Forecasting in Big Data Environments: an Adaptable and Automated Shrinkage Estimation of Neural Networks (AAShNet)

Ali Habibnia[*] *Esfandiar Maasoumi*[†]

ABSTRACT

This paper considers improved forecasting in possibly nonlinear dynamic settings, with high-dimension predictors ("big data" environments). To overcome the curse of dimensionality and manage data and model complexity, we examine shrinkage estimation of a back-propagation algorithm of a neural net with skip-layer connections. We expressly include both linear and nonlinear components. This is a high-dimensional learning approach including both sparsity L_1 and smoothness L_2 penalties, allowing high-dimensionality and nonlinearity to be accommodated in one step. This approach selects significant predictors as well as the topology of the neural network. We estimate optimal values of shrinkage hyperparameters by incorporating a gradient-based optimization technique resulting in robust predictions with improved reproducibility. The latter has been an issue in some approaches. This is statistically interpretable and unravels some network structure, commonly left to a black box. An additional advantage is that the nonlinear part tends to get pruned if the underlying process is linear. In an application to forecasting equity returns, the proposed approach captures nonlinear dynamics between equities to enhance forecast performance. It offers an appreciable improvement over current univariate and multivariate models by RMSE and actual portfolio performance.

Key Words: Big Data Eonometrics, High-dimensional Nonlinear Time Series, Forecasting, Nonlinear Shrinkage Estimation, Gradient-based Hyperparameter Optimization
JEL classification: C45, C51, C52, C53, C61.

[*]Department of Economics, Virginia Tech. Email: habibnia@vt.edu
[†]Department of Economics, Emory University.

I. Introduction

An important step in designing modern predictive models is to cope with high-dimensional data, presenting large numbers of (cor)related variables and complex properties. "Big data" is both an increase in the number of samples collected over time, and an increase in the number of potential explanatory variables and predictors. When dimension grows, the specificities of high-dimensional spaces and data must then be taken into account in the design of predictive models. While this is valid in general, its importance is heightened when using nonlinear tools such as artificial neural networks. Most nonlinear models involve more parameters than the dimension of the data space which may result in a lack of identifiability, lead to instability, and overfitting (Huber (2011);Cherkassky et al. (1994); Moody (1991)). Selection of significant predictors, and model complexity are the key tasks of designing accurate predictive models in data-rich environments.

Feature extraction and feature selection are broadly the two main approaches to dimensionality reduction. Extraction transforms the original features into a lower dimensional space preserving all its fundamentals. Feature selection methods select a small subset of the original features without a transformation. Extraction methods include principal component analysis - Pearson (1901); Eckart and Young (1936); factor analysis - Spearman (1904); canonical correlations analysis - Hotelling (1936), and several others[1].). Feature selection is accomplished by such methods as Ridge - Hoerl and Kennard (1970); LASSO - Tibshirani (1996) and Elastic Net - Zou and Hastie (2005)).

In this work, our main focus is on feature selection techniques. We apply shrinkage approaches (usually referred to as regularization in machine learning literature). We embed feature selection in the backpropagation algorithm as part of its overall operation. Accordingly, we extend our loss function to include L_1 norm for the weights of the dense network, and L_2 norm for the weights in the skip-layer. The dense network corresponds to a multilayer neural network, whereas the skip-layer denotes the direct connection from each of the input variables to each of the output variables, which is similar to a linear regression model.

[1] Cunningham and Ghahramani (2015) surveyed the literature on linear dimensionality reduction in their work.

Shrinkage is an implicitly embedded feature selection. It is an example of model selection since only a subset of variables contributes to the final predictor. It has frequently been observed that L_1 shrinkage produces many zero parameters, leading to some features being dropped and a sparse model. Only those parameters whose impact on the empirical risk is considerable appear in the fitted model Ng (2004). Shrinkage is a proper means of controlling complexity in the nonlinear component. From an optimization point of view we have a neural network learned/estimated by LASSO. This prevents hidden units from getting stuck near zero and/or exploding weights.

Simultaneously, we employ the L_2 shrinkage on the skip-layer connections (linear part of the model), in order to penalize groups of parameters, and encourage the sum of the squares of the parameters to be small. Therefore we will not drop specific features from linear component, making it possible to interpret the marginal impact of predictors on the target variable. It is worth mentioning that the linear part of the model can be interpreted as a Ridge regression.

There are other benefits to shrinkage/regularization. Empirically, penalizing the magnitude of network parameters is also a way to reduce overfitting and to increase prediction accuracy Ng (2004). This is especially true in the state-of-art models, such as deep learning models with large number of parameters. Our proposed algorithm combines the neural network's advantage of describing the nonlinear process with the superior accuracy of feature selection that is provided by a penalized loss function that combines L_1 and L_2 norms.

Many studies have suggested neural networks as a promising alternative to linear regression models. Empirical evidence on out-of-sample forecasting performance is, however, mixed. It is challenging to determine linear or nonlinear components. Linearity tests do often suggest that real world series are rarely purely linear or nonlinear.

We consider the possibility that the series (y_t) contain both a linear component, $(\mathcal{L}_t)$, and a nonlinear component $(\mathcal{N}_t)$.

$$y_t = \mathcal{L}_t + \mathcal{N}_t \tag{I.1}$$

Neural network alone is not best suited to handle both linear and nonlinear components, especially when the linear component is superior to the nonlinear component.

Two different approaches to model and forecast series with both linear and nonlinear patterns are available. The first approach is a two step methodology to combine linear time series models and neural network models. In this approach, the first step residuals are obtained from the fitted linear model $\hat{e}_t = y_t - \hat{\mathcal{L}}_t$. In the second step a nonlinear model (e.g., GARCH, neural nets) is trained on the residuals of the first step. In principle, this "hybrid" two step approach can provide superior predictions when both the linear and neural network model are well specified. In practice, however, two types of model specification errors are introduced without an ability to assess their mutual impact.

