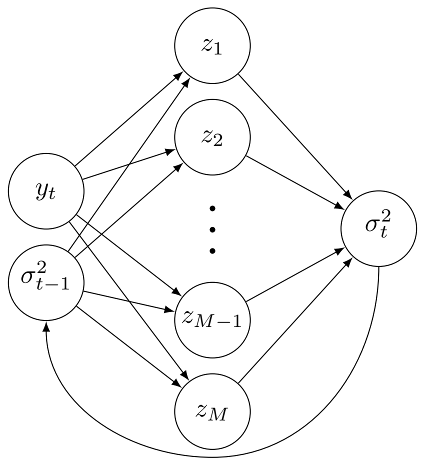
Modelling Volatility with a Recurrent Neural Network

Christian Duffau-Rasmussen

02-11-2015



Contents

1 Introduction

It is a stylized fact that financial returns exhibit a great deal of auto dependence in volatility across time Taylor (2005) and the marginal distribution has heavier tails than the normal distribution. This autocorrelation, also referred to as volatility clustering, has been somewhat successfully modelled since the seminal papers of Engle (1982) which introduced the ARCH model, and later expanded byBollerslev (1986) introducing the GARCH model. These model both account for the autocorrelation of the size of the returns and the heavy tailed marginal distribution.

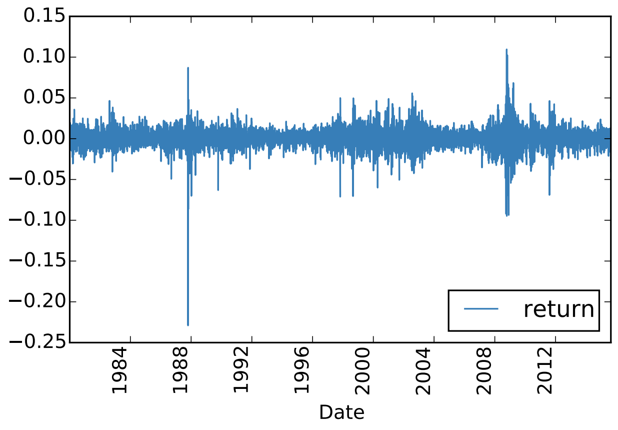


Figure 1: Daily returns of the S&P500 index.

Many extensions to the class of ARCH models have been made. It is observed empirically that negative shocks have greater impact to volatility then positive shocka, so to account for this asymmetry many variations to the ARCH-class have been made, for example Engles asymmetric GARCH Engle (1990) and Nelsons EGARCH Nelson (1991) are two classical asymmetric GARCH-type models.

The standard GARCH model is affected by the squared passed returns, emperically this results in the model reacting to violently to shocks. Paired with the very long memory of estimated GARCH-models, big shocks tends to keep the conditional variance at at high level for longer then data suggests. This issue have been met with different power transformations of the innovations, in models such as the Asymmetric Power ARCH model of Ding et al. (1993).

Lately GAS-models (Generalized Autoregressive Score) introduced by Creal et al. (2013), model the conditional variance by the scaled derivative of the conditional density at time t, where the scaling is the time dependent second derivative of the density. These GAS-models seem to be less affected by big shocks to the model due to the dynamic scaling of the innovations impact on the variance.

All of these models try to model the conditional variance of returns with some transformation of passed returns. In this paper I am seeking to estimate the transformation from passed returns to the contemporary variance. To accomplish this, I make use of a tool from the machine learning literature known as artificial neural networks. Neural networks, as the name suggests, are inspired by biological systems of neurons, but

