

IDENTIFICATION AND ADAPTIVE CONTROL OF BIPEDAL ROBOT MOTION WITH ARTIFICIAL NEURAL NETWORKS

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

By
Burak Çatalbaş
July 2024

IDENTIFICATION AND ADAPTIVE CONTROL OF BIPEDAL
ROBOT MOTION WITH ARTIFICIAL NEURAL NETWORKS

By Burak Çatalbaş

July 2024

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

Ömer Morgül(Advisor)

Selim Aksoy

Tolga Çukur

İsmail Uyanık

Aykut Yıldız

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

IDENTIFICATION AND ADAPTIVE CONTROL OF BIPEDAL ROBOT MOTION WITH ARTIFICIAL NEURAL NETWORKS

Burak Çatalbaş

Ph.D. in Electrical and Electronics Engineering

Advisor: Ömer Morgül

July 2024

Artificial Neural Networks (ANNs) is one of the most popular fields of machine learning thanks to the critical improvements in the last decade, including their applications in the field of robotics and control. The important usage of neural networks in robotics makes it possible for robots to act and interact similar to humans. In this manner, legged robots are important platforms to mimick human locomotion. However, there are significant difficulties to apply system identification and control schemes for these hybrid dynamical structures. With this purpose, this thesis focuses on using artificial neural network-based novel techniques on these problems, for reaching to an efficient walking ability for bipedal robot systems like their counterparts in the nature.

In this thesis, our work to find and apply our novel techniques is mainly divided into two parts. In the first part, inspired by a class of activation functions frequently used in deep learning literature, we propose a novel activation function and investigate its performance in various segmentation and classification tasks by using different well-known datasets. In the second part of the thesis, biped robot locomotion is chosen as the main topic. Separate datasets are created for three experiment configurations. For 2D and 3D simulations, locomotion control, system identification and adaptive control are applied with neural networks for successful periodical walking with low errors, having approximations of robot models and preparing for the adaptive learning using both control and identification blocks, respectively. For 2D physical robot system, system identification is completed for a walking dataset generated with varying speed levels. For all cases, proposed novel activation function DELU (Extended Exponential Linear Unit) and its tuned functions are tried together with other activation functions in comparison, to reach better performances.

Keywords: Artificial Neural Networks, Recurrent Neural Networks, Adaptive Control, System Identification, Robot Locomotion.

ÖZET

İKİ AYAKLI ROBOT HAREKETİNİN YAPAY SİNİR AĞLARI İLE TANILANMASI VE UYARLANABİLİR KONTROLÜ

Burak Çatalbaş

Elektrik ve Elektronik Mühendisliği, Doktora

Tez Danışmanı: Ömer Morgül

Temmuz 2024

Yapay Sinir Ağları (YSA) son 10 yıldaki kritik ilerlemeler sayesinde makine öğrenmesinin en revaçta alanlarından biridir, robotik ve kontrol alanındaki uygulamaları buna dahildir. Sinir ağlarının robotik alanındaki önemli kullanımı robotların insanlara benzer şekilde davranabilmesi ve iletişime geçebilmesini sağlar. Bu açıdan, bacaklı robotlar insan lokomasyonunu taklit etmek için önemli platformlardır. Ancak, bu hibrit dinamik yapılara sistem tanılama ve kontrol uygulamak için kayda değer zorluklar bulunmaktadır. Bu amaçla, işbu tez iki bacaklı robot sistemlerinin doğadaki karşılıkları gibi etkili yürüme kabiliyetine erişmesi için bu sorunlar üzerinde yapay sinir ağı temelli yeni teknikler kullanmaya odaklanır.

Bu tezde, yeni teknikler bulma ve uygulama çalışmamız ana olarak iki kısma ayrılmıştır. İlk kısımda derin öğrenme literatüründe sık kullanılan bir aktivasyon fonksiyonu sınıfından esinlenilerek, yeni bir aktivasyon fonksiyonu önerdik ve bu fonksiyonun performansını farklı ve iyi bilinen veri setleri üzerinde, çeşitli segmentasyon ve sınıflandırma görevlerinde araştırdık. Tezin ikinci kısmında, iki ayaklı robot lokomasyonu ana konu olarak seçilmiştir. Üç deney konfigürasyonu için ayrı veri setleri oluşturulmuştur. 2D ve 3D simülasyonları için, sinir ağlarıyla lokomasyon kontrolü, sistem tanılama ve uyarlanabilir kontrol sırasıyla düşük hatayla başarılı periyodik yürüme, robot modellerinin yakınsamalarını elde etme ve hem kontrol hem de tanılama blokları kullanılan uyarlanabilir öğrenmeye hazırlanma için uygulanmıştır. 2D fiziksel robot sistemi için, değişen hız değerleriyle oluşturulan bir veri seti için sistem tanılama yapılmıştır. Tüm konfigürasyonlar için, önerilen yeni aktivasyon fonksiyonu DELU (Genişletilmiş Üstel Lineer Birim) ve ince ayar yapılan DELU fonksiyonları daha iyi performansa erişmek için diğer aktivasyon fonksiyonlarıyla birlikte denenerek kıyaslanmıştır.

Anahtar sözcükler: Yapay Sinir Ağları, Tekrarlayan Sinir Ağları, Uyarlanabilir Kontrol, Sistem Tanılama, Robot Lokomasyonu.

Acknowledgement

My Doctoral program became a six-year adventure in which I enjoyed the support of my friends, teachers and my precious family. Here, I would like to thank them as I can.

My first thanking goes to my supervisor Prof. Dr. Ömer Morgül for his trust and support, in addition to his invaluable guidance throughout my academic career. Without all these, it would not be possible for me to constitute this thesis. I also thank to Prof. Dr. Selim Aksoy, Prof. Dr. Tolga Çukur, Assoc. Prof. Dr. İsmail Uyanık and Asst. Prof. Dr. Aykut Yıldız for approving my work, paving the way for qualification as thesis.

Another appreciation goes to my friends from Bilkent. During my undergraduate and graduate programs, I always relied on their friendship and support to be successful in my academic and personal life. I want to thank all of them.

I also thank to staff of our department, namely Mürüvet Parlakay, Ash To-suner, Ergün Hırlakoğlu, Aydan Eren, Metin Kayabaşı, Funda Bülbül, Onur Bostancı, and Ufuk Tufan for their kind understanding and charitable behaviors during my undergraduate and graduate career.

I'm appreciative of the financial support from the Scientific and Technological Research Council of Turkey (TÜBİTAK) in the scope of BİDEB 2211-E National Ph.D. Scholarship Program and ARDEB 1001 Program for my graduate studies. The work presented in this thesis was supported by TÜBİTAK through the ARDEB 1001 project with number 120E104.

I appreciate NVIDIA Corporation for their donation of NVIDIA Titan Xp GPU, Quadro P6000 GPU cards and also their software support, which was a major component of my working environment which I used to complete my thesis.

Some of the work presented in this thesis work is first published in Neural Computing and Applications journal, 35-30, 22705-22724, 2023 by Springer Nature and reproduced with permission from Springer Nature, see [1].

My last thank-you goes to my irreplaceable ones. I'd like to thank my mother Zehra Çatalbaş, my father Sezai Çatalbaş, and my brother Bahadır Çatalbaş for their moral and material support, from the beginning of my life. I'm fortunate to be part of this family. Without any one of them, I could not be here at all.

To my family, with love and gratitude...



Contents

1	Introduction	1
1.1	Background and Motivation	2
1.2	Main Contributions of the Thesis	3
1.3	Organization of the Thesis Work	4
2	Activation Functions in Neural Networks	6
2.1	Activation Functions	6
2.1.1	Classical Activation Functions	7
2.1.2	Partially Linear Activation Functions	8
2.1.3	Proposed DELU Activation Function	12
2.1.4	Effects of DELU in Training Process	14
2.1.5	Output Landscapes	18
2.1.6	Results Obtained	20
2.1.7	Hypothesis Testing and Z-Score Analysis	29
2.2	Results for Different Initialization Methods with Different Activation Functions	41
2.2.1	Proposed Truncated Towers Initialization	42
2.3	Other Methods of the Neural Networks	47
2.4	Chapter Summary	51
3	System Identification and Adaptive Control of Bipedal Locomotion by Using Artificial Neural Network Structures	52
3.1	Literature Review	53
3.2	Two Dimensional Robot Model	56
3.2.1	Biped Robot Model	56

3.2.2	Central Pattern Generator	59
3.2.3	Dataset Generation	61
3.2.4	Neural Locomotion Controller	65
3.2.5	System Identification With Neural Networks	70
3.2.6	Neural Network Based Model Reference Adaptive Control	76
3.3	Three Dimensional Robot Model	85
3.3.1	Robot Model Demonstration	85
3.3.2	Central Pattern Generator	85
3.3.3	Dataset Generation	91
3.3.4	Neural Locomotion Controller	93
3.3.5	System Identification With Neural Networks	96
3.3.6	Neural Network Based Model Reference Adaptive Control	100
3.4	Physical Biped Robot System Development	105
3.4.1	Biped Robot System	107
3.4.2	Dataset Generation	111
3.4.3	System Identification With Neural Networks	113
3.5	Chapter Summary	116
4	Conclusion and Future Works	118
	Bibliography	122
	Appendix	131

List of Figures

1.1	Retired hydrolic version of ATLAS humanoid robot performs a stunt that requires advanced computer vision and robotics precision, shown by an image from the video of Boston Dynamics [3]. .	3
2.1	ReLU and ELU activation functions are given.	9
2.2	ReLU, Softplus, FELU, ELU, SELU and MPELU activation functions are given.	11
2.3	Graphs of DELU activation function (a) and its derivative (b) for $\alpha = 1, \beta = 2$	14
2.4	Obtained relation graph for ReLU, ELU and DELU activation functions.	18
2.5	Obtained landscape graphs for three different activation functions.	19
2.6	Oxford IIIT-Pets data sample and its truth map, constructed from dataset [30].	20
2.7	Representation of our U-net, takes a image input and outputs segmented image.	21
2.8	Example data samples: CIFAR-10 at left and CIFAR-100 at right [32].	24
2.9	General ResNet structure is shown using blocks.	24
2.10	Fashion MNIST dataset samples taken from [33].	26
2.11	The non-parametric PDF for random variable S_{ELU} of success rates, which will be used according to process described here. . . .	30

2.12	(a) contains two different Gaussian-kernel probability distribution functions for ELU and DELU. (b) is the result of 9 convolution operations for these two PDFs with themselves, for both ELU and DELU separately. It is obvious that the PDF evolves to Gaussian-like shape even with convolving only 10 PDFs, as expected by CLT.	32
2.13	Gaussians related to ReLU, ELU and DELU based on Table 2.1 at left (a) and Table 2.2 at right (b). Figure 2.13 (a) and Figure 2.12 (b) are evidently similar, as CLT expected.	33
2.14	Candidate initialization method is displayed as a figure in comparison to Gaussian and Laplacian functions.	43
2.15	The PDF of chosen Truncated Towers is displayed.	44
2.16	A comparison of gradient descent and stochastic gradient descent.	47
2.17	A perceptron is given at left, a feedforward network is given at right.	50
2.18	BPTT-applied recurrent neuron at left, LSTM cell is at right [48].	50
3.1	Robot ASIMO is shown in the picture, where knees are half-closed while it stands up because of the constraint introduced by its locomotion algorithms [56].	54
3.2	The illustration of 2D robot model for bipedal robot locomotion, inspired from and drawn similar to [66].	57
3.3	Structure of central pattern generator employed to locomote biped robot model, see [66].	59
3.4	The successful walking dataset generated by CPG, is divided into training, validation and test sets as illustrated.	62
3.5	The dataset samples which contain falls are shown visually, being divided into training, validation and test sets as illustrated.	63
3.6	The control diagram with neural locomotion controller.	65
3.7	Performance of the neural controller using 43 LSTM and 74 ReLU layers with 0.1 L2 regularization, on separated training, validation and test sets.	69
3.8	Parallel system identification configuration is presented, inspired from [48].	70

3.9	The neural network structures used for system identification network for 2D robot simulations where N demonstrates time step number.	72
3.10	Training, validation and test results are shown for networks with; (a) for ReLU, (b) for DELU, (c) for Tanh layers	73
3.11	The control diagram with neural locomotion controller and neural network based biped robot model from the system identification part.	76
3.12	NICO robot is viewed from different angles using V-REP simulator. For more information about the robot model, please refer to [67].	86
3.13	The illustrations of NICO, with its servos highlighted on the left and its individual elements such as hands, feet etc. are encapsulated on the right. For more information about the robot model, please refer to [67].	87
3.14	The illustration of CPG network used for bipedal locomotion of NICO 3D robot model, inspired from [69].	90
3.15	NICO robot is standing on a flat surface generated by V-REP. . .	90
3.16	The walking dataset generated by CPG, is divided into training, validation and test sets as illustrated for the case of NICO 3D robot.	91
3.17	The trajectories of NICO robot are displayed 2D, on the X and Y axes for each sample. The movement is projected to the X-Y walking plane, and vertical Z-axis is ignored for the purpose of demonstration.	92
3.18	Neural Locomotion Controller is presented for 3D case, inspired from [48].	94
3.19	Parallel system identification configuration is presented, inspired from [48].	96
3.20	The neural network structure used for system identification network for 3D robot simulations.	98
3.21	The control diagram with neural locomotion controller and neural network based biped robot model from the system identification part for the case of NICO 3D robot.	100
3.22	Photographs from the experiments with 2D physical biped robot.	105

3.23	The 2D planarized robot with the connected system is shown on its track.	106
3.24	The complete track used for robot locomotion is shown in this photo.	106
3.25	CAD drawings of the 2D physical biped robot model.	107
3.26	The circuit is making position control using PID, inside a Dynamixel XH540-W150-R servo [77]	108
3.27	Some of RAM addresses for ID 1 servo are displayed by the program.	108
3.28	The sensor is shown from the top at left image, and the places of sensors at the robot feet is shown at the right.	109
3.29	Iteratively designed 3D parts, earliest is shown at the left and latest at the right.	110
3.30	The finalized version of our 2D planarized robot.	110
3.31	The visualization of chosen dataset samples; x axis shows the speed coefficient which is used to show the speed ratio of applied gait, and y axis shows the speed reached by the 2D biped robot.	112
3.32	System identification configuration is presented, inspired from [48].	113
3.33	Training, validation and test results are shown for networks from the first entry of the Table 3.19 to last entry, with letters from (a) to (d), respectively. Dots indicate the loss results of the networks at the epoch with lowest validation losses during their trainings.	115

List of Tables

2.1	Oxford-IIIT Pets Dataset, 20 U-Net-Like networks.	23
2.2	CIFAR-10, 10 ResNet20-v1 networks.	25
2.3	CIFAR-100, 10 ResNet164-v2 networks.	26
2.4	Fashion MNIST, 10 ResNet20-v1 networks at the end of 200 epochs.	27
2.5	Fashion MNIST, 10 ResNet20-v1 networks with highest validation success.	27
2.6	Mean times of classification and segmentation networks in seconds.	28
2.7	Alpha vs. Z-score, for hypothesis tests.	35
2.8	Alpha vs. Z-score, for hypothesis tests.	37
2.9	Fashion MNIST and CIFAR-10 mean success rates of different DELU alternatives.	39
2.10	Fashion MNIST Success Rates (%)	45
2.11	CIFAR-10 Success Rates (%)	45
2.12	Fashion MNIST Success Rates (%)	46
2.13	Optimizers and their respective success rates for Fashion MNIST.	48
2.14	Optimizers and their respective success rates for CIFAR-10 dataset.	48
2.15	Loss functions and their respective success rates for Fashion MNIST.	49
2.16	Loss functions and their respective success rates for CIFAR-10. . .	49
3.1	Properties of generated 2D robot locomotion data.	62
3.2	Walking performances of neural controller for bipedal 2D locomotion.	68
3.3	System identification network results in comparison.	74
3.4	System identification results are given in comparison for DELU activation function tunings.	74
3.5	System identification network results in comparison.	75

3.6	Performances of adaptive neural controller training for bipedal 2D locomotion using the SID network, which is trained with only successful walks.	80
3.7	Evaluation of the effect of the first neural layer in system identification neural networks on adaptive control performance under Scenario #1 conditions for bipedal 2D locomotion.	81
3.8	Evaluation of the effect of the first neural layer in system identification neural networks on adaptive control performance under Scenario #2 conditions for bipedal 2D locomotion.	82
3.9	Evaluation of the effect of the first neural layer in system identification neural networks on adaptive control performance under Scenario #3 conditions for bipedal 2D locomotion.	82
3.10	Evaluation of the effect of the first neural layer in system identification neural networks on adaptive control performance under Scenario #4 conditions for bipedal 2D locomotion.	83
3.11	Properties of generated 3D robot locomotion data.	93
3.12	Results of neural controller trained with 3D robot locomotion data.	95
3.13	System identification network results are given in comparison.	98
3.14	System identification results are given in comparison for DELU activation function tunings.	99
3.15	Change of performance metrics throughout adaptive control of 3D robot locomotion using 140 ReLU neurons in the first hidden layer of system identification neural network.	103
3.16	Change of performance metrics throughout adaptive control of 3D robot locomotion using 140 DELU neurons in the first hidden layer of system identification neural network.	104
3.17	Properties of generated 2D robot locomotion data from real-life.	111
3.18	Detailed information about the data acquired from experiments.	112
3.19	System identification network results are given in comparison.	115

Chapter 1

Introduction

This thesis starts with discussing methods of artificial neural networks and their applications, with a focus on activation functions. Afterwards, applications of different neural network structures on control, identification and adaptive control schemes on 2D and 3D systems are discussed. Then, a 2D physical biped robot is built for creating a dataset, and making system identification with neural networks using this data. Finally, the thesis is concluded with mentioning the key contributions made, results obtained and future works on the research.

To acknowledge, Titan Xp, Quadro P6000, which are donated by NVIDIA Corporation, and Quadro RTX 6000 Graphical Processing Units (GPUs) are used for training and inference of the neural networks, while HP workstations with Intel Xeon Central Processing Units (CPUs) are employed to run the experiments throughout the thesis, thanks to their high processing power.

This chapter starts with the background and motivation for usage of neural networks. Then, the main contributions planned for the thesis are explained. Afterwards, the organization of the dissertation is presented.

1.1 Background and Motivation

Artificial neural networks are inspired from real neurons. Attempts for modeling real neurons to create artificial, neuron-like structures are dated back to 20th century. McCulloch and Pitts came up with a standard perceptron structure and subsequently ADALINE and MADALINE are proposed by Widrow and Hoff in 1960s [2]. Modeling of convolutional neural networks happened two decades later, however the lack of technical equipment and resources limited their training. Feedforward neural networks are offered alongside convolutional neural networks, and because of their comparative simplicity they are started to be used through the end of 20th century. However, the emergence of other machine learning techniques which require even less amount of resource and computational power led to rise of other methods such as Support Vector Machines (SVM) for classification and regression tasks, and k-means or similar algorithms for clustering tasks and other techniques, rather than neural networks. With the 21th century though, the improvements of technology have led to the rise of neural networks and today, they are used in many different applications - an example is presented in the Figure 1.1 where computer vision, object detection and feedback systems can be controlled by using artificial neural networks. This potential is the main motivation behind neural network-based applications.

In this thesis, detailed literature reviews are completed for different neural network implementations. Chapter 2 contains related sections for the literature review on image classification and segmentation examples, and methods employed in those examples - starting with the activation functions which are used in different neural network architectures. Also, for a comprehensive review of robot locomotion, informative sections are present in Chapter 3 alongside specific parts where usage of neural networks are explained. In addition to these passages, works referenced throughout those chapters are also beneficial by providing more insight.



Figure 1.1: Retired hydrolic version of ATLAS humanoid robot performs a stunt that requires advanced computer vision and robotics precision, shown by an image from the video of Boston Dynamics [3].

1.2 Main Contributions of the Thesis

Briefly, this thesis contributes to the literature in multiple directions. It introduces Extended Exponential Linear Unit (DELU) activation function and makes experiments with different network architectures, tasks and datasets. Also, the thesis uses statistical significance analysis for assessing the results of these experiments to confirm the success rates using Z-scores and confidence intervals to find percentage of confidences for determining how DELU is better than the others. In addition, a compatibility analysis is also provided for showing the capability of DELU to work with different initialization methods and various DELU tunings are generated and tested with different datasets. Indeed, it could be possible to tune DELU parameters by treating them as hyperparameters of the network models as well. Addition of the DELU activation function with the compatibility tests and application of hypothesis testing to assess successes using statistical significance are two different contributions of this thesis to the literature.

The thesis used control, system identification and adaptive control schemes

for application on 2D and 3D simulations with different neural network configurations, which included network architectures with DELU and its tunings as well. These constitute one contribution of our thesis to the literature. Finally, thesis presents with actual experiments conducted on a 2D physical biped robot system and a system identification with this data, which pose as other contributions to the literature. In short, the thesis also makes contributions to the field of applications of neural networks for the robotics, neural networks and control systems.

1.3 Organization of the Thesis Work

The thesis dissertation consists of four main chapters. In this chapter, main contributions and organization of the thesis are explained. Second and third chapter presents the work done. Fourth chapter concludes the thesis work and discusses future directions, then bibliography and appendix sections end the document.

The detailed content of this thesis is as follows. In the second chapter, an expanded review of activation functions is made with both image classification and image segmentation tasks, and a contribution of this thesis to the literature are presented: Extended Exponential Linear Unit (DELU) activation function. Using different neural architectures, different tasks and different datasets; DELU activation function is thoroughly investigated with multiple comparisons and compatibility configurations, including testing various DELU tunings. These tests included hypothesis testing and Z-score analysis as well. Also, a brief review of deep learning methods used in the thesis work are done afterwards. In the third chapter, combination of methods from the literature for control, system identification and adaptive control using neural networks, which include our novel activation function DELU and its tunings, are used for the trials. This included completions of control systems, system identification and adaptive control tasks in the field of robotics with the usage of different neural network architectures including DELU tunings in comparison. In this manner, one more dimension is

added to the thesis work - both in terms of network architectures and application areas. In the last chapter, thesis is concluded with conclusion and future works sections about both DELU activation function and robot applications for simulation and real-life experiments.



Chapter 2

Activation Functions in Neural Networks

In this chapter, activation functions for neural networks are discussed with image classification and image segmentation examples, and then their results are comparatively presented using different neural network architectures. Afterwards, other methods and aspects which are used in these network architectures are mentioned.

2.1 Activation Functions

Activation functions are important parts of the artificial neural networks from the beginning of their invention. From Pitts-McCulloch neuron of 1943 [2] until today, many different activation functions are proposed and some of them are regarded as ‘improvement’ due to their significant success in comparison to the others. For the classification and related tasks, namely sigmoid, hyperbolic tangent and then ReLU family members are proposed [4] as an important step towards to the modern neural network architecture. They are mainly used for feedforward [5]-[6], convolutional [7],[8], recurrent [9]-[10] etc. types of artificial neural networks. The

former activation functions mentioned above, Sigmoid and hyperbolic tangent, are well known and the information about them could be found in classical textbooks and surveys, e.g. [11]-[12]. The latter activation function family, which will be considered in this work, are given with graphical representations in the next section.

Mainly to eliminate the vanishing gradient problem, in [13] the authors have started with different alternatives such as Noisy ReLU and their work has evolved to standardized ReLU. It was also claimed that building networks is like an art [14]. This points out to difficulties of the design and, sometimes, lack of clear theoretical explanation for the statistical success of methods employed in neural networks. There are different neuron types, network types, and datasets to experiment on, which makes the selection of an appropriate structure even more complex. Convolutional neural networks, for example, needs more generalization as seen in [15]. In addition to data augmentation, regularization and network pruning are among the techniques that are employed in convolutional neural networks for better performance [15]-[16].

The number of scientific works focusing on newly defined activation functions continually increases in recent years, see [17]. For more information, interested reader is referred to recent survey papers [12], [17] and references therein.

Since there are numerous activation function candidates, it would not be possible to discuss even the majority of them within the scope of our work. Hence, we will mainly focus on the ones which are mostly utilized in neural network studies and are most relevant to our work.

2.1.1 Classical Activation Functions

Especially in early neural networks, biologically inspired activation functions were used frequently. An important one is Sigmoid, and its formula is given:

$$Sigmoid(z) = \frac{1}{1 + e^{-z}} \quad (2.1)$$

This function is still used in recurrent neural networks, such as LSTM cells etc. Hard-limiter function is also one of these activation functions, given as follows:

$$hardlim(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0. \end{cases} \quad (2.2)$$

Finally, hyperbolic tangent ($tanh(.)$) is another classical activation function. Its definition is given below. Due to its zero-mean, it is used in a wider context than the Sigmoid in many neural network structures.

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (2.3)$$

2.1.2 Partially Linear Activation Functions

Rectified Linear Unit (ReLU), is first proposed in [13] as a suitable neural network activation function, boosted the training speed and success rates of the multilayer neural networks considerably in various cases. Mostly eliminating the vanishing gradient problem and speeding up the learning algorithms because of simplicity of its mathematical structure, ReLU is still used in many novel network designs for classification, clustering, recognition tasks by networks such as Inception, GoogLeNet etc. Nonetheless, its zero part named as ‘dead zone’ causes slowing down for learning process of neural network, in comparison to other activation functions. Afterwards, the alternatives to ReLU have emerged, and an important one is the Exponential Linear Unit (ELU).

$$ReLU(z) = \max\{0, z\} = \begin{cases} z & z > 0 \\ 0 & z \leq 0. \end{cases} \quad (2.4)$$

ELU is introduced in [18], and it contains a smoothness addition -as seen in the Figure 2.1- which causes slightly more complexity in exchange of a better performance. Based on good aspects of ReLU variants, such as the existence of non-zero negative part, ELU offers significant improvement of success rates especially on certain applications – as an example, the results for the pattern classification problem with ELU-based neural networks on various datasets can be found in [18]. Also, ELU introduces another free parameter whose value needs to be determined, the value of alpha parameter (α) for the activation function - whose default value is mostly chosen as 1 in the literature. ELU is also picked up as an alternative of the ReLU activation function mainly in the literature.

$$ELU(z) = \begin{cases} z & z > 0 \\ \alpha(e^z - 1) & z \leq 0. \end{cases} \quad (2.5)$$

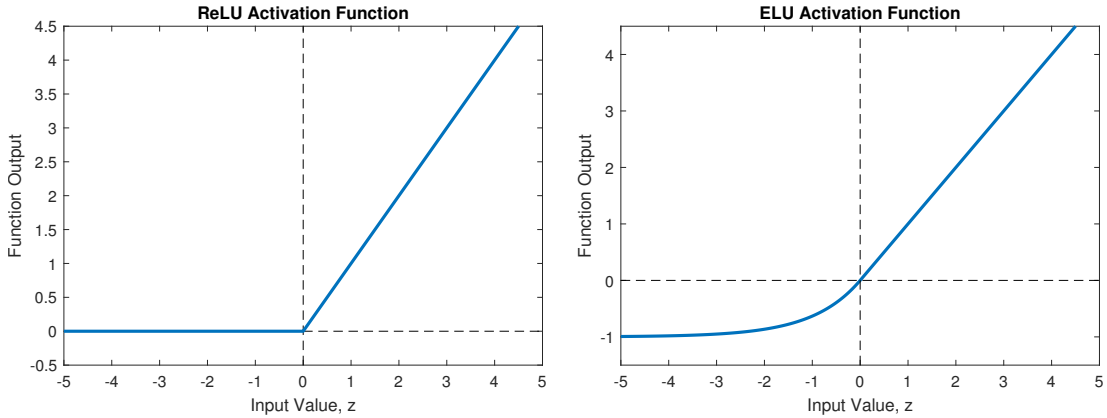


Figure 2.1: ReLU and ELU activation functions are given.

There are different classifications of activation functions, for example, specialties-based fixed shape vs. trainable shape classification [17], application-based activation function classifications [12], bounded vs. unbounded classification etc. To mention some of those activation functions, we can label Leaky ReLU and Scaled Exponential Linear Unit (SELU) activation functions. Leaky ReLU introduces a linear part at negative side of the ReLU activation function, with the ‘leak’ is determined by a parameter just like ELU [19]. This function

aims on ‘dead zone’ problem as well, however it causes exploding gradient problem for both positive and negative inputs. Softplus activation function is also offered as smooth approximation to ReLU [12]. Another activation function is SELU, which modifies the negative input part of ReLU for self-normalization procedure to learn better. However, this activation function is aimed to be used with a specific weight initialization method [19]. Recently introduced “Swish Activation Function” is a modification of classical sigmoidal function [20] which is formulated simply as $swish(z) = z \cdot sigmoid(\beta z)$ with β as a trainable parameter. It is used in some convolutional neural network applications as well [20, 21]. Remarkable differences for this activation function is suggested as its non-monotonic nature and probable slow learning encountered in the far negative part of the activation function; due to its derivate decreasing to zero as input goes to negative infinity [20]. Later, various modifications of Swish and its variants are proposed such as Shape Autotuning Activation Function [22]. In contrast, two different and novel activation functions are proposed in a comparative paper, namely IpLU and AbsLU, which are smooth and piecewise activation functions, respectively [23]. There are novel two-part piecewise activation functions as well, such as MPELU [24]. Overall, there are various different alternatives to popular activation functions with advantages and drawbacks.

$$Softplus(z) = \ln(1 + e^z) \quad (2.6)$$

$$FELU(z) = \begin{cases} z & z > 0 \\ 2^{z/\ln 2} - 1 & z \leq 0. \end{cases} \quad (2.7)$$

$$SELU(z) = \begin{cases} s \cdot z & z > 0 \\ s \cdot a \cdot (e^z - 1) & z \leq 0. \end{cases} \quad (2.8)$$

$$MPELU(z) = \begin{cases} z & z > 0 \\ \alpha(e^{\beta z} - 1) & z \leq 0. \end{cases} \quad (2.9)$$

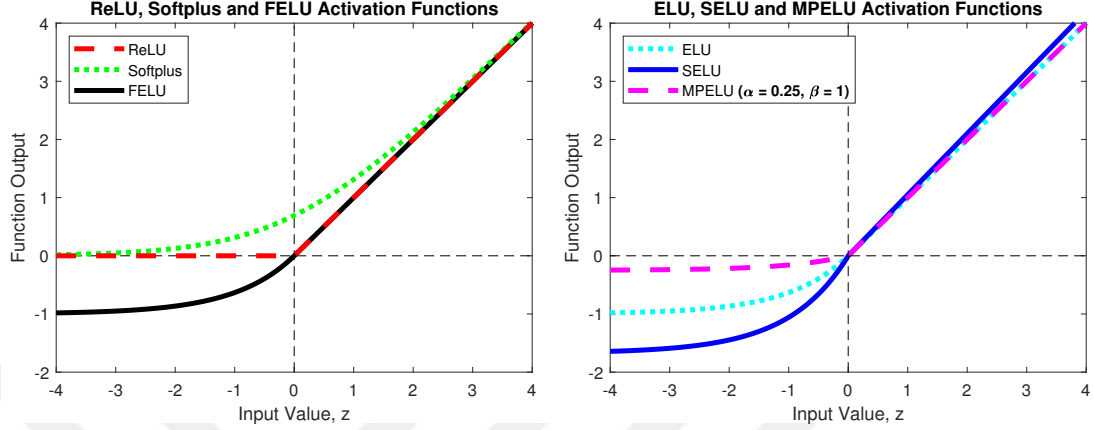


Figure 2.2: ReLU, Softplus, FELU, ELU, SELU and MPELU activation functions are given.

We note that there are many different types of activation functions, as mentioned above, and to consider all within a single framework is a difficult task. For this reason, we focus on rectifier-based activation functions in this paper, particularly on ReLU, ELU, and use their variants such as MPELU [24], SELU [19], FELU [25], and Softplus [12], since these ones are popular, suggested and mostly utilized in deep learning applications. They are presented here with figures and formulas. For SELU, $(s, a) = (1.05, 1.673)$ are used. For MPELU, the most successful parameter pair for CIFAR-10 is used, $(\alpha, \beta) = (0.25, 1)$ from the work [24] with implementation [26]. FELU is a reformulation of same function from ELU; uses $2^{f(z)}$ in the multiplication instead of $e^{f(z)}$ in favor of faster execution times. Note that there is a small difference between them, due to approximating $\ln 2$. This section introduced another activation function, named as “Extended Exponential Linear Unit” (DELU) as a replacement to ELU in the ReLU-family of neural activation functions. Based on the advantages of rectifier-based activation functions and building upon their beneficial additions, DELU appears to be a viable alternative in comparison to existing activation functions in the literature and may perform better in some applications through careful selection of its parameters. Specifically, the behaviors of ReLU and ELU activation functions around the zero point are seen as subject to improvement. In this manner, DELU is designed as a modification to these parts of existing activation functions, ReLU and ELU. From other motivations inherited to DELU, overcoming

the ‘dead zone’ problem of ReLU and seeking a better activation function than ELU are important to mention.

With these motivations, this section aims to make the following contributions:

- Introducing an activation function,
- Comparing it with rectifier-based activation functions,
- Presenting both classification and image segmentation examples,
- Using statistical methods in addition to tabulated results,
- Optimizing the parameters of this activation function.

2.1.3 Proposed DELU Activation Function

We propose the following activation function:

$$DELU(z) = \begin{cases} z & z > z_c \\ (e^{\alpha z} - 1)/\beta & z \leq z_c \end{cases} \quad (2.10)$$

where $\alpha \geq 0$, $\beta \geq 0$ and $z_c \geq 0$ are appropriate parameters to be determined. If we impose continuity on $DELU(\cdot)$ at $z = z_c$, the following equation should be satisfied:

$$\beta z_c = e^{\alpha z_c} - 1, z_c \geq 0. \quad (2.11)$$

Note that $z_c = 0$ is always a solution of (2.11) for any $\alpha \geq 0$ and $\beta \geq 0$. The relation of DELU with ELU and ReLU is clarified in the following remark.

