INFORMATICA, 2008, Vol. 19, No. 4, 477–486

Neural Network with Matrix Inputs

Povilas DANIUŠIS, Pranas VAITKUS
Vilnius University, Faculty of Mathematics and Informatics
Naugarduko 24, LT-03225 Vilnius, Lithuania
e-mail: povilas.daniusis@mif.vu.lt, vaitkuspranas@yahoo.com

Received: January 2008; accepted: April 2008

Abstract. In this paper we propose and analyze a multilayer perceptron-like model with matrix inputs. We applied the proposed model to the financial time series prediction problem, compared it with the standard multilayer perceptron model, and got fairly good results.

Keywords: neural networks, multilayer perceptron, matrix inputs, prediction, approximation.

1. Introduction and Motivation

In some regression and classification problems (for example image, textual data, multidimensional time series analysis) we need to operate with matrices. One of standard general approaches is to decompose the input matrix into the vector and work with it. Such decomposition has two disadvantages:

- It can remove an important information about an inner structure of the input matrix.
- In most cases, when the inputs are matrices, they are relatively high dimensional. If dimensionality of the input is high and cardinality of the training set is relatively low, we can face a small training sample problem.

In such cases we need other techniques to deal with the matrix inputs.

Cai *et al.* (2006) proposed a linear model $\hat{y}(X) = \mathbf{u}^T X \mathbf{v}$, where X – $m \times n$ matrix, $\mathbf{u}_i \in \mathbb{R}^m$ and $\mathbf{v}_i \in \mathbb{R}^n$ – weight vectors. From the definition we see, that in this model we need to estimate only $m + n$ parameters, and in the standard linear regression (if we represent X as $m \cdot n$ dimensional vector) we have to estimate $m \cdot n$ parameters. For example, if $m = n = 1000$, in the standard linear regression we need to estimate 1,000,000 parameters and in Cai's model – only 2000.

In this article we generalize Cai's linear model in the multilayer perceptron framework (Haykin, 1998), analyze some theoretical properties of this model, and apply it to the financial time series prediction problem.

2. The Model

The multilayer perceptron (MLP) with single hidden layer is defined by the following formula:

$$\hat{y}(x) = \sum_{i=1}^{N} \alpha_i \sigma\big(< \mathbf{w}_i, \mathbf{x} > + b_i \big), \tag{1}$$

where $\mathbf{w}_i \in \mathbb{R}^k$, α_i, $b_i \in \mathbb{R}$, $\sigma(.)$ is an activation function, $< .,. >$ – an inner product.

Sometimes MLP's with two or even more hidden layers are used. For example, an MLP with two hidden layers can be written as follows:

$$\hat{y}(x) = \sum_{i=1}^{N} \alpha_i \sigma\Bigg(\sum_{j=1}^{M_i} \beta_{i,j} \sigma\big(< \mathbf{w}_{i,j}, \mathbf{x} > + b_{i,j}\big) + c_i \Bigg). \tag{2}$$

In this article we introduce the model (and call it MNN model).

$$\hat{y}(X) = \sum_{i=1}^{N} \alpha_i \sigma\big(\mathbf{u}_i^T X \mathbf{v}_i + b_i\big), \tag{3}$$

where X is the input matrix, α_i, b_i are scalars, $\mathbf{u}_i$ and $\mathbf{v}_i$ are weight vectors, and $\sigma(.)$ is an activation function. This model is an adaptation of the MLP neural network with a single hidden layer and vector inputs to that with matrix inputs.

3. Parameter Estimation

Various techniques can be employed to estimate the parameters of model (3). In this section we will paraphrase to the MNN model two known training algorithms: the gradient descend and the extreme learning machines (Huang *et al.*, 2005).

Denote the training set by $T = (X_i, y_i)_{i=1}^{M}$, where X_i – $m \times n$ matrices (inputs) and y_i – scalars (outputs).