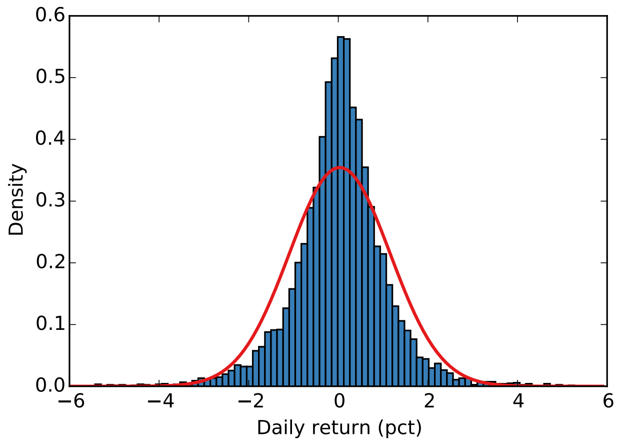


Figure 2: Marginal distribution of returns

the similarity between real life neurons and there mathematical representations has often been overstated Bishop (2006). The task of a neural network, in a machine learning perspective, is not to mimic the neurologic biology, but to make predictions from experience. Mathematically speaking a neural network is just a function from $\mathbb{R}^D$ to $\mathbb{R}^K$, nothing more. The transformation is obtained by a series of alternating linear and sigmoidal transformations of the D-dimensional input vector. The transformation is governed by a high number of parameters, which can be estimated from data. The usefulness of these networks comes from the fact that they are very flexible function approximators. Hornik et al. (1989) and Cybenko (1989) showed that any Borel-measurable function, on a compact domain, can be approximated to arbitrary precision as the number of hidden neurons grow. Borel-measurable functions, constitutes a very broad class of functions, which easily encompasses any function of practical relevance.

Volatility modelling with neural network

2 The models

2.1 GARCH model

The well known GARCH(1,1)-model,

$$\begin{aligned} \sigma_t^2 &= \omega + \alpha y_{t-1}^2 + \beta \sigma_{t-1}^2 \\ y_t &= \sigma_t \varepsilon_t \end{aligned}$$

where I in this paper assume $\varepsilon \sim NID\left(0,1\right)$.

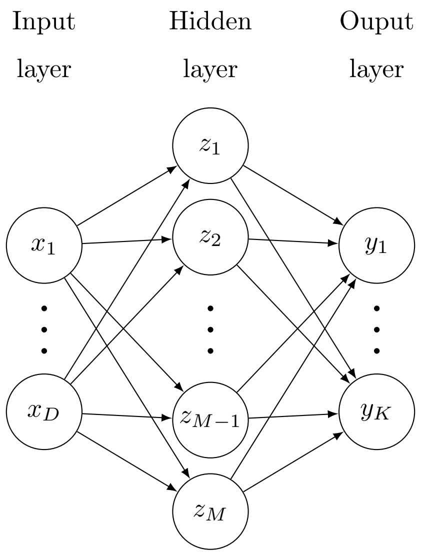


Figure 3: Architecture of the standard feed forward neural network.

2.2 The feed forward neural network

The standard feed forward neural network consists of D real inputs $x_1, \ldots, x_D$, M hidden "neurons", $z_1, \ldots, z_M$ and K real outputs, $y_1, \ldots, y_K$. The architecture or the topology of the network is that each hidden neuron, for example neuron number 1, produces a scalar output which is a function of all inputs $x = \begin{bmatrix} 1 & x_1 & \cdots & x_D \end{bmatrix}'$ augmented with a 1 in the first position, and some $D+1$ weights $w_{1,H} = \begin{bmatrix} w_{10}^H & w_{11}^H & \cdots & w_{1D}^H \end{bmatrix}'$. The output of the hidden neuron 1, z_1, is some activation function $h(\cdot)$ of the linear combination of x and $w_{1,H}$,

$$z_1 = h\left(x' w_{1,H}\right) = h\left(w_{10}^H + x_1 w_{11}^H + \cdots + x_D w_{1D}^H\right).$$

The choice of activation function is subject to active research and different context make use of different activation functions. A classic choice is some sigmoidal function, which can be generally defined as a bounded monotonically increasing function Bishop (2006). The general definition of the activation function is important in theoretical work, such as universal approximation properties like the seminal paper (in Machine learning) Hornik et al. (1989). The choice also can greatly impact the prediction performance of the network Karlik and Olgac (2011), but in this paper I stick with the classical logistic function,

$$h(x) = \frac{1}{1 + \exp(-x)},$$

for simplicity.

So each hidden neuron has it's own weight vector $w_{i,H}$ and produces a scalar output, in my case bounded between 0 and 1. These neuron outputs are collected in a vector z which serves as input to the output layer. So in a similar fashion each output value y_i is a function of all hidden neurons $z = \begin{bmatrix} 1 & z_1 & \cdots & z_M \end{bmatrix}'$ and the appropriate $M+1 \times 1$ weight vector $w_{i,O} = \left[\begin{array}{cccc} w_{10}^O & w_{11}^O & \cdots & w_{1M}^O \end{array} \right]'$. So formally,

$$y_i = g\left(z' w_{i,O}\right)$$

where $g(\cdot)$ can be the identity[1] for regression tasks, a sigmoidal function if one is interested in a probability or possibly some more complicated differentiable function.