Remark 1: Note that if we choose $\alpha = \beta = 1$ and $z_c = 0$, (2.10) becomes the same as ELU given by (2.5). On the other hand if we choose $z_c = 0$, for $\beta \rightarrow \infty$ it follows that $DELU(z) \rightarrow 0$ for $z \leq 0$. Hence ReLU given by (2.4)

could be considered as a limiting case of DELU given by (2.10). Note that a similar property also holds for Swish as $\beta \rightarrow \infty$. So, by choosing appropriate values of $\alpha \geq 0$ and $\beta \geq 0$, we expect to have similar properties that ELU or ReLU offers. $\square$

Note that, as explained above, $z_c = 0$ is always a solution of (2.11) for any $\alpha \geq 0$ and $\beta \geq 0$. However, for appropriate selection of the parameters α and β , we may have another solution of (2.11) with $z_c > 0$. This point is clarified as follows:

Fact 1: For any $\alpha \geq \beta > 0$, the only solution of (2.11) is $z_c = 0$. For $\beta > \alpha > 0$, (2.11) has a second solution $z_c > 0$. $\square$

Proof: Note that the left hand side of (2.11), $L(z) = \beta z$ is linear in z with $L(0) = 0$ and $L'(z) = \beta$, where a superscript ' here and in the following denotes derivative with respect to z . The right hand side of (2.11), $R(z) = e^{\alpha z} - 1$ is a strictly increasing function with $R(0) = 0$ and $R'(z) = \alpha e^z \geq \alpha$ for $z \geq 0$. For $\alpha \geq \beta > 0$, it is clear that the unique solution of (2.11) is $z_c = 0$. On the other hand, when $\beta > \alpha > 0$, since $L(z) < R(z)$ for sufficiently small $z > 0$. Since exponential functions grow faster than linear functions, there necessarily exists another (and unique) $z_c > 0$ such that (2.11) holds. $\square$

Remark 2: Note that $\alpha \geq 0$, $\beta \geq 0$ and $z_c \geq 0$ are considered as the parameters of DELU. Since we impose the continuity of (2.10) at $z = z_c$, (2.11) should be satisfied. Hence if we choose any two of these parameters, the remaining parameter could be found from (2.11). For example, if we choose $z_c = 1$ and $\alpha = 1$, then (2.11) yields $\beta = e - 1$. Hence then, the proposed model offers some flexibility for parameter tuning to obtain better results in neural network applications, which will be explored in the relevant section. For practicality, we should start with a selected specific parameter set for (2.10) in our work and use it for most of the experiments. Note that for $\alpha = 1$, $\beta = 2$, from (2.11) we obtain $z_c = 1.25643$. We note that this parameter set will be utilized throughout the thesis except the sections where DELU parameters are tuned. $\square$

The graphical representation of (2.10) with the parameter set given by (2.11) is given in the Figure 2.3 (a). DELU function given by (2.10) is differentiable except the point z_c , and its derivative is given in (2.12). Its graphical representation with the default parameters as indicated in Remark 2 is given in Figure 2.3 (b).

$$DELU'(z) = \begin{cases} 1 & z > z_c \\ (\frac{\alpha}{\beta})e^{\alpha z} & z \leq z_c. \end{cases} \quad (2.12)$$

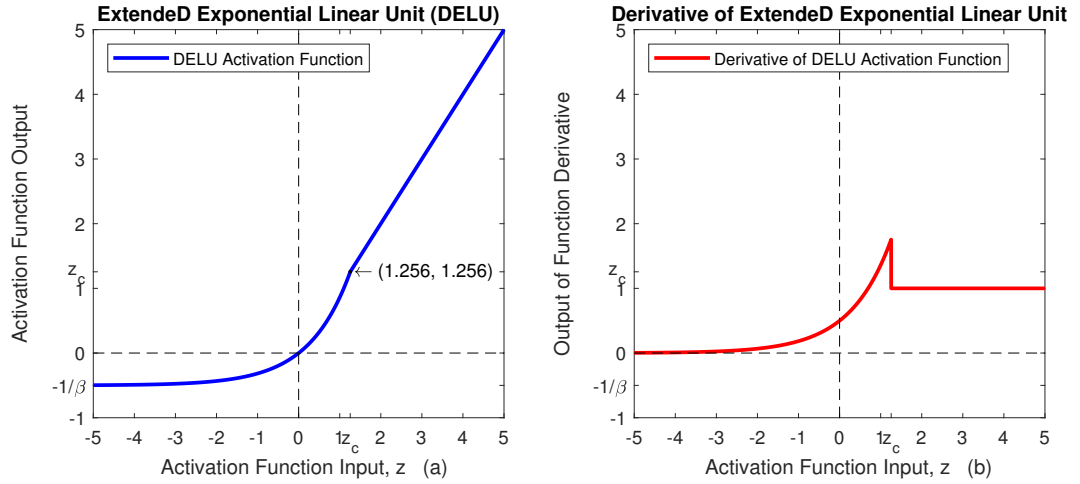


Figure 2.3: Graphs of DELU activation function (a) and its derivative (b) for $\alpha = 1, \beta = 2$.

Clearly derivative is discontinuous at $z = z_c$ unless $\alpha = \beta = 1$ and $z_c = 0$, which is the ELU case given by (2.5). Note that many variants of ReLU also have the same discontinuity problem. However, this does not pose a problem for applying the backpropagation algorithm for DELU, like those variants. In addition, the continuous values of the derivative of DELU around zero may affect the training process; this point will be explained in the next section.

2.1.4 Effects of DELU in Training Process

DELU activation function as given by (2.10) has certain differences as compared with ReLU and ELU given by (2.4) and (2.5), respectively. The main difference

is on the behavior of the activation functions around $z = 0$, especially for $z \geq 0$. While the slope of both ReLU and ELU for $z \geq 0$ is equal to 1, see (2.4) and (2.5), the slope of DELU activation function changes depending on z . To elaborate further, let us consider the DELU and its derivative as given by (2.10) and (2.12), respectively. For future reference, we will rewrite these equations with the parameters $\alpha = 1$, $\beta = 2$ and $z_c = 1.25643$, which will be our default parameter set to be used in the rest of the paper, as explained in Remark 2.

As can be seen from Figure 2.3, while $DELU'(0) < 1$, we have $DELU'(z) \geq 1$ in the range $z \in [z_x, z_c)$ for some $z_x > 0$. From (2.12) with parameters $\alpha = 1$ and $\beta = 2$, it easily follows that $z_x = \ln 2 = 0.69314$. We may consider the effect of this slope change on the early stages of the training process. Usually, the weights of the neural network are initialized randomly, and typically assigned by using a Gaussian probability distribution with zero mean. Since input of each neuron is a weighted sum of certain signals in the neural network, it is natural to expect the input of each neuron initially as a Gaussian random variable with zero mean. If the weights are updated according to standard backpropagation type minimization algorithm, then weight updates will be proportional to the derivative of the activation function around $z = 0$. At this point, behavior of DELU will be slightly different than the behavior of ReLU and ELU, since $DELU'(0) = 0.5 < 1$ and for others we have $ELU'(0) = ReLU'(0) = 1$. Hence we expect that, initially, weights will be updated slowly when DELU is used, which may prevent some neurons from early saturation, while in ReLU and ELU cases the weights will be updated by a larger amount which may force some neurons to early saturation. Intuitively, we may expect that in the early stages of training while usage of DELU offers a larger set of search directions for weight update, hence a better chance for loss minimization as compared to ReLU and ELU. The process alluded to above holds as long as $z \in [0, z_x)$. As the training evolves, we may expect that the inputs to neurons increase as well. When $z \in [z_x, z_c)$, we have $DELU'(z) > 1$, hence in this case we expect a faster weight update as compared to ReLU and ELU. Hence by combining these arguments, we expect that the usage of DELU offers a slow weight update at the early stages of the training but fast weight updates at later stages, which intuitively may result in

better loss minimization. This argument is the main reason for choosing the activation function of DELU as given by (2.10).

The arguments given above can also be analyzed by using suitably defined random variables, see e.g. [22]. To elaborate further, as alluded in the argument given above, assume that the initial neuron weights are assigned randomly and let the weights be initialized by using a Gaussian distribution or any continuous distribution centered around zero. Here, the only requirement needed is an independent and identical distribution for all neuron weight parameters, which is conventionally done in neural network initialization. In these cases, since the input z of any neuron is a weighted linear sum of such random selections, by using Central Limit Theorem [27] it is reasonable to assume that the input z of each neuron may also be associated with a random variable Z with a Gaussian distribution $N(0, \sigma^2)$, whose probability density function (PDF) is given as:

$$f_Z(z) = \frac{e^{-z^2/2\sigma^2}}{\sigma\sqrt{2\pi}} \quad (2.13)$$

As a result, if the activation function is $DELU(z)$, then output of the neuron is also a random variable, which is given as:

$$DELU(Z) = \begin{cases} Z & Z > 1.25643 \\ (e^Z - 1)/2 & Z \leq 1.25643. \end{cases} \quad (2.14)$$

The expected value and variance of the output $DELU(Z)$ can be given as:

$$E(DELU(Z)) = \int_{-\infty}^{\infty} DELU(Z) f_Z(z) dz \quad (2.15)$$

$$Var(DELU(Z)) = E(DELU(Z)^2) - (E(DELU(Z)))^2 \quad (2.16)$$

where the operators $E(\cdot)$ and $Var(\cdot)$ refers to expected value and variance of a

random variable, respectively. It follows from (2.10) and (2.12) that these values are well defined, see e.g. [22].

Using their PDFs, we calculated the mean values for only ReLU, ELU and DELU activation function choices, as they are the best ones in our CIFAR-10 comparison and for the sake of simplicity, and the summarizing figure of their mean values as a function of input standard deviation is given. Comparative results for standard deviation (which is defined as square-root of variance) and mean value (μ) functions for ReLU, ELU and DELU are given in Figure 2.4. At the beginning of the training process, DELU activation function outputs the least mean, while ReLU and ELU produce more. This corresponds to the slow allocation of neurons to drop the possibility of early over-fitting, heuristically. Smaller mean values at the beginning of training process are also preferred in well-known works such as Batch Normalization [28]. After the training speeds up and features are determined in the neural network as parameter values, the standard deviation of weights tends to increase, as pointed out in [28]. So, as the standard deviation of input Gaussian PDF increases, mean value of the output random variable increases as well. Here, DELU will give higher mean values, which will speed up the learning process by increasing derivative values in comparison to ReLU and ELU, and training with DELU might be better.

The arguments given above, as well as Figure 2.4 could be considered as an alternative confirmation which justify our earlier comments on the first slower then fast weight updates, given at the beginning of this section.

In addition to the argument given above, another comment can be made. Due to Gaussian nature and low variance values of the inputs of activation function, the change caused by DELU activation function affects more than it may seem at first glance. Many neurons are initially within the neighborhood of origin and in this vicinity, and derivative values of DELU are different than ReLU and ELU as illustrated in previous sections; making the learning slower at the beginning and faster at late stages of training - with higher derivative values as well. Keeping the relations of error gradient with activation function and derivative of activation function in mind, the aforementioned impact of activation function choice on

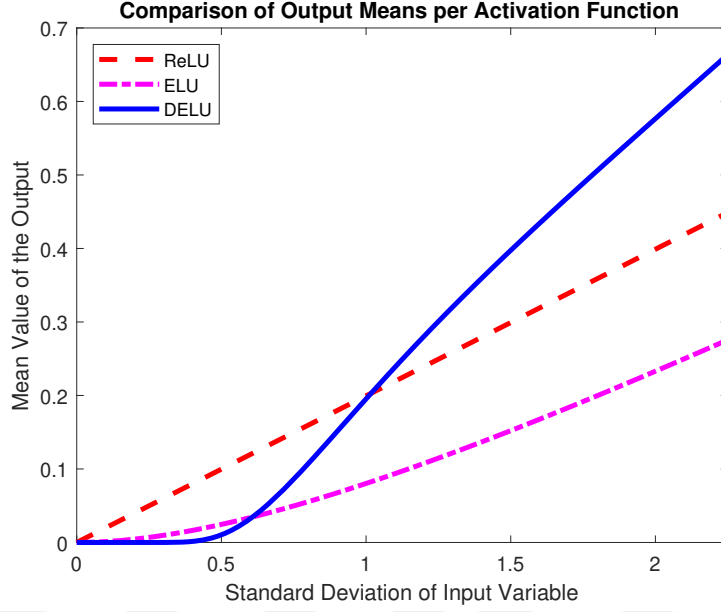


Figure 2.4: Obtained relation graph for ReLU, ELU and DELU activation functions.

the learning process may be non-negligible. Therefore, with explanation about PDF functions complete, it's important to mention that DELU brings a different behavior compared to ReLU and ELU.

2.1.5 Output Landscapes

While considering the arguments given above, it's also important to give experimental evidence about our statements on DELU. To this end, we add output landscape comparison. Output landscape is created by using a randomly initialized, multilayer neural network. We give 2 input values as x-y couples from the 2D plane, and get a single output value for each input couple, which is mapped into another 2D plane. It's centered at origin for all activation functions. Few examples are given in Figure 2.5 for ReLU, ELU and DELU.

In [22], output landscapes and their comparative smoothness are proposed as indicators for finding better neural network activations. Having a smoother output visual is described to lead to smoother loss surfaces during the training

process in [29]. For the purpose of strengthening our claim that DELU activation function offers a larger set of search directions for weight update, hence a better chance for loss minimization, we included the output landscape figures in Figure 2.5. Note that this comparison contains ReLU, ELU and DELU only.

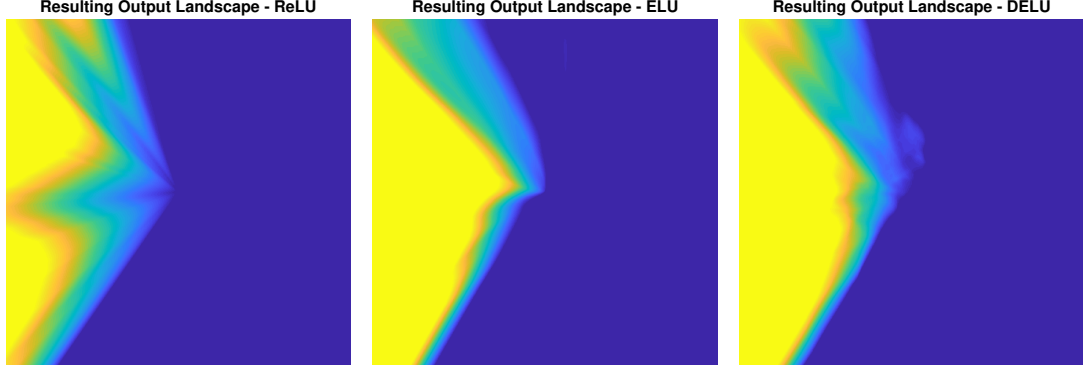


Figure 2.5: Obtained landscape graphs for three different activation functions.

As seen in the different output landscapes, different results are obtained for ReLU, ELU and DELU activation functions. Vertical and horizontal axes of outputs correspond to values of x-y input couples, while output values increase from navy blue to yellow color (darker to lighter). As discussed in [22], ReLU have sharper parts in comparison to ELU and DELU. On the other hand, DELU have the smoothest edge in the middle, while ELU has the least sharp for the top left corner in comparison to both ReLU and partially, DELU. The smoothness in the middle section of the output landscape of DELU may match with the heuristical explanation given above; not falling over sharp edges and allocating themselves to the features at early training stages. However, it isn't entirely possible to champion one activation function over the other with certainty, as the output landscapes are generated by using randomly initialized 5-layer feedforward neural networks. It's only a possible clue that DELU, with a smoother central area, can make a positive difference for learning process.

2.1.6 Results Obtained

In this section, results obtained in different tasks are presented in order. First, different activation functions are compared with each other for image segmentation tasks. Then, same activation functions are compared for classification tasks as well. Afterwards, different initialization methods are compared with a smaller amount of activation functions to observe their compatibility and comparative success rates.

2.1.6.1 Image Segmentation Task

To enhance our results, we decided to use image segmentation task in this part, in addition to image classification, to our work. Specifically, we used Oxford IIIT-Pets dataset which contains different breeds of dog and cats visuals [30]. Dataset contains colored images with varying sizes, and nearly equally splitted with 3680 training and 3669 test samples. Researchers included trimaps and annotations for determining the success rates, and provided their benchmark success for Intersection over Union (IoU) as 65% for the image segmentation task. An example is given below, alongside the formula of the IoU scores. Black pixels correspond to certain pixels of animal, white pixels stand for ambiguous region and gray pixels are certain background, in the ground truth images.

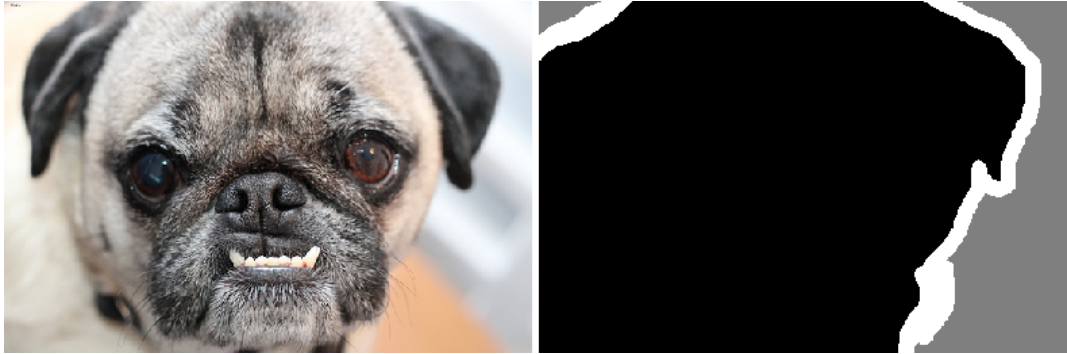


Figure 2.6: Oxford IIIT-Pets data sample and its truth map, constructed from dataset [30].

The network used in this task is different than ResNet. We used this dataset

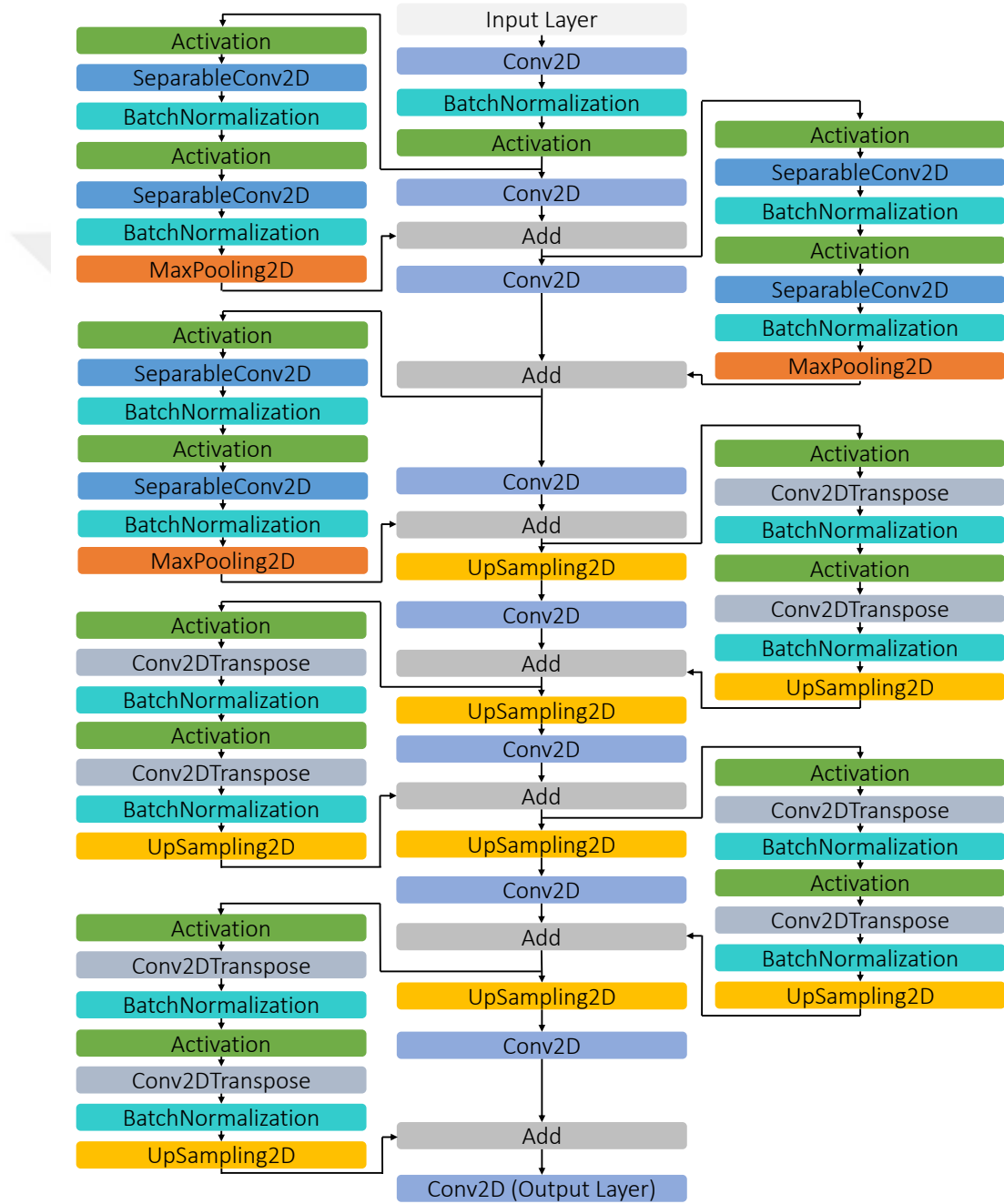


Figure 2.7: Representation of our U-net, takes a image input and outputs segmented image.

with a different network, taking a new structure from Keras library’s work “Image segmentation with a U-Net-like architecture” [31] which is given in the Figure 2.7. Here, network architecture is mainly kept same and official division of training and test samples are respected in our application. Sparse categorical cross-entropy is employed as the loss function with RMSprop learning algorithm. The output has the original size of the input, to check with the trimap segmentation ground truth for verification purposes. 41 samples from training set are put aside for validation purposes in the training process. Accuracy of the image segmentation, using overlapped black and white pixels is defined as:

$$\text{Accuracy (\%)} = \frac{\text{Overlapping Pixels of Truth Map and Our Guess}}{\text{Amount of Black \& White Pixels in Ground Truth}} \quad (2.17)$$

The success rates of trained image segmentation networks are calculated with a specific formula given in (2.17), for each sample in the dataset. In this way, ambiguous region for photographed animal is also added to the certain pixel region that differentiates the animal from the background.

The main change is the internal activation functions employed, each activation function is tried again separately for a fair comparison. In this manner, training for 300 epochs with a mini-batch size of 1 and 320x320 colored pixels input size, this time 20 independent and identical neural network results are obtained. For each activation function, minimum, mean and maximum success rates are presented in the Table 2.1 below. As DELU takes a slight lead in mean success rates, we can claim these results show that it is possible to claim that DELU can be better than others in the image segmentation networks.

Success rates are close to original authors’ benchmark of 65%, therefore these rates are still remarkable for our comparison reasons. It can be possible to test DELU with different setups and success rates, to see whether it is suitable for such tasks and configurations including different image processing tasks. This point can be investigated further.

Table 2.1: Oxford-IIIT Pets Dataset, 20 U-Net-Like networks.

Success Rate Percentages				
	Minimum	Mean	Maximum	Standard Deviation
ReLU	59.272%	61.251%	62.703%	0.898
ELU	58.920%	60.911%	62.611%	0.975
DELU	57.269%	61.299%	62.769%	1.146
SELU	58.655%	60.891%	62.189%	1.019
MPELU	58.944%	60.968%	63.226%	1.106
FELU	59.439%	61.172%	62.620%	0.820
Softplus	38.213%	59.442%	63.058%	5.210

2.1.6.2 Classification Tasks: CIFAR-10 and CIFAR-100 Networks

For comparing the activation functions, multiple datasets are classified with the well-known and reliable ResNet (Residual Networks) architecture with practical Keras library, based on Tensorflow implementation, while the only difference was the activation functions we used for a fair comparison. Our aim in this simulation analysis is to observe the effect of various activation functions for a given network structure, and to observe if one particular selection results in consistent improvement in performance. Specifically, for each activation function, 10 networks from different types of ResNet family are trained for CIFAR-10, CIFAR-100 and Fashion MNIST datasets [32], [33]. Test results are used to compare the activation function successes, both by comparison using success tables and statistical methods for the case of CIFAR datasets.

CIFAR-10 and CIFAR-100 datasets are popular datasets in classification benchmark tasks, and they are still in common use [32]. They consist of colored (RGB) 32x32 pixel images of different classes, 10 for CIFAR-10, and 100 for CIFAR-100 in fine label mode. By using results of these datasets on original ResNet architecture with different activation functions we will show the success rates and try to highlight the differences of results, using statistical methods.

Mainly, two types of the ResNet architecture as displayed in Figure 2.9 are used in this work: ResNet20-v1 for CIFAR-10 [34] and ResNet164-v2 for CIFAR-100,

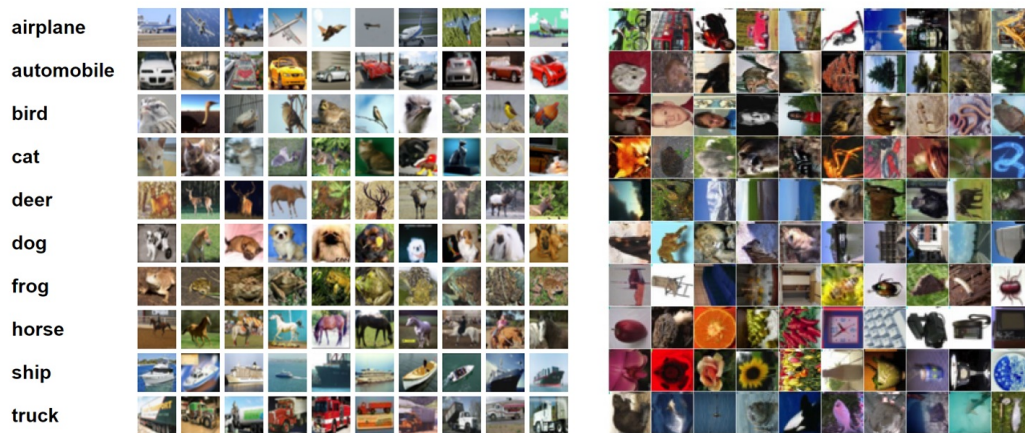


Figure 2.8: Example data samples: CIFAR-10 at left and CIFAR-100 at right [32].

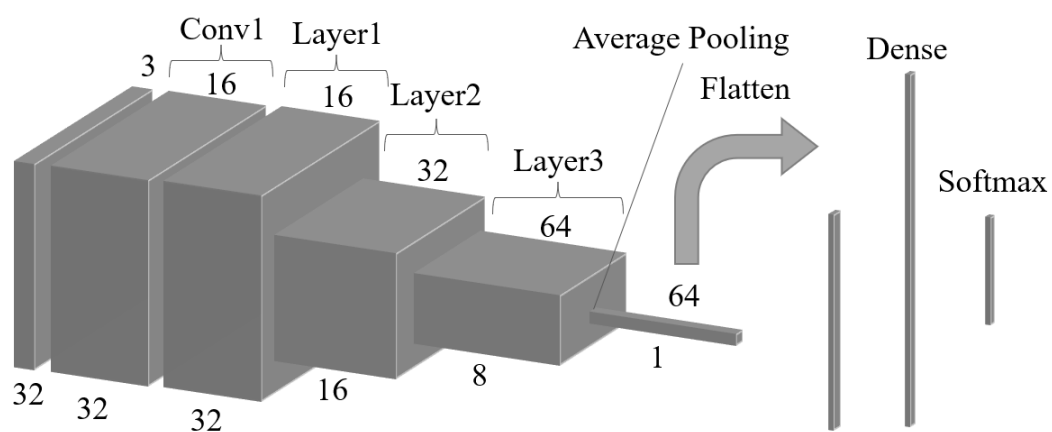


Figure 2.9: General ResNet structure is shown using blocks.

both are constructed using various blocks as illustrated. Both network architectures are thoroughly described in [35]. The default code architecture is taken from source [36] and it is designed for CIFAR-10 dataset using Keras library of Tensorflow, Python. By slightly increasing the input and output class sizes of the network in a regular way, we have adapted ResNet164-v2 for CIFAR-100 dataset (fine-mode). In total, 2 types of ResNet architectures are used for this article.

Default options for neural networks are used apart from the choice of activation functions (except Softmax outputs). Networks are initialized with He Normal [37] initialization, and kept the same for all activation functions for fairness - including SELU. Adam learning algorithm is employed for 200 epochs and categorical cross-entropy loss function is used while default data augmentation is kept. L2 regularization with parameter 0.001 is also employed for training with 45000 samples, where first 5000 are left for validation. Thus, it is possible to compare our results for different activation function choices.

Results of our simulations are given in Tables 2.2 and 2.3. Test results of 10 networks obtained in 10000 samples are compared for every activation function.

Table 2.2: CIFAR-10, 10 ResNet20-v1 networks.

	Success Rate Percentages			
	Minimum	Mean	Maximum	Standard Deviation
ReLU	89.67%	90.099%	90.40%	0.2372
ELU	90.11%	90.361%	90.77%	0.1976
DELU	90.07%	90.468%	90.74%	0.2492
SELU	88.37%	88.965%	89.34%	0.2915
MPELU	89.17%	89.603%	90.27%	0.3453
FELU	89.29%	89.566%	89.72%	0.1466
Softplus	88.64%	89.009%	89.55%	0.3028

As seen in the Tables 2.2 and 2.3, DELU leads in both sets of experiments for mean success rates. However, its standard deviation is not small, especially in CIFAR-100. Afterwards ReLU comes, then ELU is third in CIFAR-10. In the case of CIFAR-100, MPELU is second and ReLU is third, while DELU is in the lead with a slight margin. In CIFAR-10, ELU has the highest minimum and maximum success rates while in CIFAR-100, DELU has the highest maximum

Table 2.3: CIFAR-100, 10 ResNet164-v2 networks.

Success Rate Percentages				
	Minimum	Mean	Maximum	Standard Deviation
ReLU	73.11%	73.406%	73.81%	0.2223
ELU	72.06%	72.537%	72.92%	0.3335
DELU	72.82%	73.509%	74.19%	0.4157
SELU	71.38%	71.794%	72.23%	0.2476
MPELU	73.15%	73.485%	73.95%	0.2217
FELU	71.77%	72.347%	73.02%	0.1466
Softplus	72.00%	72.676%	73.47%	0.4158

and MPELU has the highest minimum success rate.

2.1.6.3 Classification Tasks: Fashion MNIST Dataset Networks

To strengthen our findings, we decided to adapt one of the ResNet networks for the Fashion MNIST dataset as explained in the previous section. This dataset is given in [33] as a replacement for MNIST which is already over-populated with 99.5% and higher success rate networks. The researchers designed this dataset as an alternative to MNIST, and because of that reason 28x28 pixel size and grayscale specialties are kept the same. Instead of digits, different accessories and clothes are used as 10 item classes of the dataset.



Figure 2.10: Fashion MNIST dataset samples taken from [33].

We used this dataset with a slight modification for not changing the ResNet20-v1 architecture. Instead of 28x28 pixel size, we added enough black pixels at top and left of the image to meet the 32x32 standard. For all color channels of RGB, we copied the content of grayscale into all there channels, making the sample size 32x32x3 - same with CIFAR samples. Thus, it became possible to use these samples with ResNet architecture coded for Keras library.

In Table 2.4, there are 10 networks trained for each activation function with 5000 validation samples, echoing the previous configuration. Success rates are obtained for each function as seen here, for 200 epochs training. For the results given in Table 2.5, we only use the network with highest validation success rate obtained during 200 epochs amongst 10 networks for each activation function.

Table 2.4: Fashion MNIST, 10 ResNet20-v1 networks at the end of 200 epochs.

	Success Rate Percentages		
	Minimum	Mean	Maximum
ReLU	93.42%	93.708%	93.87%
ELU	93.43%	93.766%	94.07%
DELU	93.41%	93.784%	94.10%
SELU	93.43%	93.681%	94.03%
FELU	93.57%	93.747%	93.84%
MPELU	93.66%	93.845%	94.17%
Softplus	93.35%	93.713%	94.05%

Table 2.5: Fashion MNIST, 10 ResNet20-v1 networks with highest validation success.

	Success Rate Percentages		
	Minimum	Mean	Maximum
ReLU	93.53%	93.694%	93.94%
ELU	93.44%	93.757%	94.02%
DELU	93.70%	93.903%	94.11%
SELU	93.37%	93.565%	93.74%
FELU	93.50%	93.749%	93.92%
MPELU	93.70%	93.870%	94.14%
Softplus	93.12%	93.739%	94.14%

As seen in Tables 2.4 - 2.5, we can see different results in each table separately. However, DELU is ahead with 93.903% mean success rate, in both of the tables. MPELU became the second best activation function in these success rates. Indeed, DELU also shares the lead in minimum success rates amongst both tables with MPELU, and this time MPELU leads in maximum success rates for all Fashion MNIST results obtained in this section. DELU's mean success rate, 93.903%, makes this result pretty good for this dataset in the existent literature -without any further configuration changes and in a network with 0.27 million parameters only- while extra data was not used in training as well.

For both network results, default data preprocessing and augmentation is left unchanged, and no additional fine-tuning is applied. In result, this additional experiment set and simple comparison between activation functions also showed that the success of DELU is not limited to CIFAR datasets, but instead it is an activation function that can perform better in different datasets as well.

2.1.6.4 Timing Benchmarks

Before concluding this section, it is important to mention the tradeoff between performance and time constraints for this new activation function in comparison to the other activation functions used in this work. DELU requires more time for training in contrast to the others, however this increase is not large. Around 15% increase is observed for the training time of U-net-like architectures used in this work with 300 epoch training settings, whilst the inference time for 3669 test samples is nearly the same with a slight increase as seen in the Table 2.6. This is not surprising because of the effect of the DELU in the backpropagation algorithm. On the other hand, increases obtained in the mean success rates are making this tradeoff worthy as the results imply.

Table 2.6: Mean times of classification and segmentation networks in seconds.

	F. MNIST		CIFAR-10		CIFAR-100		Oxford Pets	
	Train	Test	Train	Test	Train	Test	Train	Test
ReLU	5252.4	2.658	4348.6	2.404	48915.7	20.578	25864.1	11015.9
ELU	5214.8	2.654	4338.2	2.401	49100.6	20.425	25803.2	11026.0
DELU	6153.9	3.053	5127.7	2.931	63551.5	30.251	30062.4	11075.0
SELU	4159.2	3.132	3428.6	2.977	42639.9	34.535	28513.3	10944.3
MPELU	4168.6	2.760	3453.4	2.647	43977.5	25.124	28775.8	10816.7
FELU	4267.8	3.146	3572.5	2.985	47986.2	30.788	30018.7	10893.9
Softplus	5513.3	2.701	5466.2	2.410	52288.8	20.844	25681.1	11097.5

For networks with both CIFAR datasets and Fashion MNIST dataset, training and inference times are measured and inference times are tabulated as seen in Table 2.6. However, due to use of GPUs and CPUs under different loads in the training and inference of the networks in the Fashion MNIST and CIFAR datasets, it is not entirely fair to make a decision by using their timings only. However,

in general DELU needs more than 15% additional training and inference time in comparison to ReLU, ELU, and SELU; and DELU needs 2-3% less training time than MPELU for Fashion MNIST and CIFAR datasets in general. Compared to Softplus, DELU is around 10% slower in Fashion MNIST networks and faster around 7% in CIFAR-10 dataset specifically. In the light of those results, it can be said that the training time of DELU activation function slightly higher than the ReLU and ELU. However, since the training is an offline process and performed only once, but the resulting network structure afterwards utilized mainly for testing, one may argue that the inference time is more important. Indeed, many training methods actually aim at reducing inference time. Since the inference time of DELU is comparable to the other rectifier alternatives, a bit longer training could be considered as acceptable, considering the slight increase we have in the success rates.

2.1.7 Hypothesis Testing and Z-Score Analysis

We assume that the success rates given above are related to the associated neural networks in a random manner. This is a reasonable assumption, since during the training at least initial weights are generated randomly. Hence we assume that we can define random variables which generate these numbers, for each activation function. Let us call the random variable for success rates of ReLU as S_{ReLU} , which maps probabilities to the success rates from 0 to 100 continuously. This can be done for other activation functions as well, specifically S_{ELU} and S_{DELU} can be defined similarly. With these definitions, we can furthermore assume that all these random variables have an underlying Gaussian PDF, which will be shown in the next steps with minimal assumptions.

To assume that there is an underlying Gaussian PDF which explains the distributions of these rates for experiments, we need to use an approximation via well-known Central Limit Theorem (CLT) [38]. Also, to start from success rates obtained, we use a non-parametric kernel distribution [39]. To summarize, non-parametric kernel distribution is done by using n independent realizations from

the random variable and expressing the PDF of a single random variable from specialties of those n realizations. We use Gaussian kernels for our case.

The codes provided in Matlab documentaries are used for showing that this distribution evolves into Gaussian as expected by the CLT [40]. By using those assumptions and confirming that underlying PDF of random variable S_{ReLU} (like-wise S_{ELU} and S_{DELU}) is Gaussian, it will be possible to apply hypothesis testing as well; which will be explained in the rest of this section.

Although the results give us a feeling that newly designed DELU might be better than the existent ones, it is still not enough to make a solid statement. For this purpose, we will use hypothesis testing from detection and estimation theory [38]. To use this test easily, we assume inherent probability distribution of presented results as Gaussian. To this end, from Tables 2.2 - 2.3, we use mean and standard deviation values from test results of 10 networks and use CLT, while assuming their inherent probability distribution is Gaussian [38]. Next we attempt to show that this assumption is valid, by using the procedure as given below. For this purpose, we'll consider the results given in Table 2.2.

