



**IMPLEMENTATION OF A LIQUID NEURAL  
NETWORK CONTROL SYSTEM FOR MULTI-  
JOINT CYBER PHYSICAL ARM**

**2023  
MASTER THESIS  
COMPUTER ENGINEERING**

**Michael BIDOLLAHKHANI**

**Thesis Advisor  
Assist. Prof. Dr. Ferhat ATASOY  
Assist. Prof. Dr. Abdellatef HAMDAN**

**IMPLEMENTATION OF A LIQUID NEURAL NETWORK  
CONTROL SYSTEM FOR MULTI-JOINT CYBER PHYSICAL ARM**

**Michael BIDOLLAHKHANI**

**Thesis Advisors**

**Assist. Prof. Dr. Ferhat ATASOY  
Assist. Prof. Dr. Abdellatef HAMDAN**

**T.C.**

**Karabuk University  
Institute of Graduate Programs  
Department of Computer Engineering  
Prepared as  
Master Thesis**

**KARABÜK  
June 2023**

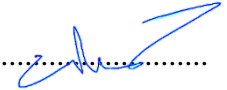
I certify that in my opinion the thesis submitted by Michael Bidollahkhani titled “IMPLEMENTATION OF A LIQUID NEURAL NETWORK (LTC) CONTROL SYSTEM FOR MULTI-JOINT CYBER PHYSICAL ARM” is fully adequate in scope and in quality as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Ferhat ATASOY .....

Thesis Advisor, Department of Computer Engineering

Assist. Prof. Dr. Abdellatef HAMDAN .....

Thesis Advisor, Department of Computer Engineering



This thesis is accepted by the examining committee with a unanimous vote in the Department of Computer Engineering as a Master of Science thesis. June 2023

Examining Committee Members (Institutions)

Signature

Chairman : Assoc. Prof. Dr. Caner ÖZCAN (KBÜ) .....

Member : Assoc. Prof. Dr. Fatih NAR (AYBÜ) .....

Member : Assist. Prof. Dr. FERHAT ATASOY (KBÜ) .....

The degree of, Master of Science by the thesis submitted is approved by the Administrative Board of the Institute of Graduate Programs, Karabuk University.

Prof. Dr. Müslüm KUZU .....

Director of the Institute of Graduate Programs

*“I declare that all the information within this thesis has been gathered and presented in accordance with academic regulations and ethical principles and I have according to the requirements of these regulations and principles cited all those which do not originate in this work as well.”*

Michael BIDOLLAHKHANI

## **ABSTRACT**

**Master Thesis**

### **IMPLEMENTATION OF A LIQUID NEURAL NETWORK CONTROL SYSTEM FOR MULTI-JOINT CYBER PHYSICAL ARM**

**Michael BIDOLLAHKHANI**

**Karabük University  
Institute of Graduate Programs  
The Department of Computer Engineering**

**Thesis Advisors:**

**Assist. Prof. Dr. Ferhat ATASOY**

**Assist. Prof. Dr. Abdellatif HAMDAN**

**June 2023, 66 pages**

Technological solutions are being produced to meet people's needs and fulfill their desires in a comfortable way. As technology becomes cheaper, more widespread, smaller in size, and able to operate independently from the power grid, the communication of devices with each other (Internet of Things) and the ability of devices to make their own decisions increase the effectiveness of solutions. In particular, the reduction in device size can be achieved by requiring less system resources and battery capacity. Therefore, existing methods need to be customized to work effectively in embedded systems.

In this thesis a novel approach called LTC-SE, which enhances the Liquid Time-Constant Neural Network (LTC) technique for embedded environments with limited processing capabilities and strict performance requirements is presented. LTC-SE

combines various neural network paradigms, including Leaky-Integrate-and-Fire (LIF) spiking neural networks, Continuous-Time Recurrent Neural Networks (CTRNNs), Neural Ordinary Differential Equations (NODEs), and customized Gated Recurrent Units (GRUs), resulting in improved adaptability, interoperability, and structural organization. In the thesis, a unified class library, is developed, called LTCCell that offers extensive configurability CTRNN, NODE, and CTGRU elements. The proposed method is evaluated by developing a control system for a multi-joint cyber-physical arm, demonstrating its effectiveness in achieving designated objectives and manipulating objects securely. The system's performance is presented through a decision support framework and multi-variable benchmarking, emphasizing the benefits of our refinements in terms of user interaction, functional coherence, and code clarity.

Furthermore, the LTC-SE technique expands the scope of liquid neural networks, finding applications in diverse machine learning domains such as robotics, causality assessment, and time-series forecasting. This thesis presents innovative contributions to the field based on the pioneering work of LTC neural network.

**Key Words** : Cyber physical Control System, Recurrent Neural Networks (RNN), Liquid Time-Constant (LTC), Explainable Artificial Intelligence (xAI), Decision Support Systems (DSS).

**Science Code** : 92432

## ÖZET

**Yüksek Lisans Tezi**

### **SIVI SİNİR AĞI KONTROL SİSTEMİNİN ÇOK EKLEMLİ SİBER- FİZİKSEL KOL İÇİN UYGULANMASI**

**Michael BIDOLLAHKHANI**

**Karabük Üniversitesi**

**Lisansüstü Eğitim Enstitüsü**

**Bilgisayar Mühendisliği Anabilim Dalı**

**Tez Danışmanları:**

**Dr. Öğr. Üyesi Ferhat ATASOY**

**Dr. Öğr. Üyesi Abdellatif Hamdan**

**Haziran 2023, 66 sayfa**

İnsanların ihtiyaçlarını giderme ve isteklerini yerine getirme talebinin konforlu bir şekilde gerçekleştirilmesi için teknolojik çözümler üretilmektedir. Teknolojinin ucuzlaması, yaygınlaşması, boyutlarının küçülmesi, elektrik şebekesinden bağımsız şekilde çalışabilir hale gelmesiyle birlikte cihazların birbirleriyle haberleşmesi (nesnelerin interneti) ve cihazların kendi kararlarını verebilecek hale getirilmesi çözümlerin etkinliğini arttırmaktadır. Özellikle cihaz boyutlarındaki küçülme daha az sistem kaynağına ve batarya kapasitesine ihtiyaç duyması ile sağlanabilmektedir. Bundan dolayı mevcut yöntemlerin gömülü sistemlerde etkili bir şekilde çalışması için bazen özelleştirilmeleri gerekmektedir.

Bu tezde sınırlı işlem kapasitesine ve sıkı performans kriterlerine sahip gömülü ortamlar için özelleştirilmiş bir sıvı sinir ağı (İng.: Liquid Time-Constant Neural Network Special Edition – LTC-SE) tekniği olan LTC-SE'nin geliştirilmiş bir versiyonu sunulmaktadır. LTC-SE Sızdır-Bütünleştir ve Ateşle darbeli sinir ağını, Sürekli-Zamanlı Özyinelemeli Sinir Ağlarını (SZÖSA'lar), Sinirsel Adi Diferansiyel Denklemleri (SADD'ler) ve Özelleştirilmiş Geçitli Tekrarlayan Birimleri (ÖGTB'ler) içeren çeşitli sinir ağları paradigmalarını kombine ederek adaptasyon yeteneğini, iş birliği kabiliyetini ve yapısal düzenlemesini güçlendirmektedir. Tezde SZÖSA, SADD ve ÖGTB elemanları için LTCCell adı verilen geniş konfigürasyon imkanı sunan birleşik bir sınıf kütüphanesi geliştirilmiştir. Önerilen yöntem, çok eklemli bir siber-fiziksel kol için bir kontrol sistemi geliştirerek, belirlenen hedeflere ulaşmada ve nesneleri güvenli bir şekilde manipüle etmede etkinliğini göstererek değerlendirilmiştir. Sistemin performansı, kullanıcı etkileşimi, işlevsel tutarlılık ve kod netliği açısından iyileştirmelerimizin faydalarını vurgulayan bir karar destek çerçevesi ve çok değişkenli kıyaslama yoluyla sunulmuştur.

Ayrıca, LTC-SE tekniği, robotik, nedensellik değerlendirmesi ve zaman serisi tahmini gibi çeşitli makine öğrenimi uygulamalarında sıvı sinir ağı kavramının kapsamını genişletmektedir. Bu tez, LTC sinir ağının öncü çalışmalarına dayanarak alana yenilikçi katkılar sunmaktadır.

**Anahtar Kelimeler :** Siber Fiziksel Kontrol Sistemi, Tekrarlı Sinir Ağı (TSA), Sıvı Sinir Ağı (SSA), Açıklanabilir Yapay Zeka, Karar Destek Sistemi (KDS).

**Bilim Kodu :** 92432



## **ACKNOWLEDGMENT**

First of all, I would like to thank God, for letting me through all the difficulties. Who supported me all the time and gave me priceless guides and helping hands to finish what I started.

I would like to acknowledge and express my deepest gratitude to my family, who supported me without hesitation anytime.

Also, I'm genuinely thankful of my Karabuk university supervisor Dr. Ferhat Atasoy and Dr. Abdellatef Hamdan (Lebanese American University). Their guidance and advice carried me through all the stages of implementation of the project. I want to appreciate the guidelines provided by Dr. Ramin Hasani (Massachusetts Institute of Technology), which gave me the motivation and seeds for starting my studies on liquid neural networks and where to start my research journey about cognitive and decision-making functionalities of brain frontal lobe.

## CONTENTS

	<u>Page</u>
ABSTRACT.....	iv
ÖZET.....	vi
ACKNOWLEDGMENT.....	viii
CONTENTS.....	ix
LIST OF FIGURES .....	xii
LIST OF TABLES .....	xiii
SYMBOLS AND ABBREVIATIONS INDEX .....	xiv
CHAPTER 1 .....	1
INTRODUCTION .....	1
1.1.    MOTIVATION AND PROBLEM STATEMENT .....	2
1.2.    OBJECTIVE .....	4
1.3.    PROJECT OBJECTIVES .....	4
1.4.    THE IMPORTANCE OF THE OBJECTIVE.....	4
CHAPTER 2 .....	6
LITERATURE REVIEW.....	6
2.1.    FUNDAMENTAL ALGORITHMS .....	6
2.2.    LIMITATIONS OF TRADITIONAL ALGORITHMS .....	6
2.3.    THE NEURAL NETWORKS .....	7
2.4.    NEURAL AND EVOLUTIONARY COMPUTING.....	8
2.4.1. Biological Neural System.....	8
2.4.1.1.    The temporal lobe of our brain.....	8
2.4.1.2.    The occipital lobe.....	9
2.4.1.3.    The frontal lobe .....	9
2.4.2. Artificial Neural Network (ANN) .....	9
2.4.2.1.    The artificial neuron.....	10
2.4.2.2.    Nodes .....	10

- 2.5. LEARNING ALGORITHM SELECTION ..... 12
  - 2.5.1. Resources and validity ..... 12
    - 2.5.1.1. Explainability ..... 12
    - 2.5.1.2. In-memory vs. out-of-memory ..... 12
    - 2.5.1.3. Number of features and examples ..... 12
    - 2.5.1.4. Categorical vs. numerical features ..... 13
    - 2.5.1.5. Nonlinearity of the data ..... 13
    - 2.5.1.6. Training speed ..... 13
    - 2.5.1.7. Prediction speed ..... 13
  - 2.5.2. Control Systems and Neural Networks: ..... 14
  - 2.5.3. Recurrent Neural Networks (RNNs): ..... 15
  - 2.5.4. RNN and NVIDIA CUDA ..... 16
  - 2.5.4. Emergence of LSTM Networks: ..... 17
  - 2.5.5. The Role of LSTM in Control Systems: ..... 17
  - 2.5.6. Other LSTM-Related Approaches and Recent Research: ..... 17
  - 2.5.7. Time-Constant Neural Networks ..... 18
  - 2.5.8. Liquid Neural Network or Liquid Time-Constant NN (LTC) ..... 19
  - 2.5.8. Leveraging Ode Solvers For Dynamic System Modeling ..... 21
- 2.6. CONCEPT AND FUNCTIONALITIES ..... 22
- 2.7. APPLICATIONS ..... 22
- 2.8. THE IMPLEMENTATION PROCESS ..... 23

CHAPTER 3 ..... 26

THEORETICAL BACKGROUNDS ..... 26

- 3.1. DYNAMIC EQUATIONS AND IO ..... 26
- 3.2. UNIVERSAL APPROXIMATORS OF LTCs ..... 28
  - 3.2.1. Theorem Alpha ..... 28
  - 3.2.2. Proof of Theorem Alpha ..... 29
- 3.3. TRACING A SIMPLE LTC FOR AN INSTANCE ..... 33
  - 3.3.1. Data preparation ..... 33
  - 3.3.2. Network initialization ..... 34
  - 3.3.3. Forward pass ..... 34
  - 3.3.4. Loss calculation: ..... 35