So defining the $D+1 \times M$ matrix W_H as the stacked collection of all $w_{i,H}$'s and likewise the $M+1 \times K$ matrix W_O consisting of all $w_{i,O}$'s, we can write the (single layer) feedforward neural network as,

$$\begin{aligned} z &= \mathbf{h}\left(x' W_H\right) \\ y &= \mathbf{g}\left(z' W_O\right) \end{aligned}$$

where the two functions $\mathbf{h}$ and $\mathbf{g}$ are the elementwise counterparts of h and g respectively.

This can of course be extended to the multilayer case,

$$\begin{aligned} z_1 &= h_1\left(x' W_{H1}\right) \\ z_2 &= h_2\left(z_1' W_{H2}\right) \\ &\vdots \\ z_n &= h_n\left(z_{n-1}' W_{Hn-1}\right) \\ y &= g\left(z_n' W_O\right) \end{aligned}$$

which is the model playing a key role in the recent wave of, so called, "deep learning" algorithms, where the term "deep" refers to a network with many layers.

2.3 The Jordan Neural Network - Jordan (1986)

To make use of the neural network for times series analysis some auto dependence is needed. In this paper I make use of the Jordan network, which is the simplest recurrent network. All of the recurrency comes from the output at time t serving as input at time $t+1$, so the architecture of the network is not any different from the simple feed forward neural network.

In the present case the output is a positive scalar σ_t^2, like the latent process in the GARCH model, and the input consists of two variables, the contemporary return y_t and the lagged variance σ_{t-1}^2. So the model is,

$$\begin{aligned} z_t &= \mathbf{h}\left(\left[\begin{array}{ccc} 1 & y_t & \sigma_{t-1}^2 \end{array} \right] W_H\right) \\ \sigma_t^2 &= g\left(z_t' W_O\right) \end{aligned}$$

where $\mathbf{h}$ is the vector valued logit function. The two parameter matrices W_H and W_O are of dimension $3 \times M$ and $(M+1) \times 1$, respectively, so in total $4M+1$ parameters need to be estimated. The activation

[1] $x \mapsto x$

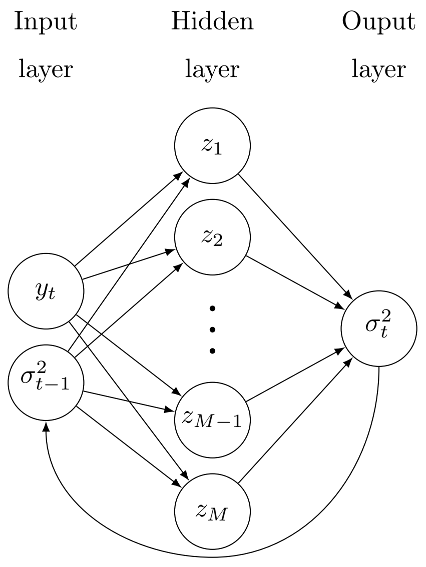


Figure 4: Architecture of the Jordan neural network.

function in the final layer is chosen toexp $(\cdot)$ since it ensures a positive variance and makes gradient derivation convenient.

Since neural nets are universal approximators the GARCH model is in principle nested in the Jordan net. This is a interesting feature since it makes it possible to evalaute how close the news impact function of the GARCH model is to the "empirical" news impact function, which in theory could be approximated arbitrarily well by the neural network.

3 The loss function

The exact log-likelihood of the observed sequence $y_{\{0,\dots,T\}} = \mathbf{y}$ is,

$$\begin{aligned}
\mathcal{L}^*(\theta;\mathbf{y}) &= f(\mathbf{y};\theta) \\
&= f(y_T|\, y_0,\dots,y_{T-1};\theta)\cdot f(y_0,\dots,y_{T-1};\theta) \\
&= f(y_T|\, y_0,\dots,y_{T-1};\theta)\cdot f(y_{T-1}|\, y_0,\dots,y_{T-2};\theta)\cdot f(y_0,\dots,y_{T-2};\theta) \\
&\vdots \\
&= \prod_{t=1}^{T} f(y_t|\, y_0,\dots,y_{t-1};\theta)\, f(y_0;\theta)\,.
\end{aligned}$$

Assume the process y_t is a Markov chain we have,

$$\mathcal{L}^*(\theta;\mathbf{y}) = \prod_{t=1}^{T} f(y_t|\, y_{t-1};\theta)\, f(y_0;\theta)\,.$$

It can be shown that maximizing $\mathcal{L}^*$ with respect to θ is equivalent to maximizing the conditional likelihood $\mathcal{L}(\theta;\mathbf{y}) = \mathcal{L}^*(\theta;\mathbf{y})/f(y_0;\theta)$, so when reffering to the likelihood of $\mathbf{y}$, the function $\mathcal{L}$ is meant.