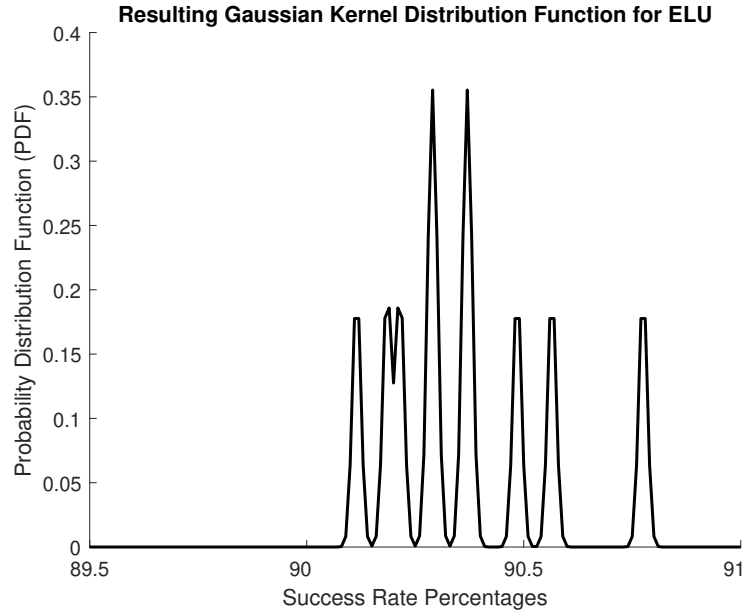


Figure 2.11: The non-parametric PDF for random variable S_{ELU} of success rates, which will be used according to process described here.

The process to obtain a Gaussian probability distribution is illustrated in Figure 2.12. Technically, we perform the operations outlined below in MATLAB. Note that changing the standard deviation values will only shrink or expand the resulting Gaussians, but not the Gaussian shape in the limit. Resulting shape is given in Figure 2.12 for ELU activation function success rate results. Looking at the resulting shapes, the CLT assumption of reaching to a Gaussian distribution after convolving the PDF with itself enough times holds.

- STEP 1: To build a probability function, we need to use obtained samples from success tables. We can take the results of ReLU activation function as an example. For each one of 10 samples, we place a Gaussian function: Centered at this sample value with a standard deviation of STD_{ReLU} , which is the standard deviation of the set of all 10 samples. This will result in a function similar to a kernel distribution.
- STEP 2: Repeat this procedure for ELU and DELU activation functions to find their corresponding kernel probability distributions. A typical application for ELU activation function is given in Figure 2.11.
- STEP 3: Then divide this resulting graph by 10 to make this function a valid PDF with its definite integral equal to 1, which was sum of 10 Gaussian PDFs and its definite integral value was 10.
- STEP 4: As this is a non-parametric representative PDF function of S_{ReLU} , we can use it to check whether it will converge to a single Gaussian PDF after convolutions. This is expected by the Central Limit Theorem (CLT), as long as the summation of independent and identically distributed random variables corresponds to the convolution of their PDFs [38]. To finish this step, simply convolve this PDF with itself 9 times, making 10 functions convolved for the CLT assumption check. Repeat this procedure for ELU and DELU activation functions.
- STEP 5: Normalize resulting Gaussian PDF for verification (use convolution with padding for preciseness).

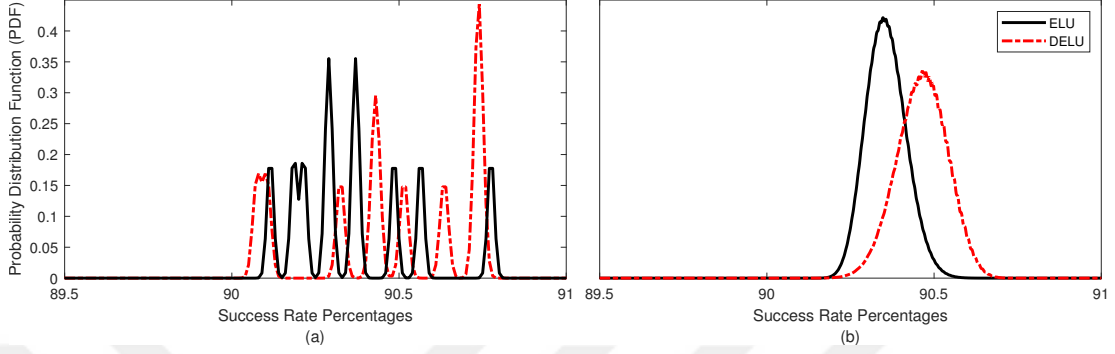


Figure 2.12: (a) contains two different Gaussian-kernel probability distribution functions for ELU and DELU. (b) is the result of 9 convolution operations for these two PDFs with themselves, for both ELU and DELU separately. It is obvious that the PDF evolves to Gaussian-like shape even with convolving only 10 PDFs, as expected by CLT.

A typical application of the steps outlined above for ELU and DELU activation functions by using the results given in Table 2.1 is given in Figure 2.12. After applying this routine for both activation function results, it is clear that the shape converges into a nearly ideal Gaussian PDF as expected by the Central Limit Theorem for ReLU, ELU and DELU activation functions, as illustrated in Figure 2.13. Thus, we have shown that the random variable, characterized by 10 success rates for each activation function, can be assumed to have a Gaussian PDF via Central Limit Theorem as illustrated above.

After we have the Gaussian shape for PDFs, we can also claim that mean and standard deviation values obtained from success rates can be used directly to place a Gaussian PDF for each one of activation functions. In our paper, this approach will be followed for simplicity and because of the closeness of mean values and evident Gaussian shapes obtained after convolution with Gaussian-kernel density PDFs. Therefore, standard deviation value is picked from the tables and divided by $\sqrt{10}$ (number of experiments in square-root), for having the closest PDFs to original outcomes of experiments from random variables as complying with CLT assumptions; making both graphs look similar.

At Figure 2.13 (a), we can see the visualization of Gaussians for the explained comparison procedure of the activation functions DELU, ELU and ReLU for

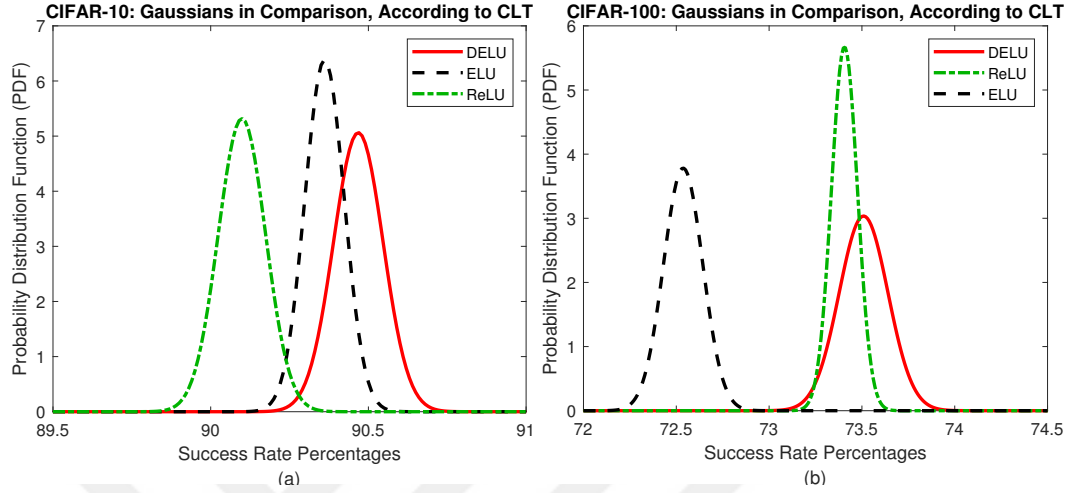


Figure 2.13: Gaussians related to ReLU, ELU and DELU based on Table 2.1 at left (a) and Table 2.2 at right (b). Figure 2.13 (a) and Figure 2.12 (b) are evidently similar, as CLT expected.

this case. Here we see that DELU has its mean success rate higher than others for CIFAR-10 and ResNet20-v1. However, the standard deviation of DELU results are higher than ReLU, as it seems by the wideness of its Gaussian in the Figure 2.13 (a). Similar to success rates obtained in Table 2.2, DELU maintains its lead with a slight margin over ELU with representative Gaussians as well. At Figure 2.13 (b), we can see aforementioned assumptions realized for DELU, ReLU and less successful one, ELU. Here we see DELU has its mean success rate higher than others for CIFAR-100 and ResNet164-v2, similar to Figure 2.13 (b) and Table 2.3. Confidence interval obtained from these figures are important, according to statistical significance value chosen for comparison reliability.

To investigate the statistical significance of these results, we resort to hypothesis testing methods which are typically utilized in decision making [41]. For this purpose, we consider two hypotheses relevant to our case. H_0 is the null hypothesis where it implies that replacing ReLU and ELU with DELU does not have a significant effect on increasing success, claiming that the change is statistically insignificant. H_1 , on the other hand, is the hypothesis which implies that replacing ELU or ReLU with DELU does have a significant effect on increasing success. In short, the hypotheses to be tested are given below.

- H0: Replacing ReLU (or ELU) with DELU does not increase success rates
- H1: Replacing ReLU (or ELU) with DELU does increase success rates

We first note that in the following analysis, we utilize the results given in Table 2.2 or 2.3. Secondly, we assume that the Gaussian distributions given in Figure 2.13 are representative of the corresponding experiments given in Tables 2.2 and 2.3. To determine which hypothesis is more likely for the experiments given by Table 2.2 or 2.3, we utilize a common hypothesis testing methodology which is frequently considered in Detection and Estimation Theory [42].

Making a mathematical claim about two different sets is not easy as a comparison between two scalar numbers. There can be outliers which may not represent the actual result, especially when a maximum or minimum operator is used to compare them. To model sets as random variables and claim results, especially when means are close, we need to introduce Z-score analysis to make a good hypothesis testing and reach a mathematical resolution. To summarize, Z-score can be named as a scalar that defines an area under a Gaussian PDF, which corresponds to probability value [42]. Definition of Z-score with multiple Gaussian curves in comparison is given later in this part in (2.19) with details.

Z-score analysis is used in different sources as transformations [43] and in the field of detection and estimation [44]. When it is possible to model the sets as Gaussian random variables, using Z-score analysis on their PDFs may help to decide between sets. To give an example, making a decision about which of two different Gaussian PDFs produce higher values can be a simple case of application. In our work, we construct our hypotheses to decide which activation function is better, defined by giving higher success rates. By using obtained success rates and their standard deviation values, PDFs are constructed and employed for Z-score analysis. In this way, our assertions will be more precise than reaching a decision by only using presented success tables.

First, let us define “Percentage of Confidence”, which is abbreviated to PoC for convenience. PoC is the strength of hypothesis H1 in comparison to H0. Hence,

if for example PoC is 90%, we can infer that H1 holds with 90% likelihood as compared to H0. To assess PoC, it's common in hypothesis testing applications to define a complementary variable; α , also called as Alpha, is given below:

$$\alpha = 1 - PoC. \quad (2.18)$$

If there is an underlying Gaussian PDF related with the hypothesis given above, α can be related to another quantity called Z-Score, and it's given below:

$$Z - Score = \frac{MEAN_{DELU} - MEAN_{ReLU}}{(STD_{ReLU})/\sqrt{SampleAmount}}. \quad (2.19)$$

Formula for calculating the Z-Score (2.19) is given in [45]. Note that (2.19) is utilized for the comparison of DELU and ReLU activation functions. Likewise for the comparison of DELU and ELU, the same formula will be utilized after replacing $MEAN_{ReLU}$ and STD_{ReLU} with $MEAN_{ELU}$ and STD_{ELU} , respectively. In this formula $MEAN_{DELU}$ is the mean value of success rates for DELU activation function-applied networks, likewise $MEAN_{ReLU}$ (similarly STD_{ELU}) is the value for ReLU (similarly ELU). STD_{ReLU} (similarly $MEAN_{ReLU}$) stands for the standard deviation of success rates obtained for ReLU (similarly ELU) activation function applied networks. $SampleAmount$ is the number of trained networks, which is 10 for classification networks. In general, we can see %90 PoC as a satisfactory value to make a decision on the comparison.

The relation between Z-Score and Alpha score, evaluated by upper tail test using Gaussian distributions can be given as in Table 2.7, for details see [45].

Table 2.7: Alpha vs. Z-score, for hypothesis tests.							
Upper-Tail Test							
Alpha	0.1	0.05	0.025	0.01	0.005	0.001	0.0001
Z-Score	1.282	1.645	1.96	2.326	2.576	3.09	3.719

Now let us consider the experiments given in Table 2.2. For a comparison between ELU and DELU, let us compute the Z-Score given by (2.19) for this pair.

Noting that $STD_{ELU} = 0.1976$, $MEAN_{DELU} = 90.468$ and $MEAN_{ELU} = 90.361$ and $SampleAmount = 10$, we compute a $Z - Score = 3.162$, which corresponds to $\alpha = 0.001$ from Table 2.7. Now, from (2.18) we can compute the value of PoC. We can interpret this result as follows: PoC for H1 hypothesis, which is replacing ELU with DELU does increase success rates, is 99.9%.

A follow-up comparison can be done with ReLU and DELU for this case. Noting that $STD_{ReLU} = 0.2372$, $MEAN_{DELU} = 90.468$, $MEAN_{ReLU} = 90.099$ and $SampleAmount = 10$, we compute a $Z - Score = 4.919$ which is higher than previous, 3.162. This result, as expected from graphs in Figure 2.13, verifies that DELU does increase success rates, this time with 99.99% PoC in comparison to ReLU. Picking the minimum, we can say that DELU have 99.9% PoC for increasing the success rate, in comparison to ReLU or ELU.

A similar comparison between ReLU and DELU for the CIFAR-100 results given in Table 2.3 yields $Z - Score = 1.465$, from values $STD_{ReLU} = 0.2223$, $MEAN_{DELU} = 73.509$, $MEAN_{ReLU} = 73.406$ and $SampleAmount = 10$. It corresponds to $\alpha = 0.1$ from Table 2.7. This can be interpreted as follows: PoC for H1 hypothesis (replacing ReLU with DELU increases success rates) is 90%.

Another follow-up comparison can be done with ELU and DELU for that case too. Note that $STD_{ELU} = 0.3335$, $MEAN_{DELU} = 73.509$, $MEAN_{ELU} = 72.537$ and $SampleAmount = 10$, we compute a $Z - Score = 9.2166$ which is much higher than previous one, 1.465. This result, as expected from graphs in Figure 2.13, verifies that DELU does increase success rates, this time with 99.99% PoC in comparison to ELU. Picking the minimum, we can say DELU have 90% PoC for increasing the success rate, compared to ReLU or ELU.

So long as only one-tail tests would be less strong, DELU must be treated with a similar understanding of hypothesis testing; where replacing DELU is considered as the statistically significant change or not. New hypotheses and correspondent negative Z-Scores given below reflect the complementing nature.

- H0: Replacing ReLU (or ELU) with DELU does not increase success rates

- H1: Replacing ReLU (or ELU) with DELU does increase success rates

$$Neg.Z - Score = \frac{MEAN_{ReLU} - MEAN_{DELU}}{(STD_{DELU})/\sqrt{SampleAmount}} \quad (2.20)$$

Similarly, Z-Score formula changes for these hypotheses as above. Note that (2.20) is used for comparison of DELU and ReLU. Likewise, for comparison with ELU, same formula will be used after replacing $MEAN_{ReLU}$ with $MEAN_{ELU}$.

The relation between Neg.Z-Score and Alpha score, evaluated by upper tail test using Gaussian distributions can be given as in Table 2.8, for details see [45]. This table will be used for adding ‘other tail comparison’ into one-tail comparisons. In this way we’ll have two-tail comparisons for CIFAR networks.

Table 2.8: Alpha vs. Z-score, for hypothesis tests.

Lower-Tail Test							
Alpha	0.1	0.05	0.025	0.01	0.005	0.001	0.0001
Z-Score	-1.282	-1.645	-1.96	-2.326	-2.576	-3.09	-3.719

Now let us consider the experiments given in Table 2.2 again. For a comparison between ELU and DELU, let us compute the Z-Score given by (2.20) for this pair. Note that $STD_{DELU} = 0.2492$, $MEAN_{DELU} = 90.468$, $MEAN_{ELU} = 90.361$ and $SampleAmount = 10$, we compute a $Z - Score = -1.358$, which corresponds to $\alpha = 0.1$ from Table 2.8. As before, by using (2.20) we can compute the PoC. We may interpret this result as follows: PoC for H0 hypothesis, which is replacing ELU with DELU does not increase success rates, is 90%.

A follow-up comparison can be done with ReLU and DELU for this case too. Noting that $STD_{DELU} = 0.2492$, $MEAN_{DELU} = 90.468$, $MEAN_{ReLU} = 90.099$ and $SampleAmount = 10$, we compute a $Z - Score = -4.683$ which is magnitude-wise higher than previous one, -1.358 . This verifies that DELU does increase success rates, this time with 99.99% PoC in comparison to ReLU. Picking the minimum, we can say that DELU have 90% PoC for increasing the success rate, in comparison to ReLU or ELU for two-tail comparison.

Applying this logic for the CIFAR-100 results obtained in Table 2.3 as done before gives us $Z - Score = -0.784$, with values $STD_{DELU} = 0.4157$, $MEAN_{DELU} = 73.509$, $MEAN_{ReLU} = 73.406$ and $SampleAmount = 10$. This corresponds to $\alpha = 0.25$ and thus, at least 75% as PoC for H0 hypothesis, which is replacing ReLU with DELU does not increase success rates. Therefore, the lowest of two values is picked and we can finally, statistically infer that DELU became more successful than ReLU with a PoC value of 75% at least, for the experiments conducted with CIFAR-10 and CIFAR-100 datasets.

Follow-up comparison for that case is comparing ELU and DELU for Table 2.3 results, which yields $Z - Score = -7.394$, with values $STD_{DELU} = 0.4157$, $MEAN_{DELU} = 73.509$, $MEAN_{ELU} = 72.537$ and $SampleAmount = 10$. It corresponds to $\alpha = 0.0001$ from Table 2.8, sets PoC as 99.999% and picking the minimum value means that DELU is more successful than ReLU or ELU with a PoC value of 75% at least, for experiments with CIFAR datasets.

In addition to the success tables, these statistical analyses make it clear that DELU is a promising activation function and a candidate for common use, although it has slightly slower application time in training. Still, it yields higher performance while keeping its training and test timings close to others.

2.1.7.1 Optimizing the Parameters of DELU

Standardized DELU uses fixed α and β , respectively 1 and 2, in its final design. In this part, we explored different DELU alternatives. In the first stage, we varied both values within chosen limits. z_c is determined automatically according to choice of α and β which are from intervals $\alpha \in [0.5, 2]$ and $\beta \in [1, 3]$, with $\beta > \alpha$ condition. We varied α and β by incrementing them by 0.5 for both values within these intervals. By this framework, we obtained Fashion MNIST and CIFAR-10 mean success rates for each DELU alternative with (α, β) pairs, with 10 trained networks which use these alternatives.

For having $z_c > 0$, we must have the relation $\beta > \alpha$ for every positive (α, β)

pair. With this condition, we chose 11 different (α, β) pairs. In the end of our search, pairs with $\alpha = 0.5$ and $\alpha = 2$ have performed less successful than $\alpha = 1$ and $\alpha = 1.5$, regardless of the β value chosen. So, we have continued with the second stage of our DELU optimization with $\alpha = 1$ and $\alpha = 1.5$.

Table 2.9: Fashion MNIST and CIFAR-10 mean success rates of different DELU alternatives.

	F.MNIST Results			CIFAR-10 Results	
	$\alpha = 1$	$\alpha = 1.5$		$\alpha = 1$	$\alpha = 1.5$
$\beta = 1.75$	93.745%	93.688%	$\beta = 1.75$	89.725%	89.696%
$\beta = 1.875$	93.824%	93.763%	$\beta = 1.875$	89.796%	89.548%
$\beta = 2$	93.784%	93.810%	$\beta = 2$	90.468%	89.881%
$\beta = 2.125$	93.855%	93.857%	$\beta = 2.125$	89.832%	89.861%
$\beta = 2.25$	93.807%	93.764%	$\beta = 2.25$	89.781%	89.844%
$\beta = 2.375$	93.866%	93.715%	$\beta = 2.375$	89.726%	89.899%
$\beta = 2.5$	93.849%	93.878%	$\beta = 2.5$	89.824%	90.030%
$\beta = 2.625$	93.776%	93.746%	$\beta = 2.625$	89.723%	89.966%
$\beta = 2.75$	93.911%	93.760%	$\beta = 2.75$	89.689%	89.897%
$\beta = 2.875$	93.801%	93.875%	$\beta = 2.875$	89.656%	89.650%
$\beta = 3$	93.868%	93.922%	$\beta = 3$	89.670%	90.048%
$\beta = 3.125$	93.840%	93.802%	$\beta = 3.125$	89.653%	89.868%
$\beta = 3.25$	93.928%	93.806%	$\beta = 3.25$	89.575%	89.760%
$\beta = 4$	93.763%	93.812%	$\beta = 4$	89.486%	89.740%

Similar to the approach followed in the design of MPELU activation function [24], we came up with different (α, β) pairs and selected a good couple amongst the alternatives in the first stage. Still, it is possible to go deeper with a finer search. For this purpose, we present Table 2.9 in the second stage. As seen in this table, limiting α values to two best ones and varying β value in a finer search gave in a better result than original DELU activation function for the Fashion MNIST dataset; with $(\alpha, \beta) = (1, 3.25)$ pair, 93.928% and 89.575% mean successes are obtained in Fashion MNIST and CIFAR-10, respectively. However, original parameter set of $(\alpha, \beta) = (1, 2)$ is still better than the others for the CIFAR-10 dataset, by a margin around 0.4%. In the result, we came up with a method to search (α, β) values to find DELU alternatives, and ended up with informative results on Fashion MNIST and CIFAR-10 in two stages. In this way, different choices of α and β values are also justified. Indeed, it is possible to go even deeper for search of (α, β) with other methods, but it's left as a future work.

In conclusion, we proposed a novel activation function in this section: DELU, which can be considered as a generalization of classical activation functions ReLU and ELU. To motivate the proposed activation function, we first give some heuristic and intuitive reasons which possibly may result in relatively higher success rates when utilized in a given neural network structure, as compared to its counterparts. In comparison to ReLU and ELU, usage of DELU increased the mean success rate 0.075% to 0.146% for Fashion MNIST dataset results, while the increase was 0.107% and 0.103% for CIFAR-10 and CIFAR-100 datasets respectively. As results given in Tables 2.1, 2.3, 2.4 and 2.5 illustrate, DELU also gives the maximum success rate amongst those results obtained for aforementioned three activation functions in comparison with higher margins. The minimum success rates obtained for these experiments also pretty close to each other as seen in Tables 2.1, 2.2, 2.3, 2.4, 2.5 and 2.7. To sum up, image segmentation and classification examples performed in this section have suggested a performance increase by replacing ReLU or ELU with proposed activation function, DELU.

We performed these simulations with 6 activation functions and the proposed activation function DELU, and the results of these simulations are discussed in detail. CIFAR-10, CIFAR-100 and Fashion MNIST datasets used for training the network are introduced, network specialties for ResNet architectures are explained and training conditions for these settings are given exclusively. Alongside those, comparison of network results are also presented for all seven activation functions exclusively. In addition to standard presentation of obtained test results, our work contains a statistical significance test based on CIFAR dataset results to compare ReLU, ELU and DELU activation function. With one-tail tests on CIFAR datasets, DELU activation function is better than ReLU and ELU with more than 90% Percentage of Confidence. In addition to this, with two-tail tests, it is still possible to confidently show that DELU performs better with at least 75% PoC using hypothesis testing methods on all CIFAR datasets used in this work. Also, ResNet networks are tried on Fashion MNIST dataset with same activation function set, and results are given for verification purposes in addition to CIFAR results. Also, we made an image segmentation example with U-net-like networks to separate cats and dogs from the background in Oxford IIIT-Pets

Dataset, with all activation functions are again compared by their minimum, mean and maximum success rates out of 20 independent runs for 300 epochs training configuration. Here, DELU scored the highest mean success rate. On the mean success rates obtained, the highest is DELU with 61.299%, while ReLU follows it with a 61.251%, and MPELU is third with 60.968% in this experiment. These results indicate that, although we cannot claim with certainty that the usage of DELU or ReLU yields better results for the image segmentation problem, success rates are comparable with the results obtained for the structure considered in our experiments. It may be possible that for different architectures, using DELU may yield better or worse performance. However, this point needs to be carefully investigated; but at least, it can be claimed that DELU cannot be declared worse than ReLU with these results.

The main contributions of this section can be listed as follows. First, we introduced a new activation function, DELU, in detail. Second, DELU is compared with different activation functions for various datasets, using well-known ResNet architectures for certain classification tasks. Third, statistical tests are used to show DELU’s success in comparison to others by hypothesis testing for CIFAR networks. Fourth, we made an image segmentation task with DELU as well. The results of these tests indicate that the utilization of the DELU may improve the success rates as compared to its counterparts from a statistical and empirical point of view.

2.2 Results for Different Initialization Methods with Different Activation Functions

Initialization methods are inseparable parts of neural network training processes. To divide into groups, there are two main types of initialization methods in the literature: Independent and dependent initialization methods. Independent initialization methods consist of single probability distribution function (PDF) generating weight parameters from this distribution only, independent of layer

specialties. However, they need tuning by changing the limit or variance, to reach the optimal success rate for the network.

Dependent initialization methods are from a generic initialization function (for example, Gaussian), however every neural network layer scale or change the probability distribution function according the neuron amount at each layer. In this way, they aim to specialize for each layer by avoiding large parameter choices for layers with small neuron amounts etc. The PDF used to generate network parameters is scaled or truncated for this purpose. There are two popular types of initialization methods: Glorot (Xavier) and He initialization schemes.

2.2.1 Proposed Truncated Towers Initialization

As an alternative to the existing initialization techniques which are frequently used in the literature, namely Uniform and Gaussian, we propose the following initialization distribution to have a fair and better comparison: Truncated Towers initialization. It specifically aims on lowering the amount of small gradient values and possible vanishing, while keeping the random values limited in a range to avoid the exploding gradient problem. Illustrations of used PDFs, histogram and figures are given in next pages. For all figures in this section, x-axis corresponds to values of generated parameters.

The functions of Laplacian, Gaussian and Candidate probability distributions are given respectively as $f_L(x)$, $f_G(x)$, $f_C(x)$, in below equations.

$$f_L(x) = \exp(-|x|)/2 \quad (2.21)$$

$$f_G(x) = \exp(\frac{-x^2}{2})/\sqrt{(2\pi)} \quad (2.22)$$

$$f_C(x) = \exp(\frac{-|x^3|}{6})/3.245302 \quad (2.23)$$

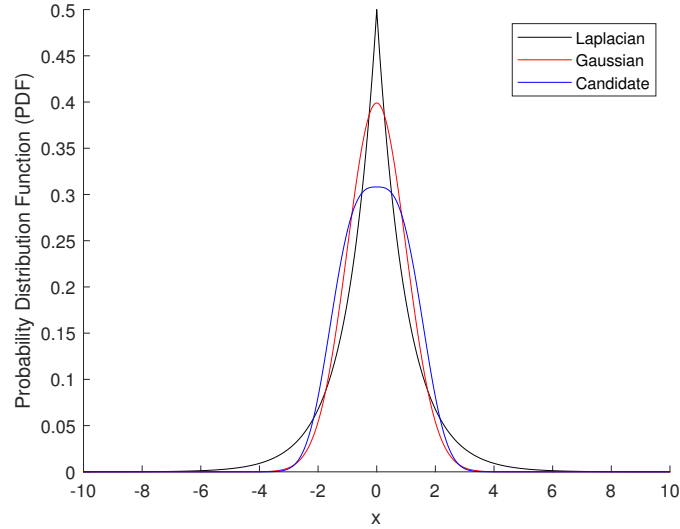


Figure 2.14: Candidate initialization method is displayed as a figure in comparison to Gaussian and Laplacian functions.

Candidate probability distribution function will be used in our proposal. To create an independent initialization method, we can truncate the function from its left and right parts, however in this case our proposal is to truncate the function from the neighborhood of zero. This is aimed to lower the amount of small weight parameters, which may slow down the learning process with the gradient updates which are calculated by multiplying with weights themselves. With this notion, below function is named as “Truncated Towers Distribution”.

$$f_{TT}(x) = \begin{cases} 0, & \text{if } \frac{3.245302}{160} = 0.203 \geq |x| \\ \exp(\frac{-|x^3|}{6})/3.20474, & \text{else} \end{cases} \quad (2.24)$$

Resulted distribution will be used in a standalone initialization method (with its own standard deviation is set around 0.05, like the Gaussian initialization), and also will be used in a truncated distribution as well, which prunes the probability distribution function similar to the He-truncation discussed earlier. In this manner, comparison of the new initialization method with both dependent and independent methods will be completed. These initialization methods are used with ReLU, ELU and DELU activation functions which will be discussed later.

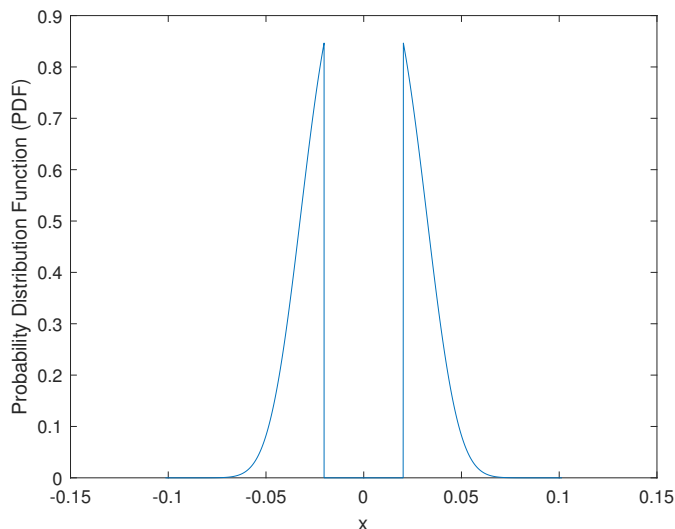


Figure 2.15: The PDF of chosen Truncated Towers is displayed.

However, it is important to note the specialty in common before moving on: Being designed to lower the amount of weight parameters to be initialized at the neighborhood of zero. Their strength of forcing the parameters though, vary between different methods. In this manner, a good start to training is set as a target and successful backpropagation steps are aimed with non-zero gradient-weight multiplications and resulting weight updates.

In this section, different initializations are paired with different activation functions to form neural networks, which are trained on Fashion MNIST and CIFAR-10 datasets afterwards. Using 200 epochs of training with Adam optimizer and categorical cross-entropy loss function, independent initialization methods are compared in 2.10 and 2.11. For dependent initialization methods, only Fashion MNIST networks are generated whose results are shown in 2.12. For each success rate, 10 neural networks with same configuration is trained and their test results which are obtained after the training is used for the calculation of minimum, mean and maximum success rates displayed in the tables.

As seen in Table 2.10, the combination with highest success rate on the Fashion MNIST dataset is Truncated Towers initialization methods with DELU activation function, in comparison to the other ones. ELU activation function with

Table 2.10: Fashion MNIST Success Rates (%)

	Activation Functions & Initialization Methods		
Minimum Successes	Uniform (Fixed)	Gaussian (Fixed)	Truncated Towers (Fixed)
ReLU	93.530	93.420	93.440
ELU	93.410	93.320	93.550
DELU	93.540	93.320	93.580
Mean Successes	Uniform (Fixed)	Gaussian (Fixed)	Truncated Towers (Fixed)
ReLU	93.706	93.719	93.652
ELU	93.630	93.523	93.789
DELU	93.694	93.660	93.810
Maximum Successes	Uniform (Fixed)	Gaussian (Fixed)	Truncated Towers (Fixed)
ReLU	93.850	93.860	94.040
ELU	93.850	93.810	94.030
DELU	93.800	94.000	94.100

Truncated Towers distribution is also interestingly successful in this table, where success rates are obtained with calculating minimum, mean and maximum of the success rates resulting from 10 neural networks for each configuration.

Table 2.11: CIFAR-10 Success Rates (%)

	Activation Functions & Initialization Methods		
Minimum Successes	Uniform (Fixed)	Gaussian (Fixed)	Truncated Towers (Fixed)
ReLU	89.560	89.760	89.220
ELU	89.270	89.450	89.870
DELU	89.710	89.760	89.800
Mean Successes	Uniform (Fixed)	Gaussian (Fixed)	Truncated Towers (Fixed)
ReLU	89.916	90.025	89.821
ELU	89.958	89.970	90.232
DELU	90.078	90.396	90.278
Maximum Successes	Uniform (Fixed)	Gaussian (Fixed)	Truncated Towers (Fixed)
ReLU	90.400	90.370	90.060
ELU	90.230	90.440	90.840
DELU	90.350	90.660	90.590

As seen in Table 2.11, DELU and ELU is more successful amongst activation function while Gaussian and Truncated Towers is more successful amongst independent initialization methods, considering minimum, mean and maximum success rates on CIFAR-10 dataset results.

Table 2.12: Fashion MNIST Success Rates (%)

	Activation Functions & Initialization Methods				
Minimum Successes	Uniform (Glorot)	Gaussian (Glorot)	Uniform (He)	Gaussian (He)	Truncated Towers (He-like)
ReLU	93.516	93.240	93.420	93.420	93.440
ELU	93.640	93.550	93.680	93.430	93.460
DELU	93.570	93.570	93.760	93.410	93.560
Mean Successes	Uniform (Glorot)	Gaussian (Glorot)	Uniform (He)	Gaussian (He)	Truncated Towers (He-like)
ReLU	93.752	93.740	93.622	93.708	93.728
ELU	93.913	93.879	93.883	93.766	93.698
DELU	93.942	93.762	93.889	93.783	93.796
Maximum Successes	Uniform (Glorot)	Gaussian (Glorot)	Uniform (He)	Gaussian (He)	Truncated Towers (He-like)
ReLU	94.000	94.110	93.850	93.870	93.950
ELU	94.040	94.270	94.100	94.070	93.960
DELU	94.130	94.000	94.060	94.100	94.210

In the Table 2.12, dependent initialization methods are compared by using same activation functions. However, this time Truncated Towers initialization with a He-like truncation method is less successful. Still, DELU is in the lead for minimum and mean success rates while in maximum, ELU is higher than the others. In terms of initialization methods, Uniform initialization with Glorot truncation is best in mean, Uniform initialization with He truncation is best in minimum, and Gaussian initialization with Glorot truncation is best in maximum success rates as seen in the table of Fashion MNIST dataset results.

In conclusion, the results for independent initialization methods with activation functions show the potential of Truncated Towers initialization. However, as seen in the Table 2.12, for the dependent initialization methods, work is needed to come up with a better method for more successful network weight initializations. Also, the DELU activation function repeats its success in this section with different initialization methods as well, which shows the compatibility of the DELU and its insensitivity to the initialization method in comparison to ReLU and ELU.

2.3 Other Methods of the Neural Networks

There are other methods and aspects used in the network architectures which are used in the thesis. These include optimizers, loss functions and network layers. In this part, they are briefly explained and some comparative results are presented.

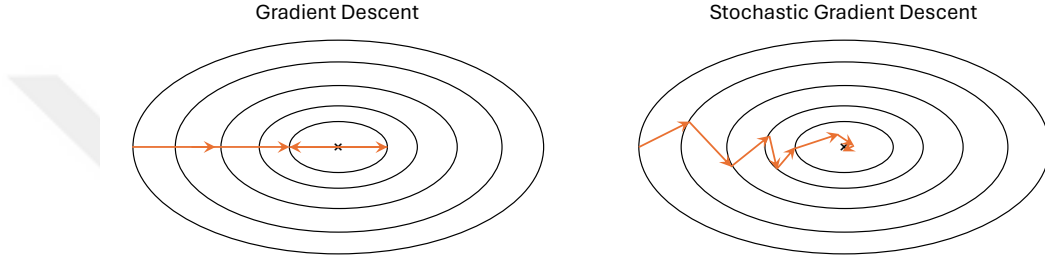


Figure 2.16: A comparison of gradient descent and stochastic gradient descent.