The alternative approach that we are proposing in this paper models both linear and nonlinear components adoptively. It is based on a neural network with skip-layer connections including both linear and nonlinear structures.

The rest of the paper is organized as follows. Section II provides the basic framework of the proposed model. In Section III we investigate proper estimation of shrinkage hyperparameters and introduce gradient-based techniques based on reverse-mode automatic differentiation (RMAD) to accomplish this. Section IV presents an application to US financial returns. Section V contains some concluding remarks.

II. The Model

In this study, we examine a feedforward neural network with one hidden layer, known as a dense network. Neural network models can be seen as generalizations of linear models, when one allows direct connections from the input variables to the output layer with a linear transfer function[2], that we refer to as the skip-layer. The model is expressed as

$$y_t = \Phi(\boldsymbol{x}; \boldsymbol{w}) = \sum_{i \to k} x_{it} w_{ik} + \sum_{j \to k} \phi_j \left(\sum_{i \to j} x_{it} w_{ij} \right) w_{jk} + \varepsilon_t, \qquad \text{(II.1)}$$

[2] Using linear function for the output unit activation function (in conjunction with nonlinear activations amongst the hidden units) allows the network to perform a powerful form of nonlinear regression. So, the network can predict continuous target values using a linear combination of signals that arise from one layer of nonlinear transformations of the input.

where Φ describes the network by a vector function. We associate subscript i with the input layer, subscript j with the hidden layer, and subscript k with the output layer. $x_{it} = (x_{1t}, x_{2t}, ..., x_{mn})$ is the value of the ith input node, which can be a constant input representing biases, a matrix of lagged values of y_t and some exogenous variables. $\phi_j(.)$ and J are activation functions and number of neurons used at the hidden layer. A single-hidden-layer neural network with skip-layer connections is shown in Figure II.1. A network with only one hidden layer and skip-layer connections has three sets of weights: those for direct connections between the inputs and the output (w_{ik}), those connecting the inputs to the hidden layer (w_{ij}), and those connecting the output of the hidden layer to the final output layer(w_{jk}).

First term in Eq.(II.1) represents a linear regression term. The second term, denoting the dense network of the two layers, hidden and output, is usually referred to as a multilayer perceptron in the literature. It has been shown to be able to perform well with nonlinear complex data. A greater capacity of the dense network, compared to the skip-layer, is realized by stacking two layers, enabling it to model more complex data. A differentiable nonlinear activation function ϕ is used in the hidden units. ε_t is a random disturbance term which captures all other factors influencing y than the x. A linear component term moves the model in the linear direction. This aids statistical interpretation and unravels the structure behind the network, otherwise left to a black box. This simultaneous approach has the advantage, when we apply shrinkage techniques to estimate network parameters for an essentially linear process, of pruning the hidden neurons.

Estimation of network elementary parameters based on prediction error minimisation is known as training/learning. The most common cost/risk function is the mean squared prediction error (MSE), $E = \frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2$. Given target values y_t and network estimated outputs $\hat{y}_t$ error functions are obtained for each parameter set, followed by tuning of the parameters.

The error surface becomes increasingly complicated with the number of input variables and network parameters. It is common to employ the conventional feed-forward neural network, trained with the popular and revolutionary gradient-descent-type algorithm known as backpropagation. The backpropagation algorithm was first introduced by

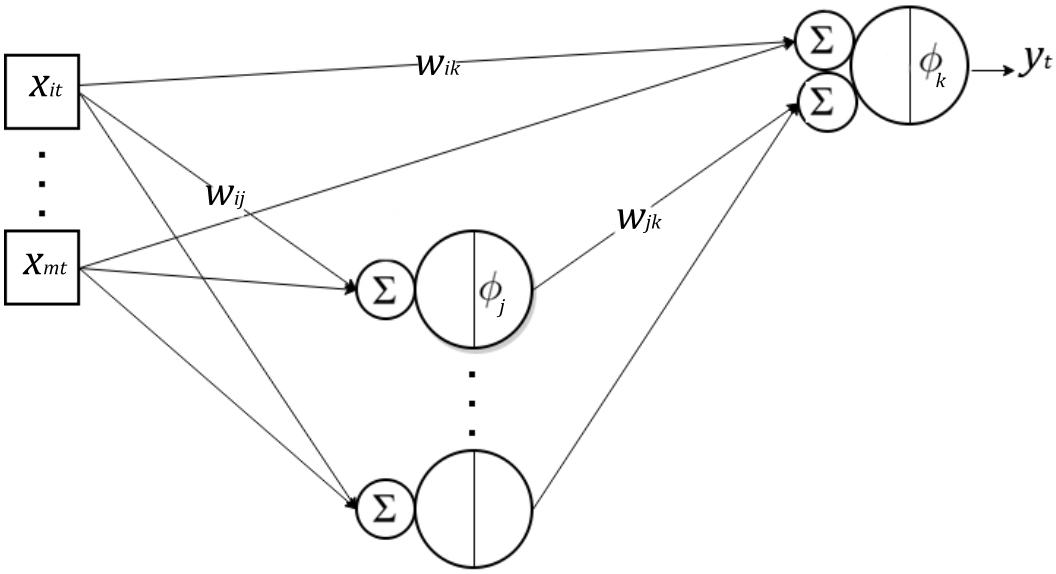

Figure II.1. A single-hidden-layer neural network with skip-layer connections

Bryson et al. (1979) and popularized in the field of artificial neural network research by Werbos (1988) and Rumelhart et al. (1986). Error function's sensitivity to network parameters is assessed via Gradient Descent optimization. Gradient is normally defined as the first order derivative of the error function with respect to each of the model parameters. Working out the gradients can be performed in a completely mechanical way known as Automatic Differentiation Baydin et al. (2017). AD employs the Jacobian matrix of gradients for each parameter w_i to identify directions that decrease the height of the error surface (see Appendix). In fact backpropagation is only a specific case of reverse-mode AD that is applied to an objective function errors as functions of model parameters. The weight adjustment is given by

$$w^{new} = w^{old} - \eta \frac{\partial E(\boldsymbol{w})}{\partial w} \tag{II.2}$$

Where the constant η is the learning rate (step size) for updating elementary parameters, its value falls between zero and one. By iteratively repeating this mechanism, the network can be trained in a way that converges to the optima. The set of new elementary parameters are repeatedly presented to the network until the error value is minimized. Around the optimum point, all the elements of the gradient would be very small, leading

to tiny changes in new parameters.

