Notes on NN for forecasting

Nicos Christofides

nicos.christofides@gmail.com

October 2009

1 Introduction

The NN is represented by a graph $G(V, E)$ where V is a set of vertices (neurons) and E a set of arcs (signal-arcs). G is a multi-stage graph with stages $p = 0, \ldots, \hat{p}$. Stage p contains the subset V_p of vertices, so $V = V_0 \cup V_1 \cup \ldots \cup V_{\hat{p}}$. The sets V_0 and $V_{\hat{p}}$ are referred to as the *input* and *output* vertices of the NN, respectively.
An arc connects a vertex, say $v_i \in V_{p-1}$, to a vertex, say $v_j \in V_p$, and is written as (v_i, v_j). For any given v_j, the set of all vertices v_i with arcs (v_i, v_j) is written as V_j^{in} (with corresponding arcs E_j^{in}) and the set of all vertices $v_k \in V_{p+1}$ with arcs (v_j, v_k) is written as V_j^{out} (with corresponding arcs E_j^{out}).

With each vertex $v_i \notin V_0$ is associated a weight u_i, and with each arc (v_i, v_j) is associated a weight w_{ij}. The *NN input signals* are allocated as the output signals from the vertices V_0. In general, each vertex $v_i \in V$ produces an output signal y_i which is transmitted to all vertices $v_j \in V_i^{out}$ along the arcs (v_i, v_j). The net input signal into a vertex v_j, say, is then

$$x_j = u_j + \sum_{v_i \in V_j^{in}} w_{ij} y_i \tag{1}$$

1.1 The transfer function

With each vertex $v_j \notin V_0$ is associated a *transfer function* $f_j(.)$ which takes as parameter the input x_j and produces the output signal $y_j = f_j(x_j)$. Normally, the same function $f(.)$ is used for all vertices and this function is nonlinear, often with its value bounded from above and below. Our code uses the *logistic function* $f(x) = \frac{1}{1+e^{-x}}$ monotonically increasing from the value 0 (as $x \to -\infty$) to the value 1 (as $x \to +\infty$) with the value 1/2 at $x = 0$. Other transfer functions (eg $\tanh x$, bounded between -1 and +1) may also be used).

2 The training set

Assume that:

- The topology of the graph G is given, with $|V_0| \equiv n$ inputs and $|V_{\hat{p}}| \equiv m$ outputs. For example we may specify $\hat{p} = 3$ for a 3-period NN with $|V_0|, |V_1|, |V_2|, |V_3|$ having the values 10, 4, 2, 2, respectively; namely with 10 inputs, two 'hidden layers' with 4 and 2 vertices (neurons) respectively, and 2 outputs.
- An ordered set of $\hat{\tau}_{train}$ n-dimensional vectors $\underline{\alpha}_\tau$, $\tau = 1, \ldots, \hat{\tau}_{train}$ representing *input training instances* is given. The value of the j^{th} component of vector $\underline{\alpha}_\tau$ at instance τ is written as $\alpha_{j\tau}$. We will not discuss the choice of inputs here other than to say that in version 1 we are using a vector of dimension $n = 19$.
- An ordered set of $\hat{\tau}_{train}$ m-dimensional vectors $\underline{\beta}_\tau$, $\tau = 1, \ldots, \hat{\tau}_{train}$ representing *target training instances* is also given. The value of the j^{th} component of vector $\underline{\beta}_\tau$ at instance τ is written as $\beta_{j\tau}$. We will not discuss the choice of targets here other than to say that in version 1 we are using a vector of dimension $m = 1$ and that the corresponding $\beta_{1\tau}$ is the return of the asset *relative* to the average return of it's sector.

2.1 Preprocessing the training set

The list of pairs of input/target training instances above is jointly referred to as the *training set*. In version 1, the NN input training instances (namely $\alpha_{j\tau}$) are normalized to have mean 0 and variance 1. As usual this means: $\alpha_{j\tau} \leftarrow \alpha_{j\tau} - \mu_j$ (where μ_j is the mean value of $\alpha_{j\tau}$ over all τ) followed by $\alpha_{j\tau} \leftarrow \alpha_{j\tau}/\sigma_j$ (where σ_j is the standard deviation of the resulting $\alpha_{j\tau}$ over all τ). The normalization parameters μ_j and σ_j are stored on file so they are applied to preprocessing other NN input instances in the future.
As mentioned earlier these NN input training instances are allocated as the output signals from $v_j \in V_0$. Therefore for vertex v_j and instance τ, we define $y_{j\tau}$ as $\alpha_{j\tau}$.

