University of Southern Denmark

Artificial Intelligence project - AI4

Robotics 2018

# Predicting Bitcoin Price Series using Artificial Neural Networks

Ahmad Gheith
ahghe13@student.sdu.dk

**Supervisor:** John Hallam

**Abstract** - In this work, I aim to predict Bitcoin price series using two types of Artificial Neural Networks. The first method is Feedforward Neural Network, where I use Sequential Forward Feature Selection algorithm to optimize the input of the Network. The second method is Echo State Network, which is an architecture and learning principle for Recurrent Neural Networks. Both methods are fed with historical data regarding open prices and 24h trading volumes. The performances of the Networks is evaluated using the Mean Squared Error method, where the goal was to minimize the error.

## 1 Introduction

Artificial Intelligence (AI) is a rapidly expanding field. A huge amount of time and money are being invested in research and method developing. As of now, countless AI applications has been created already. AI will doubtlessly affect our daily life and the modern society. It will creep into every field, such as health care, education, industry, marketing, toys, games, finance etc. In fact, in all these fields, the essence of AI is already found.
In this project, I will apply AI to a finance challenge, i.e. predicting the price series of Bitcoin. More concrete, I will aim to predict the price one day ahead.
The methods used to achieve the goal are described in section 2. The obtained results are described in section 3 and the final conclusion are to be found in section 4.

## 2 Methods

In this project, I have applied two Artificial Intelligence techniques in order to predict the Bitcoin price series. The techniques are Feedforward Neural Network and Echo State Neural Network. As their names indicate, they are both based on Artificial Neural Networks (ANN).
An ANN consists of a number of nodes, inspired by the biological neuron system. These nodes can be connected in different ways, which greatly impacts the behaviour and ability of the network and is the core difference between the applied techniques.
Before moving on, the way of perceiving the data must be made clear. Let $data(i)$ be the extracted data from the $i$'th day, and let $type_j(i)$ be the $j$'th data type value of that day, such that

$$data(i) = \begin{bmatrix} type_1(i) & type_2(i) & ... & type_M(i) \end{bmatrix}$$

where $i = 1, 2, ..., N$ and $j = 1, 2, ..., M$, thus $data \in \mathbb{R}^{N \times M}$. The data types could for instance be the open price, 24h trading volume, 24h minimum price, 24h maximum price etc.
The performance of a networks was assessed using Mean Square Error (MSE), which is expressed as

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{1}$$

where $n$ is the size of the predicted data, $y_i$ is the actual price on the $i$'th day and $\hat{y}_i$ is the prediction of the network of the price on day $i$ made the day before, i.e. day $(i - 1)$.

### 2.1 Feedforward Neural Network

A Feedforward Neural Network (FFNN) is a relatively simple ANN, where all the nodes are connected in the forward direction, see Figure 1.

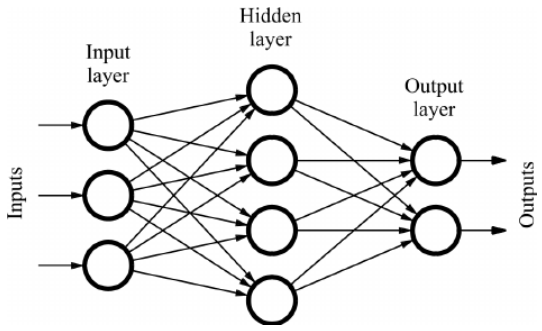


**Figure 1** – General architecture of Feedforward Neural Network

The simplification of only connecting the nodes in one direction decreases the complexity of the learning process, by simplifying the back-propagation of the error. However, this does not come without a cost, as it implies that the network does not have a reservoir, which means it is unable to use previous computations to compute future results. In other words, it has no memory.
An FFNN consists of an input layer, a hidden layer, and an output layer, as shown in Figure 1. The number of hidden layers can vary depending on the application. In simpler cases, no hidden layer is