We add the L_1 and L_2 penalties in training our modelto the loss function $\widetilde{E}(.)$, the original MSE. The following optimization problem is used for training:

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}}\ \widetilde{E}(\boldsymbol{w}|\lambda, X) = \underset{\boldsymbol{w}}{\operatorname{argmin}}\ E(\boldsymbol{w}|\lambda, X)\ +\ \Omega(\boldsymbol{w}, \boldsymbol{\lambda}) \tag{II.3}$$

where the regularization term $\Omega(\boldsymbol{w}, \boldsymbol{\lambda})$ is a combination of the L1 norm and the L2 norm of the parameter vector. λ sets the impact of shrinkage on the loss, with larger values resulting in more penalization. Using the regularized objective causes the training procedure to be inclined to smaller parameter values; unless larger parameters considerably improve the original error value (MSE). Assuming a fixed λ, to learn w^*, we only need to include the derivative of $\Omega(\boldsymbol{w}, \boldsymbol{\lambda})$ in our derivatives:

$$\begin{cases} \Delta = \frac{\partial E(\boldsymbol{w})}{\partial w} + \frac{\partial \Omega(\boldsymbol{w},\lambda)}{\partial w} \\ w^{new} = w^{old}\ -\ \eta\Delta \end{cases} \tag{II.4}$$

Where Δ is the gradient of the regularized loss function. $\lambda\ >\ 0$ is proportional to complexity of the model but is not a parameter that appears in the model. It is a hyperparameter. In the next section, we explain the impact of hyperparameters and elaborate on our procedure for tuning them.

We employ L_1 and L_2 shrinkage on the parameters of the dense network and skip-layer, respectively; as is depicted by following optimization problem:

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}}\ E(\boldsymbol{w}|\lambda, X)\ +\ \frac{\lambda_2}{2}\sum_{i\to k} w_{ik}^2\ +\ \lambda_1(\sum_{i\to j}|w_{ij}| + \sum_{j\to k}|w_{jk}|) \tag{II.5}$$

which can be realized by iteratively adjusting the parameters using the updating rules below

$$\begin{cases} w_{ik}^{new} = w_{ik}^{old} - \eta\,(\frac{\partial E(w|\lambda,X)}{\partial w_{ik}} + \lambda_2 w_{ik}^{old}) \\ w_{ij}^{new} = w_{ij}^{old} - \eta\,(\frac{\partial E(w|\lambda,X)}{\partial w_{ij}} + \lambda_1 sgn(w_{ij}^{old})) \\ w_{jk}^{new} = w_{jk}^{old} - \eta\,(\frac{\partial E(w|\lambda,X)}{\partial w_{jk}} + \lambda_1 sgn(w_{jk}^{old})) \end{cases} \tag{II.6}$$

Where λ_1 and λ_2 are non-negative values known as shrinkage hyperparameters. L_1

sparsity norm and L_2 smoothing norm are two closely related regularizers that can be used to impose a penalty on the complexity of the model that is to be learned. Shrinkage estimation of the model can be seen as an implementation of Occam's razor, introducing a controllable trade-off between fitting data and model complexity, enabling us to have models of less complexity with adequate generalization capability. Regularization in neural networks limits the magnitude of network parameters by adding a penalty for weights to the model error function. In this study, L_2 shrinkage penalizes parameters in skip-layer connections by adding sum of their squared values to the error term. L_1 shrinkage penalizes parameters in the dense network to encourage the topology of the learned network to be sparse. The relative importance of the compromise between finding small weights and minimizing the original risk function depends on the size of λ.

To use L_2 shrinkage, we add a $\lambda_2 w$ term to the gradient as the derivative of w^2 is $2w$. L_2 shrinkage works with all forms of learning algorithms, but does not provide implicit feature selection. The derivative of the absolute value of w is $w/|w|$, however L_1 norm is not differentiable at zero and hence poses a problem for gradient-based methods.

The problem can be solved using the exact gradient, which is discontinuous at zero. We can also solve the problem by the smooth approximation approach which will allow us to use gradient descent. To smooth out the L_1 norm using an approximation, we use $\sqrt{w^2 + \epsilon}$ in place of $|w|$, where ϵ is a smoothing parameter which can also be interpreted as a sort of sparsity parameter. When ϵ is large compared to w, the expression $w + \epsilon$ is dominated by ϵ and taking the squared root yields approximately $\sqrt{\epsilon}$. Lee et al. (2006)

III. Gradient-based Hyperparameter Optimization

The major drawback of shrinkage is that it introduces additional hyperparameters. In practice we have two set of parameters: model elementary parameters (network weights and biases), and learning algorithm hyperparameters (magnitude of L_1 and L_2 penalties, and learning rate). We would ideally like to determine these hyperparameters to get optimal generalization[3]. As opposed to elementary parameters, these hyperparamters cannot

[3]Generalization means building a model on one set of training data and hope that it makes effective predictions on a different set of test data.

be directly trained by the data. Whereas the elementary parameters specify how to transform the input data into the desired output, the hyperparameters define how our model and algorithm are actually structured.

The performance and robustness of neural networks relies to a large extent on hyperparameters. Tuning these hyperparameters not only makes the investigation of methods difficult, but also hinders reproducibility (Bergstra et al. (2011b)). Transparent tuning of hyperparameters can be part of an Hyperparameter Optimization (HPO), as an outer loop in training procedures.