3.1. *Gradient Descend*

Using the gradient descend algorithm (Haykin, 1998) we minimize the mean squared error:

$$Err(..) = \frac{1}{2M} \sum_{(X,y)\in T} \big(\hat{y}(X) - y\big)^2. \tag{4}$$

For that we compute the partial derivatives of (4):

$$\frac{\delta Err(..)}{\delta \alpha_k} = \frac{1}{M} \sum_{(X,y)\in T} \big(\hat{y}(X) - y\big) \sigma\big(\mathbf{u}_k^T X \mathbf{v}_k + b_k\big), \tag{5}$$

$$\frac{\delta Err(..)}{\delta u_{k,l}} = \frac{\alpha_k}{M} \sum_{(X,y)\in T} \big(\hat{y}(X) - y\big) \sigma'\big(\mathbf{u}_k^T X \mathbf{v}_k + b_k\big) \sum_{i=1}^{n} v_{k,i} X_{l,i}, \tag{6}$$

$$\frac{\delta Err(..)}{\delta v_{k,l}} = \frac{\alpha_k}{M} \sum_{(X,y)\in T} \big(\hat{y}(X) - y\big) \sigma'\big(\mathbf{u}_k^T X \mathbf{v}_k + b_k\big) \sum_{j=1}^{m} u_{k,j} X_{j,l}, \tag{7}$$

$$\frac{\delta Err(..)}{\delta b_k} = \frac{\alpha_k}{M} \sum_{(X,y)\in T} \big(\hat{y}(X) - y\big) \sigma'\big(\mathbf{u}_k^T X \mathbf{v}_k + b_k\big), \tag{8}$$

since

$$\mathbf{u}_k^T X \mathbf{v}_k = \sum_{i=1}^{n} v_{k,i} \sum_{j=1}^{m} u_{k,j} X_{j,i} = \sum_{j=1}^{m} u_{k,j} \sum_{i=1}^{n} v_{k,i} X_{j,i}. \tag{9}$$

Algorithm

1. Fix the initial weights α_i, b_i, $\mathbf{u}_i$, and $\mathbf{v}_i$, number $\epsilon > 0$ and the learning rate $\eta > 0$, sufficiently large natural number N and $i = 0$;
2. By (5), (6), (7), (8) we compute weight changes:
   (a) $\Delta\alpha_k = -\eta \dfrac{\delta Err(..)}{\delta\alpha_k}$,
   (b) $\Delta u_{k,l} = -\eta \dfrac{\delta Err(..)}{\delta u_{k,l}}$,
   (c) $\Delta v_{k,l} = -\eta \dfrac{\delta Err(..)}{\delta v_{k,l}}$,
   (d) $\Delta b_k = -\eta \dfrac{\delta Err(..)}{\delta b_k}$;
3. $i := i + 1$;
4. Repeat Step 2 until the cost (4) is greater than ϵ and $i < N$.

3.2. *Extreme Learning Machines*

The gradient descend algorithm has many well known disadvantages (for example, it is quite slow, tends to stick in local extremums, a user should have enough experience to correctly estimate various parameters, like a learning speed η, desired error ϵ, etc. (Haykin, 1998)). Huang (2006) proposed simple yet effective technique, called Extreme Learning Machines to overcome these problems. The ELM learning consists of these two phases:

1. The internal weights $\mathbf{u}_i$, $\mathbf{v}_i$, and b_i, is initialized according to some probablistic distribution with zero mean and small variance (small weights imply better generalization (see Bartlet, 1998)).
2. The external weights α_i are calculated by the formula: $\mathbf{W} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{Y}$ (the least squares method), where

$$\mathbf{H} = \begin{pmatrix} \sigma(\mathbf{u}_1^T X_1 \mathbf{v}_1 + b_1) & \sigma(\mathbf{u}_2^T X_1 \mathbf{v}_2 + b_2) & \dots & \sigma(\mathbf{u}_n^T X_1 \mathbf{v}_n + b_n) \\ \sigma(\mathbf{u}_1^T X_2 \mathbf{v}_1 + b_1) & \sigma(\mathbf{u}_2^T X_2 \mathbf{v}_2 + b_2) & \dots & \sigma(\mathbf{u}_n^T X_2 \mathbf{v}_n + b_n) \\ \dots & \dots & \dots & \dots \\ \sigma(\mathbf{u}_1^T X_M \mathbf{v}_1 + b_1) & \sigma(\mathbf{u}_2^T X_M \mathbf{v}_2 + b_2) & \dots & \sigma(\mathbf{u}_n^T X_M \mathbf{v}_n + b_n) \end{pmatrix},$$

$$\mathbf{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_M \end{pmatrix}, \quad \mathbf{W} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \dots \\ \alpha_n \end{pmatrix},$$

and (X_i, y_i), $i = 1, \dots, M$ – the training set.

