Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Thesis for the Degree of Master of Science in Computer Science with Specialization in Embedded Systems 30.0 credits

Real-time Process Modelling Based on Big Data Stream Learning

Fan He
fhe15001@student.mdh.se

Examiner: Thomas Nolte

Supervisor: Ning Xiong

May 17, 2017

Table of Contents

Abstract

Most control systems now are assumed to be unchangeable, but this is an ideal situation. In real applications, they are often accompanied with many changes. Some of changes are from environment changes, and some are system requirements. So, the goal of thesis is to model a dynamic adaptive real-time control system process with big data stream. In this way, control system model can adjust itself using example measurements acquired during the operation and give suggestion to next arrival input, which also indicates the accuracy of states under control highly depends on quality of the process model.

In this thesis, we choose recurrent neural network to model process because it is a kind of cheap and fast artificial intelligence. In most of existent artificial intelligence, a database is necessity and the bigger the database is, the more accurate result can be. For example, in case-based reasoning, testcase should be compared with all of cases in database, then take the closer one's result as reference. However, in neural network, it does not need any big database to support and search, and only needs simple calculation instead, because information is all stored in each connection. All small units called neuron are linear combination, but a neural network made up of neurons can perform some complex and non-linear functionalities. For training part, Backpropagation and Kalman filter are used together. Backpropagation is a widely-used and stable optimization algorithm. Kalman filter is new to gradient-based optimization, but it has been proved to converge faster than other traditional first-order-gradient-based algorithms.

Several experiments were prepared to compare new and existent algorithms under various circumstances. The first set of experiments are static systems and they are only used to investigate convergence rate and accuracy of different algorithms. The second set of experiments are time-varying systems and the purpose is to take one more attribute, adaptivity, into consideration.

Keywords: *control system, real-time process, deep learning, recurrent neural network, Backpropagation through time, Kalman filter*

List of figures

List of Tables

1. Introduction

1.1 Motivation

With the increasing demands of accuracy and prediction, applications and technologies of real-time control systems will be the research trend. They need to process data and perform some predetermined requirements in specified time intervals. [1][2] However, most of models in control systems are designed to learn offline based on training data, [3][4] which means once the model is complete and used, it will not be changed anymore unless the controlled plant is out of service. But in real applications, there can be a lot of changes during operation, so it is necessary to model a real-time control system process. There are two key tasks in a real-time control system to get optimal control. First is constructing the process model used in real-time system to forecast future states. And second is setting control variables dynamically based on prediction results to get expected behavior. Obviously, the accuracy of states under control highly depends on quality of the process model. Since adaptive feedback control process is from simulation of human operation, some researchers try to use Artificial Neural Network (ANN), which mimics a biological brain in a computational way and solves problem like humans do in real life.

ANN has been one hot research topic of artificial intelligence since 1980s. Over the past decade, ANN has got great development in solving pattern recognition, automation, biology and other difficult practical problems. Inspired by biological neural network, an ANN is groups of connected small processing units called artificial neuron and they form an abstract oriented network. Input signal propagates through network in the direction of connection. In contrast with biological neural network, connections between artificial neurons are not added or removed. They are weighted instead and weights are updated using learning algorithms. In supervised learning, learning algorithm adjusts weights and tries to minimize error between prediction output and desired output provided by training examples. [5]

A complete artificial neural network often refers to many training examples. This is the reason we choose a big data stream to train the network so that it can continually change the network all the time to adapt to changes in the system. Big data has been another hot research topic in the past ten years. It can be used in predictive analytics or other advanced data analytics, because large amount of data can help to get more accurate or expected outcomes undoubtedly. The term of big data has been used since 1990s. Big data is usually described by its characteristics. Laney was the first to define data growth challenges and opportunities with three dimensions: Volume, Velocity and Variety, known as 3V's of data. [6] Additional V's have been added by some organizations over ten years, like variability and veracity. [7]

1.2 Contribution and Problem Formulation

The aim of thesis is to design and implement a prototype self-learning prediction system with big data stream in a real-time process control framework so that the predicted future states can be used to predict next arrival input or modify the actual arrival input because of noise and other uncertainties. In this way, the response of a control system can be quicker and the dynamic process can be smoother. By self-learning, we mean the update of the model using example measurements acquired during the operation. In other words, the system adjust itself based on history information to predict future states and get better input and all desired inputs and outputs are from a data stream which can be a big data stream or an infinite data stream.

Since no one did some similar things before, the following research points should be considered in thesis.
1. Is neural network suitable for online mode in control system modeling? If the answer is yes, how to apply it to data stream learning?
Since neural network has been proved to be efficient in most offline real applications, only few researchers tried it in a dynamic way. This should be validated first because it is the base of using neural network to model the real-time control system process. Offline learning and online learning are completely different in design. A new way should be developed for online training.

2. What are differences among existing training algorithms? Can they all be used in online learning?
Training is the most difficult part in neural network modeling, because the structure is settled and the only difference between two neural networks is the choice of training method. So, we need to investigate them and know what are pros and cons of each one.

3. Can neural network be fast enough in a real-time system?
With the increase of data, demand of time will also grow at the same time, but real-time is a time-critical definition. Most real-time systems have hard deadlines and task period is short, so each training phase must be finished and provide high prediction accuracy in a limited time interval. This gives high timing requirements.

1.3 Outline of Thesis

In Section 2, background of neural network is introduced and related work is given in Section 3. Section 4 gives the research methodology used in the thesis. Training algorithms deployed in thesis work are fully detailed in Section 5. Section 6 describes the system model in detail, including prediction and training parts. Evaluation results are presented and discussed in Section 7. Conclusion are drawn and future work is also mentioned at the end.

2. Background

In this section, some basic definitions and structures of neural network are introduced, including neuron, feedforward neural network, deep learning and recurrent neural network.

2.1 Neural Network

A neural network is a parallel non-linear system which is composed of huge amount of connected process units. Although functionality of each unit is very simple, the network can still provide various and complex functionalities. This means information is not stored in any specific area. It is stored in every connection instead. In all, distribution and parallelism can bring great fault tolerance. First, because of distribution, if some neurons are broken, they will not degrade overall performance. For example, in real life, there are many nerve cells dying every day, but the human brain can still work well.[8] In artificial neural network, it is the same that part of broken units will not influence overall performance. Second, when input information is not clear or some information is missing, network can still recover the whole memory. Therefore, humans can recognize irregular handwriting.

Another characteristic of neural network is adaptivity. Adaptivity includes self-study and self-organization. Self-study means once environment changes, after a period of training time or perception, network can adjust some internal parameters automatically to give an expected output. And neural network can adjust the connection between two neurons according to certain rules to construct a complete network, which is called self-organization.

2.1.1 Neuron

One human brain consists of 10^{10} to 10^{12} units called neurons, which is a huge complex parallel information processing system. Each neuron is connected to other millions of neurons and communicates with them via electrochemical signals. Neuron continuously receives signal via synapses, located at the end of branches as shown in figure 1, and then performs in some specific ways. If the result is greater than the pre-defined threshold value, neuron will fire and generate a voltage and output a signal along something called an axon. [9]

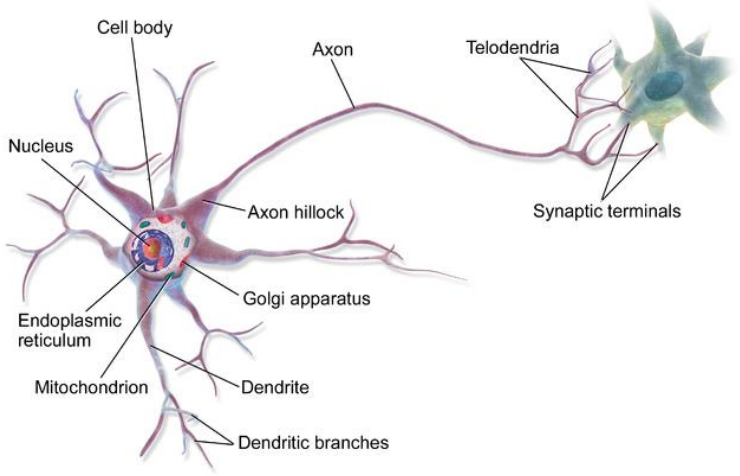

Figure 1 Biological neuron

In ANN, network is made up of many artificial neurons. These modelled neurons perform together in similar way to simulate human brain in a math way. The number of neurons can be as small as unit's digit or as big as thousand's digit. Each input into the neuron has its own corresponding weight. A weight is often a floating-point number and it is what to be adjusted when network is trained. Weights in neural network can be both negative and positive and each input is multiplied by its corresponding weight to get linear combination. Inputs and weights can be represented as $x_1, x_2 \dots x_n$ and $w_1, w_2 \dots w_n$. So, the linear combination can be written as:

$$x_1 w_1 + x_2 w_2 + x_3 w_3 + \cdots + x_n w_n = \sum_{j=0}^{n} x_j w_j \qquad (1)$$

The following picture is a basic artificial neuron structure.

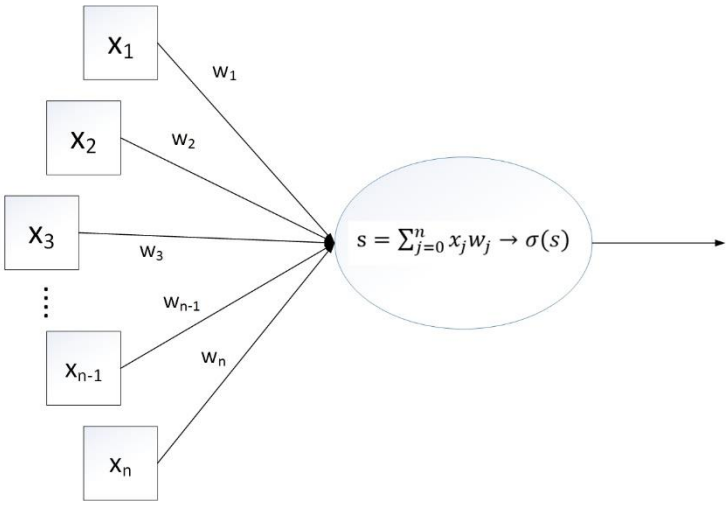

Figure 2 Artificial neuron

From figure 2, after calculating linear combination, unit gives activation σ, which is generally sigmoid function.

$$\text{sigmoid function: } f(x) = \frac{1}{1 + e^{-x}} \quad (2)$$

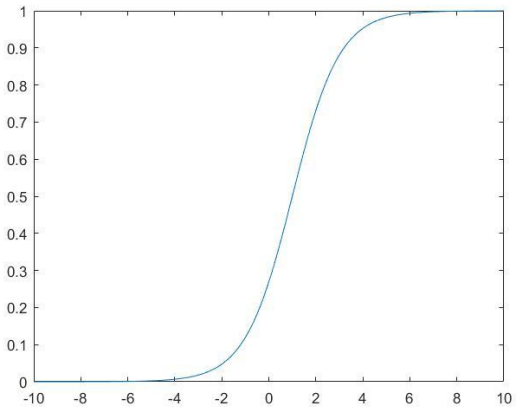

Figure 3 Sigmoid function

In this project, a new and popular activation function called Rectified Linear Units (RELU) is used.

$$\text{RELU function: } f(x) = \max(0, x) \quad (3)$$

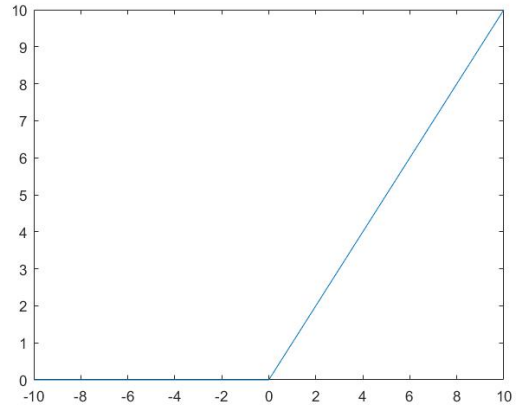

Figure 4 RELU function

It is from biology with same effect as sigmoid function. It can reduce possibility of vanishing gradient problem when training neural network, especially in deep neural network. And The other benefit of RELU is sparsity. Sparsity means the network can use as few non-zero values as possible to represent extracted features and it arises when linear combination ≤ 0 . The more units exist in a layer, the sparser resulting representation can be. On the other hand, Sigmoid always generates non-zero values to give dense representations. Obviously, sparse representations are more beneficial than dense representations. [10]

However, RELU is not all-mighty. It may be "dead" during training. If absolute value of one dimension of weights is larger than expected, it may cause the weight that the neuron

will never activate on any example again, which means the gradient will be zero forever and RELU units will irreversibly die during training. A network will be "dead" with high probability if the learning rate is too high. It is also called "dying RELU" problem. With a proper setting of learning rate, this is less frequently an issue. Or another solution is to use leaky RELU.

$$\text{Leaky RELU function: } f(x) = \max(0.01x, x) \quad (4)$$

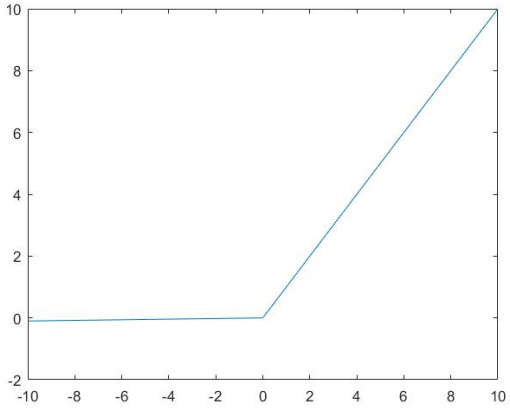

Figure 5 Leaky RELU function

When linear combination < 0, the function output is very small and we can ignore its value to some extent, but in the same time, it also prevents dying RELU problem. Therefore, it is widely used in deep neural network now.

