RECURRENT NEURAL NETWORK LEARNING WITH AN APPLICATION TO THE CONTROL OF LEGGED LOCOMOTION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

By
Bahadır Çatalbaş
August, 2015

RECURRENT NEURAL NETWORK LEARNING WITH AN APPLICATION TO THE CONTROL OF LEGGED LOCOMOTION
By Bahadır Çatalbaş
August, 2015

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Ömer Morgül (Advisor)

Prof. Dr. Hitay Özbay

Assoc. Prof. Dr. Uluç Saranlı

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Graduate School

ABSTRACT

RECURRENT NEURAL NETWORK LEARNING WITH AN APPLICATION TO THE CONTROL OF LEGGED LOCOMOTION

Bahadır Çatalbaş
M.S. in Electrical and Electronics Engineering
Advisor: Prof. Dr. Ömer Morgül
August, 2015

Use of robots for real life applications has an increasing trend in today's industry and military. The robot platforms are capable of performing dangerous and difficult tasks, which are not efficient when carried out by human beings. Most of these tasks require high motion ability. There are various robotic platform and locomotion algorithms which may solve a given task. Among these, the biped robot platforms promise high performance in realizing difficult maneuver due to their morphological similarity to legged animals. Thus, legged locomotion is highly desirable in order to perform difficult maneuvers in rough terrain environments. However both modeling and control of such structures are quite difficult due to highly nonlinear structure of the resulting equations of motion and computational load of inverse kinematic equations. Central nervous systems and spinal cords of animals take role in control of locomotion of animals together. For controlling such biped robotic platforms frequently used control algorithms are based on so-called Central Pattern Generators (CPG). The controllers based on CPGs can be realized in different ways which includes the utilization of neural networks. However CPG is only capable of imitating spinal cord type of reflex-based motions in locomotion because of their restricted parameter space to sustain stable oscillation. Fully recurrent neural networks have capability of controlling locomotion with a higher conscious level such as central nervous system, hence motion space can be enlarged. Unfortunately, training of recurrent neural networks (RNN) takes long time. Moreover, their behaviors may be unpredictable against untrained inputs and training process may encounter with instability related problems easily. In order to solve these problems, various acceleration and regularization techniques are tested in the neural network training and their successes were compared with each other. Furthermore, time constant and error gradient limitation methods are employed to sustain stable training

and their benefits are discussed. Finally leg angles of walking biped robot are taught to a group of RNNs with different configurations by benefiting from training stability enhancing methods. The resulting RNNs are then used in biped locomotion by using a classical PD controller. After that, performance of resulting RNNs and their stable locomotion generation capabilities are evaluated and effects of configuration parameters are discussed in detail.

Keywords: Recurrent Neural Network (RNN), Leaky Integrator Neuron Model, Backpropagation Through Time (BPTT), Biped Robot Locomotion.

ÖZET

TEKRARLI SİNİR AĞI ÖĞRENİMİ İLE BACAKLI HAREKET KONTROLÜNE UYGULANMASI

Bahadır Çatalbaş
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Prof. Dr. Ömer Morgül
Ağustos, 2015

Bugünün sanayisinde ve askeriyesinde robotların gerçek hayat uygulamaları için kullanımı artmakta olan bir eğilimdir. Robot platformları insanlar tarafından yürütülmesi verimli olmayan tehlikeli ve zor görevleri yerine getirebilme kapasitesine sahiptir. Verilen bir görevi halledebilen çeşitli robotik platformlar ve hareket algoritmaları vardır. Bunların arasında iki ayaklı robot platformları, ayaklı hayvanlara olan yapısal benzerliklerinden dolayı zor manevraların gerçekleştirilmesinde yüksek performans vaat etmektedir. Bu nedenle bacaklı hareket engebeli arazi ortamlarında zor manevraları uygulayabilmek için son derece arzu edilir. Ancak bu yapıların hem modellenmesi ve hem de kontrolü elde edilen büyük ölçüde doğrusal olmayan hareket denklemleri ve ters kinematik denklemlerinin hesaplama yükü sebebiyle oldukça zordur. Hayvanların merkezi sinir sistemleri ve omurilikleri hayvanın hareketinin kontrolünde beraber rol alır. Böyle iki ayaklı robot platformları kontrol etmek için sıklıkla kullanılan kontrol algoritmaları Merkezi Örüntü Üreteçlerine (MÖÜ) dayanmaktadır. Merkezi örüntü üreteçlerine dayanan kontrolörler, sinir ağlarının kullanımı da dahil olmak üzere farklı yollarla gerçekleştirilebilir. Ancak merkezi örüntü üreteçleri kararlı salınımlarını sürdürebilmek için sınırlandırılan değişken uzaylarından ötürü sadece hareketlerdeki omurilik tipi refleks temelli hareketleri taklit edebilme yeterliliğindedir. Tamamen tekrarlı sinir ağları merkezi sinir sistemi gibi daha yüksek bilinç düzeyi ile hareket kontrolü yapabilme yeterliliğine sahiptir böylece hareket uzayı genişletilebilir. Ne yazık ki, Tekrarlı Sinir Ağlarının (TSA) eğitimi uzun zamanı alır. Bundan başka eğitilmedikleri girdilere karşı davranışları tahmin edilemeyebilir ve eğitim süreci kararsızlıkla alakalı sorunlarla kolaylıkla karşılaşabilir. Bu sorunları çözebilmek için çeşitli hızlandırma ve düzenlileştirme teknikleri sinir ağlarının eğitiminde test edildi ve bunların başarıları birbirleriyle karşılaştırıldı. Ayrca kararlı eğitim sağlamak için zaman sabiti ve hata gradyanı

sınırlama metotları kullanıldı ve bunların faydaları tartışıldı. Son olarak iki ayaklı robot yürüyüş bacak açıları bir grup farklı tekrarlı sinir ağı yapılandırmasına kararlılık arttırıcı metotlardan faydalanılarak öğretildi. Ortaya çıkan tekrarlı sinir ağlarının performansları ve kararlı hareket oluşturma yetenekleri değerlendirildi ve yapılandırma değişkenlerinin etkileri detaylı bir şekilde tartışıldı.

Anahtar sözcükler: Tekrarlı Sinir Ağı (TSA), Sızdıran Bütünleştirici Sinir Modeli, Zaman Boyunca Geri Yayılım (ZBGY), İki Ayaklı Robot Hareketi.

Acknowledgement

First of all, I would like to thank my supervisor Prof. Dr. Ömer Morgül for his precious guidance, supports, patience and encouragements in my Master's Thesis study. It was not possible to finish my academic works in two years without his help and the vision that he let me inspire.

I am thankful to Prof. Dr. Hitay Özbay and Assoc. Prof. Dr. Uluç Saranlı for reading my thesis and being member of my thesis committee.

In addition, I would like to thank to my instructors throughout my education life, from primary school up to the university lecture halls.

Also, I want to appreciate the understanding of my colleague assistants and my professors while working as a teaching assistance of several lectures to fulfill the academic practice assignments of my curriculum.

I would like to thank İsmail Uyanık for his numerous help, thoughtful advices and guidance throughout my whole university life.

I am thankful to Ali Nail İnal for his endless patience and unending helps to me in my thesis works.

I am also grateful to Caner Odabaş for his valuable cooperation in courses and motivating my thesis works.

I would like to give my special thanks to my friends Onur Berkay Gamgam, Murat Aslan and Emrah Topal for listening me and motivating my academic works.

I would like to thank Bilkent University and Electrical and Electronics Engineering Department members for the experiences that I had acquired, and the support that including the use of department computers and materials.

I also thank to The Scientific and Technological Research Council of Turkey

(TÜBİTAK) foundation for their financial support to my academic research.

Finally, I am very thankful to my parents Zehra and Sezai Çatalbaş and my brother Burak Çatalbaş for their precious life-long support, patience and cooperations.

Contents

List of Figures

List of Tables

to My Dear Family

Chapter 1

Introduction

In this chapter we will give information about our motivation, contribution and content of this thesis and make a brief overview to thesis topics.

1.1 Motivation

Recurrent neural networks (RNN) with short term memory neuron model have the best scores in handwriting recognition benchmark problems, see [1]. Moreover, some other configuration of RNNs gives good performance in spoken language understanding problems, see [2]. In addition to these, they may e employed to estimate next character or word of a text sequence, see [3]. Besides all these, it is possible to use RNNs in areas such as robot posture control, see [4]. Their wide application area makes them a potential solution way of many problems.

Legged locomotion is a desired ability for robotic platforms but sustaining its stability may be a complex problem in some cases. One of the interesting control way of legged robot platforms would be the use of recurrent neural network (RNN) type of controllers. However, training of recurrent neural networks demonstrates different features and difficulties than training of classical feedforward type of

neural networks, see e.g. [5], [6]. There are some proposed techniques for feedforward type of neural networks and some of them has been already generalized to RNNs to overcome related problems with training of RNNs.

This thesis evaluates usefulness of regularization, acceleration and stability enhancing techniques, which are generally developed for feedforward type neural network, in training of RNNs with backpropagation through time algorithm (BPTT). After that, stable locomotion generation ability of RNNs are tested with a biped robot platform. Finally, we review what has been done and determine open problems as future works.

1.2 Contributions

We performed simulations to assess the usefulness of various acceleration, regularization and stability enhancing techniques in RNN training. In terms of these simulations, momentum and cross-entropy cost function are found as successful acceleration techniques over training sets which consist of single pattern or multi patterns. However, modified delta-bar-delta learning rule can accelerate training only for single training pattern.

In the scope of this thesis, three regularization techniques are tested and their success evaluated. Although L1 and L2 regularization did not succeed too much by regularization aiming, promising results were obtained with different application ways of dropout techniques.

During training, time constant and gradient limitation techniques are utilized for the purpose of enhancing stability. Then, their successes, advantages, and drawbacks are evaluated in terms of performed simulations.

Performance of activation function are compared with each other and rectifier function is found as a successful and efficient activation function. Finally biped robot model is driven with a trained RNNs and qualification of RNNs in

generating stable walking has been showed with simulations results.

1.3 Organization of Thesis

In this chapter, we give information about motivation, contribution and content of this thesis and make a brief introduction to thesis topics.

In the second chapter, different motion generating network types are introduced at various complexity levels and examples of them are given in there. Furthermore, design criteria of central pattern generators (CPG) and their difference with RNNs are explained. Finally implementation of BPTT algorithm to RNN having leaky integrator model neurons given in detail.

In the third chapter, main problems of RNN training are defined as long training times, low input generalization ability, and training instabilities and these problems are tried to be solved with acceleration, regularization and stability enhancing techniques which are generally proposed for feedforward type of neural networks and some of them has been already applied to RNNs to overcome related difficulties with training of RNNs.

In the fourth chapter, we try to teach walking biped robot leg angles to RNNs with various activation functions and compare effects of activation functions to training performance. Moreover, we introduce another learning stability enhancing technique and discuss its effects on training. Finally, we drive biped robot model with trained RNN and check its stable locomotion generation ability.

In the conclusion, we review what has been done in the thesis. Then, we determine open problems as future works. Finally, we propose possible solution ways of to some of these problems.

Chapter 2

Background: Motion Generating Network Structures and Learning Algorithms

Locomotion is an important topic in robotic field and there are different locomotion generation methods such as center of gravity and zero moment point based inverse kinematic and equations of motion depended solutions, see e.g. [7], [8]. However, there are some other methods, they rely on rhythm generator networks which have different forms and design principles, see [9] and [10]. In this Chapter we give further information about various rhythm generator networks and focus on fully recurrent neural networks (RNN).

2.1 Rhythm Generation Oscillators

There are rhythmic pattern generation methods which include use of coupled differential equations mostly. The smallest unit of these type of networks are rhythm oscillators. By combining these oscillators central pattern generator (CPG) are

designed and applied to various control problems. Their robustness against perturbations and computational efficiency made them a popular and interesting research topic, see [11].

2.1.1 Matsuoka Oscillator

Matsuoka oscillator is a well-known oscillator model which is utilized to generate rhythmic patterns and their outputs may be easily modified with tonic inputs, see [12] and [13]. By these abilities, they reached a wide application area in robotic field especially with use of CPG, see e.g. [14], [15], [16] and [17]. In addition to these, [12] and [13] explain output patterns of different combination of these type mutual inhibition networks with more than two neurons and they specify possible ways of controlling frequency and pattern of them in detail. The generic model of a Matsuoka Oscillator is given in (2.1)-(2.4).

$$\tau_i \dot{x}_i = -x_i + \sum_{j=1}^{2} w_{ij} y_j - \beta f_i + s_i \tag{2.1}$$

$$\tau_i' \dot{f}_i = -f_i + y_i \tag{2.2}$$

$$y_i = g(x_i) \tag{2.3}$$

$$v = x_1 - x_2 \tag{2.4}$$

where x_i, w_{ij}, y_j, f_i, s_i, τ_i and τ_i' stand for internal state of ith neuron, coupling weight from jth to ith neuron, output of jth neuron, self-inhibition effect of the ith neuron, external input of ith neuron and time constants in terms of [16]. In the CPG applications one oscillator which consist of two cells is used per joint of robot.

2.1.2 Amplitude Controlled Phase Oscillator

In terms of [18], Amplitude Controlled Phase Oscillator (ACPO) are modified version of Phase Oscillators (PO) which may be the one of the simplest oscillator

type. Although PO and ACPO are simple oscillators, they have a large implementation area in robotics such as modular robots, snake robots and robot arms (see [19], [20] and [21]). Their simple structures enable us to define direct contribution of its each parameter to output of oscillator. For this reason, oscillator structure enables researchers to find useful ways of modulating and programming their output such as Fourier analysis based (see [21]) and Powell's method based (see [19]) techniques. The employed ACPO oscillator model in [20] is given in (2.5)-(2.7) and its variables are explained below.

$$\dot{\theta}_i = 2\pi v_i + \sum_j r_j w_{ij} sin(\theta_j - \theta_i - \phi_{ij}) \tag{2.5}$$

$$\ddot{r}_i = a_i(\frac{a_i}{4}(R_i - r_i) - \dot{r}_i) \tag{2.6}$$

$$x_i = r_i(1 + \cos(\theta_i)) \tag{2.7}$$

where θ_i, v_i, a_i denotes phase of oscillator, intrinsic frequency, rate of convergence of r_i to R_i of oscillator output amplitude, respectively. Coupling weight w_{ij} and phase bias ϕ_{ij} determine the interaction conditions with other oscillators in network such as phase lags between oscillators. In the robot control problems with CPG implementations, one oscillator is used per joint of robot.

2.1.3 Van Der Pol Oscillator

Cell model of Van Der Pol (VDP) oscillator is given in Figure (2.8), see [11]. VDP oscillator is a second order differential equation so it is enough to use one oscillator cell per each joint in CPG applications.

$$\ddot{x} - \alpha(p^2 - x^2)\dot{x} + w^2 x = 0 \tag{2.8}$$

where α, p and w defines shape, amplitude frequency of output generally but p and α also have effect on frequency of oscillation. Output of oscillator is denoted with x variable. Coupling of VDP oscillator with others requires modification given in (2.9), see [11].

$$\ddot{x}_i - \alpha(p_i^2 - x_{ai}^2)\dot{x} + w^2 x_{ai} = 0 \tag{2.9}$$

where i denotes cell number and x_{ai} is weighted sum of neighbor cells as given in (2.10).

$$x_{ai} = \sum_{j} w_{ij} x_j \tag{2.10}$$

2.2 Central Pattern Generator Networks

Central pattern generators consist of networks of coupled oscillators which are able to generate periodic waveforms, see [21]. Central adjective denotes that oscillator network can sustain oscillation without taking any sensory feedback [10]. In addition to these, single oscillator is placed per each joint of robot in the standard configuration of CPGs hence coupling oscillators of CPG with feedback signals become easy. Under some conditions which depend on the chosen oscillator type, coupling of these oscillators may carry specific phase difference and output frequency through robot body. Thus, CPGs have capability of generating stable motion with various physical design such as [19] which claims that CPG type controller are capable of producing stable locomotion with different physical structures of a modular robot. In addition to their stable oscillation generation abilities, they serve designers who try to control robots which have high degree of freedom by restricting their parameter spaces which is a suitable way of reducing dimensionality of robot model. Also, designers can control a robot with limited number of input variable via CPG approach.

There are different CPG network designs and also ways of determining parameters of oscillators. In the design of CPG networks, we first need to choose suitable oscillator model in terms of our control problems after that parameters of oscillators need to be determined. At this stage there are mainly two ways; supervised and unsupervised learning methods. Gradient descent learning algorithms are good examples for supervised learning method. These type methods need a desired data set and try to minimize a error function which is difference between desired set patterns and CPG output. Evolutionary algorithms and reinforcement type algorithm are good examples for unsupervised learning algorithms, see [22] and [23]. In these type algorithms, designer gives the definition of a good network

output and learning algorithm tries to find out oscillator parameters which satisfy desired behaviours.

Although CPGs have these advantages, they are able to produce limited amount of output pattern because of their dimensionality reduction property. These limitations prevent multipurpose use of limbs. However, fully recurrent neural networks (RNN) type controllers allow multipurpose usage of limbs. For this reason, we will focus on RNNs in the remaining part of the thesis.

2.3 Recurrent Neural Networks

Recurrent neural networks consist of neuron units which have mutual connections with other neurons of neural network. Definition of these neuron units may also involve differential terms and these differential equations may help them to behave in a similar way to biological neuron models, see [24]. As an illustration of these, equations of Leaky integrator neuron model is given in (2.11) and (2.12), see [25]. Leaky integrator neuron model is well-known and simple neuron model and its output is similar to simplified version of EEG signals [26].

$$\tau_i \frac{\mathrm{d}y_i(t)}{\mathrm{d}t} = -y_i(t) + \sigma(x_i(t)) + I_i^{ext}(t) \quad i = 1, 2, \ldots, N \tag{2.11}$$

$$x_i(t) = \sum_{j=1}^{N} w_{ij} y_j(t) \tag{2.12}$$

where parameters y, w, τ, x, σ and I^{ext} are activation level vector, connection weight matrix to be modified by the learning, time constant vector to be modified by the learning, weighted input to neuron unit, neuron activation function, and time-varying vector of external input, respectively.

Typical structure of a RNN is given in Figure 2.1. Leaky integrator neuron model is a continuous neuron model and its network is also a continuous time recurrent neural network (CTRNN) so we need to employ some numerical methods in order to determine output of these type of RNNs. When we apply Euler

method in (2.13) to (2.11) with Δt time steps, we obtain discrete version of (2.11) as shown in (2.14).

$$\frac{\mathrm{d}y_i(t)}{\mathrm{d}t} \approx \frac{y_i(t+\Delta t) - y_i(t)}{\Delta t} \tag{2.13}$$

$$\tilde{y}(t+\Delta t) = \left(1 - \frac{\Delta t}{\tau_i}\right) \tilde{y}(t) + \frac{\Delta t}{\tau_i} \sigma(\tilde{x}_i(t)) + \frac{\Delta t}{\tau i} I_i^{ext}(t) \tag{2.14}$$

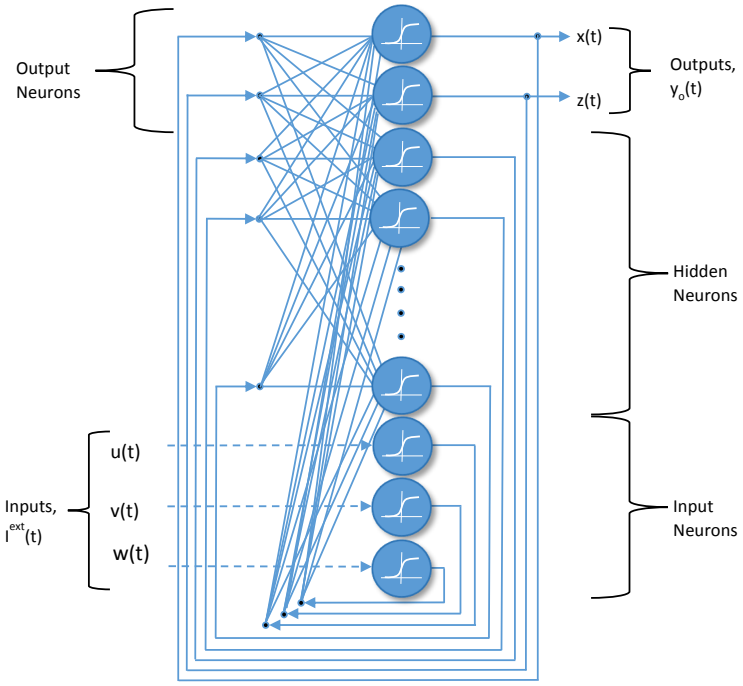

Figure 2.1: Recurrent neural network configuration.

where network consists of two output neurons, unspecified number of hidden neurons and three input neurons.

When we demonstrate change of networks inputs and outputs with Δt time steps, we reach a layered structure in time such as shown in Figure 2.2. In this figure, each column denotes same neuron but in a different time step. Basically neurons take their previous outputs as inputs to generate new outputs at a time step. From this point of view, it seems similar to deep neural networks (DNN) but RNN has outputs at each time layer so its learning procedure becomes more complicated and the use of backpropagation algorithm is prevented with training purposes. At this stage, well known learning technique backpropagation, which is used in training of DNN widely, is generalized to RNNs in order to train RNNs

efficiently, see [27]. The name of this new algorithm is Backpropagation Through Time (BPTT) and we explain the implementation of BPTT in the Section 2.4 in detail.