This learning method is very fast and simple. The numerical simulations show that in many cases it is more effective than the gradient descend learning algorithm (Huang *et al.*, 2005; Huang, 2006). The only parameter that a user must set by hand is the number of hidden units N. In the next section we discuss one algorithm proposed by Chen *et al.* (2005) which can be used to estimate N.

3.3. *Optimal Architecture*

Following the ELM theory we set an internal weights ($\mathbf{u}_i$, $\mathbf{v}_i$, and b_i) with a small pseudorandom numbers. Denote the training set by $T = (X_i, y_i)_{i=1}^{M}$, where X_i are $m \times n$ order matrices (inputs), y_i – scalars (outputs). We will find an optimal parameters α_i and order N of the (3) model ($\hat{y}(X) = \sum_{i=1}^{N} \alpha_i \sigma(\mathbf{u}_i^T X \mathbf{v}_i + b_i)$). Denote

$$y_i^{(0)} = y_i, \quad y_i^{(k)} = y_i^{(k-1)} - \alpha_k \cdot \sigma\big(\mathbf{u}_k^T X_i \mathbf{v}_k + b_k\big), \tag{10}$$

where $i = 1, 2, \ldots, M$.

Then $Err_k(..) = \frac{1}{2M} \sum_{i=1}^{M} (y_i^{(k)})^2 = \frac{1}{2M} \sum_{i=1}^{M} (\sum_{j=1}^{k} \alpha_j \sigma(\mathbf{u}_j^T X_i \mathbf{v}_j + b_j) - y_i)^2$ – mean squared error when model has k neurones.

We will start with the simplest model: a single neurone with matrix inputs ($N = 1$). By (11) (least squares method) we estimate the weight α_1 and check the mean squared error of the model. If the model is sufficiently good, then we stop the learning process. In an opposite case we add a new neurone ($N = 2$). Weight α_2 is also estimated by (11) . In that way, by adding new neurones we stop when the model is sufficiently good or too complex (N is too large):

$$\alpha_k = \frac{\sum_{i=1}^{M} y_i^{(k-1)} \cdot \sigma(\mathbf{u}_k^T X_i \mathbf{v}_k + b_k)}{\sum_{i=1}^{M} \sigma^2(\mathbf{u}_k^T X_i \mathbf{v}_k + b_k)}. \tag{11}$$

4. Questions about Density

In this section we will prove two propositions about approximation of a continuous function with (3) model.

It is well known that the multilayer perceptron with single hidden layer and arbitrary non-polynomial continuous activation function $\sigma(.)$ is a universal approximator in the space of the continuous functions (Pinkus, 1999). In a general case our model does not have this property. At first we will prove this proposition by applying Vostrecov–Kreines's theorem.

Theorem 1 (Vostrecov–Kreines, 1961). *Let $A \subset \mathbb{R}^k$. Then*

$$\mathcal{R}(A) = \left\{ f(\boldsymbol{x})\colon\ f(\boldsymbol{x}) = \sum_{i=1}^{n} g_i\big(< \boldsymbol{w}_i, \boldsymbol{x} > \big),\ \boldsymbol{w}_i \in A,\ g_i \in C(\mathbb{R}),\ n \in \mathbb{N} \right\} \tag{12}$$

is dense in $C(\mathbb{R}^k)$ in the topology of uniform convergence on compacta, if and only if there is no nontrivial homogenous polynomial that vanishes on A.

PROPOSITION 1. Let $K \subset \mathbb{R}^{m\times n}$ is a compact set, $S = \{f(X)\colon f(X) = \sum_{i=1}^{N} \alpha_i\sigma(\mathbf{u}_i^T X\mathbf{v}_i + b_i), X \in K\}$, and $\min(m,n) > 1$. Then S is not dense in $C(K)$ in the topology of uniform convergence on K.

Proof. If $\min(m,n) = 1$, we have a standard case (the MLP with one hidden layer and vector inputs), witch is proved to be a universal approximator (Pinkus, 1999)). So we need to check the case when $\min(m,n) > 1$.