3.3.5. Backward pass: .....	35
3.3.6. Iteration:.....	36
3.4. CONCLUSION AND REMARKS FOR THEORETICAL BACKGROUNDS .....	37
CHAPTER 4 .....	38
METHODOLOGY.....	38
4.1. THE PROPOSED ALGORITHM.....	39
4.2. TIME-CONSTANT NEURAL NETWORKS .....	42
4.3. METHODOLOGY STEPS .....	45
4.4. IMPLEMENTATION .....	46
4.5. PROJECT REPOSITORY ON GITHUB.....	50
CHAPTER 5 .....	51
RESULTS AND DISCUSSION .....	51
5.1. ORIGINAL LTC MODEL.....	52
5.2. OPTIMIZED LTC MODEL (Proposed By Researchers) .....	53
5.3. THE POTENTIAL OF LTC NEURAL NETWORKS .....	54
CHAPTER 6 .....	57
6.1. CONCLUSION .....	57
6.2. FUTURE WORKS .....	59
REFERENCES.....	62
RESUME .....	66

## LIST OF FIGURES

	<u>Page</u>
Figure 1.1. An instance of chained mechanical robot consisting of joints, rigids, actuators and electronics .....	3
Figure 2.1. Comparison between Classical Statistics and Overparameterization Era .	7
Figure 2.2. Brain lobes and their known major functionality [6].....	9
Figure 2.3. Node with inputs (x), weights (w), output (y) .....	11
Figure 2.4. Demonstration of a Multi-Layer Feed-Forward NN .....	15
Figure 2.5. Demonstration of a Multi-Layer RNN Architecture .....	16
Figure 2.6. Demonstration of Liquid Neural Network (LTC) Architecture .....	20
Figure 2.7. An instance for a multi-joints arm system.....	24
Figure 4.1. Rigid Bodies Tree view of some kinematics instances; here Joints are labeled as (v) and Rigids by (e).....	39
Figure 4.2. Reverse-mode differentiation through an ODE solver requires solving an augmented system backwards in time. This adjoint state is updated by the gradient at each observation (Credit: Chen et al. NeurIPS, 2018).....	44
Figure 4.3. Demonstration of Bedframe and Fixed transformation between frames addressing.....	47
Figure 5.1. Memory usage comparison for different time series prediction tasks and algorithms.....	55

## LIST OF TABLES

**Sayfa**

Table 5.1. Performance Metrics for Comparing Original and Optimized LTC Models Outcomes.....	53
---	----

## SYMBOLS AND ABBREVIATIONS INDEX

### SYMBOLS

- $V_i(t)$  : Dynamics of a hidden or output neuron  $i$  at time  $t$
- $C_{mi}$  : Membrane capacitance of neuron  $i$
- $dV_i/dt$  : Rate of change of the internal state of neuron  $i$  with respect to time
- $G_{Leaki}$  : Leak conductance of neuron  $i$
- $V_{Leaki}$  : Leak reversal potential of neuron  $i$
- $n$  : Total number of neurons
- $I_{in}(ij)$  : External current input to neuron  $i$  from neuron  $j$
- $I_{sij}$  : Synaptic current from neuron  $j$  to neuron  $i$
- $w_{ij}$  : Weight of the chemical synapse from neuron  $j$  to neuron  $i$
- $\mu_{ij}$  : Presynaptic membrane state parameter for the sigmoidal nonlinearity
- $\gamma_{ij}$  : Parameter for the sigmoidal nonlinearity
- $E_{ij}$  : Reversal potential of the synapse from neuron  $j$  to neuron  $i$
- $\omega^{ij}$  : Weight of the electrical synapse (gap-junction) between neuron  $j$  and neuron  $i$
- $v_j(t)$  : Membrane potential of neuron  $j$  at time  $t$
- $\sigma_i(V_j(t))$  : Sigmoidal nonlinearity function dependent on the presynaptic membrane state of neuron  $j$
- $\tau_i$  : Time constant of neuron  $i$
- $u(t)$  : Internal states of interneurons (hidden units) and motor neurons (output units) in an LTC RNN at time  $t$
- $W$  : Weight matrix of the LTC RNN
- $\sigma(x)$  : C1-sigmoid function applied element-wise
- $A$  : Vector of resting states of motor and interneurons in the LTC RNN

- $B$  : Vector of synaptic reversals for the motor and interneurons in the LTC RNN
- $\alpha, \beta$  : Range bounds for the entries of A and B
- $\tau$  : Time constant of the LTC RNN system
- $F(x)$  : Mapping function for the autonomous ordinary differential equation
- $x'$  : Derivative of the state vector  $x$  with respect to time
- $\eta$  : Parameter in the range  $(0, \min\{\epsilon, \lambda\})$
- $\epsilon$  : Positive constant for approximation
- $\lambda$  : Distance between the compact subset  $D^\sim$  and the boundary  $\delta S$  of  $S$
- $D_\eta$  : Compact subset of  $S$
- $LF$  : Lipschitz constant of  $F$  on  $D_\eta$
- $\epsilon l$  : Positive constant satisfying  $\epsilon l < (\eta LF) / (2(\exp(LF)T - 1))$
- $N$  : Integer representing the number of hidden units in the LTC RNN
- $x$  : State vector of the  $n$ -dimensional dynamical system
- $R_n$  :  $n$ -dimensional Euclidean space
- $F^\sim(x)$  : Mapping function for the modified LTC RNN system
- $\tau_{\text{sys}}$  : Time constant of the modified LTC RNN system
- $x_0$  : Initial value of the state vector  $x$
- $D$  : Compact subset of the  $n$ -dimensional Euclidean space  $R_n$
- $y^\sim$  : Intermediate variable defined as  $Cx^\sim + \mu$
- $E$  : Matrix representing the product of matrices  $C$ ,  $W_1$ , and  $B$
- $G^\sim(z)$  : Mapping function for the modified LTC RNN system
- $z$  : State vector of the modified LTC RNN system



## KISALTMALAR

3d	: Three Dimensional
AI	: Artificial Intelligence
ANN	: Artificial Neural Network
ART	: Adaptive Resonance Theory
BPTT	: Back Propagation Over Time
CPU	: Central Processing Unit
DSS	: Decision Support System
DAE	: Differential Algebraic Equations
HPC	: High Performance Computing
SGD	: Stochastic Gradient Descent
IO	: Input or/and Output
KNN	: K-Nearest Neighbor
LSTM	: Long-Short Term Memory
LTC <sup>1</sup>	: Liquid Neural Network / Liquid Time-Constant Neural Network
LTC-SE	: Liquid Neural Network for Scalable AI and Embedded Systems
ODE	: Ordinary Differential Equation
PID	: Proportional-Integral-Derivative
RAM	: Random Access Memory
RNN	: Recurrent Neural Network
SVM	: Support Vector Machine
TCN	: Temporal Convolutional Networks
xAI	: Explainable Artificial Intelligence

---

<sup>1</sup> Liquid neural networks (LNNs) are also known as liquid time-constant (LTC) networks. They are a type of continuous-time neural network model that uses linear first-order dynamical systems modulated via nonlinear interlinked gates [39] [37].

## **CHAPTER 1**

### **INTRODUCTION**

The source of inspiration for inventing and designing artificial intelligence and machine learning systems has always been based on natural intelligence. It can be safely said that one of the most complex intelligent systems is the human brain. Designing an artificial neural network inspired by the temporal lobe of the brain with the ability to classify and solve regression problems, or inventing Convolutional Neural Networks inspired by the occipital lobe to provide solutions for problems related to machine vision, can be mentioned among these efforts.

Recent progress in the field of developing algorithms and methods of artificial intelligence and machine learning by modeling neural networks and the brain has become highly competitive. This competition is due to the ability to learn and high adaptability in facing non-linear and complex problems. Factors such as learning speed, learning cost, effectiveness of learning rate under noisy data effect, the amount of ground-truth data required to achieve the desired accuracy and the probability of encountering the vanishing point problem are among the competitive criteria presented between these new methods. Recently, in 2021, LTC was presented by R. M. Hasani. The presented type of neural networks exhibits stable bounded behavior, yields good expressiveness within the family of ordinary neural differential equations, and improves performance on time series forecasting tasks. This method is inspired by the principles of communication in the nervous system of species. It allows continuous mapping approximation models with a small number of computational units.

Here, researchers introduced a new functionality of the recently presented Liquid Neural Network (LTC), which is to model decision support systems variables. To benchmark and test the achievements, a basic multi-joint arm controlling system will

be implemented using the proposing algorithm. The proposed algorithm is highly adaptable with a wide array of input data to perform in different environments with desired accuracy. The resulting model represents a dynamic system with liquid time constants that vary with their hidden states, with outputs computed using differential equation solvers and fraction. A multi-joint arm is made of independent joints and rigids. The algorithm will control the joints angles and positioning to control the arm to reach the targeted situation or to grab an object. Finally, the evaluation of its performance in controlling multi-joint arm robot using the developed decision support system will be demonstrated using multi-variable benchmarking.

### **1.1. MOTIVATION AND PROBLEM STATEMENT**

Traditional neural networks and machine learning algorithms have several fundamental problems to work on. Missing values in input data for training the model and extracting the features are a problem that may be faced during work in traditional neural networks. This is effectively changing the performance rate of the model generated by the neural network. The traditional neural networks are not able to recognize complex patterns in the data due to lack of connection between multiple layers to change of nodes vertical and horizontal nodes. The problem expands with limiting the traditional neural networks only working well in few-steps forecast, not in long term forecasting.

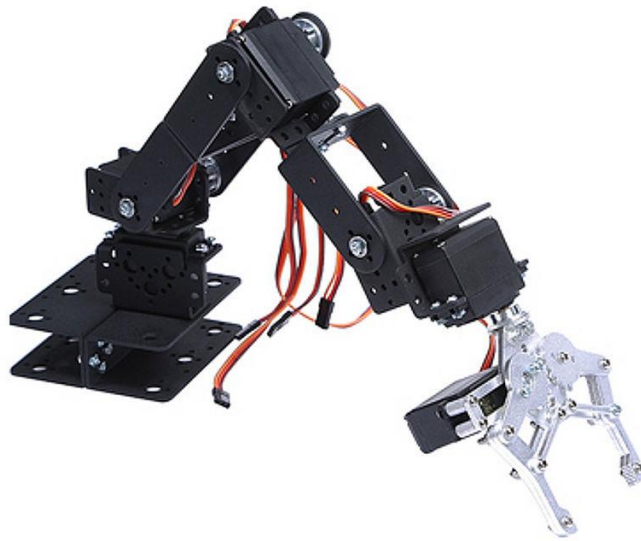


Figure 1.1. An instance of chained mechanical robot consisting of joints, rigids, actuators and electronics

- ✎ The manipulator algorithm for chained mechanical robots, such as autonomous arm robot, are notions of forward and inverse kinematics.
- ✎ The forward kinematics is given all the angles or the translational degrees of motion of your robot like all motors and actuators positions. The combination of these positions is going to lead to the gripper or the end effector of the arm, being in a certain position.
- ✎ So forward kinematics, is going from your joint space to the physical position in 3d of the robot.
- ✎ Inverse kinematics is a tougher problem, which is the opposite; it's if you want the robot to be in a specific position and orientation in the physical world. Like how to position the actuators.
- ✎ The forward kinematics: This problem usually involves some sort of iterative algorithm.
- ✎ The inverse kinematics: To make a model for this kind of problem, usually the Rigid Bodies Tree is used, and Artificial Neural Network (ANN) is used for decision making and autonomous robot manipulation.

## **1.2. OBJECTIVE**

To apply a Liquid neural network for decision making for a time-series multiple joints to control, using TensorFlow framework.

## **1.3. PROJECT OBJECTIVES**

We reached the following objectives after completion of this study (and implementation of the solution):

- ✎ Expressivity of neural ordinary differential equation networks (ODE-Nets) in their current formalism, compared to the LTC.
- ✎ Improvements of LTC structure to enable better representation learning, based on the subject.

## **1.4. THE IMPORTANCE OF THE OBJECTIVE**

Here we highlight the importance of LTC recurrent neural networks in various applications. Firstly, LTC networks are developed as neural state-based data processing systems inspired by the brain, with continuous-time semantics. This enables them to effectively process information in a manner similar to how the brain operates. Secondly, thorough theoretical stability and universality analyses of LTCs are conducted to ensure their reliability and suitability for control and decision support system designs using explainable states demonstration system. Thirdly, the superior expressivity of LTCs compared to other types of recurrent neural networks is illustrated, particularly in modeling time-series data. This demonstrates their ability to capture complex temporal patterns. Additionally, the objective is to introduce novel network-design principles for LTC neuronal models and equip them with a search-based learning algorithm, enabling them to effectively control robotic tasks. Lastly, a lightweight dynamical systems-based algorithm is designed and implemented to systematically interpret the behavior of RNNs, specifically focusing on response characterization. This helps to uncover insights and understand the inner

workings of these networks. Overall, these objectives emphasize the significance of LTC networks in advancing various fields of research and applications.