In version 1, the NN target training instances (namely $\beta_{j\tau}$) are standardized to lie between 0 and 1. As usual this means: $\beta_{j\tau} \leftarrow (\beta_{j\tau} - \beta_j^\vee)/(\beta_j^\wedge - \beta_j^\vee)$, where $\beta_j^\wedge$ is the maximum value of $\beta_{j\tau}$ over all τ and $\beta_j^\vee$ is the minimum value of $\beta_{j\tau}$ over all τ. The standardization parameters $\beta_j^\vee$ and $\beta_j^\wedge$ are stored on file so they are applied in the future (in reverse order) to NN output instances to convert the output to a forecast in the original target units. (Note: Because the logistic function reaches 0 and 1 only asymptotically, it is preferable to increase $\beta_j^\wedge$ by a factor $p > 1$ and reduce $\beta_j^\vee$ by a factor $1/p$, prior to standardization. This ensures that the actual maximum and minimum targets are achievable with finite inputs into the output vertices.)
In version 1, the value of p is taken as 1.05

2.2 The NN objective function

When an input vector instance $\underline{\alpha}_\tau$ is presented to the network, a vector output instance $\underline{\phi}_\tau$ results. We want $\underline{\phi}_\tau$ to be as close to the target $\underline{\beta}_\tau$ as possible, for all instances $\tau = 1, \ldots, \hat{\tau}_{train}$. If $\| \underline{\beta}_\tau - \underline{\phi}_\tau \|$ is some measure of the error between the target and NN output, then we want to find values of u_i and w_{ij} (vertex and arc weights, respectively) such that the total error $z = \sum_{\tau=1}^{\hat{\tau}_{train}} \| \underline{\beta}_\tau - \underline{\phi}_\tau \|$ is minimized. The error measure most often used is the mean-square error (MSE) $z = \sum_{\tau=1}^{\hat{\tau}_{train}} (\underline{\beta}_\tau - \underline{\phi}_\tau)^2$ which is convenient for computing the derivatives, etc. needed in the conjugate gradient techniques used in the optimization algorithm.

In our code we use the MSE measure to produce an initial locally MSE-optimal solution, but then change the objective to a *profitability index (PI)* and perform a local second-stage optimization - with this second objective - from there (see below). For the rest of this note, when we refer to 'performance', we mean w.r.t. the PI.

3 The training process

3.1 Overfitting

It is easy to see, that for a NN $G = (V, E)$ there are $|V| - |V_0|$ vertex weights and $|E|$ arc weights to determine. For most practical-size NNs, it means that there is a large number of variables to optimize. A rule of thumb is for the number of training instances $\hat{\tau}_{train}$ to be 5 to 10 times the number of free variables. Nevertheless, when using a NN for forecasting financial time series (which always contain a lot of noise) it is easy for the optimization to fit not only the main characteristics of the time series, but also the noise. This *overfitting* invariably leads to 'good' performance for the training set, but much worse performance in forecasting unseen future data. Three techniques can be used to reduce the risk of overtraining whilst still achieving 'reasonable' in-sample performance and, hopefully, improve the out-of-sample performance. (What is meant by 'reasonable' is a very gray area and in our code we avoid having to decide on this issue.)

- `Small NN:` The size of the NN is kept to the minimum possible, consistent with reasonable performance. There is no theoretical reason to choose a NN of a given size (although some bounds are derivable from 'immersing' the time series into a multidimensional space and using the box-counting algorithm). In our code we simply train different size NNs, and choose the 'best'. (See below.)
- `Reducing the number of inputs:` Striving for variable reduction also implies using as small a number (n) of NN inputs as possible. Obviously, this

also leads to a smaller NN. Reducing the number of NN inputs can be achieved by extracting from a potentially large input set a smaller subset as follows:

  - Use as NN inputs a (smaller) number of principal/independent components of the potential input time series, or
  - Use as NN inputs the smallest possible subset of the potential input set, consistent with reasonable performance. This is what we use in our case. It is not automated in the code. The user must try different combinations of the inputs. The intention (for version 2.0) is to collect more potential inputs (currently we have 19) [1] and for each asset, do a single run on a supercomputer to try every combination of those. For a given asset, the set of inputs will then be fixed 'forever'. (As mentioned earlier, in version 1.0 all 19 inputs are used.

- `Validation set:` Using a separate 'validation' technique to prematurely terminate the training process. This is used in our code and described separately below.