2.1.2 Feedforward Neural Network

One common way of connecting neurons is to organize neurons into a design called feedforward neural network. It gets its name from the way that neurons in each layer are connected to next layer and feed their output again forward to the third layer until getting final output from neural network. The following is a very simple three-layer feedforward network.

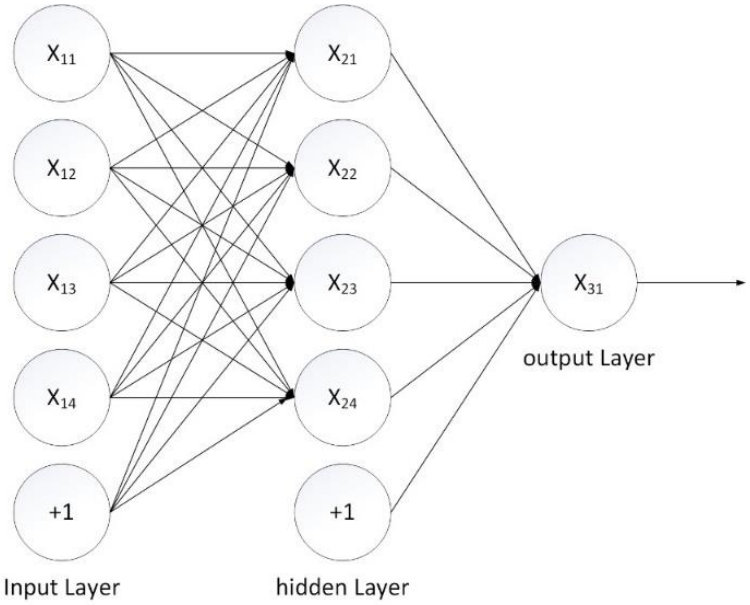

Figure 6 Feedforward neural network structure

Each neuron receives input values from the previous layer and sends output to next layer without feedback. All neurons in a network can be categorized in three kinds: input, hidden and output. Input neurons compose input layer. This layer only performs communication and sends information into network. In the same way, output layer consists of output neurons and receives information from network. Input layer and output layer are directly connected to environment, so they are called visible layers as well. Hidden layers, also called middle layers, do not have any relationship with outside. In hidden layers and output layer, every unit has multiple inputs, but only one output, which is also one of the input for next layer. The picture above is a fully-connected network, and each neuron in one layer is connected to all of neurons in the previous layer, including one special neuron whose value equals +1. The special one is for bias and its main functionality is affine transformation. The correct output representation of one neuron should be:

$$\text{output} = \sigma\left(1 * b_k + \sum_{j=1}^{n} x_j w_j\right) \quad (5)$$

So, bias is often seen as a part of weights (w_0) and corresponding input as +1. This assumption not only makes structure clear, but also makes calculation less complex.

We can see that hidden layers play an important role in a feedforward neural network because they can grab high-order statistical features. The more layers it has, the more valuable this ability can be. To start from the macro-view, a feedforward neural network is a static non-linear mapping system. It obtains powerful non-linear processing ability by some simple non-linear composite mappings.

2.2 Recurrent neural network

Unlike feedforward neural network, recurrent neural network (RNN) can use its internal memory to remember and process sequential data. This makes them applicable to time series prediction tasks. [11]

The idea behind RNN is to make use of sequential information. In a traditional feedforward neural network, all inputs, states and outputs are independent of each other. But for many real applications, this is a very bad idea. For example, if you want to predict the next word in a sentence you had better know which words came before. RNN can do this because they perform same tasks on every element of a sequence and they have a "memory" which collects information about what has been calculated so far.

The following case of basic structure was the first RNN by Jeff Elman. A three-layer network was used, with the addition of a set of "context units". There are connections from the hidden layer to these context units fixed with a weight of one. [12]

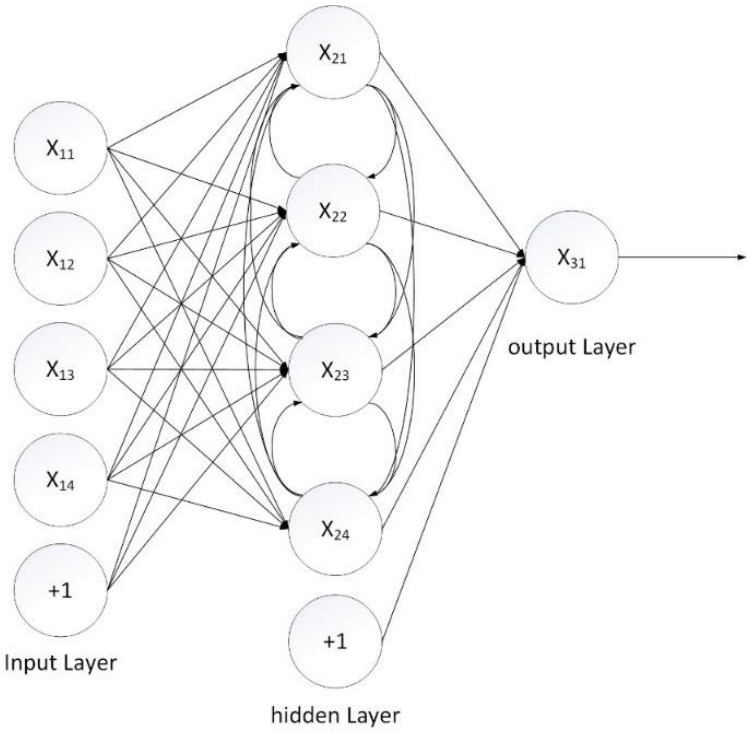

Figure 7 Basic recurrent neural network structure

However, the picture above is not a good way to show how it works, so typically, an unrolled diagram is used, as figure 8 shows, where rectangle means a whole layer with several neurons, including bias.

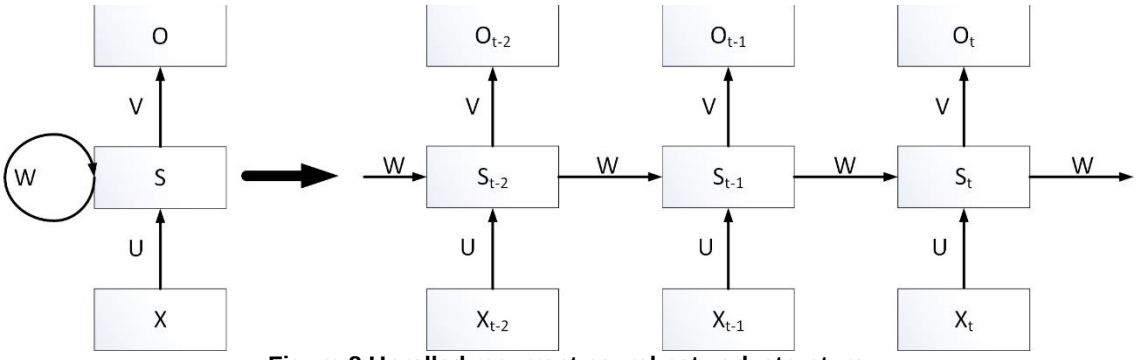

Figure 8 Unrolled recurrent neural network structure

By unrolling, it simply means that network can be written with complete sequence. For example, if the sequence is a sentence of 5 words, the network would be unrolled into a 5-layer neural network, one layer for each word.

The key point and difference between RNN and traditional ANN is that output of hidden-layer not only refers to inputs, but also depends on the hidden-layer states from previous time steps. We can use the following equations to calculate:

$$o_t = \sigma(\mathrm{V}S_t) = \sigma\left(\sum_j v_{kj} s_j(t)\right) \quad (6)$$

$$S_t = \sigma(UX_t + WS_{t-1}) \quad (7)$$

Equation (6) is calculation of output layer. Each neuron in output layer is connected to all of neurons in last hidden layer, so it is a fully-connected layer. V is weights matrix of output layer. Equation (7) is calculation of hidden layers, which are recurrent layers. U and W are weights matrixes of input layer and hidden layer at previous time step respectively.

There are 3 kinds of recurrent neural network structure for different problems: many-to-one, many-to-many and many-to-many with delay. Many-to-one is used in trend prediction or regression problem. It has totally multiple inputs in different unfolding networks, but only one output. Many-to-many is generally used in typing correction. For example, it is common that someone missed one letter when typing one word or someone do not remember the correct spelling, then the system can detect it and correct it automatically. The third one, many-to-many with delay, is widely used in mobile phones. It is the best solution to word association. Output starts after a complete input words sequence or just one word, then network captures some important information from the input sequence and tells what is the meaning input says and search the most relevant words in database.

2.3 Deep Learning

2.3.1 Description

Various high-level descriptions of deep learning include two key aspects:

- Models consisting of multiple layers of non-linear information processing;

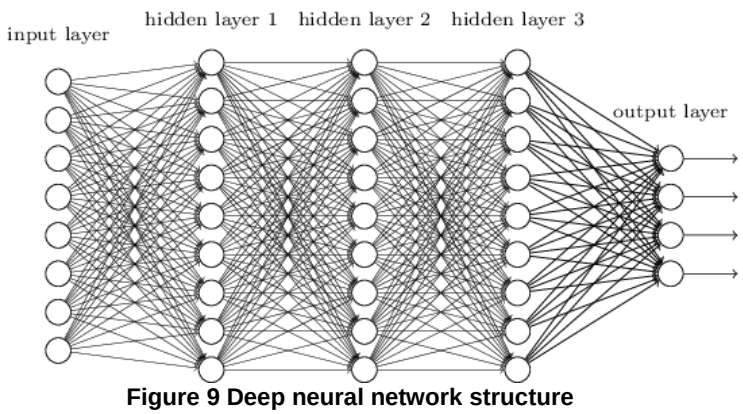

Figure 9 Deep neural network structure

- Methods for supervised or unsupervised learning of feature representation at successively higher, more abstract layers.