## **CHAPTER 2**

### **LITERATURE REVIEW**

Control systems play a crucial role in various industries and applications, facilitating the regulation and control of processes to achieve desired outcomes. Control system design has evolved significantly over time, transitioning from traditional algorithms to advanced approaches utilizing neural networks. This literature review explores the development of control system design, focusing on the limitations of classical algorithms in handling time-continuous data and the emergence of RNNs and LSTMs networks. By examining relevant literature and research, this article aims to provide insights into the reasons behind the adoption of RNNs, LSTM, and related approaches in control system design.

#### **2.1. FUNDAMENTAL ALGORITHMS**

In the early stages of control system design, engineers heavily relied on classical algorithms, particularly the PID control algorithm [1]. While effective in many applications, these algorithms faced limitations when dealing with complex and time-continuous processes.

#### **2.2. LIMITATIONS OF TRADITIONAL ALGORITHMS**

As control systems became more intricate, classical algorithms encountered difficulties in handling time-continuous data streams. Real-time control systems operate in dynamic and unpredictable environments where continuous data must be processed rapidly and efficiently. However, traditional algorithms exhibited limitations due to fixed parameter settings and lack of inherent memory [2].

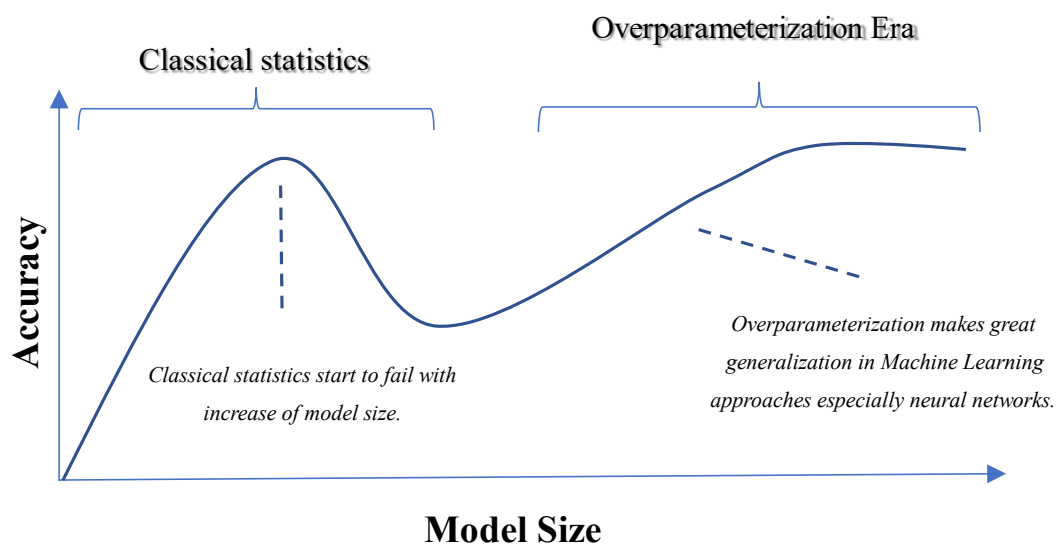


Figure 2.1. Comparison between Classical Statistics and Overparameterization Era

Unlike traditional control system methods that rely on statistical approaches, machine learning methods using neural networks offer a distinct advantage in handling a large number of parameters. While traditional methods may suffer from decreased accuracy as the model size increases, neural networks excel in overparameterization [3]. This characteristic allows them to be trained more effectively and achieve higher levels of accuracy. Consequently, neural network-based machine learning methods prove highly suitable for tackling intricate control system applications due to their ability to handle an extensive array of parameters. Overparameterizing a neural network increases its complexity and flexibility, leading to a higher risk of overfitting. To address this, regularization techniques, such as L1 and L2 regularization, are used to strike a balance between bias and variance, mitigating overfitting by controlling the model's complexity.

### 2.3. THE NEURAL NETWORKS

To overcome the limitations of traditional algorithms, researchers used neural networks, which draw inspiration from the human brain's ability to process and learn from data. Neural networks consist of interconnected nodes, or artificial neurons, organized in layers. Neural networks excel in learning from data, adapting to changing environments, and effectively handling time-continuous data [4].



## **2.4. NEURAL AND EVOLUTIONARY COMPUTING**

Evolutionary computation is the domain of optimization theory, where instead of using classical numerical methods to solve optimization problems, inspiration from biological evolution is used to "design" good solutions. When dealing with scenarios where the derivative of the fitness function is unknown, as in reinforcement learning, or when the fitness function exhibits numerous local extrema and involves sequential methods, evolutionary computation is often employed as a preferred approach over standard numerical methods. Applications of evolutionary computing are numerous, including solving optimization problems, designing robots, building decision trees, optimizing data mining algorithms, training neural networks, and hyperparameter optimization. Data science and machine learning models, and nearly all statistical and "black box" models are designed to solve optimization problems. There are methods for estimating the capabilities of implemented evolutionary computing algorithms. The goal is to make sure these values are minimized.

### **2.4.1. Biological Neural System**

The four lobes of the brain are the frontal, parietal, temporal, and occipital lobes (**Error! Reference source not found.**). The frontal lobe is located in the forward part of the brain, extending back to a fissure known as the central sulcus. The frontal lobe is involved in reasoning, motor control, emotion, and language. It contains the motor cortex, which is involved in planning and coordinating movement; the prefrontal cortex, which is responsible for higher-level cognitive functioning; and Broca's area, which is essential for language production [5].

#### **2.4.1.1. The temporal lobe of our brain**

The temporal lobe of the human brain, responsible for long-term memory, can be compared to artificial neural networks that are commonly used for classification and regression tasks.

### 2.4.1.2. The occipital lobe

The occipital lobe, which is associated with vision, can be analogous to convolutional neural networks (CNNs) primarily utilized for computer vision problems. However, it is worth noting that temporal convolutional networks (TCNs) can also be applied to analyze time series data.

### 2.4.1.3. The frontal lobe

The frontal lobe, known for its role in short-term memory, shares similarities with recurrent neural networks (RNNs) that are widely used for analyzing sequences, lists, and time series. For example, in language processing, RNNs are employed to process sequences of characters, words, and sentences following specific grammatical rules. Similarly, RNNs can effectively analyze time series data comprising sequential observations.

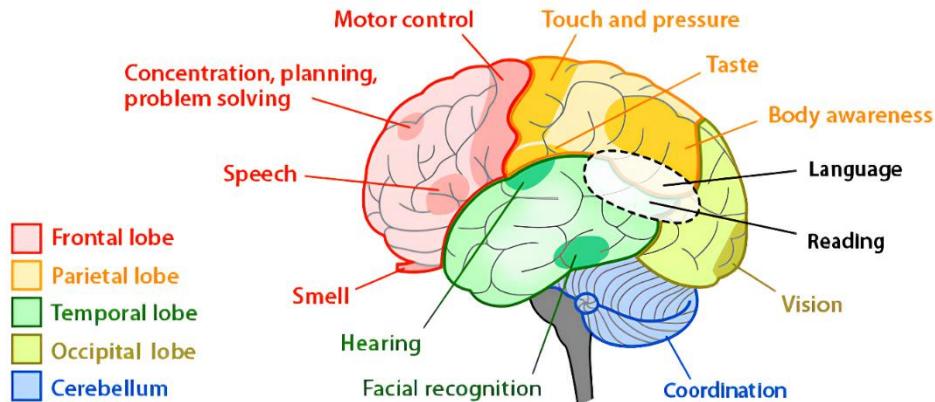


Figure 2.2. Brain lobes and their known major functionality [6]

## 2.4.2. Artificial Neural Network (ANN)

The ANNs are software and or hardware implementations of the neuronal structure of the brains. That means artificial neural network attempt to simplify and mimic this brain behavior [7] [8] [9]. In a certain type of artificial neural network (ANN) training called supervision, the network learns by utilizing paired input and output data samples. The ultimate goal is to enable the ANN to generate the desired output

when given a specific input. On the other hand, unsupervised learning within an ANN aims to facilitate the network in comprehending the inherent structure of the input data without external guidance [10].

#### **2.4.2.1. The artificial neuron**

An artificial neural network (ANN) emulates the behavior of a biological neuron through the utilization of an activation function. In tasks such as email spam identification, the activation function employed in the network must exhibit a distinctive "switch on" feature. Essentially, when the input surpasses a specific threshold, the output of the function should transition from one state to another, such as from 0 to 1, from -1 to 1, or from 0 to a value greater than 0. This emulation imitates the process of a biological neuron being activated. One frequently utilized activation function for this purpose is the sigmoid function.

$$f(z) = \frac{1}{1 + \exp(-z)} \quad (2.1) \text{ The sigmoid function}$$

#### **2.4.2.2. Nodes**

As previously discussed, biological neurons form interconnected hierarchical networks, where the outputs of certain neurons serve as inputs for others. To represent these networks, we can utilize connected layers of nodes. Each node within these layers receives multiple inputs with associated weights, applies the activation function to the sum of these inputs, and generates an output as a result of this process [11]. Consider the diagram below:

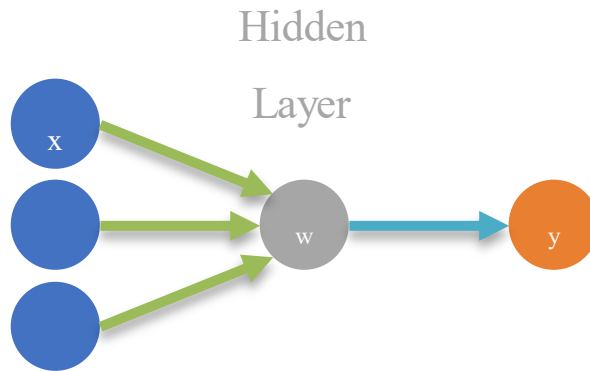


Figure 2.3. Node with inputs (x), weights (w), output (y) <sup>2</sup>

In the provided image, the circular shape corresponds to the representation of a node. This node acts as the "processing element" or "unit" responsible for implementing the activation function. It takes the weighted inputs, adds them together, and then feeds them into the activation function.

The weights associated with the inputs are real-valued numbers, meaning they are not restricted to binary values of 1 or 0. These weights are multiplied with the inputs and subsequently aggregated within the node. Hence, to describe the weighted input to the depicted node, it can be expressed as follows:

$$x_1w_1 + x_2w_2 + x_nw_n + b \quad (2.2)$$

Here the  $w_i$  values are weights. These variables undergo modifications during the learning process and, in conjunction with the input, govern the node's output. The bias element, denoted as "b," is responsible for introducing a desirable level of flexibility to the node. This augmentation is most effectively illustrated through the use of an illustrative example.

---

<sup>2</sup> A perceptron neural network.

## **2.5. LEARNING ALGORITHM SELECTION**

Selecting an appropriate machine learning algorithm can pose a challenging task.

### **2.5.1. Resources and validity**

Considering the constraints of time and resources, it is highly impactful to conduct extensive testing across various algorithms [12]. However, practical limitations often impose restrictions on the time and resources available for problem-solving. Researchers can address this by asking themselves a series of questions before embarking on the task. Based on their answers, they can narrow down the list of algorithms and experiment with them using the available data. [13].

#### **2.5.1.1. Explainability**

Does the model have to be explainable to a non-technical audience?

Many highly accurate learning algorithms are considered "black boxes." While these models exhibit minimal error rates, understanding the specific reasoning behind their predictions can be extremely challenging and often defies explanation. Neural networks and ensemble models exemplify such algorithms. On the other hand, kNN, linear regression, and decision tree learning algorithms generate models that may not always be the most accurate, but they offer a straightforward approach to prediction.

#### **2.5.1.2. In-memory vs. out-of-memory**

Is it possible to load the dataset entirely into the server or personal computer's RAM? If affirmative, researchers can explore a wide range of algorithms. Otherwise, they may prefer incremental learning algorithms that gradually enhance the model by incorporating additional data.

#### **2.5.1.3. Number of features and examples**

How many training examples does the researcher possess in the dataset related to the problem? Additionally, what is the number of features per example? Certain

algorithms, such as neural networks and gradient boosting, can handle large volumes of examples and millions of features. Conversely, others like SVMs may have more modest capacities.

#### **2.5.1.4. Categorical vs. numerical features**

Does the dataset solely consist of categorical or numerical features, or is it a mixture of both? Depending on the answer, some algorithms may not be directly compatible with the dataset. Researchers would need to employ techniques like one-hot encoding to convert categorical features into numerical representations.

#### **2.5.1.5. Nonlinearity of the data**

Can the data be linearly separated or modeled using a linear approach? In the affirmative case, algorithms such as SVM with a linear kernel, logistic regression, or linear regression can be suitable choices. Conversely, deep neural networks or ensemble algorithms may deliver better results for non-linear data.

#### **2.5.1.6. Training speed**

How much time can be allocated for the learning algorithm to construct a model? Training neural networks is known to be time-consuming. In contrast, simpler algorithms like logistic and linear regression, as well as decision tree learning, offer faster training speeds. Specialized libraries with efficient implementations of specific algorithms can be sought online. Algorithms like random forests benefit from utilizing multiple CPU cores, resulting in significant reductions in model building time on machines equipped with numerous cores.