If we assume the iid errors ε_t are standard normal and $E[y_t|y_{t-1}] = 0$, the likelihood is,

$$\begin{aligned}\mathcal{L}(\theta) &= \prod_{t=1}^{T} f(y_t|\, y_{t-1};\theta) \\ &= \prod_{t=1}^{T}\left(\frac{1}{\sqrt{2\pi\sigma_t^2}}\exp\left[-\frac{y_t^2}{2\sigma_t^2}\right]\right)\end{aligned}$$

taking the log of $\mathcal{L}(\theta)$ we get,

$$\log \mathcal{L}(\theta) = -\frac{1}{2}\left(T\log(2\pi) + \sum_{t=1}^{T}\left(\log\left(\sigma_t^2\right) + \frac{y_t^2}{\sigma_t^2}\right)\right).$$

To stay in line with machine learning literature I define the negative of the log-likelihood as the loss function, that is $L = -\log \mathcal{L}$.

In estimation of neural network overfitting is a very relevant issue. Overfitting is the case where the model approximates the variation in the estimation sample to well, so it generalizes poorly when facing new data. A way to combat overfitting is by adding a so called regularizing term, so the loss function becomes,

$$L^* = L + \lambda\frac{1}{2}\mathbf{w}'\mathbf{w},$$

where $\mathbf{w}$ is a vector containing all parameters. So adding a scaled version of sum of squared parameters forces the optimization to set some weights close to zero, and in that way finding a suboptimal solution in the strict sense. The motivation for $\lambda\frac{1}{2}\mathbf{w}'\mathbf{w}$ comes from a Bayesian view where, assuming a Normal prior distribution of $\mathbf{w}$,L^* is the log of the posterior joint density, and λ is a function of the assumed parameters in the prior normal distribution. The regularized loss function is only used in estimation of the Jordan NN model, the classic GARCH model is estimated maximizing the likelihood.

3.1 Estimation of the GARCH model

Estimation of the GARCH model is done using the BFGS algorithm with numerically computed gradient and Hessian, to minimize the loss function. The procedure can be summarized in pseudo-code (see Algorithm 1).

It is important to notice that each step in the out minimization loop requires a evaluation over all T observations and the computation of the gradient and Hessian requires at least one evaluation of the loss function per parameter, plus an extra evaluation for the starting point of the secants. So at least so in the order of $T \cdot D + T$ evaluations of the each loss contribution L_t. For the case of the GARCH(1,1) model this is inexpensive since $D = 3$.

Algorithm 1 Estimation of GARCH model

```
Initialize with random parameters $\theta_0 = (\omega_0, \alpha_0, \beta_0)$
while Minimization criteria of $L(\theta_k)$ is not met do
    while $t < T$ do
        Compute $\sigma_t^2(\omega_k, \alpha_k, \beta_k)$
        Compute $t$'th loss function contribution $L_t(\theta_k) = \frac{1}{2}\left(\log(2\pi) + \log\left(\sigma_t^2\right) + \frac{y_t^2}{\sigma_t^2}\right)$
    end while
    Compute loss function: $L(\theta_k) = \sum_{t=1}^{T} L_t(\theta_k)$
    Compute numerical gradient: $\nabla L(\theta_k) = \sum_{t=1}^{T} \nabla L_t(\theta_k)$
    Compute approximate Hessian.
    Update $\theta_k \mapsto \theta_{k+1}$ according to BFGS
end while
Return $\theta_k$
```

3.2 Estimation of the Jordan neural net

Stochastic gradient decent

When estimating a Neural Network stochastic gradient descent (SGD) is the standard approach Bishop (2006). Gradient descent is a simpler version of the Newton-Raphson algorithm, where the update of the parameters/weights is done without taking the hessian into account. In its simplest form all weights are changed by the negative of the partial derivative in the given weight direction scaled by some constant ρ.

The "stochastic" part of SGD comes from taking a small random sequence of the data before each optimization step. This way one saves computations since the computation of the gradient in each of the D dimensions in done one much fewer data points. In the case of neural networks this has practical importance since D can easily be in the thousands.

The gradient

In the RNN setup $\sigma_t^2(\mathbf{w})$ is the output and a function of all the weights $\mathbf{w}$, where $\mathbf{w}$ is a $((D+1)M + (M+1)) \times 1$ vector containing all the weights in the neural network.