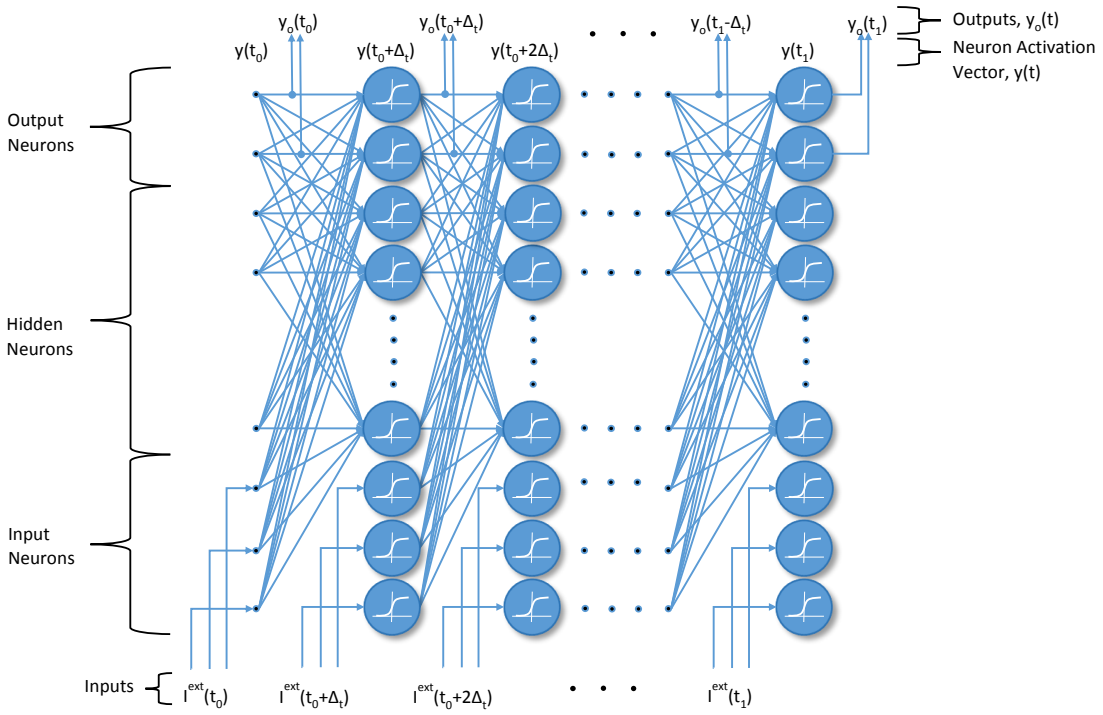

Figure 2.2: Recurrent neural network configuration.

2.4 Back-propagation Through Time

Leaky integrator neuron model is given in (2.11)-(2.12) and (2.14). In terms of [28], we first need to define variation of error with respect to of activation of an output neuron in order to begin the propagating error through network so we define (2.15).

$$e_i(t) = \frac{\delta E}{\delta y_i(t)} \tag{2.15}$$

Let us choose classical mean squared error (MSE) definition which is given in (2.16) in which y_i and d_i denote activation level of output neurons and desired activation level, respectively.

$$E = \frac{1}{2}\sum_i \int_{t_0}^{t_1} (d_i(t) - y_i(t))^2 \mathrm{d}t, \text{ for output neurons} \tag{2.16}$$

In terms of MSE definition, variation of error with respect to activation of output neuron takes the form of (2.17)

$$e_i(t) = \frac{\partial E}{\partial y_i(t)} = y_i(t) - d_i(t), \text{ for output neurons, otherwise } e_i(t) = 0 \tag{2.17}$$

Error gradient with respect to activation of a neuron is given in (2.18) at continuous time.

$$\frac{dz_i}{dt} = \frac{1}{\tau_i} z_i - e_i - \sum_j \frac{1}{\tau_j} w_{ij} \sigma'(x_j) z_j \text{ with boundary condition } z_i(t_1) = 0 \tag{2.18}$$

By employing Euler method, which is given in (2.13), discrete version of (2.18) can be found as in (2.19).

$$\tilde{z}_i(t) = \left(1 - \frac{\Delta t}{\tau_i}\right) \tilde{z}_i(t+\Delta t) + \Delta t \sum_j^N \frac{1}{\tau_j} w_{ij} f'(x_i(t)) \tilde{z}_j(t+\Delta t) + \Delta t e_i(t) \text{ with } \tilde{z}_i(t_1+\Delta t) = 0 \tag{2.19}$$

Continuous time and their discrete versions of error gradients with respect to network parameters are given in (2.20) and (2.21).

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \frac{1}{\tau_i} \int_{t_0}^{t_1} z_i(t) \sigma'(x_i(t)) y_j(t)\, \mathrm{d}t \\ &\approx \Delta t \sum_{k=1}^{\frac{t_1 - t_0}{\Delta t}} \frac{1}{\tau_i} z_i(t_0 + k\Delta t) \sigma'(x_i(t_0 + (k-1)\Delta t)) y_j(t_0 + (k-1)\Delta t) \end{aligned} \tag{2.20}$$

$$
\begin{aligned}
\frac{\partial E}{\partial \tau_i} &= -\frac{1}{\tau_i}\int_{t_0}^{t_1} z_i(t)\frac{\mathrm{d}y_i(t)}{\mathrm{d}t}\,\mathrm{d}t \\
&= -\frac{1}{\tau_i^2}\int_{t_0}^{t_1} z_i(t)[-y_i(t)+\sigma(x_i(t))+I_i^{ext}(t)]\,\mathrm{d}t \\
&\approx -\frac{\Delta t}{\tau_i^2}\sum_{k=1}^{\frac{t_1-t_0}{\Delta t}} z_i(t_0+k\Delta t)[-y_i(t_0+(k-1)\Delta t) \\
&\quad +\sigma(x_i(t_0+(k-1)\Delta t))+I_i^{ext}(t_0+(k-1)\Delta t)]
\end{aligned}
\tag{2.21}
$$

Update rule of connection matrix w and time constant vector τ are given in (2.22) and (2.23), respectively. Hence network parameters move in negative gradient direction and error is minimized.

$$
\text{new } w_{ij} = w_{ij} + \Delta w_{ij} = w_{ij} - c\frac{\partial E}{\partial w_{ij}} \tag{2.22}
$$

$$
\text{new } \tau_i = \tau_i + \Delta \tau_i = \tau_i - c\frac{\partial E}{\partial \tau_i} \tag{2.23}
$$

Chapter 3

Application of Various Regularization and Acceleration Techniques for Recurrent Neural Network Training

In [27], back-propagation algorithm has been generalized to recurrent neural networks (RNN) and finite length pattern generation aimed training procedure of this type of networks have become a topic of interest afterwards. Despite of these developments, predefined pattern training in this of type networks still have some difficulties, which is a subject under investigation. To demonstrate the possible problems encountered in the training of RNN's as a pattern generator, in this chapter we will consider the generation of Figure 8 pattern which appears to be inherently difficult and has been investigated in various papers, see e.g. [25]. Although a RNN with ten hidden neurons may be able to reproduce single Figure 8 at the end of the training, see [25], RNN with thirty hidden neurons may not repeat the same performance corresponding to given, untrained but close inputs to trained network with eight different training pattern, see [28]. This is a well-known problem encountered in the training of RNN's and shows their

possible drawbacks for possible applications, and it may cause unpredictable errors. When different inputs like test set inputs are applied to recurrent neural networks, even if network was trained with close inputs and learnt them well, generated patterns can be deformed in an undesired way. Although predefined structures and restricted connections of oscillator schemes prevent this type of problems, larger capabilities of RNNs are still worth trying to solve these difficulties. Basically, this problem is similar to overfitting phenomenon and there are some regularization techniques which have been developed to overcome this problem in feedforward type of networks. In addition to this problem, training time is another important criterion for both of feedforward and recurrent type of neural networks and there are some ways to decrease the convergence time for feedforward type of networks.

In this chapter, we have evaluated performance of various acceleration and regularization techniques in pattern generation with RNN, applied to the generation of various Figure 8 patterns.

3.1 Simulation Template

Figure 8 pattern generation problem is taken from [28], in order to be used as a test problem for interpreting performance of applied algorithms and it is a well-known benchmark problem for the evaluation of both training and testing performance of RNN, see e.g. [25] and [29]. The Figure 8 have crossing points so that it requires hidden neurons to memorize the state, thus problem becomes a hard enough problem for both training and testing.

For the generation of the both training and test sets, the following formula given by (3.1) is utilized. In (3.1), θ represents the rotation angle, t is the time, the bias term $(0.5, 0.5)^t$ represents the center of the Figure 8. As a typical figure for e.g. $\theta = 0$, see the first figure in Figure 3.1. The training set outputs are chosen with equally spaced eight θ values for $\theta = n\pi/4, n = 0, 1, ..., 7$. These training set values are shown in Figure 3.1. For the test patterns we chose $\theta =$

$n\pi/4 + \pi/8, n = 0, 1, ..., 7$, as shown in Figure 3.2. To generate these signals, we consider a neural network structure with 2 outputs, representing x and y components and 3 inputs, which are chosen as

$$y_o(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = 0.4 \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} \sin(\pi t/16) \\ \sin(\pi t/8) \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \tag{3.1}$$

$$I^{ext}(t) = \begin{bmatrix} u(t) \\ v(t) \\ w(t) \end{bmatrix} = \begin{bmatrix} \sin(\theta) \\ \cos(\theta) \\ 0.5 \end{bmatrix} \tag{3.2}$$

These patterns are employed to evaluate acceleration capabilities of employed algorithms in following sections. First and second rows of Figure 3.1 and Figure 3.2 may seem similar but there are 180 degree difference between them and motion directions are specified with arrows to prevent any confusion in these figures.

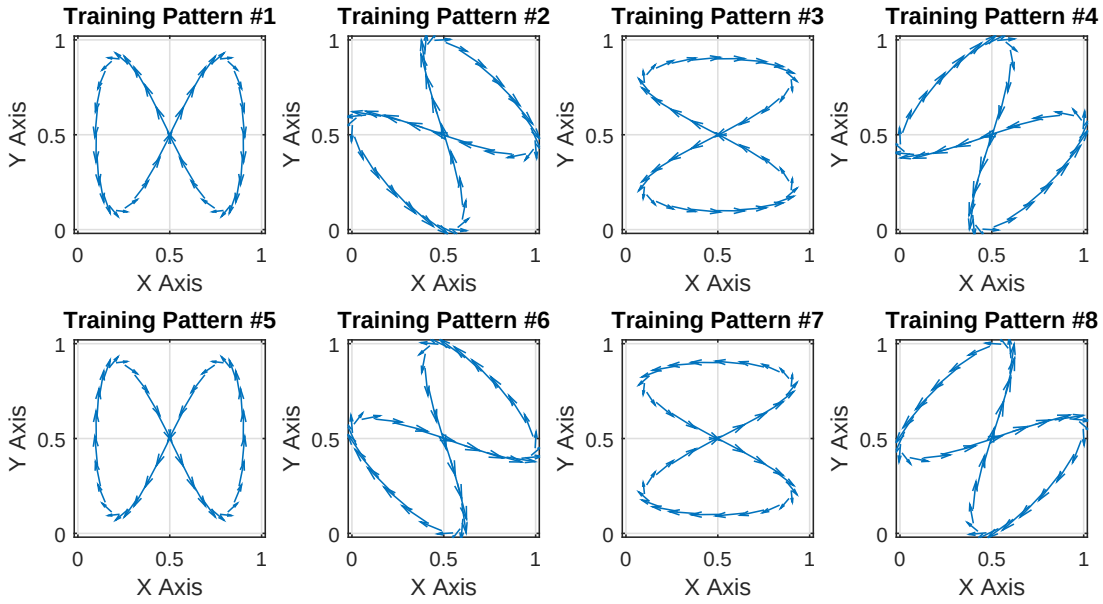

Figure 3.1: Training Set Patterns.

In the remaining part of this chapter, same neural network configuration, see 2.1, and initial values were employed to compare the results of different algorithms, unless any difference is specified for simulations. In this network, 2

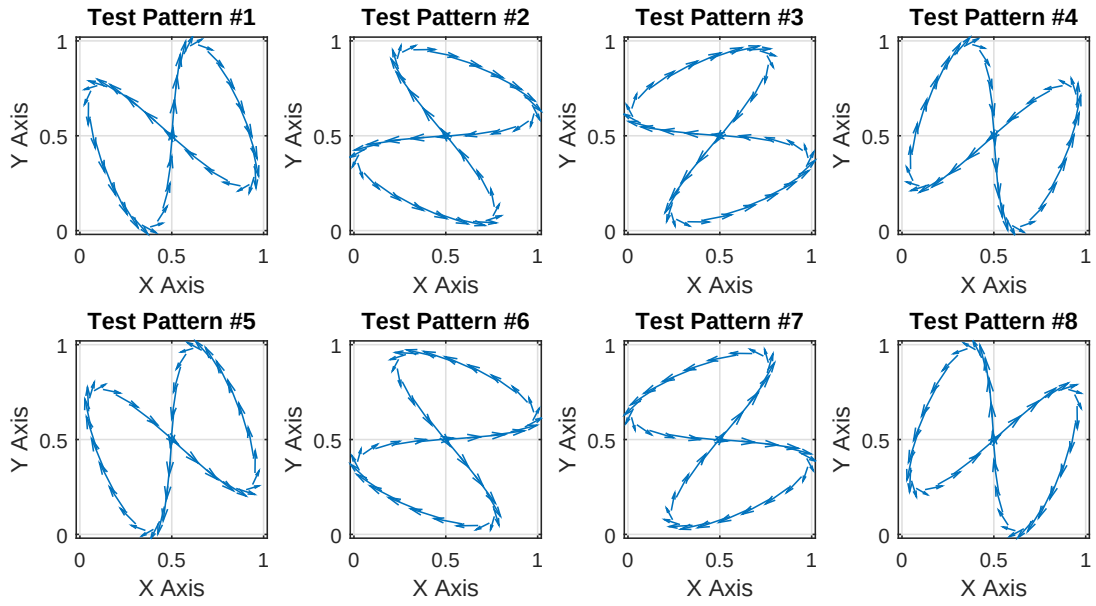

Figure 3.2: Test Set Patterns.

output, 30 hidden and 3 input neurons with sigmoid type of activation function were trained with benefiting from back-propagation through time algorithm, see (2.11)-(2.23). In addition to these, definition of sigmoid function is given in (3.3). In the learning process, time constant of each neuron unit was initialized as one, same initial w connection weight matrix, which has zero incoming weights for three input neurons, was utilized all time. Time constants and connection weights of input neurons were not updated until the end of the training procedure. Furthermore, desired patterns consist of two period of figure 8 data, in order to be sure for learning of periodicity of pattern by trained network.

$$\sigma(x_i(t)) = \frac{1}{1 + e^{-x_i(t)}} \tag{3.3}$$

3.2 Learning Constant

The selection of learning constants is an important step of implementation part of gradient descent type learning algorithms. Optimum learning constant needs to be small enough in order to track negative gradient direction, so that undesired oscillations can be prevented. Also, it needs to be big enough to be able to finish learning in a reasonable time. In the feedforward type of neural networks, suitable learning constant may be found by trial and error method most of time, although there are some techniques to find a good starting point for fine tuning operation. In this section, learning constant selection was performed with trial and error method by benefitting from simulations for RNN.

In these simulations, we considered the training algorithm given by (2.11)-(2.23), with a neural network structure has 3 input neurons, 30 hidden neurons and 2 output neurons as shown in Figure 2.1. During training process, inputs of network are generated with (3.2) for $\theta = n\pi/4, n = 0, 1, ..., 7$ values and network is trained with corresponding 8 training patterns in Figure 3.1. In test part, inputs of network are generated with (3.2) for $\theta = n\pi/4 + \pi/8, n = 0, 1, ..., 7$ values and output of network is compared with corresponding 8 test patterns in Figure 3.2. The results of performed simulations for different learning constants, $c = 0.1, 0.3, 1, 3$ and 10 are given in Figure 3.3.

In terms of Figure 3.3, large learning constants accelerate training but may cause oscillations as seen for $c = 10$, on the other hand, small learning constants may cause overfitting and postpone convergence over training set as occurred for $c = 0.1$ and $c = 0.3$. From this point of view, it can be concluded that the best learning constant for this configuration is $c = 1$ and as a result RNN showed a similar behavior with behaviors of feedforward type of neural networks for different learning constants. Error value change of each of training and test patterns can be seen in Figure 3.4 along training process. In terms of Figure 3.4, after some point further training decreases test set performance generally and this is another similar behavior with feedforward type neural networks. Figure 3.5 is outputs of trained RNN at the end of the training and it can be seen that there

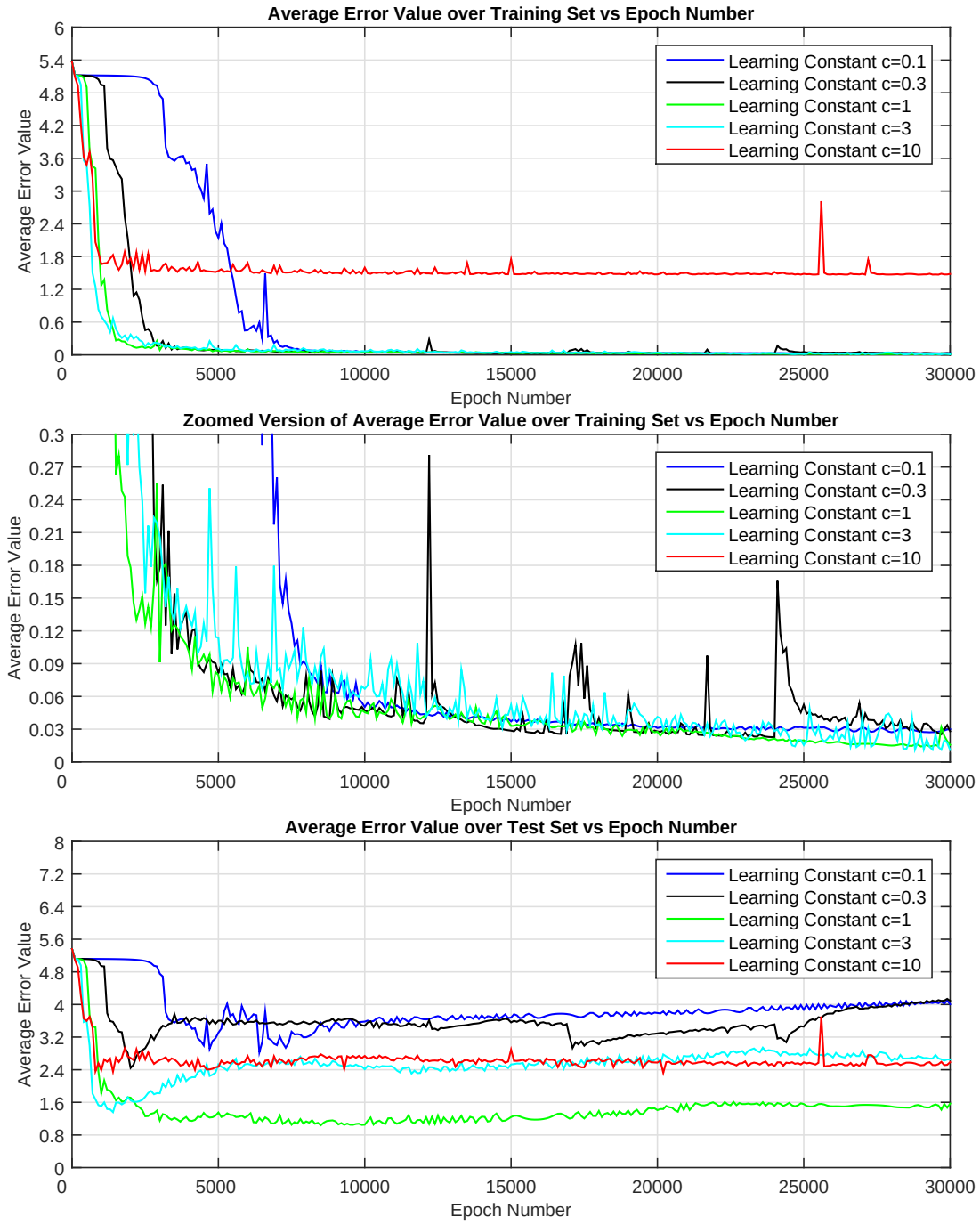

Figure 3.3: Performance comparison for different learning constant parameters along training.

are deformations in addition to expected required rotation in test set patterns when training set patterns are very similar to desired ones. This means networks partially learnt how to generalize output of network according to given inputs by applied training, but its success rate is very low when it is compared with training set results. In the remaining part of this chapter, learning rate $c = 1$ will be added to all comparison plots and it will be used as a base performance level.