#### **2.5.1.7. Prediction speed**

What is the required prediction speed of the model? Will the model be deployed in a production environment where high throughput is crucial? Certain algorithms, such as SVMs, linear and logistic regression, or specific types of neural networks, excel in

swift prediction times. Conversely, algorithms like kNN, ensemble methods, and deep or recurrent neural networks may exhibit slower prediction speeds [13].

### **2.5.2. Control Systems and Neural Networks:**

There are three options to solve fundamental problems of traditional neural networks and machine learning algorithms such as, missing values can really affect the performance of the models; not being able to recognize complex patterns in the data and usually not suitable in long term forecast.

Implementation of Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) time-constant neural networks are introduced as a solution for the mentioned problems. In this section, researchers will introduce and provide the essential keywords descriptions for the audience to cover the thesis project. The new neural network has a great impact on how we process time-series data. The LTC can anticipate future behavior in the system by analyzing data in real-time. In addition to addressing the fundamental problems of traditional neural networks and machine learning algorithms, control systems can further enhance the models by incorporating action models. Action models provide a framework for integrating control actions into the neural network architecture, enabling the models to not only analyze data in real-time but also actively influence and shape future behavior in the system. This combination of time-constant neural networks and action models revolutionizes the processing of time-series data, offering significant advancements in forecasting and control capabilities.

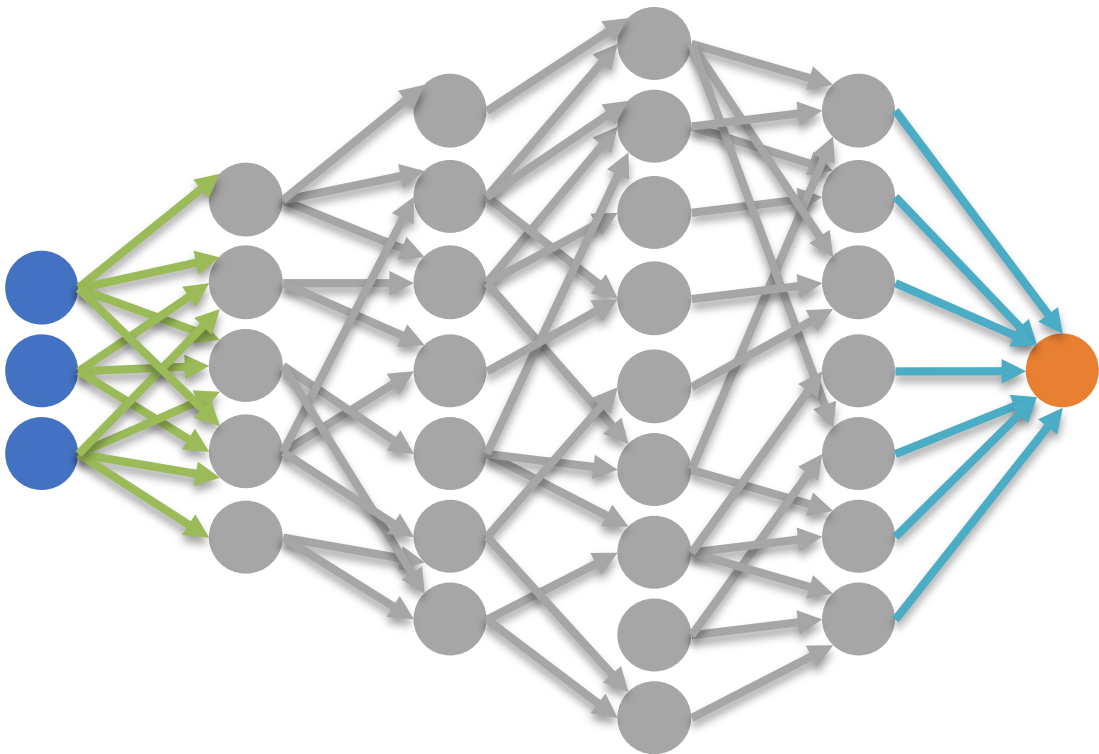


Figure 2.4. Demonstration of a Multi-Layer Feed-Forward NN

### 2.5.3. Recurrent Neural Networks (RNNs):

The introduction of RNNs marked a significant breakthrough in control system design. RNNs possess a unique architecture that incorporates feedback connections among network nodes, enabling the capture of temporal dependencies [14]. By retaining memory of past inputs and leveraging this information for predictions based on previous states, RNNs proved effective in handling sequential data, including time-continuous processes.



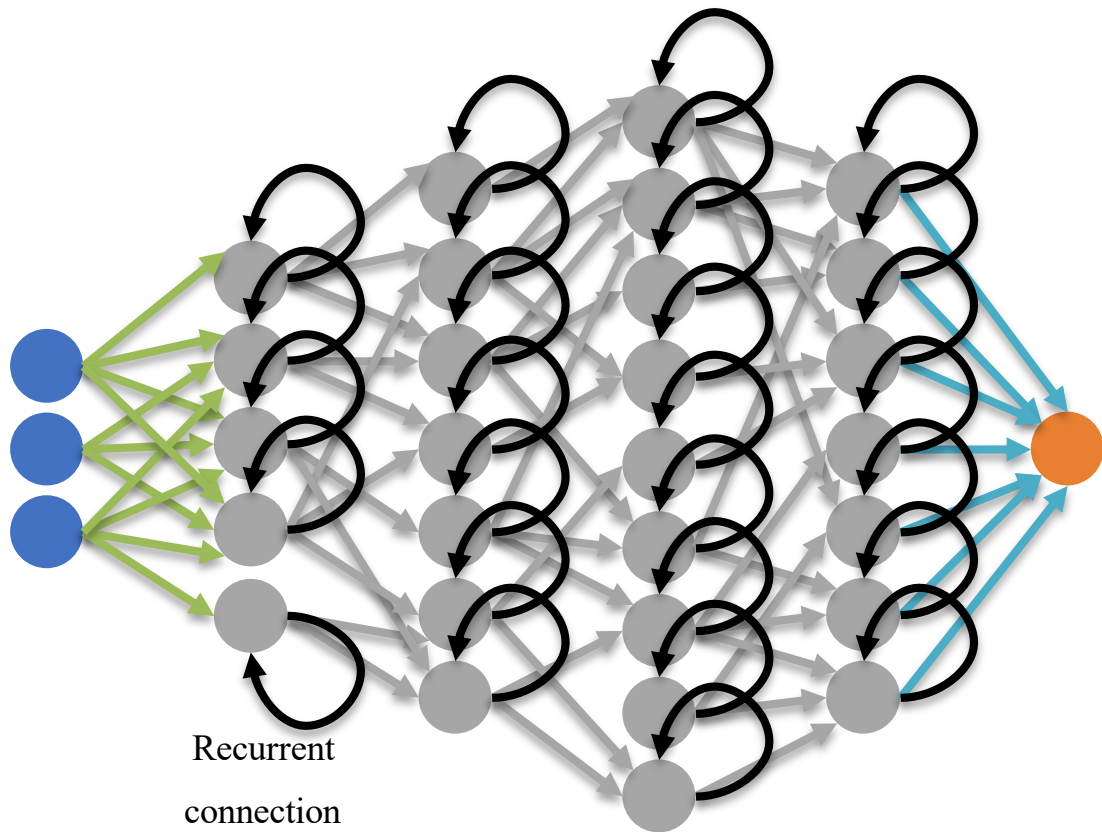


Figure 2.5. Demonstration of a Multi-Layer RNN Architecture

A recurrent neural network is a neural network based on processing the data sequence  $x(t) = x(1), \dots, x(\tau)$  with time step index  $t$  ranging from 1 to  $\tau$ . It is commonly used in speech recognition and natural language processing [15]. Iterative neural networks recognize continuous properties in data and use patterns to predict the next scenario. RNNs remember all information over time. Useful for time series forecasting only because of its ability to remember previous inputs as well [16] [17].

#### 2.5.4. RNN and NVIDIA CUDA

The NVIDIA® CUDA® Toolkit provides a development environment for creating high performance GPU-accelerated applications. With the CUDA Toolkit, you can develop, optimize, and deploy your applications on GPU-accelerated embedded systems, desktop workstations, enterprise data centers, cloud-based platforms and HPC supercomputers. The toolkit includes GPU-accelerated libraries, debugging and optimization tools, a C/C++ compiler, and a runtime library to deploy your

application [18]. Using built-in capabilities for distributing computations across multi-GPU configurations, scientists and researchers can develop applications that scale from single GPU workstations to cloud installations with thousands of GPUs. Here in this research we are using the advancement of NVIDIA multi processing and GPU-accelerated mechanisms to enhance the speed of the process.

#### **2.5.4. Emergence of LSTM Networks:**

While RNNs showed promise, they faced challenges in capturing long-term dependencies, especially in time-continuous processes. To address this issue, LSTM networks were introduced [14] [19]. LSTM networks feature memory cells that selectively retain or forget information over extended sequences, enabling them to overcome the vanishing gradient problem that hindered the learning of long-term dependencies in traditional RNNs. The architectural modifications in LSTM networks empowered them to excel in capturing long-term dependencies and efficiently handling time-continuous data.

#### **2.5.5. The Role of LSTM in Control Systems:**

LSTM networks have demonstrated exceptional performance in control system design, particularly in scenarios involving long-term dependencies and time-continuous data. By effectively capturing and retaining essential information over extended periods, LSTM networks enable accurate predictions and precise control actions [20]. This characteristic makes LSTM networks highly suitable for applications such as autonomous vehicles, robotics, and process control, where real-time decision-making is crucial.

#### **2.5.6. Other LSTM-Related Approaches and Recent Research:**

In addition to LSTM networks, researchers have proposed various approaches to enhance control system design. One notable example is the Gated Recurrent Unit (GRU), which exhibits similarities to LSTM networks but with a simplified architecture [21]. GRU networks have demonstrated comparable performance to

LSTM networks while being computationally more efficient, making them an appealing alternative for specific applications.

Moreover, ongoing research in control system design explores hybrid control systems that combine classical algorithms with neural networks, reinforcement learning-based control strategies, and attention mechanisms [22] [23]. These advancements aim to address specific challenges in control system design, such as improved adaptability, robustness, and efficiency. The evolution of control system design from traditional algorithms to recurrent neural networks has revolutionized the field, enabling more effective handling of time-continuous data. The emergence of LSTM networks has further enhanced control system performance by addressing the challenges associated with capturing long-term dependencies. Ongoing research efforts continue to explore innovative techniques, including hybrid control systems and reinforcement learning, to further improve the efficiency and adaptability of control systems. By leveraging the power of recurrent neural networks and staying at the forefront of emerging research, control system engineers can drive the development of safer, more efficient, and intelligent systems across a broad range of applications.

### **2.5.7. Time-Constant Neural Networks**

Time-Constant Neural Networks (TCNNs) emerged as a complementary approach to LSTM networks, even though LSTM networks were already introduced. While LSTM networks are renowned for capturing long-term dependencies and handling sequential data, TCNNs offer distinct advantages in specific control system applications. TCNNs incorporate time constants directly into their architecture, allowing them to explicitly model and capture the temporal dynamics of a system. This characteristic proves valuable in scenarios involving time-delayed and dynamic control tasks, where considering time constants becomes critical. By explicitly accounting for time constants, TCNNs can effectively retain and utilize historical information within a defined time span, leading to enhanced performance in control systems that entail intricate and time-varying dynamics. The advent of TCNNs, therefore, introduces an additional tool in the neural network repertoire,

complementing the capabilities of LSTM networks and providing an alternative approach for modeling and controlling control systems.

TCNNs have gained recognition as a valuable approach for control systems across diverse fields. TCNNs, a type of recurrent neural network, incorporate time constants into their architecture to effectively capture and model the temporal dynamics of a system. This unique feature allows TCNNs to handle time-delayed and dynamic control tasks proficiently. By explicitly considering the system's time-dependent behavior, TCNNs can retain historical information over a defined time span, enabling them to make informed decisions based on past inputs. As a result, TCNNs have found practical applications in robotics, process control, and autonomous vehicles, where complex and time-varying dynamics are prevalent. The adaptability and learning capabilities of TCNNs in the presence of time-dependent data contribute to achieving accurate and responsive control, thereby enhancing the overall performance and stability of control system applications.