required at all. Theoretically, the impact of varying the number of hidden layers can almost always be achieved by adjusting the number of neurons in a single hidden layer. In order to the reduce the dimensionality of the problem, a single hidden layer is used.
As the network lacks memory, the data required to predict the future of the price series must be provided through the input layer. Selecting which data to feed into the FFNN was determined using Sequential Forward Feature Selection (SFFS) algorithm, which is going to be explained shortly. The number of input neurons must fit the size of data determined by SFFS.
The size of the output layer depends on the desired prediction. One might be interested in predicting the price series for a several days ahead, which will also require several output neurons, $n_o$. However, as we are exclusively interested in predicting the price one day ahead, a single output neuron will do the job.
The number of hidden neurons, $n_h$, affects the performance of the Network. As a rule of thumb, $n_h$ should be somewhere between the number of input and output neurons. Here, the method of Shibata and Ikeda [1] has been adopted, which suggests:

$$n_h = \sqrt{n_i \cdot n_o} \tag{2}$$

For both, applying SFFS and training the final network, the Levenberg-Marquardt algorithm was used for training. The method is explained in detail in [2].
In order to avoid overfitting, the data was splitted into three subsets: Training subset, testing subset, and validation subset. The training subset was used to train the network. The testing subset was used to evaluate the performance of the network by computing the MSE, and the trainig was stopped once a minima point was reached. The validation set was used only to compute the final MSE of the network.

### 2.1.1 Sequential Forward Feature Selection

In order to predict the price series using FFNN, a window of the data from previous days has to be used. Let $p$ be the present day and let $w \in \mathbb{R}^{w_{size} \times M}$ be the window, where

$$w = \begin{bmatrix} w_{w_{size}-1} \\ w_{w_{size}-2} \\ \vdots \\ w_0 \end{bmatrix} = \begin{bmatrix} data(p-(w_{size}-1)) \\ data(p-(w_{size}-2)) \\ \vdots \\ data(p) \end{bmatrix}$$

where $w_{size}$ is the window size.
The motivation behind applying SFFS is that the network might perform better by only using a subset of the matrix $w$. SFFS is used to explore that. The data in $w$ is indexed in the vector $F$, such that

$$F = \begin{bmatrix} 0 & 1 & ... & w_{size} \cdot M - 1 \end{bmatrix}$$

where the first element in $F$ is the index of the last data type in the data of the present day, i.e. $type_M(p)$, the second element is the index of the second last data type and so on. The last element is

an index for $type_1(w_{size} - 1)$. The goal of SFFS can now be defined as finding a subset, $C$, of $F$ that optimizes the performance of the Network. SFFS is explained in Algorithm 1.

**Algorithm 1:** Sequential Forward Feature Selection

**Result:** Finds a subset, $C$, of $F$ that optimizes the Network performance

```
bestPerf ← ∞;
F ← {0, 1, ..., (w_size · M − 1)};
C ← {};
for s in F do
    V = [];
    for t in F do
        if F(t) is not in C then
            Construct net with the input C ∪ F(t);
            Train net;
            pNet ← performance of net;
            V ← [V; pNet, F(t)];
        end
    end
    b ← index of the row in V containing the best performance;
    if Performance of V(b) is better than bestPerf then
        bestPerf ← Performance of V(b);
        C ← C∪(element of V(b)) ;
    else
        done;
    end
end
```

SFFS is a greedy algorithm. However, compared to Brute Force, it reduces the number of input combinations from $2^{(w_{size}M)}$ to $w_{size}M \cdot (NM + 1)/2$, which is a necessary reduction in order to make it possible to find a good solution within a reasonable amount of time using a reasonable window size and several number of data types.

## 2.2 Echo State Network

Feedforward Neural Networks are the most common type of ANN due to their implementational and computational simplicity. However, comparing them to biological neurons, they are quite different, as biological neurons are usually cross-connected. The sub-type of ANNs that deals with cross-connected neurons are referred to as Recurrent Neural Networks (RNNs). Here, I am going to apply an RNN-method, which is called Echo State Network (ESN).

An ESN consists of three parts: Input layer, reservoir pool, and output layer, see Figure 2.