Model (3) is a special case of model (1), since $\mathbf{u}^T X\mathbf{v}$, $\mathbf{u} \in \mathbb{R}^m$, $\mathbf{v} \in \mathbb{R}^n$ can be represented as an inner product <**w**, **x**>, where

$$\mathbf{w} = \big(v_1 \cdot \mathbf{u}, v_2 \cdot \mathbf{u}, ..., v_n \cdot \mathbf{u}\big) \tag{13}$$

and **x** – the vector, constructed by concatenation of all n columns of the matrix X. Take the subset of $\mathbb{R}^{m\times n}$

$$K := \Big(\mathbf{w} := \big(v_1 \cdot \mathbf{u}, v_2 \cdot \mathbf{u}, ..., v_n \cdot \mathbf{u}\big),\ \mathbf{u} \in \mathbb{R}^m,\ \mathbf{v} \in \mathbb{R}^n\Big). \tag{14}$$

Now we need to check the condition of the Vostrecov–Kreines theorem when $A = K$. We can look at K elements as into $n \times m$ matrices M with $rank(M) = 1$. But then, if we take submatrix of M of size $\min(m,n) \times \min(m,n)$, we see that its determinant equals to zero. This determinant is a homogenous $m \cdot n$ variable polynomial, that vanishes on K. Therefore, model (3) is not a universal approximator in the space of the continuous functions.

If we add the second hidden layer, MNN model is flexible enough to uniformly approximate any continuous function on the given compact set. We will prove the following proposition by applying Kolmogorov's superposition theorem (Pinkus, 1999) and one-dimensional result achieved by Pinkus (1999).

Theorem 2 (Kolmogorov's superposition theorem). *There exist N constants $\lambda_i > 0$, $i = 1, 2, \ldots, N$, $\sum_{i=1}^{N} \lambda_i \leqslant 1$ and $2N + 1$ strictly increasing continuous functions ϕ_i: $[0,1] \to [0,1]$, $i = 1, 2, \ldots, 2N + 1$ that every continuous function of N variables f: $[0,1]^N \to \mathbb{R}$ can be represented in the form*

$$f(x_1, x_2, ..., x_N) = \sum_{i=1}^{2N+1} g\Big(\sum_{j=1}^{N} \lambda_j \phi_i(x_j)\Big) \tag{15}$$

for some $g \in C[0,1]$ depending on f.

Theorem 3 (Pinkus, 1999). *Let $\sigma \in C(\mathbb{R})$, $\epsilon > 0$. Then for any $f \in C(\mathbb{R})$ and for any $K \subset \mathbb{R}$ real numbers α_i, λ_i and θ_i, number $n \in \mathbb{N}$ exist, that*

$$\sup_{t\in K}\left|f(t) - \sum_{i=1}^{n}\alpha_i\sigma\bigl(\lambda_i t - \theta_i\bigr)\right| < \epsilon, \tag{16}$$

if and only if σ is not a polynomial.

Theorem 4. *Let $K \subset R^{m\times n}$ – a compact set. Then for any $f\colon K \to \mathbb{R}$ exist constants d_i, $c_{i,j}$, $b_{i,j}$, γ_i, vectors $\boldsymbol{u}_{i,j} \in \mathbb{R}^m$ and $\boldsymbol{v}_{i,j} \in \mathbb{R}^n$, natural numbers K and K_i, such, that for any $\epsilon > 0$*

$$\sup_{\boldsymbol{X}\in K}\left|f(\boldsymbol{X}) - \sum_{i=1}^{K}d_i\sigma\Bigl(\sum_{j=1}^{K_i}c_{i,j}\sigma\bigl(\boldsymbol{u}_{i,j}^T X\boldsymbol{v}_{i,j} + b_{i,j}\bigr) + \gamma_i\Bigr)\right| < \epsilon, \tag{17}$$

where σ – any continuous, non-polynomial function.

Proof. Without the loss of generality we will prove the theorem for $K = [0,1]^{m\times n}$. Take $\epsilon > 0$ and any $f \in C([0,1]^N)$ $(N = m\times n)$. According to the Kolmogorov superposition theorem

$$f(x) = \sum_{i=1}^{2N+1} g\Bigl(\sum_{j=1}^{N}\lambda_j\phi_i(x_j)\Bigr), \tag{18}$$

where $x = (x_1, x_2, \ldots, x_N)$.