In the following I consider a Jordan style neural network, with D input variables at each point in time, M hidden neurons and one output neuron, namely σ_t^2. To be able to minimize the loss function L, w.r.t. to the weight vector $\mathbf{w}$, we are interested in computing the partial derivatives w.r.t.$\mathbf{w}$. Omiting multiplicative and additional constants which are irrelevant for minimization, the loss function is the sum of T terms on the form $\log\left(\sigma_t^2\right) + y_t^2/\sigma_t^2$. So each partial derivative is just the sum of each derivative contribution. So analyzing a single contribution to the loss function,

$$\frac{\partial L_t}{\partial \mathbf{w}}$$

Algorithm 2 Stochastic Gradient Descent

```
Initialize with random weigths $w_0$
while Minimization criteria of $L(w_k)$ is not met do
    Take a random subsequence of size $T_{sub}$
    while $t < T_{sub}$ do
        Compute $\sigma_t^2(w_k)$
        Compute $t$'th loss function contribution $L(\theta_k) = \frac{1}{2}\left(\log(2\pi) + \log(\sigma_t^2) + \frac{y_t^2}{\sigma_t^2}\right)$
    end while
    Compute reguralized loss: $L^*(w_k) = \sum_{t=1}^{T} L_t(\theta_k) + \frac{\lambda}{2} w'w$
    Compute the gradient: $\nabla L^*(w_k) = \sum_{t=1}^{T} \nabla L_t^*(w_k)$
    Update $w_k \mapsto w_{k+1}$ according to $w_{k+1} = w_k - \rho \nabla L^*(w_k)$.
end while
Return $w_k$
```

will suffice for getting the whole gradient. According to the chain rule we have,

$$\frac{\partial L_t}{\partial \mathbf{w}} = \frac{dL}{d\sigma_t^2} \frac{\partial \sigma_t^2(\mathbf{w})}{\partial \mathbf{w}}$$

and the first term i easily calculated to be, $\left(\sigma_t^2 - y_t^2\right)/\sigma_t^4$.

In the neural network there are two types of weights, the $(M+1) \times 1$ matrix of weights in the output layer, W_O, and the $(D+1) \times M$ matrix weights in the hidden layer, W_H. I am first considering the gradient of σ_t^2 w.r.t. the output weights. Using the chain rule again we get,

$$\frac{\partial \sigma_t^2}{\partial W_O} = \frac{d\sigma_t^2}{dg} \frac{\partial g(a_t)}{\partial W_O}$$

where a_t is the activation of the output neuron, that is the linear combination $z_t' w_O$. Applying the chain rule once again we obtain,

$$\begin{aligned} \frac{\partial \sigma_t^2}{\partial W_O} &= \frac{d\sigma_t^2}{dg} \frac{dg(a_t)}{da_t} \frac{\partial (z_t' W_O)}{\partial W_O} \\ &= \frac{d\sigma_t^2}{dg} \frac{dg(a_t)}{da_t} z_t. \end{aligned}$$

Since the output activation function g was chosen to be $\exp(\cdot)$ and $d\sigma_t^2/dg = 1$, the expression simplifies to,

$$\frac{\partial \sigma_t^2}{\partial W_O} = \exp(z_t' W_O)\, z_t = \sigma_t^2 z_t.$$

Combining with $dL_t/d\sigma_t^2$, the partial derivatives of the loss function w.r.t. w_O is simply,

$$\frac{\partial L_t}{\partial W_O} = \left(1 - \frac{y_t^2}{\sigma_t^2}\right) z_t = \left(1 - \frac{y_t^2}{\sigma_t^2}\right) \begin{bmatrix} 1 \\ z_{1,t} \\ \vdots \\ z_{M,t} \end{bmatrix}.$$

Now considering the Jacobian of σ_t^2 w.r.t. the weights of the hidden layer,

$$W_H = \begin{bmatrix} w_{0,1}^H & w_{1,2}^H & \cdots & w_{0,M}^H \\ \vdots & \vdots & \ddots & \vdots \\ w_{D,1}^H & w_{D,2}^H & \cdots & w_{D,M}^H \end{bmatrix}.$$

I focus on the partial derivative of a single weight w_{ij}^H, and apply the chain rule to get,

$$\frac{\partial \sigma_t^2}{\partial w_{ij}^H} = \frac{d\sigma_t^2}{dg}\frac{\partial g\left(a_t^O\right)}{\partial w_{ij}^H}$$

where $a_t^O = z_t' w_O$, is the activation of the output neuron. Applying the chain rule once again we get,

$$\frac{\partial \sigma_t^2}{\partial w_{ij}^H} = \frac{d\sigma_t^2}{dg}\frac{dg\left(a_t^O\right)}{da_t^O}\frac{\partial z_t' W_O}{\partial w_{ij}^H}.$$