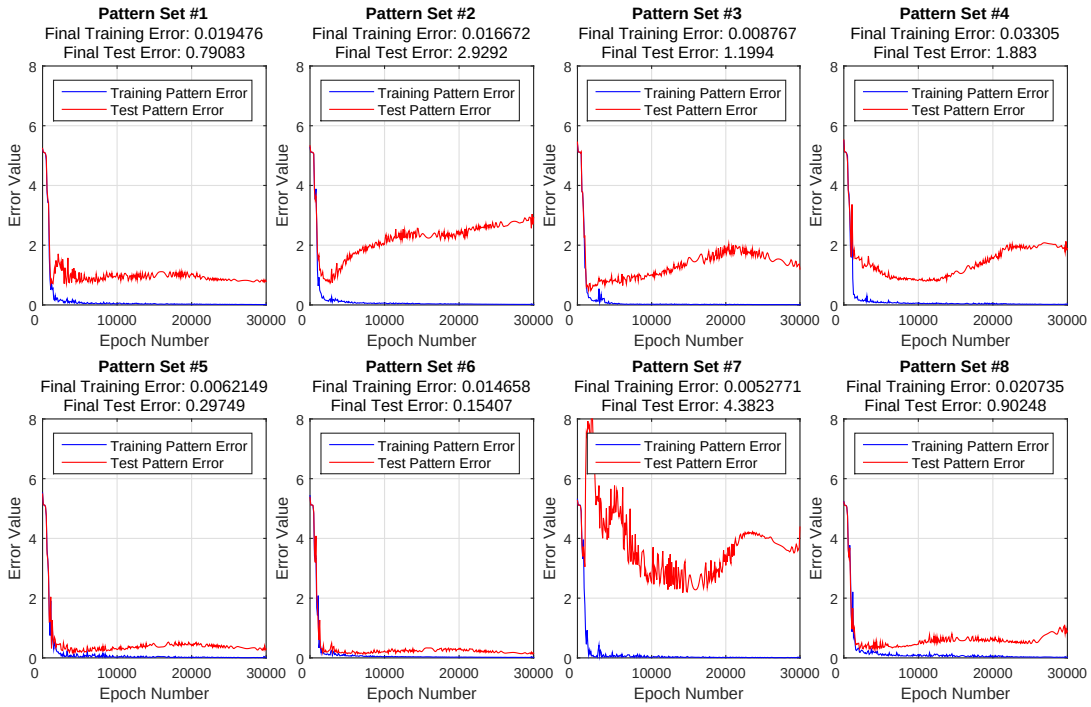

Figure 3.4: Changes on error values of test and training patterns for learning constants $c = 1$ along training.

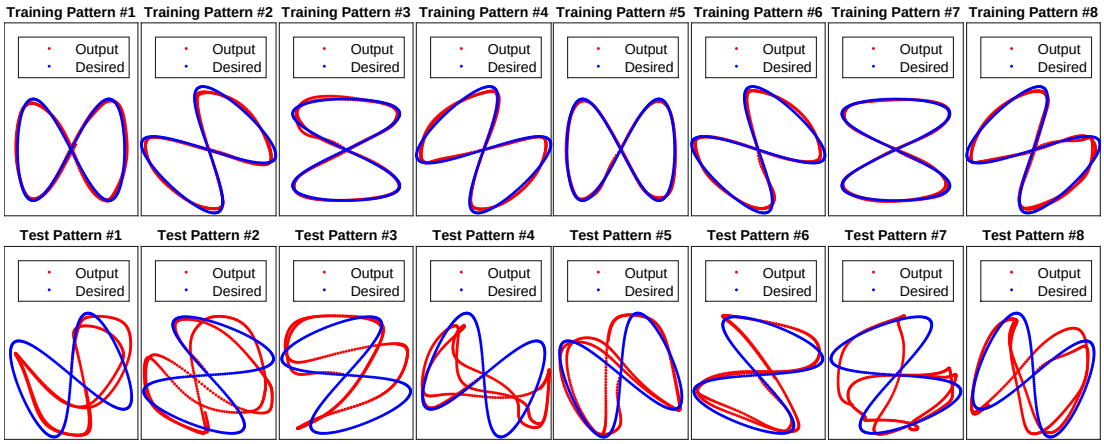

Figure 3.5: Output of trained recurrent neural network for learning constant $c = 1$.

3.3 Modified Delta-Bar-Delta Learning Rule

Importance and hardness of selection of suitable learning constant has directed researchers to find an autonomous way for this operation. In this section, one algorithm which automatize most of selection process of the learning constant is explained and related simulation results are given in Section 3.3.1 and Section 3.3.2. The algorithm, which is called delta-bar-delta learning rule, is proposed for feedforward type of neural networks in [30], and it is modified and applied to RNNs in [31]. In this algorithm, each trainable parameter of neural network has its own learning rate and this differentiates the algorithm from original gradient descent algorithm, which has one global learning rate for all parameters. By this algorithm, performance improvement has been obtained in training of neural networks, because learning rates are adjusted in terms of instant needs of training process continuously.

In terms of proposed algorithm, each parameter in network is updated with (3.4)), and (3.5) instead of (2.22), and (2.23), respectively.

$$w_{ij}(t) = w_{ij}(t-1) + \Delta w_{ij}(t) = w_{ij}(t-1) - c_{ij}(t)\frac{\partial E}{\partial w_{ij}} = w_{ij}(t-1) - c_{ij}(t)\delta_{ij}(t) \tag{3.4}$$

$$\tau_i(t) = \tau_i(t-1) + \Delta \tau_i(t) = \tau_i(t-1) - c_i(t)\frac{\partial E}{\partial \tau_i} = \tau_i(t-1) - c_i(t)\delta_i(t) \tag{3.5}$$

Secondly, learning rate of each parameter in network is updated in following way

$$c_{ij}(t) = c_{ij}(t-1) + \Delta c_{ij}(t) \tag{3.6}$$

$$c_i(t) = c_i(t-1) + \Delta c_i(t) \tag{3.7}$$

Finally, rate of change of learning rates are determined by

$$\Delta c_{ij}(t) = \begin{cases} K & \text{if } \delta_{ij}(t-1)\delta_{ij}(t) > 0 \\ -\phi c_{ij}(t) & \text{if } \delta_{ij}(t-1)\delta_{ij}(t) < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.8}$$

$$\Delta c_i(t) = \begin{cases} K & \text{if } \delta_i(t-1)\delta_i(t) > 0 \\ -\phi c_i(t) & \text{if } \delta_i(t-1)\delta_i(t) < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.9}$$

where K, ϕ, c_{ij} and c_i are additive increase, multiplicative decrease, learning rate of connection matrix and time constants parameters, respectively. In addition to this equation, additive increase parameter, K, needs to be adjusted when error rate of network was too small, a global parameter, Λ, was introduced to algorithm with Formula (3.10) in order to make proportional additive increase to instant error rate.

$$K = \Lambda E(t) \tag{3.10}$$

In the following, we will apply this algorithm for the training and testing of Figure 8 patterns. In the following simulations, we will use (2.11)-(2.21) and (3.4)-(3.9) for a RNN structure with 3 input neurons, 30 hidden neurons and 2 output neurons as shown in Figure 2.1.

During the simulations, it is realized that temporary spikes in error value, E, decreases stability and speed of algorithm. In order to prevent this problem, an error upper bound which is close to initial error value of untrained network is set and usage of this modification is specified in comparison plots with $min(E, 5)$ term instead of only E term. Moreover, 0.05 is chosen as lower bound of time constants to enhance stability of training procedure. The reason of time constant limitation is explained in the Section 3.9.

3.3.1 Single Pattern Training

In [31], only one pattern was trained and there was no test set result. In this thesis, this algorithm was run for two times. In the first run, single pattern training for $\theta = 0$ degree was performed for configuration of ϕ=0.5, see (3.8) and (3.9), and some Λ parameters, then its performance was compared with performance of constant learning rate utilization in Figure 3.6.

In terms of Figure 3.6, it is easy to say that high Λ usage, like $\Lambda = 0.2$, with maximum error value limitation is more successful than low Λ usage such as $\Lambda = 0.01$ with or without maximum error limitation because it decreases the convergence time of network successfully. On the other hand, when we compare

$\Lambda = 0.2$ and $\Lambda = 0.01$ utilization without maximum error limitation configurations, it can be concluded that high Λ usages, like $\Lambda = 0.2$, without maximum error value limitation, effect the performance of algorithm in a negative way as seen in Figure 3.6. From this point of view, error upper bound utilization is a beneficial tool for RNN training with this algorithm. Furthermore, algorithm generally outperformed constant learning rate utilization because it gives better results under maximum error limitation than constant learning rate $c = 1$ utilization in training and test pattern as seen in Figure 3.6. The performance of trained RNN with maximum error limitation, time constant limitation and Λ=0.2 configuration can be examined with Figure 3.7, and Figure 3.8. These figures shows that network learnt well Pattern set #1 which consists of one training and one test patterns for $\theta = 0$ and $\theta = 22.5$ degree, respectively but it cannot show same performance over other training and test patterns because network was trained for only first training pattern so network cannot give any reasonable meaning to the corresponding inputs of other patterns.

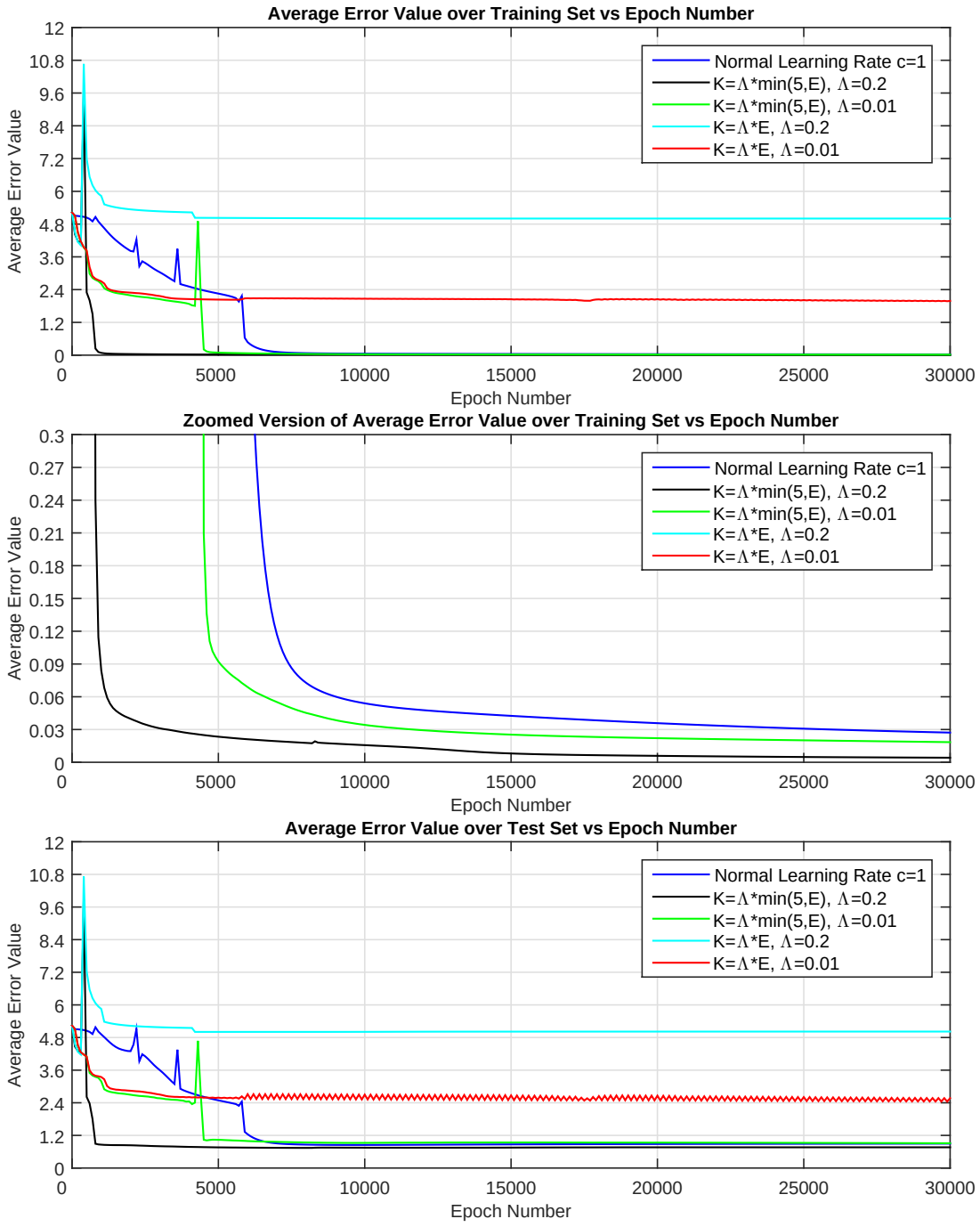

Figure 3.6: Performance comparison for different Λ parameters along training.

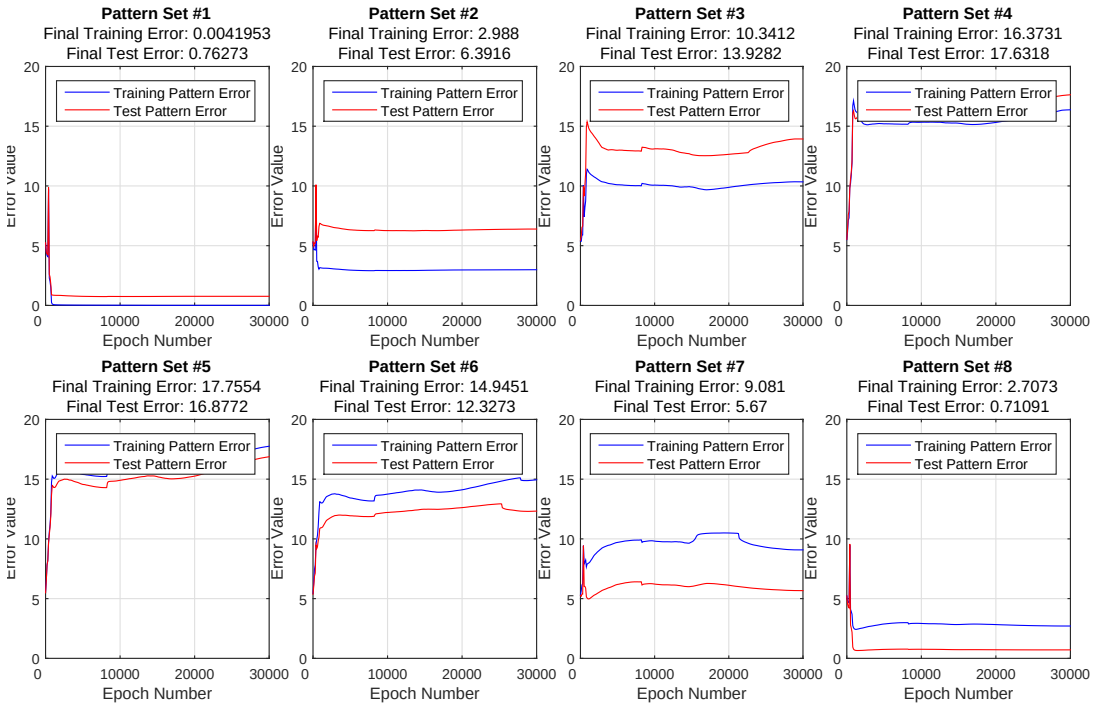

Figure 3.7: Changes on error values of test and training patterns for $\Lambda = 0.2$ along training.

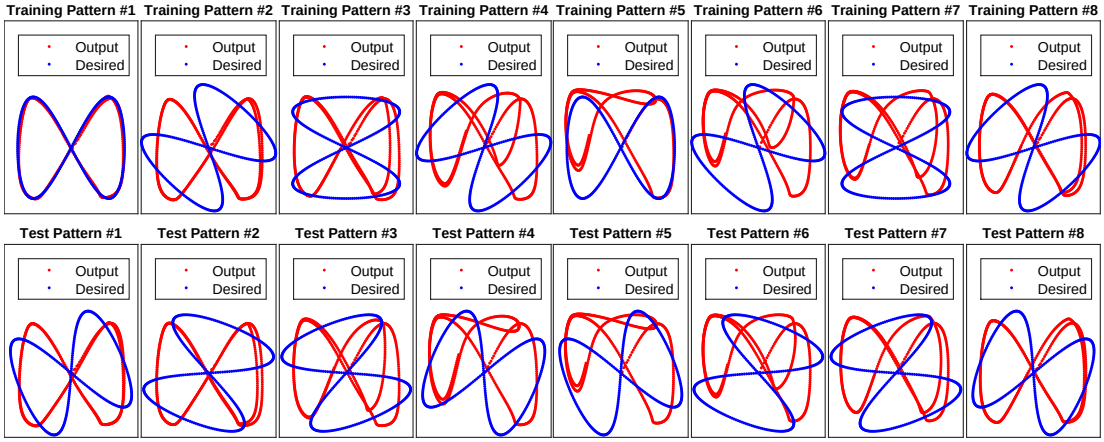

Figure 3.8: Output of trained recurrent neural network for $\Lambda = 0.2$.

3.3.2 Multi Pattern Training

In [31], there was no result about success of proposed algorithm over a set of training patterns. In the second run, multi pattern training for $\theta = n\pi/4, n = 0, 1, ..., 7$ degree was performed for configuration of ϕ=0.5, see (3.8) and (3.9), and some Λ parameters, then its performance was compared with performance of constant learning rate utilization in Figure 3.9.

Although proposed algorithm accelerates single pattern training, it was neither able to accelerate training process and nor able to track negative error gradient direction as seen in Figure 3.9. Even if maximum error value was applied to all training trials, a stable training was not obtained for $\Lambda = 0.01$ and $\Lambda = 0.2$. For $\Lambda = 0.2$, training patterns are taught until approximately 9500^{th} epoch but after it network diverged from negative gradient direction. If we examine test set performance of algorithm in Figure 3.9, it is also unsuccessful inevitably. Hence as a result, it appears that this method is not suitable for the training of a set of patterns as it is and apparently needs some modifications but this point needs further investigations.

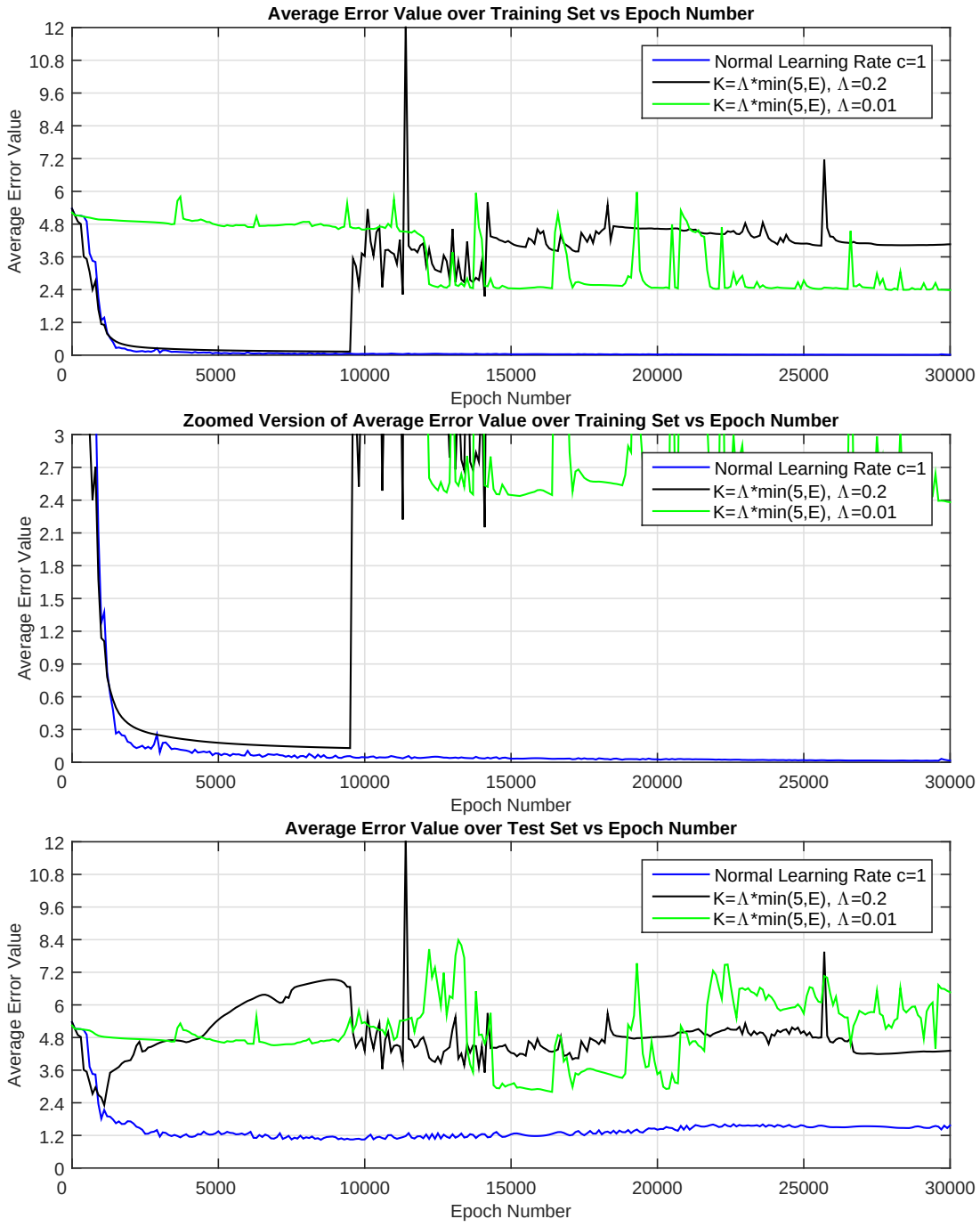

Figure 3.9: Performance comparison for different Λ parameters along training.