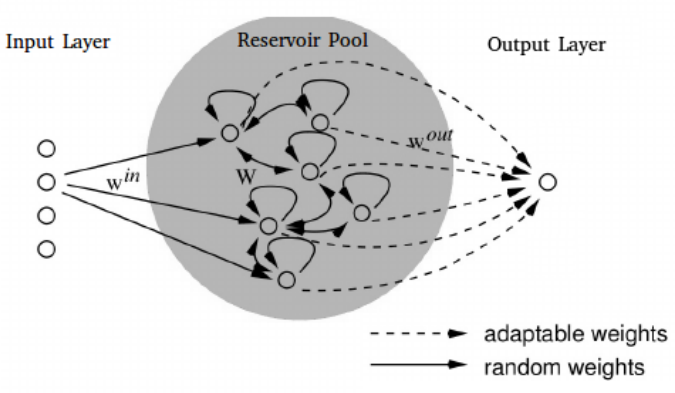


**Figure 2** – General architecture of Echo State Networks

The input layer contains a number of input neurons, $n_{in}$, that matches the desired input data size. For the ESN, this is simply the number of extracted data each day, which is $M$. Similarly, the output layer contains a number of output neurons, $n_{out}$. In our case $n_{out} = 1$ as the desired output is exclusively the prediction of the price of tomorrow.
The reservoir pool contains a number of randomly interconnected neurons, $n_{res}$. These neurons are connected to both input and output neurons.
The ESN model can be expressed using 3 weight matrices: $W_{in}$, $W$, and $W_{out}$, also shown in Figure 2. Please notice that capital $W$s are used here, in order to avoid confusion between $w$ for window (related to FFNNs) and $W$ for weights.
$W_{in}$ contains the weights of the connections between the input layer and the reservoir pool, in addition to a bias, thus $W_{in} \in \mathbb{R}^{n_{res} \times (n_{in}+1)}$. $W$ contains the weights of the neurons in the reservoir pool, thus $W \in \mathbb{R}^{n_{res} \times n_{res}}$. The last weight matrix, $W_{out}$, contains the weights of the connections between the output layer and both input layer and reservoir pool, thus $W_{out} \in \mathbb{R}^{n_{out} \times (n_{in}+1+n_{res})}$. All the weight matrices are randomly initialized between $-0.5$ and $0.5$ uniformly distributed. In order to achieve a stable network, the weights in the reservoir pool has been normalized using using Spectral Radius [3]. Using these definitions, the activations of the reservoir neurons can be computed as

$$x(i) = (1 - \alpha)x(i-1) + \alpha tanh(W_{in}[1; data(i)^T] + Wx(i-1)) \tag{3}$$

where $x(i) \in \mathbb{R}^{n_{res}}$, and $\alpha$ is the leaking rate. Notice that $x(0) := [0, 0, ..., 0]^T$.
Now, training such a network by updating all the weights is quite difficult. What is done instead is to train the weights in $W_{out}$ only. This makes the performance of the network dependent on the randomly initialized and untrained weights in $W_{in}$ and $W$. However, by making $n_{res}$ sufficiently large, the probability of having a good combination of neurons in the reservoir pool increases and hopefully, a good network will be achieved.
In order to train and test the network, the data set is split into 3 sub-sets: Initializing sub-set of size $d_{init}$, training sub-set of size $d_{train}$, and testing sub-set of size $d_{test}$, thus $d_{init} + d_{train} + d_{test} = N$.
The initializing sub-set is used to initialize $x(i)$, simply by computing $x(i)$ for $i = 1, 2, ..., d_{init}$. The purpose of this is to compensate for $x(0)$.

The training subset is used to train the network. This is achieved by computing the matrix $X \in \mathbb{R}^{(1+n_{in}+n_{res})\times d_{train}}$ whose columns are composed of a 1 (for bias), $data(i)^T$, $x(i)$ for $i = d_{init}+1, d_{init}+2, ..., d_{init}+d_{train}$. Now, the predictions for each day in the matrix $X$, say $\hat{Y}$, can be computed by

$$\hat{Y} = W_{out} \cdot X \tag{4}$$

Applying Moore-Penrose Inverse on X and substituting the predictions, $\hat{Y}$, by the actual prices we are trying to predict, $Y$, we can train the network by computing the output weights, $W_{out}$, that minimizes the error between $\hat{Y}$ and $Y$ as follows

$$W_{out} = Y \cdot X^{+} \tag{5}$$

where $Y \in \mathbb{R}^{1\times(d_{train})}$ contains the prices from day $d_{init}+2$ to $d_{train}+1$.
The last subset is used to test the network by making it predict the price series and comparing the prediction to the actual price of the next day. After computing $W_{out}$, a single-day prediction can be made by

$$\hat{y}(i) = W_{out} \cdot x(i) \tag{6}$$

# 3 Results and Discussion

The data used for this project was acquired from `data.bitcoinity.org`. It extends over the period from 17th of September 2011 to 28th of September 2017. For each day in the period, the data set contains information about the Open Price (OP) and the 24h Trading Volume (TV). The dimensions of the matrix *data* is shown in Table 1.