#### **2.5.8. Liquid Neural Network or Liquid Time-Constant NN (LTC)**

The LTC is an unconventional artificial neural network inspired by the brain's information processing mechanisms. It gets its name from its unique interconnected structure, resembling a liquid, where all neurons are intricately connected. This liquid-like state enables dynamic and adaptable information processing, making it ideal for tasks such as pattern recognition, prediction, and control. LTC offers several advantages over other neural networks, including LSTM. One notable advantage is its highly parallel and distributed processing capabilities. Unlike traditional layered networks, LTC operates in a liquid state, with connections between neurons constantly changing. This parallelism allows LTC to efficiently process information in a massively parallel manner, facilitating fast computations. This is achieved through the use of recurrent connections, which allow information to flow in both forward and backward directions. Unlike traditional feedforward networks with fixed connections, liquid neural networks exhibit plasticity, meaning that the strength and structure of connections between neurons can adapt and evolve over time based on the input and network dynamics. Another key advantage of LTC lies in its ability to

effectively handle temporal information and capture dynamic patterns. LTC incorporates time constants into its architecture, making it well-suited for modeling and analyzing dynamic systems [24]. This feature allows LTC to explicitly account for temporal dynamics, making it particularly useful in control systems that involve time-delayed and time-varying dynamics.

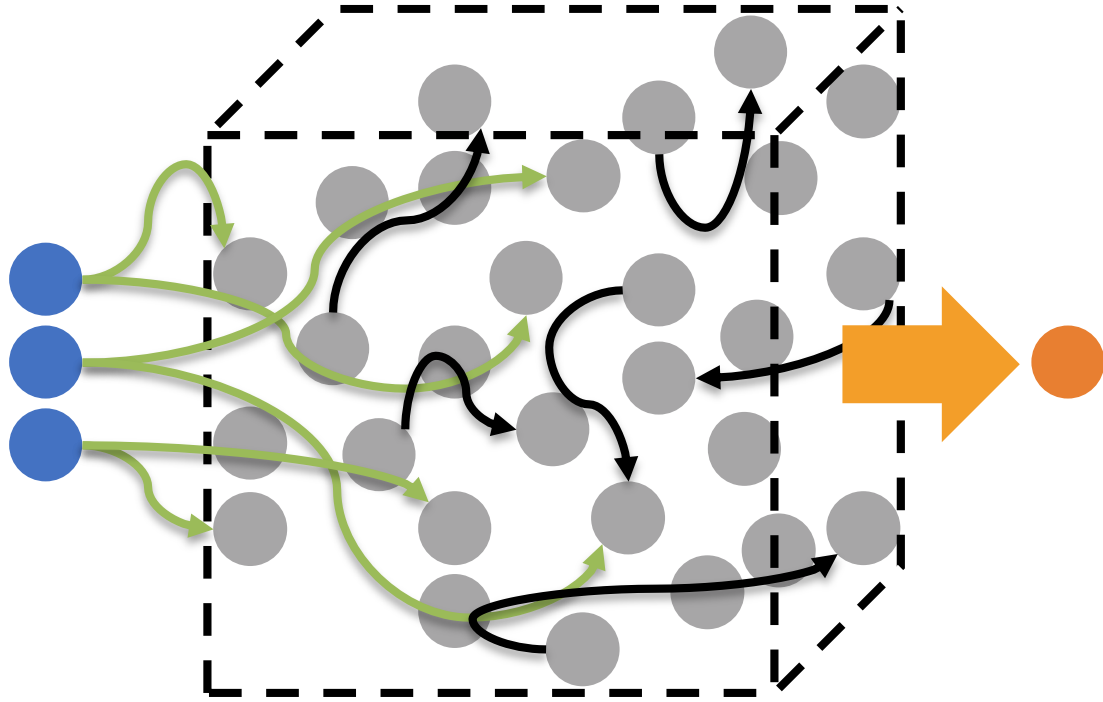


Figure 2.6. Demonstration of Liquid Neural Network (LTC) Architecture

Furthermore, LTC's liquid state and parallel processing lend themselves to enhanced computational efficiency. The network can rapidly adapt to changing input signals and learn complex patterns in real-time. This characteristic makes LTC highly suitable for time-sensitive applications where responsiveness and accuracy are paramount. LTC recurrent neural networks (RNN)s are a subclass of continuous-time RNNs, with varying neuronal time-constant realized by their nonlinear synaptic transmission model. This feature is inspired by the communication principles in the nervous system of small species. It enables the model to approximate continuous mapping with a small number of computational units [24]. In summary, the LTC distinguishes itself through its parallel and distributed liquid state, its capability to capture dynamic patterns, and its computational efficiency. These advantages

position LTC as a potent tool for various applications, including pattern recognition, prediction, and control systems dealing with time-dependent dynamics.

### **2.5.8. Leveraging Ode Solvers For Dynamic System Modeling**

In the realm of neural networks, the integration of ODE Solvers has proven instrumental in expanding the capabilities of these models, particularly in architectures like the LTC discussed earlier. ODE Solvers are pivotal algorithms designed to approximate solutions to ordinary differential equations (ODEs) numerically. Their significance lies in enabling the simulation and analysis of dynamic systems with continuous-time dynamics, thereby enhancing the functionality of neural networks. ODE Solvers serve a vital role in LTC and similar models by facilitating the modeling of neural networks as a set of differential equations. These equations encapsulate the network's internal state evolution over time, effectively capturing its dynamic behavior and enabling information processing and prediction. The role of ODE Solvers comes into play when solving these differential equations and determining the state variables of the network at discrete time steps.

Functionally, ODE Solvers excel in integrating differential equations and calculating the rates of change for the network's state variables. This involves approximating the continuous dynamics of the network into discrete time steps, enabling efficient computation. ODE Solvers employ various numerical techniques like Euler's method or Runge-Kutta methods to iteratively update the state variables based on the input signals and differential equations. By harnessing the power of ODE Solvers, neural networks, including LTC, gain the ability to effectively capture temporal dependencies and dynamic patterns within data. The continuous-time dynamics encapsulated by the differential equations empower the network to adapt and respond to changing input signals, making it a potent tool for analyzing complex systems.

The integration of ODE Solvers with neural networks demonstrates the symbiotic relationship between mathematical modeling and machine learning techniques. This fusion allows for the simulation and control of time-dependent systems, opening up

avenues in robotics, control systems, and predictive modeling. This integration enriches the functionality of neural networks by enabling the simulation and analysis of continuous-time dynamics. These solvers provide a means to model and comprehend the behavior of dynamic systems by numerically approximating solutions to differential equations. The utilization of ODE Solvers within neural networks, such as LTC, unlocks the potential for advanced control systems and predictive modeling, bridging the gap between mathematical modeling and machine learning. As a simplified instance for Ordinary Differential Equations, suppose we want to model the trajectory of a ball thrown into the air. We can use a Neural ODE to model the trajectory of the ball as it moves through the air. The Neural ODE would take as input the initial position and velocity of the ball and output the position and velocity of the ball at any given time.

## **2.6. CONCEPT AND FUNCTIONALITIES**

These neural networks exhibit stable and bounded behavior, yield superior expressivity within the family of neural ODE, and give rise to improved performance on time-series prediction tasks. To demonstrate these properties, in 2021 scientists took a theoretical approach to find bounds over their dynamics and compute their expressive power by the trajectory length measure in a latent trajectory space. They then conduct a series of time-series prediction experiments to manifest the approximation capability of LTCs compared to classical and modern RNNs [25].

## **2.7. APPLICATIONS**

- ✎ The new neurons will have a great impact on how we process time-series data. Researchers believe the world is all about sequences.
- ✎ The LTC can anticipate future behavior in the system by analyzing data in real-time.
- ✎ Researchers could see the use of LTC in medical diagnosis and self-driving vehicles.

## 2.8. THE IMPLEMENTATION PROCESS

### I. Code implementation using Python

- Python allows for the development and operation of software solutions across a variety of platforms and operating systems. Linux, Windows, Mac, Solaris, and more are a few examples. Python machine learning programming is now much more practical as a result. Because of this, Python is selected for the proposed algorithm to be implemented with.

### II. Algorithm development using TensorFlow

- TensorFlow has a reputation for simplicity, ease of use, flexibility, efficient memory usage using parallel processing and compression methods, and dynamic computational graphs. It also feels native, making coding more manageable and increasing processing speed.

### III. Dataset will be joints 3d position data.

- A multi-joint arm is made of independent joints and rigids. The algorithm will control the joints angles and positioning to control the arm to reach the targeted situation or to grab an object. The length of the rigids and angle of joints can be addressed using a simulation to make the ground truth positioning dataset.



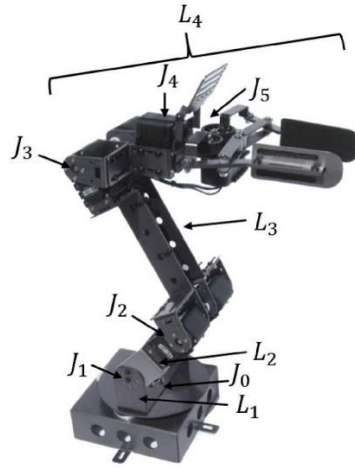


Figure 2.7. An instance for a multi-joints arm system

- The data frame will be as follows:

$$\{[J_0] [L_1, J_1] [L_2, J_2] \dots [L_n, J_n]\}$$

#### IV. The algorithm input:

- Train dataset: joints degree of movement and gestures,
- Test dataset: 4 random gestures to be predicted by joints alignment and settings.

#### V. Algorithm optimization for being scalable.

- Refining the method (Reducing the size of the reservoir and the default ODE solver process)
- Enhanced adaptability (Model compression)
- Optimized performance based on resources (Quantization)
- Efficiency for embedded systems with limited resources (Hardware acceleration)

#### VI. Final Test

- The entire algorithm will be tested using comparison between the predicted outcomes and real-world data.

## VII. Success-rate Estimation

- Using a comparison between proposed algorithm's settings, output and accuracy and the other similar algorithms output, the percentage of differences between outcome and ground-truth will be estimated. Here two other famous algorithms in this field is implemented for this reason.

## CHAPTER 3

### THEORETICAL BACKGROUNDS

#### 3.1. DYNAMIC EQUATIONS AND IO

Dynamics of a hidden or output neuron  $i$ ,  $V_i(t)$ , of an LTC RNN are modeled as a membrane integrator with the following ordinary differential equation (ODE):

$$C_{m_i} \frac{dV_i}{dt} = G_{Leak_i} (V_{Leak_i} - V_i(t)) + \sum_{j=1}^n I_{in}^{(ij)} \quad (3.3)$$

with neuronal parameters:  $C_{m_i}$ ,  $G_{Leak_i}$  and  $V_{Leak_i}$ .  $I_{in}^{(ij)}$  represents the external currents to the cell. Hidden nodes are allowed to have recurrent connections while they synapse into motor neurons in a feed-forward setting. Chemical synapses – Chemical synaptic transmission from neuron  $j$  to  $i$ , is modeled by a sigmoidal nonlinearity  $(\mu_{ij}, \gamma_{ij})$ , which is a function of the presynaptic membrane state,  $V_j(t)$ , and has maximum weight of  $w_i$ :

$$I_{s_{ij}} = \frac{w_{ij}}{1 + e^{-\gamma_{ij}(V_j + \mu_{ij})}} (E_{ij} - V_i(t)) \quad (3.4)$$

The synaptic current,  $I_{s_{ij}}$  is then linearly depends on the state of the neuron  $i$ .  $E_{ij}$  sets whether the synapse excites or inhibits the succeeding neuron's state. An electrical synapse (gap-junction), between node  $j$  and  $i$ , was modeled as a bidirectional junction with weight,  $\hat{\omega}_{ij}$ , based on Ohm's law:

$$\hat{I}_{ij} = \hat{\omega}_{ij} (v_j(t) - v_i(t)) \quad (3.5)$$

Internal state dynamics of neuron  $i$ ,  $V_i(t)$ , of an LTC network, receiving one chemical synapse from neuron  $j$ , can be formulated as:

$$\frac{dV_i}{dt} = \frac{G_{Leak_i}}{C_{m_i}} (V_{Leak_i} - V_i(t)) + \frac{w_{ij}}{C_{m_i}} \sigma_i(V_j(t)) (E_{ij} - V_i), \quad (3.6)$$

Where  $\sigma_i(V_j(t)) = 1/1 + e^{-\gamma_{ij}(V_j + \mu_{ij})}$  (Equation 7). If we set the time constant of the neuron  $i$  as:  $\tau_i = \frac{C_{m_i}}{G_{Leak_i}}$  (Equation 8) we can reform this equation as follows:

$$\frac{dV_i}{dt} = - \left( \frac{1}{\tau_i} + \frac{w_{ij}}{C_{m_i}} \sigma_i(V_j) \right) V_i + \left( \frac{V_{leak}}{\tau_i} + \frac{w_{ij}}{C_{m_i}} \sigma_i(V_j) E_{ij} \right) \quad (3.9)$$

( 3.9 presents an ODE system with a nonlinearly varying time-constant defined  $\tau_{system} = \frac{1}{1/\tau_i + w_{ij}/C_{m_i} \sigma_i(V_j)}$  (Equation 3.10) which distinguishes the dynamics of the LTC cells compared to the CTRNN cells.