Looking closer at the last term we notice that w_{ij}^H is only present in the j-th element of z_t, so the partial derivative is just,

$$\frac{\partial z_t' W_O}{\partial w_{ij}^H} = w_j^O \frac{\partial z_{j,t}}{\partial w_{ij}^H}.$$

Defining $a_{j,t}^H = x_t' w_{H,j}$ we have that $z_{j,t} = h\left(a_{j,t}^H\right)$ and applying the chain rule yet again, we get,

$$\begin{aligned}\frac{\partial z_t' W_O}{\partial w_{ij}^H} &= w_j^O \frac{\partial z_{j,t}}{\partial w_{ij}^H} \\ &= w_j^O \frac{\partial z_{j,t}}{\partial a_j^H}\frac{\partial a_j^H}{\partial w_{ij}^H}.\end{aligned}$$

Since h is the logistic function the partial derivative[2] of $z_{j,t}$ w.r.t a_{ij}^H is just $z_{j,t}\left(1 - z_{j,t}\right)$, and since a_j^H is just the linear combination of the inputs the derivative is,

$$\frac{\partial z_t' W_O}{\partial w_{ij}^H} = w_j^O z_{j,t}\left(1 - z_{j,t}\right) x_i.$$

So all in all the Jacobian of σ_t^2 w.r.t. W^H is,

$$\frac{\partial \sigma_t^2}{\partial w_{ij}^H} = \sigma_t^2 w_j^O z_{j,t}\left(1 - z_{j,t}\right) x_i.$$

4 VaR forecast

Conditional Value at risk (VaR) is defined as the lower quartile of the conditional distribution of returns,

$$V\hat{a}R_{t|t-1}^{\alpha} = \hat{\mu} + \hat{\sigma}_t \Phi^{-1}\left(\delta\right),$$

where I have assumed the iid innovations to be normally distributed.

[2] $\frac{dh}{da} = \frac{e^{-a}}{\left(1+e^{-a}\right)^2} = \frac{1+e^{-a}-1}{\left(1+e^{-a}\right)^2} = h(1-h)$

4.1 Kupiec test for unconditional coverage

To test whether the the α of the VaR is in line with data i compute the unconditional coverage, that is the number of times the VaR band is exceeded by the observed returns. So defining $I_t = \mathbf{1}\left(y_t < V\hat{a}R^{\alpha}_{t|t-1}\right)_t$ as the indicator of the returns exceeding the VaR band, the unconditional coverage is,

$$\hat{p} = \frac{1}{T}\sum_t I_t.$$

A formal test for whether $\hat{p}$ is different from α, that is testing the hypothesis,

$$H_0 : \hat{p} = \alpha$$
$$H_a : \hat{p} \neq \alpha$$

can be derived as a likelihood ratio test assuming I_t is iid Bernoulli distributed. The likelihood ratio is,

$$\Lambda = \frac{\hat{p}^{n_1} \cdot (1-\hat{p})^{(T-n_1)}}{\alpha^{n_1} \cdot (1-\alpha)^{(T-n_1)}}$$

where $n_1 = \sum_t I_t$, and according to Wilk's theorem the statistic $2\log(\Lambda)$ is χ^2 distributed with one degree of freedom.

4.2 VaR comparison

In this section I compare the one period conditional VaR forecast of the two models. Both models are estimated on the 4495 demeaned returns from 1980-01-03 to 1997-10-10 and the VaR forecast are made out of sample on the period 1997-10-13 to 2015-08-24.

The GARCH model is estimated as described earlier and the parameter estimates of $\left(\hat{\omega}, \hat{\alpha}, \hat{\beta}\right) = \left(1.21 \cdot 10^{-6}, 0.0667, 0.92\right)$ are as expected.

The Jordan neural network is estimated with 5 hidden neurons which gives a total 21 parameters to estimate. Estimation of the neural network was subject to many practical issues and the validity of the estimated model is questionable, even though the deviations in VaR forecast are in line with the GARCH model..

The figure 5 shows the news impact curve of the estimated GARCH model and the neural network model. The value at zero is deducted from both functions to put them on equal scale.

In figure 6 we see that the estimated neural network exhibits very low persistence and the VaR plot is similar to that of ARCH models. My conclusion is that the neural network is not at it's global maximum since i would expect a VaR forecast more similar to the GARCH one.

For both models I can reject the null of $\hat{p} = \delta$ in the Kupiec test. It is important to bare in mind that the test is performed out-of-sample which explains the poor performance of the GARCH model.