Deep learning is in the intersections among research areas of neural network and artificial intelligence. Three important reasons why deep learning is so popular now are the greatly increased chip processing abilities, the significantly increased size of training data, and the recent advances in machine learning and information processing research, which have enabled deep learning methods to effectively exploit complex, compositional non-linear functions, to learn hierarchical feature representations. [13]

One traditional deep learning network has the same structure as shallow neural network like figure 9 shows. It can also use a flow graph to show how the calculation works. But one of special attribute is depth, which is the length of the longest path from input layer to output layer. One more hidden layer means more abstract and features can be subdivided. For example, there is a picture and we want to identify whether the person in this picture is a boy or a girl with long or short hair. The following picture shows why a deep network can work better than a one-hidden-layer neural network: [14]

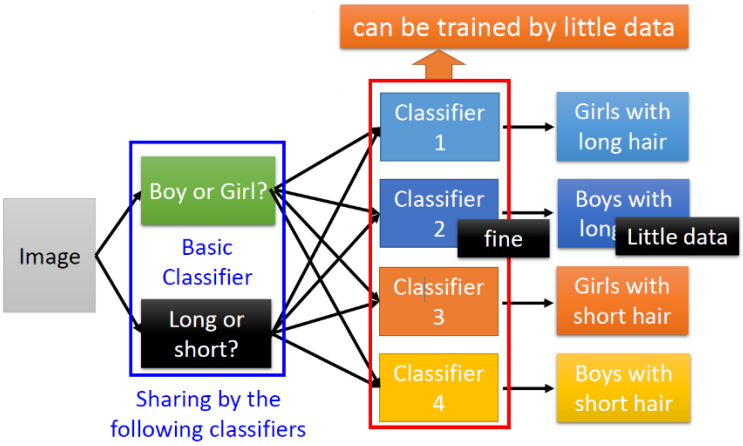

Figure 10 Modularization in deep neural network

2.3.2 Deep Recurrent Neural Network

Deep recurrent neural network is the final choice in this project. Compared with traditional deep neural network, it has recurrent hidden layers to store history information to deal with sequential problem. And as mentioned before, the more hidden layers there are in a neural network, the stronger the ability that can grab high-order statistical features is. So, it can solve more complex systems and make more precise results than a shallow recurrent neural network. [15]

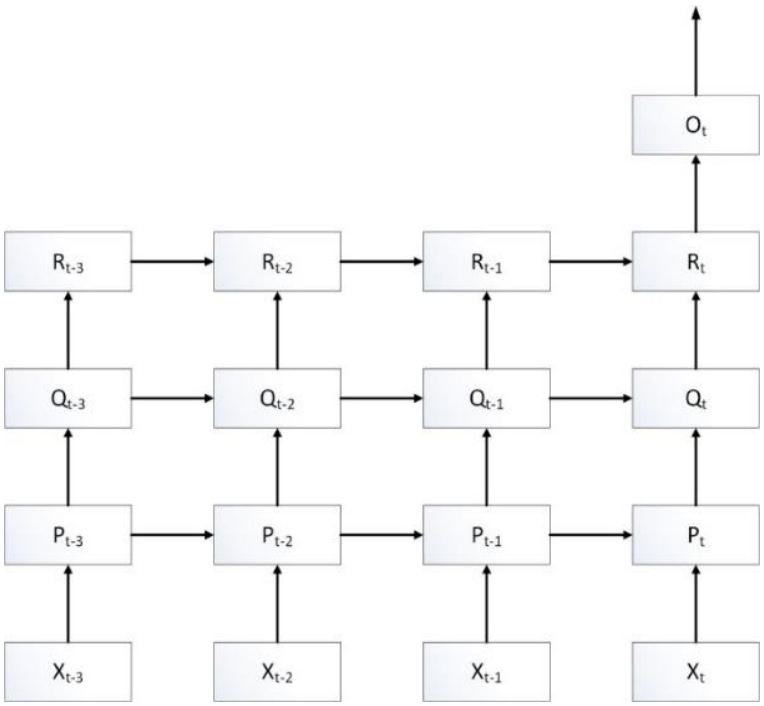

Figure 11 Deep recurrent neural network structure

2.4 Existent Training Algorithms

To train a recurrent neural network, one of widely-used and efficient method is Backpropagation through time (BPTT) learning algorithm. BPTT comes from Backpropagation algorithm (BP), but it takes time steps into consideration to be applicable to RNN. [24] In this project, another algorithm, Kalman filter, is also used, Since Kalman filter only works for linear system, a compromised method, extended Kalman filter (EKF), is employed to make it combined with gradient and weight,

Gradient descent, also known as steepest descent, is a basic idea to minimize a function $J(\theta_0, \theta_1, \dots, \theta_n)$ by updating parameters in the opposite direction of gradient. The learning rate η is the size of steps. In other words, the output is like a ball. It rolls downhill on the surface of function in the direction of slope until reaching valley.

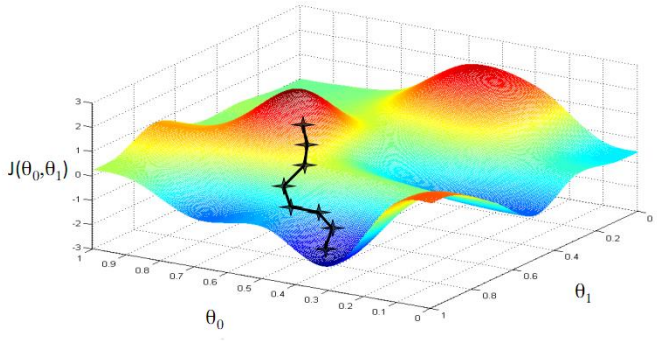

Figure 12 Gradient descent

2.4.1 Backpropagation Through Time

2.4.1.1 General Backpropagation

General BP is a basic optimization for feedforward neural network.

Backpropagation steps	
1	Initialization: If it is a totally new network, then randomize all of weights. They should be a normal distribution with 0 being mean value and 1 being variance.
2	Provide training examples set $\{(x(n), d(n))\}_{n=1}^{N}$.
3	Calculate outputs of hidden layers and output layer of each training example.
4	Calculate error between prediction result and desired result and propagate error signal to each hidden layer.
5	Update all of weights based on error signal.
6	Iteration: Back to step 3 until the network satisfy given rules.

A neural network with the following parameters:

- w_{kj}^{l} means the weight from k^{th} neurons in $(l-1)^{th}$ layer to j^{th}neurons in l^{th} layer.
- b_{j}^{l} means the bias for j^{th} neurons in l^{th} layer.
- z_{j}^{l} means the linear combination of j^{th} neurons in l^{th} layer:

$$z_{j}^{l} = b_{j}^{l} + \sum_{k} w_{kj}^{l} a_{k}^{l-1} \qquad (8)$$

- a_j^l means the output of j^{th} neurons in l^{th} layer:

$$a_j^l = \sigma\left(b_j^l + \sum_k w_{kj}^l \, a_k^{l-1}\right) \quad (9)$$

Backpropagation learning algorithm can be divided into two phases: backward propagation and weight update.

First is backward propagation. We need to define a loss function for output first. Then, propagate error signal back to all output and hidden neurons. A commonly-used loss function is root-mean-square deviation.

$$\mathrm{E} = \frac{1}{2}(y(x) - a^L(\mathrm{x}))^2 \quad (10)$$

x and y are training examples' input and output (desired input and output) respectively. a^L is the prediction value and L is maximum of layer number.

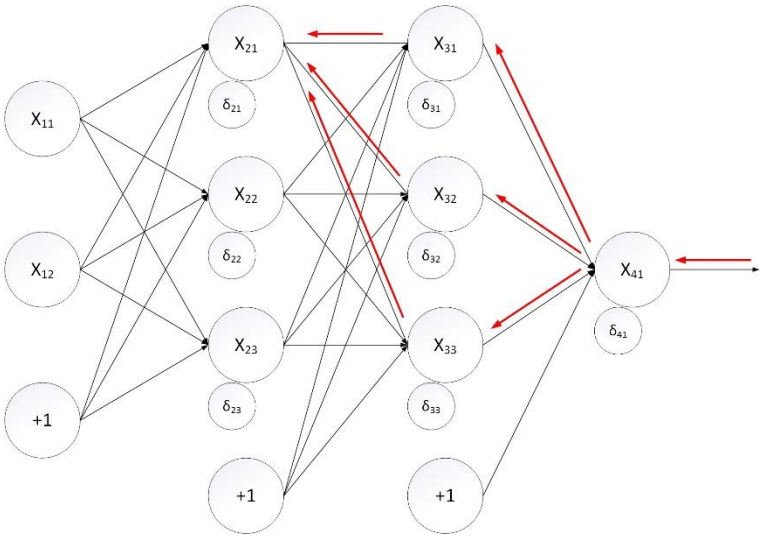

Figure 13 General Backpropagation

Error term of j^{th} neurons in l^{th} layer is defined as:

$$\delta_j^l = \frac{\partial E}{\partial z_j^l} \quad (11)$$

So, the error term of last (output) layer is:

$$\delta^L = \frac{\partial E}{\partial z_j^L} = \frac{\partial E}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial E}{\partial a^L}\sigma'(z^L) \quad (12)$$

Then, based on the calculated error term of last (output) layer, calculate other error terms backward layer by layer.

$$\begin{aligned}\delta_j^l = \frac{\partial E}{\partial z_j^l} &= \sum_m \frac{\partial E}{\partial z_m^{l+1}} \frac{\partial z_m^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_j^l} \\ &= \delta_m^{l+1} \frac{\partial \left(b_m^{l+1} + \sum_m w_{jm}^{l+1} * a_j^l\right)}{\partial a_j^l} \sigma'\left(z_j^l\right) \\ &= \sum_m \delta_m^{l+1} w_{jm}^{l+1} \sigma'(z_j^l)\end{aligned} \tag{13}$$

After calculating all of neurons' error terms, the next step is calculating each bias.

$$\frac{\partial E}{\partial b_j^l} = \frac{\partial E}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l \frac{\partial \left(b_j^l + \sum_k w_{kj}^l a_k^{l-1}\right)}{\partial b_j^l} = \delta_j^l \tag{14}$$

And each dimension of weights.

$$\frac{\partial E}{\partial w_{kj}^l} = \frac{\partial E}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{kj}^l} = \delta_j^l \frac{\partial \left(b_j^l + \sum_k w_{kj}^l a_k^{l-1}\right)}{\partial w_{kj}^l} = \delta_j^l a_k^{l-1} \tag{15}$$

The third phase is weight update.

$$\begin{cases} w_{kj}^l - \nabla w_{kj}^l = w_{kj}^l - \eta \dfrac{\partial E}{\partial w_{kj}^l} \\ b_j^l - \nabla b_j^l = b_j^l - \eta \dfrac{\partial E}{\partial b_j^l} \end{cases} \tag{16}$$

2.4.1.2 Backpropagation Through Time

BPTT do same things but do not only backpropagate to input layer but also to hidden layers in previous time steps to update weights. The following pictures simply shows how error signal is propagated through each hidden layer at different time steps.

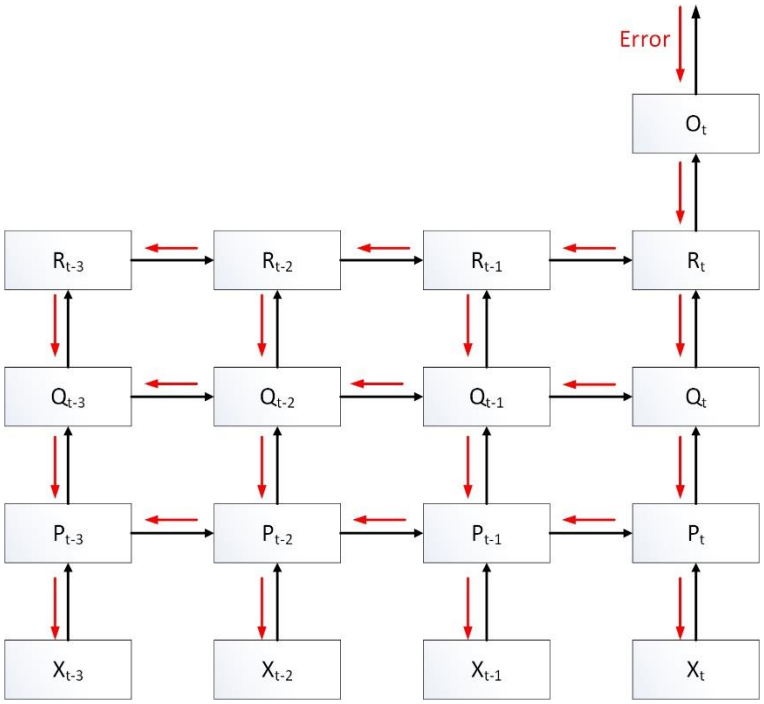

Figure 14 Backpropagation through time

- w_{kj}^{l} means the weight from k^{th} neurons in $(l-1)^{th}$ layer to j^{th}neurons in l^{th} layer.
- v_{ij}^{l} means the weight from i^{th} neurons in l^{th} layer in unfolding $u+1^{th}$ time to j^{th}neurons in l^{th} layer at unfolding u^{th} time.
- b_j^l means the bias of j^{th} neurons in l^{th} layer.
- z_{ju}^{l} means linear combination of j^{th} neurons in l^{th} layer at unfolding u^{th} time.