3.4 Momentum

Momentum is a widely utilized and successful training acceleration technique in feedforward type of neural networks. In terms of [30], momentum term increases its weight in update functions when consecutive error gradients have similar directions thus it shows a similar behaviour with adaptive learning rate algorithm, given in Section 3.3. In addition to its training acceleration capability, it also increases stability of training procedure. At the same time, it is a suggested acceleration technique for recurrent neural networks, see [28], and [31]. We utilize this technique in our work in order to compare its training and test set performances with other techniques.

Momentum method is implemented by modifying (2.22), and (2.23) as in (3.11), and (3.12) as given in [30].

$$w_{ij}(t) = w_{ij}(t-1) + \Delta w_{ij}(t) = w_{ij}(t-1) - (1-a)c\frac{\partial E}{\partial w_{ij}} + a\Delta w_{ij}(t-1) \quad (3.11)$$

$$\tau_i(t) = \tau_i(t-1) + \Delta \tau_i(t) = \tau_i(t-1) - (1-a)c\frac{\partial E}{\partial \tau_i} + a\Delta \tau_i(t-1) \quad (3.12)$$

where $0 < a < 1$ is the momentum constant, c is the learning constant.

In these simulations, we considered the training algorithm given by (2.11)-(2.21) and (3.11)-(3.12), with a neural network structure has 3 input neurons, 30 hidden neurons and 2 output neurons as shown in Figure 2.1. During training process, inputs of network are generated with (3.2) for $\theta = n\pi/4, n = 0, 1, ..., 7$ values and network is trained with corresponding 8 training patterns which are output of (3.1). In test part, inputs of network are generated with (3.2) for $\theta = n\pi/4 + \pi/8, n = 0, 1, ..., 7$ values and output of network is compared with corresponding 8 test patterns which are output of (3.1). The results of performed simulations for different momentum constants, $a = 0, 0.1, 0.4, 0.5$ and 0.9 are given in Figure 3.10.

First of all, convergence was obtained with all tried momentum constants as can be seen in Figure 3.10. Even if $a = 0.1$ was a small momentum constant selection, it increased the speed of convergence of network over training set and

enhanced generalization capability of network over test set. Although momentum constants $a = 0.4$ and $a = 0.5$ were bigger than $a = 0.1$, they were not able to decrease training time. Furthermore, the highest momentum constant selection, $a = 0.9$, accelerated training process but it decreased the generalization performance of network over test set patterns which is a similar behaviour that seen for high learning constant utilization in Figure 3.3. As can be seen, momentum constant usage generally accelerates training procedure of network but its effects on test set patterns are in different ways and cannot be generalized easily. Although test set performance of network was increased with $a = 0.1$, and $a = 0.5$ momentum constants compared to $a = 0$ usage, network showed a worse performance for $a = 0.4$, and $a = 0.9$ values than $a = 0$ utilization. This strange behaviour can be related with overfitting, and coincidence because for momentum constant $a = 0.5$ very good regularization performance was obtained, on the other hand, another similar momentum constant which is $a = 0.4$ was applied to verify and determine the reason of this success, unfortunately, network was not be able repeat same performance for $a = 0.4$. For momentum constant $a = 0.5$ change of error values over test and training patterns along training are given in Figure 3.11 and output of trained network can be seen in Figure 3.12.

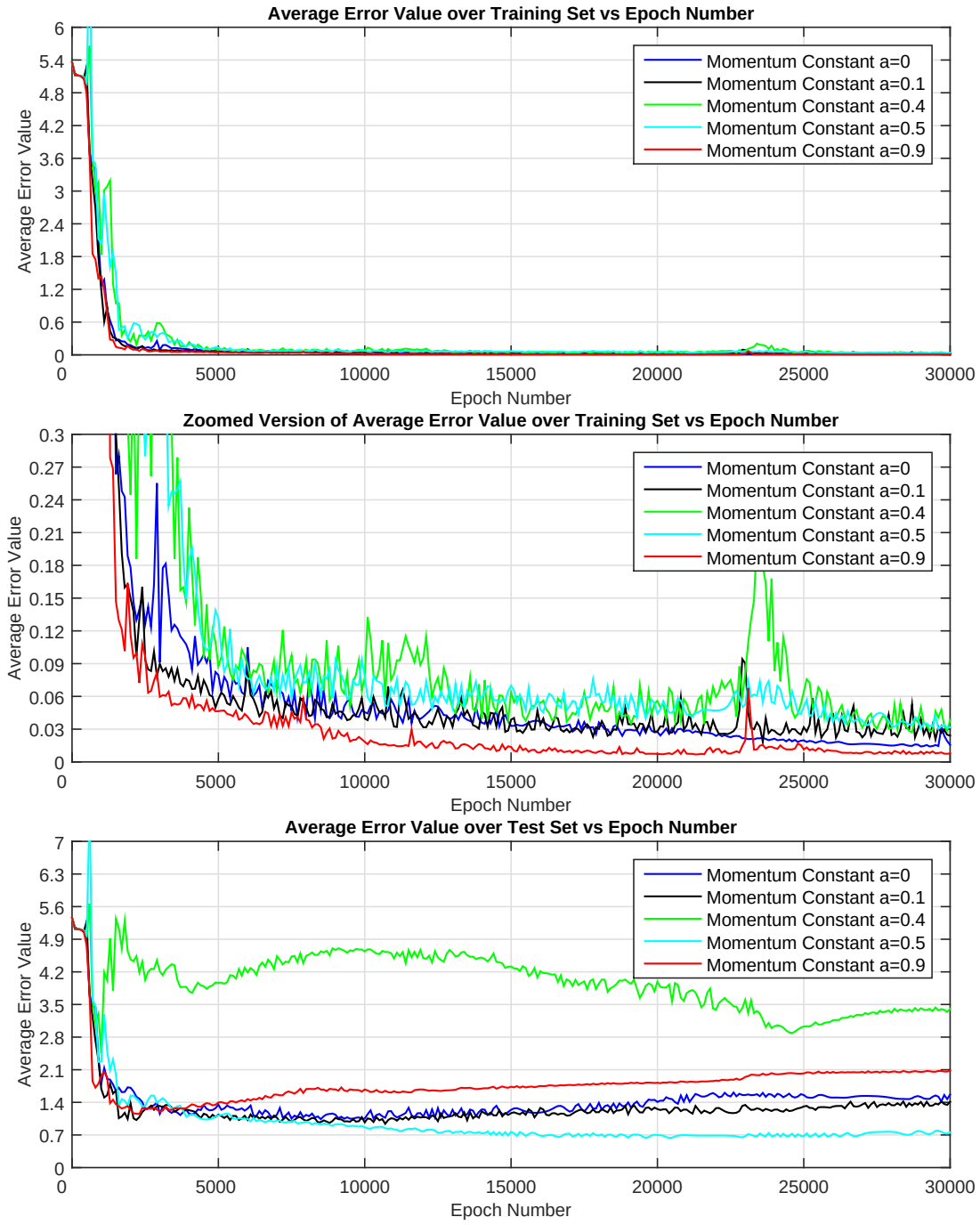

Figure 3.10: Performance comparison for different momentum constant parameters along training.

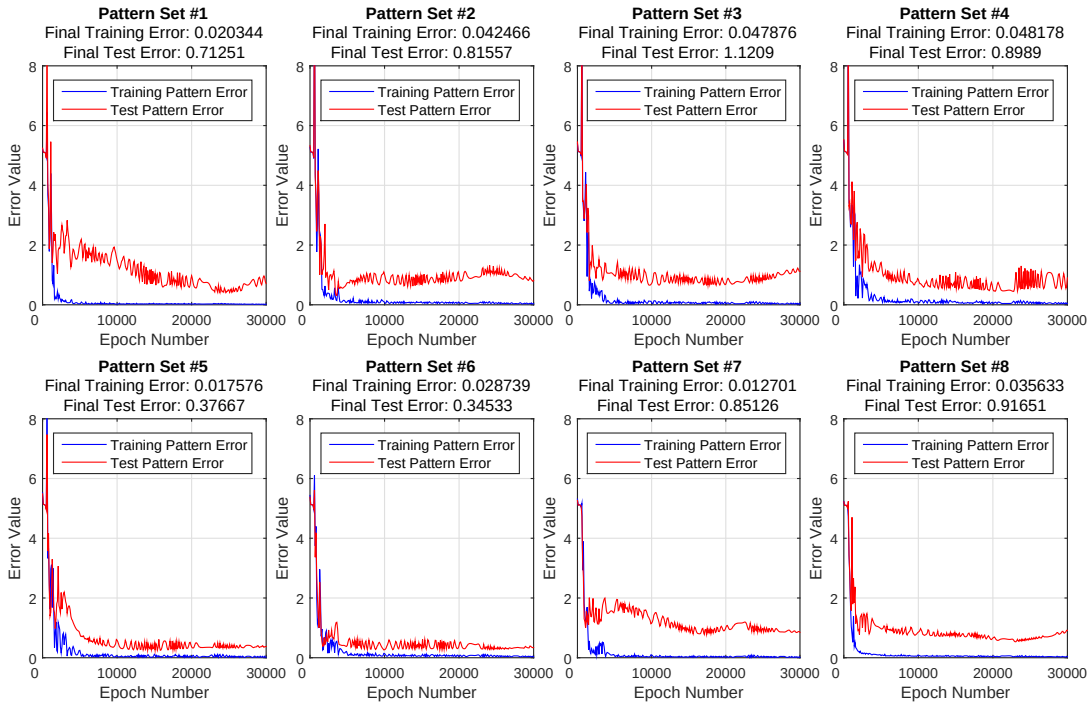

Figure 3.11: Changes on error values of test and training patterns for momentum constant $a = 0.5$ along training.

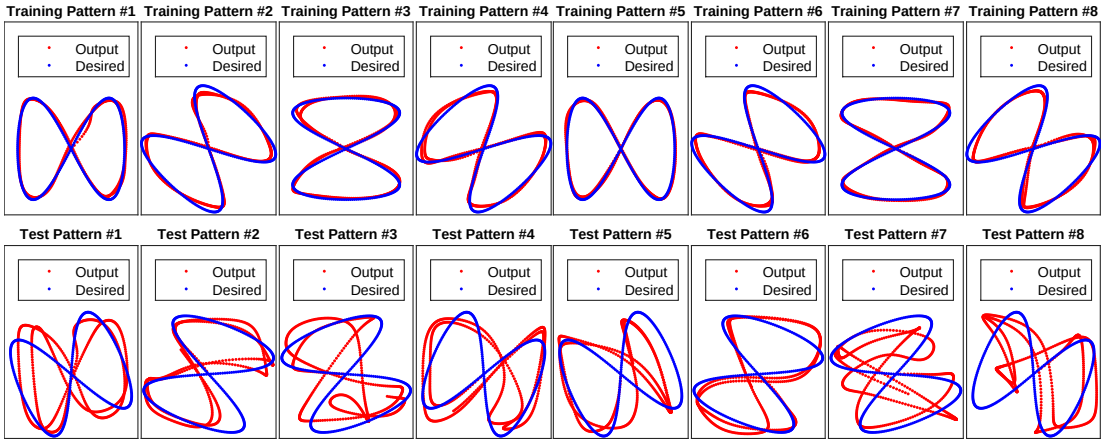

Figure 3.12: Output of trained recurrent neural network for momentum constant $a = 0.5$.

3.5 Cross-Entropy Cost Function

The cross-entropy function utilization is an effective method to increase training speed and performance of feedforward type of neural networks in classification problems, see [32], because of its harmony with nature of classification. In terms of [33], it outperforms mean squared error (MSE) definition and its utilization rate increases day after day in literature. According to [34], its implementation basically requires a different cost function, (3.13), instead of given MSE function definition in (2.16), and its derivative cancels out downside effects of employed activation function related with learning speed decrement, hence performance of training algorithm is enhanced.

$$E = -\sum_{i} \int_{t_0}^{t_1} (d_i(t) \ln y_i(t) + (1 - d_i(t)) \ln(1 - y_i(t))) \mathrm{d}t \tag{3.13}$$

where $d_i(t)$ and $y_i(t)$ are given as desired output vector and values vector of output neurons, respectively.

This cost function modification changes thedefinition of partial derivative with respect to output neuron value, (2.15), to (3.14).

$$e_i(t) = -\frac{d_i(t) - y_i(t)}{y_i(t)(1 - y_i(t))}, \text{ for output neurons, otherwise } e_i(t) = 0 \tag{3.14}$$

In these simulations, we considered the training algorithm given by (2.11)-(2.19), (3.14), (2.20)-(2.23), with a neural network structure has 3 input neurons, 30 hidden neurons and 2 output neurons as shown in Figure 2.1. During training process, inputs of network are generated with (3.2) for $\theta = n\pi/4, n = 0, 1, ..., 7$ values and network is trained with corresponding 8 training patterns which are output of (3.1). In test part, inputs of network are generated with (3.2) for $\theta = n\pi/4 + \pi/8, n = 0, 1, ..., 7$ values and output of network is compared with corresponding 8 test patterns which are output of (3.1). The results of simulations with cross-entropy error definition for different learning constants, $c = 0.01, 0.02, 0.06, 0.1, 0.2, 0.4$ and with MSE definition for learning constant $c = 1$ are given in Figure 3.13 which is generated with error definition in (2.16) instead of (3.14) to be able to compare performance of this algorithm with others.

The effects of different learning rates for a network which uses cross-entropy type of error definition is similar to one that uses (MSE) definition. First, Figure 3.13 demonstrates that small learning rates, $c = 0.01, c = 0.02$ and $c = 0.06$, increases training time of network and shows a bad regularization effects over test set patterns. However large learning rates, $c = 0.2$ and $c = 0.4$, showed better test and training set performance than MSE function utilization with learning rate $c = 1$. They decreased convergence time and reached a better generalization level as shown in 3.13 with learning rate $c = 0.4$. For this configuration, error rate change along training can be evaluated with Figure 3.14 and output of network can be seen in Figure 3.15.

From this point of view, it can be concluded that different error definitions may help to regularize a recurrent neural network and decrease training time at the same time.

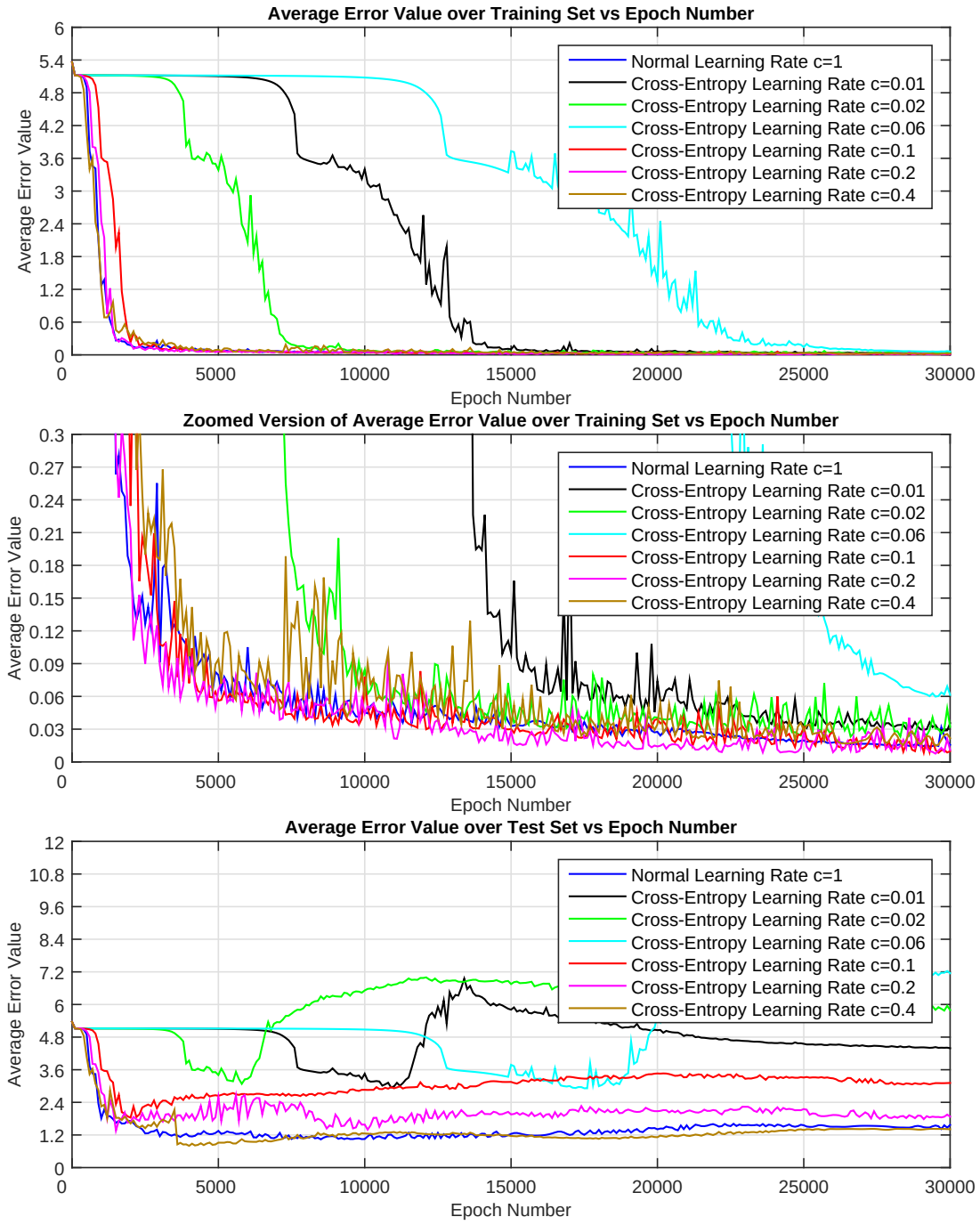

Figure 3.13: Performance comparison for different learning constant parameters along training.

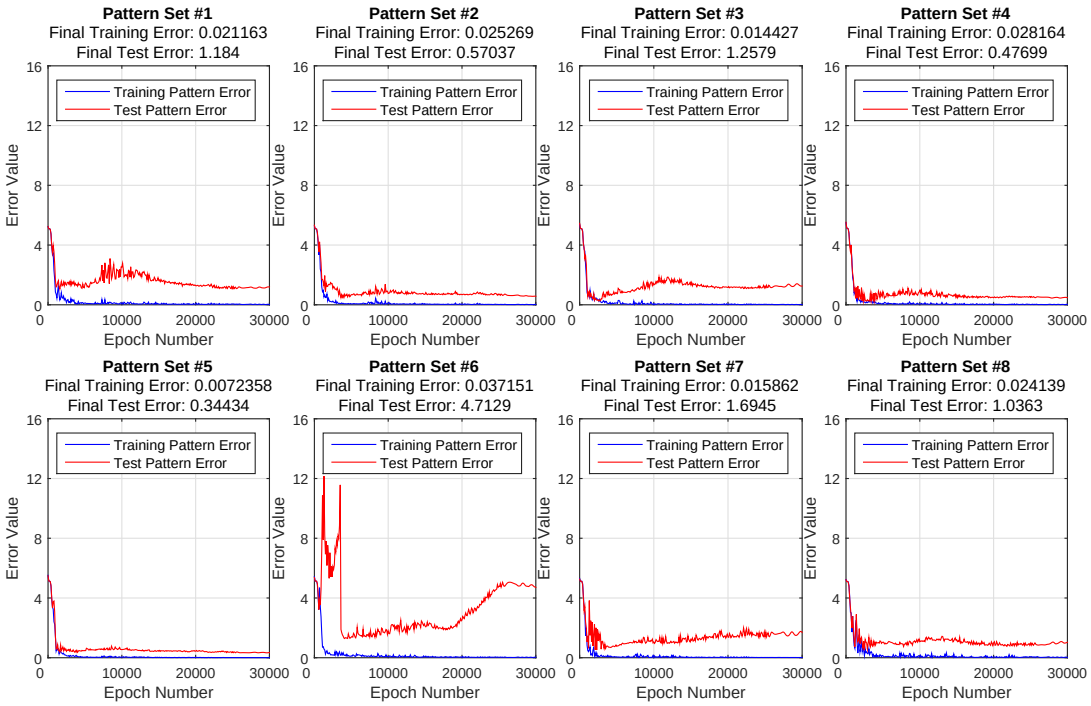

Figure 3.14: Changes on error values of test and training patterns for learning constant $c = 0.4$ parameters along training.

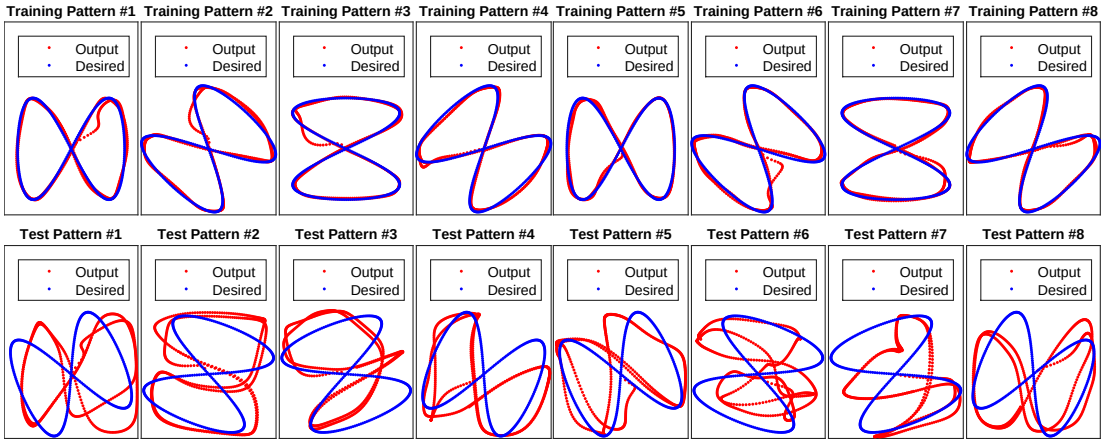

Figure 3.15: Output of trained recurrent neural network for learning constant $c = 0.4$.