The de-facto naïve approach of searching through combinations of potential values of hypergradients and choosing the one that performed the best (a.k.a. grid search) is very time-consuming and becomes quickly infeasible as the dimension of hyperparameter space grows. In many practical applications manually searching the space of hyperparameter settings is tedious and tends to lead to unsatisfactory outcomes. Bergstra and Bengio (2012) show empirically and theoretically that random search more efficient than grid search. Statistical techniques such as cross-validation Wahba (1990), bootstrapping Efron and Tibshirani (1994), and Bayesian methods MacKay (1992) can also assist in determining hyperparameters.

HPO must be guided by some performance metric, typically measured by cross-validation (CV) on the training set, or evaluation on a held-out validation set. The rationale behind CV is to split the data into the training samples used for learning the algorithm, and the validation samples (one or several folds) for estimating the risk of each algorithm and for evaluation of its performance. CV consists of averaging several hold-out estimators (folds) of the risk corresponding to different splits of the data, and selecting the algorithm with the smallest estimated risk. Within each fold, hyperparameters are fixed and we only estimate model elementary parameters. The validation samples play the role of new unseen data as long as the data are i.i.d.[4] For a general description of the CV see Geisser (1975), and Arlot and Celisse (2010) for a comprehensive review on cross-validation procedures and their applications in different algorithms and frameworks. Several studies such as Rivals and Personnaz (1999) show cases in which CV performance is less than satisfactory.

[4] This assumption can be relaxed. see: Chu and Marron (1991).

Recently, automated approaches for estimation of hyperparameters have been proposed which can provide substantial improvements and transparency. Although one may also "hyperparameterize" certain discrete choices in design of the model (e.g. number of hidden units), we focus only on the continuous hyperparameters in this work. There are a number of gradient-free automated optimization methods (Hutter et al. (2011); Bergstra et al. (2011a); Bergstra et al. (2013); Snoek et al. (2012)), all of which rely on multiple complete training runs with varied fixed hyperparameters. Hyperparameters are chosen to optimize the validation loss after complete training of the model parameters.

Gradient-based HPO approaches, proposed by Larsen et al. (1996) and Andersen et al. (1997), emerged in the 1990s. We can distinguish two main approaches of gradient-based optimization: Implicit differentiation and iterative differentiation.

Implicit differentiation, first proposed by Larsen et al. (1996), computes the derivative of the cost L_{valid} with respect to λ based on the observation that, under some regularity conditions, the implicit function theorem can be applied in order to calculate the gradients of the loss function. In particular, the cost function is assumed to smooth and converge to local minima. The inner optimization $w(\lambda) \in \text{argmin}_w L_{train}$ can be characterized by the implicit equation $\nabla_w L_{train} = 0$. Bengio (2000) derived the gradients for unconstrained cost function and applied the algorithm to $L2$ shrinkage for linear regression. The method has also been used to find kernel parameters of Support Vector Machines Keerthi et al. (2007). Pedregosa (2016) proposes *HOAG* which uses inexact gradients, allowing the gradient with respect to hyperparameters to be computed approximately.

In iterative differentiation, first proposed by Domke (2012), the gradient for hyperparameters are calculated by differentiating each iteration of the inner optimization loop and using the chain rule to aggregate the results. However, the problem with this reverse-mode approach is that one must retain the entire history of elementary parameter updates, making a naïve implementation impractical due to memory constraints. Reverse-mode differentiation requires intermediate variables to be maintained in the memory for the reverse pass and evaluation of validation loss needs hundreds or thousands of inner optimization iterations. Maclaurin et al. (2015) later extended this for setting of stochastic gradient descent via reverse mode automatic differentiation of validation loss.The burden of storing

the entire training trajectory $w_1, \cdots, w_T$ is avoided by an algorithm that exactly reverses SGD with momentum to compute gradients with respect to all training parameters, only using a relatively small memory footprint, making a solution feasible for large-scale big data machine learning problems.

We defined the updating rule for elementary parameters as $w_{t+1} = w_t - \eta \nabla L_{train}$ where $L_{train} = \widetilde{E}(w_t|\lambda, X_{train})$ is the regularized loss value on train data. To calculate hypergradients we rely on the unregularized loss function, that is $L_{valid} = E(w_t|\lambda, X_{valid})$, as the actual generalization performance of the model, on unseen data points, does not directly depend on regularizers; otherwise the model with no regularization would be always selected:

$$\begin{aligned} \lambda^* &= \text{argmin}_\lambda L_{valid} \\ \text{s.t.} \quad w(\lambda) &\in \text{argmin}_w L_{train} \end{aligned} \tag{III.1}$$

There are cases where SGD can become very slow. The method of momentum is designed to accelerate learning, especially in the face of high curvature, small but consistent gradients, or noisy gradients Goodfellow et al. (2016). We modify our training (Algorithm 1) to include a velocity variable v storing the momentum by calculating exponentially decaying moving average of past gradients.

Algorithm 1 Stochastic gradient descent with momentum

1: **input:** initial $\mathbf{w}_1$, decays $\boldsymbol{\gamma}$, learning rates $\boldsymbol{\eta}$, loss L_{train}
2: initialize $\mathbf{v}_1 = \mathbf{0}$
3: **for** $t = 1$ **to** T **do**
4: $\quad \mathbf{g}_t = \nabla_{\mathbf{w}} L_{train}$
5: $\quad \mathbf{v}_{t+1} = \gamma_t \mathbf{v}_t - (1 - \gamma_t)\mathbf{g}_t$
6: $\quad \mathbf{w}_{t+1} = \mathbf{w}_t + \eta_t \mathbf{v}_t$
7: **end for**
8: **output** trained parameters $\mathbf{w}_T$

where γ_t is the momentum decay rate. The training procedure starts with elementary parameters velocity $v_1 = 0$ and w_1 and ends with v_T and $w_T = w_{T-1} + \eta_{T-1} v_{T-1}$. Algorithm 2 is then used to calculate the gradients of validation loss with regard to the hyperparameters.