The overall network dynamics of the LTC RNNs with  $u(t) = [u_1(t), \dots, u_{n+N}(t)]^T$  (Equation 3.11) representing the internal states of  $N$  interneurons (hidden units) and  $n$  motor neurons (output units) can be written in matrix format as follow:

$$\dot{u}(t) = - \left( 1/\tau + W\sigma(u(t)) \right) u(t) + A + W\sigma(u(t))B \quad (3.12)$$

In which  $\sigma(x)$  is  $C^1$  -sigmoid functions and is applied element-wise.  $\tau_{n+N} > 0$  includes all neuronal time constants,  $A$  is an  $n+N$  vector of resting states,  $B$  depicts an  $n+N$  vector of synaptic reversals, and  $W$  is a  $n+N$  vector produced by the matrix multiplication of a weight matrix of shape  $(n+N) \times (n+N)$  and an  $n+N$  vector containing the reversed value of all  $C_{m_i}$  s. Both  $A$  and  $B$  entries are bound to a range:  $[-\alpha, \beta]$  for  $0 < \alpha < +\infty$ , and  $0 \leq \beta < +\infty$ .  $A$  contains all  $V_{leak_i}/C_{m_i}$  and  $B$  presents all  $E_{ij}$  s.

## 3.2. UNIVERSAL APPROXIMATORS OF LTCs

To prove the back-bone theorem behind LTC; which is defining that any given finite trajectory of an n-dimensional dynamical system can be approximated by the internal and output states of an LTC. with n outputs, N interneurons and a proper initial condition.

Let  $x = [x_1, \dots, x_n]^T$  be the n-dimensional Euclidean space on  $\mathbb{R}^n$ .

Researchers base the poof on the fundamental universal approximation theorem [26] on feed-forward neural networks [27] [28] [26], recurrent neural networks (RNN) [28] [29] and time-continuous RNNs [30].

### 3.2.1. Theorem Alpha

Let  $S$  be an open subset of  $\mathbb{R}^n$  and  $F : S \rightarrow \mathbb{R}^n$ , be an autonomous ordinary differential equation, be a  $C^1$ -mapping, and  $\dot{x} = F(x)$  determine a dynamical system on  $S$ . Let  $D$  denote a compact subset of  $S$  and we consider a finite trajectory of the system as:  $I = [0, T]$ . Then, for a positive  $\epsilon$ , there exist an integer  $N$  and an LTC RNN with  $N$  hidden units,  $n$  output units, such that for any given trajectory  $\{x(t); t \in I\}$  of the system with initial value  $x(0) \in D$ , and a proper initial condition of the network, the statement below holds:

$$\max_{t \in I} |x(t) - u(t)| < \epsilon \quad (3.13)$$

#### ✎ Lemma 1

For an  $F: \mathbb{R}^n \rightarrow \mathbb{R}^n$  which is a bounded  $C^1$ - mapping, the differential equation

$$\dot{x} = -(1/\tau + F(x))x + A + BF(x) \quad (3.14)$$

In which  $\tau$  is a positive constant, and  $A$  and  $B$  are constants coefficients bound to a range  $[-\alpha, \beta]$  for  $0 < \alpha < +\infty$ , and  $0 \leq \beta < +\infty$ , has a unique solution on  $[0, \infty)$ .

#### ✎ Proof of Lemma 1

Based on the assumptions, we can take a positive  $M$ , such that

$$0 \leq F_i(x) \leq M (\forall i = 1, \dots, n) \quad (3.15)$$

By looking at the solutions of the following differential equation

$$\dot{y} = -(1/\tau + M)y + A + BM \quad (3.16)$$

We can show that

$$\min \left\{ |x_i(0)|, \frac{\tau(A + BM)}{1 + \tau M} \right\} \leq x_i(t) \leq \max \left\{ |x_i(0)|, \frac{\tau(A + BM)}{1 + \tau M} \right\} \quad (3.17)$$

If we set the output of the max to  $C_{\max i}$  and the output of the min to  $C_{\min i}$  and also set  $C_1 = \min\{C_{\min i}\}$  and  $C_2 = \max\{C_{\max i}\}$ , then the solution  $x(t)$  satisfies

$$\sqrt{n}C_1 \leq x(t) \leq \sqrt{n}C_2 \quad (3.18)$$

Based on *Lemma 2* and *Lemma 3* in [30], a unique solution exists on the interval  $[0, +\infty)$ . *Lemma 1* demonstrates that an LTC network defined by (3.14), has a unique solution on  $[0, \infty)$ , since the output function is bound and  $C^1$ .

### 3.2.2. Proof of Theorem Alpha

For proving *Theorem Alpha*, we adopt similar steps to that of Funahashi and Nakamura on the approximation ability of continuous time RNNs [30], to approximate a dynamical system with a larger dynamical system given by an LTC RNN.

#### ✂ Part 1

We choose an  $\eta$  which is in range  $(0, \min\{\varepsilon, \lambda\})$ , for  $\varepsilon > 0$ , and  $\lambda$  the distance between  $\tilde{D}$  and boundary  $\delta S$  of  $S$ .  $D_\eta$  is set:

$$D_\eta = \{x \in \mathbb{R}^n; \exists z \in \tilde{D}, |x - z| \leq \eta\} \quad (3.19)$$

$D_\eta$  stands for a compact subset of  $S$ , because  $D^\sim$  is compact. Thus,  $F$  is Lipschitz on  $D_\eta$  by *Lemma 1* in [30]. Let  $L_F$  be the Lipschitz constant of  $F|_{K_\eta}$ , then, we can choose an  $\epsilon_l > 0$ , such that

$$\epsilon_l < \frac{\eta L_F}{2(\exp L_F T - 1)} \quad (3.20)$$

Based on the universal approximation theorem, there is an integer  $N$ , and an  $n \times N$  matrix  $B$ , and an  $N \times n$  matrix  $C$  and an  $N$ -dimensional vector  $\mu$  such that

$$\max |F(x) - B\sigma(Cx + \mu)| < \frac{\epsilon_l}{2} \quad (3.21)$$

We define a  $C^1$ -mapping  $\tilde{F}^\sim : \mathbb{R}^n \rightarrow \mathbb{R}^n$  as:

$$\begin{aligned} \tilde{F}(x) = & -(1/\tau + W_l\sigma(Cx + \mu))x + A \\ & + W_l B\sigma(Cx + \mu) \end{aligned} \quad (3.22)$$

with parameters matching that of (3.12 with  $W_l = W$ .

We set the system's time constant,  $\tau_{sys}$  to:

$$\tau_{sys} = \frac{1}{1/\tau + W_l\sigma(Cx + \mu)} \quad (3.23)$$

We chose a large  $\tau_{sys}$ , conditioned with the following:

$$(a) \forall x \in D_\eta; \left| \frac{x}{\tau_{sys}} \right| < \frac{\epsilon_l}{2} \quad (3.24)$$

$$(b) \left| \frac{\mu}{\tau_{sys}} \right| < \frac{\eta L_{\tilde{G}}}{2(\exp L_{\tilde{G}} T - 1)} \text{ and } \left| \frac{1}{\tau_{sys}} \right| < \frac{L_{\tilde{G}}}{2} \quad (3.25)$$

where  $L_{\tilde{G}}/2$  is a Lipschitz constant for the mapping  $W_l\sigma : \mathbb{R}^{n+N} \rightarrow \mathbb{R}^{n+N}$  which we will determine later. To satisfy conditions (a) and (b),  $\tau W_l \ll 1$  should hold true. Then by (3.21) and (3.22), we can prove:

$$\max_{x \in D_\eta} |F(x) - \tilde{F}(x)| < \epsilon_l \quad (3.26)$$

Let's set  $x(t)$  and  $\tilde{x}(t)$  with initial state  $x(0) = \tilde{x}(0) = x_0 \in D$ , as the solutions of equations below:

$$\dot{x} = F(x) \quad (3.27)$$

$$\dot{\tilde{x}} = \tilde{F}(x) \quad (3.28)$$

Based on *Lemma 5* in [30], for any  $t \in I$ ,

$$|x(t) - \tilde{x}(t)| \leq \frac{\epsilon_l}{L_F} (\exp L_F t - 1) \quad (3.29)$$

$$\leq \frac{\epsilon_l}{L_F} (\exp L_F T - 1) \quad (3.30)$$

Thus, based on the conditions on  $\epsilon$ ,

$$\max_{t \in I} |x(t) - \tilde{x}(t)| < \frac{\eta}{2} \quad (3.31)$$

## Part 2

Let's Considering the following dynamical system defined by  $\tilde{F}$  in Part 1:

$$\dot{\tilde{x}} = -\frac{1}{\tau_{sys}} \tilde{x} + A_1 + W_l B \sigma(C \tilde{x} + \mu) \quad (3.32)$$

Suppose we set  $\tilde{y} = C \tilde{x} + \mu$ ; then:

$$\dot{\tilde{y}} = C \dot{\tilde{x}} = -\frac{1}{\tau_{sys}} \tilde{y} + E \sigma(\tilde{y}) + A_2 + \frac{\mu}{\tau_{sys}} \quad (3.33)$$

where  $E = C W_l B$ , an  $N \times N$  matrix. We define:

$$\tilde{z} = [\tilde{x}_1, \dots, \tilde{x}_n, \tilde{y}_1, \dots, \tilde{y}_n]^T \quad (3.34)$$

and we set a mapping  $\tilde{G} : \mathbb{R}^{n+N} \rightarrow \mathbb{R}^{n+N}$  as:

$$\tilde{G}(\tilde{z}) = -\frac{1}{\tau_{sys}} \tilde{z} + W \sigma(\tilde{z}) + A + \frac{\mu_1}{\tau_{sys}} \quad (3.35)$$



$$W^{(n+N) \times (n+N)} = \begin{pmatrix} 0 & B \\ 0 & E \end{pmatrix} \quad (3.36)$$

$$\mu_1^{n+N} = \begin{pmatrix} 0 \\ \mu \end{pmatrix}, A^{n+N} = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \quad (3.37)$$

By using *Lemma 2* in [30], we can show that solutions of the following dynamical system:

$$\dot{\tilde{z}} = \tilde{G}(\tilde{z}), \tilde{y}(0) = C\tilde{x}(0) + \mu \quad (3.38)$$

are equivalent to the solutions of the (3.32).

Let's define a new dynamical system  $G : \mathbb{R}^{n+N} \rightarrow \mathbb{R}^{n+N}$  as follows:

$$G(z) = -\frac{1}{\tau_{sys}}z + W\sigma(z) + A \quad (3.39)$$

where  $z = [x_1, \dots, x_n, y_1, \dots, y_n]^T$ . Then the dynamical system below

$$\dot{z} = -\frac{1}{\tau_{sys}}z + W\sigma(z) + A \quad (3.40)$$

Can be realized by an LTC RNN, if we set  $h(t) = [h_1(t), \dots, h_N(t)]^T$  as the hidden states, and  $u(t) = [U_1(t), \dots, U_n(t)]^T$  as the output states of the system. Since  $\tilde{G}$  and  $G$  are both  $C^1$ -mapping and  $\sigma'(x)$  is bound, therefore, the mapping  $\tilde{z} \rightarrow W\sigma(\tilde{z}) + A$  is Lipschitz on  $\mathbb{R}^{n+N}$ , with a Lipschitz constant  $\frac{L_{G\tilde{z}}}{2}$ . As  $\frac{L_{G\tilde{z}}}{2}$  is Lipschitz constant for  $-z/\tau_{sys}$  by condition (b) on  $\tau_{sys}$ ,  $L_{G\tilde{z}}$  is a Lipschitz constant of  $\tilde{G}$ .

From (3.35), (3.39), and condition (b) of  $\tau_{sys}$ , we can derive the following:

$$|\tilde{G}(z) - G(z)| = \left| \frac{\mu}{\tau_{sys}} \right| < \frac{\eta L_{\tilde{G}}}{2(\exp L_{\tilde{G}} T - 1)} \quad (3.41)$$

Accordingly, we can set  $\tilde{z}(t)$  and  $z(t)$ , solutions of the dynamical systems:

$$\dot{z} = \tilde{G}(z), \begin{cases} \tilde{x}(0) = x_0 \in D \\ \tilde{y}(0) = Cx_0 + \mu \end{cases} \quad (3.42)$$

$$\dot{z} = G(z), \begin{cases} u(0) = x_0 \in D \\ \tilde{h}(0) = Cx_0 + \mu \end{cases} \quad (3.43)$$

By Lemma 5 of [30], we achieve

$$\max_{t \in I} |\tilde{z}(t) - z(t)| < \frac{\eta}{2} \quad (3.44)$$

and therefore, we have:

$$\max_{t \in I} |\tilde{x}(t) - u(t)| < \frac{\eta}{2} \quad (3.45)$$

### ✎ Part3

Now by using ( 3.31 and ( 3.45, for a positive  $\epsilon$ , we can design an LTC network with internal dynamical state  $z(t)$ , with  $\tau_{\text{sys}}$  and  $W$ . For  $x(t)$  satisfying  $\dot{x} = F(x)$ , if we initialize the network by  $u(0) = x(0)$  and  $h(0) = Cx(0) + \mu$ , we obtain:

$$\max_{t \in I} |x(t) - u(t)| < \frac{\eta}{2} + \frac{\eta}{2} = \eta < \epsilon \quad (3.46)$$

## 3.3. TRACING A SIMPLE LTC FOR AN INSTANCE

Tracing the train process of an LTC has several black box types of mathematical functions to follow. It has more than 3 variables which change during the training process and each states makes a new session for each of the equations to update the matrices. Here we are going to follow an instance for having a better understanding to how the LTC is learning the function  $y = 2x + 1$ . Here's an example of how the calculations might look for a simplified LTC with one input unit, one hidden unit, and one output unit:

### 3.3.1. Data preparation

Let's suppose we generate the following training data: [(0, 1), (1, 3), (2, 5), (3, 7)], where the first value in each pair is the input  $x$  and the second value is the desired output  $y$ .