Since g is continuous, by Theorem 3 we have that for any compact set $S \subset R$ exist β_m, a_m and b_m, that

$$\sup_{t\in S}\left|g(t) - \sum_{m=1}^{M}\beta_m\sigma(a_m t + b_m)\right| < \frac{\epsilon}{2(2N+1)}. \tag{19}$$

For convenience we denote that $\phi_{i,j}(X) = \phi_i(x_j)$. Again, by Theorem 3 we have that for any $\delta > 0$ the following inequality is correct:

$$\sup_{X\in[0,1]^N}\left|\phi_{i,j}(X) - \sum_{k=1}^{n_{i,j}}\alpha_{k,i,j}\sigma\bigl(\mathbf{u}_{k,i,j}^T X\mathbf{v}_{k,i,j} + \theta_{k,i,j}\bigr)\right| < \frac{\delta}{|a_m|}, \tag{20}$$

since by the transformation $\mathbf{u}^T X\mathbf{v} + \theta$ we can access any linear combination $\lambda x_{i,j} + \theta$ of any X element $x_{i,j}$.

Because constants of the Kolmogorov superposition theorem $\lambda_i > 0$ and $\sum_{i=1}^{N}\lambda_i \leqslant 1$ we have

$$\sup_{X\in[0,1]^N}\left|\sum_{j=1}^{N}\lambda_j\phi_{i,j}(X) - \sum_{j=1}^{N}\lambda_j\sum_{k=1}^{n_{i,j}}\alpha_{k,i,j}\sigma(\mathbf{u}_{k,i,j}^T X\mathbf{v}_{k,i,j} + \theta_{k,i,j})\right| < \frac{\delta}{|a_m|}.$$

Since σ is continuous, it is uniformly continuous in any closed interval, therefore, we can chose δ small enough, that

$$\begin{aligned}\sup_{X\in[0,1]^N}\bigg|&\sigma\Big(a_m\Big\{\sum_{j=1}^{N}\lambda_j\phi_{i,j}(X)\Big\}+b_m\Big)\\&-\sigma\Big(a_m\Big\{\sum_{j=1}^{N}\lambda_j\sum_{k=1}^{n_{i,j}}\alpha_{k,i,j}\sigma\big(\mathbf{u}_{k,i,j}^TX\mathbf{v}_{k,i,j}+\theta_{k,i,j}\big)\Big\}+b_m\Big)\bigg|\\&<\frac{\epsilon}{2(2N+1)\sum_{m=1}^{M}|\beta_m|}.\end{aligned}$$

From these inequalities we have

$$\begin{aligned}\sup_{X\in[0,1]^N}\bigg|&\sum_{i=1}^{2N+1}g\Big(\sum_{j=1}^{N}\lambda_j\phi_{i,j}(X)\Big)\\&-\sum_{i=1}^{2N+1}\sum_{m=1}^{M}\beta_m\sigma\Big(a_m\sum_{j=1}^{N}\lambda_j\sum_{k=1}^{n_{i,j}}\alpha_{k,i,j}\sigma(\mathbf{u}_{k,i,j}^TX\mathbf{v}_{k,i,j}+\theta_{k,i,j})+b_m\Big)\bigg|\\&\leqslant\sum_{i=1}^{2N+1}\sup_{X\in[0,1]^N}\bigg|g\Big(\sum_{j=1}^{N}\lambda_j\phi_{i,j}(X)\Big)-\sum_{m=1}^{M}\beta_m\sigma\Big(a_m\sum_{j=1}^{N}\lambda_j\phi_{i,j}(X)+b_m\Big)\bigg|\\&+\sum_{i=1}^{2N+1}\sup_{X\in[0,1]^N}\bigg|\sum_{m=1}^{M}\beta_m\sigma\Big(a_m\sum_{j=1}^{N}\lambda_j\phi_{i,j}(X)+b_m\Big)\\&-\sum_{m=1}^{M}\beta_m\sigma\Big(a_m\sum_{j=1}^{N}\lambda_j\sum_{k=1}^{n_{i,j}}\alpha_{k,i,j}\sigma\big(\mathbf{u}_{k,i,j}^TX\mathbf{v}_{k,i,j}+\theta_{k,i,j}\big)+b_m\Big)\bigg|\\&<(2N+1)\frac{\epsilon}{2(2N+1)}+(2N+1)\sum_{m=1}^{M}|\beta_m|\frac{\epsilon}{2(2N+1)\sum_{m=1}^{M}|\beta_m|}=\epsilon.\end{aligned}$$

Therefore, f can be uniformly approximated by MNN model with two hidden layers.

