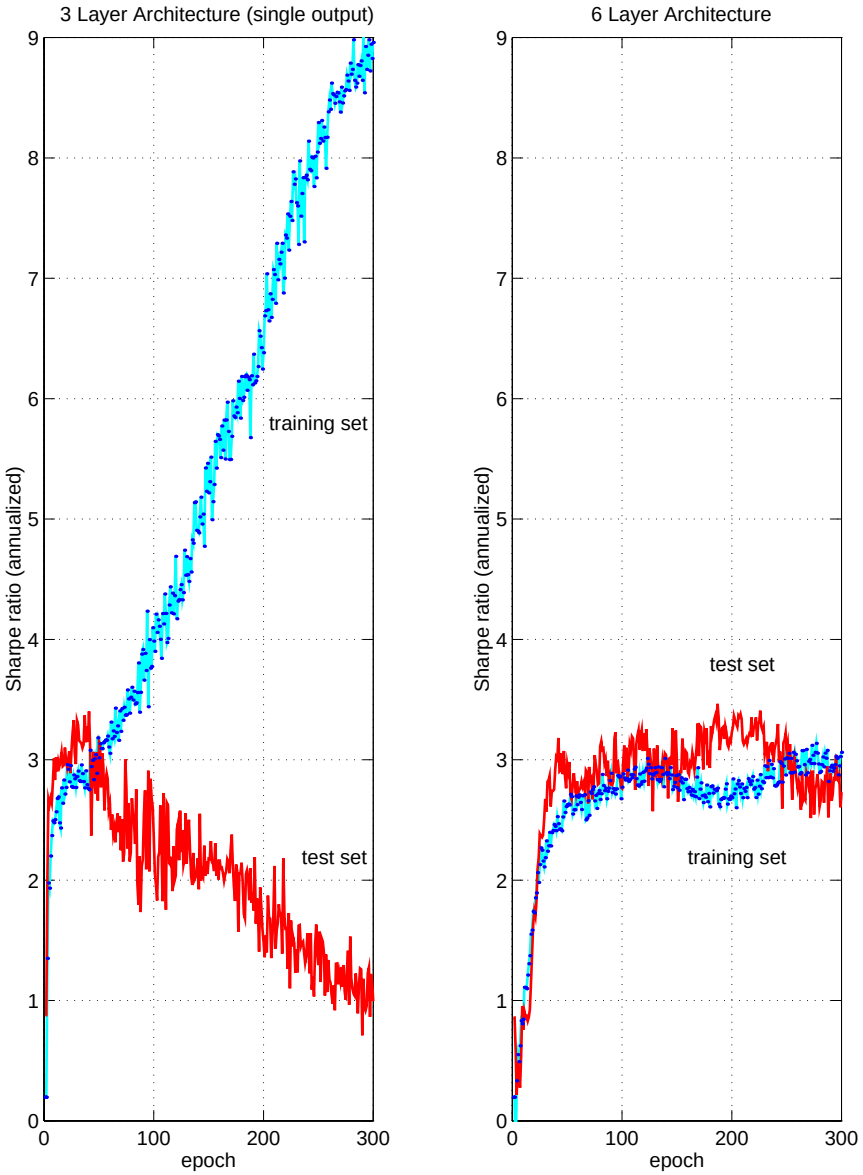

Figure 8: Sharpe ratio as a function of the training time. This network was not optimized on the risk-adjusted performance measure but only evaluated on the annualized return divided by the standard deviation of the returns. (No risk-free interest rate is taken into account.)