The optimizers (learning algorithms) used in this thesis are originated from stochastic gradient descent, such as RMSprop and other variants. Amongst them, specifically Adam and AdamW optimizers are chosen respectively for this and next chapter of the thesis. Their formulas are given, (2.25) for Adam [46] and (2.26) for AdamW [47]. They are compared with other ones for Fashion MNIST and CIFAR-10 classification networks, and their results are presented next.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, w_t = w_{t-1} - \frac{\eta}{\sqrt{v_{t-1}} + \epsilon} m_{t-1} \quad (2.25)$$

$$w_t = w_{t-1} - \eta \frac{\beta_1 m_{t-1} + (1 - \beta_1)(\nabla f_t + L_2 w_{t-1})}{\sqrt{v_t} + \epsilon} \quad (2.26)$$

In the experiments, 32 trials are done for each optimizer to increase the statistical significance, with 200 epochs of training in ResNet20-v1 network architectures with Glorot Uniform weight initialization, DELU activation function and Categorical Cross-Entropy loss function. For Fashion MNIST results given in Table 2.13, Adam leads all optimizers with a slight margin for minimum, mean and maximum success rates.

Table 2.13: Optimizers and their respective success rates for Fashion MNIST.

	Success Rate Percentages			
	Minimum	Mean	Maximum	Standard Deviation
RMSprop	93.520%	93.834%	94.080%	0.123
Adam	93.550%	93.861%	94.140%	0.150
Nadam	93.460%	93.772%	94.120%	0.136
AdaMax	93.180%	93.428%	93.840%	0.155
AdamW	93.220%	93.740%	94.020%	0.190

Table 2.14: Optimizers and their respective success rates for CIFAR-10 dataset.

	Success Rate Percentages			
	Minimum	Mean	Maximum	Standard Deviation
RMSprop	90.050%	90.567%	90.960%	0.182
Adam	90.090%	90.622%	91.050%	0.212
Nadam	90.260%	90.654%	91.140%	0.212
AdaMax	88.260%	88.724%	89.190%	0.254
AdamW	90.080%	90.615%	91.060%	0.234

In Table 2.14, Adam and AdamW are slightly less successful than Nadam at mean success results for Fashion MNIST results. However, combined with the minimum, mean and maximum results from the previous table, they confirm the choice of using Adam and AdamW optimizer with this short analysis.

For the case of loss functions, Mean Squared Error (MSE) as formulized in (2.27), and Categorical Cross-Entropy (CCE) [7] as formulized in (2.28) are used throughout the thesis. ‘Categorical’ stands specifically for the classification tasks, and if this loss function is used in a different configuration, changes are needed accordingly. Their comparison with other counterparts are presented as tables in the next page. Similar to optimizer comparison, ResNet20-v1 with Glorot Uniform initialization, DELU activation function and Adam optimizer is used. As before, 32 samples are trained for 200 epochs for each loss function.

$$\mathcal{L}_{MSE} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 \quad (2.27)$$

$$\mathcal{L}_{CCE} = -\frac{1}{n} \sum_{i=1}^n [y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})] \quad (2.28)$$

Table 2.15: Loss functions and their respective success rates for Fashion MNIST.

	Success Rate Percentages			
	Minimum	Mean	Maximum	Standard Dev.
Mean Squared Error	93.100%	93.483%	93.870%	0.159
Poisson	93.520%	93.786%	94.140%	0.153
KL Divergence	93.530%	93.868%	94.050%	0.129
Huber	92.910%	93.217%	93.460%	0.129
Categorical Cross-Entropy (CCE)	93.550%	93.861%	94.140%	0.150

Table 2.16: Loss functions and their respective success rates for CIFAR-10.

	Success Rate Percentages			
	Minimum	Mean	Maximum	Standard Dev.
Mean Squared Error	89.230%	89.694%	90.150%	0.245
Poisson	90.380%	90.837%	91.290%	0.221
KL Divergence	90.230%	90.684%	91.170%	0.242
Huber	88.060%	88.643%	89.130%	0.297
Categorical Cross-Entropy (CCE)	90.090%	90.622%	91.050%	0.212

In Table 2.15, KL Divergence leads, Categorical Cross-Entropy (CCE) is second. Poisson is a close third. For the maximum successes, CCE and Poisson is tied with 94.140% while in minimums, CCE has a slight lead with a 93.550% over KL Divergence and Poisson. In Table 2.16, Poisson loss takes the lead in mean values, KL Divergence and CCE follows. Still, although differences are more than Fashion MNIST, doubling of the standard deviations imply similarity. For minimums and maximums, Poisson leads other loss functions this time. Three loss functions lead the table overall: CCE, Poisson and KL Divergence. This situation justifies our usage of CCE as a loss function for classification tasks in the thesis.

As of network layers used in the thesis, convolutional layers are frequently explained in detail and used in this chapter. However, feedforward and recurrent layers are also present in the thesis and they should be briefly explained as well.

Feedforward layers are designed with an inspiration taken from neuron cells. Their primitive ancestor is the perceptron [2], then multi-layer perceptron comes in the literature. Perceptron is an artificial neuron cell with zero or one output. A modern feedforward neural network is as illustrated in Figure 2.17. Inputs are fed into neurons, multiplied by the weights and summed up. Universal Approximation Theory says with multiple layers, it is able to approximate any function [49]. Almost all network architectures in this work have feedforward layers.

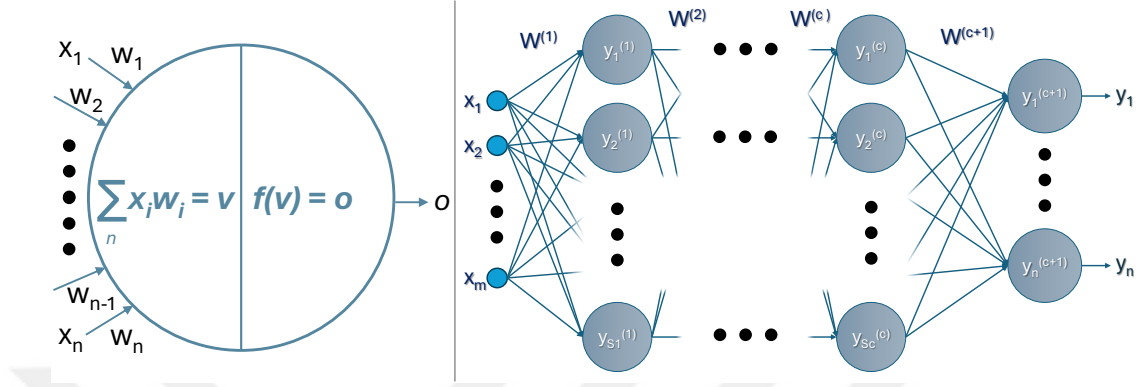


Figure 2.17: A perceptron is given at left, a feedforward network is given at right.

Another well-known layer type is the recurrent, consisting of different types of neurons in the literature. Including hidden state vectors in addition to weights, inputs, and outputs; it is a significant change of the governing equations [50]. By introducing time phenomenon to the training of those networks by Backpropagation Through Time algorithm (BPTT), many tasks became solvable. Recurrent neurons connect to other neurons, but they also make recurrent connections, see Figure 2.18. Thus, they are formulated differently, such as Leaky Integrator Neuron model [48] that can be configured as continuous or with time discretization.

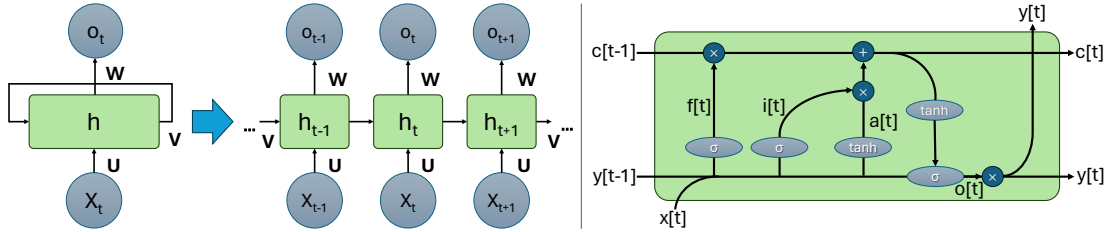


Figure 2.18: BPTT-applied recurrent neuron at left, LSTM cell is at right [48].

Using BPTT for Leaky Integrator Neuron model, time-wise update of a recurrent neuron can be done. An error, $e_i(t)$ is calculated first, by using a loss function (e.g., MSE). Then, variation of error is calculated according to output, y . Later, by applying Euler method to approximate the derivative of error gradient, weight and time constant vector updates are found to lower the loss by minimizing the error. There are different recurrent neurons, such as Long Short-Term Memory cells (LSTMs), see Figure 2.18. Relations of input x , output y , hidden state vector c and parameter matrices W are governed by specific equations [48].

2.4 Chapter Summary

In this chapter, activation functions for neural networks are investigated and our new proposal, DELU activation function is introduced with details. Using different networks, image classification and segmentation tasks are completed on Fashion MNIST, CIFAR-10, CIFAR-100 and Oxford IIIT-Pets datasets for comparison purposes. Resulting success tables are used to make hypothesis tests for comparing DELU with ReLU and ELU activation functions, using the case of CIFAR datasets. In addition, other comparisons are done as well, such as tabulating training and inference timings for all seven activation functions, generating output landscape figures and mean output value visualizations for input standard deviation values using DELU, ELU and ReLU activation functions. Also, parameter optimization is done by changing α and β values for DELU activation function, using Fashion MNIST and CIFAR-10 results. Following these sections, neural network initializations are summarily discussed with the introduction of our new initialization, Truncated Towers. Then, a compatibility analysis is completed using ReLU, ELU and DELU activation functions in combination with Uniform, Gaussian and Truncated Towers initializations, for independent and dependent initialization cases. Results are obtained for Fashion MNIST and CIFAR-10 dataset and presented in comparison. Finally, by shortly discussing learning algorithms, loss functions and network layers used in these neural networks, the chapter is concluded.

Chapter 3

System Identification and Adaptive Control of Bipedal Locomotion by Using Artificial Neural Network Structures

Autonomous robotic systems are expanding their usage areas day by day, thanks to increasing processing power and developed artificial intelligence algorithms. As a subset of robotic systems, legged robots promise a wide range of motion and agile mobility with their structural similarities to animals. Among legged platforms, biped robots have distinct application potential since they have the ability to access already prepared infrastructure for human mobility. Despite all these advantages, the nonlinear dynamics of legged robots make it difficult to design controllers for these systems and find optimum joint trajectories or optimum joint torques.

In this study, we seek answers to these problems: Determining the walking control and system identification capability of artificial neural networks for hybrid dynamic systems and testing the applicability of the adaptive control scheme for neural network based walking controllers. With these objectives, we benefit from

two and three-dimensional two-legged robot models to show the competence of our proposed algorithms since bipedal locomotion control is a challenging problem due to its highly nonlinear and hybrid dynamical nature.

Using these robot models, walking control is implemented using the neural network-based model reference adaptive control framework. With an approach in which neural network-based parallel system identification is employed to identify the system, results are obtained such that artificial neural networks are able to control the walking of the robot model as well. In addition, we verified the applicability of adaptive control for the hybrid dynamic two-legged robot model in simulations. Moreover, we built a real-life 2D robot model as visualized in this thesis. While doing these, novel methods presented in the previous chapter are also tried for these neural networks with varying degrees of success.

In this chapter, we focus on the implementation of neural networks on bipedal locomotion control in robotic systems. First, we start with an introduction of literature about legged locomotion. After that, we present two and three dimensional robot models utilized in our studies. Later, we present our studies on closed-loop neural controller development, robot model system identification with neural networks, and neural network-based model reference adaptive control implementation. Finally, we introduce our works to develop a two-legged physical biped robot system that move in 2D using a planarizer in our laboratory.

3.1 Literature Review

Legged robots have taken their inspiration from real-life examples, such as animals and humans, who have significant advantages of legged locomotion compared to wheeled locomotion; such as larger action space, higher maneuverability, and more agile movement. On the other hand, legged robots include highly nonlinear dynamics represented by hard-to-solve governing equations. It is challenging to define and achieve locomotion stability in legged platforms because of the high degree of freedom, nonlinearities, and the hybrid dynamic nature of legged

platforms.

As a result, different walking stability criteria focused on various gait properties are proposed in the literature. Center of gravity (COG) [51] and zero-moment point (ZMP) [52, 53] based stability analysis methods are frequently utilized for simplicity of their implementation, but like in the case of retired ASIMO robot [54, 55], they are not able to replicate human bipedal locomotion exactly. The former lets us find statically stable walking trajectories that can be utilized at low walking speeds. The latter is more capable of finding stable walking trajectories for fast movement where inertial forces are more effective on system dynamics.



Figure 3.1: Robot ASIMO is shown in the picture, where knees are half-closed while it stands up because of the constraint introduced by its locomotion algorithms [56].

To address the artificial-looking joint trajectories problem, Central Pattern Generator (CPG) named locomotion controllers, which do not include some limitations of COG and ZMP techniques [57], can be used as another alternative in the literature. By mimicking the reflex arc of the spinal cord, this technique may produce joint trajectories closer to biological counterparts than those generated by COG and ZMP techniques. Unfortunately, generated walking trajectories are generally restricted to periodical movements. Also, this method might need hand-tuning, which causes the need for time and resources during finding solutions.

Successive advances in the field of artificial intelligence have led researchers

to think that artificial intelligence-based solutions that surpass human performance in various subjects will be realized sooner than expected [58]. In this scenario, developments in artificial intelligence is considered to speed up addition of autonomous system in the labor force. In this stage, high-performance physical interaction with the real world will be essential to convey tasks that can be automated easily to robots. Thus, developments in artificial intelligence have propelled the need for robots, especially humanoid robots for the aforementioned reasons.

As robots are advantageous for working in hazardous environments, their precise control is also highly required in this manner. With the improving technology and new techniques, there are new solutions to these problems from the literature. With the usage of neural controllers for locomotion, it is possible to walk on rough terrain [9, 59] and control walking speed at different ground inclinations [60]. In addition, neural networks are also used for system identification with using series-parallel or parallel schemes in control systems [10, 48].

With artificial neural networks, it became possible to enlarge the limited parameter space of classical controller design and this advancement may help in increasing control performance for nonlinear and hybrid dynamical systems [61]. Moreover, neural networks offer a high-performance control of nonlinear and complex platforms under supervised and unsupervised learning schemes without requiring hand-tuning as in the case of CPG [61, 62]. There are different works on robot motion control, which employ neural network structures [63, 64, 65].

In this thesis, Taga’s two-legged six-joint robot model, whose foot design includes a parallel spring and damper, is employed for the two-dimensional locomotion simulations [66]. This foot model supports soft contact model with ground that lets non-instantaneous double support phases during walking. Moreover, a CPG controller is proposed with the robot model to generate realistic walking joint trajectories [66]. For similar reasons, NICO robot model is utilized for the three-dimensional locomotion simulations [67]. These robot models are presented in subsequent sections in detail. The efforts to build a 2D, two-legged physical robot and resulting locomotion data are presented in the last section.

3.2 Two Dimensional Robot Model

In this section, we present our work performed by using a biped robot model moving in two dimensions. First, the robot model used for this purpose is demonstrated and simulation conditions are explained, based on using joint torques for robot control. Second, we review Central Pattern Generator (CPG), which is utilized to drive the robot model. Third, dataset generation using the robot model and the CPG is discussed in detail. Fourth, employed Neural Locomotion Controller (NLC) is explained. Fifth, system identification with neural networks is discussed. Finally, we introduce and explain Neural Network-Based Model Reference Adaptive Control (NNBMRAC) with its details.

3.2.1 Biped Robot Model

Inspired by work in [66], we utilized the 2D robot model, which is designed to be driven by the outputs of the CPG for generating the walking dataset. As seen in Figure 3.2, there are hip, knee, and ankle joints in each leg of the robot model. Unlike simpler models in the literature, its thigh and shank limbs are designed to have mass. The upper body of the robot is represented by a point mass placed on the hip joint. In addition, it is driven by six joint torques, and ground reaction and friction forces that emerge by interactions with the ground are modeled with lossy springs.

In short, the robot model consists of four-mass limbs and one point mass at the hip, and while it is controlled with six joint torques, it is exposed to either two or four ground interaction-related forces depending on the number of feet touching the ground. In the studies presented within the scope of this thesis, the equations of motion of the robot model were sampled at 10 KHz and solved numerically using the Euler method.

In the Figure 3.2, x_1 and x_2 correspond to horizontal and vertical hip coordinates, respectively. Moreover, θ_i for $i = 1, 2, 3, 4$ denotes limb angles, l_1 and l_2

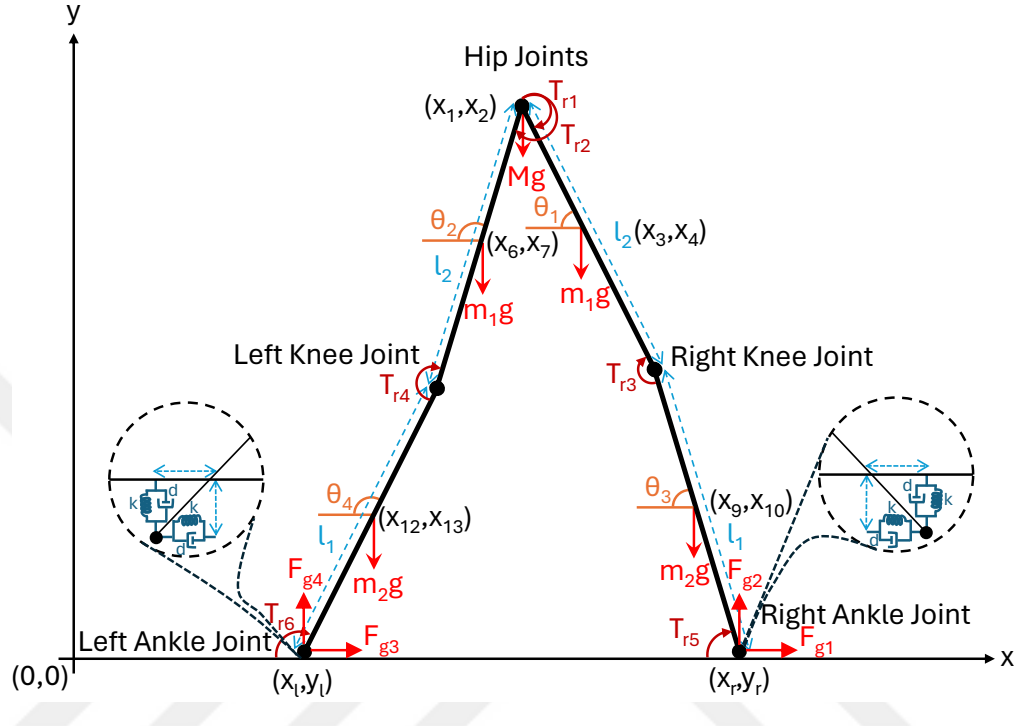


Figure 3.2: The illustration of 2D robot model for bipedal robot locomotion, inspired from and drawn similar to [66].

show the lengths of thigh and shank limbs, respectively. In addition, M , m_1 , and m_2 stand for the point mass on the hip, thigh limb mass, and shank limb mass, respectively.

Shown in joint points, the torque values are given with their directions in the figure and labeled as T_{ri} for $i = 1, 2, 3, 4, 5, 6$. F_{gi} for $i = 1, 2, 3, 4$ represents the forces related to ground interaction, such as ground reaction and friction forces, acting on the ankle when it enters the ground. Presented in zoomed-in sections, k and d are spring and damping coefficients, respectively. Finally, g is the gravitational acceleration term.

To explain the motion equations, we need to define an $\mathbf{x}$ state vector which includes central limb positions and angles as seen in the Figure 3.2:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \theta_1 \\ x_6 \\ x_7 \\ \theta_2 \\ x_9 \\ x_{10} \\ \theta_3 \\ x_{12} \\ x_{13} \\ \theta_4 \end{bmatrix} \quad (3.1)$$

The general form of equation of motion can be expressed as follows [66]:

$$\ddot{\mathbf{x}} = P(\mathbf{x})\mathbf{F} + \mathbf{Q}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{T}_r(\mathbf{y}), \mathbf{F}_g(\mathbf{x}, \dot{\mathbf{x}})) \quad (3.2)$$

In the (3.2), $\ddot{\mathbf{x}}$ and $\dot{\mathbf{x}}$ are derived from the state vector $\mathbf{x}$ which is the 14-element vector as explained in (3.1). $\mathbf{F}$ is 8-element vector of constraint forces, $\mathbf{T}_r$ is 6-element vector of torques, $\mathbf{F}_g$ is 4-element vector of forces on ankle depending to state of ground plane, $\mathbf{y}$ is the 12-element vector of the Central Pattern Generator (CPG) which is explained at the next part. $\mathbf{Q}$ is a 14-element vector and P is a (14 x 8) matrix relating the force and torque values to position and angle values of robot limbs. Mainly this equation is used for the simulation for 2D biped robot locomotion, alongside the specific CPG designed for this robot model. For the detailed explanation of the model and expressions of the terms given in (3.2), see [66].

3.2.2 Central Pattern Generator

Central Pattern Generators (CPGs) are inspired by periodical reflex-arc-like movements of animals controlled by their spinal cord, [57, 66, 68]. We can exemplify such periodical movements by undulation in snakes, walking and running in humans, and pacing, trotting, and galloping in horses. In literature, CPGs are utilized to produce appropriate trajectories to locomote various robots such as snake, bipedal, and quadruped robots, e.g. [66, 69, 70, 71, 72].

A CPG consists of a network of neural oscillators which are generally modeled with coupled differential equations to generate sustained oscillations without requiring oscillatory inputs [57, 66]. There are different neural oscillator models proposed in the literature such as van der Pol, phase, amplitude controlled phase and Matsuoka oscillators, for further details see [73, 74, 75, 76].

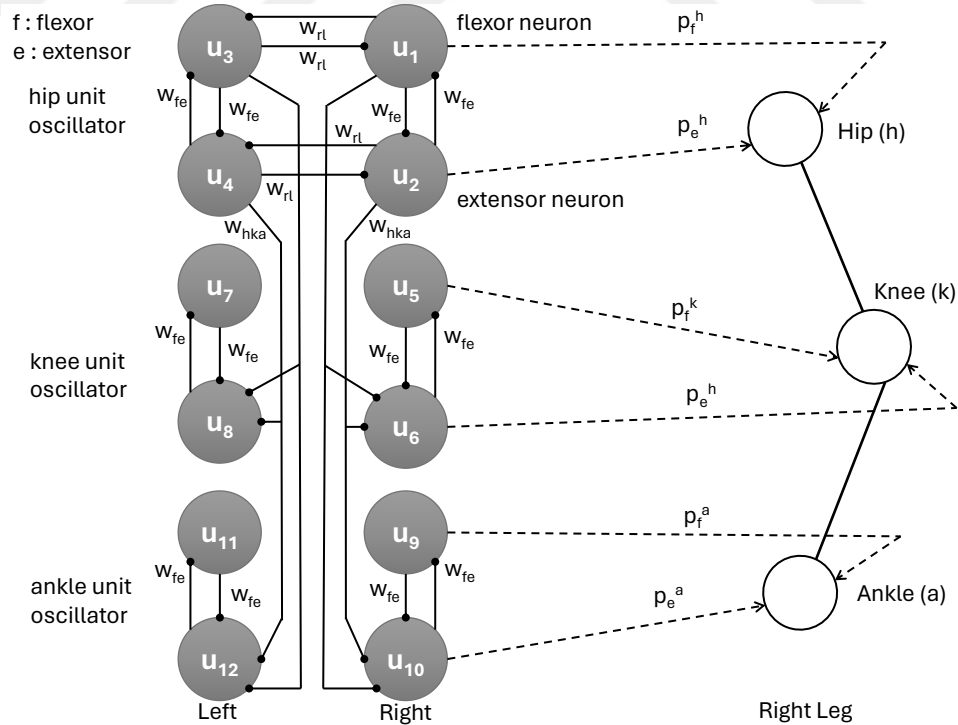


Figure 3.3: Structure of central pattern generator employed to locomote biped robot model, see [66].

In this thesis, we benefit from a central pattern generator defined in the inspirational work of Taga et al. [66] with some modifications to drive the biped robot model presented in the previous subsection. While doing that, we utilize optimized torque multipliers in [59] to enlarge robot motion space. In detail, the utilized central pattern generator consists of six Matsuoka oscillators as shown in Figure 3.3. All oscillators are fed with excitation value input, u_0 , which is a constant that affects the oscillation sizes at the oscillator output, see [66]. As discussed in the previous part, $\mathbf{y}$ is the 12-element vector generated by the CPG which couples with the biped robot. Coupled oscillator equations are given below for $i = 1, \dots, 12$ from [66]:

$$\tau'_i \dot{u}_i = -u_i + \sum_{j=1}^{12} w_{ij} y_j - \beta v_i + u_0 + Feed_i(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{F}_g(\mathbf{x}, \dot{\mathbf{x}})) \quad (3.3)$$

$$\tau'_i \dot{v}_i = -v_i + y_i \quad (3.4)$$

In these equations, $Feed_i$ is the feedback signal from the musculo-skeleton system as explained in [66]. u is the inner state of a neuron, v is the variable of adaptation/self-inhibition of a neuron, u_0 is the excitation constant, w is the connecting weight, τ and τ' are time constants of inner state and adaptation effect and y_i is the output of i th neuron. Finally, by concating all y_i values, $\mathbf{y}$ vector can be expressed as:

$$\mathbf{y} = \max(\mathbf{0}, \mathbf{u}) \quad (3.5)$$

The output, $\mathbf{y}$ is used for the equation (3.2) to generate the torques necessary for the robot locomotion. Finally, differential equations of neuron models inside the Matsuoka oscillators are numerically solved by Fourth-Order Runge Kutta at 10 KHz sampling frequency.

3.2.3 Dataset Generation

Datasets which consist of walking patterns are needed to train and test the proposed neural networks in the control and system identification under the supervised learning scheme. Using the 2D biped robot model and CPG presented in previous sections, we generated a dataset to obtain training, validation, and test sets, which are utilized in training neural networks that will be used for the purpose of applying control and system identification. The dataset comprises 10-second-long walking patterns and shorter unsuccessful fall patterns, produced with different walking speeds and ground inclination angles. For each data pattern, 6 joint torques, 4 ground contact-related forces, 4 limb angles, and 10 position states are recorded. In addition to these, 12 feedback connections are added to walking data as defined in [66]. Lastly, speed excitation value, ground inclination angle, and average walking speed achieved after the first second of the walk are appended to other time series. As an exception for fall patterns below a duration of 1 second, walking speed is calculated from the beginning.

Recorded time series are decimated at the rate of 100:1 to speed up the training of neural networks. Hence, the sampling frequency of data is decreased from 10 KHz to 100 Hz, which is considered enough to control and identify robot locomotion dynamics. Thus, the lengths of the time series are reduced to 1000 time steps. In addition, recorded time series are normalized to $[-1/3, 1/3]$ interval to increase the performance of neural networks and the applicability of different neuron models.

Details about dataset generation are given in Table 3.1 and Figure 3.4 shows the distribution of training, validation, and test samples distribution with respect to walking speed and ground inclination angle values. Inclination angle values are changed to manipulate walking terrain in the simulation environment. Reported walking speeds are obtained by changing excitation values at equal intervals. A non-linear relation exists between the given excitation input and the reached walking speed in the CPG-controlled robot locomotion. For this reason, some gait changes and resulting speed jumps can be observed even though the excitation

value increases at equal intervals, as seen in Figure 3.4. This makes it difficult to control goal velocity with the CPG and results in the phenomenon observed in the following graphs.

For the 2D robot dataset, speed excitation and inclination angle intervals are chosen as $[3, 10]$ and $[-9^\circ, 9^\circ]$, respectively. From the speed excitation value interval, 40 evenly spaced values are chosen. Similarly, evenly spaced 39 values are selected from the inclination angle interval. Using combinations of these two values, in total, 1560 walks are attempted, and 546 of them (35%) are successful walks, without falling, under the control of CPG. Later on, successful walking patterns are divided into training, validation, and test sets. Amongst those 546 walks, 384 of them (70.33%) are chosen as training, 64 of them (11.72%) are selected as validation, and 98 of them (17.95%) are chosen as test samples.

Table 3.1: Properties of generated 2D robot locomotion data.

Speed excitation interval	Ground inclination angle interval	Total walking attempts amount	Total successful walking amount	Training set walking amount	Validation set walking amount	Test set walking amount
$[3, 10]$	$[-9^\circ, 9^\circ]$	1560	546	384	64	98

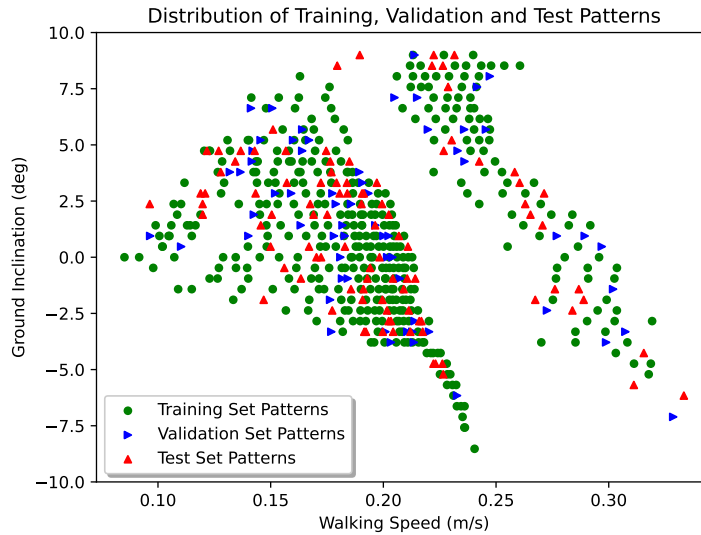


Figure 3.4: The successful walking dataset generated by CPG, is divided into training, validation and test sets as illustrated.

3.2.3.1 Dataset Version With Fall Patterns

For the purposes of better understanding and better learning with neural networks, a different version of the dataset is created. In this version, the configuration of the dataset is kept as the same. However, in addition to successful (stable) walking patterns, patterns which contain robot falls are added into this version of the dataset for enlargement. To display the samples added to the dataset, which are 1014 patterns, below visual is presented where training, validation and testing patterns are present. In this configuration, 896 samples (88.36%) are allocated for the training set, 64 samples (6.31%) are separated as the validation set and 54 samples (5.33%) are forming the test set.

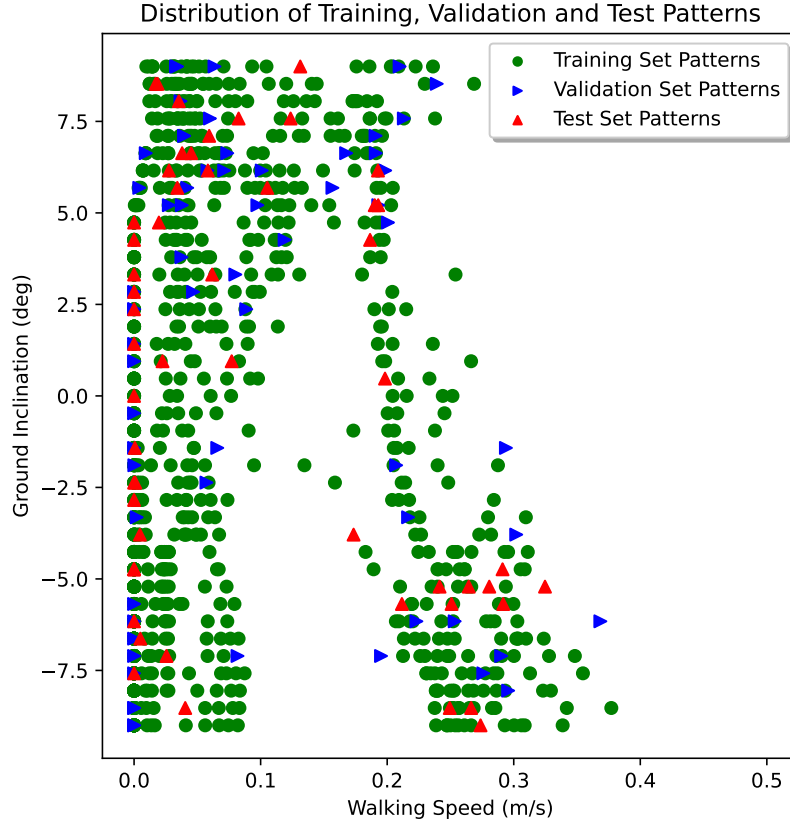


Figure 3.5: The dataset samples which contain falls are shown visually, being divided into training, validation and test sets as illustrated.

The reason behind this choice is to make system identification more original, by letting neural networks to learn what kind of movements cause the falling behavior for the robot. In this manner, different system identification networks are trained with this version of the dataset as well. For their results, readers are referred to relevant subsections in this section.



3.2.4 Neural Locomotion Controller

In this part, we discuss our studies in applying neural network-based torque controllers in the closed-loop biped locomotion control problem. Figure 3.6 shows the applied control diagram for this discussion. While doing that, we highlight the effectiveness and weakness of neural network-based elements on the control systems and propose a solution to one of the stated weaknesses with this thesis work as a contribution to the literature.

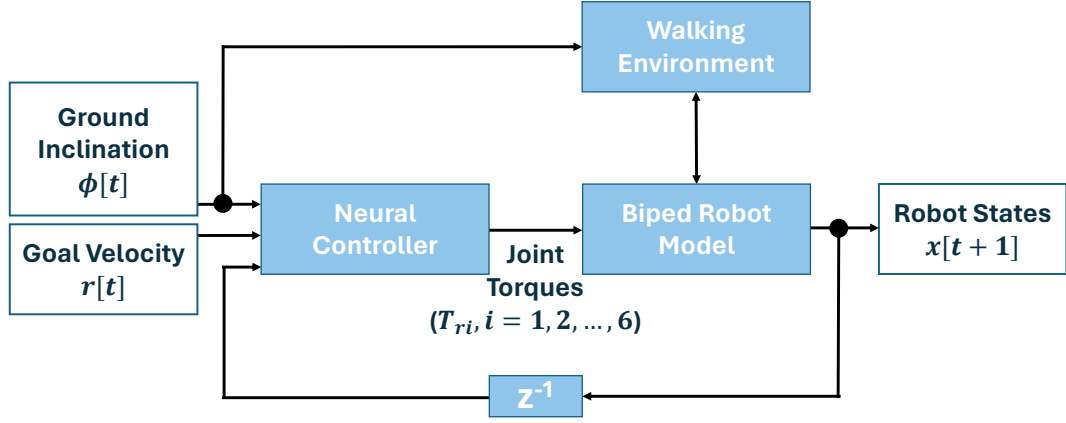


Figure 3.6: The control diagram with neural locomotion controller.

Figure 3.6 presents the “Neural Control Diagram”, which shows the details of the control structure. There are four blocks in the control diagram. First, “Walking Environment” block represents the simulation environment where the robot moves on a surface with a constant inclination according to ground inclination reference input, and the robot model interacts with the walking environment during the experiments via ground contact-related forces. Second, “Biped Robot Model” block denotes the robot model which takes six joint torques $(T_{ri}[t], i = 1, 2, \dots, 6)$ and outputs next time step ten robot limb position states $(x_i[t], i = 1, 2, 3, 4, 6, 7, 9, 10, 12, 13)$, four ground interaction related forces $(F_{gi}[t], i = 1, 2, 3, 4)$, and twelve feedback connections as explained in [66]. In this way, it supplies the information of robot states. The third block is “Neural Controller” which generates six joint torques $(T_{ri}[t], i = 1, 2, \dots, 6)$ with respect to taken constant ground inclination angle $(\phi[t])$, constant goal velocity value

($r[t]$), robot limb position states ($x_i[t], i = 1, 2, 3, 4, 6, 7, 9, 10, 12, 13$), ground interaction related forces ($F_{gi}[t], i = 1, 2, 3, 4$), and twelve feedback connections as inputs. The last one is the unit time step delay block. The “Neural Controller” block takes inputs through the delay block other than reference ground inclination and goal velocity reference inputs, which correspond to ground inclination and average walking speed in the dataset, respectively.

Here, the neural controller block includes an artificial neural network that is trained with the generated training dataset, and it is utilized to locomote the robot model. In the literature, such neural torque controllers are found helpful to satisfy stable locomotion in wider ground inclination angle range and more robust against rough terrain compared to CPG utilized in dataset generation, [59].

Studies in the scope of this thesis show these types of neural controllers can cause speed steady-state errors. In this context, we investigate the effects of different neuron models and network architectures on walking success and speed steady-state error.