$$\begin{cases} z_{ju}^{l} = b_j^l + \sum_k w_{kj}^{l} a_{ku}^{l-1} + \sum_i v_{ij}^{l} a_{i,u+1}^{l}, \mathrm{u} < \mathrm{U} \\ z_{ju}^{l} = b_j^l + \sum_k w_{kj}^{l} a_{ku}^{l-1}, \mathrm{u} = \mathrm{U} \end{cases} \quad (17)$$

- a_{ju}^{l} means the output for j^{th} neurons in l^{th} layer at unfolding u^{th} time.

$$\begin{cases} a_{ju}^{l} = \sigma\left(b_j^l + \sum_k w_{kj}^{l} a_{ku}^{l-1} + \sum_i v_{ij}^{l} a_{i,u+1}^{l} \right), \mathrm{u} < \mathrm{U} \\ a_{ju}^{l} = \sigma\left(b_j^l + \sum_k w_{kj}^{l} a_{ku}^{l-1} \right), \mathrm{u} = \mathrm{U} \end{cases} \quad (18)$$

The error term of last (output) layer is the same as BP because output layer is not a recurrent layer:

$$\delta^L = \frac{\partial E}{\partial z_j^L} = \frac{\partial E}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial E}{\partial a^L}\sigma'(z^L) \tag{19}$$

Then, based on the error term of last (output) layer, we can calculate other error terms backward layer by layer.

$$\begin{aligned}
\delta_{ju}^l &= \frac{\partial E}{\partial z_{ju}^l} \\
&= \sum_m \frac{\partial E}{\partial z_{mu}^{l+1}} * \frac{\partial z_{mu}^{l+1}}{\partial a_{ju}^l} * \frac{\partial a_{ju}^l}{\partial z_{ju}^l} + \sum_n \frac{\partial E}{\partial z_{n,u-1}^l} * \frac{\partial z_{n,u-1}^l}{\partial a_{ju}^l} * \frac{\partial a_{ju}^l}{\partial z_{ju}^l} \\
&= \delta_{mu}^{l+1} \frac{\partial\left(b_m^{l+1} + \sum_m w_{jm}^{l+1} * a_{ju}^l + \sum_g v_{gm}^{l+1} * a_{g,u+1}^{l+1}\right)}{\partial a_{ju}^l} \sigma'\left(z_{ju}^l\right) \\
&+ \delta_{n,u-1}^l \frac{\partial\left(b_n^l + \sum_h w_{hn}^l * a_{h,u-1}^{l-1} + \sum_n v_{jn}^l * a_{ju}^l\right)}{\partial a_{ju}^l} \sigma'\left(z_{ju}^l\right) \\
&= \sum_m \delta_{mu}^{l+1} w_{jm}^{l+1} \sigma'\left(z_{ju}^l\right) + \sum_n \delta_{n,u-1}^l v_{jn}^l \sigma'\left(z_{ju}^l\right), \mathrm{u} > 1
\end{aligned} \tag{20}$$

$$\begin{aligned}
\delta_{ju}^l &= \frac{\partial E}{\partial z_{ju}^l} = \sum_m \frac{\partial E}{\partial z_{mu}^{l+1}} * \frac{\partial z_{mu}^{l+1}}{\partial a_{ju}^l} * \frac{\partial a_{ju}^l}{\partial z_{ju}^l} \\
&= \delta_{mu}^{l+1} \frac{\partial\left(b_m^{l+1} + \sum_m w_{jm}^{l+1} * a_{ju}^l + \sum_g v_{gm}^{l+1} * a_{g,u+1}^{l+1}\right)}{\partial a_{ju}^l} \sigma'\left(z_{ju}^l\right) \\
&= \sum_m \delta_{mt}^{l+1} w_{jm}^{l+1} \sigma'\left(z_{ju}^l\right), \mathrm{u} = 1
\end{aligned} \tag{21}$$

After calculating all of neurons' error terms, the next step is calculating each bias.

$$\begin{aligned}
\frac{\partial E}{\partial b_{ju}^l} = \frac{\partial E}{\partial z_{ju}^l}\frac{\partial z_{ju}^l}{\partial b_j^l} &= \delta_{ju}^l \frac{\partial\left(b_j^l + \sum_k w_{kj}^l a_{ku}^{l-1} + \sum_i v_{ij}^l a_{i,u+1}^l\right)}{\partial b_j^l} \\
&= \delta_{ju}^l
\end{aligned} \tag{22}$$

And each dimension of weights.

$$\frac{\partial E}{\partial w_{kju}^{l}} = \frac{\partial E}{\partial z_{ju}^{l}} \frac{\partial z_{ju}^{l}}{\partial w_{kj}^{l}}$$
$$= \delta_{ju}^{l} \frac{\partial \left(b_j^l + \sum_k w_{kj}^l a_{ku}^{l-1} + \sum_i v_{ij}^l a_{i,u+1}^l \right)}{\partial w_{kj}^l} \quad (23)$$
$$= \delta_{jt}^{l} a_{ku}^{l-1}$$

$$\frac{\partial E}{\partial v_{iju}^{l}} = \frac{\partial E}{\partial z_{ju}^{l}} \frac{\partial z_{ju}^{l}}{\partial v_{ij}^{l}} = \delta_{ju}^{l} \frac{\partial \left(b_j^l + \sum_k w_{kj}^l a_{ku}^{l-1} + \sum_i v_{ij}^l a_{i,u+1}^l \right)}{\partial v_{ij}^l} \quad (24)$$
$$= \delta_{jt}^{l} a_{i,u+1}^{l}$$

The third phase is weight update.

$$\begin{cases} w_{kj}^l - \nabla w_{kj}^l = w_{kj}^l - \eta \sum_u \frac{\partial E}{\partial w_{kju}^l} \\ v_{ij}^l - \nabla v_{ij}^l = v_{ij}^l - \eta \sum_u \frac{\partial E}{\partial v_{iju}^l} \\ b_j^{\mathrm{l}} - \nabla b_j^l = b_j^l - \eta \sum_u \frac{\partial E}{\partial b_{ju}^l} \end{cases} \quad (25)$$

2.4.2 Extended Kalman filter

Kalman filter is an algorithm that uses states from previous time steps, inputs at current time step and noise (typically Gaussian noise) to give an estimated state. It has been widely used in dynamic control systems, such as robot, aircraft and spacecraft and so on. Moreover, because of its characteristics, it can also be applicable to time series prediction in communication and economy. The main idea behind Kalman filter is to update states and covariance of states to preliminary guesses by extrapolating from previous values. Then adjust these preliminary guesses by incorporating the information contained in measurements. And the key property of Kalman filter is the product of two Gaussian functions is another Gaussian function. [25]

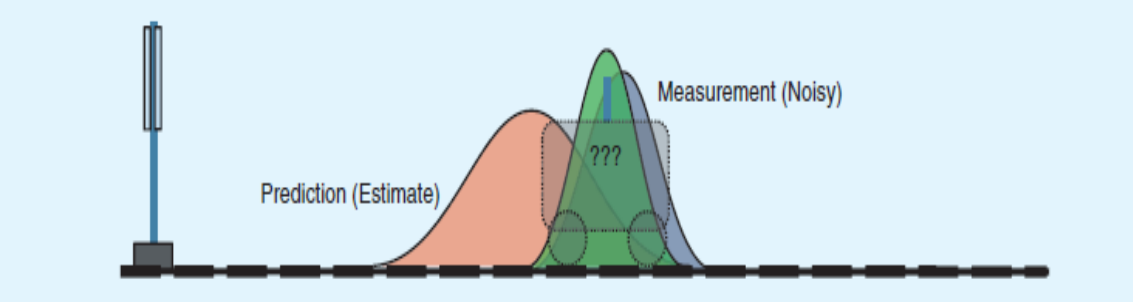

Figure 15 Kalman filter property

In this thesis, Kalman filter is combined with terms of gradient and weight. However, the original Kalman filter is limited to linear dynamic systems. If the system is non-linear, which is the case of neural network, it will not work. So, we must extend Kalman filter by linearization procedure, which is called extended Kalman filter (EKF). [26][27] To apply EKF to the task of estimating weights of neural network, we need to see weights as states in dynamic systems. To start from the macro-view, the network is like:

$$\text{output} = \text{h}(\text{w}, \text{input}(0,1,\dots,\text{n})) \quad (26)$$

If we render function h a time-dependent function h_n, it will become what we need, a function between weights and output, which can be transformed into input and output in EKF:

$$\text{output} = h_n(w(n)) \quad (27)$$

There are only 2 phases in Kalman filter, predict and update.

- Predict

1. Predict current state based on the previous state and current input.

$$\widehat{x_{t|t-1}} = F_t \widehat{x_{t-1|t-1}} + B_t u_t \quad (28)$$

Where:

- t is the time step.

- x_t is the state containing the terms of interest for the system. In neural network training, it represents weight.

- F_t is the state transition matrix. In neural network training, since weights are assumed to be equal when they are at t^{th} time step and $t-1^{th}$ time step, so F_t is identity matrix.

- B_t is the control input matrix which applies the effect of each control input. For weights in neural network, there is no input, so B_t is 0.

- u_t is is the control input. We do not need new input for training.

2. Predict covariance.

$$P_{t|t-1} = F_t P_{t-1|t-1} F_t^T + Q_t \quad (29)$$

Where:

- P_t is the covariance matrix which stores variances and covariances of states (weights).

- Q_t is the process noise covariance matrix of noisy control inputs

- Update

1. Calculate covariance of residual

$$S_t = H_t P_{t|t-1} H_t^T + R_t \quad (30)$$

H_t is the transformation matrix that maps states into measurements. This is the only difference in training neural network between original Kalman filter and EKF. In original Kalman filter, measurements of the linear system can be described as

$$z_t = H_t x_t + v_t, v_t \sim N(0, R_t) \quad (31)$$

Where R_t is the measurement noise covariance.

By contrast, in EKF, the measurement model need not be linear functions of states but non-linear functions instead.

$$z_t = h(x_t) + v_t, v_t \sim N(0, R_t) \quad (32)$$

It is obvious that function h cannot be applied to covariance calculation directly. Instead, a matrix of partial derivatives is computed. The basic idea is linearization of the latest state estimate $h(x_t)$. The first-order Taylor series is used to make it linearized.

$$h(x_t) \approx \mathrm{h}(\bar{\mu}) + \mathrm{h}'(\bar{\mu})(x_t - \bar{\mu_t}) = \mathrm{h}(\bar{\mu}) + H_t(x_t - \bar{\mu_t}) \quad (33)$$

In BP, Gradient is partial derivative of error with respect to each weight, but in EKF, it should be partial derivative of output with respect to each weight $H_t = \frac{\partial h(x_t)}{\partial x}$. So, the error term of output layer is replaced with "output term":

$$\gamma^L = \frac{\partial h}{\partial z_j^L} = \frac{\partial h}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial h}{\partial a^L} \sigma'(z^L) \quad (34)$$

Then calculate the other "output terms" backward layer by layer as BPTT does.

$$
\begin{aligned}
\gamma_{ju}^{l} &= \frac{\partial h}{\partial z_{ju}^{l}} \\
&= \sum_{m} \frac{\partial h}{\partial z_{mu}^{l+1}} * \frac{\partial z_{mu}^{l+1}}{\partial a_{ju}^{l}} * \frac{\partial a_{ju}^{l}}{\partial z_{ju}^{l}} + \sum_{n} \frac{\partial h}{\partial z_{n,u-1}^{l}} * \frac{\partial z_{n,u-1}^{l}}{\partial a_{ju}^{l}} * \frac{\partial a_{ju}^{l}}{\partial z_{ju}^{l}} \\
&= \gamma_{mu}^{l+1} \frac{\partial\left(b_{m}^{l+1} + \sum_{m} w_{jm}^{l+1} * a_{ju}^{l} + \sum_{g} v_{gm}^{l+1} * a_{g,u+1}^{l+1}\right)}{\partial a_{ju}^{l}} \sigma'\left(z_{ju}^{l}\right) \\
&+ \gamma_{n,u-1}^{l} \frac{\partial\left(b_{n}^{l} + \sum_{h} w_{hn}^{l} * a_{h,u-1}^{l-1} + \sum_{n} v_{jn}^{l} * a_{ju}^{l}\right)}{\partial a_{ju}^{l}} \sigma'\left(z_{ju}^{l}\right) \\
&= \sum_{m} \gamma_{mu}^{l+1} w_{jm}^{l+1} \sigma'\left(z_{ju}^{l}\right) + \sum_{n} \gamma_{n,u-1}^{l} v_{jn}^{l} \sigma'\left(z_{ju}^{l}\right), \mathrm{u} > 1
\end{aligned}
\tag{35}
$$

$$
\begin{aligned}
\gamma_{ju}^{l} &= \frac{\partial h}{\partial z_{ju}^{l}} = \sum_{m} \frac{\partial h}{\partial z_{mu}^{l+1}} * \frac{\partial z_{mu}^{l+1}}{\partial a_{ju}^{l}} * \frac{\partial a_{ju}^{l}}{\partial z_{ju}^{l}} \\
&= \gamma_{mu}^{l+1} \frac{\partial\left(b_{m}^{l+1} + \sum_{m} w_{jm}^{l+1} * a_{ju}^{l} + \sum_{g} v_{gm}^{l+1} * a_{g,u+1}^{l+1}\right)}{\partial a_{ju}^{l}} \sigma'\left(z_{ju}^{l}\right) \\
&= \sum_{m} \gamma_{mu}^{l+1} w_{jm}^{l+1} \sigma'\left(z_{ju}^{l}\right), \mathrm{u} = 1
\end{aligned}
\tag{36}
$$