PROPOSITION 2 (Pinkus, 1999). Let $\sigma\in C(\mathbb{R})$. If the input $\mathbf{x}\in\mathbb{R}^k$ is converted to $\mathbf{h}(\mathbf{x})=(h_1(\mathbf{x}),h_2(\mathbf{x}),\dots,h_m(\mathbf{x}))$ for some fixed functions $h_j\in C(\mathbb{R}^k)$, then set

$$\mathcal{F}_{\mathbf{h}}(\sigma)=\Big\{f(x)\colon\ f(x)=\sum_{i=1}^{N}\alpha_i\sigma\big(<\mathbf{w}_i,\mathbf{h}(\mathbf{x})>+b_i\big)\Big\}\tag{21}$$

is dense in the topology of a uniform convergence on compacta in $C(\mathbb{R}^m)$ if and only if σ is not a polynomial and $\mathbf{h}$ separates points, that is, $\mathbf{x}_1\neq\mathbf{x}_2$ implies $\mathbf{h}(\mathbf{x}_1)\neq\mathbf{h}(\mathbf{x}_2)$.

Note, that if an activation function $\sigma(.)$ also has an inverse, proposition of the Theorem 4 follows from an universal approximation property of a MLP with a single hidden layer and the Proposition 2. In Theorem 4 we considered a more general case when an activation function is an arbitrary continuous, non-polynomial function.

5. Computer Experiment

Various methods (Raudys and Mockus, 1999) are used for the economic time series forecasting. In this section we will use the MNN model to predict the next day value of the Euro (EUR). The purpose of this section is to compare the model (3) and the standard MLP model with various settings and training algorithms.

5.1. *The Data*

We use the high frequency financial data. The open, high, low, and close prices of the EUR are observed every 5 minutes from 2007-01-07 22:35:00 to 2007-01-09 16:11:00. For convenience we preprocess the original data by subtracting the mean and dividing by the standard deviation.

The input matrix at the time step t is defined by

$$X_t = \left[x_{open}(k), x_{high}(k), x_{low}(k), x_{close}(k)\right]_{k=t}^{t+w-1}, \tag{22}$$

where w is the window parameter.

The target value for this matrix is defined by $y_t = x_{close}(t+w)$. The training set is defined by $S_{training} = (X_t, y_t)_{t=1}^{300}$ and the testing set is defined by $S_{testing} = (X_t, y_t)_{t=301}^{493}$.

5.2. *Prediction Algorithms*

We will use four prediction algorithms:

- MNN(ELM) – the MNN model, trained with the ELM algorithm.
- MNN(GD) – the MNN model, trained with the gradient descend algorithm.
- NN(ELM) – the standard MLP model, trained with the ELM algorithm.
- NN(GD) – the standard MLP model, trained with the gradient descend algorithm.

5.3. *Experiment*

The measure of performance we use is the sum squared error (SSE) over the testing set. All algorithms are trained to minimize the regularized mean squared error (RMSE):

$$RMSE(..) = MSE(..) + \lambda \cdot \sum_{\theta} \theta^2 \tag{23}$$

(second sum is over all parameters of the model) with regularization parameter $\lambda = 0.05$. For the models, trained with gradient descend algorithm, the target RMSE is 0.07. We set window parameter $w = 6$. Each prediction algorithm is trained and tested 100 times.

5.4. *Comparison of the Prediction Models*

The Tables A and B show the average SSE over the test set for the NN($\cdot$) and MNN($\cdot$) prediction algorithms with various settings. The differences of the predictions are statistically significant with the p-values in the "P" column (low p-values indicate the statistical significance).