To elaborate on details, we can explain walking success and steady-state error definitions. Walking success is defined as follows: A walk is considered successful (i.e. stable) only if the robot doesn’t fall during the experiment. Otherwise, the walk is counted as unsuccessful (i.e. unstable). Steady-state error is defined as the deviation of goal speed from the targeted goal speed by the controller, and it is only calculated for the successful walks.

The neural controller block consists of neural networks with one or two hidden layers. In the hidden layers, $\tanh(\cdot)$, ReLU, ELU, LeakyReLU, Gated Recurrent Unit (GRU), LSTM, Elman Recurrent Neural Network Unit (ERNN) are used in addition to our novel activation function, DELU. AdamW optimizer is used for these networks with 0.001 as the learning rate constant at the beginning. L2 regularization is employed for the neural networks, with 0.5, 0.1, or 0 values where 0 means no L2 regularization application. Their training lasts 3000 epochs, and for every 100 epochs, network weights are recorded. The best-performing network over the validation set is found among those recorded networks and then used for

inference on all datasets. Finally, neural networks ends with six output neurons, which are all linear, corresponding to joint torques.

On the next page, Table 3.2 contains the results of various neural network architectures are given. These results are obtained with trained neural networks via NVIDIA Quadro RTX 6000 model GPUs. In the inference stage, required calculations are performed using two Intel Xeon Gold 6248R CPUs, since two processors can calculate faster than the GPU for neural networks of this size. Apart from the 100 LSTM and 200 LSTM cases, learnable parameter numbers are chosen as close to each other as possible for a fair comparison. Due to small sample size, walking success and steady-state error rates for training, validation and test sets can be close to each other. Here, the minimum acceptable walking success threshold is chosen as 75%, and the maximum allowable speed steady state error limit is chosen as 5%. These constraints are met by three neural networks: 43 LSTM and 74 ReLU layered network with no L2 regularization, 43 LSTM and 74 ReLU layered network with 0.1 L2 regularization, and lastly the neural network with 43 LSTM and 74 DELU hidden layers with 0.1 L2 regularization constant. Only these three networks satisfy the walking success threshold and error limit, and their results are highlighted in bold at the table. Also, it is seen that LSTM and ERNN layers are better than GRU layers for this configuration, and two hidden layer structures are more successful than single-layer networks in general. Results of best entry of the table, 43 LSTM-74 ReLU network with no L2 regularization are visualized in the next figure on all sets.

Table 3.2: Walking performances of neural controller for bipedal 2D locomotion.

Hidden Layers	Weight Amount	L_2	Best Epoch	Training Set		Validation Set		Test Set	
				Walking Success	Speed Steady State Error	Walking Success	Speed Steady State Error	Walking Success	Speed Steady State Error
50 LSTM	16306	0	2400	81.77%	6.62%	79.69%	6.25%	83.67%	5.24%
		0.1	3000	79.95%	5.69%	79.69%	5.71%	78.57%	6.59%
		0.5	2800	84.38%	9.44%	85.94%	9.01%	85.71%	8.87%
100 LSTM	52606	0	2300	77.86%	7.5%	78.12%	7.64%	78.57%	6.79%
		0.1	1900	87.24%	8.05%	84.38%	9.49%	86.73%	7.93%
		0.5	500	94.53%	14.65%	95.31%	13.13%	93.88%	12.82%
200 LSTM	185206	0	1000	68.75%	4.48%	75%	4.49%	74.49%	3.82%
		0.1	1600	74.74%	7.6%	78.12%	7.67%	75.51%	7.68%
		0.5	1500	77.86%	13%	81.25%	12.19%	83.67%	13.07%
31 LSTM, 31 LSTM	15692	0	1900	70.57%	9.96%	71.88%	9.69%	76.53%	9.69%
		0.1	2100	77.86%	7.66%	75%	6.03%	83.67%	6.76%
		0.5	800	94.27%	12.39%	90.62%	11.06%	94.9%	10.69%
71 Tanh, 33 LSTM	16255	0	2000	65.36%	5.26%	70.31%	5.13%	70.41%	5.59%
		0.1	2800	90.1%	9.78%	90.62%	8.64%	92.86%	8.89%
		0.5	1300	92.97%	13.56%	90.62%	11.97%	92.86%	11.94%
71 LeakyReLU, 33 LSTM	16255	0	1600	74.22%	6.64%	75%	5.75%	76.53%	5.74%
		0.1	2700	79.69%	13.32%	76.56%	14.95%	78.57%	14.57%
		0.5	2700	83.85%	10.16%	85.94%	10.1%	80.61%	9.86%
43 LSTM, 74 LeakyReLU	16262	0	900	76.56%	5.31%	79.69%	5.1%	78.57%	4.81%
		0.1	700	88.54%	10.98%	84.38%	9.53%	91.84%	10.6%
		0.5	2400	88.02%	12.14%	84.38%	13.98%	85.71%	11.73%
43 LSTM, 74 ReLU	16262	0	1100	84.11%	4.19%	82.81%	3.84%	84.69%	3.16%
		0.1	1100	84.38%	4.27%	85.94%	4.78%	85.71%	4.09%
		0.5	2200	92.45%	11.79%	92.19%	11.98%	92.86%	10.3%
43 LSTM, 74 ELU	16262	0	500	57.81%	6.83%	53.12%	8.6%	61.22%	7.2%
		0.1	2900	78.91%	13.89%	78.12%	12.89%	80.61%	13.18%
		0.5	800	84.38%	10.33%	89.06%	9.5%	84.69%	8.81%
43 LSTM, 74 DELU	16262	0	100	62.24%	5.29%	59.38%	3.52%	66.33%	3.67%
		0.1	2600	75.52%	4.85%	76.56%	4.99%	78.57%	4.54%
		0.5	1800	77.6%	6.88%	79.69%	6.94%	83.67%	7.6%
59 GRU	16113	0	400	62.76%	7.19%	65.62%	6.1%	70.41%	5.37%
		0.1	700	70.31%	4.6%	70.31%	4.26%	77.55%	4.54%
		0.5	2700	73.96%	4.51%	65.62%	4.53%	75.51%	4.22%
110 ERNN	16066	0	2800	77.6%	17.38%	76.56%	16.5%	76.53%	17.21%
		0.1	2500	81.51%	5.26%	78.12%	4.74%	82.65%	4.76%
		0.5	2000	58.07%	21.88%	59.38%	16.85%	66.33%	18.24%
83 ERNN, 74 ReLU	16045	0	1900	71.09%	6.3%	68.75%	5.75%	76.53%	4.88%
		0.1	1900	77.34%	7.53%	87.5%	8.08%	82.65%	7.53%
		0.5	2600	68.49%	24.7%	70.31%	23.39%	74.49%	20.72%
83 ERNN, 74 LeakyReLU	16045	0	1100	70.05%	11.35%	75%	12.48%	70.41%	12.93%
		0.1	2900	77.86%	6.67%	78.12%	7%	79.59%	6.19%
		0.5	2100	79.69%	11.76%	79.69%	10.51%	84.69%	9.59%

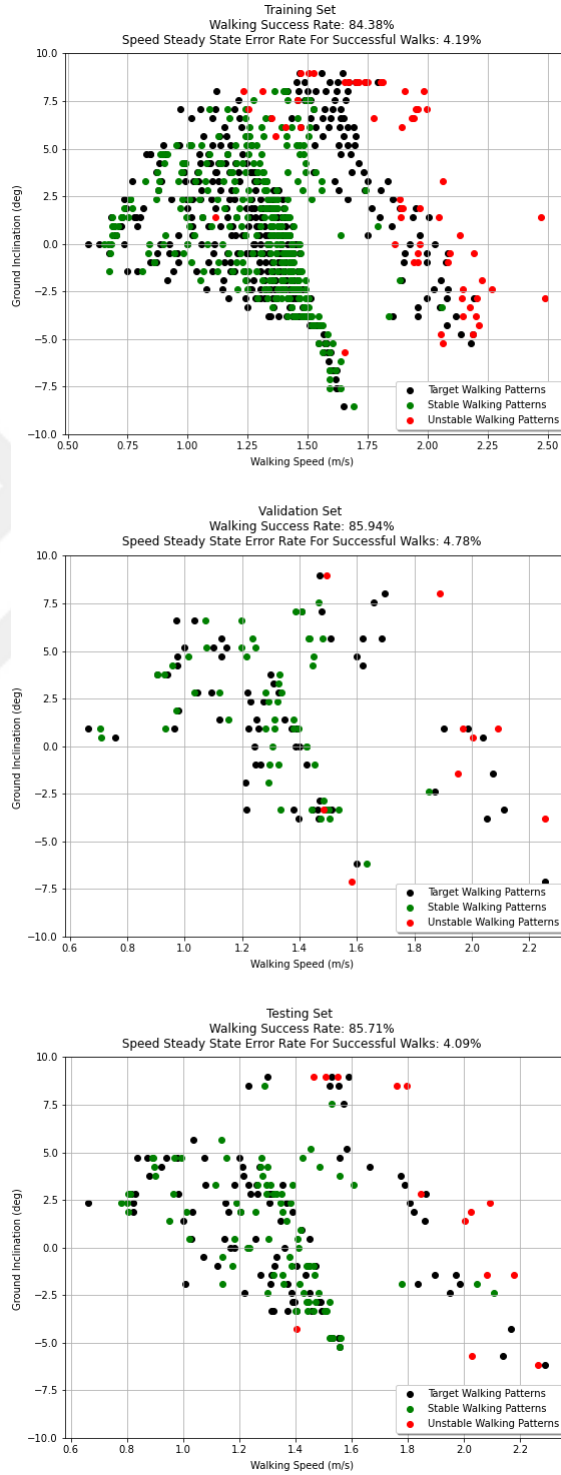


Figure 3.7: Performance of the neural controller using 43 LSTM and 74 ReLU layers with 0.1 L2 regularization, on separated training, validation and test sets.

3.2.5 System Identification With Neural Networks

In addition to our studies about using neural networks as locomotion controllers, we also employ them for system identification. Since the Universal Approximation Theorem implies that neural networks with enough learnable parameters can approximate functions with the desired accuracy, we expect them to be able to approximate a highly nonlinear, biped robot model that we use with enough accuracy. In this study, we use the parallel system identification scheme shown in the diagram in Figure 3.8.

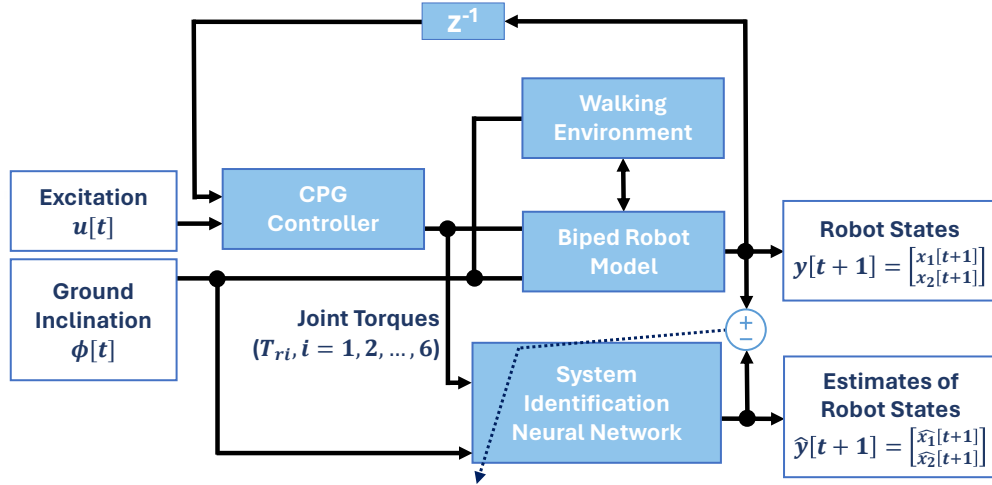


Figure 3.8: Parallel system identification configuration is presented, inspired from [48].

Basically, there are three main blocks in this diagram. First, a CPG controller takes the excitation value constant alongside the previous time step robot state values. It forwards the produced joint torque values to the second block, Biped Robot Model -explained in the previous sections- and the third block, System Identification Neural Network. Here, if we exclude the third block, the remaining diagram will be the same as the dataset generation. In this study, we use the previously generated dataset. During the walk, the System Identification Neural

Network block estimates the next time step robot hip vertical and horizontal position states (the only robot states used for this scheme), according to the joint torques produced by the CPG controller and ground inclination angle information. Hence, it is aimed that this neural network approximates the input and output relationship of the biped robot model. The prediction error is defined as the difference between the outputs of the identified plant and the identification neural network. The loss is calculated using the Mean Squared Error (MSE) definition and then summed up cumulatively by adding vertical and horizontal positions of the hip, and halved afterwards.

Only horizontal and vertical positions of robot hip (states $x_1[t]$ and $x_2[t]$, respectively) are chosen as the representative parameters, because of the fact that such setting is found as an compatible configuration for the aim of adaptive learning scheme presented in the next sections. These values are obtained from the dataset generated using the biped robot model and the CPG.

Outputs of the system identification block which are horizontal and vertical position states are listed inside $\mathbf{y}[t]$ vector and the inputs are six joint torque values ($T_{ri}[t], i = 1, 2, \dots, 6$) and ground inclination angle ($\phi[t]$). The equation used to calculate the MSE loss metric is explained below:

$$\mathbf{y}[t] = \begin{bmatrix} x_1[t] \\ x_2[t] \end{bmatrix} \quad (3.6)$$

$$\hat{\mathbf{y}}[t] = \begin{bmatrix} \hat{x}_1[t] \\ \hat{x}_2[t] \end{bmatrix} \quad (3.7)$$

$$\mathbf{e}[t] = \mathbf{y}[t] - \hat{\mathbf{y}}[t] = \begin{bmatrix} x_1[t] - \hat{x}_1[t] \\ x_2[t] - \hat{x}_2[t] \end{bmatrix} \quad (3.8)$$

$$l[t] = \frac{\mathbf{e}[t]^T * \mathbf{e}[t]}{2} \quad (3.9)$$

$$L = \sum_{t=1}^N \frac{l[t]}{N} \quad (3.10)$$

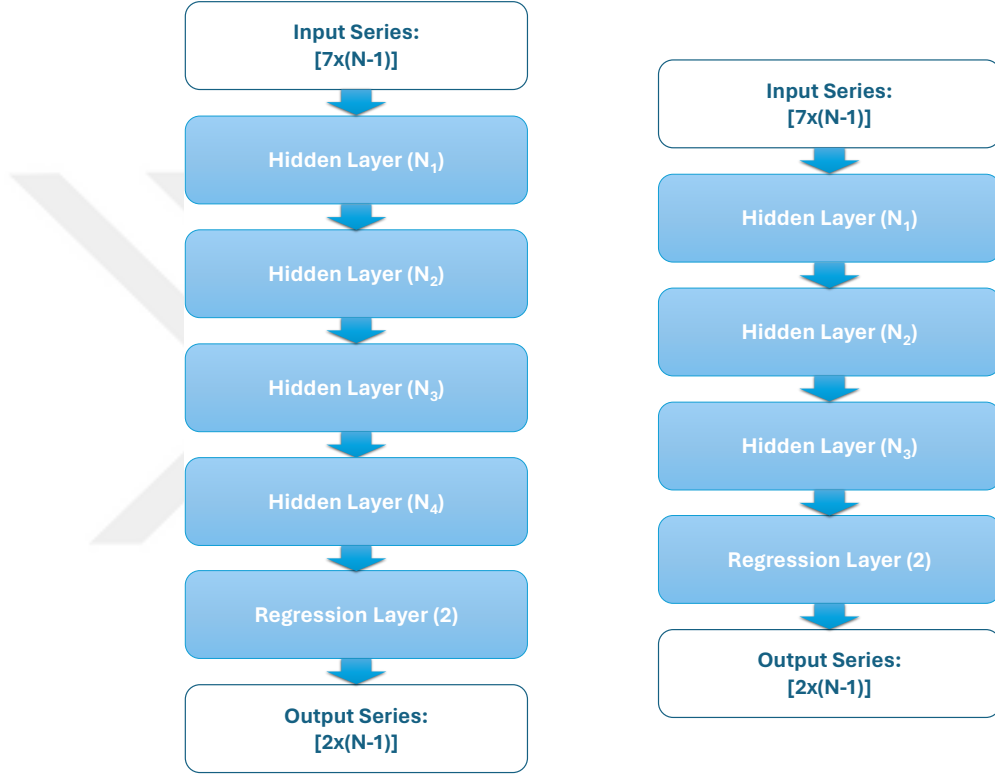
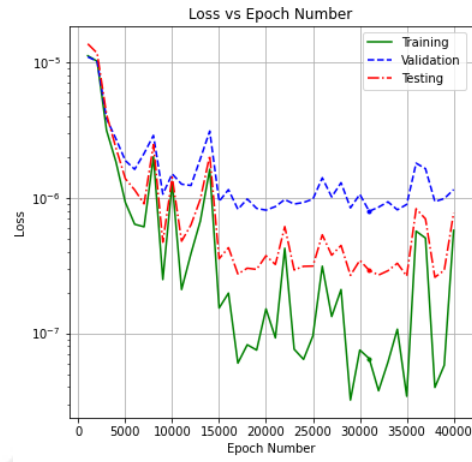
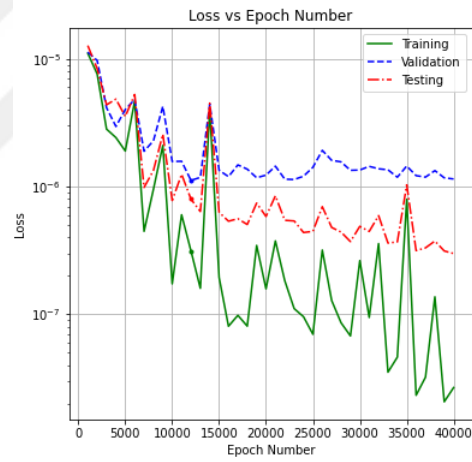


Figure 3.9: The neural network structures used for system identification network for 2D robot simulations where N demonstrates time step number.

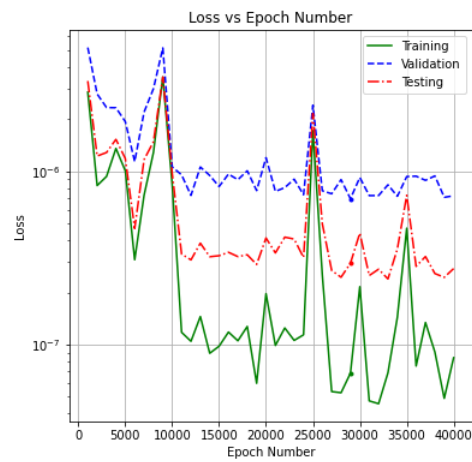
As seen in the Figure 3.10, the top 3 most successful networks for the supervised system identification training are the ones which include ReLU, Tanh and DELU activation functions in their first layer. However, please note that their results presented in Table 3.3 are pretty close especially at the end of the training, where original DELU is more successful for the case of test loss, in comparison to its representative result in the table - the chosen network with lowest validation loss.



(a)



(b)



(c)

Figure 3.10: Training, validation and test results are shown for networks with; (a) for ReLU, (b) for DELU, (c) for Tanh layers

Table 3.3: System identification network results in comparison.

Hidden Layers	Best Epoch	Training	Validation	Test
		Loss Scaled Mean	Loss Scaled Mean	Loss Scaled Mean
70 Tanh, 231 LSTM, 231 LSTM	29000	$6.84 \cdot 10^{-6}$	$6.94 \cdot 10^{-7}$	$2.95 \cdot 10^{-7}$
70 ReLU, 231 LSTM, 231 LSTM	31000	$6.50 \cdot 10^{-8}$	$7.93 \cdot 10^{-7}$	$2.91 \cdot 10^{-7}$
70 DELU, 231 LSTM, 231 LSTM	12000	$3.14 \cdot 10^{-7}$	$1.12 \cdot 10^{-6}$	$7.99 \cdot 10^{-7}$
70 LeakyReLU, 229 LSTM, 229 LSTM, 70 LeakyReLU	37000	$1.25 \cdot 10^{-7}$	$2.41 \cdot 10^{-6}$	$6.98 \cdot 10^{-7}$
70 Tanh, 229 LSTM, 229 LSTM, 70 Tanh	13000	$1.38 \cdot 10^{-7}$	$5.69 \cdot 10^{-7}$	$3.84 \cdot 10^{-7}$
70 ReLU, 229 LSTM, 229 LSTM, 70 ReLU	13000	$1.31 \cdot 10^{-7}$	$8.67 \cdot 10^{-7}$	$4.76 \cdot 10^{-7}$
70 DELU, 229 LSTM, 229 LSTM, 70 DELU	40000	$2.69 \cdot 10^{-8}$	$1.86 \cdot 10^{-6}$	$7.12 \cdot 10^{-7}$

3.2.5.1 Tuning of DELU Activation Function

As an expansion to this section, DELU activation function is tuned to reach better success rates for the system identification of 2D biped robot simulation. For this tuning, 70 DELU-231 LSTM-231 LSTM network architecture is chosen. Similar to the section in the previous chapter, the tuning is done by varying the β parameter of the activation function between 1.25 and 3.25 values. Here, α value is fixed as 1 and z_c value is found using (2.11) with these α and β values. Resulting neural networks are tested as seen in the Table 3.4 given below. Here, original DELU is compared to other DELU tunings. As indicated in their names, DELUvb1.25 stands for DELU activation function tuning with its β value is set to 1.25 instead of the original β value, 2. Likewise, for 8 DELU tunings with different β values, results are obtained and compared with original DELU results.

Table 3.4: System identification results are given in comparison for DELU activation function tunings.

Hidden Layers	Best Epoch	Training	Validation	Test
		Loss Scaled Mean	Loss Scaled Mean	Loss Scaled Mean
70 DELUvb1.25, 231 LSTM, 231 LSTM	40000	$3.52 \cdot 10^{-8}$	$6.96 \cdot 10^{-7}$	$2.98 \cdot 10^{-7}$
70 DELUvb1.5, 231 LSTM, 231 LSTM	34000	$1.90 \cdot 10^{-7}$	$9.14 \cdot 10^{-7}$	$5.22 \cdot 10^{-7}$
70 DELUvb1.75, 231 LSTM, 231 LSTM	19000	$1.29 \cdot 10^{-7}$	$1.18 \cdot 10^{-6}$	$8.91 \cdot 10^{-7}$
70 DELU, 231 LSTM, 231 LSTM	12000	$3.24 \cdot 10^{-7}$	$1.12 \cdot 10^{-6}$	$7.99 \cdot 10^{-7}$
70 DELUvb2.25, 231 LSTM, 231 LSTM	29000	$8.59 \cdot 10^{-8}$	$1.58 \cdot 10^{-6}$	$7.44 \cdot 10^{-7}$
70 DELUvb2.5, 231 LSTM, 231 LSTM	40000	$2.41 \cdot 10^{-8}$	$7.70 \cdot 10^{-7}$	$7.53 \cdot 10^{-7}$
70 DELUvb2.75, 231 LSTM, 231 LSTM	40000	$5.71 \cdot 10^{-8}$	$1.75 \cdot 10^{-6}$	$8.49 \cdot 10^{-7}$
70 DELUvb3, 231 LSTM, 231 LSTM	9000	$2.21 \cdot 10^{-7}$	$1.59 \cdot 10^{-6}$	$9.02 \cdot 10^{-7}$
70 DELUvb3.25, 231 LSTM, 231 LSTM	35000	$2.87 \cdot 10^{-8}$	$1.12 \cdot 10^{-6}$	$5.4 \cdot 10^{-7}$

As presented in the table, for training success rates, DELUvb3.25 has the least

loss value. More importantly, for the validation and test losses DELUvb1.25 has the least loss value in comparison to original DELU and other tunings. Obtained results are very close to ReLU and Tanh networks which lead in Table 3.3 for test success rates, but DELUvb1.25 would be third in that table by a difference of only $3 \cdot 10^{-9}$ for Tanh and $7 \cdot 10^{-9}$ for ReLU. In result, by applying tuning to DELU activation function for this section, the result obtained with original DELU is improved and resulting test success became much more close to first two entries of the Table 3.3.

3.2.5.2 Training by Using Both Walking and Falling Patterns

To increase the capability of system identification neural networks, walking dataset is expanded by adding fall patterns into its samples. In this way, system identification is done for unsuccessful walking patterns as well, as an attempt to enhance the learning by forcing the neural network to analyze the gaits and movements which cause falling behavior for the robot.

Table 3.5: System identification network results in comparison.

Hidden Layers	Best Epoch	Training	Validation	Test
		Loss	Loss	Loss
		Scaled Mean	Scaled Mean	Scaled Mean
70 DELU, 231 LSTM, 231 LSTM	26750	$3.12 \cdot 10^{-6}$	$4.36 \cdot 10^{-6}$	$5.19 \cdot 10^{-6}$
70 LeakyReLU, 231 LSTM, 231 LSTM	23500	$1.50 \cdot 10^{-6}$	$4.46 \cdot 10^{-6}$	$3.61 \cdot 10^{-6}$
70 ReLU, 231 LSTM, 231 LSTM	37750	$7.52 \cdot 10^{-7}$	$1.45 \cdot 10^{-6}$	$1.44 \cdot 10^{-6}$
70 Tanh, 231 LSTM, 231 LSTM	29750	$1.72 \cdot 10^{-6}$	$4.83 \cdot 10^{-6}$	$3.43 \cdot 10^{-6}$

As explained in the dataset section, best networks are picked up by comparing the losses for the 64-sample successful walking validation set patterns. No L2 regularization is used for any of these networks. Resulting tables of comparison for different neural network configurations are presented in the Table 3.5, where 70 ReLU-231 LSTM-231 LSTM network leads. This experiment and presence of a limited test loss increase has shown the capability of these system identification neural networks to cope with both walking and falling patterns in terms of learning process, which led to their usage for system identification in the adaptive learning configuration presented in the next subsection, for different scenarios.

3.2.6 Neural Network Based Model Reference Adaptive Control

This section introduces our Neural Network-Based Model Reference Adaptive Control (NNBMRAC) studies. Figure 3.11 below shows details of our NNBMRAC implementation. This scheme is designed to apply adaptive control by utilizing multiple neural networks in different roles to fine-tune walking performance, such as overcoming speed steady-state errors made by the neural controllers.

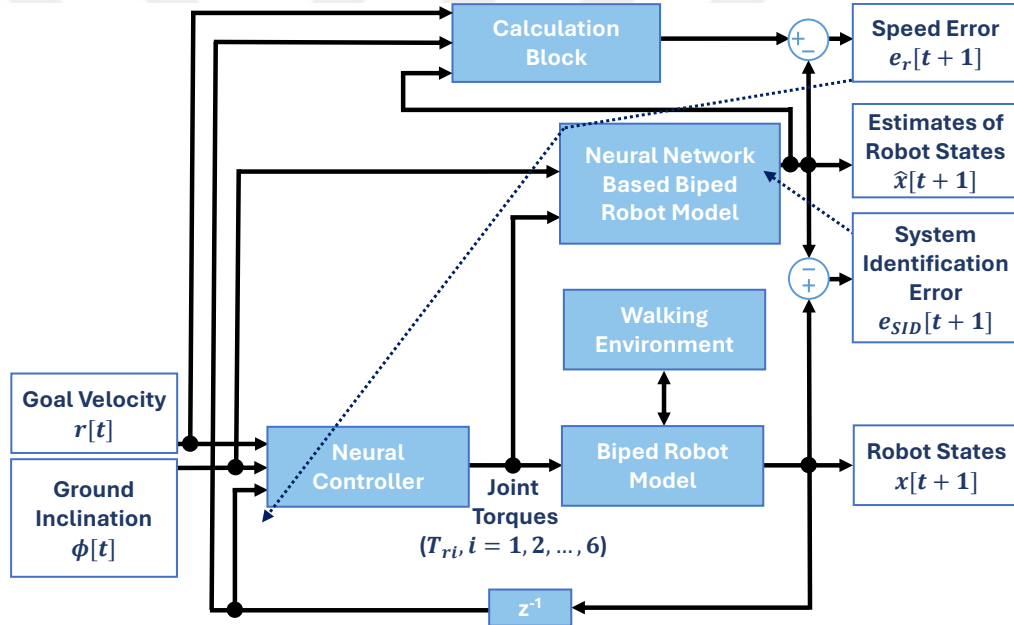


Figure 3.11: The control diagram with neural locomotion controller and neural network based biped robot model from the system identification part.

To decrease speed steady-state error, we need to update the controller neural network parameters in a way that will minimize the determined error metrics. To do this, the calculated error gradient using the chosen error metric must be back-propagated to the neural controller. Unfortunately, passing the error gradient over the biped robot model in Figure 3.6 is challenging since the robot model includes nonlinear and hybrid dynamics. To overcome the backpropagation problem, Biped Robot Model and Neural Network Based Biped Robot Model (NNBBRM) block is utilized in the adaptive control scheme in Figure 3.11 as system identification block based on a neural network. Since propagating error

gradient is relatively more straightforward over neural networks than biped robot model, this system identification block is utilized in this diagram. Here, we assume that if the system identification neural network approximates the input and output relationship of the robot model well enough, then the propagated error gradient over it will be capable of updating the parameters of the controller neural network to minimize the determined error metric.

The control diagram in Figure 3.11 shows the places of both the neural locomotion controller and system identification blocks. Inputs and outputs of these blocks are explained in previous subsections 3.2.4 and 3.2.5. The proposed NNBMRA algorithm does not require generated datasets using CPG to optimize trajectory. Instead, it employs estimated and actual walking trajectories to tune weights of neural networks. To do that, the steps which should be followed are listed in a pseudocode-like form below. As a first step, controller neural network is fed with goal velocity, ground inclination angle, and taken feedback from robot model. After that, joint torques are calculated and delivered to both Biped Robot Model and NNBBRM blocks. In the NNBBRM block, in other words the system identification block, forward propagation continues and robot states are estimated, $\hat{\mathbf{x}}[t+1]$, and then they are excluded from actual robot states calculated with robot equation of motions, $\mathbf{x}[t+1]$, by Biped Robot Model block to obtain error, $\mathbf{e}_{SID}[t+1]$. After that, we calculate mean squared error (MSE) metric as explained in Subsection 3.2.5 and it is used for backpropagation on NNBBRM block as explained in the previous part. Afterwards, new robot hip position estimates $\hat{\mathbf{x}}_{new}[t+1]$ are calculated using updated weights. Also, the robot state estimates produced by this block are utilized in the training of the neural locomotion controller block. To do that, hip horizontal position estimates, $\hat{x}_1[t+1]$, are multiplied by the speed correction rate, which is calculated by dividing the target speed $r[t]$ by the actual speed, $v[t]$. Thus, step sizes are scaled according to the speed steady state error. In addition, hip horizontal position estimates, $\hat{x}_1[t+1]$, are multiplied by the speed correction rate, which is calculated by dividing the target speed, $r[t]$, by the actual speed, $v[t]$. Similarly, hip vertical position estimates, $\hat{x}_2[t+1]$, are updated with respect to the ground inclination angle, $\phi[t]$. After that, new hip horizontal position, $\hat{x}_{new,1}[t+1]$, and hip vertical

position, $\hat{x}_{new,2}[t+1]$, estimates are subtracted from scaled hip trajectories to obtain horizontal $e_{r,1}[t+1]$ and vertical $e_{r,2}[t+1]$ position errors for the robot hip. The calculations to find the error, including the feeding of state information from NNBBRM block are represented with the calculation block shown at the top of in the Figure 3.11. After that, another backpropagation is performed on identification and controller neural networks using differences with newly estimated states and scaled estimated states. In this way, a causal adaptation process that is applicable to real life is designed to optimize neural networks, heuristically. This process is stepwise listed in detail in the following.

NNBMRAC Algorithm

1. After forward propagation done for a training pattern, network outputs are obtained for backpropagation.
 2. Calculate $\mathbf{e}_{SID}[t+1] = \mathbf{x}[t+1] - \hat{\mathbf{x}}[t+1]$
 3. $\mathbf{e}_{SID}[t+1]$ is used for backpropagation over Neural Network Based Biped Robot Model block and its parameters are updated. Then, new robot states, $\hat{\mathbf{x}}_{new}[t+1]$ is calculated with updated weights.
 4. Calculate $e_{r,1}[t+1] = (\hat{x}_1[t+1] \cdot \frac{r[T]}{v[T]} - \hat{x}_{new,1}[t+1])$ and $e_{r,2}[t+1] = (\hat{x}_2[t+1] + \hat{x}_1[t+1] \cdot \tan(\phi[t]) \cdot (\frac{r[T]}{v[T]} - 1) - \hat{x}_{new,2}[t+1])$ where $v[t] = \sqrt{\dot{x}_1^2[t+1] + \dot{x}_2^2[t+1]}$ where T index shows the last time instance in the time series.
 5. $\mathbf{e}_r[t+1]$ is used for another error backpropagation through Neural Network Based Biped Robot Model and Neural Controller block for this time. Then weights of Neural Controller is updated.
 6. Another training step starts and after the step finishes, we return to Step 1. (Algorithm continues until the training is terminated)
-

In this subsection, both training and inference neural network calculations are done using GPUs. That is because, in addition to training, inferences done using a combination of controller and identification networks are calculated faster on the GPU compared to the use of dual CPUs available. Small variations on reported success rates and speed steady state error in the previous subsection are observed due to change of computation unit for the neural controller inference calculations. In this scope, walking success rates are found as 84.38%, 85.94%, and 82.65% on training, validation, and test sets, respectively. Similarly, speed steady state error rates are measured as 4.37%, 3.77%, and 3.27% on training, validation, and test sets, respectively. These values constitute the starting values for the NNBMRAC algorithm.

The performance of the proposed NNBMRAAC algorithm is tested under four main scenarios. In the scope of this study, adaptive learning is continued for 41 epoch in each scenario. In the first scenario, Scenario #1, for each walking speed and ground inclination angle combination in training, validation, and test sets, special controller and identification neural networks are assigned for that configuration. Then, algorithm is applied for a walking pattern if the controller neural network successfully completes the walk for that pattern and speed steady state error increased larger than 1%. After that, trained neural networks are stored to be used in the next epoch. Here, we aim to analyze performance of proposed adaptive controller algorithm in the real use conditions by letting learning on validation and test set patterns in simulation environment. Thus, the performance of the adaptive controller is investigated in an online manner.

In Scenario #2, we aim to measure the adaptation ability of the proposed adaptive controller to the weight changes in the robot model. To simulate this, weights of robot limbs are increased by 5%, and this caused an increase in speed steady state error values achieved with the neural controller, as expected. In this context, new walking success rates are found as 81.77%, 81.25%, and 83.67% on training, validation, and test sets, respectively. Similarly, new speed steady state error rates are measured as 5.73%, 5.48%, and 5.51% on training, validation, and test sets, respectively. It is important to note that a 5% weight increase caused a higher speed steady state error increase. The reason for this situation is the training of the neural controller, which is done with a standard weighted biped robot. Instead of repeating the training of the neural controller with different robot weights, we plan to use the NNBMRAAC algorithm to minimize speed steady state error. This problem constitutes a good example to determine the usefulness of the proposed algorithm. Apart from these, adaptive learning is performed under the same conditions in Scenario #1.

In Scenario #3, we focus on contribution of unsuccessful walking attempts done by neural controller throughout the adaptive learning process. To this end, adaptive learning is also applied not only for successful walking attempts, but also for falling walking trials using generated data, until a fall event is detected. Except for relaxing successful walking constraint, same learning conditions are

utilized from Scenario #1. This scenario is proposed to test the importance of falling patterns in adaptive control performance.

In the last scenario, Scenario #4, we investigate the generalization ability of the proposed adaptive controller. For this purpose, we derive a new scenario from Scenario #3 by putting further limitations, and then we analyze the performance of the proposed learning algorithm. First, adaptive learning is done only for training set patterns, speed steady state error rates on training set patterns are recorded, and average performance is reported. Later on, the performance of the trained neural controller at the end of this epoch is reported on validation and test sets. Second, the same controller and identification neural networks are trained for all training patterns, different from all other scenarios. Apart from these changes, adaptive learning is performed under the same conditions in Scenario #3.