After calculating all of neurons' terms, calculate each bias.

$$
\begin{aligned}
\frac{\partial h}{\partial b_{ju}^{l}} = \frac{\partial h}{\partial z_{ju}^{l}} \frac{\partial z_{ju}^{l}}{\partial b_{j}^{l}} &= \gamma_{ju}^{l} \frac{\partial\left(b_{j}^{l} + \sum_{k} w_{kj}^{l} a_{ku}^{l-1} + \sum_{i} v_{ij}^{l} a_{i,u+1}^{l}\right)}{\partial b_{j}^{l}} \\
&= \gamma_{ju}^{l}
\end{aligned}
\tag{37}
$$

And each dimension of weights.

$$
\begin{aligned}
\frac{\partial h}{\partial w_{kju}^{l}} &= \frac{\partial h}{\partial z_{ju}^{l}} \frac{\partial z_{ju}^{l}}{\partial w_{kj}^{l}} \\
&= \gamma_{ju}^{l} \frac{\partial\left(b_{j}^{l} + \sum_{k} w_{kj}^{l} a_{ku}^{l-1} + \sum_{i} v_{ij}^{l} a_{i,u+1}^{l}\right)}{\partial w_{kj}^{l}} \\
&= \gamma_{ju}^{l} a_{ku}^{l-1}
\end{aligned}
\tag{38}
$$

$$
\begin{aligned}
\frac{\partial h}{\partial v_{iju}^{l}} = \frac{\partial h}{\partial z_{ju}^{l}} \frac{\partial z_{ju}^{l}}{\partial v_{ij}^{l}} &= \gamma_{ju}^{l} \frac{\partial\left(b_{j}^{l} + \sum_{k} w_{kj}^{l} a_{ku}^{l-1} + \sum_{i} v_{ij}^{l} a_{i,u+1}^{l}\right)}{\partial v_{ij}^{l}} \\
&= \gamma_{ju}^{l} a_{i,u+1}^{l}
\end{aligned}
\tag{39}
$$

The gradient of each weight can be calculated as:

$$\begin{cases} \dfrac{\partial h}{\partial w_{kj}^{l}} = \sum_{u} \dfrac{\partial h}{\partial w_{kju}^{l}} \\ \dfrac{\partial h}{\partial v_{ij}^{l}} = \sum_{u} \dfrac{\partial h}{\partial v_{iju}^{l}} \\ \dfrac{\partial h}{\partial b_{j}^{l}} = \sum_{u} \dfrac{\partial h}{\partial b_{ju}^{l}} \end{cases} \quad (40)$$

2. Calculate Kalman gain.

$$K_t = P_{t|t-1} H_t^T S_t^{-1} \quad (41)$$

3. Update state estimate

$$\widehat{x_{t|t}} = \widehat{x_{t|t-1}} + K_t \tilde{y}_t \quad (42)$$

Where:

$$\tilde{y}_t = z_t - H_t \widehat{x_{t|t-1}} \quad (43)$$

This is the most important part in Kalman filter. It means that the bigger Kalman gain is, the more trustworthy measurement value can be.

4. Update estimate covariance

$$P_{t|t} = P_{t|t} - K_t H_t P_{t|t} \quad (44)$$

3. Related Work

This thesis is fundamentally a time series prediction system with big data stream. Time series forecasting with streaming data plays an important role in many real applications. However, real data is often accompanied with anomalies and changes, which can make the learned models deviate from the underlying patterns of time series, especially in the context of online learning mode. This difficulty can be reduced by using recurrent neural network, [16] which has been proved effective in prediction problem. In [17] [18] and [19], artificial neural network is used in different domains to solve time series prediction problem. All of authors choose Backpropagation as their training method and it can achieve good predicted results. [20] presents an accurate and reliable short-term load forecasting system model which can be used to optimize power system operations, like the aim of this thesis, but the difference is that it does not need to consider adaptivity and high timing requirements. In [3], authors present a neuro-controller with adaptive deadzone compensation to solve the problem of controlling an unknown SISO non-linear systems. The system is based on recurrent neural network and it gets low tracking error, approximately 0.2. All the works show that recurrent neural networks have been widely-used in many domains to solve time series prediction problem. Researchers tried different methods to improve their own network, and the common one is Backpropagation.

Deep learning is suitable for learning data with sophisticated and large-scale structure. This is the biggest advantage that shallow neural network does not have. So, in future, it will be applied to even larger-scale data, like big data in cloud or a big data stream. In [21], a deep learning model predicts increase and decrease in the sales of a retail store. The predictive accuracy varies between 75% to 86% with changes in the number of product attributes, which proves a multi-layer neural network can be better than a shallow one. The following two works aim at optimizing operations by predicting future states with deep neural network, but they are both offline learning. In [22], a new deep learning approach for the VM workload prediction in the cloud system is presented. The experimental results show that the proposed prediction approach based on deep belief network composed of multi-layer restricted Boltzmann machines can improve accuracy of the CPU utilization prediction than other existing prediction approaches. In [23], traditional PID controller is replaced with a deep learning controller. The deep learning controller is trained offline with simulation data. Then remove the original PID controller, and use deep learning controller to accomplish the functionalities PID can do. All in all, in some real applications which has high accuracy requirements, a multi-layer neural network is always the best choice.

4. Research Method

Figure 12 is the process of research method that were followed in this thesis.

Figure 16 Research method

It includes totally 5 steps.

1. Problem formulation

This thesis process started by formulating problems. From the characteristics of data stream and real-time control system, we can foresee some challenges and problems.

2. Related work

By reading different papers and researches about neural network, we can have the basic knowledge and idea to model an artificial neural network.

3. Implementation

Not only a dynamic adaptive real-time control system process will be modeled, but also two existing training algorithms will be implemented. And a new hybrid training algorithm will be presented as well.

4. Methods validation

There are many uncertainties in a neural network. Different combinations may lead to various consequences and problems. In this step, we mainly test and discard the options which will probably bring very bad results. For example, we need to test if leaky RELU function can solve dying RELU problem in a deep recurrent neural network. Another important thing is to test if two existing algorithms and the proposed one work in such a model.

5. Evaluation

In this step, we mainly choose the best one from several uncertainties options to construct the final model and compare the performance among three different algorithms. The indexes taken into consideration should be not only accuracy, but also speed and adaptivity, since a real-time control system is a time-varying system with high timing requirements.

5. Training Algorithms Comparison

From the above, we can know that EKF is a second-order gradient descent algorithm. Compared with traditional gradient-descent-based techniques like Backpropagation, it has much faster convergence. For general gradient descent algorithm, the curvature of error surface is different in different directions, so learning rate must be small enough to keep stable when there is high curvature. However, if learning rate is very small, rate of convergence will no doubt be slow with low curvature. So, one of the method is to calculate second-order derivative to incorporate curvature information into gradient descent process. That is why EKF can have both fast convergence and good performance.

However, everything has pros and cons. There is one disadvantage when using EKF. Although EKF only needs one single step for each training example, faster convergence leads to overfitting problem under some specific conditions, especially when training one example for the first time, like following pictures (from the first period of first experiment) show.

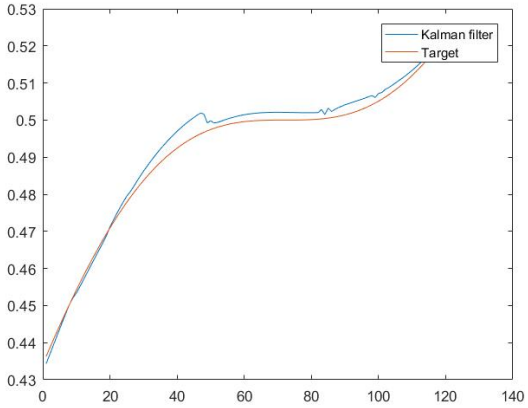

Figure 17 Kalman filter fitness

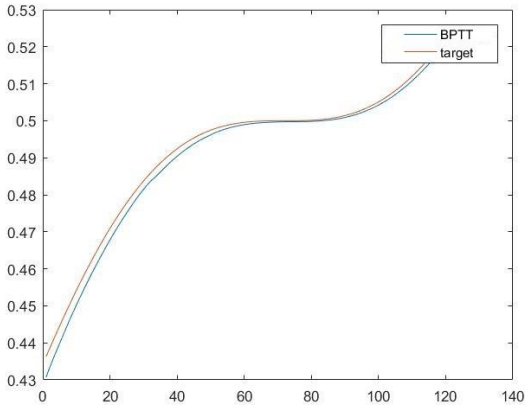

Figure 18 BPTT fitness

We can see that Kalman filter prediction result is highly dependent on information from previous time steps. For example, the curve rises significantly at the beginning 30 data

points and rises slowly after 30 data points, but the feature neural network learned from these 30 data points is only curve rising significantly, which is why if any big change happens to system, there will be a fluctuation process to re-adapt. But BPTT does not have this kind of problem. It is much smooth and can do greater job on turning part because of slower convergence.

To solve this problem, we present a training algorithm to take the essence of characteristics from EKF and BPTT. EKF can provide fast convergence and this is one key point to take into consideration in a dynamic system because there is no much time for each training example. Besides, accuracy is always one of concerns in neural network modelling. All training steps are for future states prediction, so we cannot only concentrate on current example's accuracy. That is why we need BPTT to prevent overfitting problem.

6. Proposed Training Method

6.1 Forward Pass

In this thesis, inputs are all from a real-time big data stream. Based on this assumption, we choose sliding window to fill in inputs of deep recurrent neural network, because sliding window can split a data stream into several parts and each part can be one individual training example. In this way, it can produce a training example set for neural network training. The advantage of using sliding window is that we do not need a real storage to store all of history parts (training examples). Once the new arrival input is received and used for training, then the new created part can be discarded, because information has been recorded in each connection of neural network. This is how we deal with data stream learning.

Due to the characteristics of recurrent neural network, even though each training example is independent, two neighbor training examples still have hidden relationship, because each arrival input is trained for several times, not only one. For example, in table 1, inputs of window n are x(n), x(n+1), …, x(n+k), and inputs of window n+1 are x(n+1), x(n+2), …, x(n+k+1). Inputs x(n+1), x(n+2), …, x(n+k) can be trained for 2 times. Therefore, a sliding window can improve convergence.

Table 1 Sliding window

Window	Input	Output
1	x(1), x(2), …, x(k)	o(k)
2	x(2), x(3), …, x(k+1)	o(k+1)
3	x(3), x(4), …, x(k+2)	o(k+2)
⋮	⋮	⋮
n	x(n), x(n+1), …, x(n+k)	o(n+k)
n+1	x(n+1), x(n+2), …, x(n+k+1)	o(n+k+1)

Where k is the unfolding time from recurrent neural network and n is the number of window. x(n) and y(n) are normalized inputs and outputs for neural network respectively. Once one example is input from environment, it must be normalized before being sent into neural network and output must be renormalized. So, there are 5 steps for each training example.

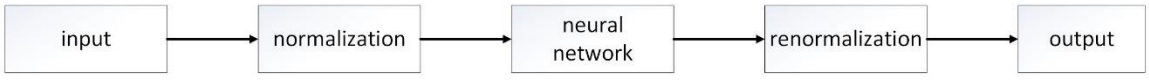

Figure 19 Prediction steps

The following is normalization equation.

$$\text{normalization: } x(n) = \frac{u(n) - u_{min}}{u_{max} - u_{min}} \quad (45)$$

Where u is the actual input from environment. u_{max} is the pre-defined maximum environment value based on domain knowledge and u_{min} is the pre-defined minimum.