From the Tables A and B we see that the MNN model gives better predictions than the NN model. The input matrices clearly have an inner structure, since columns (high, low, open, and close prices of the EUR) are strongly correlated (in our data all correlation coefficients between columns are > 0.97). We think, this is the reason why the MNN model generalizes better on this data. MNN model have significantly less parameters and we need less training examples to estimate them. When the ELM training algorithm is used we need much more hidden units, since we have access only to the linear parameters of the model (see subsection 3.2). From the Tables A and B we see that the much faster ELM training algorithm in most cases also predicted better than the GD algorithm. As pointed by referee, many sophisticated time series prediction methods are outperformed by "naive" hypothesis (prediction for the next period equals the value of the current period). We tested for this, and found that for our Euro data the "naive" hypothesis performed better (SSE $= 8.57$) than NN or MNN. We also tested the standard linear regression (SSE $= 8.53$) and Cai's linear regression with matrix inputs – $\hat{y}(X) = \mathbf{u}^T X \mathbf{v}$ (Cai *et al.*, 2006), which can be interpreted as MNN model with one neurone and linear activation function (SSE $= 8.23$). The prediction of Cai's model was best (smallest SSE over the test set). Thus we can conclude, that matrix-based models performed better than

Table A

Comparison of the MNN(ELM) and NN(ELM)

Hidden units	MNN(ELM)	NN(ELM)	P
10	14.60	20.62	0.01
20	11.76	15.74	0.01
30	11.06	14.38	0.01
40	11.14	14.06	0.01
50	11.48	14.09	0.01

Table B

Comparison of the MNN(GD) and NN(GD)

Hidden units	MNN(GD)	NN(GD)	P
1	11.12	19.20	0.01
2	12.89	23.34	0.01
3	13.56	16.05	0.07
4	11.56	15.00	0.01
5	12.61	13.92	0.01

their vector-based analogues. This confirms the guess about importance of the inner structure of the data. The nature of our data seems to be linear, since linear models performed better than non-linear ones.

References

Bartlett, P.L. (1998). The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. In *IEEE Transactions on Information Theory*. pp. 522–536.

Cai, D., X. He and J. Han (2006). *Learning with Tensor Representation*. Preprint.

Chen, S., X.X. Wang and D.J. Brown (2005). Sparse incremental regression modeling using correlation criterion with boosting search. *IEEE Signal Processing Letters*, **12**(3), 198–201.

Haykin, S. (1998). *Neural Networks*: *A Comprehensive Foundation*. 2nd Edition. Prentice Hall.

Hornik, K., M. Stinchcombe and H. White (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 359–366.

Huang, G.-B. (2006). Universal approximation using incremental constructive feedforward networks with random hidden nodes. In *IEEE Transactions on Neural Networks*. pp. 879–892.

Huang, G.-B., N.-Y. Liang, H.-J. Rong, P. Saratchandran and N. Sundararajan (2005). On line sequential extreme learning machine. In *The IASTED International Conference on Computational Intelligence* (*CI 2005*). Calgary, Canada, July 4–6.

Pinkus, A. (1999). Approximation theory of the MLP model in neural networks. *Acta Numerica*, 143–195.

Raudys, A., and J. Mockus (1999). Comparison of ARMA and multilayer perceptron based methods for economic time series forecasting. *Informatica*, **10**(2), 231–244.

Zuo, W.-M., K.-Q. Wang and D. Zhang (2006). Assembled matrix distance metric for 2DPCA-based face and palmprint recognition. *Pattern Recognition Letters*.

P. Daniušis is a PhD student at Vinius University, Faculty of Mathematics and Informatics. His research interests include machine learning, theoretical and practical aspects of neural networks.

P. Vaitkus is an associate professor at Vilnius University, Faculty of Mathematics and Informatics. His research interests include automated biosensor response analysis, evolutionary algorithms and neural networks.

Neuroninis tinklas su matriciniais įėjimais

Povilas DANIUŠIS, Pranas VAITKUS

Šiame darbe pasiūlytas daugiasluoksnio perceptrono tipo modelis su matriciniais įėjimais, šio modelio parametrų radimo procedūros, ištirtos pasiūlyto modelio aproksimavimo savybės, modelis palygintas su standartiniu daugiasluoksniu perceptronu prognozuojant euro kursą.