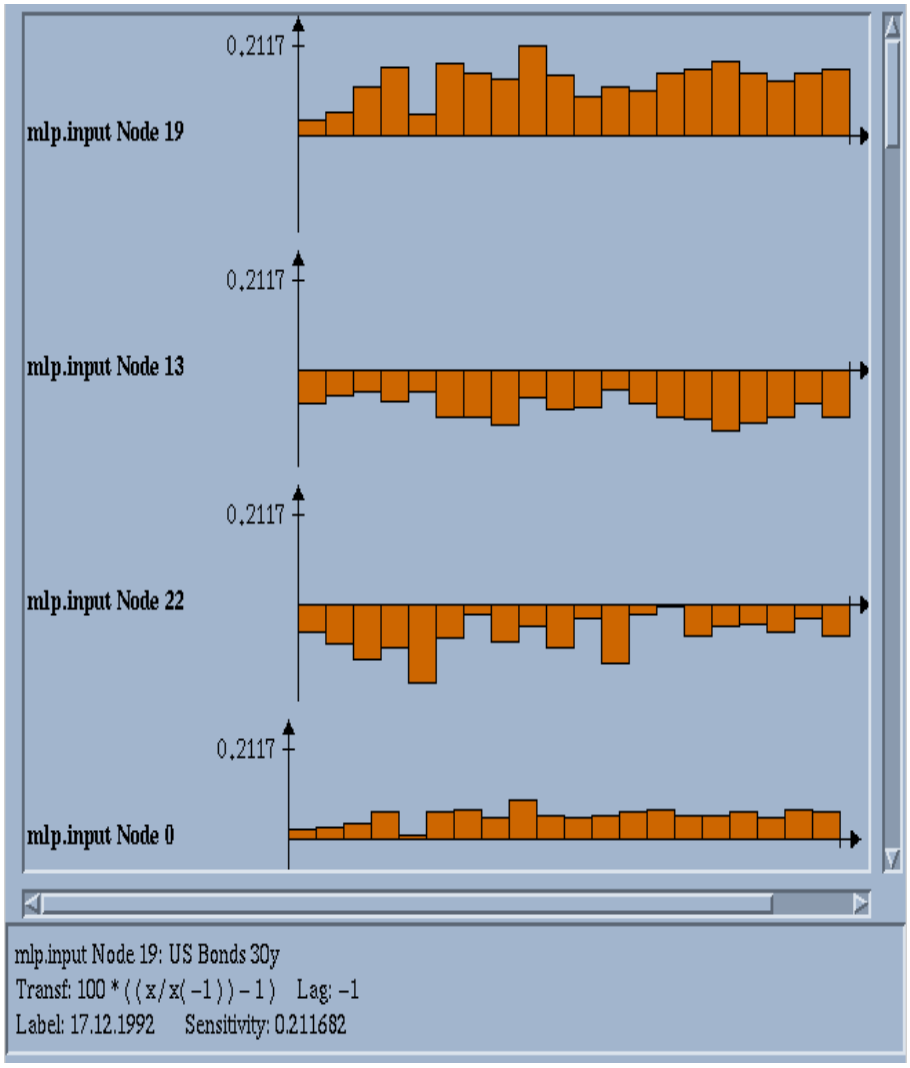

Figure 9: Sensitivity analysis from SENN (Software Environment for Neural Networks). The partial derivative of the final output with respect to each input is shown for the 20 days leading up to the date shown (December 17, 1992). The ranking of the input importance is, starting with the most important one: (1) U.S. 30-year bond (2) Dow Jones Industrial Average, (3) Gold price, and (4) DAX price.

network architecture. This type of penalty is not a bias towards linear models or any structural feature unrelated to the task—the central point is to formulate the time series problem in terms of dynamics variable on several time scales.

From the viewpoint of early stopping, one may argue that a simple reduction of the learning rate would also stabilize the learning behavior. Beside the facts (1) that overfitting often starts already right in the first epoch, and (2) that there is a well-known bias towards a linear solutions,[f] a smaller learning rate would change the global stochastic search to a local optimization which may end in the nearest local minimum. On the other hand, if the local learning does not stop in the nearest local minimum, it has a bias towards maximum likelihood solutions instead of Bayesian solutions. In general, it is important to exploit noise as a central part of the training procedure. Furthermore, we introduced another idea for the input side, independent of the interaction layer on the output side. The new preprocessing layer on the input side consists of adaptive weights on the diagonal and zeros everywhere else (i.e., each input is connected to one tanh preprocessing hidden unit). The purpose is to allow for an adaptive outlier suppression for each input individually.