3.2 Validation

The conjugate gradient algorithm used to optimize the NN weights, is an iterative procedure that generates a sequence of weight 'solutions' from the initial one $(\mathbf{u}, \mathbf{w})^0$ (see below) to the locally optimal one $(\mathbf{u}, \mathbf{w})^{\hat{s}}$ in $\hat{s}$ steps. The corresponding values of the NN objective function (error) are $z^0_{train}, z^1_{train}, \ldots, z^{\hat{s}}_{train}$. This sequence decreases monotonically.

Now consider another set of, say, $\hat{\tau}_{valid}$ input/target instances and call this the *validation* set. When the above solution sequence is evaluated on the validation set, the values of the NN objective function (error) are $z^0_{valid}, z^1_{valid}, \ldots, z^{\hat{s}}_{valid}$, and generally this sequence is not monotonically decreasing. *Generalization* is a loose term used in the NN literature to measure the propensity of a NN trained on some data, to also perform well on other unseen data of the same problem. Forecasting, clearly needs good generalization, and a measure of this is the correlation between the two sequences $z^0_{train}, z^1_{train}, \ldots, z^{\hat{s}}_{train}$ and $z^0_{valid}, z^1_{valid}, \ldots, z^{\hat{s}}_{valid}$. Let $\breve{s}$ be the value of s for which z^s_{valid} is minimum. In our code we compute the correlation coefficient between the above two subsequences from $\breve{s}$ to $\hat{s}$. If the correlation is positive, we drop the first entry from the subsequences and repeat, if the correlation is negative we drop the last entry from the subsequences and repeat. In all cases, we stop when not enough entries remain in the subsequences to compute the correlation (say 5 entries) and declare the first entry of the remaining subsequences the 'best' optimization iteration s^*.

[1] Yazid has provided another 7

The weights at that iteration $(\mathbf{u}, \mathbf{w})^{s^*}$ are the final NN weights.

3.3 Testing

Once a NN is calibrated it must be tested on totally-unused data called the *test* set. This set consists of, say, $\hat{\tau}_{test}$ input/target instances. The generated NNs are then ordered for generalization depending only on their performance on the test set.

4 Generating a population of NNs

4.1 The modelling set

The training, validation and test sets, are referred to jointly as the *modelling* set. The choice of the modelling set is vital for the performance of a NN. A number of choices have to be made, and these are:

- The number $\hat{\tau}_{train}, \hat{\tau}_{valid}, \hat{\tau}_{test}$ of instances to use. In our code we use a total of 400 instances, divided into training, validation and testing as mentioned below.
- The number $n = |V_0|$ of inputs to use for each input instance. In our code we use 19.
- The number $m = |V_{\hat{p}}|$ of targets to use for each target instance. In our code we use 1.
- The dates to be used for collecting the modelling set. These need not be consecutive dates. In our code we use the following procedure:
  - Select 400 consecutive days finishing today (say time t_0).
  - Construct the 400 target instances, and the corresponding 400 input instances: Note that if we are forecasting d days ahead, the last date for an input instance is $t_0 - d$. Also note that the input instance for, say, time t (in order to forecast for time $t + d$) uses information from, say, d_{hist} days preceeding time t. For example, average returns over the 20 days preceeding time t may be used as an input at time t, in which case $d_{hist} = 20$. In our code $d_{hist} = 100$, because we use a rolling-window of the previous 100 days to compute Independent Components (and hence residuals) for time t.
  - Of the 400 instances above, we allocate 200 for training, 100 for validation, 100 for testing.
  - Training, validation and test subsets of the modelling set are generated by randomly selecting the required number of instances in each case, except that the testing set includes the last (i.e. most recent) 10 days $t_0 - 9, t_0 - 8, \ldots, t_0$. These last 10 days are called the *reserve*.

4.2 The NN generation process

Once the modelling set is selected:

1. Different partitions of the modelling set into training, validation and test subsets are produced as mentioned above.

2. Different topologies for the NN models to be created are proposed.

3. For each NN topology, different starting points for the NN weights are considered. Because weight optimization only leads to a local optimum, different starting points lead to different local optima.

By considering different combinations of the above 3 choices, we generate a population of trained NNs. All NNs are evaluated on the test set. Those with negative PI are eliminated and the rest are placed in a *model pool*. In our model we take 2 choices for item 1 above, 4 choices for item 2 and 3 choices for item 3, leading to a population of 24 NNs, the ones with positive PI forming the model pool.