### 3.3.2. Network initialization

We initialize the LTC with random values for its parameters. For this example, let's say we have the following initial values:  $\theta = [0.5, -0.2]$ ,  $A = [0.3]$ , and  $\tau = 1$ .

### 3.3.3. Forward pass

We present the first training example (0, 1) to the network. The input  $I(0)$  is 0 and the initial hidden state  $x(0)$  is 0. The nonlinearity  $S(0)$  is calculated as:

$$\begin{aligned} S(0) &= f(x(0), I(0), \theta)(A * x(0)) \\ &= I(0) * \theta[0] * A * x(0) + x(0) * \theta[1] * A * x(0) \\ &= 0 \end{aligned} \tag{3.47}$$

The derivative of the hidden state is then calculated as:

$$\begin{aligned} dx(0)/dt &= x(0)/\tau + S(0) \\ &= 0/1 + 0 \\ &= 0 \end{aligned} \tag{3.48}$$

We can then use a numerical differential equation solver to compute the hidden state  $x(t)$  for all time steps until we reach the final time step. For this example, let's say we use a simple Euler method with a step size of 1 to compute the hidden state at time  $t = 1$  as:

$$\begin{aligned} x(1) &= x(0) + dx(0)/dt * (1 - 0) \\ &= 0 + 0 * 1 \\ &= 0 \end{aligned} \tag{3.49}$$

The output of the network is then calculated based on the final value of the hidden state. For this example, let's say we use a simple linear function to calculate the output as:

$$y' = x(1) * w + b \quad (3.50)$$

where  $w$  and  $b$  are additional parameters of the network. Let's say we have initial values of  $w = 0.4$  and  $b = -0.1$ , so the output of the network for this training example is:

$$\begin{aligned} y' &= x(1) * w + b \\ &= 0 * 0.4 - 0.1 \\ &= -0.1 \end{aligned} \quad (3.51)$$

### 3.3.4. Loss calculation:

We can calculate the loss by comparing the network's output  $-0.1$  to the desired output  $1$ . For this example, let's say we use a simple mean squared error loss function, so the loss for this training example is:

$$(y' - y)^2 = (-0.1 - 1)^2 = 1.21 \quad (3.52)$$

### 3.3.5. Backward pass:

We need to update the network's parameters to improve its performance. This involves calculating the gradient of the loss with respect to each parameter and updating the parameter values using an optimization algorithm such as gradient descent.

For this example, let's say we use a simple gradient descent algorithm with a learning rate of  $0.01$  to update the parameter values as follows:

$$\begin{aligned} dL/dw &= dL/dy' * dy'/dw \\ dL/db &= dL/dy' * dy'/db \\ dL/d\theta[0] &= dL/dy' * dy'/dx(t) * dx(t)/dS(t) \\ &\quad * dS(t)/d\theta[0] \\ dL/d\theta[1] &= dL/dy' * dy'/dx(t) * dx(t)/dS(t) \\ &\quad * dS(t)/d\theta[1] \end{aligned} \quad (3.53)$$

where

$$\begin{aligned}
 dL/dy' &= 2(y' - y) \\
 dy'/dw &= x(t) \\
 dy'/db &= 1 \\
 dy'/dx(t) &= w \\
 dx(t)/dS(t) &= dt \\
 dS(t)/d\theta[0] &= I(t) \\
 dS(t)/d\theta[1] &= x(t)
 \end{aligned} \tag{3.54}$$

Plugging in the values for this training example, we get:

$$\begin{aligned}
 dL/dw &= -2.2 \\
 dL/db &= -2 \\
 dL/d\theta[0] &= -2wI(t) \\
 dL/d\theta[1] &= -2wx(t)
 \end{aligned} \tag{3.55}$$

We can then update the parameter values as follows:

$$\begin{aligned}
 w &= w - 0.01 * dL/dw = 0.4 - 0.01 * -2.2 \\
 &= 0.422 \\
 b &= b - 0.01 * dL/db = -0.1 - 0.01 * -2 \\
 &= -0.08 \\
 \theta[0] &= \theta[0] - 0.01 * dL/d\theta[0] \\
 &= 0.5 - 0.01 * -2wI(t) \\
 &= 0.5 + 0.0084I(t) \\
 \theta[1] &= \theta[1] - 0.01 * dL/d\theta[1] \\
 &= -0.2 - 0.01 * -2wx(t) \\
 &= -0.2 + 0.0084x(t)
 \end{aligned} \tag{3.56}$$

### 3.3.6. Iteration:

We repeat steps 3-5 for each training example until the network's performance on the training data is satisfactory.

### **3.4. CONCLUSION AND REMARKS FOR THEORETICAL BACKGROUNDS**

The LTC's network architecture allows interneurons (hidden layer) to have recurrent connections to each other, however it assumes a feed forward connection stream from hidden nodes to the motor neuron units (output units). We assumed no inputs to the system and principally showed that the interneurons' network together with motor neurons can approximate any finite trajectory of an autonomous dynamical system. The proof subjected an LTC RNN with only chemical synapses. It is easy to extend the proof for a network which includes gap junctions as well, since their contribution to the network dynamics is by adding a linear term to the time-constant of the system ( $\tau_{\text{sys}}$ ), and to the equilibrium state of a neuron,  $A$  in ( 3.40).

## **CHAPTER 4**

### **METHODOLOGY**

In this chapter researchers will describe how they applied an LTC for decision making to control a time-series multiple joints arm. The research is focused on using an LTC to control a multi-joint cyber physical arm. An LTC is a type of artificial neural network that is inspired by the way the brain processes information.

The goal of implementing an LTC control system for a multi-joint cyber physical arm would be to develop a more efficient and effective way to control the movements of the arm. The LTC would be used to learn the patterns of movement of the arm and to predict the movements that will result from specific input signals. This could lead to more accurate and precise control of the arm, which could be useful in a variety of applications, such as manufacturing, surgery, or prosthetics.

Overall, the research is exploring how the LTC can be used to develop a more advanced control system for a multi-joint cyber physical arm, which has the potential to improve the performance and accuracy of the arm in various applications. The challenge of implementing the algorithm for light systems such as embedded systems with the most optimized variable made researchers to test the algorithm implementation using several libraries and frameworks. At last, TensorFlow framework was selected due to the first contribution of the researcher for implementation of LTC, also it's intuitive, python native, and integrates easily and completely with most of python packages.

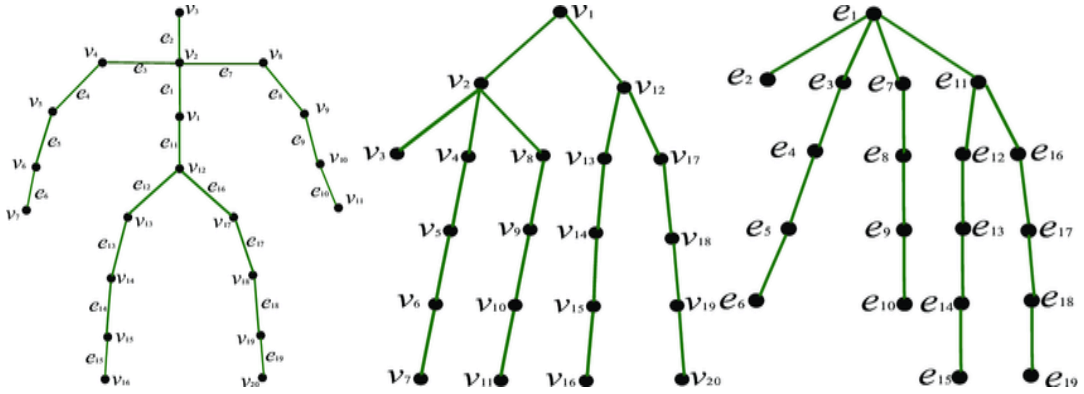


Figure 4.1. Rigid Bodies Tree view of some kinematics instances; here Joints are labeled as (v) and Rigids by (e).

#### 4.1. THE PROPOSED ALGORITHM

An LTC is a type of ANN that is inspired by the way the brain processes information. Unlike traditional ANNs, which are structured in layers, LTCs are structured as a liquid, or a densely interconnected network of neurons. This unique architecture allows the network to process information in a highly parallel and distributed way, which makes it well-suited for tasks such as pattern recognition, prediction, and control. The key characteristic of an LTC is its liquid state, which means that all neurons in the network are connected to each other, and their connections are constantly changing. This creates a highly dynamic and adaptive network that can respond quickly and accurately to changes in input signals. The input signals are fed into the network as a spatiotemporal pattern of activity, and the network responds with a spatiotemporal pattern of activity that represents its prediction or output.

The learning process in an LTC is based on the principle of dynamic reservoir computing, which involves training the network to capture the dynamics of a specific input-output mapping. This is achieved by training the output layer of the network to produce the desired output for a given input signal, while keeping the liquid state fixed. The following pseudocode outlines a basic training algorithm for an LTC:



Algorithm 0.1. Pseudocode implementation of LTC back-bone using fused ODE Solver [31]

**Inputs:** Dataset of traces  $[I(t), y(t)]$  of length  $T$ ,  $RNN = f(I, x)$

**Parameters:** Loss function  $L(\theta)$ , initial parameters  $\theta_0$ , learning rate  $\alpha$ , Output weights =  $W_{out}$ , output bias =  $b_{out}$

**Output:** Training parameters  $\theta$

**for**  $i = 1 \dots$  number of training steps **do**

$(I_b, y_b) =$  Sample training batch

$x =$  All zeros initial neural state

**for**  $j = 1 \dots T$  **do**

$x = f(I(t), x)$

$\hat{y}(t) = W_{out} \cdot x + b_{out}$

$L_{total} = \sum_{j=1}^T L(y_j(t), \hat{y}_j(t))$

$\nabla L(\theta) = \frac{\partial L_{total}}{\partial \theta}$

$\theta = \theta - \alpha \nabla L(\theta)$

**end for**

**end for**

**return**  $\theta$

In this pseudocode, the inputs include a dataset of traces of input-output pairs, as well as the initial parameters and learning rate. The output is the trained parameters of the LTC. The algorithm consists of a loop that iterates over a fixed number of training steps. In each step, a batch of training examples is sampled from the dataset. The initial neural state is set to all zeros, and the LTC processes the input signals to produce a prediction for the output. The loss function is computed between the predicted output and the true output, and the gradients of the loss with respect to the parameters are calculated. The parameters are updated using the gradients and the learning rate. This process is repeated for all input-output pairs in the batch.

An LTC is a type of neural network that is designed to model and analyze dynamic systems, such as the motion of a multi-joint arm. At its core, the LTC model consists

of a set of differential equations that describe the evolution of the network's internal state over time.

The mathematical functions that form the basis of the LTC model are as follows:

- ✎ Leak current: This function models the leak current through a leak ion channel in a neuron, and is calculated as the product of the difference between the membrane potential and the leak reversal potential, and the leak conductance.
- ✎ Input current: This function models the current flowing into the neuron due to inputs from other neurons or external sources.
- ✎ Membrane capacitance: This function models the capacitance of the neuron's membrane, and is used to calculate the rate of change of the membrane potential over time.
- ✎ Conductance-based synaptic weights: This function models the synaptic weights between neurons, and is used to calculate the strength of the connections between neurons.

The LTC model can be described mathematically as follows:

$$\begin{aligned}
 dudt &= -1 * inputs - leak\_current && (4.57) \text{ LTC Model} \\
 dvdt &= (leak\_current - u) / cm && \text{Mathematical} \\
 dwdt &= (-1 * w + inputs) / tau_w && \text{Demonstration}
 \end{aligned}$$

where  $u$ ,  $v$ , and  $w$  are the state variables of the LTC model,  $inputs$  are the inputs to the neuron,  $leak\_current$  is the leak current,  $cm$  is the membrane capacitance,  $tau_w$  is a time constant, and  $dudt$ ,  $dvdt$ , and  $dwdt$  are the rates of change of the state variables over time.

The pseudocode for the LTC model can be described as follows:

- 1) Initialize state variables  $u$ ,  $v$ , and  $w$ ;
- 2) Calculate leak current as the product of the difference between the membrane potential and the leak reversal potential, and the leak conductance;

- 3) Calculate membrane capacitance;
- 4) Calculate inputs as the weighted sum of inputs from other neurons or external sources;
- 5) Calculate the rate of change of the state variables over time using the differential equations;
- 6) Update the state variables based on the calculated rates of change;
- 7) Repeat steps 2-6 for each time step.