3.6 L1 Regularization

In terms of [35], L1 regularization forces absolute sum of connection matrix parameters to be small and it may prevent overfitting of networks weights over training data in feedforward type of neural networks, hence it helps to increase recognition performance of feedforward type of neural networks over test set data. This increment results from modification in cost function as shown in (3.15), and it changes weight update formula (2.22) to (3.16) in terms of [34].

$$E = \frac{1}{2}\sum_{i}\int_{t_0}^{t_1}(y_i(t) - d_i(t))^2 \mathrm{d}t + \frac{\Lambda}{n}\sum_{w}|w| \tag{3.15}$$

$$\text{new } w_{ij} = w_{ij} - c\frac{\partial E}{\partial w_{ij}} - \frac{c\Lambda}{n}sgn(w_{ij}) \tag{3.16}$$

where $d_i(t)$, $y_i(t)$, Λ, c, n and $sgn(w_{ij})$ are given as desired output vector, value vector of output neurons, regularization weight parameter, learning constant, chosen time step number in discretization process and sign of w_{ij}, respectively.

In these simulations, we considered the training algorithm given by (2.11)-(2.21), (3.16) and (2.23), with a neural network structure has 3 input neurons, 30 hidden neurons and 2 output neurons as shown in Figure 2.1. During the training process, inputs of network are generated with (3.2) for $\theta = n\pi/4, n = 0, 1, ..., 7$ values and network is trained with corresponding 8 training patterns which are output of (3.1). In the test part, inputs of network are generated with (3.2) for $\theta = n\pi/4 + \pi/8, n = 0, 1, ..., 7$ values and output of network is compared with corresponding 8 test patterns which are output of (3.1). In order to evaluate performance of L1 regularization, simulations in Figure 3.16 are performed for four different Λ value which are Λ=0.05, 0.005, 0.0005 and Λ=0. The Figure 3.16 is generated with error definition in (2.16) instead of (3.15) to be able to compare performance of this algorithm with others easily.

In terms of simulations in Figure 3.16, high Λ value utilization may prevent convergence such as seen for $\Lambda = 0.05$. In addition to this, L1 regularization did not accelerate training and increase test set performance for $\Lambda = 0.005$ and $\Lambda = 0.0005$ values. It may be concluded that L1 regularization is not beneficial for

acceleration and generalization purposes in RNNs. Although L1 regularization shows a poor performance, error change plot Figure 3.17 and output plot Figure 3.18 of trained network are added to demonstrate effects of L1 regularization in each pattern for Λ=0.0005 value.

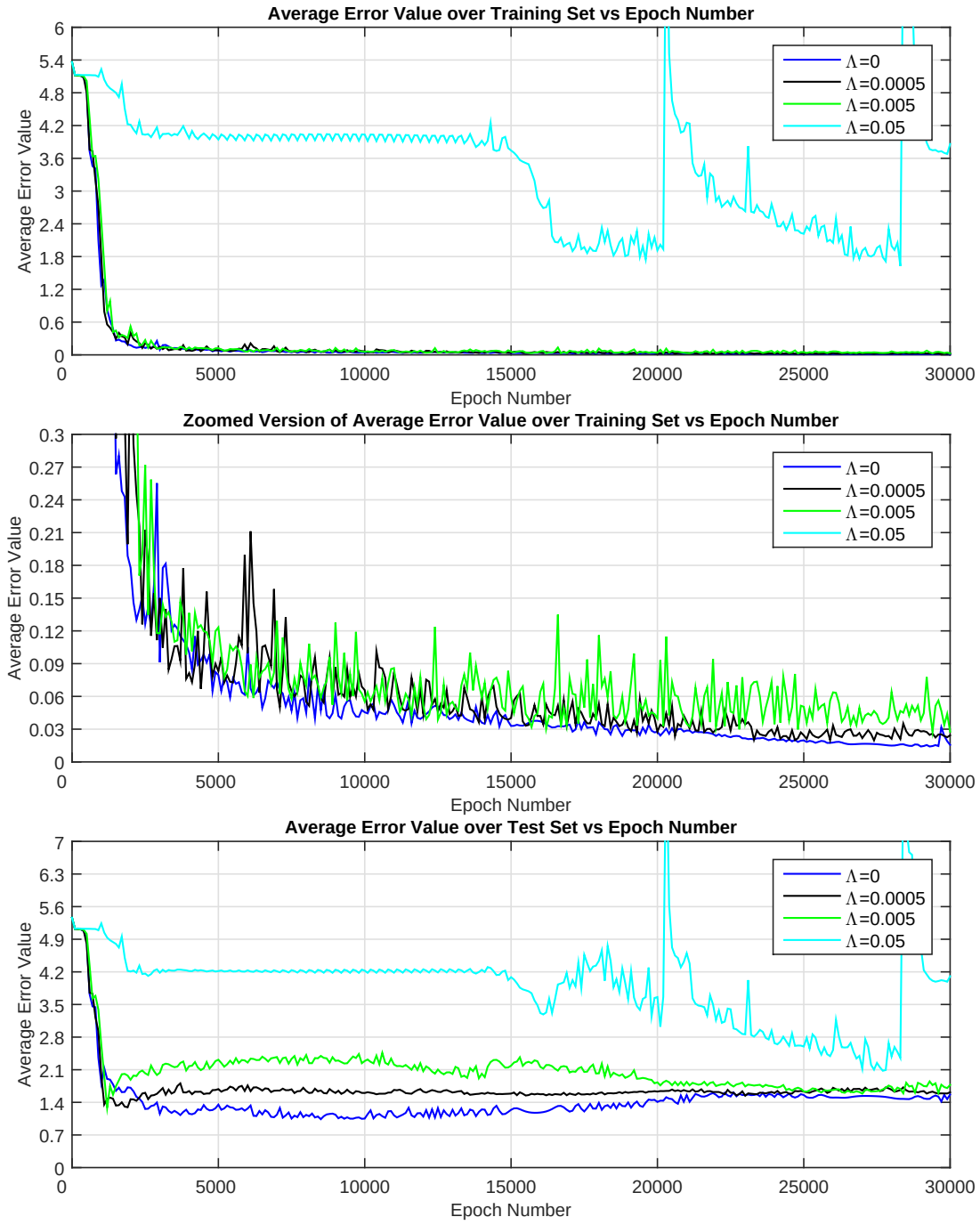

Figure 3.16: Performance comparison for different L1 regularization weight parameters along training.

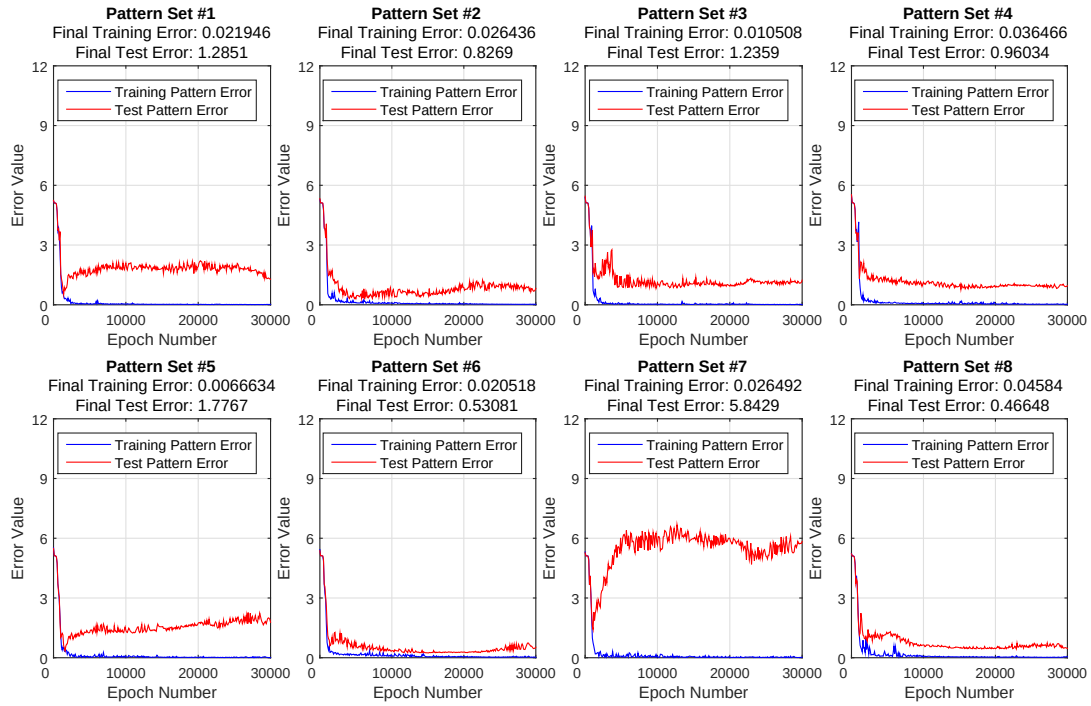

Figure 3.17: Changes on error values of test and training patterns for L1 regularization weight $\Lambda = 0.0005$ along training.

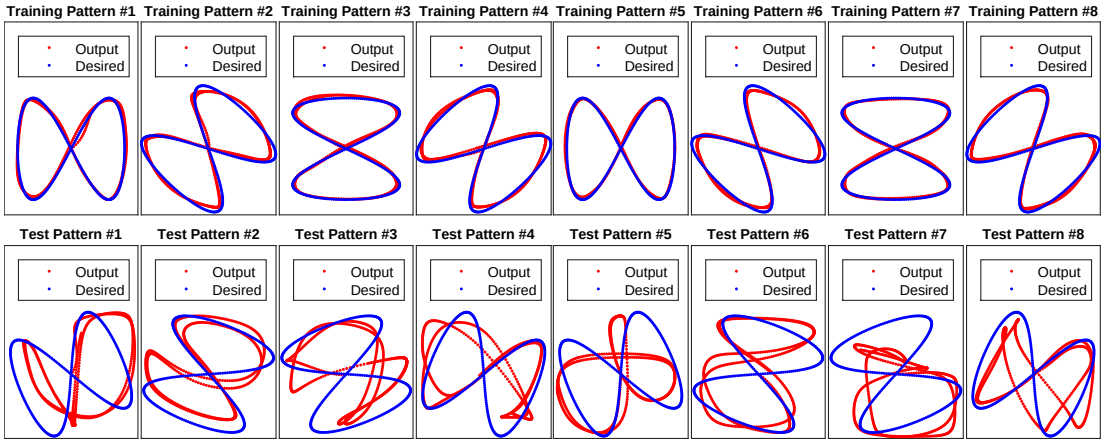

Figure 3.18: Output of trained recurrent neural network for L1 regularization weight $\Lambda = 0.0005$.

3.7 L2 Regularization

L2 regularization function is another well-known method to prevent performance loss over test set recognition ability in later stages of training. In order to succeed this mission, it forces sum of squares of connection matrix parameters to be small, see [35]. To implement the algorithm, MSE function, (2.16), is modified as shown in (3.17) and this modification changes connection matrix weight update formula as in (3.18) in terms of [34].

$$E = \frac{1}{2}\sum_{i}\int_{t_0}^{t_1}(y_i(t) - d_i(t))^2 \mathrm{d}t + \frac{\Lambda}{2n}\sum_{w} w^2 \tag{3.17}$$

$$\text{new } w_{ij} = w_{ij} - c\frac{\partial E}{\partial w_{ij}} - \frac{c\Lambda}{n}w_{ij} \tag{3.18}$$

where $d_i(t)$, $y_i(t)$, Λ, c and n are given as desired output vector, value vector of output neurons, regularization weight parameter, learning constant and chosen time step number in discretization process, respectively.

In these simulations, we considered the training algorithm given by (2.11)-(2.21), (3.16) and (2.23), with a neural network structure has 3 input neurons, 30 hidden neurons and 2 output neurons as shown in Figure 2.1. During the training process, inputs of network are generated with (3.2) for $\theta = n\pi/4, n = 0, 1, ..., 7$ values and network is trained with corresponding 8 training patterns which are output of (3.1). In the test part, inputs of network are generated with (3.2) for $\theta = n\pi/4 + \pi/8, n = 0, 1, ..., 7$ values and output of network is compared with corresponding 8 test patterns which are the output of (3.1). In order to evaluate performance of L2 regularization, simulations in Figure 3.19 are performed for four different Λ values which are $\Lambda = 0.05, 0.005, 0.0005$ and $\Lambda = 0$. The Figure 3.19 is generated with error definition in (2.16) instead of (3.17) to be able to compare performance of this algorithm with others easily.

In terms of these simulations, high Λ values may cause oscillations and L2 regularization does not accelerate training over training set patterns. In addition to these, it does not help network to generate better test results by generalizing

training inputs. For only $\Lambda = 0.005$, network was able increase test set performance so most probably this gain may be result of a coincidence. To be able to examine effects of L2 regularization on each pattern more deeply, error change plot Figure 3.20 and output plot Figure 3.21 of trained network are given for $\Lambda = 0.005$ value.

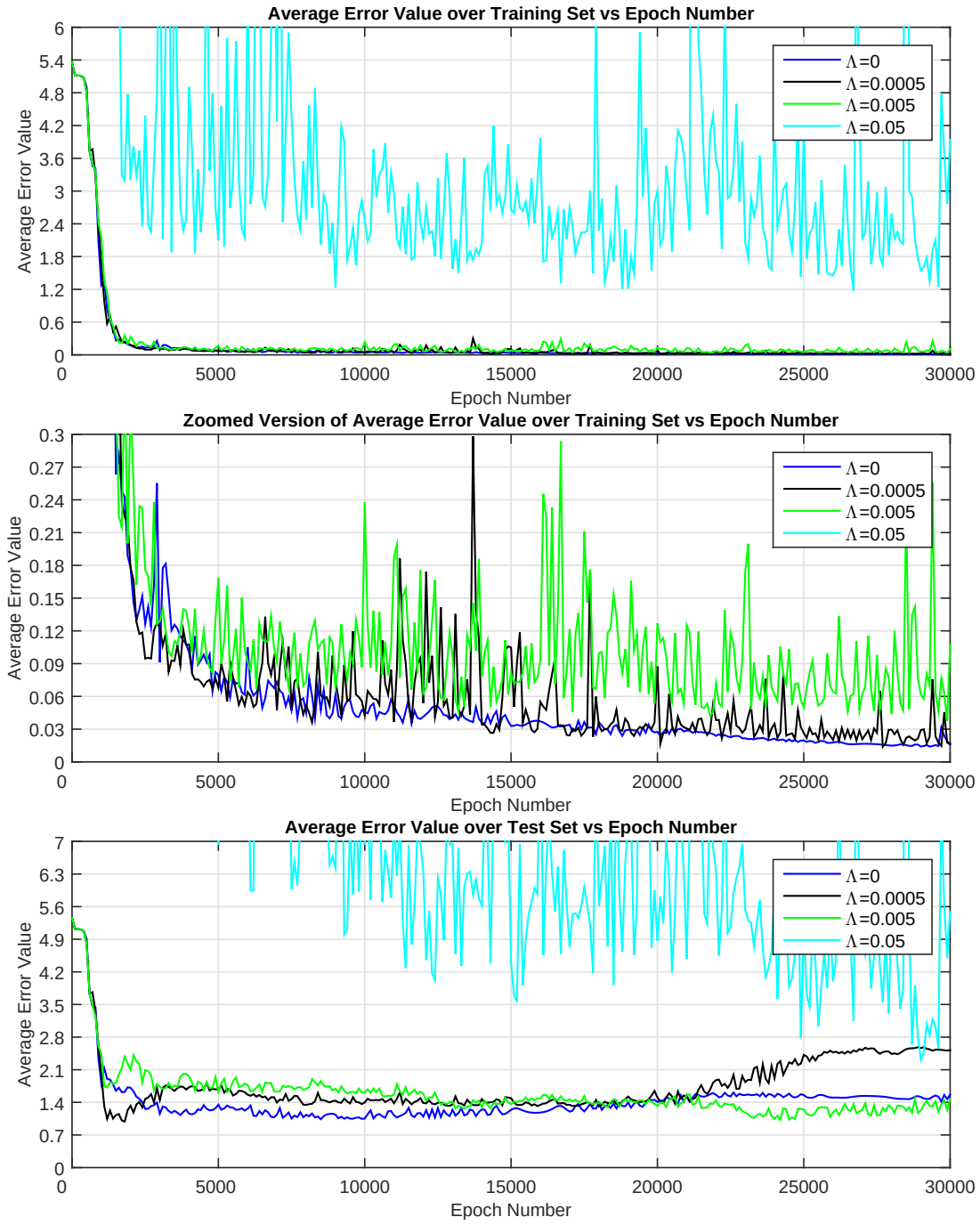

Figure 3.19: Performance comparison for different L2 regularization weight parameters along training.

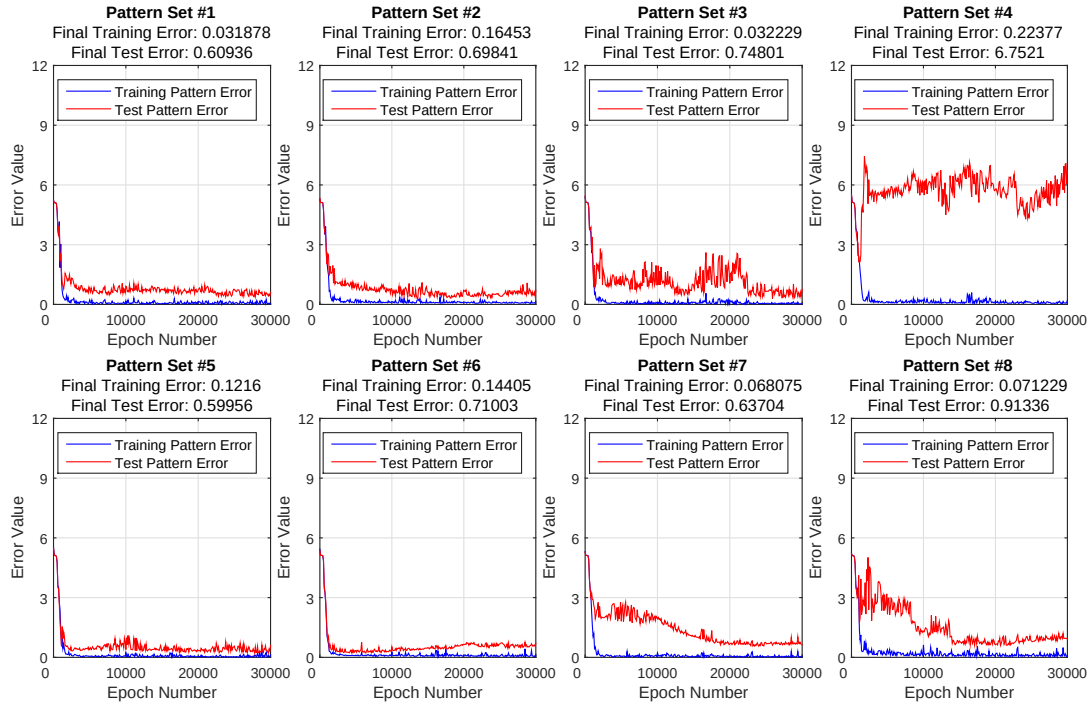

Figure 3.20: Changes on error values of test and training patterns for L2 regularization weight $\Lambda = 0.005$ along training.

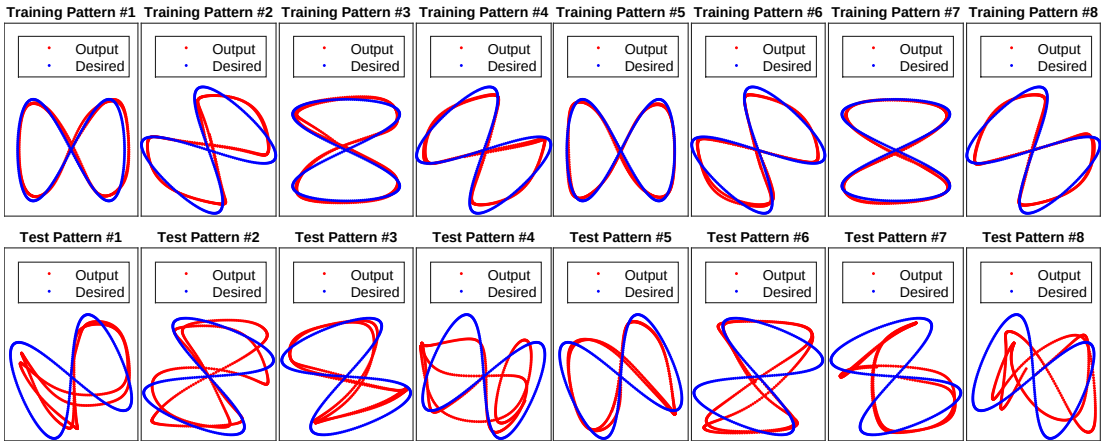

Figure 3.21: Output of trained recurrent neural network for L2 regularization weight $\Lambda = 0.005$.