Under conditions of these scenarios, the adaptive controller neural networks are trained using system identification neural network with 70 Tanh-231 LSTM-231 LSTM hidden layers reported in Table 3.3. It is important to note that this system identification neural network is trained only with successful walking patterns in the training set. Under different test scenarios, obtained results are presented in Table 3.6.

Table 3.6: Performances of adaptive neural controller training for bipedal 2D locomotion using the SID network, which is trained with only successful walks.

Test Scenario Number	Hidden Layers of Chosen Controller Neural Network	Hidden Layers of System Identification Neural Network	Training Set		Validation Set		Test Set	
			Walking Success	Speed Steady State Error	Walking Success	Speed Steady State Error	Walking Success	Speed Steady State Error
#1	43 LSTM, 74 ReLU	70 Tanh, 231 LSTM, 231 LSTM	79.95%	2.9%	79.69%	2.58%	82.65%	2.42%
#2	43 LSTM, 74 ReLU	70 Tanh, 231 LSTM, 231 LSTM	78.13%	5.03%	79.69%	4.66%	82.65%	4.86%
#3	43 LSTM, 74 ReLU	70 Tanh, 231 LSTM, 231 LSTM	83.33%	3.1%	82.81%	2.81%	84.69%	2.38%
#4	43 LSTM, 74 ReLU	70 Tanh, 231 LSTM, 231 LSTM	84.64%	3.55%	84.38%	2.96%	85.71%	2.32%

Table 3.6 shows us that the NNBMRAAC algorithm can successfully decrease speed steady state error rates for all scenarios. For the speed steady state errors obtained among the scenarios, the lowest training error rate is obtained in Scenario #1, the lowest validation error rate is obtained in Scenario #1, and the lowest test set error is obtained in Scenario #4. Understandably, the highest error rates are obtained for Scenario #2, where the robot limb weights are increased.

The use of falling walking patterns in the adaptive control algorithm seems to be helpful in decreasing test set speed steady state error in Scenario #3 and #4. Lastly, increasing the epoch number appears to be helpful in decreasing the speed steady state error in all scenarios.

The system identification neural network is an indispensable part of the proposed adaptive controller algorithm, and its performance has an essential effect on the success of the adaptive learning that will be performed. For this reason, we want to optimize the system identification neural network training further by including fall patterns in the CPG-driven locomotion dataset, since the use of falling walking patterns in the adaptive control algorithm yields decrement in testing set error rates in Table 3.6. By doing so, we aim to increase its understanding of biped robot model dynamics.

To this end, four system identification neural networks with different first hidden layer types are trained using successful and unsuccessful walking patterns in the generated data set. After that, these identification neural networks are utilized in adaptive controller training under predescribed scenarios. Results are presented in Table 3.7, 3.8, 3.9, and 3.10. To this end, changes of walking success percentages and speed steady state error values are observed as seen in these tables.

Table 3.7: Evaluation of the effect of the first neural layer in system identification neural networks on adaptive control performance under Scenario #1 conditions for bipedal 2D locomotion.

Hidden Layers of Chosen Controller Neural Network	First Layer of System Identification Network Used	Training Set		Validation Set		Test Set	
		Walking Success	Speed Steady State Error	Walking Success	Speed Steady State Error	Walking Success	Speed Steady State Error
43 LSTM, 74 ReLU	70 Tanh Neurons	78.91%	2.82%	78.12%	2.53%	82.65%	2.3%
43 LSTM, 74 ReLU	70 DELU Neurons	78.91%	3.05%	79.69%	2.53%	81.63%	2.53%
43 LSTM, 74 ReLU	70 LeakyRelu Neurons	79.17%	3.02%	81.25%	2.63%	81.63%	2.41%
43 LSTM, 74 ReLU	70 ReLU Neurons	80.21%	2.96%	82.81%	2.83%	81.63%	2.46%

Table 3.6 compares different activation functions in the first hidden layer of the system identification neural network under the conditions of Scenario #1, where only successful walking attempts are utilized in the adaptive controller algorithm.

Among tested activation functions, hyperbolic tangent (Tanh) achieved the highest walking success and the lowest speed steady state error rates. While the test set success rate stayed the same, the speed steady state error decreased to 2.3% from 2.42% compared to Table 3.6. Similarly, there are also speed steady state error rate decreases in training and validation sets. For this scenario, we may conclude that the Tanh activation function gives the best performance among tested activation functions and adding falling patterns to the training of system identification seems helpful.

Table 3.8: Evaluation of the effect of the first neural layer in system identification neural networks on adaptive control performance under Scenario #2 conditions for bipedal 2D locomotion.

Hidden Layers of Chosen Controller Neural Network	First Layer of System Identification Network Used	Training Set		Validation Set		Test Set	
		Walking Success	Speed Steady State Error	Walking Success	Speed Steady State Error	Walking Success	Speed Steady State Error
43 LSTM, 74 ReLU	70 Tanh Neurons	77.6%	4.9%	79.69%	4.71%	82.65%	4.66%
43 LSTM, 74 ReLU	70 DELU Neurons	77.6%	4.82%	79.69%	5.13%	82.65%	4.79%
43 LSTM, 74 ReLU	70 LeakyReLU Neurons	77.08%	4.79%	79.69%	4.64%	82.65%	4.65%
43 LSTM, 74 ReLU	70 ReLU Neurons	76.56%	4.53%	81.25%	4.82%	82.65%	4.78%

Table 3.8 compares different activation functions in the first hidden layer of the system identification neural network under Scenario #2 conditions, where only successful walking attempts are utilized in the adaptive controller algorithm with increased robot limb weights. Among these trials, ReLU, LeakyReLU, and LeakyReLU activation functions achieved the lowest training, validation, and test set speed steady state error values, respectively. For this scenario, we may conclude that LeakyReLU activation function generally gave better performance, and adding falling patterns to the system identification training is beneficial on training and test sets compared to Tanh activation function usage in Table 3.6.

Table 3.9: Evaluation of the effect of the first neural layer in system identification neural networks on adaptive control performance under Scenario #3 conditions for bipedal 2D locomotion.

Hidden Layers of Chosen Controller Neural Network	First Layer of System Identification Network Used	Training Set		Validation Set		Test Set	
		Walking Success	Speed Steady State Error	Walking Success	Speed Steady State Error	Walking Success	Speed Steady State Error
43 LSTM, 74 ReLU	70 Tanh Neurons	84.11%	3.18%	82.81%	2.74%	87.76%	2.23%
43 LSTM, 74 ReLU	70 DELU Neurons	84.11%	3.22%	81.25%	2.49%	85.71%	2.43%
43 LSTM, 74 ReLU	70 LeakyReLU Neurons	84.11%	3.09%	87.5%	2.81%	84.69%	2.36%
43 LSTM, 74 ReLU	70 ReLU Neurons	83.85%	3.13%	82.81%	2.83%	87.76%	2.39%

Table 3.9 compares different activation functions in the first hidden layer of the system identification neural network under Scenario #3 conditions, where both successful and unsuccessful walking attempts are utilized in the adaptive controller algorithm. Among these trials, LeakyReLU, DELU, and Tanh activation functions achieved the lowest training, validation, and test set speed steady state error values, respectively. Since falling patterns are utilized in the adaptive control, some of them were recovered so that success rates are generally increased with all activation functions, and also speed steady state error rates generally decrease compared to 3.8. It is important to note that here the lowest test set speed steady state error rate and the highest test set success rate are acquired with the Tanh activation function among all scenarios. For this scenario, we may conclude that adding falling patterns to the training of system identification and adaptive control is generally beneficial in the sense of increasing walking success and decreasing speed steady state error.

Table 3.10: Evaluation of the effect of the first neural layer in system identification neural networks on adaptive control performance under Scenario #4 conditions for bipedal 2D locomotion.

Hidden Layers of Chosen Controller Neural Network	First Layer of System Identification Network Used	Training Set		Validation Set		Test Set	
		Walking Success	Speed Steady State Error	Walking Success	Speed Steady State Error	Walking Success	Speed Steady State Error
43 LSTM, 74 ReLU	70 Tanh Neurons	83.85%	3.41%	82.81%	3.1%	84.69%	2.36%
43 LSTM, 74 ReLU	70 DELU Neurons	83.07%	3.69%	84.38%	3.01%	83.67%	2.82%
43 LSTM, 74 ReLU	70 LeakyReLU Neurons	83.59%	3.45%	85.94%	3.01%	83.67%	2.59%
43 LSTM, 74 ReLU	70 ReLU Neurons	83.85%	3.57%	85.94%	3.01%	84.69%	2.69%

Table 3.10 compares different activation functions in the first hidden layer of the system identification neural network under Scenario #4 conditions, where both successful and unsuccessful walking attempts are utilized in the adaptive controller algorithm. In addition, a single neural controller and identification network couple is utilized for all patterns to measure the generalization ability of the proposed NNBMRAAC algorithm. Among these trials, Tanh activation function achieved the lowest training and test set speed steady state error values, respectively. For the validation set, DELU, LeakyReLU, and RELU activation functions performed better in the speed steady state error metric than Tanh. Compared to Table 3.9, slightly lower walking success rates and higher speed steady state errors are obtained in general, due to preventing specialization of

utilized controller and identification neural networks by limiting their number to single pair for all patterns. Nevertheless, the proposed NNBMRAC algorithm succeeded in generalizing among all patterns and lower speed steady state error rates. For this scenario, we may conclude that the Tanh activation function generally performed better.



3.3 Three Dimensional Robot Model

In this section, the work on 3D robot is presented in detail. First, the robot model used for this purpose, NICO, is demonstrated. Second, usage of Central Pattern Generator (CPG) is explained. Third, dataset generation is discussed with details. Fourth, Neural Locomotion Controller is explained. Fifth, system identification with neural networks is discussed. Finally, Neural Network-Based Model Reference Adaptive Control is introduced and explained with its details. Note that robot control is based on using angular positions of joints, not the joint torques as done in previous section.

3.3.1 Robot Model Demonstration

The NICO robot model, as illustrated in the figures, is used for this part. It consists of two legs, two arms, a torso and a head as seen in the figure. There are different servos for each joints as seen in the Figure 3.13, and for each feet there are 4 different force and torque sensors, shown by blue prisms in the visuals.

In total, 6 sagittal and 4 frontal joints are used for our version of the NICO robot model. More information and relations of robot components, namely 3D biped robot model and CPG units, are presented in the next section for a coherent delivery.

3.3.2 Central Pattern Generator

For this section, a different CPG network is used. Inspired from the works of [67] and [69], multiple units of the CPG (named as oscillators in this section) are employed for generating walking gaits in the simulation environment. It is modified for our work by immobilizing shoulder and neck oscillators according to robot representation chosen for this section. It basically consists of a pacemaker oscillator connected to other 10 active oscillators which drive the joints, which

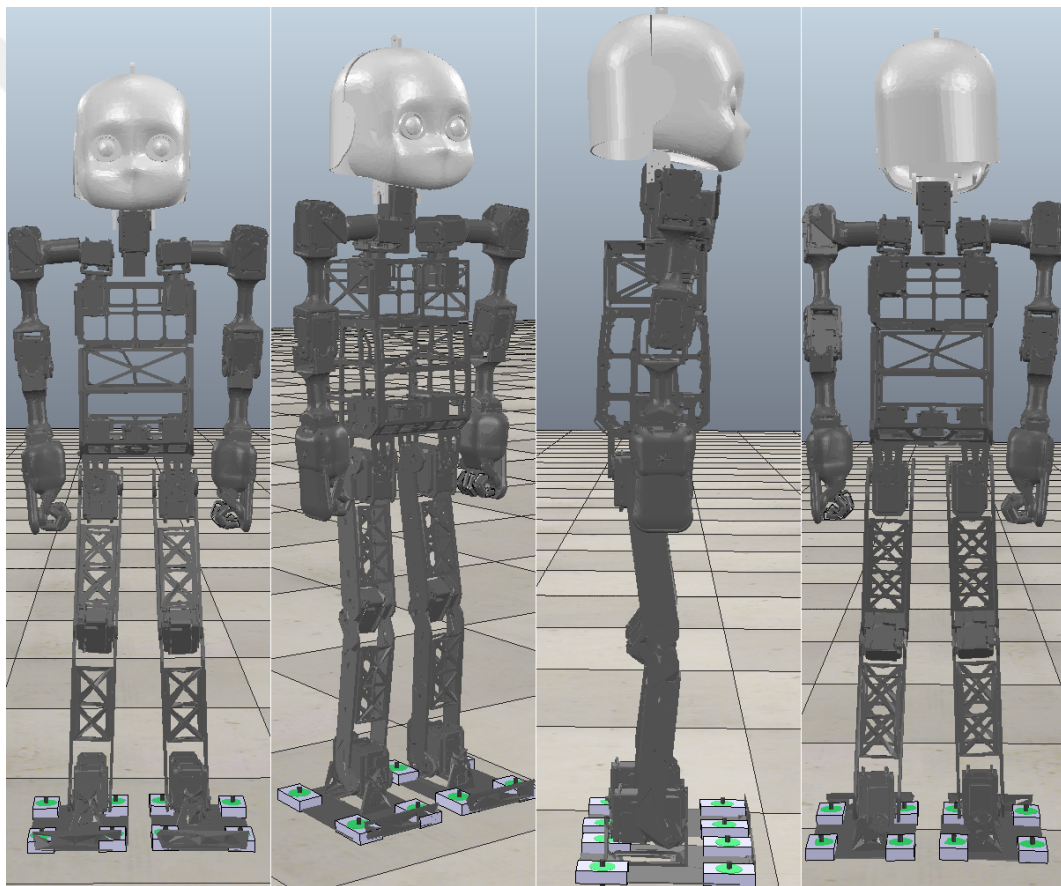


Figure 3.12: NICO robot is viewed from different angles using V-REP simulator. For more information about the robot model, please refer to [67].

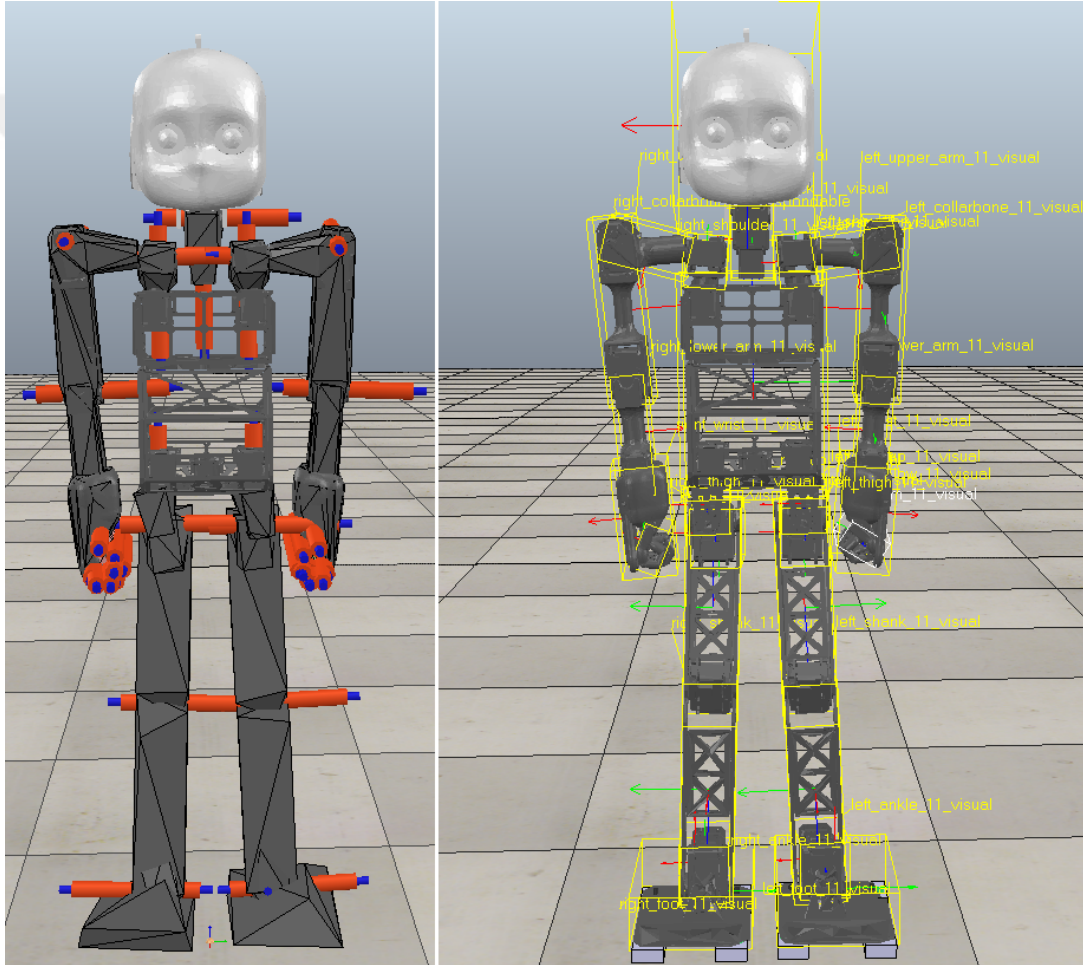


Figure 3.13: The illustrations of NICO, with its servos highlighted on the left and its individual elements such as hands, feet etc. are encapsulated on the right. For more information about the robot model, please refer to [67].

are all separate units of a single CPG. Explanatory equations about the relations of oscillators are presented in (3.11-3.14) and visualized in Figure 3.14.

The pacemaker oscillator weakly determines the speed of walking gait. For the other 10 oscillators, each one of them outputs a value that is multiplied by specific gain amounts. Then, resulting values are summed with specific bias values for sagittal joints whereas for the frontal joints, those bias values are not present. Indeed, in the original work, the state of the robot torso is described as the vector $[\alpha, \beta, \gamma, \dot{\alpha}, \dot{\beta}, \dot{\gamma}, x, y, z, \dot{x}, \dot{y}, \dot{z}]$ which consists of angular position (α, β, γ) and velocity $(\dot{\alpha}, \dot{\beta}, \dot{\gamma})$, and the Cartesian position (x, y, z) and velocity $(\dot{x}, \dot{y}, \dot{z})$ of the robot torso in 3D. This vector is to be fed to high level controller they designed for direction, which is not employed by this thesis. However, the presumed outputs of the high controller, ψ_l and ψ_r are instead set manually from an equally divided interval for our dataset generation. For the case of neural networks, ψ couple is outputted by the network to turn the robot to different directions while walking. ψ_l is associated to u_3 and ψ_r is associated to u_4 as additional gain amounts to turn the robot by changing the speed of their associated joints. Related equations from this work are present in the Appendix, which are partially used in this thesis. For a more detailed explanation of the model, see [69].

NICO robot is fed with low-level control signal from the CPG units and gives a low-level feedback, which is angle feedback, to the CPG units. The target joint angles are vectorized as $[\theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7, \theta_8, \theta_9, \theta_{10}, \theta_{11}, \theta_{12}, \theta_{13}]$ in [69]. However, NICO robot model is specifically modified for the work of this thesis. Its shoulders and neck are nearly immobilized to represent the 3D robot model that is needed for this thesis work, making it effectively a bipedal robot model to be used in the locomotion experiments. Therefore, the important target joint angles effectively becomes the vector of $[\theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7, \theta_8, \theta_9, \theta_{10}, \theta_{11}]$. Low-level feedback of angle works as follows [69]: Actual angular position of each sagittal hip joint is multiplied with weight k and then fed back into corresponding CPG. This is highlighted as better to the other methods in the original work, where each unit of CPG is a Matsuoka oscillator, which are governed using below equations (i stands for ith unit of CPG):

$$\tau'_0 \kappa \dot{u}_i^e = -u_i^e - w_0 y_i^f - \beta v_i^e + u_t + f_i^e + s_i^e \quad (3.11)$$

$$\tau'_0 \kappa \dot{u}_i^e = -v_i^e + y_i^e \quad (3.12)$$

$$\tau'_0 \kappa \dot{u}_i^f = -u_i^f - w_0 y_i^e - \beta v_i^f + u_t + f_i^f + s_i^f \quad (3.13)$$

$$\tau'_0 \kappa \dot{u}_i^f = -v_i^f + y_i^f \quad (3.14)$$

where $y_i^e = \max(0, u_i^e)$ and $y_i^f = \max(0, u_i^f)$ for $i = 1, \dots, 11$ for the case of our implementation of the CPG. State variables $u_i^{e,f}$ control the discharge rate while $v_i^{e,f}$ control the self-inhibitions of the extensor and flexor neurons, as indicated by the first letters respectively [69]. β and w_o are the constants of self-inhibition and mutual-inhibitions respectively, while κ parameter modulates the CPG frequency [69]. Also, $y_i^{e,f}$ are the activations of extensor and flexor neurons which are used for the robot locomotion and $f_i^{e,f}$ are the feedback terms, where $f_i^e = k\theta'_i$ and $f_i^f = -k\theta'_i$ govern the angular feedback relations. Here, the variables defined by the equations $s_i^{e,f} = w_{ij}u_j^{e,f}$ stands for the interaction between connected CPG units [69]. The output, o_i for each CPG unit is calculated as $o_i = -y_i^e + y_i^f$, and then it is multiplied with a gain as shown in the Figure 3.14; for sagittal CPG units, a bias is also added to generate target angles [69]. For o_2 and o_3 outputs related to turn the robot, aforementioned ψ_r and ψ_l values are also used as below.

$$\theta_2 = o_2 \psi_l g_1 + b_1, \theta_3 = o_3 \psi_r g_1 + b_1 \quad (3.15)$$

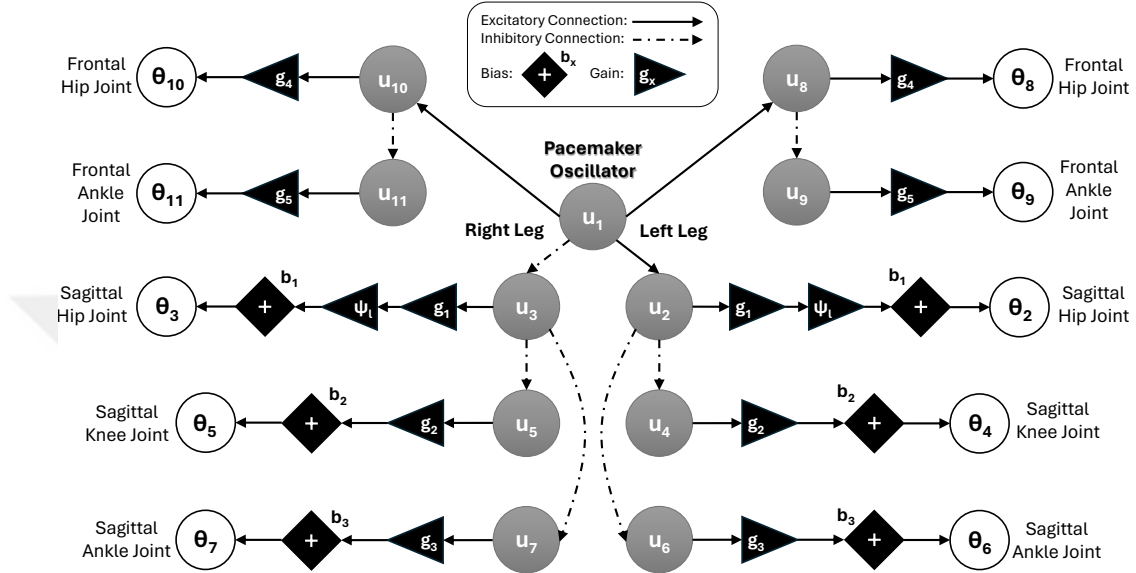


Figure 3.14: The illustration of CPG network used for bipedal locomotion of NICO 3D robot model, inspired from [69].

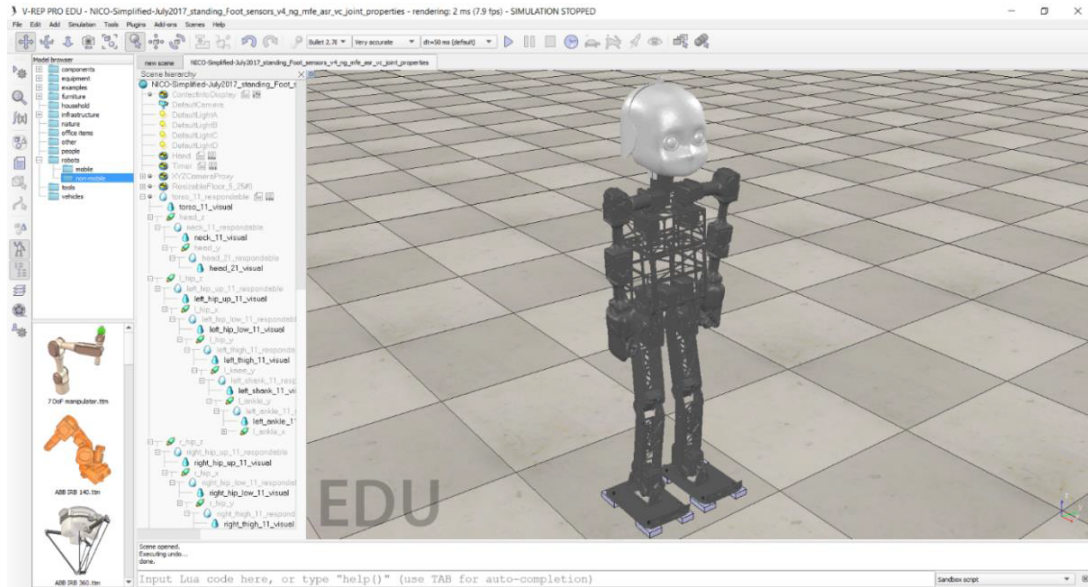


Figure 3.15: NICO robot is standing on a flat surface generated by V-REP.

3.3.3 Dataset Generation

With this purpose of training and testing, a separate dataset is generated using the CPG for cases of NICO 3D robot. These datasets included different walks with different directions of walking and different speeds. The simulation setup can be seen in the Figure 3.15. It basically contains a flat and rigid surface with NICO robot placed in its center, and while the robot model is placed and walking through this experiment setup, biped robot model and CPG values are recorded for generating the dataset.

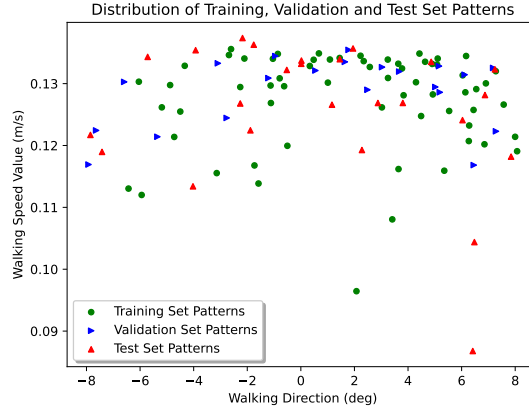


Figure 3.16: The walking dataset generated by CPG, is divided into training, validation and test sets as illustrated for the case of NICO 3D robot.

Using the NICO biped robot model and the specialized CPG presented in previous sections, another dataset is generated to obtain training, validation, and test sets for the case of 3D. These sets are going to be used in the training of neural networks, with a purpose of applying control and system identification for the case of 3D. The dataset contains 109 successful walks, starting from the center (origin of X and Y axes) as 20-second-long walking patterns sampled at 100 Hz, which are produced on flat surface (i.e. X-Y walking plane) with different walking speeds, and targeting different directions. For each data pattern, x-y-z positions of head, torso and 10 servos of the robot, 10 joint torque values, 12 angle position states, 4 ground contact-related forces and 4 torques for each feet with their x-y-z components which make 48 values, fall or successful walk information, walking duration, fitness values according to walking duration and distance, CPG

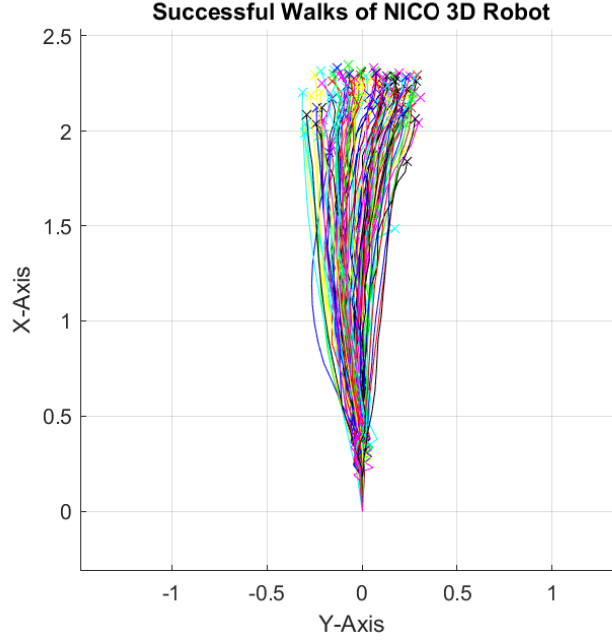


Figure 3.17: The trajectories of NICO robot are displayed 2D, on the X and Y axes for each sample. The movement is projected to the X-Y walking plane, and vertical Z-axis is ignored for the purpose of demonstration.

oscillator values and the offspring values from the genetic algorithm are recorded. Implemented by [67], the genetic algorithm cuts and combines parts of offspring values from different experiments with respect to their successes to generate more successful ones, while passing generations in this endeavor. In addition to these, a tilt value constant, which sets the walking direction by manipulating $\psi_{r,l}$ values as discussed before, is also added to each sample. Lastly, in case when a fall occurs, recording of these values are stopped just after robot falls. For this thesis work, only successful walks are employed and for visualization, the trajectories of the x-y position of robot torso are displayed at Figure 3.17 from our dataset.

Details about dataset and visualization of samples are given in Table 3.11 and Figure 3.16 respectively. Figure 3.16 shows the distribution of training, validation, and test samples distribution. Direction angles are utilized to make the walking attempts more realistic in the 3D simulation environment. Reported walking directions are obtained by changing the orientation values ($\psi_{r,l}$) at equal intervals in the CPG configuration. A non-linear relation exists between the given

input and resulting walking direction for the CPG-controlled robot locomotion. For this reason, some walks (which are not perfectly linear) are observed as seen in Figure 3.17. The robot nature, which is not perfectly symmetrical, makes it difficult to control the direction angles with the CPG and $\psi_{r,l}$ values, and this leads to observed this phenomenon in the Figure 3.17.

For the 3D robot dataset, direction angle interval is chosen as $[-10^\circ, 10^\circ]$. From this interval, 201 evenly spaced values are chosen. Using combinations of these two values, in total, 201 walks are attempted, and 109 of them (54.23%) are successful walks without falling, under the control of CPG. Later on, successful walking patterns are divided into training, validation, and test sets. Amongst those 109 walks, 63 of them (57.80%) are chosen as training, 64 of them (19.27%) are selected as validation, and 25 of them (22.93%) are chosen as test samples.

Table 3.11: Properties of generated 3D robot locomotion data.

Direction angle interval	Total walking attempts amount	Total successful walking amount	Training set walking amount	Validation set walking amount	Test set walking amount
$[-10^\circ, 10^\circ]$	201	109	63	21	25

3.3.4 Neural Locomotion Controller

In this part, we present our studies in applying neural network-based controllers in the closed-loop biped locomotion control problem. Figure 3.18 shows the applied control diagram for this discussion. While doing that, we highlight the effectiveness and weakness of neural network-based elements on the control systems and propose a solution to one of the stated weaknesses.

Figure 3.18 illustrates the “Neural Control Diagram”, showing the details of the control structure. Again, resembling the case of 2D, there are four blocks in the control diagram but this time, joint torques are replaced with joint angles (i.e. angular positions). First, “Walking Environment” block represents the simulation environment where the robot moves on a flat surface according to a walking

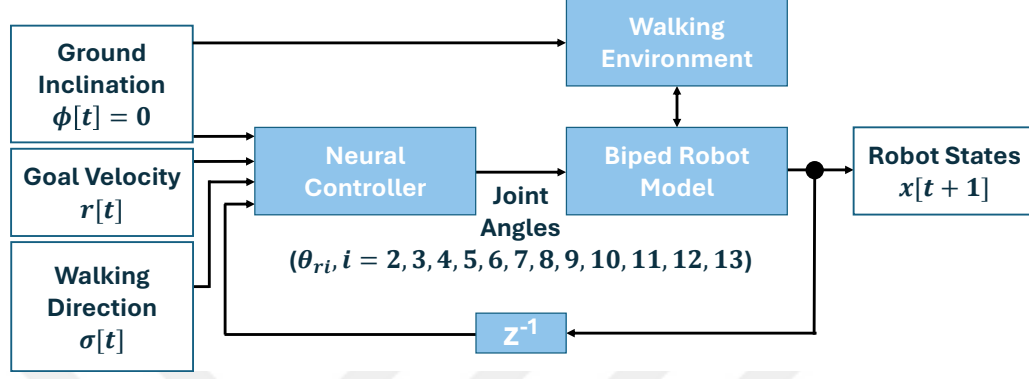


Figure 3.18: Neural Locomotion Controller is presented for 3D case, inspired from [48].

direction reference input, and the robot model interacts with the walking environment during the experiments via ground contact-related forces. Second, “Biped Robot Model” block denotes the NICO robot model which takes twelve target angles $(\theta_{ri}[t], i = 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)$ (last two angles are zeroed, because their joints are immobilized). This block outputs twelve robot limb angle position states at next timestep $(x_i[t], i = 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)$, four ground interaction related forces from sensors at each feet $(F_{gi}[t], i = 1, 2, 3, 4)$ with x-y-z components, making 24 force values, and torques from sensors at each feet $(T_{gi}[t], i = 1, 2, 3, 4)$ with x-y-z components, making 24 torque values, and finally x-y-z positions of robot torso, head and servos, making 36 values, and in total 97 values of outputs. In this configuration, a large amount of information about NICO robot states is obtained, in fact robot fall status is also determined using these values. The third block is “Neural Controller” which generates twelve joint angles $(\theta_{ri}[t], i = 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)$ with respect to 100 input values. These values are taken walking direction reference input $(\sigma[t])$, zeroed ground inclination input $(\phi[t] = 0)$, constant goal velocity value $(r[t])$. In addition these three values, 12 robot limb angle position states $(x_i[t], i = 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)$, ground interaction related forces from sensors at each feet $(F_{gi}[t], i = 1, 2, 3, 4)$ with x-y-z components as 24 values, torques from sensors at each feet $(T_{gi}[t], i = 1, 2, 3, 4)$ with x-y-z components as 24 values, and 36 values consisting of x-y-z positions of robot torso, head and

servos are fed to neural controller, adding up to 99 values. The last one is the unit time step delay block. The “Neural Controller” block takes inputs through the delay block other than walking direction reference input, ground inclination and goal velocity reference inputs.

Here, the neural controller block includes an artificial neural network that is trained with the generated training dataset, and it is utilized to locomote the robot model. Such neural controllers can be found helpful to satisfy stable locomotion in wider walking direction reference input, compared to CPG utilized in dataset generation. Studies in the scope of this thesis show these types of neural controllers can cause speed steady-state errors. In this context, we investigate the effects of different neuron models and network architectures on walking success and speed steady-state error.

The neural controller block consists of a neural network with two hidden layers. Similar to the network used in 2D neural torque controller, a 43 LSTM-74 ReLU network is trained successfully for being used as the neural controller for this case. There are ten output neurons corresponding to joint angles. AdamW optimizer is used for these networks with 0.001 as the learning rate constant at the beginning. Results are obtained by training and running inference using in GPUs only, and the training lasted 20000 epochs.

In the Table 3.12, results of this neural network architecture are given comparatively with the results of 43 LSTM-74 DELU neural network. However, 43 LSTM-74 DELU network did not learn much and became unsuccessful in its training, in comparison to 43 LSTM-74 ReLU network. Therefore, the 43 LSTM-74 ReLU neural network is used in the Neural Network Based Model Reference Adaptive Control (NNBMRAC) as the neural controller.

Table 3.12: Results of neural controller trained with 3D robot locomotion data.