After getting normalized inputs, the next step is sending normalized inputs to the neural network and calculating layer by layer till output layer, and finally renormalize the output value.

$$\text{reverse normalization: } y(n) = o(n)(y_{max} - y_{min}) + y_{min} \quad (46)$$

Where y is the actual prediction output from neural network. y_{max} is the pre-defined maximum enviroment value from domain knowledge and y_{min} is the pre-defined minimum.

6.2 Hybrid Training Algorithm

As discussed in section 5, because of limitations of both training algorithms, a new combined training algorithm is presented. When a new training example is input, the first step is receiving weights from BPTT at previous time step and using new weights to predict:

$$\text{output} = h_n(W^{BPTT}) \quad (47)$$

Then use weights from Kalman filter at previous time step to train with Kalman filter again and BPTT sequentially. The overall process should be like:

$$W_t^{EKF} = f_{EKF}(W_{t-1}^{EKF}) \quad (48)$$

$$W_t^{BPTT} = f_{BPTT(}W_t^{EKF}) \quad (49)$$

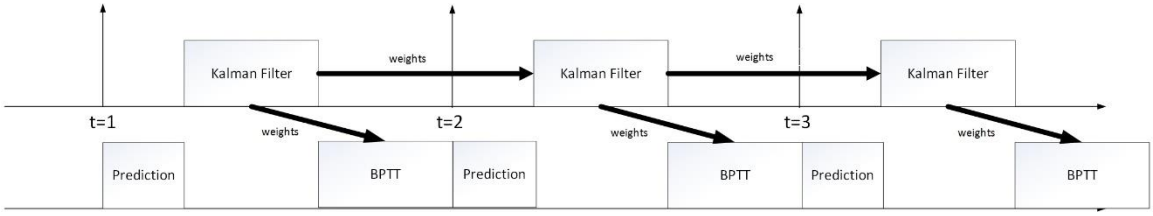

Figure 20 Training algorithm

The main idea behind this design is to make full use of Kalman filter's fast convergence and BPTT's stability. In this way, weights will still highly depend on Kalman filter's estimate, but it will not be influenced by overfitting problems caused by Kalman filter because of BPTT's correction. On the other hand, BPTT can also get much better initial weights to train than the original one, so to some extent, it can not only assist Kalman filter fitness to be smooth, but also make fitness better in stationary phase. One thing to mention is that learning rate in BPTT should be small, or the fitness will deviate much and slow convergence down. It cannot be too small as well, or it cannot change too much and this is just waste of time.

From the theory point, the aim of hybrid mode is to take essence from both algorithms, but Kalman filter uses state and covariance at previous time step to predict state and covariance at current time step, which means both two variables are one-to-one and we cannot just update states(weights) by BPTT and ignore covariance, or the result will be completely wrong because of distribution calculation. So, we need to draw weights alone, then update by BPTT. In this way, both variables can be kept for next Kalman filter training and they will not be influenced by extra BPTT training.

The whole procedure is like two processes in embedded system. The first one is responsible for Kalman Filter algorithm. It not only updates weights every time unit, but also stores them and sends to BPTT calculation task. The other process receives weights from primary process and updates weights by BPTT, then predicts future states. So, if there is a multi-core processor, this algorithm can run on two different processes to save time (prediction part).

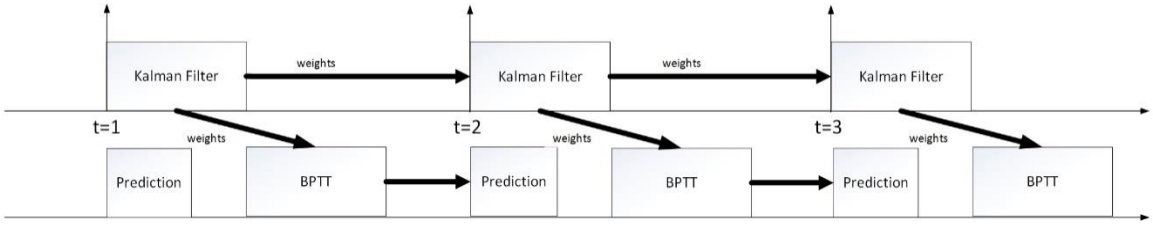

Figure 21 Training algorithm in multi-core processor

7. Evaluation

To compare the performance of two existent training algorithms and new proposed one, several experiments were made. Three aspects are taken into consideration, rate of convergence, adaptivity, and accuracy. Rate of convergence means how much time the algorithm needs to train neural network to get a closer optimum, and adaptivity means once some change happens whether the neural network can adjust itself and how fast it can be. Since we can only get one number from each group of prediction output and desired output, we use relative error to define accuracy relationship between prediction result and desired result. In equation 48, y' is the prediction output and y^d is the desired output of each example.

$$\varphi = 1 - \left| \frac{y' - y^d}{y^d} \right| \quad (50)$$

7.1 Static System

In this section, we mainly investigate performance of convergence rate and accuracy in a static system and we totally prepared two experiments for it.

7.1.1 System Identification

- Identify a non-linear static system, from [28], described by:

$$y(t) = u(t)^3 + \frac{y(t-1)y(t-2)[y(t-1)-1]}{1+y(t-1)^2} \quad (51)$$

And the input is:

$$u(t) = 0.5\sin(4\pi t) \quad (52)$$

- The other non-linear static system, from [29], is described by:

$$y(t) = 0.3y(t-1) + 0.6y(t-2) + u(t)^3 + 0.3u(t)^2 - 0.4u(t) \quad (53)$$

And the input is:

$$u(t) = \sin\left(\frac{2\pi t}{250}\right) \quad (54)$$

7.1.2 First Experiment

7.1.2.1 Existent Algorithms

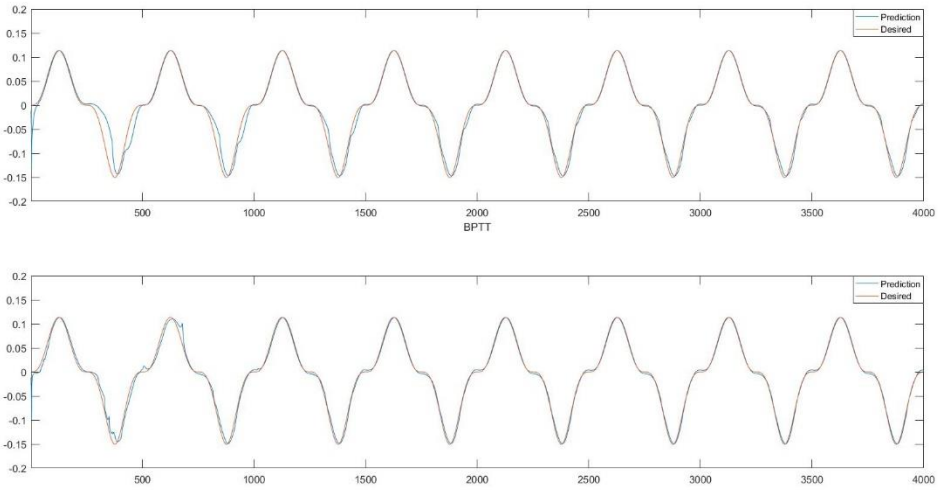

Figure 22 BPTT and Kalman filter result in static system

The picture above shows the result of BPTT and Kalman filter with beginning 4000 data points (totally 8 periods). The red line is desired stream and the blue is prediction result. From the picture, they both have their own advantages when training neural network. As discussed before, BPTT does better job than Kalman filter when fitting turning part because of overfitting problem. But in other cases, Kalman filter seems better. Since the result is hard to recognize which is the best choice, the following picture gives some more detailed accuracy information of two algorithms. All points in the picture are average accuracy in each period and it includes totally 16000 data points (32 periods).

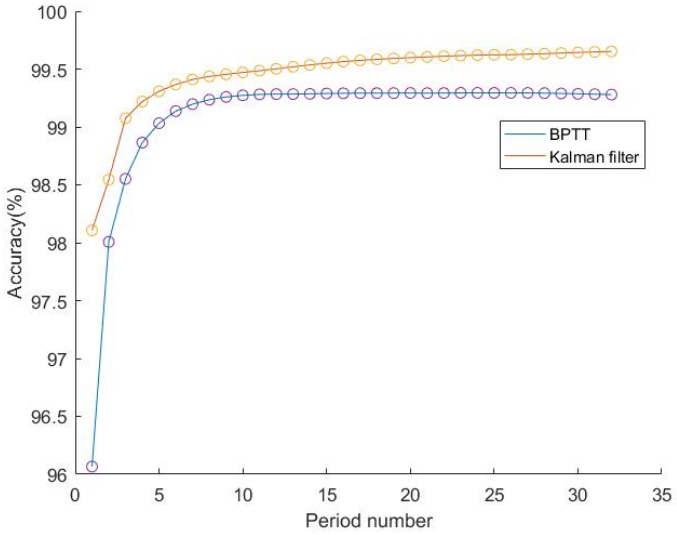

Figure 23 Accuracy information of BPTT and Kalman filter in static system

The accuracy information apparently shows Kalman filter has much faster convergence than BPTT, especially in the first period. Average accuracy of Kalman filter is 98.10%, 2.04% more than BPTT's, which means previous training examples have been learned and memorized quickly when using Kalman filter so that it can recognize examples with same features and finally, it can get approximately 99.65% accuracy after 32 periods, but BPTT cannot make it so fast. BPTT optimizes very slowly after 10 periods, increases by only 0.002% per iteration and gets 99.29% after 32 periods, but Kalman keeps rising much throughout. That is the benefit of second-order derivative calculation. All the weights share the same learning rate in BPTT, but Kalman filter does not.

7.1.2.2 Proposed Algorithm

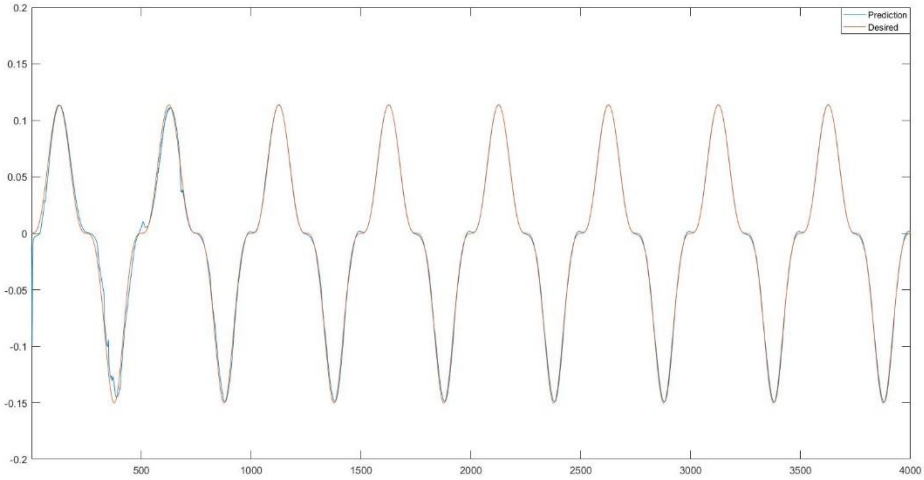

Figure 24 Proposed algorithm result in static system

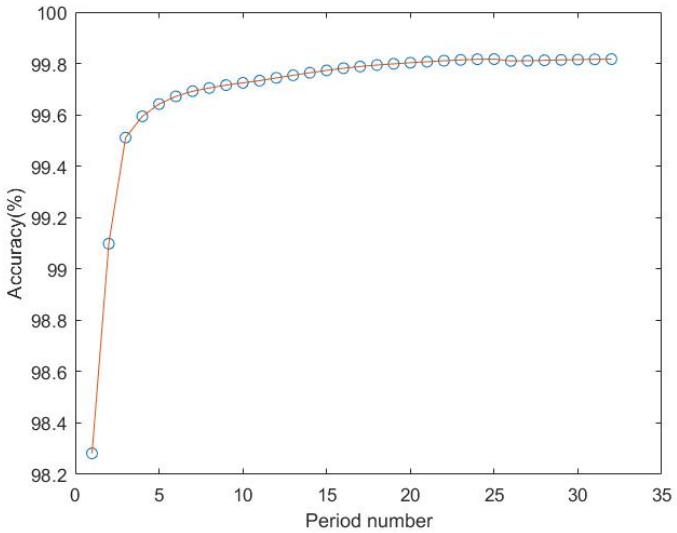

Figure 25 Accuracy information of proposed algorithm in static system

Based on the result from previous experiments, we can know both existent algorithms have their own strengths. Figure 24 and 25 are result and accuracy information of proposed algorithm respectively. The result shows that the new one keeps most characteristics from Kalman filter and rate of convergence is almost the same as Kalman filter, but it also makes some small improvements based on BPTT, which can be found from not only accuracy information, but also details in figure 24. Compared with pure Kalman filter, even though at the beginning, first period accuracy is only improved by 0.18%, second period increases by 0.55% and final accuracy rises from 99.65% to 99.82%, and the profit it gets from BPTT is the stable adjustment under some special circumstances, like turning part at the beginning periods.