3.8 Dropout

In terms of [34], dropout is another successful regularization technique for feedforward type of neural networks, and all internal structure of neural network are changed instead of modifying only weight update equation, like in L1 and L2 regularization techniques, in the implementation of this technique. It has different applications like training of multilayer perceptron networks or designing of autoencoder which are used for dimensionality reduction and feature extraction aims. In addition to these, dropout may outperform L1 and L2 regularization techniques to prevent overfitting in recognition tasks, see e.g. [36]. In spite of its widely usage in feedforward networks, there is no work which show its implementation to recurrent neural networks with leaky integrator neuron model. There are some works, see e.g. [1], [37], which use dropout in the feedforward layers of neural networks which have recurrent and feedforward neural layers together. [2] claims that dropout technique is applied to hidden neurons of RNN in order to solve overfitting related recognition difficulties, however, there is no further information about how it is implemented in recurrent layers. Under these condition, we tried to implement dropout technique in various ways in order to evaluate performance of them as seen in Figure 3.22.

In these simulations, we considered the training algorithm given by (2.11)-(2.23), with a neural network structure has 3 input neurons, 30 hidden neurons and 2 output neurons as shown in Figure 2.1. To implement dropout method, drop process is applied to only hidden neurons in RNN in order to protect functionality of the network. During the training process, inputs of network are generated with (3.2) for $\theta = n\pi/4, n = 0, 1, ..., 7$ values and network is trained with corresponding 8 training patterns which are output of the (3.1). In the test part, inputs of network are generated with (3.2) for $\theta = n\pi/4 + \pi/8, n = 0, 1, ..., 7$ values and output of network is compared with the corresponding 8 test patterns which are the output of (3.1).

Dropout method is implemented in following two different ways. In the first way, a drop rate, p, is determined between 0 and 1 and corresponding amount of

hidden neurons are isolated from rest of the network in the first epoch and trained remaining ones along one epoch. In subsequent epoch, we randomly delete same amount of hidden neurons again and trained network and training was continued like this as applied in feedforward networks. By utilizing this method, simulations are performed for $p = 0.033, 0.066, 0.1$ and, 0.5 drop rates and the results are obtained in Figure 3.22. Despite of long training, any convergence cannot be obtained for $p = 0.5$ drop rate as seen in Figure 3.22 with Random Isolation. For $p = 0.1$ drop rate some amount of convergence over training set patterns is obtained until approximately $13000^t h$ epoch, after that network diverged from training set patterns, suddenly. For $p = 0.033$ and $p = 0.066$ convergence over training set was obtained and configuration with $p = 0.066$ gave very good test set performance, on the contrary, configuration with $p = 0.033$ drop rate shows bad test set performance as seen in Figure 3.22. In terms of simulations, random nature of this technique complicates reasoning from simulations but we can conclude that small drop rates generally gave better training and test set results than big drop rates. In the second way, we divided network to two isolated groups, each consist of 15 neurons, and let them to make interconnections between neurons in same group along training, and prevent any connection between these two preselected neuron groups. Under these conditions, training is performed by alternating training neuron group and isolating other group from rest of the network in each epoch. Thus, in total two neurons group are trained and these are given in Figure 3.22 by Upper Half Part, p=0.5 and Lower Half Part, p=0.5 labels. In addition to these, a third network is generated by combining these two neuron groups and is added to Figure 3.22 with Combination of Parts, p=0.5 label. It is interesting to see that combination of these two small networks gave better training and test set performance than each one of the small networks. The high generalization performance of combined network shows that combination of separately trained networks may reach better generalization levels than trained network as a whole has same neuron number in total.

Although some dropout implementations reach better test set performance with these methods, training set performance and reliability of these techniques are low. The low performance of this technique may be related with function of

hidden neurons which behaves like a finite state machine and stores current state of network when some of them are randomly deleted from network, performance losses occurs inevitably. Utilization of second technique satisfies better training set results than techniques which include random isolation method. Moreover it shows a better test performance than base performance level normal learning constant $c = 1$ configuration in Figure 3.22. To evaluate performance of it, its error change patterns along training process and output patterns are given in Figure 3.23 and Figure 3.24.

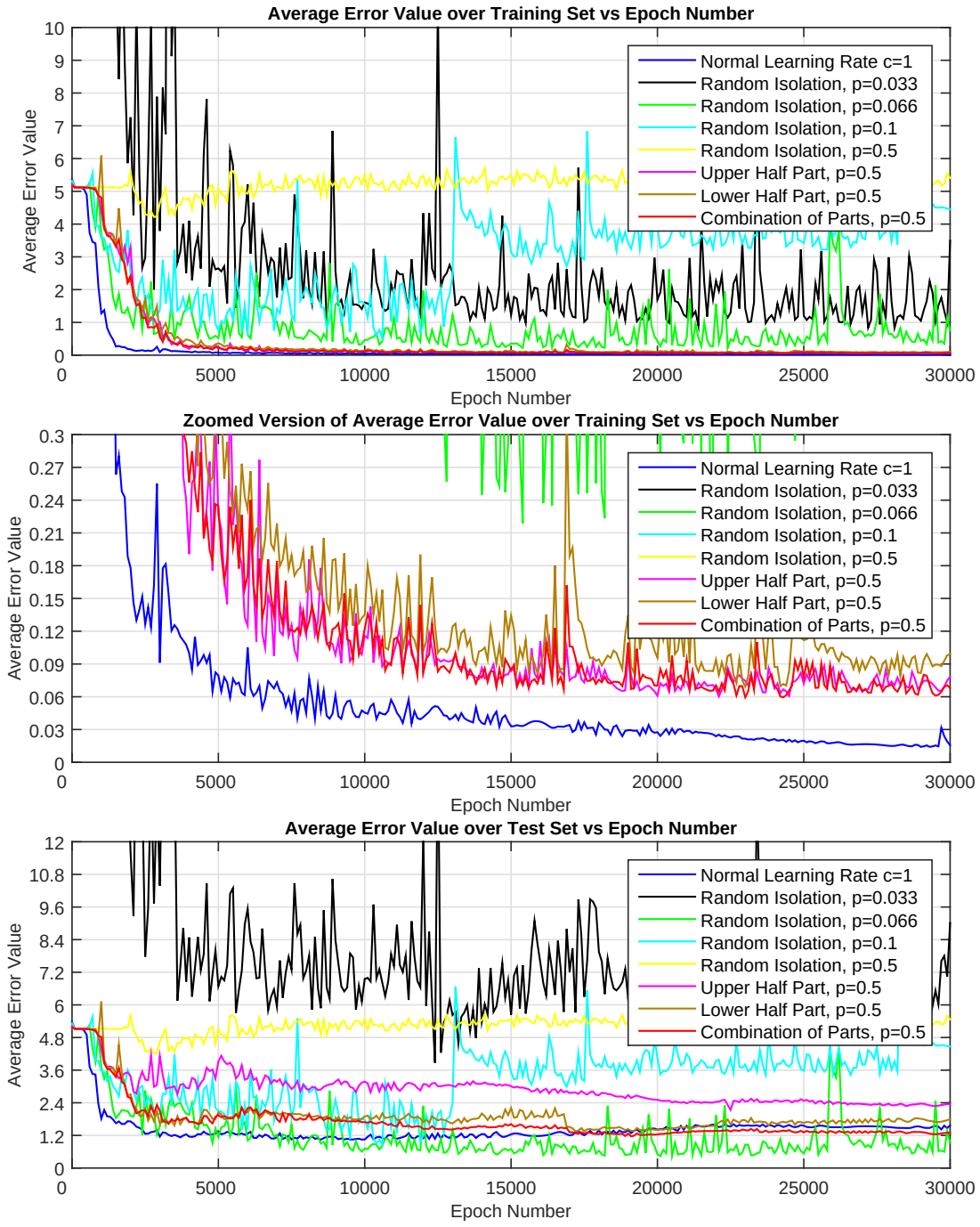

Figure 3.22: Performance comparison for different drop rates along training.

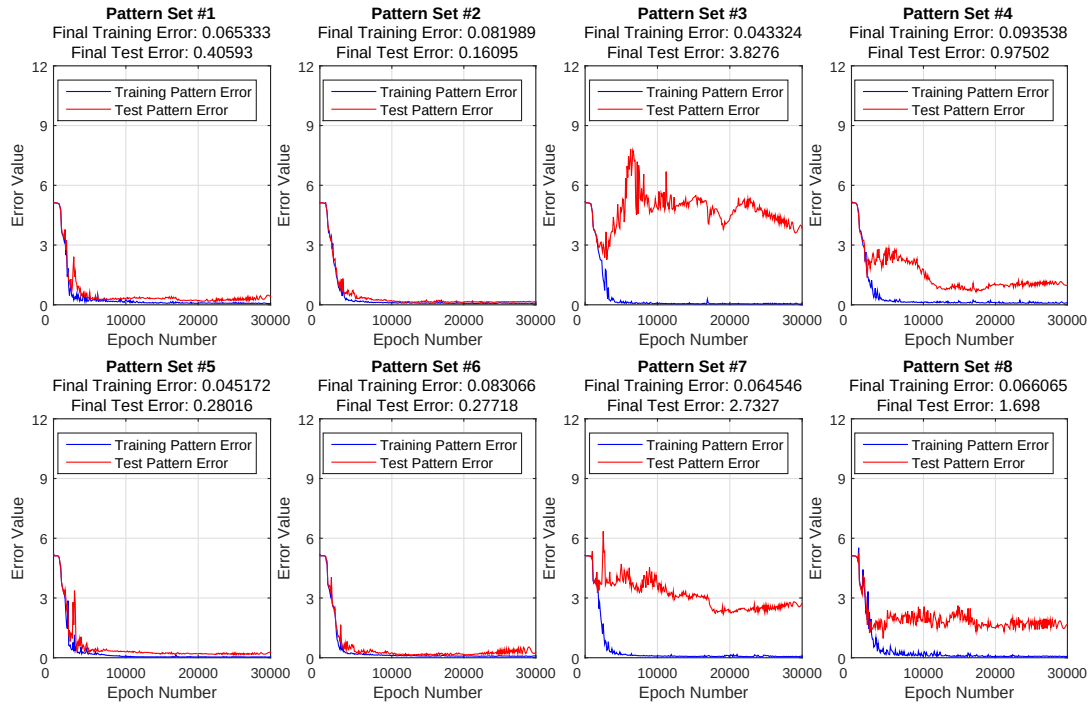

Figure 3.23: Changes on error values of test and training patterns for combination of two networks configuration along training.

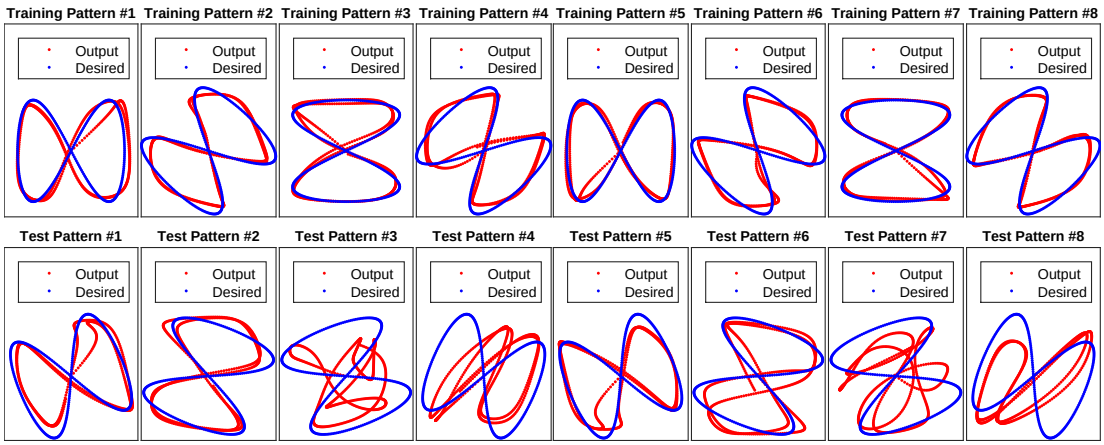

Figure 3.24: Output of trained recurrent neural network for combination of two networks configuration.

3.9 Time Constant Limitation

In training process, connection matrix weights and time constant parameters are updated to teach desired behaviour to network but sometimes this update stage may generate some problems, especially time constant updates may be problematic because if one of the time constant parameter is close to zero, applied parameter update may push it into the negative values as a result of it, neuron with negative time constant generates unbounded outputs in following forward propagation part of subsequent learning stage. This unbounded output increases error gradient and pushes all other weights of network to unreasonable values. In order to overcome this problem, a positive time constant value must be determined as lower bound, see e.g. [31]. In this thesis, simulations in Section 3.3, Section 3.6, Section 3.7, and Section 3.8 are performed with time constant limitation with 0.05 lower bound to enhance stability of training procedure.

3.10 Analysis and Discussion

In terms of simulations in this chapter, momentum and cross-entropy cost function is beneficial techniques to accelerate teaching process of single pattern or a set of patterns to RNNs. In addition to these, a single pattern need to be taught to RNN, delta-bar-delta learning rule is another effective acceleration technique but it works only in training of single pattern well. From this point of view, most of the acceleration techniques of feedforward type of neural networks work at RNNs either fully or partially. On the other hand, regularization techniques do not seem to be beneficial for generalizing input values to desired output patterns. This failure may be related with nature of regularization techniques which are developed to increase recognition performance of feedforward type of neural networks but in our case we try to produce patters in terms of given input sets so there is no need to apply any feature reduction technique in our problem, although it is known that some of regularization techniques works in a similar way with feature reduction notion. It clearly needs to be stated that although there

are some works which proposes implementation of dropout on feedforward layers of RNNs, in our network there was no feedforward layer neuron except input neurons so different possible utilization ways of dropout were tried and results was obtained. In terms of simulations, random neuron isolation based dropout use is not feasible with high drop rates. Combination of separately trained of recurrent neural networks may be a better choice for training and test set performances than random isolation technique but any type of application of dropout generally needs further investigation to reach a conclusion in recurrent layers. Finally, time constant limitation is a helpful precaution for training of RNN which has leaky integrator model neurons. It increases stability of learning and prevents unbounded oscillation through learning process so that it accelerates the training indirectly.

Chapter 4

Application to Biped Robot Model Locomotion

Biped robotic platforms promise high performance in realizing difficult maneuver but control of such platforms are quite difficult due to highly nonlinear structure of their equations of motion. For controlling such biped robotic platforms researchers benefit from various algorithm such as center of gravity and zero moment point type robot posture calculation including techniques, see e.g. [7], [8], but the due to their dependence on inverse kinematic equations, they cause high computation loads. Alternatively, there are different central pattern generator (CPG) type of controllers which require less computation than inverse kinematic depended solutions. These controllers are enough to satisfy stable locomotion in some cases, see e.g. [38], [14], but they may require much more complex design stages. The CPGs proposed in [38] and [14] are basically recurrent neural networks (RNN) with limited amount of free parameters. Even though free parameter limitation of CPG type of controller eases design and training phases, it shrinks neural network output space substantially.

In this chapter, locomotion generation for a biped robot model has been examined with benefitting from different RNNs which are utilized as locomotion

pattern generator. In the design phase, it is allowed to establish connections between all neurons except the incoming connection weights and time constants of input neurons as show in Figure 4.3 in order to get rid of output limitations of controllers based in CPGs.

4.1 Biped Robot Platform

Biped robot model, which can be seen in Figure 4.1, is taken from [16] and it is driven with a CPG controller in original work; hence adaptation ability of robot to variable environment conditions are examined under CPG controller. The platform consists of five parts and includes hip, knee and ankle joints at legs. Motion occurs according to generated torque at the each joint. To examine success of controller robot model needs to be in interaction with physical world so environmental effects are delivered to robot platform with reaction forces applied to ankle joints. Model of lower leg includes Spring Loaded Inverted Pendulum model so that let us perform simulations under variable ground friction and slope conditions. In [16], equations of motion, which are given in Appendix A in detail, are modelled by benefitting from Newton-Euler method which is an efficient calculation technique especially for real time implementation.

To demonstrate correlation between torques and second derivative of limb angles (4.2) is given below. In these equations of motion, F_i, $i = 1, 2, 3, 4$ generic functions corresponds to collection of all linear and nonlinear terms in original equations of motion except torque terms. Moreover, there are I_1 and I_2 variables which are inertias of upper and lower legs parts, respectively and they are given in (4.1). Directions of torques are shown in Figure 4.2.

$$\begin{aligned} I_1 &= \frac{m_1 l_1^2}{12} \\ I_2 &= \frac{m_2 l_2^2}{12} \end{aligned} \tag{4.1}$$

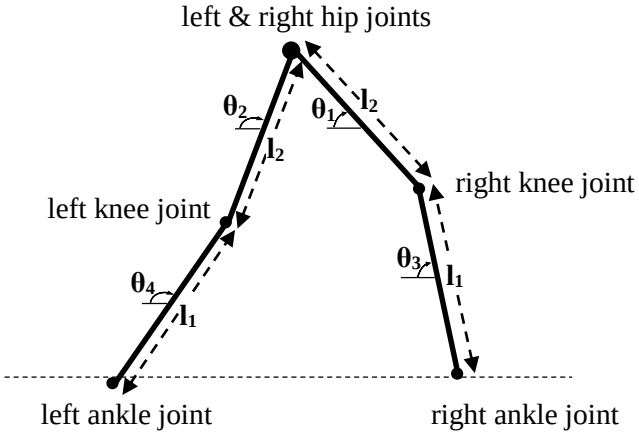

Figure 4.1: Biped robot model limb angles.

$$
\begin{aligned}
I_1\ddot{\theta}_1 &= F_1(\theta_1,\theta_2,\theta_3,\theta_4,\dot{\theta}_1,\dot{\theta}_2,\dot{\theta}_3,\dot{\theta}_4) + T_{r1} + T_{r3} \\
I_1\ddot{\theta}_2 &= F_2(\theta_1,\theta_2,\theta_3,\theta_4,\dot{\theta}_1,\dot{\theta}_2,\dot{\theta}_3,\dot{\theta}_4) + T_{r2} + T_{r4} \\
I_2\ddot{\theta}_3 &= F_3(\theta_1,\theta_2,\theta_3,\theta_4,\dot{\theta}_1,\dot{\theta}_2,\dot{\theta}_3,\dot{\theta}_4) - T_{r3} - T_{r5} \\
I_2\ddot{\theta}_4 &= F_4(\theta_1,\theta_2,\theta_3,\theta_4,\dot{\theta}_1,\dot{\theta}_2,\dot{\theta}_3,\dot{\theta}_4) - T_{r4} - T_{r6}
\end{aligned}
\tag{4.2}
$$

In original work, neural circuitry of robot platform consists of 6 Matsuoka Oscillators, see [12] and [13], and each oscillator has two mutually connected leaky integrator neurons, see [29] and [26], with rectifier type of activation function, see [39]. Neural network involves 6 primary and 6 supplementary neurons, in total 12 neurons.

At the first stage of our study, we focus on decreasing simulation time and selecting the best numerical integration algorithm for our robot model. In terms of our trials, first order Euler's Method and Fourth Order Runge-Kutta numerical integration method are chosen as the best configuration to calculate equations of motion of robot body and CPG outputs at 10 KHz sampling frequency, respectively.

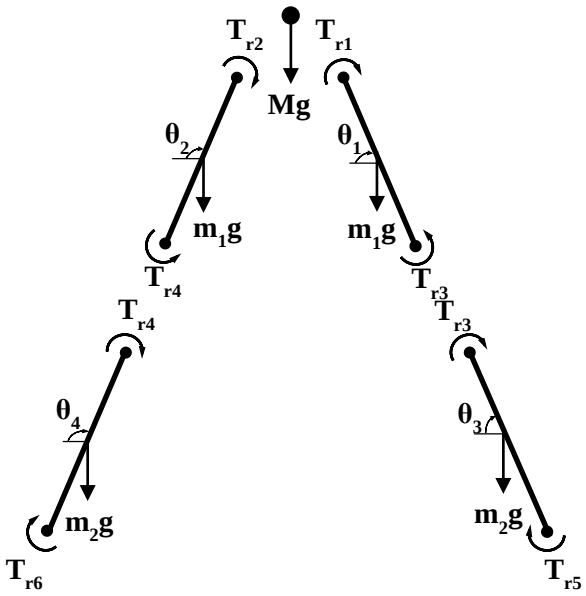

Figure 4.2: Biped robot model torque outputs.

4.2 Recurrent Neural Network Controller Configuration

For controlling such biped robotic platforms an interesting technique would be the utilization of RNNs. In this section, employed neural network configuration in simulations will be explained in detail. There are a lot of different ways of training a neural network, see e.g. [28], [14] and [15], but in the scope of this thesis, we will consider supervised training algorithms, such as back-propagation through time (BPTT) algorithm. In order to generate a training data set, we benefitted from existing neural rhythm generator of biped robot template given in [16]. By running simulation for different input variables (see (4.3)), we recorded the lower and upper leg angles of robot. After this, we need to apply decimation process with 100 to 1 ratio in order to prevent any aliasing distortion on training and test data sets. First recorded data was filtered with 401^{st} order low pass finite impulse response (FIR) filter and then downsampling was performed with 100 to 1 ratio to apply decimation procedure. Hence, sampling frequency of training data was decreased from 10 KHz to 100 Hz without any loss of important data, and training time of network is also increased with this decimation operation.