Neural Network Hidden Layers	Best Epoch	Training Set			Validation Set			Test Set		
		Success Rate (%)	Speed Steady State Error (%)	Walking Direction (°) Error (RMSE)	Success Rate (%)	Speed Steady State Error (%)	Walking Direction (°) Error (RMSE)	Success Rate (%)	Speed Steady State Error (%)	Walking Direction (°) Error (RMSE)
43 LSTM, 74 ReLU	42500	65.08	14.48	6.85	76.19	14.63	8.98	48.00	12.69	7.56
43 LSTM, 74 DELU	35000	31.75	54.46	18.55	47.62	45.90	17.20	24.00	46.11	15.26

3.3.5 System Identification With Neural Networks

In this part, we used neural networks in the system identification block for the case of 3D locomotion, similar to previous sections. Like the case of 2D locomotion, Universal Approximation Theorem implies the capability of neural networks to approximate functions with arbitrary accuracy. Therefore we expect that highly nonlinear biped 3D robot model can be approximated as well. Parallel system identification scheme employed for this purpose is given in Figure 3.19.

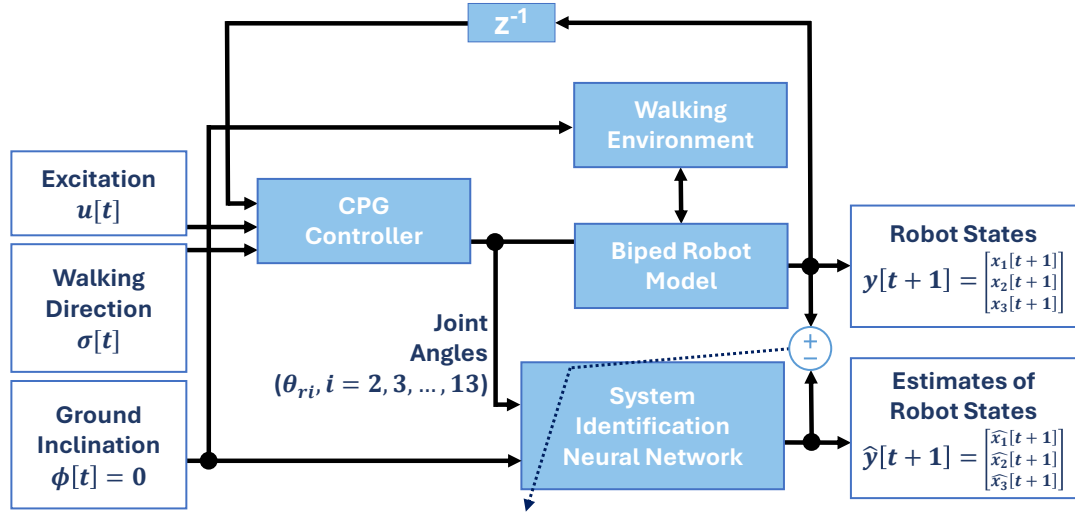


Figure 3.19: Parallel system identification configuration is presented, inspired from [48].

Resembling the previous parts, there are three main blocks in this scheme. First, a neural CPG controller takes the excitation and previous robot states as inputs and outputs the joint angles. Biped Robot Model takes these angles and walking direction to move the robot joints, by interacting with the walking environment and outputs the complete set of robot states; only the robot location components in x-y-z axes are used by this scheme are labeled as “robot states”. The third block, System Identification Neural Network takes the ground inclination and joint angles commanded by the controller to estimate the x, y and z components (interchangeably x_1 , x_2 and x_3) of robot location. The error

obtained in this estimation is used for backpropagation of this neural network, with an aim to make the network and approximator for the input and output relationship of the biped 3D robot model, similar to system identification case for 2D robot. Prediction error is calculated as the difference between the actual position of NICO biped robot and estimate of system identification network, and the loss is calculated with x, y and z position differences using the Mean Squared Error (MSE) definition and summed up cumulatively, adding x, y and z positions of the hip, and halving the resulting value.

Outputs of the system identification block, which are x, y and z positions as states ($x_1[t]$, $x_2[t]$ and $x_3[t]$, respectively), are listed inside $\mathbf{y}[t]$ vector. The inputs are normalized joint (servo) angles setted for the robot and zeroed ground inclination angle ($\phi(t) = 0$); making a total amount of 13 values (including 2 immobilized joint angles). The equation used to calculate the MSE loss metric is explained below:

$$\mathbf{y}[t] = \begin{bmatrix} x_1[t] \\ x_2[t] \\ x_3[t] \end{bmatrix} \quad (3.16)$$

$$\hat{\mathbf{y}}[t] = \begin{bmatrix} \hat{x}_1[t] \\ \hat{x}_2[t] \\ \hat{x}_3[t] \end{bmatrix} \quad (3.17)$$

$$\mathbf{e}[t] = \mathbf{y}[t] - \hat{\mathbf{y}}[t] = \begin{bmatrix} x_1[t] - \hat{x}_1[t] \\ x_2[t] - \hat{x}_2[t] \\ x_3[t] - \hat{x}_3[t] \end{bmatrix} \quad (3.18)$$

$$l[t] = \frac{\mathbf{e}[t]^T * \mathbf{e}[t]}{3} \quad (3.19)$$

$$L = \sum_{t=1}^N \frac{l[t]}{N} \quad (3.20)$$

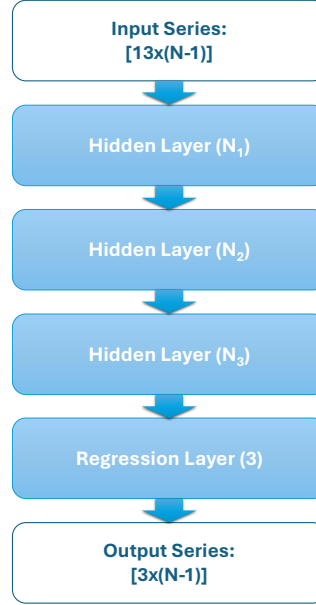


Figure 3.20: The neural network structure used for system identification network for 3D robot simulations.

In this section, the dataset generated with neural CPG controller for this purpose is used, as explained in previous sections. The best network amongst all networks tried is the 140 DELU-231 LSTM-231 LSTM network by the test loss scaled mean values. In this example, DELU managed to outperform ReLU and LeakyReLU activation functions by a significant amount of decrease on the error metric. Also, the first entry of the table, 70 LeakyReLU-231 LSTM-231 LSTM network is taken from an existent thesis work for comparison purposes [48]. The trainings are done with a minibatch size of 64, AdamW optimizer, no regularization, and 0.01 Dropout rate.

Table 3.13: System identification network results are given in comparison.

Neural Network Hidden Layers	Best Epoch	Training	Validation	Test
		Loss Scaled Mean	Loss Scaled Mean	Loss Scaled Mean
70 LeakyReLU, 231 LSTM, 231 LSTM [48]	53000	$5.57 \cdot 10^{-4}$	$1.74 \cdot 10^{-3}$	$3.4 \cdot 10^{-3}$
140 LeakyReLU, 231 LSTM, 231 LSTM	46000	$5.9 \cdot 10^{-5}$	$1.9 \cdot 10^{-3}$	$3.62 \cdot 10^{-3}$
70 ReLU, 231 LSTM, 231 LSTM	114000	$7.16 \cdot 10^{-5}$	$1.92 \cdot 10^{-3}$	$3.11 \cdot 10^{-3}$
140 ReLU, 231 LSTM, 231 LSTM	73000	$4.93 \cdot 10^{-4}$	$1.42 \cdot 10^{-3}$	$3.06 \cdot 10^{-3}$
70 DELU, 231 LSTM, 231 LSTM	62000	$7.91 \cdot 10^{-4}$	$1.67 \cdot 10^{-3}$	$2.75 \cdot 10^{-3}$
140 DELU, 231 LSTM, 231 LSTM	26000	$1.07 \cdot 10^{-3}$	$1.78 \cdot 10^{-3}$	$2.25 \cdot 10^{-3}$

3.3.5.1 Tuning of DELU Activation Function

As an expansion to this section, DELU activation function is tuned to reach better success rates for the system identification of 2D biped robot simulation. For this tuning, 140 DELU-231 LSTM-231 LSTM network architecture is chosen. Similar to the previous section in this chapter, the tuning is done by varying the β parameter of the activation function between 1.25 and 3.25 values. Here, α value is fixed as 1 and z_c value is found using (2.11) with these α and β values. Resulting neural networks are tested as seen in the Table 3.14 given below. Here, original DELU is compared with same eight DELU activation functions which are configured as in the previous section at 2D robot case. To remind, DELUvb1.25 stands for DELU activation function tunings with its β value is set to 1.25 instead of the original β value, 2.

Table 3.14: System identification results are given in comparison for DELU activation function tunings.

Neural Network Hidden Layers	Best Epoch	Training	Validation	Test
		Loss Scaled Mean	Loss Scaled Mean	Loss Scaled Mean
140 DELUvb1.25, 231 LSTM, 231 LSTM	63000	$1.68 \cdot 10^{-4}$	$1.54 \cdot 10^{-3}$	$3.29 \cdot 10^{-3}$
140 DELUvb1.5, 231 LSTM, 231 LSTM	63000	$1.61 \cdot 10^{-5}$	$1.26 \cdot 10^{-3}$	$2.46 \cdot 10^{-3}$
140 DELUvb1.75, 231 LSTM, 231 LSTM	18000	$7.79 \cdot 10^{-4}$	$1.49 \cdot 10^{-3}$	$2.48 \cdot 10^{-3}$
140 DELU, 231 LSTM, 231 LSTM	26000	$1.07 \cdot 10^{-3}$	$1.78 \cdot 10^{-3}$	$2.25 \cdot 10^{-3}$
140 DELUvb2.25, 231 LSTM, 231 LSTM	69000	$5.59 \cdot 10^{-4}$	$1.32 \cdot 10^{-3}$	$2.68 \cdot 10^{-3}$
140 DELUvb2.5, 231 LSTM, 231 LSTM	35000	$7.05 \cdot 10^{-5}$	$1.32 \cdot 10^{-3}$	$3.62 \cdot 10^{-3}$
140 DELUvb2.75, 231 LSTM, 231 LSTM	65000	$1.15 \cdot 10^{-4}$	$1.88 \cdot 10^{-3}$	$2.59 \cdot 10^{-3}$
140 DELUvb3, 231 LSTM, 231 LSTM	110000	$3.00 \cdot 10^{-5}$	$1.31 \cdot 10^{-3}$	$3.12 \cdot 10^{-3}$
140 DELUvb3.25, 231 LSTM, 231 LSTM	53000	$1.19 \cdot 10^{-4}$	$1.58 \cdot 10^{-3}$	$2.78 \cdot 10^{-3}$

As presented in the table, for training and validation success rates, DELUvb1.5 has the least loss value. However, for the test losses original DELU has the least loss value in comparison to other DELUs. Obtained results does not change the leading entry of the Table 3.3 for test success rates, but for training and validation rates, DELUvb1.5 would have the best results in the Table 3.14. In result, another application of tuning the DELU activation function for this section, and this time the result obtained with original DELU stayed as the entry with least loss value for test success rates in both tables.

3.3.6 Neural Network Based Model Reference Adaptive Control

This section introduces Neural Network Based Model Reference Adaptive Control (NNBMRAC) for the case of 3D robot locomotion. Here, neural network trainings and inferences are completed using GPUs. Figure 3.21 presents the scheme of NNBMRAC with details of implementation. It is designed to apply adaptive control using different neural network blocks in various roles for fine-tuning the walking performance and accuracy; lowering the speed steady state error and walking direction errors caused by the neural controller block.

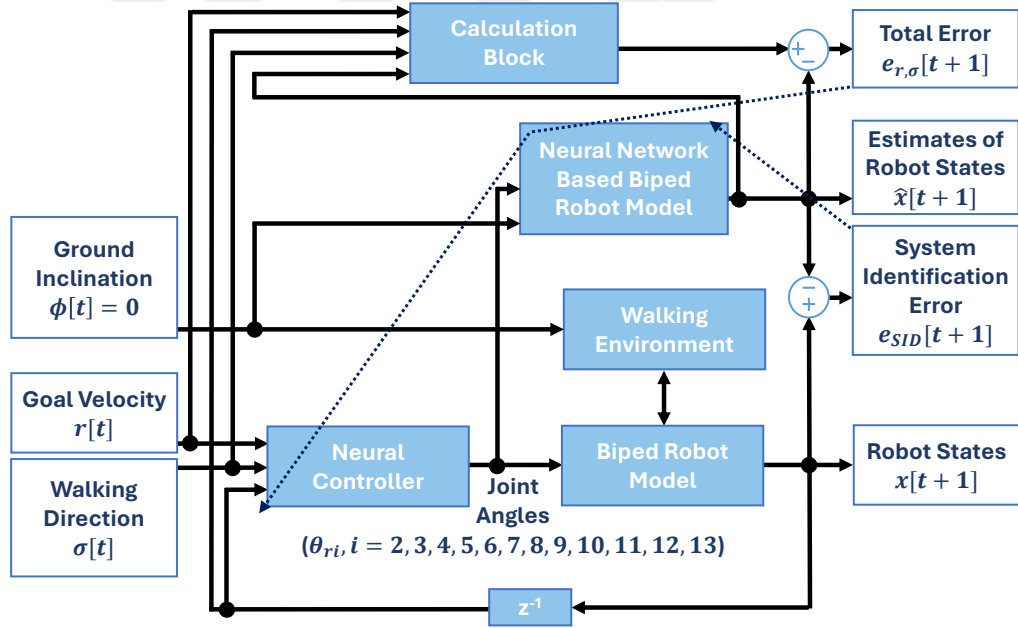


Figure 3.21: The control diagram with neural locomotion controller and neural network based biped robot model from the system identification part for the case of NICO 3D robot.

In this part, we focus on decreasing deviation in walking speed from goal velocity and the angle of deviation in the walking direction from the target walking angle. As explained previously, there is a steady state speed error in the speed, which we aim to decrease by updating the controller neural network parameters with the well-known backpropagation algorithm on neural network blocks. Again, passing the gradient over the biped robot model in Figure 3.21 is not

feasible due to the nonlinear and hybrid dynamics of the 3D robot model, similar to the 2D robot case. To implement adaptive control, we add a new feedback loops to calculate the walking speed and direction deviation error, $e_{r,\sigma}[t + 1]$ into control diagram in the Figure 3.18, as illustrated in Figure 3.21. Since it is easier to backpropagate the error gradient over the neural network than the mathematical biped 3D robot model, the system identification block with Neural Network Based Biped Robot Model name is added into the diagram to acquire robot state estimates in a way that is similar to the 2D robot locomotion control shown in Figure 3.11. Again, we assume that system identification network approximates the input-output relationship of the robot model close enough, such that the propagated error gradient will be able to update the parameters of neural controller to minimize the error.

Control diagram in Figure 3.21 shows the placement of the neural controller and system identification blocks. Inputs and outputs of these blocks are explained in previous subsections, 3.3.4 and 3.3.5. The system identification block produces estimates of robot states which are employed to calculate error metric; the calculations to find the error, including the feeding of state information from NNBBRM block are represented with the calculation block shown at the top of in the Figure 3.21. Here, we compare system identification neural network-estimated states with actual robot states calculated with robot equation of motions and calculate errors using mean squared error (MSE) metric as explained in Subsection 3.3.5. These errors are used for backpropagation on neural network based biped robot model which consists from the system identification neural network as explained in the previous part.

Also, the robot state estimates produced for this block are also used for the backpropagation on the neural locomotion controller block by subtracting the estimated robot states from a corrected robot state trajectory in the sense of walking speed and direction. In this manner, backpropagation for both network blocks are made. Controller and identification neural networks block are trained with $e_{r,\sigma}[t + 1]$ and $e_{SID}[t + 1]$ error definitions in an online manner under the conditions presented in Scenario #3 in Subsection 3.2.6. In essence, a process similar to the NNBMRAAC algorithm explained in Subsection 3.2.6 is utilized.

Differences from it can be seen in the pseudocode given below.

NNBMRAC Algorithm

1. After forward propagation done for a training pattern, network outputs are obtained for backpropagation.
2. Calculate $\mathbf{e}_{SID}[t+1] = \mathbf{x}[t+1] - \hat{\mathbf{x}}[t+1]$
3. If the speed steady state error is less than 1% and direction deviation error is less than 1 degree, directly pass to the State 8.
4. $\mathbf{e}_{SID}[t+1]$ is used for backpropagation over Neural Network Based Biped Robot Model block and its parameters are updated. Then, new robot states, $\hat{\mathbf{x}}_{new}[t+1] = \begin{bmatrix} \hat{x}_{new,1}[t+1] \\ \hat{x}_{new,2}[t+1] \\ \hat{x}_{new,3}[t+1] \end{bmatrix}$ is calculated with updated weights.
5. Calculate direction deviation error $e_\sigma[t+1] = \sigma[t+1] - \arctan(-\frac{x_2[t+1]}{x_1[t+1]})$ for $\phi[t] = 0$.
6. For $\phi[t] = 0$,

$$\mathbf{e}_{r,\sigma}[t+1] = R(z, e_\sigma[T]) \begin{bmatrix} \hat{x}_{new,1}[t+1] \cdot r[T]/v[T] \\ \hat{x}_{new,2}[t+1] \cdot r[T]/v[T] \\ \hat{x}_{new,3}[t+1] \end{bmatrix} - \hat{\mathbf{x}}_{new}[t+1] \text{ where}$$

$v[t] = \sqrt{\dot{x}_1^2[t+1] + \dot{x}_2^2[t+1] + \dot{x}_3^2[t+1]}$, R is the rotation matrix around z axis with $e_\sigma[T]$ angle, T index shows the last time instance in the time series.

7. $\mathbf{e}_{r,\sigma}[t+1]$ is used for another error backpropagation through Neural Network Based Biped Robot Model and Neural Controller block for this time. Then, weights of Neural Controller is updated.
 8. Another training step starts and after the step finishes, we return to Step 1. (Algorithm continues until the training is terminated)
-

In this subsection, both training and inference neural network calculations are done using GPUs like the Subsection 3.2.6. The reason is similar; in addition to training, inferences done using a combination of controller and identification networks are calculated faster on the GPU compared to the use of only CPU.

It is important to note that 3D robot model simulations are done in V-REP simulators in real-time as much as possible at 20 Hz sampling frequency. At each time step, required input parameters for neural controller calculations are retrieved from the simulator, and forward propagation is done in a neural network to determine the next goal joint angles. After that, acquired target joint angles are sent to the robot model. Since this process is completed with varying time durations, some differences emerge in performance metrics such as success rate,

walking speed, and direction between runs performed under the same initial conditions. This situation makes it hard to analyze the performance of the proposed algorithm.

One instance of implementing adaptive control to the 3D biped robot is summarized in Table 3.15. Here, iterations are listed with epoch numbers. Epoch 1 constitutes the starting values for the NNBMRAc algorithm using the neural controller with 43 LSTM and 74 ReLU hidden layers found in Subsection 3.3.4. Unfortunately, it is not possible to take exactly the same performance measurements from robot model simulation since the simulation is done in real time. For this reason, there are differences between values in Table 3.12 and Epoch 1 in Table 3.15. For the choices of system identification neural network in the adaptive control system configuration, 140 ReLU-231 LSTM-231 LSTM network and 140 DELU-231 LSTM-231 LSTM network are separately used from the previous subsection, whose results are presented in Table 3.15 and Table 3.16, respectively. The performance of the proposed NNBMRAc algorithm is tested after 10 epochs of adaptive training. Gradient descent is used as the learning algorithm with a learning constant equal to 0.00001 and without Dropout or L2 regularization for the results at both tables.

Table 3.15: Change of performance metrics throughout adaptive control of 3D robot locomotion using 140 ReLU neurons in the first hidden layer of system identification neural network.

Epoch Number	Training Set			Validation Set			Test Set		
	Success Rate (%)	Speed Steady State Error (%)	Walking Direction (°) Error (RMSE)	Success Rate (%)	Speed Steady State Error (%)	Walking Direction (°) Error (RMSE)	Success Rate (%)	Speed Steady State Error (%)	Walking Direction (°) Error (RMSE)
1	82.54	18.64	6.59	80.95	20.82	7.47	80.00	17.93	6.42
5	79.37	19.28	5.21	71.43	20.54	6.73	80.00	19.04	4.88
10	61.90	20.58	6.00	71.43	20.32	7.75	80.00	18.09	5.57

For the results with the usage of system identification neural network that includes ReLU layer, the deviation from the target direction is lowered by 8.95% in previous training and 13.24% in previous test sets; however, it is increased by 3.75% in the previous validation set as seen in Table 3.15. In addition to that, speed steady state error is lowered by 2.4% in previous validation set; however, it is increased by 10.41% in the previous training set and 0.89% in the previous test set as seen in Table 3.15.

Table 3.16: Change of performance metrics throughout adaptive control of 3D robot locomotion using 140 DELU neurons in the first hidden layer of system identification neural network.

Epoch Number	Training Set			Validation Set			Test Set		
	Success Rate (%)	Speed Steady State Error (%)	Walking Direction (°) Error (RMSE)	Success Rate (%)	Speed Steady State Error (%)	Walking Direction (°) Error (RMSE)	Success Rate (%)	Speed Steady State Error (%)	Walking Direction (°) Error (RMSE)
1	77.78	19.19	5.28	95.24	19.85	5.89	84.00	19.18	6.69
5	69.84	18.44	5.55	71.43	18.41	6.19	80.00	18.91	6.41
10	61.90	18.96	5.31	80.95	18.53	5.62	72.00	19.17	5.43

For the results with the usage of system identification neural network that includes DELU layer, speed steady state error is lowered by 1.2% in previous training set, 6.65% in previous validation set, and 0.05% in previous test sets as seen in Table 3.16. In addition to that, the deviation from the target direction is lowered by 4.58% in previous validation, 18.83% in previous test; however, it is increased by 0.57% in the previous training set as seen in Table 3.16.

For both tables, walking success rates fluctuate as the learning progresses, which can be observed in previous training, previous validation and previous test set results - except previous test set results at Table 3.15, where it stays as constant at 80%.

Depending on our observations for both tables, these values are prone to change for each run, but in general there is a trend of decreasing speed steady state and walking direction errors. To decrease variations in simulation results, the simulation should be stopped after performing calculations of robot equations of motions, and required neural network calculations should be done. After that, the simulation should be continued, so that results will have less variance and interpretation will be easier. This improvement is left as a future work.

3.4 Physical Biped Robot System Development

In this section, the physical biped robot system is introduced. The system consists of three main elements: robot, planarizer and experiment track. Using this system, multiple experiments are completed to form a dataset and make system identification with neural networks, which are explained in the next sections with details.

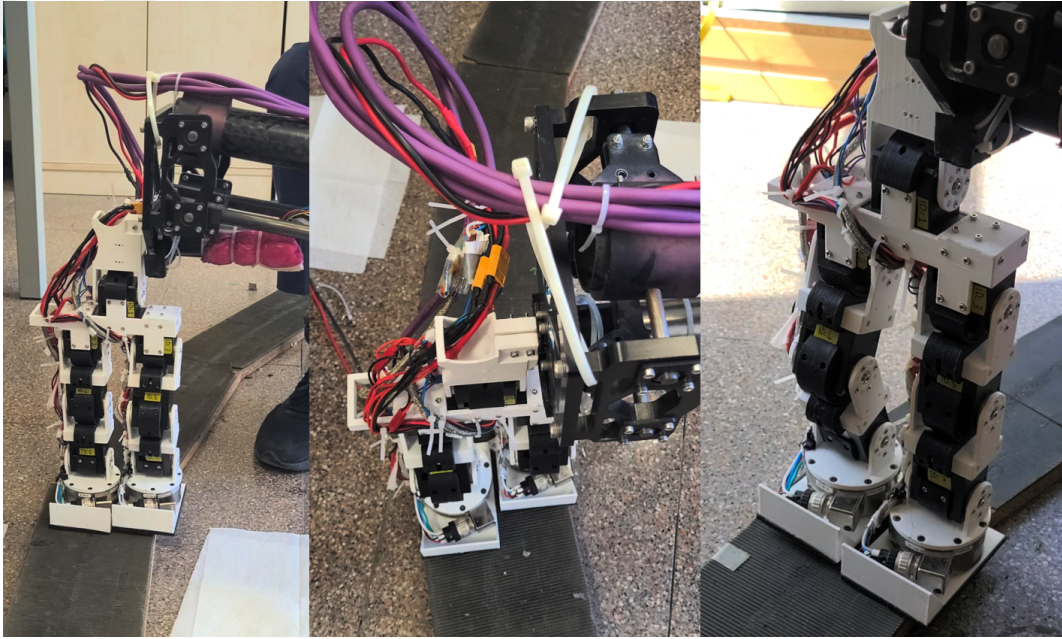


Figure 3.22: Photographs from the experiments with 2D physical biped robot.

Our two-legged, biped physical robot is shown in action at Figure 3.22, more details about robot structure are presented in Section 3.4.1. Robot is connected to a planarizer for make 2D locomotion possible, as seen in Figure 3.23. Planarizer logs the necessary information about the movements and sends them to the single board computer for calculations and loggings. Robot walks by itself, without any actuation from the planarizer - in fact the robot also carries a partial weight due to physical connection of the planarizer. Both robot and planarizer had to be used to complete the experiments for the purposes of this thesis work.

Robot moves on the experiment track, which is shown in the Figure 3.24 as the bigger gray path. The track surface is a complete circle. Apart from a few

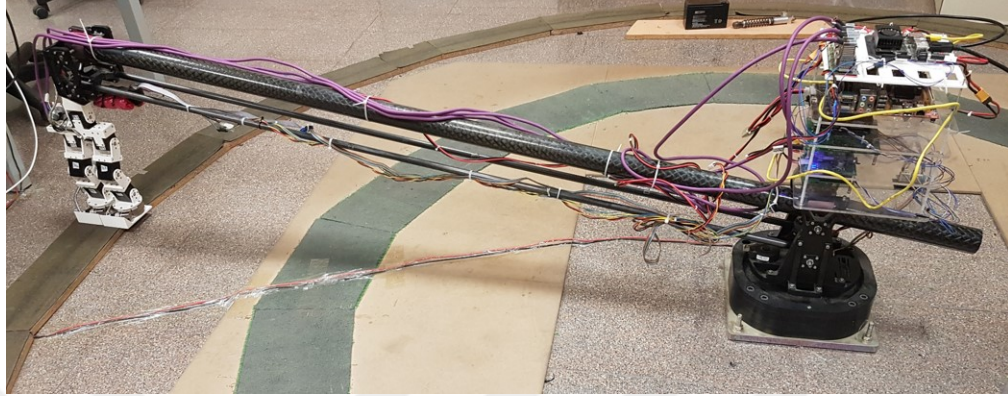


Figure 3.23: The 2D planarized robot with the connected system is shown on its track.

wearied points on the rubber surface, it is completely flat. The dataset obtained in Section 3.4.2 is generated using this experiment system, by choosing different start points to create various walking patterns.

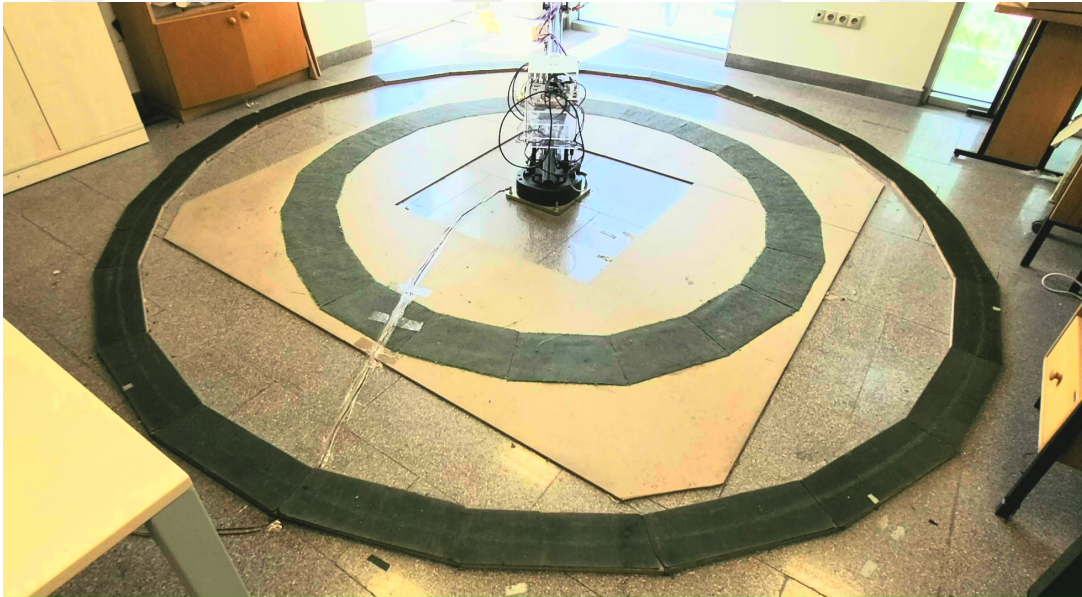


Figure 3.24: The complete track used for robot locomotion is shown in this photo.

3.4.1 Biped Robot System

Inspired from the works of bipedal motion by Taga and his colleagues, the biped robot system is designed by CAD software and consists of different components. Main components related to actuation and sensing are servos and sensors, respectively. Their specific details are explained in the next subsections. In Figure 3.25, CAD drawings of the robot model can be viewed from several angles.

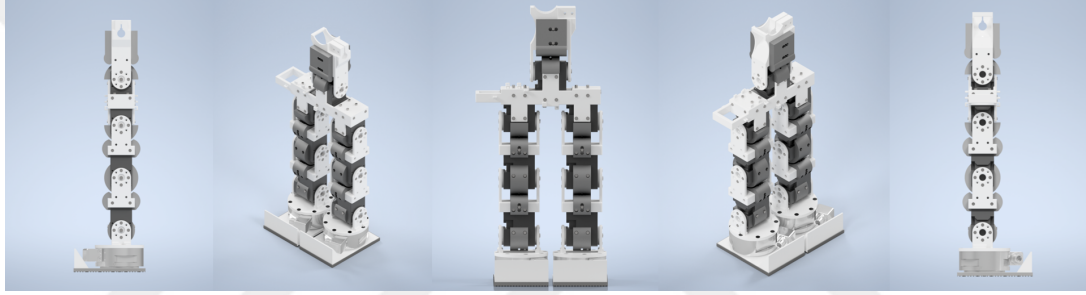


Figure 3.25: CAD drawings of the 2D physical biped robot model.

3.4.1.1 Robot Servos

At each joint of a robot, there should be servos for making actuation and robot locomotion possible. In our case, Dynamixel XH540-W150-R servos are chosen as the robot servo type. These servos are capable of communicating with 4.5 million bits per second baud rate, they can measure their temperature and input voltage values. This type of servo contains a PID control block in its circuitry as shown in the Figure 3.26, for operating in different modes for movement. In our case, position control mode is employed for robot locomotion.

XH540-W150-R servos have ROM and RAM spaces where important information are held. Goal position, present velocity and present position values are written in these areas by multiple bytes. A screenshot from official interface application shows those specialties of servos in Figure 3.27. Our code at the single board computer communicates with servos to set and obtain values for goal and present variables, respectively. For the position control mode, some variables are crucially important. PID and feedforward gains, goal position and current,

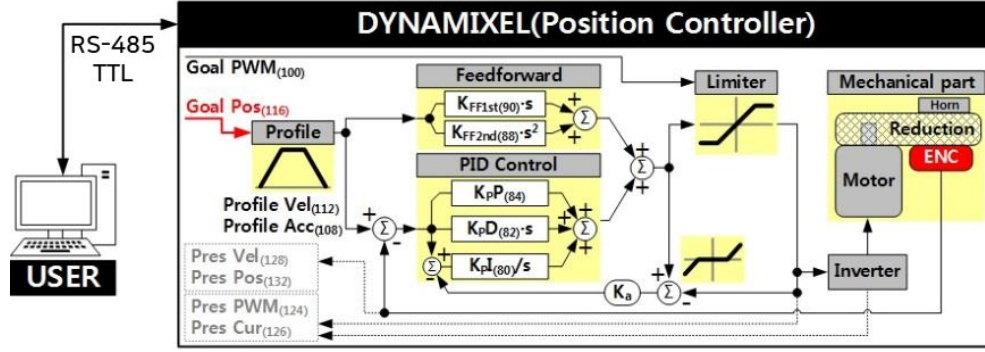


Figure 3.26: The circuit is making position control using PID, inside a Dynamixel XH540-W150-R servo [77]

present velocity, current and position values are checked in every cycle of our control loop. According to walking trajectory, specific goal positions are sent to the servos as instructions. Meanwhile, current velocity, current and position values are obtained and recorded to the single board computer.

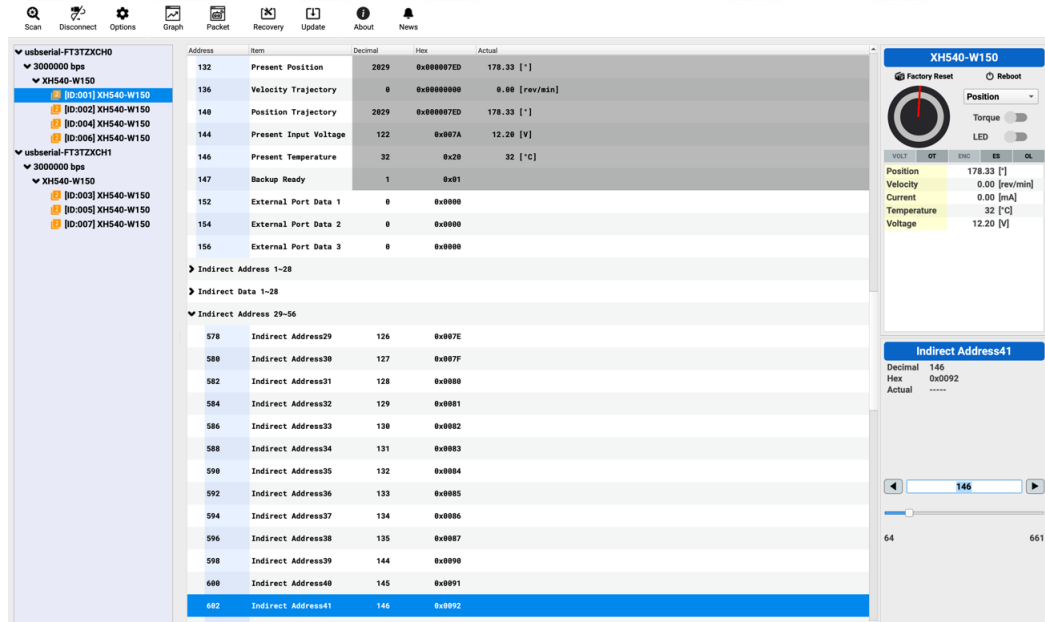


Figure 3.27: Some of RAM addresses for ID 1 servo are displayed by the program.

In total, 7 servos are used for the 2D biped robot. From ID 1 to ID 7, their exact positions in the robot structure can be viewed from the Figure 3.25. ID 1 stands for the servo at top of the robot, connected to the planarizer motor shaft. To remind again, planarizer motor is not driven throughout the experiments.

Other servos correspond to hip, knee and ankle joints as seen in the Figure 3.25.

3.4.1.2 Robot Sensors

At each feet of the robot, a force and torque sensor is placed for giving sensing abilities to the robot system. Specifically, ATI-IA Gamma SI-130-10 Force/Torque Sensor is chosen as the sensor type for our robot. This sensor is capable of generating 3-axis force and torque vectors, and while communicating with 115200 bits per second baud rate, 500 samples are sent from these sensors per second in our robot configuration.



Figure 3.28: The sensor is shown from the top at left image, and the places of sensors at the robot feet is shown at the right.

Sensors are addressed by our code at single board computer, which obtains the force and torque values to record them. Before each experiment, both sensors are calibrated by rising each leg from the ground individually, for increasing the accuracy of measurements. More information about the sensor data are present in the Section 3.4.2.

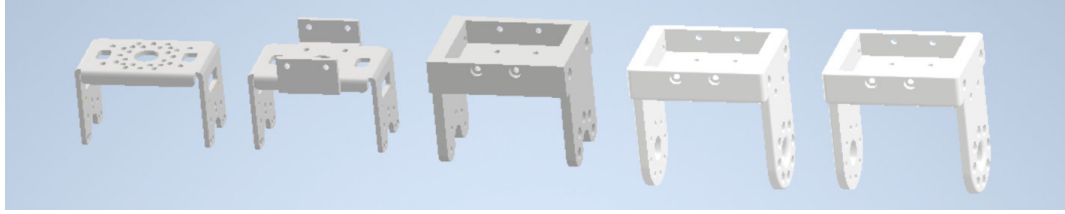


Figure 3.29: Iteratively designed 3D parts, earliest is shown at the left and latest at the right.