**Table 1** – Data Dimensions.

| | Notation | Value |
|---|---|---|
| No. of days | N | 2183 |
| No. of data types | M | 2 |

The methods were implemented in MATLAB and can be found at: `github.com/ahghe13/AI4`

## 3.1 Feedforward Neural Network

The adjustable parameters for the FFNN were the window size ($w_{size}$), and the sizes of the training subset, testing subset, and validation subset. The window size was chosen such that it was possible to compute the SFFS repeatedly in a reasonable amount of time. The size of the 3 subsets were sat according to the default configuration in MATLAB. Table 2 lists the parameters of the FFNN and their respective values.

**Table 2** – Parameters of the Feedforward Neural Network.

| Parameter | Notation | Value |
|---|---|---|
| No. of input neurons | $n_i$ | Depends on SFFS: 2 |
| No. of hidden neurons | $n_h$ | $\sqrt{n_i n_o}$ |
| No. of output neurons | $n_o$ | 1 |
| Window size | $w_{size}$ | 10 |
| Training subset | - | 70% of $(N - w_{size}) = 1521$ |
| Testing subset | - | 15% of $(N - w_{size}) = 325$ |
| Validation subset | - | 15% of $(N - w_{size}) = 325$ |

The number of input neurons, $n_i$, is determined by SFFS, which is dependent of the window size. The size of the hidden layer was computed using the method of Shibata and Ikeda [1] and the size of the output layer is fixed to 1 due the goal of the project.

### 3.1.1 Sequential Forward Feature Selection

As mentioned earlier, SFFS outputs a subset, $C$, of $F$, that optimizes the network. However, the output sequence, $C$, was not consequent. Therefore, SFFS was repeatedly run, in order to find and hopefully adopt the most frequently occurring sequence.
SFFS was run 500 times. A total of 190 different sequences were outputted. Figure 3 shows the frequency of the occurrence of each sequence. Please notice that the sequences are indexed in the figure.

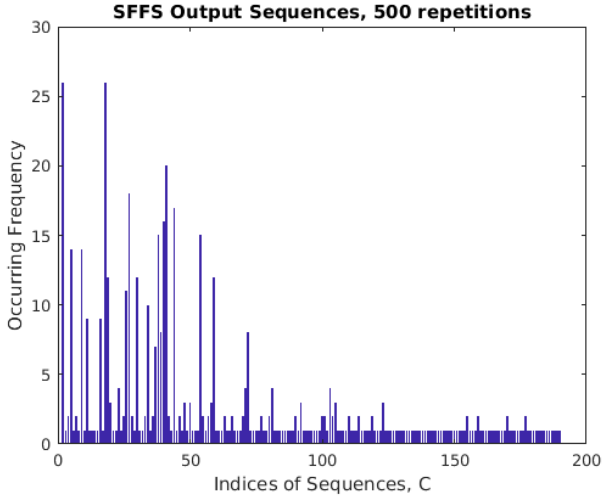


**Figure 3** – Running SFFS 500 times, a total of 190 different C sequences were outputted. The histogram shows the frequency of the occurring of each sequence.

The bars of the indices 2 and 18 in Figure 3 are clearly taller than the others. They both have an occurring frequency of 26 times. The sequence with the index 41 is is next tallest. It has an occurring frequency of 20. These three sequences are shown in Table 3.

**Table 3** – Index, occurring frequency, sequence, and sequence interpretation of the 3 most frequently outputted sequences by SFFS. OP is Open Price.

| Index | Occurring frequency | Sequence | Sequence interpretation |
|---|---|---|---|
| 2 | 26 | {1} | {OP today} |
| 18 | 26 | {1,3} | {OP today, OP 1 day ago} |
| 41 | 20 | {1,9} | {OP today, OP 4 days ago} |

The results shown in Table 3 are disappointing. The sequence at index 2 is expressing that it is possible to predict the price of tomorrow using only information about the open price of today. The open price of today might be the most influential parameter, but using that solely, the prediction cannot be better than a linear model. As we are aiming at a better model, this sequence will be discarded.
The sequence at index 18 is also using the Open Price of today in addition to the Open Price of yesterday. This seems a bit better than the previous sequence, especially because they have achieved the same occurring frequency.
The last sequence at index 41 is using same amount of information as the sequence at index 18. However, as sequence 41 has obtained a lower occurring frequency, we will apply the sequence at index 18 for the final network.
The results shown in Table 3 suggests that the 24h trading volume plays a minor role in the prediction process, as none of the top 3 prediction sequences uses the trading volume at all.