Even though the Jordan model seems to have a rather strange VaR forecast, as seen in figure 6, it seems to do equally good in the unconditional coverage for the estimated parameter vector.

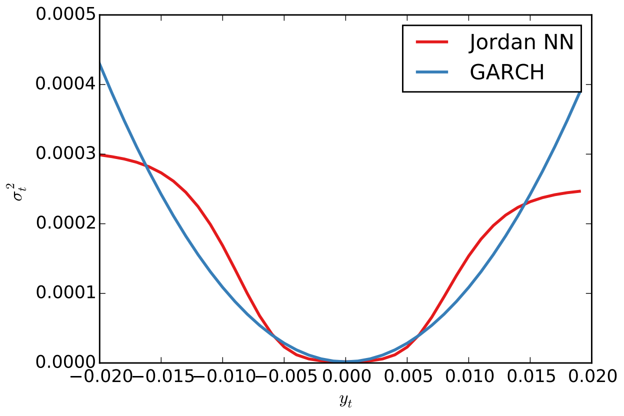
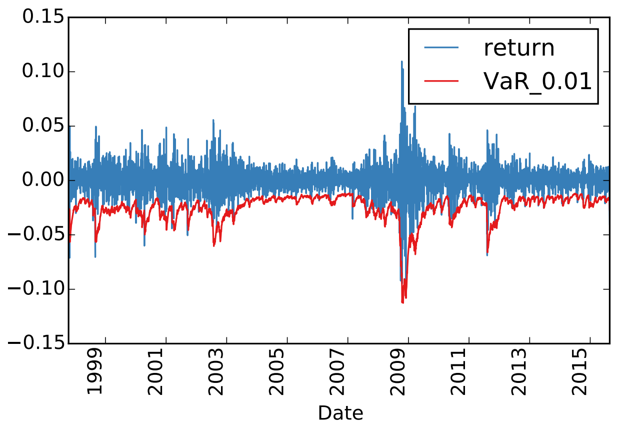
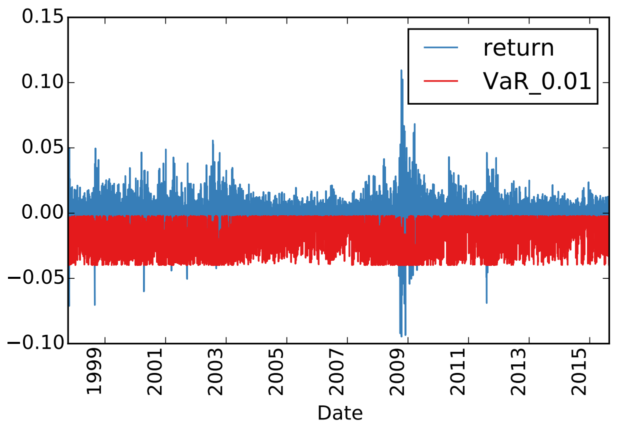


Figure 5: News impact curve of GARCH and Jordan Neural Network



(a) GARCH



(b) Jordan NN, 5 hidden neurons

Figure 6: One period 1%-CVaR forecast and SP500 returns

The two figures 7 and 8 are autocorrelation plots of the VaR violation indicator $I_t^{(\alpha)}$. The idea behind these test is to see whether or not the VaR violation actually are iid. We see that the GARCH indicator has a significant positive autocorrelation at lag 2 and 4 across all quantile levels, but stays within the confidence bands, more or less, for all other lags.

The Jordan model on the other hand does a terrible job in exceeding the VaR band in a iid fashion, This suggest some problem with the model.

5 Conclusion

Unfortunately due to practical issues in estimation I have not been able to unleash the full power of the neural network. When estimating neural networks the so-called hyperparameters like the number of hidden neurons and the regularization coefficient, λ, are not known a priori. So different choices of these parameters

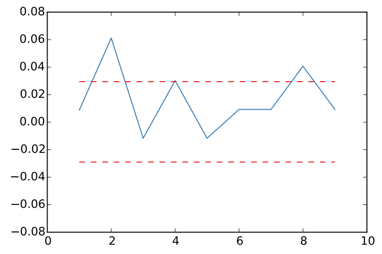
α	$\hat{p}$	Λ	p-value
1%	2.20%	48.9	0.000
2.5%	4.03%	36.4	0.000
5%	6.14%	11.5	0.000