3.4.1.3 Other Components

To make the robot as a whole, multiple connections are required. Robot is assembled by using bolts, cables and 3D parts which are printed to meet specific criteria. To create the optimal connection parts, an iterative production process is followed for 3D parts, as seen in the Figure 3.29. According to the manuals of servos and sensors and physical constraints, TPU and PLA materials are used for printing several components with CAD software. Due to current distribution requirements, coaxial cables and proper sockets are used for lowering the resistance values. Resulting robot structure can be seen in Figure 3.30.

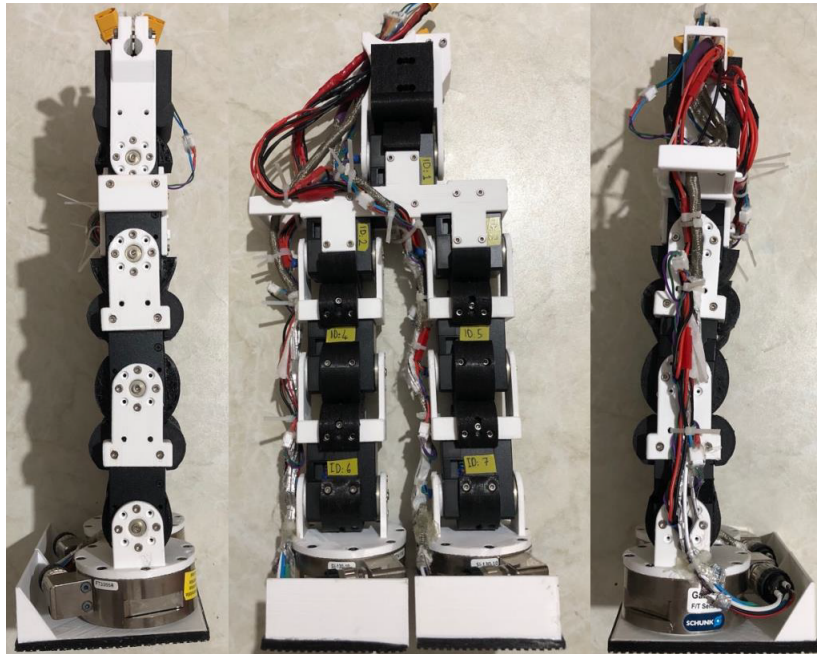


Figure 3.30: The finalized version of our 2D planarized robot.

3.4.2 Dataset Generation

With the purpose of training and testing, a dataset is generated using the walking trajectories generated for the locomotion robot, with inspirations from ZMP and CoG algorithms, and by fine-tuning the PID controllers of the servos. This dataset includes multiple walks with various speeds. While the robot is placed and walked in this experiment setup described previously, biped robot and planarizer are logging values into single board computer for generating a data sample for each walking pattern.

For each pattern, logging is made with a frequency of 20 Hz. The speed of locomotion is resulting from walking trajectories scanning from 0.5x to 1.5x, when speed coefficient for the reference walking gait which is considered as 1x. General information about the dataset are presented in Table 3.17.

Table 3.17: Properties of generated 2D robot locomotion data from real-life.

Total successful walking amount	Walks chosen for the dataset	Training set walking amount	Validation set walking amount	Test set walking amount
61	36	25	5	6

For this 2D real-life robot dataset, 61 successful walking patterns are obtained and 36 of them are chosen for the dataset, and they are divided into training, validation, and test sets. Among those 36 walks, 25 of them (69.44%) are chosen as training, 5 of them (13.89%) are selected as validation, and 6 of them (16.67%) are chosen as test samples. They are visualized using their correspondent speed coefficient and actual walking speeds, in Figure 3.31. Data normalization is done similar to previous sections, and each pattern contains 120 time steps which contain various information obtained from sensors, servos and planarizers about the robot locomotion.

Details about the successful walks obtained in the experiments are given in Table 3.18 and Figure 3.31. The division of training, validation and test samples

Table 3.18: Detailed information about the data acquired from experiments.

Walking duration interval (seconds)	Walking distance interval (meters)	Walking speed interval (m/s)
[6.012, 6.079]	[0.029, 0.230]	[0.005, 0.038]

are given in Figure 3.31. A wide speed coefficient range is selected for experiments, to make the system identification attempts more realistic in the physical environment. Reported walking durations are around 6 seconds, walking speed and distances are obtained programmatically. As expected, a non-linear relation exists between the given input and resulting walking speed for the robot locomotion as observed in the previous sections.

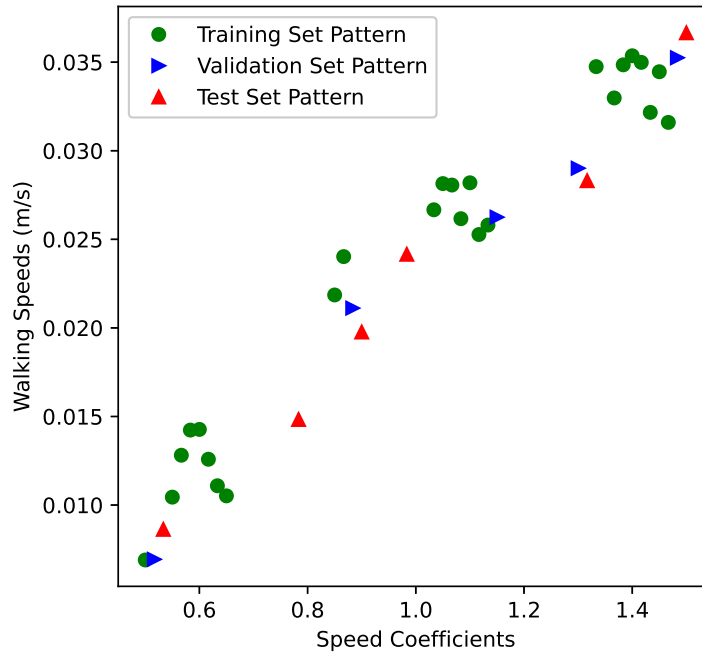


Figure 3.31: The visualization of chosen dataset samples; x axis shows the speed coefficient which is used to show the speed ratio of applied gait, and y axis shows the speed reached by the 2D biped robot.

3.4.3 System Identification With Neural Networks

In this part, we used neural networks in the system identification block for the case of 2D locomotion. Universal Approximation Theorem implies the capability of neural networks to approximate functions with arbitrary accuracy. Therefore, we expect that our 2D physical robot system can be approximated as well. System identification scheme employed for this purpose is given in Figure 3.32.

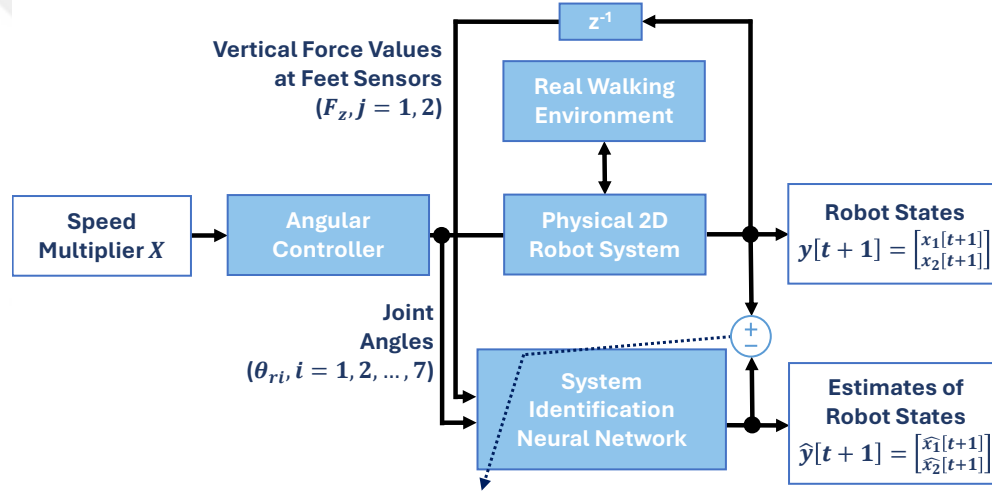


Figure 3.32: System identification configuration is presented, inspired from [48].

Resembling the previous parts, there are three main blocks in this scheme. First, an angular controller takes the input from Speed Multiplier (for walking trajectories with different speed coefficients, from 0.5x to 1.5x) and as an open loop controller, outputs the joint angles. Physical 2D Robot System takes these angles to move the robot joints, by interacting with the walking environment and outputs the complete set of robot states; only the robot location components in x-y axes (interchangeably x_1 and x_2) are used by this scheme are labeled as “robot states”. Also, Physical 2D Robot System use PID controllers and other blocks inside the servos -as shown in Figure 3.26- to make the optimal movement as ordered by the angular controller. The third block, System Identification Neural Network takes joint angles commanded by the controller and z component of

sensed force at each feet to estimate the x and y components of robot location. The error obtained in this estimation is used for backpropagation of this neural network, with an aim to make the network and approximator for the input and output relationship of the 2D physical robot system, similar to system identification case for 2D robot. Prediction error is calculated as the difference between the actual position of 2D physical robot system and estimate of system identification network, and the loss is calculated with x and y position differences using the Mean Squared Error (MSE) definition and summed up cumulatively, adding x and y positions of the hip, and halving the resulting value.

Outputs of the system identification block, which are x and y positions states ($x_1[t]$ and $x_2[t]$, respectively), are listed inside $\mathbf{y}[t]$ vector and the inputs are seven joint angle values ($\theta_{ri}[t], i = 1, 2, \dots, 7$) and vertical (z-axis) force values at both feet ($F_{zj}[t], j = 1, 2$). The equation used to calculate the MSE loss metric is explained below:

$$\mathbf{y}[t] = \begin{bmatrix} x_1[t] \\ x_2[t] \end{bmatrix} \quad (3.21)$$

$$\hat{\mathbf{y}}[t] = \begin{bmatrix} \hat{x}_1[t] \\ \hat{x}_2[t] \end{bmatrix} \quad (3.22)$$

$$\mathbf{e}[t] = \mathbf{y}[t] - \hat{\mathbf{y}}[t] = \begin{bmatrix} x_1[t] - \hat{x}_1[t] \\ x_2[t] - \hat{x}_2[t] \end{bmatrix} \quad (3.23)$$

$$l[t] = \frac{\mathbf{e}[t]^T * \mathbf{e}[t]}{2} \quad (3.24)$$

$$L = \sum_{t=1}^N \frac{l[t]}{N} \quad (3.25)$$

As explained previously, by using walking trajectories generated for the specific speed coefficients, various speed values are used to create walking patterns to

generate a dataset. The dataset is then used to make the system identification with neural networks. The best network amongst all networks tried is the 70 DELU-231 LSTM-231 LSTM network by the test loss scaled mean values. In this example, DELU managed to outperform ReLU, LeakyReLU and ELU activation functions by a small amount of decrease on the error metric for test losses. The trainings are done with a minibatch size of 5, AdamW optimizer and no L2 regularization for 5000 epochs training. Resulting training, validation and test loss rate graphs are also presented in Figure 3.33.

Table 3.19: System identification network results are given in comparison.

Neural Network Hidden Layers	Best Epoch	Training Loss Scaled Mean	Validation Loss Scaled Mean	Test Loss Scaled Mean
70 ReLU, 231 LSTM, 231 LSTM	1200	$1.21 \cdot 10^{-4}$	$3.49 \cdot 10^{-4}$	$3.42 \cdot 10^{-3}$
70 LeakyReLU, 231 LSTM, 231 LSTM	2100	$1.30 \cdot 10^{-4}$	$4.18 \cdot 10^{-4}$	$5.00 \cdot 10^{-3}$
70 ELU, 231 LSTM, 231 LSTM	800	$1.53 \cdot 10^{-4}$	$4.20 \cdot 10^{-4}$	$4.13 \cdot 10^{-3}$
70 DELU, 231 LSTM, 231 LSTM	1100	$2.00 \cdot 10^{-4}$	$6.03 \cdot 10^{-4}$	$3.07 \cdot 10^{-3}$

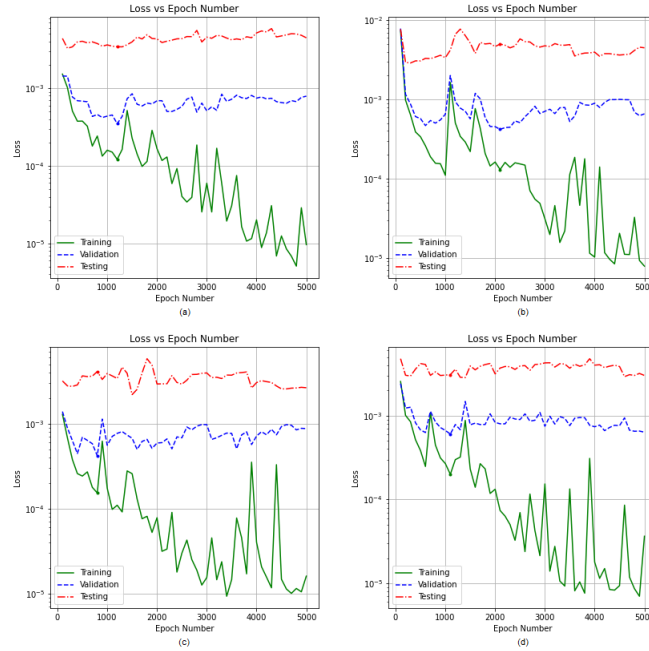


Figure 3.33: Training, validation and test results are shown for networks from the first entry of the Table 3.19 to last entry, with letters from (a) to (d), respectively. Dots indicate the loss results of the networks at the epoch with lowest validation losses during their trainings.

3.5 Chapter Summary

This chapter started with a literature review on legged robots and their locomotion, mentioning the advantages and disadvantages. In the next section, a 2D, two-legged robot model with six joints from [66] is presented, and it is used to generate a walking dataset for different ground inclination values, with a specialized Central Pattern Generator (CPG). After that, neural controller architectures are trained on this dataset for locomoting the robot with a goal velocity, which included layers with DELU and other activation functions as well. Then, system identification neural networks are employed for approximating this robot model. These networks are used in various configurations, these configurations include using different dataset settings and using different architectures, with DELU and other activation functions used in the first layer of neural networks. Also, DELU activation function is tuned similar to the previous chapter, for trying to reach better performance with the system identification neural networks. Afterwards, adaptive control is introduced with the NNBMRAAC algorithm we proposed and using four different scenarios, the effectiveness of this algorithm to lower the speed steady state errors is tested with obtained results. Similar to before, first layers of the neural networks included different activation function including our proposal, DELU.

In the following section, a 3D, two-legged robot model called NICO [67] is presented, and it is used to generate a walking dataset on a flat surface with varying walking directions, by a specialized Central Pattern Generator (CPG). After that, neural controller architectures are trained on this dataset for locomoting the robot with desired goal velocity and walking direction. Similar to the previous part, neural network blocks used in this part included layers with DELU and other activation functions. Then, system identification neural networks are employed for approximating NICO robot model, which included different neural network architectures with DELU and other activation functions at the first layer. Again, DELU activation function is tuned similar to the previous section, for trying to reach better performance with the system identification neural networks. Afterwards, adaptive control is done for the case of Scenario #3 with

the renewed NNBMRAc algorithm, for the case of 3D robot locomotion. In this case, the effectiveness of this algorithm to lower both speed steady state error and walking direction deviation errors is tested by obtaining results. The result tables are given separately for the case of using DELU or ReLU activation function at the first layer of system identification networks, to highlight the differences.

In the last section of this chapter, a 2D physical robot system is introduced. The design, build and specialties of this robot and its components, such as servos and sensors, are specified. After describing the walking environment as well, dataset generation is described in the next section where the data is visualized similar to the previous parts. This dataset is then used for training different system identification neural networks which included DELU and other activation functions at their first layer, and resulting loss values during their trainings are visualized similar to the previous sections, which concludes this chapter.

Chapter 4

Conclusion and Future Works

In conclusion, this thesis is mainly consisted of two different parts. In the first part, Chapter 1 and 2, a brief review of neural networks literature is completed by going over important parts of neural network architectures. In the second chapter, DELU activation function is introduced and tested in detail, with different network architectures, different tasks and different datasets. In the third chapter, control, system identification and adaptive control systems are introduced which employ neural network blocks in their diagrams. Both for 2D and 3D, simulations are conducted for our experiments and results are presented in comparison, separately. Finally, a real-life 2D biped robot system is constructed and it is used to generate a dataset, which is then used to make system identification featuring different neural networks, including DELU.

To relist the contributions of this thesis to the literature, we start from the second chapter. DELU activation function is introduced as a viable alternative to ReLU, ELU and other rectifier-based activation functions. Using both image classification and image segmentation networks for comparison -alongside different initialization methods as mentioned previously- the performance of DELU is investigated. In addition to these comparisons, obtained results are analyzed with hypothesis testing using Z-scores to find percentage of confidences for determining how DELU is better than ReLU or ELU, with high values of percentage

of confidences. In total, 4 datasets are used for comparing DELU with ReLU, ELU, SELU, FELU, MPELU and Softplus and in addition to these works, DELU parameters are varied in a specific section to figure out the optimal set of parameters for Fashion MNIST and CIFAR-10 dataset. With this thorough research, a novel activation function is added to the literature. Applying this methodology from detection and estimation theory to the field of machine learning can be counted as another contribution to the literature.

To discuss the relative success rate improvements, we can start by presenting the baselines. The default choices for the ResNet20-v1 was ReLU activation function for the internal layers, which resulted in 93.694% mean success rate for Fashion MNIST dataset, 90.099% mean success rate for CIFAR-10 dataset and 73.406% mean success rate for CIFAR-100 dataset. These mean success rates are all improved by replacing ReLU with DELU activation function, and 93.903% for Fashion MNIST, 90.468% for CIFAR-10 and 73.509% for CIFAR-100 dataset within their respective training and inference configurations. By changing the parameters of DELU, even higher mean success rates were obtained for Fashion MNIST dataset as well (93.922%). Similarly, an increase of mean success is also obtained with DELU for image segmentation task on the Oxford-IIIT Pets dataset (0.048%). Afterwards, for the case of adding Truncated Towers initialization into picture, mean success rate was taken from 93.694% to 93.810% for the comparison between independent Gaussian and Uniform initializations on the Fashion MNIST dataset.

For continuing to relist the contributions of this thesis to the literature, we can move to the third chapter. Here, control, system identification and adaptive control systems -which use similar neural network blocks in their diagrams- are introduced to the literature with different configurations. By using various neural network architectures to control robot locomotion, to make system identification and apply adaptive control for 2D and 3D simulations which have different datasets, another contribution is also done. Furthermore, in these neural networks, DELU activation function and its tunings are also used for attempting to maximize the success rates, and generating stable walks which obey to the error constraints and avoid falling. Finally, a physical realization of 2D biped

robot is built using a planarizer and a dataset is generated. With this dataset, system identification is made using the neural networks which included DELU activation function for some networks as well.

To go into the details, DELU is used for both 2D and 3D robot simulation results obtained in the experiments. For the case of 2D, neural torque controller with 43 LSTM-74 ReLU hidden layers (with no regularization) was the best amongst other tried network architectures with 84.69% walking success and 3.16% speed steady state error, but neural torque controller with 43 LSTM-74 DELU hidden layers also close exceeded the threshold and displayed a significant performance, with 78.57% walking success and 4.54% speed steady state error. Again, for the case of 2D robot simulations, system identification network with 70 ReLU-231 LSTM-231 LSTM hidden layers performed best overall. With adaptive learning system, lowerings of error rates on different success and error metrics are obtained. For the case of 3D robot simulations, a slightly modified version of neural controller with 43 LSTM-74 ReLU hidden layers is used for the case of 3D and system identification networks with 140 DELU-231 LSTM-231 LSTM hidden layers performed better than ReLU and LeakyReLU cases, implying an increase of performance for this example: Test loss scaled mean is fallen to $2.25 \cdot 10^{-3}$ value. This increase is also reflected in adaptive control results, where system identification network with DELU lowered error rates more than ReLU, mostly. In addition to those, the parameters of DELU activation function is tuned for both 2D and 3D system identification networks. An improvement is obtained for the case of 2D, and DELU tuning with beta value of 1.25 became a close third in the table, very near to the ReLU results.

In the near future, we will be expanding our research to various application areas and using different network types, such as recurrent neural network (RNN). This will help us to evaluate the efficiency of DELU activation function with various initialization methods in different neural network architectures. In addition, a bounded version of DELU might be applicable to RNN units, such as LSTM cells, for different tasks like time-series prediction and neural language processing, as a possible future work. Key research directions are applying DELU into

different network configurations and obtaining results, trying it out with state-of-the-art architectures and determining DELU's effect on them, continuing to optimize DELU with different setups and possibly, adaptively changing α and β values by varying them during the training process for possible improvement in different applications. Also, by using 2D physical robot system -which is built in scope of the TÜBİTAK ARDEB 1001 project, numbered 120E104- to create more real-time data and larger datasets, making different system identifications with neural networks is possible. Furthermore, for the real-life applications, implementation of locomotion control using neural networks in such configurations and employing resulted control and identification blocks to make adaptive robot motion control and real-time testing of those structures can be presented as possible future works.

Bibliography

- [1] Bu. Çatalbaş and Ö. Morgül, “Deep learning with Extended Exponential Linear Unit (DELU),” *Neural Computing and Applications*, vol. 35, no. 30, pp. 22705–22724, Springer, 2023.
- [2] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.
- [3] A. Kooser, “Watch a boston dynamics atlas robot grab and throw like it’s nothing,” cnet.com. <https://www.cnet.com/science/see-a-boston-dynamics-atlas-robot-grab-and-throw-like-its-nothing/> (accessed June 19, 2024).
- [4] D. W. Rosołowski and K. Jedrzejewski, “Experimental evaluation of a digital predistortion based on simple feedforward neural network,” in *2020 23rd International Microwave and Radar Conference (MIKON)*, pp. 293–296, IEEE, 2020.
- [5] B. Ding, H. Qian, and J. Zhou, “Activation functions and their characteristics in deep neural networks,” in *2018 Chinese Control Decision Conference (CCDC)*, pp. 1836–1841, June 2018.
- [6] Bu. Çatalbaş, Ba. Çatalbaş, and Ö. Morgül, “Human activity recognition with different artificial neural network based classifiers,” in *2017 25th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, May 2017.
- [7] Bu. Çatalbaş, “Improved artificial neural network training with advanced methods,” Master’s thesis, Bilkent University, 2018.

- [8] Y. Wang, Y. Li, Y. Song, and X. Rong, “The influence of the activation function in a convolution neural network model of facial expression recognition,” *Applied Sciences*, vol. 10, no. 5, p. 1897, 2020.
- [9] Ba. Çatalbaş and Ö. Morgül, “Tekrarlayan sinir ağları ile iki ayaklı robot hareket kontrolü,” in *2019 21. Otomatik Kontrol Ulusal Toplantısı TOK 2019*, pp. 270–277, 2019.
- [10] Ba. Çatalbaş, Bu. Çatalbaş, and Ö. Morgül, “Two-legged robot system identification with artificial neural networks,” in *2020 28th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, IEEE, 2020.
- [11] S. Haykin, *Neural networks: A comprehensive foundation*. Prentice Hall, New Jersey, NY, 1999.
- [12] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, “Activation functions in deep learning: A comprehensive survey and benchmark,” *Neurocomputing*, 2022.
- [13] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.
- [14] A. Williams, “The art of building neural networks,” *TheNewStack*. Retrieved from <https://thenewstack.io/art-building-neural-networks/>, 2017.
- [15] Q. Zheng, M. Yang, J. Yang, Q. Zhang, and X. Zhang, “Improvement of generalization ability of deep cnn via implicit regularization in two-stage training process,” *IEEE Access*, vol. 6, pp. 15844–15869, 2018.
- [16] H. Li, N. Zeng, P. Wu, and K. Clawson, “Cov-net: A computer-aided diagnosis method for recognizing covid-19 from chest x-ray images via machine vision,” *Expert Systems with Applications*, vol. 207, p. 118029, 2022.
- [17] A. Apicella, F. Donnarumma, F. Isgrò, and R. Prevete, “A survey on modern trainable activation functions,” *Neural Networks*, vol. 138, pp. 14–32, 2021.

- [18] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (elus),” *arXiv preprint arXiv:1511.07289*, 2015.
- [19] F. Sakketou and N. Ampazis, “On the invariance of the selu activation function on algorithm and hyperparameter selection in neural network recommenders,” in *Artificial Intelligence Applications and Innovations: 15th IFIP WG 12.5 International Conference, AIAI 2019, Hersonissos, Crete, Greece, May 24–26, 2019, Proceedings 15*, pp. 673–685, Springer, 2019.
- [20] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for activation functions,” *arXiv preprint arXiv:1710.05941*, 2017.
- [21] K. Adem, “P+ felu: Flexible and trainable fast exponential linear unit for deep learning architectures,” *Neural Computing and Applications*, vol. 34, no. 24, pp. 21729–21740, 2022.
- [22] Y. Zhou, D. Li, S. Huo, and S.-Y. Kung, “Shape autotuning activation function,” *Expert Systems with Applications*, vol. 171, p. 114534, 2021.
- [23] A. A. Alkhoully, A. Mohammed, and H. A. Hefny, “Improving the performance of deep neural networks using two proposed activation functions,” *IEEE Access*, vol. 9, pp. 82249–82271, 2021.
- [24] Y. Li, C. Fan, Y. Li, Q. Wu, and Y. Ming, “Improving deep neural network with multiple parametric exponential linear units,” *Neurocomputing*, vol. 301, pp. 11–24, 2018.
- [25] Z. Qiumei, T. Dan, and W. Fenghua, “Improved convolutional neural network based on fast exponentially linear unit activation function,” *IEEE Access*, vol. 7, pp. 151359–151367, 2019.
- [26] Github, “Code for improving deep neural network with multiple parametric exponential linear units,” *Github*. Retrieved from <https://github.com/Coldmoon/Code-for-MPELU> (accessed June 9, 2024).
- [27] P. Billingsley, *Probability and Measure*. New York, NY: John Wiley & Sons, third ed., 1995.

- [28] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International Conference on Machine Learning*, pp. 448–456, pmlr, 2015.
- [29] E. Alcaide, “E-swish: Adjusting activations to different network depths,” *arXiv preprint arXiv:1801.07145*, 2018.
- [30] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. Jawahar, “Cats and dogs,” in *2012 IEEE conference on computer vision and pattern recognition*, pp. 3498–3505, IEEE, 2012.
- [31] F. Chollet, “Image segmentation with a U-net-like architecture,” *Keras*. Retrieved from keras.io/examples/vision/oxford_pets_image_segmentation, 2020.
- [32] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” Retrieved from www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf, 2009.
- [33] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” *arXiv preprint arXiv:1708.07747*, 2017.
- [34] S. Shan, E. Willson, B. Wang, B. Li, H. Zheng, and B. Y. Zhao, “Gotta catch’em all: Using concealed trapdoors to detect adversarial attacks on neural networks,” *arXiv preprint arXiv:1904.08554*, vol. 2, 2019.
- [35] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” in *Computer Vision—ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pp. 630–645, Springer, 2016.
- [36] Keras, “Trains a resnet on the cifar10 dataset,” Retrieved from https://keras.io/zh/examples/cifar10_resnet/, n.d.
- [37] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

- [38] A. Papoulis, *Probability, random variables, and stochastic processes*. New York, NY: McGraw-Hill, second ed., 1985.
- [39] V. A. Epanechnikov, “Non-parametric estimation of a multivariate probability density,” *Theory of Probability & Its Applications*, vol. 14, no. 1, pp. 153–158, 1969.
- [40] MathWorks, “Kernel distribution,” Retrieved from <https://www.mathworks.com/help/stats/kernel-distribution.html>, n.d.
- [41] H. V. Poor, *An introduction to signal detection and estimation*. Springer Science & Business Media, 2013.
- [42] G. R. DeVore, “Computing the z score and centiles for cross-sectional analysis: a practical approach,” *Journal of Ultrasound in Medicine*, vol. 36, no. 3, pp. 459–473, 2017.
- [43] S. Urolagin, N. Sharma, and T. K. Datta, “A combined architecture of multivariate lstm with mahalanobis and z-score transformations for oil price forecasting,” *Energy*, vol. 231, p. 120963, 2021.
- [44] A. Adler, P. Gaggero, and Y. Maimaitijiang, “Distinguishability in eit using a hypothesis-testing model,” in *Journal of Physics: Conference Series*, vol. 224, p. 012056, IOP Publishing, 2010.
- [45] W. W. LaMorte, Boston University School of Public Health. “Hypothesis testing: Upper-, lower, and two tailed tests,” https://sphweb.bumc.bu.edu/otlt/mph-modules/bs/bs704_hypothesistest-means-proportions/bs704_hypothesistest-means-proportions3.html (accessed June 13, 2024).
- [46] D. P. Kingma and J. Ba, “A method for stochastic optimization,” in *3rd International Conference for Learning Representations, San Diego, 2015*, vol. 1412, 2015.
- [47] I. Loshchilov and F. Hutter, “Fixing weight decay regularization in adam,” *Submitted to ICLR 2018*. Retrieved from <https://openreview.net/forum?id=rk6qdGgCZ>, 2018.

- [48] Ba. Çatalbaş, Control and system identification of legged locomotion with recurrent neural networks. PhD thesis, Bilkent University, 2022.
- [49] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, Springer, 1989.
- [50] A. M. Schäfer and H. G. Zimmermann, “Recurrent neural networks are universal approximators,” *Artificial Neural Networks–ICANN 2006: 16th International Conference, Athens, Greece, 2006, Proceedings, Part I, 16*, pp. 632–640, Springer, 2006.
- [51] 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.
- [52] 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*, vol. 3, pp. 2684–2689, IEEE, 2002.
- [53] J. R. Gosyne, C. M. Hubicki, X. Xiong, A. D. Ames, and D. I. Goldman, “Bipedal locomotion up sandy slopes: Systematic experiments using zero moment point methods,” in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pp. 994–1001, IEEE, 2018.
- [54] S. Shigemi, A. Goswami, and P. Vadakkepat, “Asimo and humanoid robot research at honda,” *Humanoid robotics: A reference*, pp. 55–90, 2018.
- [55] C.-F. Huang, Y.-C. Tung, H.-T. Lu, and T.-J. Yeh, “Balancing control of a bicycle-riding humanoid robot with center of gravity estimation,” *Advanced Robotics*, vol. 32, no. 17, pp. 918–929, 2018.
- [56] E. Ackerman, “Honda robotics unveils next-generation asimo robot,” [ieee.org. https://spectrum.ieee.org/honda-robotics-unveils-next-generation-asimo-robot](https://spectrum.ieee.org/honda-robotics-unveils-next-generation-asimo-robot) (accessed 19 June, 2024).

- [57] A. J. Ijspeert, “Central pattern generators for locomotion control in animals and robots: A review,” *Neural Networks*, vol. 21, no. 4, pp. 642–653, 2008.
- [58] K. Grace, H. Stewart, J. F. Sandkühler, S. Thomas, B. Weinstein-Raun, and J. Brauner, “Thousands of ai authors on the future of ai,” *arXiv preprint arXiv:2401.02843*, 2024.
- [59] Ba. Çatalbaş and Ö. Morgül, “Two-legged robot motion control with recurrent neural networks,” *Journal of Intelligent & Robotic Systems*, vol. 104, no. 4, p. 59, 2022.
- [60] Bu. Çatalbaş, Ba. Çatalbaş, and Ö. Morgül, “İki bacaklı robot hareket kontrolünde yapay sinir ağlarının kullanımı,” in *2023 24. Otomatik Kontrol Ulusal Toplantısı TOK 2023*, TOK, September 2023.
- [61] F. Lewis, S. Jagannathan, and A. Yesildirak, *Neural network control of robot manipulators and non-linear systems*. CRC press, 1998.
- [62] S. Haykin, *Neural networks and learning machines, 3/E*. Pearson Education India, 2009.
- [63] D. Janglova, “Neural networks in mobile robot motion,” *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, pp. 2, SAGE Publications, 2004.
- [64] S. X. Yang and M. Meng, “An efficient neural network approach to dynamic robot motion planning,” *Neural Networks*, vol. 13, no. 2, pp. 143–148, Elsevier, 2000.
- [65] S. M. Prabhu and D. P. Garg, “Artificial neural network based robot control: An overview,” *Journal of Intelligent and Robotic Systems*, vol. 15, pp. 333–365, Springer, 1996.
- [66] 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.

- [67] M. Kerzel, E. Strahl, S. Magg, N. Navarro-Guerrero, S. Heinrich, and S. Wermter, “Nico—neuro-inspired companion: A developmental humanoid robot platform for multimodal interaction,” in *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pp. 113–120, IEEE, 2017.
- [68] T. G. Brown, “On the nature of the fundamental activity of the nervous centers,” *Journal of Physiologie*, vol. 48, pp. 18–46, 1914.
- [69] S. Auddy, S. Magg, and S. Wermter, “Hierarchical control for bipedal locomotion using central pattern generators and neural networks,” in *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pp. 13–18, IEEE, 2019.
- [70] A. Crespi and A. J. Ijspeert, “Amphibot ii: An amphibious snake robot that crawls and swims using a central pattern generator,” in *Proceedings of the 9th international conference on climbing and walking robots (CLAWAR 2006)*, pp. 19–27, 2006.
- [71] C. Liu, Y. Chen, J. Zhang, and Q. Chen, “Cpg driven locomotion control of quadruped robot,” in *2009 IEEE International Conference on Systems, Man and Cybernetics*, pp. 2368–2373, IEEE, 2009.
- [72] G. Bellegarda and A. Ijspeert, “Cpg-rl: Learning central pattern generators for quadruped locomotion,” *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 12547–12554, 2022.
- [73] Z. Song, X. Huang, and J. Xu, “Spatiotemporal pattern of periodic rhythms in delayed van der pol oscillators for the cpg-based locomotion of snake-like robot,” *Nonlinear Dynamics*, vol. 110, no. 4, pp. 3377–3393, 2022.
- [74] A. H. Cohen, P. J. Holmes, and R. H. Rand, “The nature of the coupling between segmental oscillators of the lamprey spinal generator for locomotion: A mathematical model,” *Journal of mathematical biology*, vol. 13, pp. 345–369, 1982.

- [75] J. Buchli and A. J. Ijspeert, “Distributed central pattern generator model for robotics application based on phase sensitivity analysis,” in *International Workshop on Biologically Inspired Approaches to Advanced Information Technology*, pp. 333–349, Springer, 2004.
- [76] K. Matsuoka, “Sustained oscillations generated by mutually inhibiting neurons with adaptation,” *Biological cybernetics*, vol. 52, no. 6, pp. 367–376, 1985.
- [77] Robotis Company Limited, “XH540-W150-T/R.”, robotis.com. <https://emanual.robotis.com/docs/en/dxl/x/xh540-w150/> (accessed June 13, 2024).

Appendix

Directional walking calculations for NICO Robot according to [69]:

Application of CPG modulation by the linear controller to set values of $\phi_{l,r}$, where G stands for gain of the linear controller and $bound(x)$ restricts x within range $[0, 1]$:

$$(\phi_l, \phi_r) = \begin{cases} (bound(G \times |d_y|), 0) & d_y > 0 \\ (0, bound(G \times |d_y|)) & d_y < 0 \end{cases}$$

Calculating $\psi_{l,r}$ values with $\phi_{l,r}$ and ξ value which determines the control of high-level controller over low-level controller:

$$\psi_l = 1.0 - (1.0 - \xi)\phi_l$$

$$\psi_r = 1.0 - (1.0 - \xi)\phi_r$$

Calculating directional inputs with $\psi_{l,r}$ which will interacts with CPG network to change the direction of the robot:

$$\theta_2 = o_2\psi_l g_1 + b_1$$

$$\theta_3 = o_3\psi_r g_1 + b_1$$

In this thesis, instead of the derivations made in first three equations, only last two equations are used and $\psi_{l,r}$ values are directly given to the NICO robot, either manually for generating the dataset or by the neural networks used.