7.1.3 Second Experiment

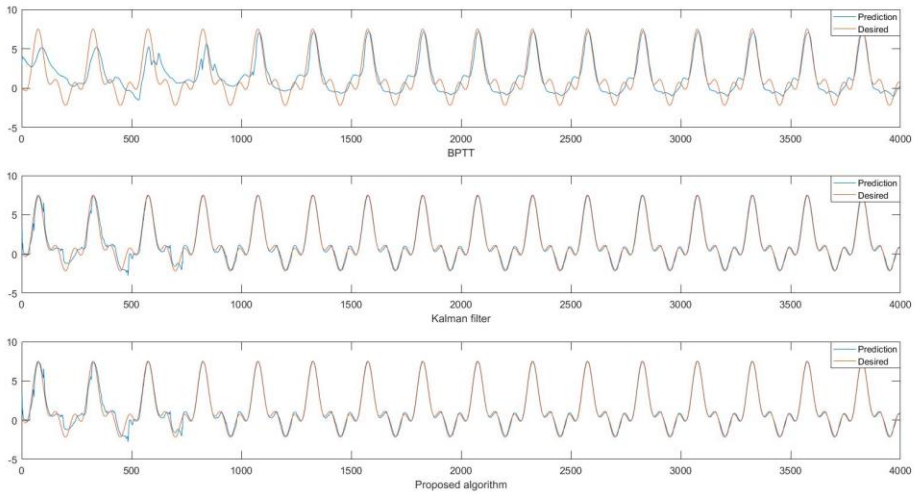

Figure 26 Result of second experiment in static system

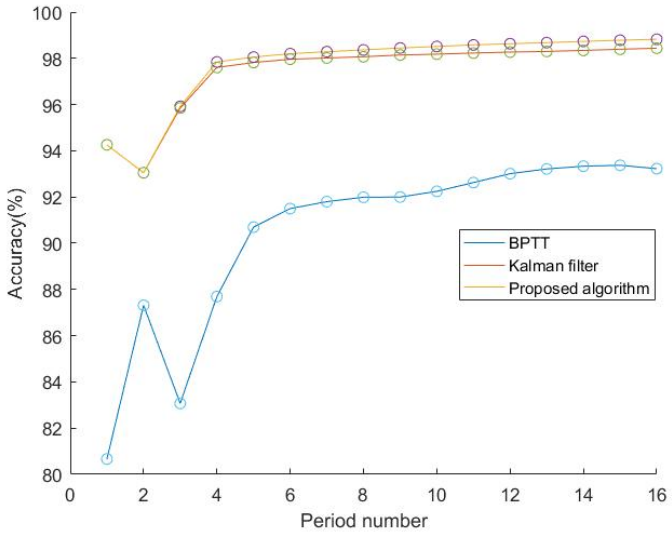

Figure 27 Accuracy information of second experiment in static system

In this experiment, we chose a bad initial weight set to show how big the difference of convergence rate among three algorithms can be. First, it is obvious that all three curves decrease in the beginning. The reason it happens in Kalman filter is that Kalman filter changes each weight too much so that new weights cannot fit history training examples. By contrast, BPTT curve drops back to 83.07% in 3rd period, which means a not proper learning rate may fit only some weights in 2nd period. Maybe average accuracy in this period is better, but that is just because of average value's role. Second, algorithms rise greatly first after the mentioned drop, then they enter a stationary phase, but Kalman filter and Kalman-filter-based algorithms continuously increase and BPTT experiences some fluctuations. That means once BPTT enters saturation region, the shared learning rate will be a drag. Third, we can also grab from the result that BPTT does not do a good job in this system, even after 4000 data points, it cannot get an approximate optimum. A proper setting of learning rate is important for pure BPTT, but it is hard to find such a number for all weights. The deeper neural network is, the more weights there are and the harder to find the proper setting.

7.2 Time-varying System

Since this thesis is to model a dynamic adaptive real-time control system process, in this section, we not only focus on the performance of convergence rate and accuracy, but also concentrate on the adaptivity of different training algorithms in a time-varying system. We changed some parameters of system after several periods and kept the same original weight set so that weights will not influence the result.

7.2.1 System Identification

- Now, the first non-linear system is described by:

$$\begin{cases} \mathrm{y(t)} = u(t)^3 + \dfrac{y(t-1)y(t-2)[y(t-1)-1]}{1+y(t-1)^2}, t < 8000 \\ \mathrm{y(t)} = (1.5 * u(t)^3 + \dfrac{y(t-1)y(t-2)[y(t-1)-1]}{1+y(t-1)^2}, t > 8000 \end{cases} \quad (55)$$

Where the input is:

$$\begin{cases} \mathrm{u(t)} = 0.5\sin(4\pi \mathrm{t}), \mathrm{t} < 8000 \\ \mathrm{u(t)} = 0.5\sin(2\pi \mathrm{t}), \mathrm{t} > 8000 \end{cases} \quad (56)$$

- The second non-linear system is:

$$\begin{cases} \mathrm{y(t)} = 0.3\mathrm{y(t-1)} + 0.6\mathrm{y(t-2)} + u(t)^3 + 0.3u(t)^2 - 0.4\mathrm{u(t)}, t < 4000 \\ \mathrm{y(t)} = 0.3\mathrm{y(t-1)} + 0.6\mathrm{y(t-2)} + 1.5u(t)^3 + 0.5u(t)^2 - \mathrm{u(t)}, t > 4000 \end{cases} \quad (57)$$

Where the input is:

$$u(t) = \sin\left(\frac{2\pi t}{250}\right) \quad (58)$$

7.2.2 First Experiment

7.2.1.1 Existent Algorithms

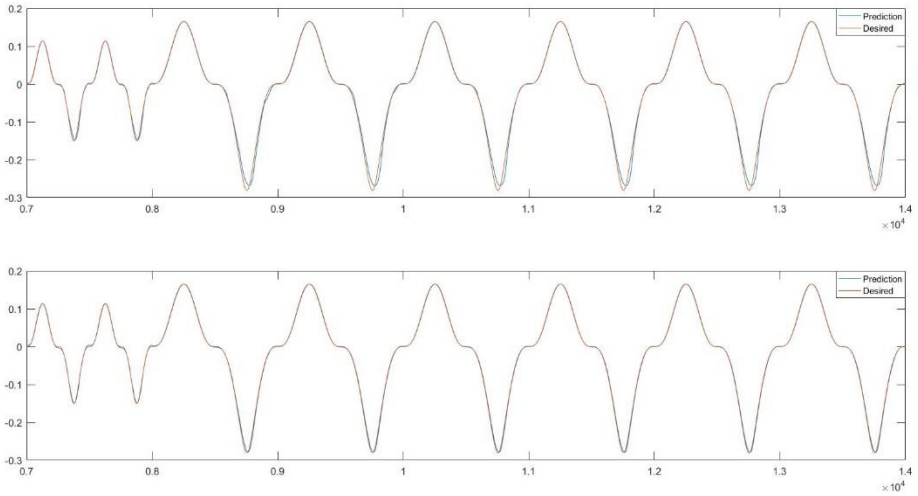

Figure 28 BPTT and Kalman filter result in time-varying system

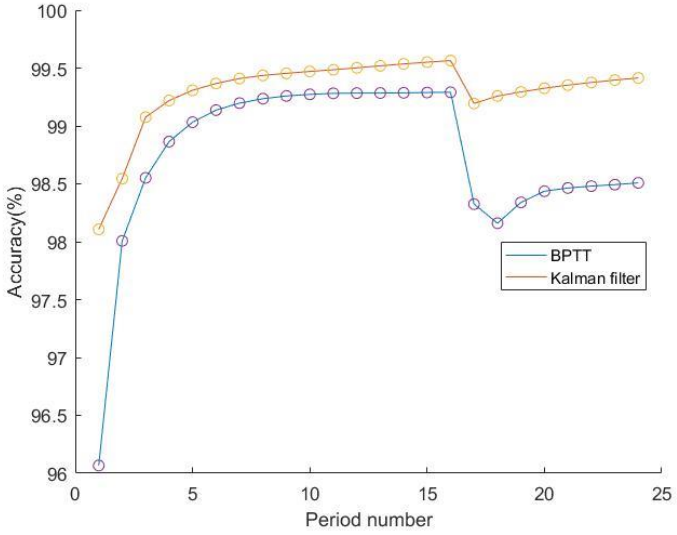

Figure 29 Accuracy information of BPTT and Kalman filter in time-varying system

The pictures above are result and accuracy information in a time-varying system with BPTT and Kalman filter. The first figure is the result when change happens and the second one is overall accuracy information. There is a drop in 1st period after change happens. Accuracy decreases from 99.56% to 99.19%. But then accuracy does not drop anymore when using Kalman filter, which means neural network starts adapting to

changes. On the other hand, BPTT still needs one more period to train and re-adapt. From the number, we can also see that BPTT's average accuracy drops by 0.96%, which is much bigger than the drop in Kalman filter, about 0.37%, then drops more, about 0.16%. The reason of it is the rate of convergence. Kalman filter can change each dimension of weights fast and individually.

7.2.2.2 Proposed Algorithm

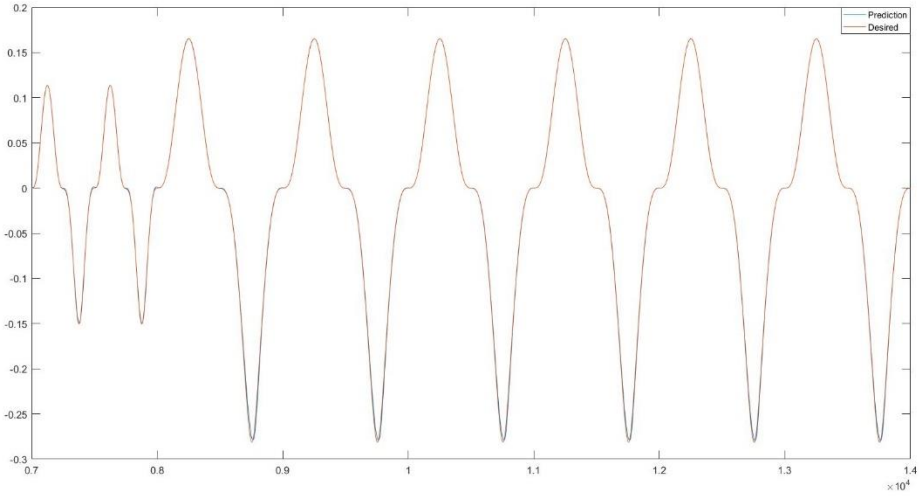

Figure 30 Proposed algorithm result in time-varying system

Since the new one is a Kalman-filter-based algorithm improved by BPTT, so figure 31 shows the comparison of accuracy information between presented algorithm and pure Kalman filter.