4.3 NN model use at an arbitrary date t

For a given asset, the NN models in the model pool were produced and evaluated at a given time t_0. They are to be used from this time until some other future time t_{end} when it is decided to recreate the models. Although the NNs were created to have good performance and generalization at time t_0, their performance at time t, $t_0 < t \leqslant t_{end}$ must be re-evaluated at each time t that a forecast is needed. The re-evaluation is as follows:

- Generate the input and target instances for the $\hat{\tau}_{test}$ times preceeding time t, and refer to these instances by the index k. $k = 1$, for time $t - \hat{\tau}_{test} + 1$; $k = 2$, for time $t - \hat{\tau}_{test} + 2$, ..., and $k = \hat{k}$ for time t.

- For each NN (say NN_j) in the model pool, compute its performance using the above $\hat{k}$ input and target instances. Temporarily eliminate from the model pool any NN with negative performance, and let $\hat{\ell}$ be the size of the resulting pool.

4.3.1 A single forecasting model

In version 1.0 a single forecast is produced by averaging the forecasts of the $\hat{\ell}$ NNs in the model pool. We compute the forecast return as z and it's variance as Ω^2. In version 2.0 we will do the following:
Let $c_\ell(k)$ be the profit of NN_ℓ at instance k, and let $\bar{c}_\ell$ be it's expected profit. Let $q_{\ell\ell'}$ be the covariance between c_ℓ and $c_{\ell'}$ computed over the daily forecasted profits for the

$\hat{k}$ instances. We want to produce a forecast by weighing the forecast of NN_ℓ with ξ_ℓ. The best forecast is then given by the solution to:

$$\begin{aligned} \Omega^2 = \min_{\underline{\xi}} \quad & \underline{\xi}^T Q \underline{\xi} \\ s.t. \quad & \textstyle\sum_{\ell=1}^{\hat{\ell}} \xi_\ell = 1 \\ & \textstyle\sum_{\ell=1}^{\hat{\ell}} \xi_\ell \bar{c}_\ell \geqslant z \end{aligned}$$

where $Q = [q_{\ell\ell'}]$ is the covariance matrix and z is a forecasted value we want to achieve. For feasibility, any number $\min_\ell[\bar{c}_\ell] \leqslant z \leqslant \max_\ell[\bar{c}_\ell]$ will do, but we will set $z = \frac{1}{\hat{\ell}} . \sum_{\ell=1}^{\hat{\ell}} \bar{c}_\ell$. The above solution then produces a return above the average of forecasted expected returns, whilst minimizing the forecasted variance of those returns. The value of $\underline{\xi}^*$ producing the minimum above is used to generate the best single forecast. The corresponding value of z is the forecasted return and Ω^2 the variance of that forecast.

5 A forecaster's performance index (PI)

We use MSE as the objective function of a NN and compute vertex and arc weights to minimize MSE obtaining an 'optimal' solution $(\mathbf{u}, \mathbf{w})^*$. Following this initial optimization we perform a second-stage optimization as follows.
Imagine a small hypercube of dimension $|(\mathbf{u}, \mathbf{w})|$ centered on point $(\mathbf{u}, \mathbf{w})^*$. (The size of this hypercube is decided below.)
- Generate a number h of maximally-dispersed point-samples in the hypercube.
- At each one of these points (plus the centre point) evaluate a more complex but more realistic objective function called the *Performance Index* (PI).
- Choose for the NN weights that point which maximizes the PI.

Here we briefly describe the PI we use.
As mentioned earlier, in version 1 we use just one output vertex, and as a result we will drop the vertex's index when no confusion arises. For a given asset, we compute the target (which is the asset return relative to that of the corresponding sector) over the $\hat{k}$ instances described in the previous section, and group the values into quartile *ranges* (*not* discrete sets) $Q1, Q2, Q3, Q4$ counting from the top.
The following items affect the computation of the PI for a given NN forecasting model.

1. `The closeness of the forecast of an asset's return to the target.` We use the mean absolute deviation (MAD) measure $M_k = \mid \beta_k - \phi_k \mid$ for closeness.

2. `The probability of 'acting' (ie including the asset in the portfolio, long or short) based on the forecast.` Let

N_0 be the total number of assets in the problem and N be the number we wish to include (long or short) in the portfolio. For the purposes of computing the PI, we take this probability as $q_k = 0.2\frac{N}{N_0}$ if $\phi_k \in Q2 \quad \text{or} \quad Q3$ and $q_k = 0.8\frac{N}{N_0}$ if $\phi_k \in Q1 \quad \text{or} \quad Q4$.