The velocity v_t is needed to reverse the path, otherwise without momentum, g_t and

Algorithm 2 Reverse-mode differentiation of SGD

input: $\mathbf{w}_T$, $\mathbf{v}_T$, $\boldsymbol{\gamma}$, $\boldsymbol{\eta}$, train loss L_{train}, validation loss L_{valid}
initialize $d\mathbf{v} = \mathbf{0}$, $d\boldsymbol{\lambda} = \mathbf{0}$, $d\eta_t = \mathbf{0}$, $d\gamma = \mathbf{0}$
initialize $d\mathbf{w} = \nabla_{\mathbf{w}} L_{valid}$
for $t = T$ **counting down to** 1 **do**
    $d\eta_t = d\mathbf{w}^{\mathsf{T}} \mathbf{v}_t$
    $\mathbf{w}_{t-1} = \mathbf{w}_t - \eta_t \mathbf{v}_t$
    $\mathbf{g}_t = \nabla_{\mathbf{w}} L(\mathbf{w}_t, \boldsymbol{\lambda}, t)$
    $\mathbf{v}_{t-1} = [\mathbf{v}_t + (1 - \gamma_t)\mathbf{g}_t]/\gamma_t$
    $d\mathbf{v} = d\mathbf{v} + \eta_t d\mathbf{w}$
    $d\gamma_t = d\mathbf{v}^{\mathsf{T}}(\mathbf{v}_t + \mathbf{g}_t)$
    $d\mathbf{w} = d\mathbf{w} - (1 - \gamma_t) d\mathbf{v} \nabla_{\mathbf{w}} \nabla_{\mathbf{w}} L_{train}$
    $d\boldsymbol{\lambda} = d\boldsymbol{\lambda} - (1 - \gamma_t) d\mathbf{v} \nabla_{\boldsymbol{\lambda}} \nabla_{\mathbf{w}} L_{train}$
    $d\mathbf{v} = \gamma_t d\mathbf{v}$
end for
output gradient of L_{valid} w.r.t $\boldsymbol{\lambda}$

η_t alone would not be able to recover w_{t-1}. Notice that the loss of information caused by finite precision arithmetic in computers leads to failure of this algorithm. For this reason, we need to store the bits lost in v_t when multiplied by γ_t.

Given this powerful gradient-based mechanism for finding hyperparameters, a natural extension to our model is to introduce a hyperparameter α denoting the contribution of skip-layer and dense-network in producing predictions with higher generalization. That is to say, our model can be reformulated as:

$$y_t = \Phi(\boldsymbol{x}; \boldsymbol{w}) = \alpha \sum_{i \to k} x_{it} w_{ik} + (1 - \alpha) \sum_{j \to k} \phi_j \left(\sum_{i \to j} x_{it} w_{ij} \right) w_{jk} + \varepsilon_t, \qquad \text{(III.2)}$$

where α assumes a value between zero and one. Appreciating that the skip-layer and the dense network have unbalanced effect on the outcome, one can see how this may result in faster convergence of training procedure. More importantly, α can be interpreted as the activation of skip-layer and dense network and can point to linearity or nonlinearity components.

IV. Case Study: Return Prediction

Research into modelling and forecasting financial returns has a long history. Several models are described in Tsay (2005) and Campbell et al. (1996) that attempt to explain re-

turn time series using linear combinations of one or more financial market factors. The most widely studied single factor model is the capital asset pricing model (CAPM) of Sharpe (1964) and Lintner (1965) that relates the expected return of equities to the expected rate of return on a market index (such as the Standard and Poor's 500 Index). The empirical performance of CAPM is poor as it cannot explain the behaviour of asset returns, see Fama and French (2004). This failure is perhaps due to the absence of multiple factors. Arbitrage pricing theory (APT) is a general model proposed by Ross (1976) to account for these deficiencies. APT presents a linear approximate model of expected asset returns based on an unknown number of macroeconomic "factors" or market indices. The relationship between the factors and historical returns is routinely determined linearly.

Return time series present characteristics such as comovement, nonlinearity, non-Gausianity (skewness and heavy tails), volatility clustering and leverage effect. This makes the modelling task very challenging, see Hsieh (1991); Bollerslev et al. (1994); Brooks (1996); Cont (2001).

The data are daily returns of $m = 418$ equities on the S&P 500 index from 03.01.2006 through 28.09.2018, for a total of 3208 observations. The initial sample 03.01.2006 - 28.09.2017 is used for estimation (training), with $T = 2957$ in-sample size. The holdout sample period 01.10.2017 - 28.09.2018 (251 observations) is employed to examine the models' out-of-sample forecasting performance. 1-step (here one day) ahead forecasts of targets ($\hat{y}_{it+1|t}$) are based on a rolling estimation window. Parameter estimates are updated every five steps.

We believe accounting for comovements between financial returns is important in forecasting returns. Consequently, the lags of other equities are included as predictors for any return series. We examine the nonlinear high-dimensional forecasting model described in the prior sections (AAShNet model) as well as several competing models and benchmarks.

We compare our proposed model with a benchmark, the sample mean of $\boldsymbol{y}_t$ over the in-sample window, as the 1-step ahead forecast. This corresponds to assuming the log daily price of follows a random walk (RW) with drift. It is almost equivalent to the "zero forecast" when the in-sample window is large enough. Furthermore, a buy-and-hold (B&H)

strategy in the market portfolio (S&P 500 Index) has been considered as another benchmark. To understand whether allowing nonlinearity improves portfolio performance we examine the AAShNet algorithm (with Ridge and Lasso) optimized by cross-validation.

Since predictability of financial returns has major consequences for financial decision making, the model with minimal forecast error is deemed optimal. However, the model with minimum forecast error does not necessarily guarantee profit maximization, the primary objective of financial decision makers. Armstrong and Collopy (1992), Pesaran and Timmermann (1995, 2000), Granger and Pesaran (2000) and Engle and Colacito (2006) argue that a forecast evaluation criterion should be related to decision making and judge predictability of financial returns in terms of portfolio simulation. More specifically, a trading (portfolio) simulation approach assumes that all competing models are applied with stock market virtual investment decisions, and out-of-sample portfolio performances are used to evaluate the predictability of alternative models.