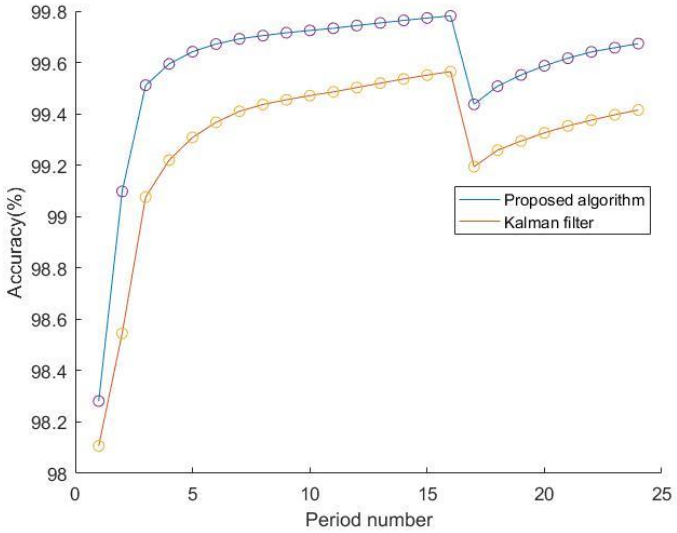

Figure 31 Accuracy information of proposed algorithm in time-varying system

After change happens, the new presented algorithm rises faster than pure Kalman filter and the difference is gradually enlarged because of BPTT's role. This also happens at the beginning, which shows that combination of two existent algorithms makes double efforts to both re-adaption and optimization. From this picture, we can know that the average difference is about 0.2%. For real-time control systems and neural networks which require high accuracy, it is a great improvement.

7.2.3 Second Experiment

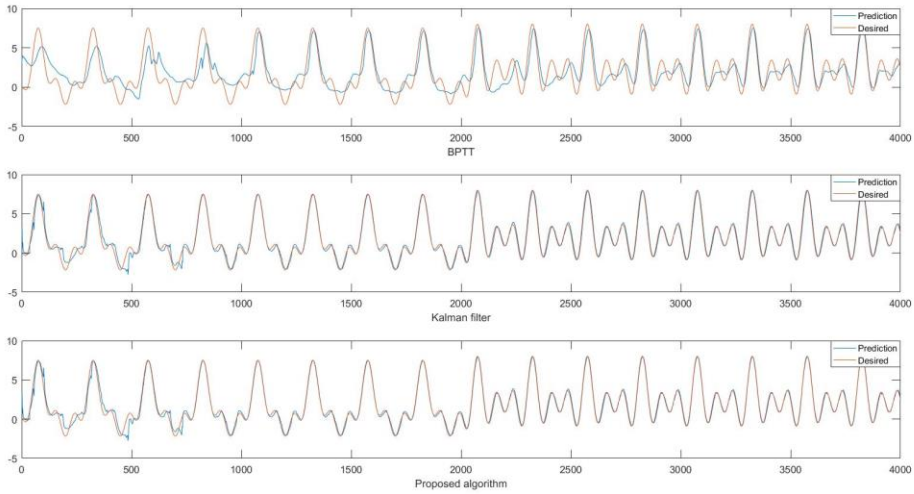

Figure 32 Result of second experiment in time-varying system

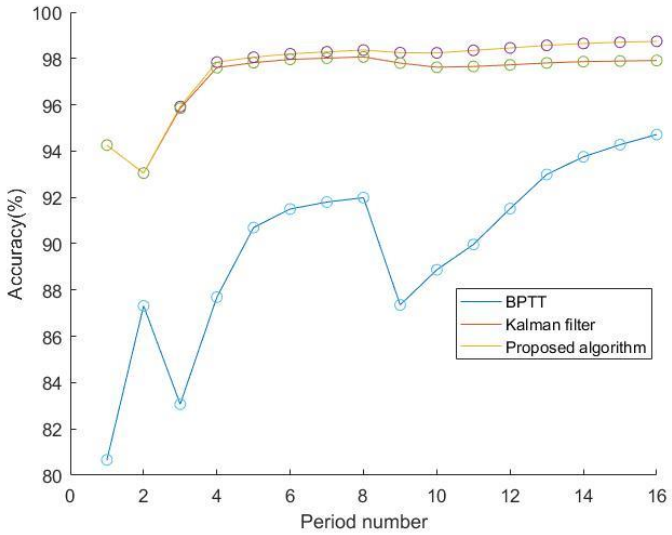

Figure 33 Accuracy information of second experiment in time-varying system

The accuracy information shows that the change leaves little influence on Kalman filter and proposed algorithm. Especially for the new one, the accuracy just decreases by 0.11%. This proves Kalman filter's fast convergence again. Compared with two Kalman-filter-based algorithms, BPTT does not give good enough results in this experiment. That

means once a bad initial weights set are given, using BPTT always means a long time to get optimum weights set. That is why in the proposed algorithm, we need EKF first, and give new updated weights to BPTT task secondly.

7.3 Discussion

In static systems, when initial weights set is not bad for training, differences among three algorithms are not big. All accuracy curves rise smoothly, but BPTT almost stop rising because of shared learning rate. Benefiting from second-order gradient calculation, Kalman filter does better on this point, especially when given bad initial weights set. After the drop after the first period, Kalman filter can quickly find an approximate optimum set, but BPTT seems very slow.

In time-varying systems, because of shared learning rate, change makes more effect on BPTT fitness so that BPTT accuracy drops much. On the other side, due to Kalman filter's fast convergence, neural network has good adaptivity to face challenges from system requirements or environments. However, under some special circumstances, we can still find BPTT has its own advantages from both kinds of systems, like fitting ability in turning part, which is the reason we need it to overcome overfitting problem in new proposed training algorithm.

To sum up, Kalman filter and Kalman-filter-based algorithm is the better choice, but the new proposed one should give the credit to improvement from BPTT as well. Additional BPTT adjustment is like combustion adjuvant. Kalman filter plays more important role in training, and BPTT is responsible for solving some small and infrequent problems or improving more.

8. Conclusion

There are not many prediction models for control systems now, but, there can be some changes in a system with time going and environment changing. It is not always possible to let them out of service once the system need to be changed. At this moment, a dynamic adaptive process model with prediction functionality can help a lot.

In this thesis work, we propose a process model with new Kalman-filter-based training algorithm for prediction in dynamic control systems. This model is fundamentally an artificial recurrent neural network and it takes the essence from two training algorithms, Kalman filter and Backpropagation through time because of complementary advantages. Compared with traditional feedforward neural network, a network with recurrent layers can store history data in these additional layers. New presented training algorithm is a combination of Kalman filter and BPTT. Kalman filter brings fast convergence but its instability may cause some unpredictable problems, and BPTT can help to prevent them and improve more.

The results of experiments show that new algorithm improves original two algorithms. It inherits most characteristics, especially fast convergence from Kalman filter and then applies BPTT to make small adjustments to avoid some problems caused by Kalman filter and make the model more accurate and more stable. Even if there is any change, the model also has good adaptivity to cope with coming challenges.

9. Future Work

Because of complexity of recurrent neural network and limitation of time, not all parameter combinations have been tested, including layer number of neural network, neuron number in each layer, learning rate and noise P and Q in Kalman filter. There is also some hidden relationship between two of them, like the smaller layer number is, the bigger unfolding time should be, but there is no clear definition now. This kind of exploration is time-consuming and huge.

We used MATLAB to implement this model, not C language or assembly language, so we did not test how much time it needs in real applications. In MATLAB, each training example needs 24ms per iteration. So, if it is transferred to other language, it should be faster.

This thesis only focuses on the prediction part in an adaptive control system. After getting an estimated result, the result should be used in another model to suggest creating new approximate inputs for next time step. In this way, created inputs can be more promising than those only from outside.

Reference

[1] V. Sigarev, T. Kuzima, A. Krasilnikov, "Real-time control system for a DC motor". *NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW)*. IEEE, 2016

[2] F. A. Qureshi, V. Spinu, K. Wijnands et al., "A real-time control system structure for industrial power amplifiers". *American Control Conference (ACC), 2013*. IEEE, 2013

[3] J. H. Perez-Cruz, I. Chairez, J. J. Rubio et al., "Identification and control of class of non-linear systems with non-symmetric deadzone using recurrent neural networks", *IET Control Theory & Applications (Volume: 8, Issue: 3, Feb.13 2014).* IEEE, 2014

[4] M. Aamir, "On Replacing PID Controller with ANN Controller for DC Motor Position Control". *International Journal of Research Studies in Computing, Consortia Academia Publishing, vol. 2 no. 1, pp. 21-29, April 2013*

[5] B. Wilson, "The Machine Learning Dictionary". 2012

[6] Beyer, Mark, "Gartner Says Solving 'Big Data' Challenge Involves More Than Just Managing Volumes of Data". Retrieved 13 July 2011.

[7] G. A. Lakshen, S. Vraneš, V. Janev, "Big Data and Quality: A Literature Review". *Telecommunications Forum (TELFOR), 2016 24th*. IEEE,2017

[8] R. S. Nowakowski, "Stable neuron numbers from cradle to grave". *Proceedings of the National Academy of Sciences. 103 (33): 12219–12220*

[9] "neuron". *Oxford English Dictionary (3rd ed.)*. Oxford University Press. September 2005.

[10] V. Nair, G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines". *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*

[11] D. Brezak, T. Break, D. Majetic et al., "A comparison of feed-forward and recurrent neural networks in time series forecasting". *Computational Intelligence for Financial Engineering & Economics (CIFEr), 2012 IEEE Conference*. IEEE, 2012

[12] J. L. Elman, "Finding Structure in Time". *Cognitive Science. 14 (2): 179–211*. 1990

[13] L. Deng, D. Yu, "Deep Learning: Methods and Applications". *Foundations and Trends in Signal Processing. 7 (3-4): 1–199*. 2014

[14] L. Hungyi, "Lecture I: Introduction of Deep Learning". Retrieved from http://people.tamu.edu/~gengxbtamu/pdf/Deep_Learning_Tutorial_Hungyi_Lee.pdf

[15] M. Hermans, B. Schrauwen, "Training and Analyzing Deep Recurrent Neural Networks". *Advances in Neural Information Processing Systems 26*

[16] T. Guo, Z. Xu, X. Yao et al., "Robust Online Time Series Prediction with Recurrent Neural Networks" *Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on*. IEEE, 2016

[17] D. X. Niu, H. Shi, J. Q. Li et al., "Research on short-term power load time series forecasting model based on BP neural network" *Advanced Computer Control (ICACC), 2010 2nd International Conference*. IEEE, 2010

[18] D. A. Kumar, S. Murugan, "Performance analysis of Indian stock market index using neural network time series model". *Pattern Recognition, Informatics and Mobile Engineering (PRIME), 2013 International Conference*. IEEE, 2013

[19] S. H. Yu, J. Y. Ou, "Forecasting Model of Agricultural Products Prices in Wholesale Markets Based on Combined BP Neural Network -Time Series Model". *Information Management, Innovation Management and Industrial Engineering, 2009 International Conference*. IEEE, 2009

[20] N. Siddarameshwara, A. Yelamali, K. Byahatti, "Electricity Short Term Load Forecasting Using Elman Recurrent Neural Network". *Advances in Recent Technologies in Communication and Computing (ARTCom), 2010 International Conference*. IEEE, 2010

[21] Y. Kaneko, K. Yada, "A Deep Learning Approach for the Prediction of Retail Store Sales". *Data Mining Workshops (ICDMW), 2016 IEEE 16th International Conference*. IEEE, 2017

[22] F. Qiu, B. Zhang, J, Guo, "A deep learning approach for VM workload prediction in the cloud". *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2016 17th IEEE/ACIS International Conference on*. IEEE, 2016

[23] K. Cheon, J. Kim, M. Hamadache et al., "On Replacing PID Controller with Deep Learning Controller for DC Motor System". Journal of Automation and Control Engineering vol. 3, no. 6, pp. 452-456

[24] J. Mazumdar, R. G. Harley, "Recurrent Neural Networks Trained with Backpropagation Through Time Algorithm to Estimate Nonlinear Load Harmonic Currents". *IEEE Transactions on Industrial Electronics (Volume: 55, Issue: 9, Sept. 2008)*. IEEE, 2008

[25] R. Faragher, "Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation". *IEEE SIGNAL PROCESSING MAGAZINE [128] SEPTEMBER 2012*. IEEE, 2012

[26] P. Trebatický, J. Pospíchal, "Neural Network Training with Extended Kalman Filter Using Graphics Processing Unit". *ICANN 2008, Part II, LNCS 5164, pp. 198–207, 2008*

[27] S. Singhal, W Lance, "Training multilayer perceptrons with the extended Kalman algorithm", *Advances in neural information processing systems 1 Pages 133-140*

[28] L. Xingyu, Y. Chunyan, Y. Bangyan et al., "A Hybrid Recurrent Neural Network for Machining Process Modeling". *ISNN 2009, Part I, LNCS 5551, pp. 635-642*. 2009.

[29] M.C. Nechyba, X. Yangsheng, "Neural network approach to control system identification with variable activation functions". *Intelligent Control, 1994., Proceedings of the 1994 IEEE International Symposium*. IEEE, 2002