3. `The result if the forecast is acted upon.` This is computed as follows:

   - Let $L > S$ be two real numbers representing the 'utility' of a large or small profit respectively. Also let $\lambda > 1$ so that $-\lambda L$and $-\lambda S$ be the 'utility' of a large or small loss respectively. We use the following procedure to compute the utility of a forecast that is acted upon:

$$\begin{array}{rcll} U_k & = & L; & \text{if} \quad \beta_k \in Q1 \quad \& \quad \phi_k \in Q1 \cup Q2, \text{ or} \\ & & & \beta_k \in Q4 \quad \& \quad \phi_k \in Q4 \cup Q3 \\ & = & S; & \text{if} \quad \beta_k \in Q2 \quad \& \quad \phi_k \in Q1 \cup Q2, \text{ or} \\ & & & \beta_k \in Q3 \quad \& \quad \phi_k \in Q4 \cup Q3 \\ & = & -\lambda.L; & \text{if} \quad \beta_k \in Q1 \quad \& \quad \phi_k \in Q4 \cup Q3, \text{ or} \\ & & & \beta_k \in Q4 \quad \& \quad \phi_k \in Q1 \cup Q2 \\ & = & -\lambda.S; & \text{if} \quad \beta_k \in Q2 \quad \& \quad \phi_k \in Q4 \cup Q3, \text{ or} \\ & & & \beta_k \in Q3 \quad \& \quad \phi_k \in Q1 \cup Q2 \end{array}$$

4. `The opportunity loss if the forecast is not acted upon.` On the basis that missing an 'opportunity' is not as important as an actual gain or loss, we define $\mu < 1$ and define the opportunity loss as $-\mu.U_k$

Thus, the PI (to be maximized) is computed as

$$\text{PI} = \sum_{k=1}^{\hat{k}} [-\gamma.M_k + (1-\gamma)U_k\{q_k - (1-q_k).\mu\}]$$

In our code we use the following parameters (with no experimentation). $L = 2,\ S = 0.3,\ \lambda = 2,\ \mu = 0.1,\ \gamma = 0.3$. Note that increasing γ puts more emphasis on the statistical accuracy of the forecast and less on it's consequences, and increasing λ increases the risk aversion.

6 Initialization of NN weights

Since the transfer functions for the vertices in periods $1, 2, \ldots, \hat{p}$ are all logistic, they exhibit saturation of the output when the input signal is much greater than 1. What is therefore required, is for the initial random initialization of the weights to lead to vertex input signals with expected value equal to zero and variance of approximately 1.

6.1 For the first period

Equation **??** gives the total input signal into a vertex v_j. Let us assume that the arc weights w_{ij} for the arcs $(v_i, v_j), v_i \in V_0, v_j \in V_1$ are generated from a uniform distribution in the range $[-a, a]$. We will determine a suitable value for a. It is clear that the expected value of x_j is 0 regardless of the value of a. The variance of the w_{ij} is $a^2/3$, so the variance of x_j is approximately $n.a^2/3$. Setting this to 1, so as to satisfy the above requirement, leads to $a = \sqrt{\frac{3}{n}}$.

The vertex weights u_j for the vertices $v_j \in V_1$, can be set to zero. However, it is better for the conjugate gradient algorithm used for training, not to have equal weights. It is, therefore, better to generate the u_j randomly from the uniform distribution $[-b, b]$ where $b \ll a$. In version 1, we generate the weights as described above with $b = a/10$.

6.2 For all other periods

Referring to equation **??** again, and considering vertices $v_i \in V_p$ and $v_j \in V_{p+1}$, $1 \leqslant p < \hat{p}$, we know that the value of y_i has expected value 0.5. The variance of y_i is, say, λ^2. Because y_i lies in the range 0.5 ± 0.5, λ cannot be greater than 0.5. Because the variance of the input to v_i was required to be equal to 1, the output y_i varies (approximately) in the range 0.5 ± 0.25 and $\lambda \approx O(0.25)$.
Let us assume that the arc weights w_{ij} for the arcs $(v_i, v_j) : v_i \in V_p,\ v_j \in V_{p+1}$, $1 \leqslant p < \hat{p}$ are generated from a uniform distribution in the range $[-h, h]$. We will determine a suitable value for h. The variance of x_j is approximately $(0.25)^2|V_p|.h^2/3$. Setting this to 1, so as to satisfy the above requirement, leads to $h = \sqrt{\frac{48}{|V_p|}}$.

If $|V_p|$ is even, the expected value of x_j is zero, and as for the case of period 1, we generate the vertex weights u_j randomly from the uniform distribution $[-\ell, \ell]$ where $\ell \ll h$. In version 1, we generate the weights as described above with $\ell = h/10$.
If $|V_p|$ is odd, the expected value of x_j is 0.5, and we set $u_j = -0.5 + \epsilon_j$ where ϵ_j is generated randomly from the uniform distribution $[-\ell, \ell]$ where ℓ is as given above.