Consequently, this paper examines both statistical and portfolio performance measures (the out-of-sample RMSE and the portfolio performance during the out-of-sample period). Figure IV.1 illustrates portfolio excess returns for the out-of-sample period for the proposed model (ASShNet) against competing approaches. We randomly selected 50 stocks out of 418 stocks to construct the portfolio. However, the forecast of each selected stock is based on the lags of all 418 equities.

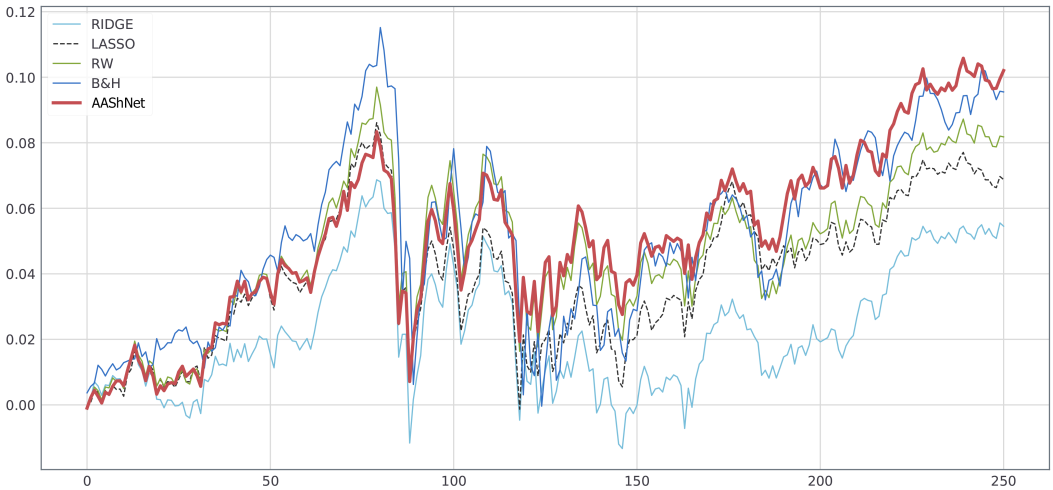

Figure IV.1. Comparison of AAShNet, and the competing models based on the portfolio excess returns in the out-of-sample period.

Consider a passive, equally weighted $(1/M)$ portfolios with short selling. This portfolio is known to be a very stringent benchmark that many optimization models fail to outperform (see DeMiguel et al. (2009). We compute the portfolio's out-of-sample excess returns and volatility as well as the Sharpe ratio. Sharpe ratio measures risk-adjusted returns, a portfolio with a greater Sharpe ratio offers greater returns for the same risk. If a portfolio with lower Sharpe ratio has returned better over a time period than another portfolio with a higher ratio, the risk of losing by investing in the former fund will be higher.

Table I

Portfolio	Return	Sharp Ratio	Ave(RMSE)
AAShNet	10.2%	1.143	0.01558
B&H	9.55%	0.767	-
RW	8.17%	0.770	0.01564
Ridge	5.45%	0.561	0.01613
Lasso	6.87%	0.741	0.01590

The proposed penalized neural net behaves noticeably better in this empirical analysis. Table I provides evidence for out-of-sample forecasting ability of this model vis-à-vis competing approaches in terms of the Sharp ratio. AAShNet also offers an appreciable improvement over linear shrinkage models and benchmarks based on RMSE and actual portfolio performance. In the Ridge and Lasso regressions, the best model is selected by cross-validation. We perform generalized cross-validation, which is an efficient leave-one-out cross-validation.

AAShNet produces higher returns (10.24%) at the end of the out-of-sample period, with a Sharpe ratio of (1.143) that is superior to alternative models. This indicates that significantly improved forecast is obtained by modelling nonlinear dynamics among variables. One should note that Random Walk with drift and AR(1) are special cases of shrinkage models and AAShNet when there is no dependence on other equities.

V. Concluding Remarks

Forecasting with many predictors has received a good deal of attention in recent years. Shrinkage methods are one of the most common approaches for forecasting with many

predictors. Such methods have generally ignored nonlinear dynamic relations among predictors and the target variable.

In this study, we suggested an Adaptable and Automated Shrinkage Estimation of Neural Networks (AAShNet). We explained how skip-layer connections move the model in the right direction when the data contains both linear and nonlinear components. To overcome the curse of dimensionality and to manage model complexity, we penalized the model loss function with L_1 and L_2 norms. Setting the size of shrinkage is still an open question. Recent studies have proposed automated approaches for estimation of algorithm hyperparameters. We employed the gradient-based automated approaches which treat shrinkage hyperparameters in the same manner as the network weights during training, and simultaneously optimize both sets of parameters.

The empirical application to forecasting daily returns of equities in the SP 500 index from 2006 to 2018 provides support for the out-of-sample forecasting ability of AAShNet algorithm vis-à-vis some competing approaches, both in terms of statistical criteria and trading simulation performance. Our empirical results encourage further research toward other possible applications of the proposed model.

VI. Appendix: Automatic Differentiation

There are three main approaches that computer can work out the derivatives: Numerical, Symbolic and Automatic differentiation. Automatic differentiation refers to a family of procedures to automatically calculate exact derivatives of any function, including program subroutines, with time complexity at most a small constant factor of the time complexity of the original function. It is not inherently ill-conditioned and unstable similar to the numerical method and has much less computational complexity. It also does not suffer from expression swell problem of symbolic differentiation.

AD augments the standard computation with calculation of derivatives whose combination through chain rule gives the derivative for overall composition. AD can be applied on evaluation trace of arbitrary program subroutines which can be more than closed-form functions and are in fact capable of incorporating complex control flows which do not di-

rectly alter values. An automatic differentiator takes a code subroutine that computes a function of several independent variables as input and gives as output a code that computes the original function along the gradient of the function with respect to the independent variables. As most of the functions are piece-wise differentiable and control flows not directly interfering with calculations, chain rule can be used repeatedly in such a way that gradients are calculated along intermediate values being computed.