For simulations in Section 4.4, we considered the training algorithm given by (2.11)-(2.23), with a neural network structure having 2 input neurons, 10 hidden neurons and 4 output neurons as shown in Figure 4.3. In addition to these, lower bound of time constants are determined as 0.05 in order to enhance training process. During training process, inputs of network are generated with (4.3) for $u_0 = 4, 4.3, 4.7, 5, 5.3, 5.7, 6, 6.3$ values and network is trained with corresponding 8 training patterns and one of them are given in Figure 4.4. In test part, inputs of network are generated with (4.3) for $u_0 = 4.15, 4.5, 4.85, 5.15, 5.5, 5.85, 6.15, 6.45$ values and output of network, (4.4), is compared with corresponding 8 test patterns and one of them are given in Figure 4.5. During the simulations, performance of three different activation functions, four different learning constants and maximum error gradient with respect to neuron output limitation are investigated and compared with others in Section 4.3.

As indicated before, for the training of the desired biped leg angles $\theta_i,\ i = 1, 2, 3, 4$ which are shown in Figure 4.1, we consider the recurrent neural network structure as shown in Figure 4.3. In this structure, we have two input neurons, whose inputs are defined as

$$I^{ext}(t) = \begin{bmatrix} u(t) \\ v(t) \end{bmatrix} = \begin{bmatrix} u_0 \\ 0.5 \end{bmatrix} \tag{4.3}$$

Here u_0 will be chosen as a constant value which will indicate either selected training or test set patterns and 0.5 is bias term. Moreover, we have 4 output neurons which correspond to biped limb angle, i.e. if $y_o(t)$ is the set of output neurons, we have:

$$y_o(t) = \begin{bmatrix} \theta_1(t) \\ \theta_2(t) \\ \theta_3(t) \\ \theta_4(t) \end{bmatrix} \tag{4.4}$$

In our simulations we have chosen 10 hidden layer neurons. This number is chosen by trial-and-error and it seems that it is sufficiently low to yield reasonable training times with acceptable training set errors.

We considered scaled sigmoidal, linear and rectifier as activation functions. Definition of scaled sigmoidal activation function is given in (4.5).

$$\sigma(x_i(t)) = \frac{3}{1 + e^{-x_i(t)}} \tag{4.5}$$

Linear activation function with gain k is given below:

$$\sigma(x_i(t)) = kx_i(t) \tag{4.6}$$

where we have chosen $k = 1$ in our simulations for simplicity.

The rectifier activation function and its derivative are given in (4.7) and (4.9), respectively. We note that the rectifier activation function utilization is computationally more efficient and may show better performance in certain cases than

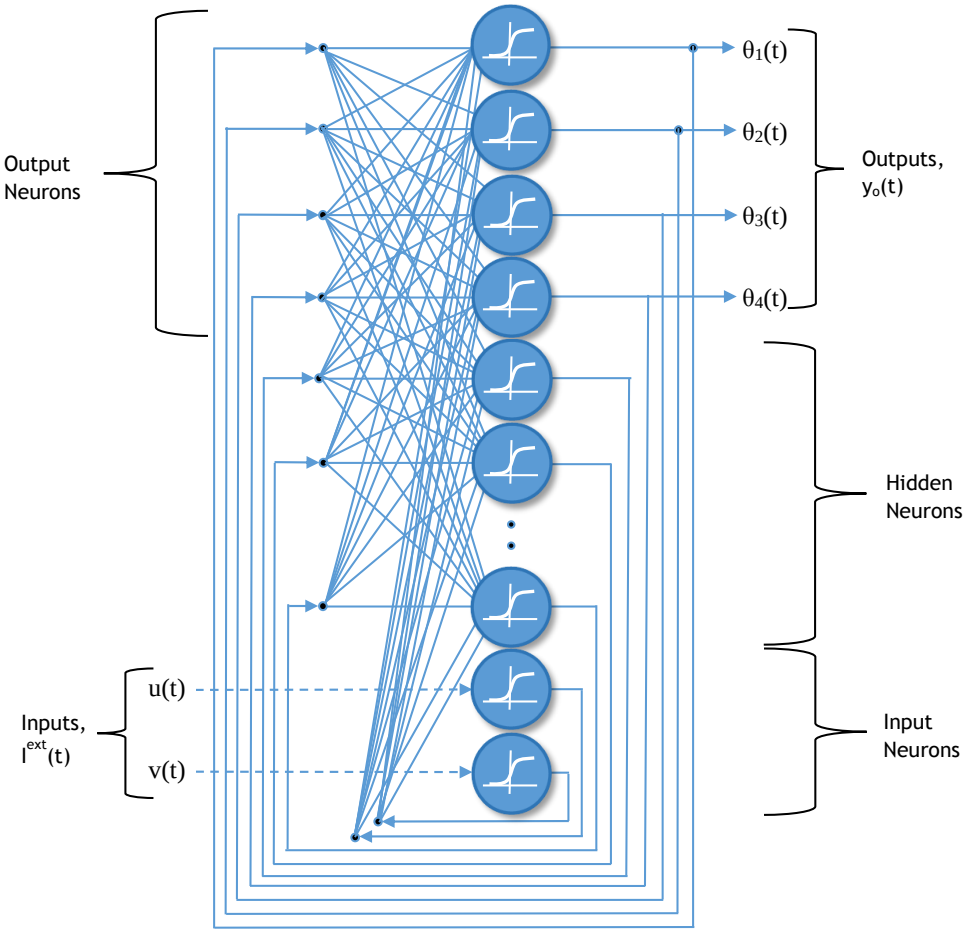

Figure 4.3: Recurrent neural network configuration for the biped robot model.

its counterparts, see e.g. [40], [41].

$$\sigma(x_i(t)) = \begin{cases} x_i(t) & \text{if } x_i(t) > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4.7}$$

$$\sigma'(x_i(t)) = \begin{cases} 1 & \text{if } x_i(t) > 0 \\ 0 & \text{if } x_i(t) < 0 \\ indefinite & \text{otherwise.} \end{cases} \tag{4.8}$$

4.3 Error Gradient Normalization

In addition to existing difficulties of recurrent neural networks training with gradient descent method, see [42], there are gradient exploding and gradient vanishing type problems in training of recurrent neural networks, see [6], [5]. The gradient vanishing and exploding are related with layered structure of network in time and training algorithm can encounter with this type of problems in error gradient propagation between consecutive time layers, see [5]. Gradient vanishing slows down learning process and we did not encounter with a this type of problem in our works. Gradient exploding is a much more important problem for our learning process because it may push parameters of whole network to irreversible place with only one update step and fail all learning process. In order to solve this problem, we benefitted from the proposed method in [5] which is called scaling down the gradient and it is given below.

$$\frac{\partial E}{\partial y_i} = \begin{cases} \frac{\text{threshold}}{||\frac{\partial E}{\partial y_i}||} \frac{\partial E}{\partial y_i} & \text{if } ||\frac{\partial E}{\partial y_i}|| > \text{threshold} \\ \frac{\partial E}{\partial y_i} & \text{if } ||\frac{\partial E}{\partial y_i}|| \leqslant \text{threshold} \end{cases} \tag{4.9}$$

4.4 Training Results

In this section simulations are carried out in two stages. In the first stage, simulations of 24 different neural network parameter configuration are performed along one million epoch and their results are given in Table 4.1, Table 4.2 and Table 4.3. The simulations which need to be terminated because of instability or divergence are denoted with '*' symbol next to them in the tables and the lowest error value of these ended simulations are given in the tables. To evaluate its effects and enhance the stability of training, simulations of each of learning constant and activation function combinations are repeated for $threshold = 10$ limit with error gradient normalization algorithm which is given in Section 4.3.

Under these conditions sigmoid activation function gives the worst results because all simulations were diverged from training patterns and training was ended

before one million epoch. Linear activation function has a stable training period and its all configurations converged to training patterns by different amounts. It is easy to see that error gradient limitation decreased the speed of training for all configurations with linear activation function because when it normalizes a gradient, learning speed of network decreases such as working with a lower learning constant. Rectifier activation function shows interesting and important behaviours during training. First of all it gave the lowest error values among all activation function simulations and interestingly these configurations are diverged after reaching these results. In 4.3 appears that error gradient normalization is useful to prevent instabilities during training. Even though normalization decreases learning speed, one of the configuration of rectifier activation function with error gradient normalization achieves a lower error value than all configurations of other activation functions without losing stability.

Table 4.1: Performance of sigmoid function utilization in neuron model for one million epoch training.

	Error Value							
	c=0.001		c=0.01		c=0.1		c=1	
	Training	Test	Training	Test	Training	Test	Training	Test
$\|\|z\|\| < \infty$	1.219*	1.282*	1.223*	1.287*	1.241*	1.305*	1.628*	1.695*
$\|\|z\|\| < 10$	1.218*	1.282*	1.223*	1.287*	1.248*	1.310*	2.213*	2.275*

Table 4.2: Performance of neural model with linear activation for one million epoch training.

	Error Value							
	c=0.001		c=0.01		c=0.1		c=1	
	Training	Test	Training	Test	Training	Test	Training	Test
$\|\|z\|\| < \infty$	1.245	1.306	0.442	0.480	0.396	0.427	0.398	0.438
$\|\|z\|\| < 10$	1.245	1.307	0.574	0.653	0.444	0.521	0.403	0.466

In the second stage, trained networks with rectifier and linear activation functions are trained for one million epoch more and the results are given in Table 4.4 and Table 4.5. Sigmoid activation function utilized networks are not trained further because their training process need to be ended before completing the

Table 4.3: Performance of rectifier function utilization in neuron model for one million epoch training.

	Error Value							
	c=0.001		c=0.01		c=0.1		c=1	
	Training	Test	Training	Test	Training	Test	Training	Test
$\|\|z\|\| < \infty$	1.243	1.304	0.256*	0.287*	0.326*	0.392*	0.606*	0.641*
$\|\|z\|\| < 10$	1.243	1.304	0.376	0.480	0.935	1.083	1.099	1.164

first one million epoch length training. In this second training phase, generally error rates are decreases but do not change too much as seen in the first phase of training. This slow convergence is most probably related with plain error surface in which networks continue to learn patterns in the direction of error gradient.

Table 4.4: Performance of neural model with linear activation for two million epoch training.

	Error Value							
	c=0.001		c=0.01		c=0.1		c=1	
	Training	Test	Training	Test	Training	Test	Training	Test
$\|\|z\|\| < \infty$	1.220	1.280	0.435	0.476	0.375	0.415	0.390	0.431
$\|\|z\|\| < 10$	1.220	1.281	0.496	0.587	0.436	0.523	0.405	0.474

Table 4.5: Performance of rectifier function utilization in neuron model for two million epoch training.

	Error Value							
	c=0.001		c=0.01		c=0.1		c=1	
	Training	Test	Training	Test	Training	Test	Training	Test
$\|\|z\|\| < \infty$	1.212	1.273	0.256*	0.287*	0.326*	0.392*	0.606*	0.641*
$\|\|z\|\| < 10$	1.213	1.273	0.349	0.387	0.988	1.029	1.659	1.914

Among these simulations the lowest error value is achieved with rectifier activation function and learning constant $c = 0.01$ configuration. Comparisons of outputs of this network configuration with one of training and one of test patterns are given in Figure 4.4 and Figure 4.5, respectively.

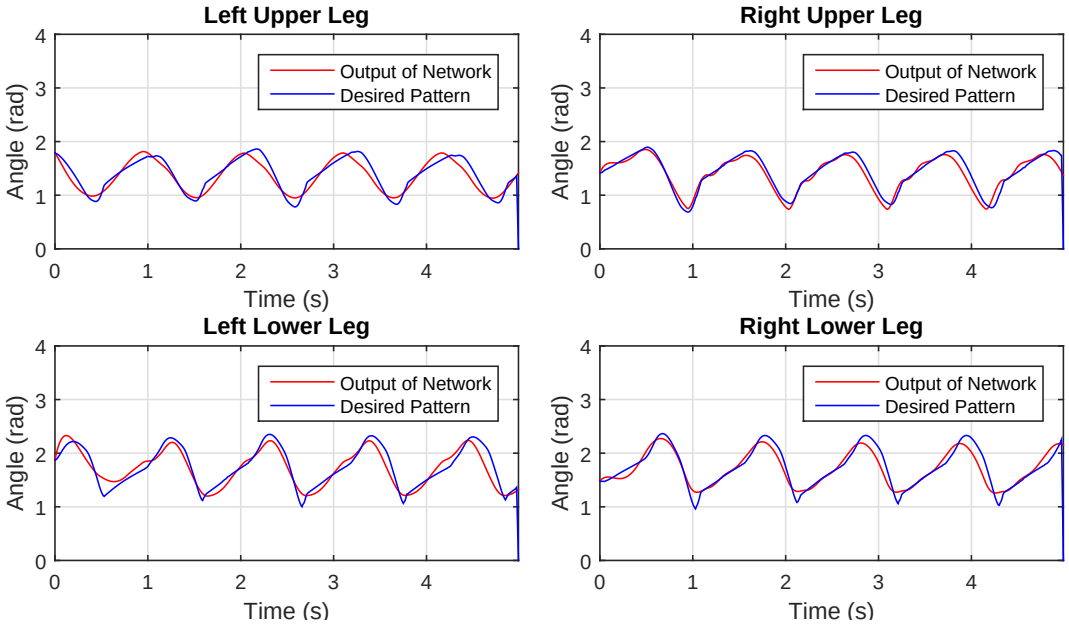

Figure 4.4: Comparison of desired pattern and output of network for $u_0 = 5$ training input.

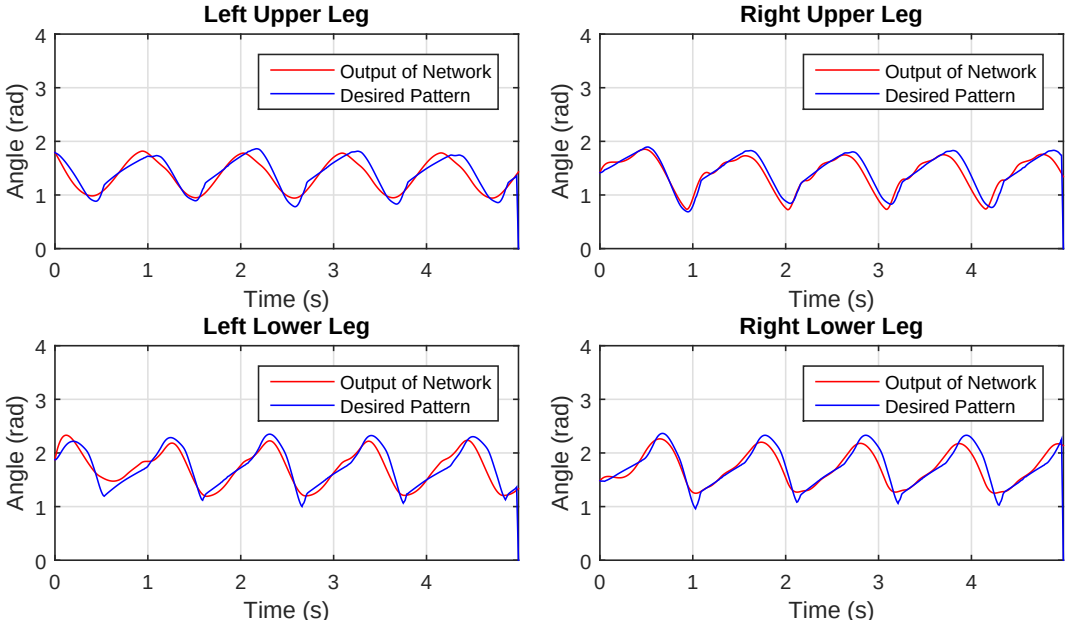

Figure 4.5: Comparison of desired pattern and output of network for $u_0 = 5.15$ test input.

4.5 Riped Robot Control Simulations

In the training of RNNs, recorded limb angles of a walking biped robot are utilized as desired patterns and network outputs have converged them with various success rates in Section 4.4. After that we have a RNN which needs to be able to reproduce limb angles and sustain stable locomotion for same biped robot model at different speed levels although robot platform has lots of nonlinearity in its equations of motion, see Appendix A. To test this hypothesis, we need to drive biped platform by output of trained RNN which produces as limb angles but equations of dynamics of biped controlled via torques, for this reason we used classical PD controllers, which are successful in control of complex dynamic systems, to convert angle outputs to required torques. The implementation of PD controllers are given in (4.10) and each PD controller has identical K_p and K_d parameters and they are chosen by trial-and-error as 7000 and 700, respectively. It seems that there is a large set of K_p and K_d parameters which are capable of sustaining stable locomotion.

$$\begin{aligned}
T_{r1} =& I_1(K_p(\theta_1^{RNN} - \theta_1) + K_d(\dot{\theta}_1^{RNN} - \dot{\theta}_1)) \\
T_{r2} =& I_1(K_p(\theta_2^{RNN} - \theta_2) + K_d(\dot{\theta}_2^{RNN} - \dot{\theta}_2)) \\
T_{r3} =& I_1(K_p(\theta_1^{RNN} - \theta_1) + K_d(\dot{\theta}_1^{RNN} - \dot{\theta}_1)) \\
& - I_2(K_p(\theta_3^{RNN} - \theta_3) + K_d(\dot{\theta}_3^{RNN} - \dot{\theta}_3)) \\
T_{r4} =& I_1(K_p(\theta_2^{RNN} - \theta_2) + K_d(\dot{\theta}_2^{RNN} - \dot{\theta}_2)) \\
& - I_2(K_p(\theta_4^{RNN} - \theta_4) + K_d(\dot{\theta}_4^{RNN} - \dot{\theta}_4)) \\
T_{r5} =& I_2(K_p(\theta_3^{RNN} - \theta_3) + K_d(\dot{\theta}_3^{RNN} - \dot{\theta}_3)) \\
T_{r6} =& I_2(K_p(\theta_4^{RNN} - \theta_4) + K_d(\dot{\theta}_4^{RNN} - \dot{\theta}_4))
\end{aligned} \tag{4.10}$$

where $\theta_i^{RNN},\ i = 1, 2, 3, 4$ is output of RNN and $\theta_i,\ i = 1, 2, 3, 4$ is actual limb angles of robot.

We drive the biped robot platform by a RNN which is the trained RNN in Section 4.4 with rectifier activation function and learning constant $c = 0.01$ configuration and set of PD controllers in (4.10). Meanwhile, we sampled its posture with 20 Hz sampling frequency for two different input value during the first 5

seconds of locomotion. Hence we obtain Figure 4.6 and Figure 4.7. In these figures, it is easy to see that locomotion speed increases with given input u_0 as required and stable locomotion can be achieved with a position generating RNNs for different input values.

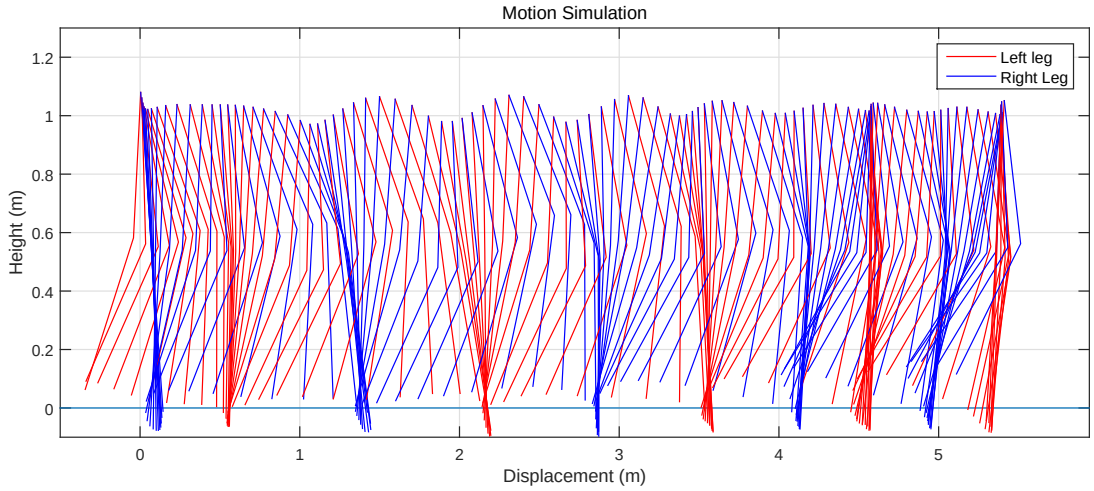

Figure 4.6: Locomotion pattern of RNN driven biped robot platform for $u_0 = 5$ training input during 5 seconds.

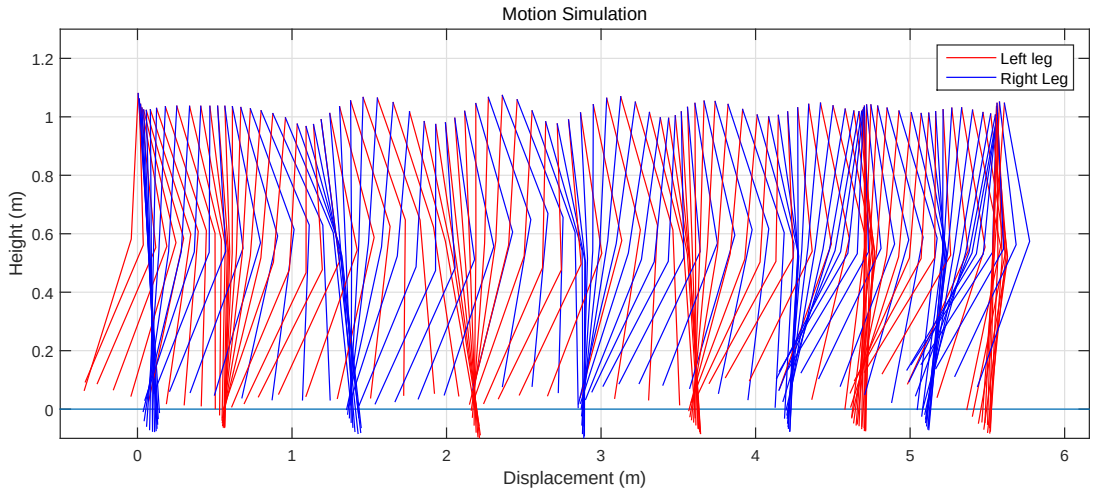

Figure 4.7: Locomotion pattern of RNN driven biped robot platform for $u_0 = 5.15$ test input during 5 seconds.