For the interaction layer, the penalty effect vanishes for noise-free data as well as for extreme overfitting on noisy data when the network completely memorizes the data. This paper does not focus on this situation; many approaches including additive noise on the inputs (Weigend *et al.*, 1991) and weight pruning methods (Zimmermann, 1994) have been proposed to curb overfitting. We did not want to clutter the discussion and obscure the results of the new architecture by also including methods against overfitting here. We would like to note, however, that the additional topology optimization by pruning methods rests on an evaluation of the error signals flowing through the network. We thus expect these techniques to also profit from the extended error signal created by the new six layer architecture proposed in this paper.

Acknowledgments. We thank Mark Choey for his kind help in the final production of this paper during his time with the Information Systems Department of NYU's Leonard N. Stern School of Business. The simulations were performed with the Software Environment for Neural Networks (SENN) by Siemens Nixdorf. This work was partially supported by an award (ECS-9309786) from the National Science Foundation to the first author.

[f] LeBaron & Weigend (1998) give an empirical evaluation on financial data.

References

Bates, J. M. and C. W. J. Granger. 1969. The combination of forecasts. *Operations Research Quarterly* **20**, 451–468.

Caruana, R. A. 1994. Multitask connectionist learning. In *Proceedings of the 1993 Connectionist Models Summer School*, M. C. Mozer, P. Smolensky, D. S. Touretzky, J. L. Elman and A. S. Weigend (eds), pp. 372–379, Hillsdale, NJ. Lawrence Erlbaum Associates.

Caruana, R. 1997. Multitask learning. *Machine Learning* **28**, 41–75.

Choey, M. and A. S. Weigend. 1997. Nonlinear trading models through Sharpe Ratio maximization. In *Decision Technologies for Financial Engineering (Proceedings of the Fourth International Conference on Neural Networks in the Capital Markets, NNCM-96)*, A. S. Weigend, Y. S. Abu-Mostafa and A.-P. N. Refenes (eds), pp. 3–22, Singapore. World Scientific.

Elton, E. J. and M. J. Gruber. 1995. *Modern Portfolio Theory and Investment Analysis*. John Wiley and Sons, New York, fifth edition.

Gershenfeld, N. A. 1989. An experimentalist's introduction to the observation of dynamical systems. In *Directions in Chaos*, H. B. Lin (ed), vol. 2, pp. 310–384. World Scientific, Singapore.

Granger, C. W. J. 1989. Combining forecasts–Twenty years later. *Journal of Forecasting* **8**, 167–173.

LeBaron, B. and A. S. Weigend. 1998. A bootstrap evaluation of the effect of data splitting on financial time series. *IEEE Transactions on Neural Networks* **9**, forthcoming.

Shi, S. and A. S. Weigend. 1997. Taking time seriously: Hidden Markov experts applied to financial engineering. In *Proceedings of the IEEE/IAFE 1997 Conference on Computational Intelligence for Financial Engineering (CIFEr)*, pp. 244–252, Piscataway, NJ. IEEE Service Center.

Timmer, J. and A. S. Weigend. 1997. Modeling volatility using state space models. *International Journal of Neural Systems* **8**, forthcoming.

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

Zimmermann, H. G. 1994. Neuronale Netze als Entscheidungskalkül. In *Neuronale Netze in der Ökonomie (In German)*, H. Rehkugler and H. G. Zimmermann (eds), pp. 1–87, München. Vahlen Verlag.

Zimmermann, H. G., R. Neuneier, H. Dichtl and S. Siekmann. 1996. Modeling the German stock index DAX with neuro-fuzzy. In *EUFIT'96, Fourth European Congress on Intelligent Techniques and Soft Computing*, H.-J. Zimmermann (ed), vol. 3, pp. 2187–2190.