Based on modus operandi of automatic differentiation there can be two implementations of this technique; the *forward mode* and the *reverse mode*. We investigate each method, by applying them on the same trivial function $y = f(x_1, x_2) = x_1 x_2 - cos(x_1)$ at $(x_1, x_2) = (6, 3)$.

$$\begin{aligned} v_1 &= x_1 && = 6 \\ v_2 &= x_2 && = 3 \\ v_3 &= v_1 v_2 && = 6 \times 3 \\ v_4 &= cos(v_1) && = cos(6) \\ v_5 &= v_3 - v_4 && = 18 - 0.96 \\ y &= v_5 && = 17.04 \end{aligned} \tag{VI.1}$$

In forward mode, we build a *Forward Primal Trace* of the values propagating through the function and a corresponding *Forward Tangent Trace*. Eq. VI.1 shows the forward evaluation of primals. Forward primal trace depicts the natural flow of composition. Eq. VI.2 is the corresponding tangent trace for $\dot{y} = \frac{\partial f}{\partial x_1}$, that is the rate of change of the function f with respect to the input x_1. Notice that both traces are evaluated as written, top to bottom. To calculate the derivative with respect to n different parameters, n forward mode differentiations would be needed. This makes the forward-mode very inefficient for deep learning models where the number of parameters may amount to millions.

$$
\begin{aligned}
\dot{v_1} = \dot{x_1} & & = 1 \\
\dot{v_2} = \dot{x_2} & & = 0 \\
\dot{v_3} = \dot{v_1}v_2 + \dot{v_2}v_1 & & = 1 \times 3 + 0 \times 6 \\
\dot{v_4} = \dot{v_1} \times -sin(v_1) & & = 1 \times -sin(6) \\
\dot{v_5} = \dot{v_3} - \dot{v_4} & & = 3 - 0.279 \\
\dot{y} = \dot{v_5} & & = 2.72
\end{aligned}
\tag{VI.2}
$$

The reverse mode works by complementing each intermediate variable v_i with an adjoint $\bar{v_i}$ representing the sensitivity of output y to changes in v_i. In reverse mode the code is executed and the trace is stored in memory at first stage. At second stage, the adjoints are calculated in opposite direction of the execution of the original function. The reverse adjoint trace corresponding to Eq. VI.1 is depicted in VI.3.

$$
\begin{aligned}
\bar{v_5} = \bar{y} & & = 1 \\
\bar{v_4} = \bar{v_5}\frac{\partial v_5}{\partial v_4} & & = \bar{v_5} \times -1 = -1 \\
\bar{v_3} = \bar{v_5}\frac{\partial v_5}{\partial v_3} & & = \bar{v_5} \times 1 = 1 \\
\bar{v_1} = \bar{v_4}\frac{\partial v_4}{\partial v_1} & & = \bar{v_4} \times -sin(v_2) = -0.27 \\
\bar{v_2} = \bar{v_3}\frac{\partial v_3}{\partial v_2} & & = \bar{v_3} \times v_1 = 6 \\
\bar{v_1} = \bar{v_1} + \bar{v_3}\frac{\partial v_3}{\partial v_1} & & = \bar{v_1} + \bar{v_3} \times v_2 = 2.72 \\
\bar{x_1} = \bar{v_1} & & = 2.72 \\
\bar{x_2} = \bar{v_2} & & = 6
\end{aligned}
\tag{VI.3}
$$

For a function $f : \mathbb{R}^n \to \mathbb{R}^m$ whose number of operations to be evaluated is denoted by ops(f), the complexity of calculating the Jacobian by forward and reverse modes are $n \times c \times$ ops(f) and $m \times c \times$ ops(f), respectively, where it is guaranteed that $c <$ 6Griewank and Walther (2008). That is if $n \gg m$, backward-mode is preferable, although it would have increased memory requirements. And forward mode should be used when the number of dependent variables is greater than the number of independent variables.

References

Andersen, L. N., Larsen, J., Hansen, L. K., and Hintz-Madsen, M. (1997). Adaptive regularization of neural classifiers. *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing*, VII:24–33.

Arlot, S. and Celisse, A. (2010). A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4(0):40–79.

Armstrong, J. and Collopy, F. (1992). Error measures for generalizing about forecasting methods: Empirical comparisons. *International Journal of Forecasting*, 8(1):69–80.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2017). Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18(153):1–153.

Bengio, Y. (2000). Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011a). Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, 24:2546–2554.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305.

Bergstra, J., Yamins, D., and Cox, D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *International Conference on Machine Learning*, page 115–123.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011b). Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554.

Bollerslev, T., Engle, R. F., and Nelson, D. B. (1994). Arch models. *Handbook of Econometrics*, Volume IV(Elsevier Science):2961–3031.

Brooks, C. (1996). Testing for non-linearity in daily sterling exchange rates. *Applied Financial Economics*, 6(4):307–317.

Bryson, A. E., Ho, Y.-C., and Siouris, G. M. (1979). Applied optimal control: Optimization, estimation, and control. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(6):366–367.

Campbell, J. Y., Lo, A. W., and MacKinlay, C. A. (1996). *The econometrics of financial markets*. Princeton University Press, United States, 2 edition.

Chen, D. and Hagan, M. T. (1999). Optimal use of regularization and cross-validation in neural network modeling. page 1275–1289, International Joint Conference on Neural Networks.

Cherkassky, V., Friedman, J. H., and Wechsler, H., editors (1994). *From statistics to neural networks: Theory and pattern recognition applications*. Springer-Verlag Berlin and Heidelberg GmbH & Co. K, Berlin.

Chu, C.-K. and Marron, J. S. (1991). Comparison of two bandwidth selectors with dependent errors. *The Annals of Statistics*, 19(4):1906–1918.