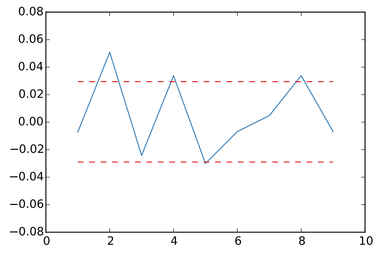
(a) GARCH

α	$\hat{p}$	Λ	p-value
1%	0.71%	4.19	0.024
2.5%	1.16%	41.4	0.000
5%	4.14%	7.5	0.0035

(b) Jordan NN, 5 hidden neurons

Table 1: Results of unconditional coverage test


(a) $\alpha = 1\%$


(b) $\alpha = 2.5\%$

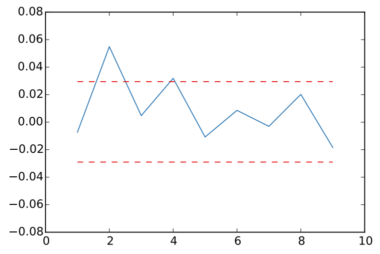

(c) $\alpha = 5\%$

Figure 7: ACF of indicator of y exceeding the VaR band- GARCH.

can yield very different models.

On the Kupiec test the Jordan neural net performs equally with the GARCH model, but from visual inspection of the VaR forecast and from the autocorrelograms on their VaR violation indicator, I conclude that the estimations procedure has not given a global optimum for the neural net. Resolving these issues will be the basis of my further work with neural nets.

References

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

T. Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3): 307–327, 1986. URL `http://EconPapers.repec.org/RePEc:eee:econom:v:31:y:1986:i:3:p:307-327`.

D. Creal, S. J. Koopman, and A. Lucas. Generalized autoregressive score models with applications. *Journal of Applied Econometrics*, 28(5):777–795, 2013. ISSN 1099-1255. doi: 10.1002/jae.1279. URL `http://dx.doi.org/10.1002/jae.1279`.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

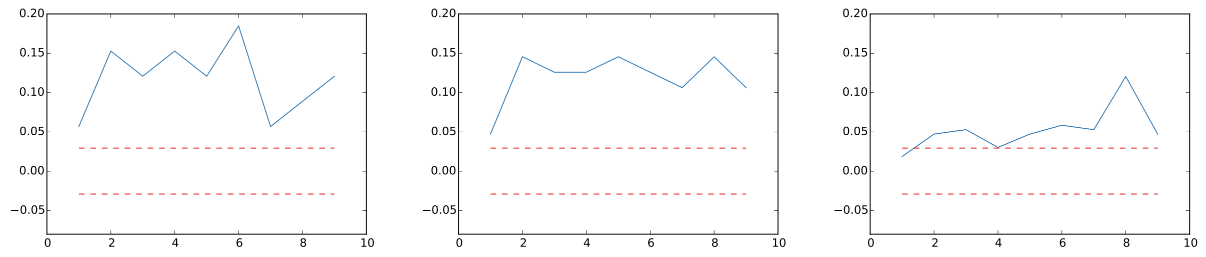

(a) $\alpha = 1\%$ (b) $\alpha = 2.5\%$ (c) $\alpha = 5\%$

Figure 8: ACF of indicator of y exceeding the VaR band - Jordan NN .

Z. Ding, C. W. Granger, and R. F. Engle. A long memory property of stock market returns and a new model. *Journal of empirical finance*, 1(1):83–106, 1993.

R. F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4):987–1007, 1982. URL `http://EconPapers.repec.org/RePEc:ecm:emetrp:v:50:y:1982:i:4:p:987-1007`.

R. F. Engle. Stock volatility and the crash of '87: Discussion. *The Review of Financial Studies*, 3(1):103–106, 1990. ISSN 08939454, 14657368. URL `http://www.jstor.org/stable/2961959`.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5):359–366, July 1989. ISSN 0893-6080. doi: 10.1016/0893-6080(89)90020-8. URL `http://dx.doi.org/10.1016/0893-6080(89)90020-8`.

M. I. Jordan. Serial order: A parallel distributed processing approach. Technical Report ICS Report 8604, Institute for Cognitive Science, University of California, San Diego, 1986.

B. Karlik and A. V. Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *Int. J. Artificial Intell. Expert Syst*, 1(4):111–122, 2011.

D. B. Nelson. Conditional heteroskedasticity in asset returns: A new approach. *Econometrica: Journal of the Econometric Society*, pages 347–370, 1991.

S. J. Taylor. *Asset Price Dynamics, Volatility, and Prediction.* Princeton University Press, stu - student edition edition, 2005. ISBN 9780691134796.