### 3.1.2 Final Feedforward Neural Network

The final FFNN was created by constructing a network using the obtained results and parameters so far. The final network had an $MSE$ of 869.03. Its ability to predict the Bitcoin price series is illustrated in Figure 4.

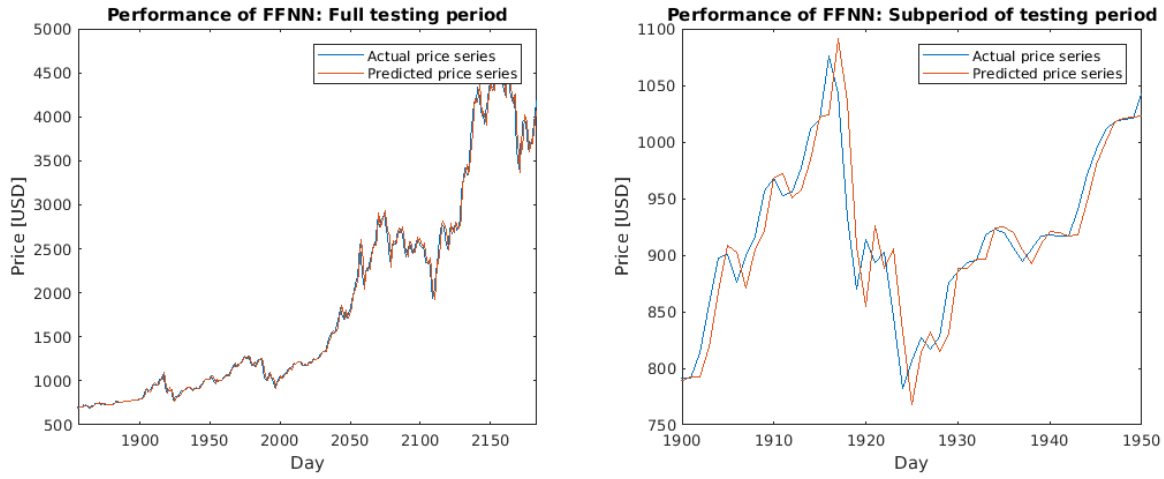


**Figure 4** – Predicted vs. actual Bitcoin price series. Left is the full testing subset of the data shown. Right is a subset of the testing subset shown.

The graphs in Figure 4 contains two curves each: One for the actual price series (Blue) and one for the predicted price series (orange).

The graph at the left shows the predictions of the full testing subset. This subset has not been used for training, nor for $MSE$ optimization. The match between the predicted and the actual price seems perfect. However, as we are only predicting one day ahead, we have to take a closer look before we can judge the performance of the network.
A closer look at the graph is shown on the right side of the figure. Here, it is clear that the predicted prices are oscillating around the actual prices. By analyzing it more carefully, the predictions seems to be shifted a day behind the actual price. In other words, the network is guessing that the price of tomorrow is almost similar to the price of today.

### 3.2 Echo State Network

The number of Input and Output Neurons were fixed. The number of reservoir neurons, the leaking rate, the initialization length, the training length and testing length were adjustable parameters. They were manually tuned and adjusted to minimize the MSE. A full list of the parameter values are shown in Table 4. Using these parameters, the ESN was constructed.

**Table 4** – Parameters of the Echo State Network.

| Parameter | Notation | Value |
|---|---|---|
| No. of Input Neurons | $n_{in}$ | 2 |
| No. of Reservoir Neurons | $n_{res}$ | 300 |
| No. of Output Neurons | $n_{out}$ | 1 |
| Leaking rate | $\alpha$ | 0.5 |
| Initialization Length | $d_{init}$ | 50 |
| Training Length | $d_{train}$ | 40% of $(N - d_{init}) = 853$ |
| Testing Length | $d_{test}$ | 60% of $(N - d_{init}) = 1279$ |

The MSE of the ESN was 1959.23. Compared to the FFNN, its $MSE$ is much bigger. The network's ability to predict the price series is illustrated in Figure 5.