Chui, C. K. and Li, X. (1992). Approximation by ridge functions and neural networks with one hidden layer. *Journal of Approximation Theory*, 70(2):131–141.

Cont, R. (2001). Empirical properties of asset returns: Stylized facts and statistical issues. *Quantitative Finance*, 1(2):223–236.

Cunningham, J. P. and Ghahramani, Z. (2015). Linear dimensionality reduction: Survey, insights, and generalizations. *Journal of Machine Learning Research*, 16:2859–2900.

DeMiguel, V., Garlappi, L., and Uppal, R. (2009). Optimal versus naive diversification: How inefficient is the 1/ n portfolio strategy? *Review of Financial Studies*, 22(5):1915–1953.

Domke, J. (2012). Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pages 318–326.

Eckart, C. and Young, G. (1936). The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218.

Efron, B. and Tibshirani, R. J. (1994). *An introduction to the Bootstrap, Vol. 57*. Chapman & Hall/CRC, Boca Raton, FL.

Engle, R. and Colacito, R. (2006). Testing and valuing dynamic correlations for asset allocation. *Journal of Business & Economic Statistics*, 24(2):238–253.

Fama, E. F. and French, K. R. (2004). The capital asset pricing model: Theory and evidence. *Journal of Economic Perspectives*, 18(3):25–46.

Franceschi, L., Donini, M., Frasconi, P., and Pontil, M. (2017). Forward and reverse gradient-based hyperparameter optimization. *arXiv preprint arXiv:1703.01785*.

Geisser, S. (1975). The predictive sample reuse method with applications. *Journal of the American Statistical Association*, 70(350):320.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Granger, C. W. J. and Pesaran, M. H. (2000). Economic and statistical measures of forecast accuracy. *Journal of Forecasting*, 19(7):537–560.

Griewank, A. and Walther, A. (2008). *Evaluating derivatives: principles and techniques of algorithmic differentiation*, volume 105. Siam.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning data mining, inference, and prediction: With 200 full-color illustrations*. Springer-Verlag New York, New York, 4 edition.

Hoerl, A. E. and Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55.

Hotelling, H. (1936). Relations between two sets of variates. *Biometrika*, 28(3/4):321.

Hsieh, D. A. (1991). Chaos and nonlinear dynamics: Application to financial markets. *The Journal of Finance*, 46(5):1839–1877.

Huber, P. J. (2011). *Data analysis: What can be learned from the past 50 years*. John Wiley & Sons, United States.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. *Proceedings of the 5th international conference on Learning and Intelligent Optimization*, pages 507–523.

Keerthi, S. S., Sindhwani, V., and Chapelle, O. (2007). An efficient method for gradient-based adaptation of hyperparameters in svm models. In *Advances in neural information processing systems*, pages 673–680.

Larsen, J., Hansen, L., Svarer, C., and Ohlsson, M. (1996). Design and regularization of neural networks: The optimal use of a validation set. *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing*, VI(Piscataway, New Jersey):62–71.

Larsen, J., Svarer, C., Andersen, L. N., and Hansen, L. K. (1998). Adaptive regularization in neural network modeling. In *Neural Networks: Tricks of the Trade*, pages 113–132. Springer.

Larsen, J., Svarer, C., Andersen, L. N., and Hansen, L. K. (2012). Adaptive regularization in neural network modeling. *Neural Networks: Tricks of the Trade*, pages 111–130.

Larson, S. C. (1931). The shrinkage of the coefficient of multiple correlation. *Journal of Educational Psychology*, 22(1):45–55.

Lee, S.-I., Lee, H., Abbeel, P., and Ng, A. Y. (2006). Efficient l1 regularized logistic regression. Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06).

Lintner, J. (1965). The valuation of risk assets and the selection of risky investments in stock portfolios and capital budgets. *The Review of Economics and Statistics*, 47(1):13.

Luketina, J., Berglund, M., Greff, K., and Raiko, T. (2016). Scalable gradient-based tuning of continuous regularization hyperparameters. In *International Conference on Machine Learning*, pages 2952–2960.

MacKay, D. J. C. (1992). Bayesian interpolation. *Maximum Entropy and Bayesian Methods*, pages 39–66.

Maclaurin, D., Duvenaud, D., and Adams, R. P. (2015). Gradient-based hyperparameter optimization through reversible learning. *Arxiv preprint*, arXiv:1502.03492.

Moody, J. E. (1991). Note on generalization, regularization and architecture selection in nonlinear learning systems. *Neural Networks for Signal Processing Proceedings of the 1991 IEEE Workshop*.

Ng, A. Y. (2004). Feature selection, l1 vs. l2 regularization, and rotational invariance. In ICML.

Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 6*, 2(11):559–572.

Pedregosa, F. (2016). Hyperparameter optimization with approximate gradient. *arXiv preprint arXiv:1602.02355*.

Pesaran, M. H. and Timmermann, A. (1995). Predictability of stock returns: Robustness and economic significance. *The Journal of Finance*, 50(4):1201.

Pesaran, M. H. and Timmermann, A. (2000). A recursive modelling approach to predicting uk stock returns. *The Economic Journal*, 110(460):159–191.

Rivals, I. and Personnaz, L. (1999). On cross validation for model selection. *Neural Computation*, 11(4):863–870.

Ross, S. A. (1976). The arbitrage theory of capital asset pricing. *Journal of Economic Theory*, 13(3):341–360.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Sharpe, W. F. (1964). Capital asset prices: A theory of market equilibrium under conditions of risk. *The Journal of Finance*, 19(3):425.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, 25:2960–2968.

Spearman, C. (1904). General intelligence objectively determined and measured. *The American Journal of Psychology*, 15(2):201–292.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B*, 58(1):267–288.

Tsay, R. S. (2005). *Analysis of Financial Time Series*. Wiley, 3rd edition.

Wahba, G. (1990). *Spline models for observational data*. Society for Industrial & Applied Mathematics,U.S., Philadelphia, PA, 4 edition.

Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320.