4.6 Analysis and Discussion

Training simulations in Section 4.4 show us that activation function selection is an important step of RNN training and learning constant selection has also effect on stability and speed of learning process. To reach a conclusion, relation between activation function and features of training patterns needs to be further investigated but it is easy to say that in our simulations important amount of performance difference has been observed between different activation functions. In addition to these, limiting norm of error gradient is a helpful technique to preserve stability of training. On the other hand, it slows down the training speed due to applied normalization operations.

In terms of Figure 4.4 and Figure 4.5, we can claim that RNN learnt almost all main features of desired waveforms and it is known that this RNN showed same success at the other output patterns which correspond to other training and test inputs. From this point of view, we may assume that RNNs have capability of learning an arbitrary waveform and they may reproduce these waveforms when the same inputs applied to RNNs.

Trained RNN with rectifier activation function and learning constant $c = 0.01$ is tested with various training and test set inputs and it shows a good locomotion performance as shown in Figure 4.6 and Figure 4.7. Hence we can conclude that it is possible to drive a biped robot model by reproducing taught motion pattern via RNNs and RNNs are capable of modulating speed of locomotion in terms of input values. Furthermore we need to note that biped robot which is driven by trained RNN has a different stable walking speed upper and lower limits than original model in [16] and to understand reasons of it we need further investigation.

Chapter 5

Conclusion and Future Works

The goal of this thesis is to teach motor patterns to a RNN in an efficient and fast way. In addition to these, we evaluate the stable locomotion generation ability of trained RNNs. At the beginning, we examined proposed motor pattern generating network models in Chapter 2.

Among these networks, we focused on recurrent neural networks (RNN) and performed detailed examination about RNNs by simulations. In these simulations we assessed the performance of some methods which are proposed to overcome encountered problems in the training of neural networks. These problems are long training times and low generalization capability of RNNs. We evaluated the usefulness of various training acceleration and regularization techniques in Chapter 3.

In Chapter 4, we evaluated learning performance of different activation functions and controlled a biped robot platform with trained neural network as motion pattern generator. Finally we see that RNNs are capable of producing stable locomotion and may modulate speed of locomotion.

In terms of performed simulations we conclude that momentum and cross-entropy cost function are found as successful acceleration techniques over training

sets which consist of single pattern or multi patterns. However, modified delta-bar-delta learning rule can accelerate training only for single training pattern. In addition to these, three regularization techniques are tested and we understand that although L1 and L2 regularization did not succeed too much for regularization aiming, promising results were obtained with different application ways of dropout techniques.

Also time constant and gradient limitation techniques are utilized for the purpose of enhancing stability, during training. Then, their successes, advantages, and drawbacks are evaluated in terms of performed simulations.

In terms of performance comparison among activation functions, rectifier function was found as a successful and efficient activation function. Finally biped robot model is driven with a trained RNNs and qualification of RNNs in generating stable walking has been showed with simulations results.

With these results we will direct our research effort to solve three main issues about RNN as future works. Thus, we plan to contribute literature about use of RNN in robotic systems.

First, training speed of BPTT algorithm need to be increased and this will be achieved in two ways. The first way is that parallel implementation capability of this learning algorithm will be investigated and simulations will be performed to confirm its achievements. The second way is that multi pattern performance of delta-bar-delta rule will be tried to enhance and possible alternatives of it will be searched in the literature.

Second, generalization ability of RNNs need to be improved in some ways. In other words, RNN needs to comprehend correlations between inputs and outputs instead of memorizing whole pattern with respect to corresponding input. To do that, we will try to develop new regularization techniques which are compatible for pattern generation. Alternatively, new implementation ways of dropout technique will be investigated for the reason of promising results of it in simulations.

Third, we aim to include dynamics of robot template to neural network training

algorithm to be able to process feedback information during locomotion. Hence walking stability may be increased and adaptation capability of neural controller to different ground conditions enlarged.

Bibliography

[1] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, "Dropout improves recurrent neural networks for handwriting recognition," in *In: 14th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pp. 285 – 290, 2014.

[2] G. Mesnil, X. He, L. Deng, and Y. Bengio, "Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding," in *Interspeech*, pp. 3771 – 3775, 2013.

[3] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1017–1024, 2011.

[4] P. Hénaff, V. Scesa, F. B. Ouezdou, and O. Bruneau, "Real time implementation of ctrnn and bptt algorithm to learn on-line biped robot balance: Experiments on the standing posture," *Control Engineering Practice*, vol. 19, no. 1, pp. 89–99, 2011.

[5] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of the 30th International Conference on Machine Learning*, pp. 1310 – 1318, 2013.

[6] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107 – 116, 1998.

[7] C.-L. Shih, "Ascending and descending stairs for a biped robot," *IEEE Transactions on Systems, Man and Cybernetics-Part A: Systems and Humans*, vol. 29, no. 3, pp. 255 – 268, 1999.

[8] K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue, "Online generation of humanoid walking motion based on a fast generation method of motion pattern that follows desired zmp," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2684 – 2689, 2002.

[9] S. Aoi and K. Tsuchiya, "Stability analysis of a simple walking model driven by an oscillator with a phase reset using sensory feedback," *IEEE Transactions on Robotics*, vol. 22, pp. 391–397, April 2006.

[10] "Central pattern generators for locomotion control in animals and robots: A review," *Neural Networks*, vol. 21, no. 4, pp. 642–653.

[11] J. H. Barron-Zambrano and C. Torres-Huitzil, *CPG implementations for robot locomotion: Analysis and design.* INTECH Open Access Publisher, 2012.

[12] K. Matsuoka, "Sustained oscillations generated by mutually inhibiting neurons with adaptation," *Biological Cybernetics*, vol. 52, no. 6, pp. 367–376, 1985.

[13] K. Matsuoka, "Mechanisms of frequency and pattern control in the neural rhythm generators," *Biological Cybernetics*, vol. 56, no. 5-6, pp. 345–353, 1987.

[14] Y. Maeda, A. Ito, and H. Ito, "Central pattern generator and its learning via simultaneous perturbation method," in *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pp. 1 – 6, 2012.

[15] C.-S. Park, Y.-D. Hong, and J.-H. Kim, "Evolutionary-optimized central pattern generator for stable modifiable bipedal walking," *IEEE/ASME Transactions on Mechatronics*, vol. 19, no. 4, pp. 1374 – 1383, 2014.

[16] G. Taga, Y. Yamaguchi, and H.Shimizu, "Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment," *Biological Cybernetics*, vol. 65, no. 3, pp. 147 – 159, 1991.

[17] Q. Lu and J. Tian, "Research on walking gait of biped robot based on a modified cpg model," *Mathematical Problems in Engineering*, vol. 2015, p. 9, 2015.

[18] J. Buchli, L. Righetti, and A. J. Ijspeert, "Engineering entrainment and adaptation in limit cycle systems," *Biological Cybernetics*, vol. 95, no. 6, pp. 645–664, 2006.

[19] A. Sproewitz, R. Moeckel, J. Maye, and A. J. Ijspeert, "Learning to move in modular robots using central pattern generators and online optimization," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 423–443, 2008.

[20] A. Crespi and A. Ijspeert, "Online optimization of swimming and crawling in an amphibious snake robot," *Robotics, IEEE Transactions on*, vol. 24, pp. 75–87, Feb 2008.

[21] Y. Farzaneh, A. Akbarzadeh, and A. Akbari, "New automated learning cpg for rhythmic patterns," *Intelligent Service Robotics*, vol. 5, no. 3, pp. 169–177, 2012.

[22] A. Ijspeert and J. Kodjabachian, "Evolution and development of a Central Pattern Generator for the swimming of a lamprey," *Artificial Life*, vol. 5, no. 3, pp. 247–269, 1999.

[23] Y. Nakamura, T. Mori, M.-a. Sato, and S. Ishii, "Reinforcement learning for a biped robot based on a cpg-actor-critic method," *Neural Networks*, vol. 20, pp. 723–735, Aug. 2007.

[24] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, no. 4, pp. 500–544, 1952.

[25] B. A. Pearlmutter, "Learning state space trajectories in recurrent neural networks," *Neural Computation*, vol. 1, no. 2, pp. 263 – 269, 1989.

[26] P. Graben, T. Liebscher, and J. Kurths, "Neural and cognitive modeling with networks of leaky integrator units," in *Lectures in Supercomputational Neurosciences* (P. Graben, C. Zhou, M. Thiel, and J. Kurths, eds.), Understanding Complex Systems, pp. 195–223, Springer Berlin Heidelberg, 2008.

[27] F. J. Pineda, "Generalization of back-propagation to recurrent neural networks," *Physical Review Letters*, vol. 59, no. 19, pp. 2229 – 2232, 1987.

[28] B. A. Pearlmutter, "Gradient calculations for dynamic recurrent neural networks: A survey," *IEEE Transactions on Neural Networks*, vol. 6, no. 5, pp. 1212 – 1228, 1995.

[29] H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert, "Optimization and application of echo state networks with leaky-integrator neurons," *Neural Networks*, vol. 20, no. 3, pp. 335 – 352, 2007.

[30] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, pp. 295 – 307, 1988.

[31] Y. Fang and T. J. Sejnowski, "Faster learning for dynamic recurrent backpropagation," *Neural Computation*, vol. 2, pp. 270 – 273, 1990.

[32] G. P. Zhang, "Neural networks for classification: A survey," *IEEE Transactions on Systems, Man, and Cybernatics*, vol. 30, no. 4, pp. 451 – 462, 2000.

[33] P. Golik, P. Doetsch, and H. Ney, "Cross-entropy vs. squared error training: a theoretical and experimental comparison," in *Interspeech*, pp. 1756 – 1760, 2013.

[34] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[35] Y. N. Andrew, "Feature selection, l1 vs. l2 regualarization and rotational invariance," in *Proceedings of the twenty-first international conference on Machine learning*, p. 78, 2004.

[36] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929 – 1958, 2014.

[37] O. A.-H. Li Deng and D. Yu, "A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion," in *International Conference on Acoustic, Speech and Signal Processing*, 2013.

[38] T. Mori, Y. Nakamura, M.-A. Sato, and S. Ishii, "Reinforcement learning for a cpg-driven biped robot," in *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, pp. 623 – 630, AAAI Press, 2004.

[39] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the 14th International Conference on Artifical Intelligence and Statistic (AISTATS)*, vol. 15, pp. 315–323, 2011.

[40] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proceedings of the 30th International Conference on Machine Learning*, vol. 28, 2013.

[41] L. Tóth, "Phone recognition with deep sparse rectifier neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6985 – 6989, 2013.

[42] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157 – 166, 1994.

Appendix A

Equations of Biped Robot Model

Following equations are taken from [16] and they are employed to test locomotion control ability of trained neural network in Chapter 4. In Chapter 4, x_5, x_8, x_{11} and x_{14} are called as θ_1, θ_2, θ_3 and θ_4 for the sake of simplicity, respectively.

System Parameters:

$M = 48,\ m_1 = 7,\ m_2 = 4$

$l_1 = 0.5,\ l_2 = 0.6$

$I_1 = \dfrac{m_1 l_1^2}{12},\ I_2 = \dfrac{m_2 l_2^2}{12}$

$b_1 = 10,\ b_2 = 10$

$b_k = 1000,\ k_k = 10000$

$k_g = 10000,\ b_g = 1000$

$g = 9.8$

Initial Conditions:

$x_1 = 0.0$

$x_2 = 1.09$

$x_3 = x_1 + \frac{l_1}{2} \cos x_5$

$x_4 = x_2 - \frac{l_1}{2} \sin x_5$

$x_5 = 0.45\pi$

$x_6 = x_1 + \frac{l_1}{2} \cos x_8$

$x_7 = x_2 - \frac{l_1}{2} \sin x_8$

$x_8 = 0.57\pi$

$x_9 = l_1 \cos x5 + \frac{l_2}{2} \cos x_{11}$

$x_{10} = x_2 - l_1 \sin x5 - \frac{l_2}{2} \sin x_{11}$

$x_{11} = 0.45\pi$

$x_{12} = l_1 \cos x8 + \frac{l_2}{2} \cos x_{14}$

$x_{13} = x_2 - l_1 \sin x8 - \frac{l_2}{2} \sin x_{14}$

$x_{14} = 0.57\pi$

$\dot{x}_i = 0$, (i=1,2,.., 14)

Equations of Kinematic:

$$x_1 = x_3 - \frac{l_1}{2}\cos x_5 = x_6 - \frac{l_1}{2}\cos x_8$$
$$x_2 = x_4 - \frac{l_1}{2}\sin x_5 = x_7 - \frac{l_1}{2}\sin x_8$$
$$x_3 + \frac{l_1}{2}\cos x_5 = x_9 - \frac{l_2}{2}\cos x_{11}$$
$$x_4 - \frac{l_1}{2}\sin x_5 = x_1 0 + \frac{l_2}{2}\sin x_{11}$$
$$x_6 + \frac{l_1}{2}\cos x_8 = x_{12} - \frac{l_2}{2}\cos x_{14}$$
$$x_7 + \frac{l_1}{2}\sin x_8 = x_{13} + \frac{l_2}{2}\sin x_{14}$$
$$(x_r, y_r) = (x_9 + \frac{l_2}{2}\cos x_{11}, x_{10} - \frac{l_2}{2}\sin x_{11})$$
$$(x_l, y_l) = (x_{12} + \frac{l_2}{2}\cos x_{14}, x_{13} - \frac{l_2}{2}\sin x_{14})$$

$$\ddot{x}_1 - \ddot{x}_3 - \frac{l_1}{2}\sin x_5\ddot{x}_5 = \frac{l_1}{2}\cos x_5\dot{x}_5^2$$
$$\ddot{x}_2 - \ddot{x}_4 - \frac{l_1}{2}\cos x_5\ddot{x}_5 = -\frac{l_1}{2}\sin x_5\dot{x}_5^2$$
$$\ddot{x}_1 - \ddot{x}_6 - \frac{l_1}{2}\sin x_8\ddot{x}_8 = \frac{l_1}{2}\cos x_8\dot{x}_8^2$$
$$\ddot{x}_2 - \ddot{x}_7 - \frac{l_1}{2}\cos x_8\ddot{x}_8 = -\frac{l_1}{2}\sin x_8\dot{x}_8^2$$
$$\ddot{x}_3 - \frac{l_1}{2}\sin x_5\ddot{x}_5 - \ddot{x}_9 - \frac{l_2}{2}\sin x_{11}\ddot{x}_{11} = \frac{l_1}{2}\cos x_5\dot{x}_5^2 + \frac{l_2}{2}\cos x_{11}\dot{x}_{11}^2$$
$$\ddot{x}_4 - \frac{l_1}{2}\cos x_5\ddot{x}_5 - \ddot{x}_{10} - \frac{l_2}{2}\cos x_{11}\ddot{x}_{11} = -\frac{l_1}{2}\sin x_5\dot{x}_5^2 - \frac{l_2}{2}\sin x_{11}\dot{x}_{11}^2$$
$$\ddot{x}_6 - \frac{l_1}{2}\sin x_8\ddot{x}_8 - \ddot{x}_{12} - \frac{l_2}{2}\sin x_{14}\ddot{x}_{14} = \frac{l_1}{2}\cos x_8\dot{x}_8^2 + \frac{l_2}{2}\cos x_{14}\dot{x}_{14}^2$$
$$\ddot{x}_7 - \frac{l_1}{2}\cos x_8\ddot{x}_8 - \ddot{x}_{13} - \frac{l_2}{2}\cos x_{14}\ddot{x}_{14} = -\frac{l_1}{2}\sin x_8\dot{x}_8^2 - \frac{l_2}{2}\sin x_{14}\dot{x}_{14}^2$$

Equations of Motion:

$$
\begin{aligned}
M\ddot{x}_1 =& F_1 + F_3 \\
M\ddot{x}_2 =& F_2 + F_4 - Mg \\
m_1\ddot{x}_3 =& -F_1 + F_5 \\
m_1\ddot{x}_4 =& -F_2 + F_6 - m_1 g \\
I_1\ddot{x}_5 =& -F_1\frac{l_1}{2}\sin x_5 - F_2\frac{l_1}{2}\cos x_5 - F_5\frac{l_1}{2}\sin x_5 - F_6\frac{l_1}{2}\cos x_5 \\
& - b_1|x_5 - \frac{\pi}{2}|\dot{x}_5 - (b_2 + b_k f(x_5 - x_{11}))(\dot{x}_5 - \dot{x}_{11}) - k_k h(x_5 - x_{11}) + T_{r1} + T_{r3} \\
I_1\ddot{x}_8 =& -F_3\frac{l_1}{2}\sin x_8 - F_4\frac{l_1}{2}\cos x_8 - F_7\frac{l_1}{2}\sin x_8 - F_8\frac{l_1}{2}\cos x_8 \\
& - b_1|x_8 - \frac{\pi}{2}|\dot{x}_8 - (b_2 + b_k f(x_8 - x_{14}))(\dot{x}_8 - \dot{x}_{14}) - k_k h(x_8 - x_{14}) + T_{r2} + T_{r4} \\
I_2\ddot{x}_{11} =& -F_5\frac{l_2}{2}\sin x_{11} - F_6\frac{l_2}{2}\cos x_{11} - F_{g1}\frac{l_2}{2}\sin x_{11} - F_{g2}\frac{l_2}{2}\cos x_{11} \\
& - (b_2 + b_k f(x_5 - x_{11}))(\dot{x}_{11} - \dot{x}_5) + k_k h(x_5 - x_{11}) - T_{r3} - T_{r5} \\
I_2\ddot{x}_{14} =& -F_7\frac{l_2}{2}\sin x_{14} - F_8\frac{l_2}{2}\cos x_{14} - F_{g3}\frac{l_2}{2}\sin x_{14} - F_{g4}\frac{l_2}{2}\cos x_{14} \\
& - (b_2 + b_k f(x_8 - x_{14}))(\dot{x}_{14} - \dot{x}_8) + k_k h(x_8 - x_{14}) - T_{r4} - T_{r6} \\
f(x) =& max(0, x) \\
h(x) =& \begin{cases} 0 & (x \leqslant 0) \\ 1 & (x > 0) \end{cases} \\
F_{g1} =& \begin{cases} -k_g(x_r - x_{r0}) - b_g\dot{x}_r & \text{for } y_r - y_g(x_r) < 0 \\ 0 & otherwise \end{cases} \\
F_{g2} =& \begin{cases} -k_g(y_r - y_{r0}) - b_g f(-\dot{y}_r) & \text{for } y_r - y_g(x_r) < 0 \\ 0 & otherwise \end{cases} \\
F_{g3} =& \begin{cases} -k_g(x_l - x_{l0}) - b_g\dot{x}_l) & \text{for } y_l - y_g(x_l) < 0 \\ 0 & otherwise \end{cases} \\
F_{g4} =& \begin{cases} -k_g(y_l - y_{l0}) - b_g\dot{y}_l) & \text{for } y_l - y_g(x_l) < 0 \\ 0 & otherwise \end{cases}
\end{aligned}
$$

In the simulations we have chosen the torque control inputs $T_{ri},\ i = 1, 2, ..., 6$, as follows and the control pairs are chosen as $K_p = 7000$ and $K_d = 700$.

Design of PD controller:

$$
\begin{aligned}
T_{r1} =& I_1(K_p(\theta_1^{RNN} - \theta_1) + K_d(\dot{\theta}_1^{RNN} - \dot{\theta}_1)) \\
T_{r2} =& I_1(K_p(\theta_2^{RNN} - \theta_2) + K_d(\dot{\theta}_2^{RNN} - \dot{\theta}_2)) \\
T_{r3} =& I_1(K_p(\theta_1^{RNN} - \theta_1) + K_d(\dot{\theta}_1^{RNN} - \dot{\theta}_1)) \\
& - I_2(K_p(\theta_3^{RNN} - \theta_3) + K_d(\dot{\theta}_3^{RNN} - \dot{\theta}_3)) \\
T_{r4} =& I_1(K_p(\theta_2^{RNN} - \theta_2) + K_d(\dot{\theta}_2^{RNN} - \dot{\theta}_2)) \\
& - I_2(K_p(\theta_4^{RNN} - \theta_4) + K_d(\dot{\theta}_4^{RNN} - \dot{\theta}_4)) \\
T_{r5} =& I_2(K_p(\theta_3^{RNN} - \theta_3) + K_d(\dot{\theta}_3^{RNN} - \dot{\theta}_3)) \\
T_{r6} =& I_2(K_p(\theta_4^{RNN} - \theta_4) + K_d(\dot{\theta}_4^{RNN} - \dot{\theta}_4))
\end{aligned}
$$

where $\theta_i^{RNN},\ i = 1, 2, 3, 4$ is output of RNN and $\theta_i,\ i = 1, 2, 3, 4$ is actual limb angles of robot. In addition to these θ_1, θ_2, θ_3 and θ_4 are state variables x_5, x_8, x_{11} and x_{14} in original biped robot model, see [16], respectively.