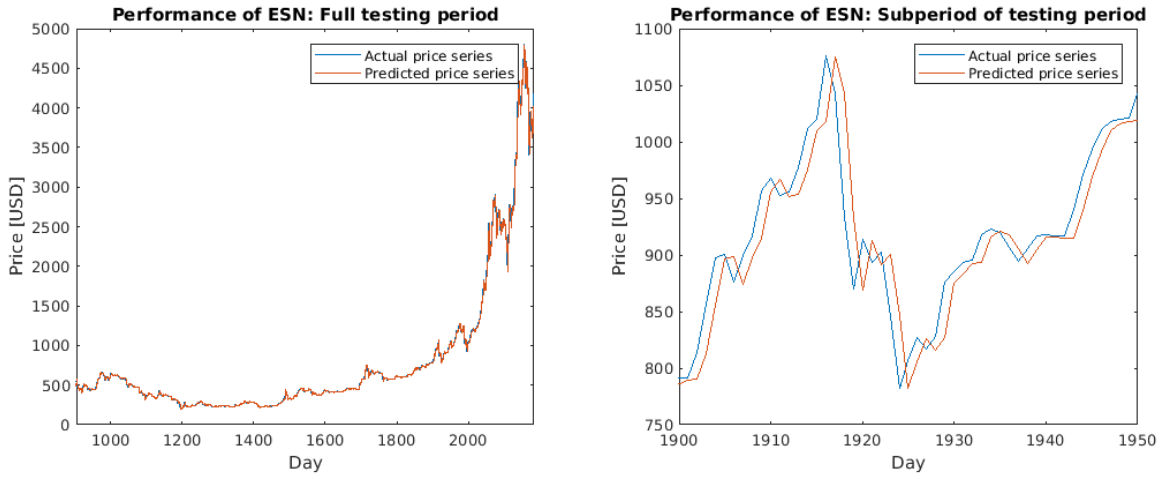


**Figure 5** – Predicted vs. actual Bitcoin price series. Left is the full test period show (from $d_{init} + d_{train}$ to the end of the data). Right is sub-period shown.

Figure 5 is similar to Figure 4, except that its data was produced by the ESN. Overall, the predictions seem to be catching up with the actual price pretty well. However, looking at the sub-period, it seems to be shifted one day. Here, the phenomenon is even more clear than the results for FFNN.
A reason why both networks are unable to predict the price series significantly accurate might simply be that the price series is unpredictable using information about Open Price and Trading Volume alone. The data we are trying to predict comes from a highly dynamic and sophisticated environment, and is influenced by various factors.
Another thing that should be considered is the performance evaluation method. Applying MSE as a performance heuristic was not be optimal, as a network that predicts accurately, but makes a few large mistakes, will be classified as a bad network, even though it is more useful than a network that always makes small mistakes.

# 4 Conclusion

In this project, I have successfully constructed a Feedforward Neural Network and an Echo State Network for predicting Bitcoin price series.
The input of the Feedforward Neural Network was determined using Selective Forward Feature Selection, which suggested that the minimum Mean Squared Error can be obtained by feeding the network with information about the Open Prices of the prediction day and the day before. The Mean Squared Error of the network was 869.03. Its predictions did not look sufficiently accurate, and the network is therefore not really useful.
The Echo State Network was constructed using hand-tuned parameters. The input of the network was the open price and 24h trading volume. The Mean Squared Error of the Echo State Network was 1959.23. Its predictions did not seem sufficiently accurate either, as the network predicted that the price of tomorrow will be the same as the price of today.
Comparing the two networks, the Feedforward Neural Network performed a bit better. Its predictions was bit less following the price of the prediction day, and its Mean Squared Error was a lower.

# 5 References


[1] Katsunari Shibata, Yusuke Ikeda, *"Effect of number of hidden neurons on learning in large-scale layered neural networks"*, 2009.

[2] Henri P. Gavin, *"The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems"*, 2017.

[3] Wang Yuanbiao, Jun Ni, Xu Zhiping, *"Effects of Spectral Radius on Echo-State-Network's Training"*, 2009.