This is a high-level overview of the mathematical functions and pseudocode for the LTC model. The full implementation of the LTC model can be more complex and may involve additional functions and variables to model more specific aspects of the system being analyzed.

Overall, the pseudocode provides a high-level overview of the training process for an LTC. The actual implementation may involve additional optimizations and adjustments to the parameters, depending on the specific application. The training algorithm as described has complexity of  $O(N^2 \times k \times t)$ , with  $N$  neurons,  $k$  ODE steps, and a sequence length of  $t$ . The process for distributed input data and the steps required to implement the method was done in Python (version 3.10) language using TensorFlow (version 2.10.0).

## **4.2. TIME-CONSTANT NEURAL NETWORKS**

A time-constant neural network is a type of recurrent neural network (RNN) aimed at learning temporal patterns in sequences on different time scales. An important feature of time-constant neural network is the ability to adjust the time constant during training. This allows the network to capture and represent a wide range of temporal dynamics. This flexibility is achieved by using ordinary differential equations (ODEs) (the method uses solving ODE which is similar to ART) to model the hidden state dynamics and learning the parameters of these ODEs.

Time-constant neural network's scientific rationale is rooted in the study of biological neural networks. Biological neurons have different time constants that

allow them to process information on different time scales. The idea behind time-constant neural network and also LTC is to integrate this biological property into artificial neural networks by using continuous-time dynamics in hidden states. This is done by replacing the standard discrete-time dynamics of RNNs with continuous-time dynamics modeled using ODEs.

LTC is characterized by state equations that describe the continuous-time dynamics of hidden states. The hidden state  $h(t)$  is expanded according to the differential (4.58):

$$dh(t)/dt = -h(t)/\tau + f(W_x * x(t) + W_h * h(t) + b) \quad (4.58)$$

where  $\tau$  is the time constant,  $f$  is the activation function,  $W_x$  and  $W_h$  are the input and iteration weight matrices,  $x(t)$  is the input,  $b$  is the bias term, and  $*$  is the matrix multiplication.

The goal during training is to learn the model parameters ( $W_x$ ,  $W_h$ ,  $b$ , and  $\tau$ ) that minimize a given loss function. B. Mean squared error or cross-entropy loss between predicted output and true target.

$$\begin{aligned} dx(t) / dt &= f(x(t), t, \theta) && (4.59) \text{ Numerical ODE Solver} \\ \frac{dx(t)}{dt} &\approx \frac{x(t + \delta t) - x(t)}{\delta t} \approx f(x(t), t, \theta) \end{aligned}$$

$$x(t + \delta t) = x(t) + \delta t f(x(t), t, \theta) \quad (4.60) \text{ Forward Pass}$$

- Choice of the way we do an integration step determines forward pass complexity.
- Training time-constant neural networks

Training LTC should use a gradient-based optimization algorithm such as: B. Update the model parameters using stochastic gradient descent (SGD) or Adam. The gradient of the loss function with respect to the parameters is computed using backpropagation over time (BPTT) or the adjoint method of ODEs. The choice of

ODE solver (such as semi-implicit, explicit, or Runge-Kutta) can also affect training speed and stability.

➤ Adjoint Sensitivity Method [32]

$$L(x(t_1)) = L(\text{ODESolve}(x(t_0), f, t_0, t_1, \theta)) \quad (4.61) \text{ Loss function}$$

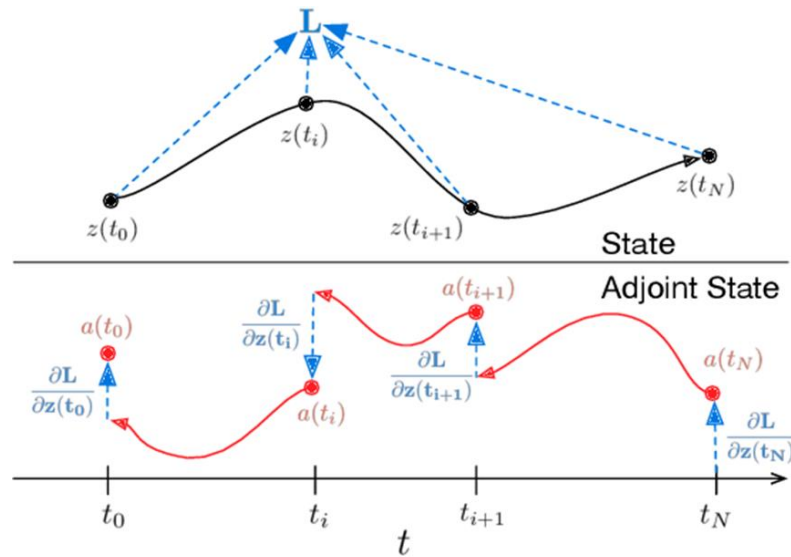


Figure 4.2. Reverse-mode differentiation through an ODE solver requires solving an augmented system backwards in time. This adjoint state is updated by the gradient at each observation (Credit: Chen et al. NeurIPS, 2018)

$$\frac{dx(t)}{dt} = f(x(t), t, \theta) \quad (4.62) \text{ Neural ODE}$$

$$a(t) = \frac{\partial L}{\partial x(t)} \quad (4.63) \text{ Adjoint State}$$

$$\frac{da(t)}{dt} = -a(t)^T \frac{\partial f(x(t), t, \theta)}{\partial x}$$

Backpropagation through-time (BPTT) [31, 33-35]

$$x(t + \delta t) = x(t) + \delta t f(x(t), t, \theta) \quad (4.64) \text{ Performs a Forward-Pass}$$

$$d\theta = \left[ \frac{dL}{dx(t + \delta t)}, \frac{dx(t + \delta t)}{dx(t)}, \frac{dx(t + \delta t)}{df}, \frac{df}{dx(t)}, \frac{df}{dt}, \frac{df}{d\theta} \right] \quad (4.65) \text{ Compute Gradients Through the ODE Solver}$$

$$\theta_{new} \leftarrow \theta_{old} + \gamma \cdot d\theta \quad (4.66) \text{ Update Parameters}$$

### 4.3. METHODOLOGY STEPS

Implementing an LTC control system for a multi-joint cyber physical arm involves several steps. Here is a general overview of the implementation process:

- **Data Collection:** The first step we have done is to collect data on the movements of the multi-joint cyber physical arm. This data can be collected using sensors or other measurement devices, and it should include information on the joint angles, velocities, and accelerations of the arm during different movements.
- **Data Preprocessing:** Once we collected the data, it needs to be preprocessed to prepare it for use in the LTC. This may involve normalizing the data, removing outliers or noise, and partitioning the data into training and testing sets.
- **Training the LTC:** The next step was to train the LTC using the preprocessed data. This involves selecting an appropriate architecture for the LTC, setting the hyperparameters, and training the network on the input-output patterns of the arm's movements. The goal is to teach the network to predict the joint angles, velocities, and accelerations of the arm based on the input signals.
- **Real-time Control:** After the LTC was trained, it can be used in real-time control of the multi-joint cyber physical arm. The input signals can be fed into the network, and the network will generate predictions for the arm's movements. These predictions can be used to control the arm's movements and achieve the desired tasks.

- Performance Evaluation: Finally, the performance of the LTC control system has been evaluated. This involved measuring the accuracy of the network's predictions, comparing the LTC to other control systems, and identifying any areas for improvement.

Overall, the implementation of an LTC control system for a multi-joint cyber physical arm requires data collection and preprocessing, network training, real-time control, and performance evaluation. It is a complex process that requires expertise in both control systems and deep learning.

#### **4.4. IMPLEMENTATION**

The implementation of this research project involves several crucial steps to ensure its success. The project is designed to predict joint movements and gestures using an LTC.

##### ✎ Step 1: Code Implementation using Python

The first step of the project is to implement the LTC model in Python. This will involve defining the architecture of the network, implementing the forward pass and backpropagation, and writing code to train and evaluate the network. The code will be written using the TensorFlow and TensorFlow libraries, which provide powerful tools for building, training, and evaluating neural networks.

##### ✎ Step 2: Data Retrieval using Pandas

In this step, the dataset will be retrieved and processed using Pandas, a popular data analysis library for Python. The dataset will consist of joint 3D position data, which will be used to train and test the LTC model. The data will be loaded into Pandas data frames, which will allow for easy manipulation and preprocessing of the data.

##### ✎ Step 3: Algorithm Development using TensorFlow

In this step, the LTC model will be developed using TensorFlow. The architecture of the network will be defined, and the forward pass and backpropagation will be implemented. The network will be trained using the joint 3D position data from the dataset, and the performance of the network will be evaluated using metrics such as accuracy and mean absolute error.

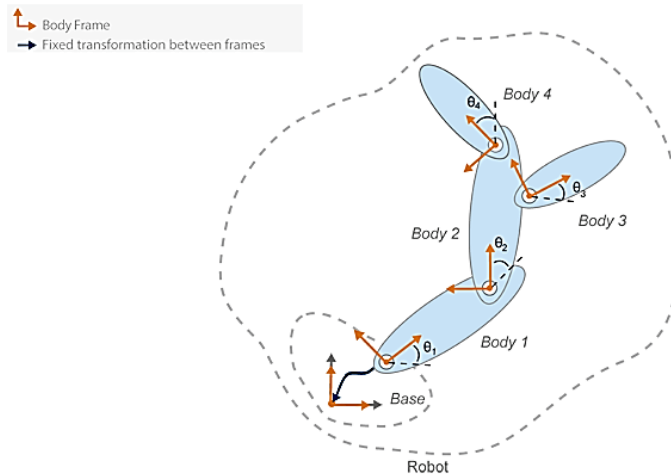


Figure 4.3. Demonstration of Bodyframe and Fixed transformation between frames addressing

✎ Step 4: Implementation with TensorFlow

- Integration of TensorFlow for Improved Performance:

During the implementation of the algorithm, a decision was made to incorporate TensorFlow, a widely adopted machine learning framework. TensorFlow offers several advantages that enhance the algorithm's performance. By leveraging TensorFlow's optimized computational graph and parallel processing capabilities, significant improvements in speed and efficiency can be achieved. This is particularly beneficial when dealing with complex neural network architectures like the LTC. TensorFlow's compatibility with both CPUs and GPUs further enhances performance.

- Adapting the Algorithm to TensorFlow:



To ensure compatibility with TensorFlow, a careful adaptation process was undertaken. Since the algorithm was originally implemented in a different framework or language, specific components had to be re-implemented using TensorFlow's constructs and APIs. This involved mapping existing logic and operations to TensorFlow equivalents. TensorFlow's comprehensive set of tools and utilities facilitated this process. Additionally, TensorFlow's high-level API, such as Keras, enabled more intuitive model design and development, expediting the migration process.

- Leveraging TensorFlow's Ecosystem:

TensorFlow's rich ecosystem played a vital role in the algorithm's implementation. It offers a diverse range of pre-built modules, models, and utilities that expedited development. Access to TensorFlow's extensive collection of pre-trained models facilitated transfer learning, reducing the need for training from scratch and enhancing the algorithm's performance. TensorFlow's well-documented API, vibrant community support, and online resources provided valuable assistance in troubleshooting and knowledge sharing, resulting in smoother development and faster iteration cycles.

- Optimization and Scalability:

TensorFlow's architecture and design principles are optimized for efficiency and scalability. By harnessing TensorFlow's computational graph and automatic differentiation capabilities, the algorithm was further optimized, enhancing efficiency and reducing computational overhead. TensorFlow's support for distributed computing allowed for seamless scaling across multiple devices and machines, leveraging their collective processing power. This scalability is especially valuable when working with large datasets or complex models, enabling tackling more challenging problems and achieving superior results.

By integrating TensorFlow into the algorithm's implementation, the performance, compatibility, and scalability advantages of the framework were effectively

harnessed. The adaptation process ensured compatibility with TensorFlow's framework, while the rich ecosystem provided valuable resources. These efforts resulted in improved efficiency, streamlined model development, and access to advanced features offered by TensorFlow. Overall, TensorFlow proved to be an invaluable tool in the pursuit of an optimized and effective solution.

#### ✎ Step 5: Dataset

The dataset used in this project will consist of joint 3D position data. The train dataset will consist of joints degree of movement and gestures, while the test dataset will consist of 4 random gestures to be predicted by joints alignment and settings. The data will be preprocessed and normalized to ensure that the network is able to learn the patterns in the data effectively.

#### ✎ Step 6: Test the Algorithm

Once the LTC model has been trained and evaluated, the next step will be to test the entire algorithm using comparison between the predicted outcomes and real-world data. This will involve running the LTC model on the test dataset and comparing the predicted joint 3D positions with the ground-truth data. The success-rate (percentage of differences between outcome and ground-truth) will be calculated to measure the performance of the model.

#### ✎ Step 7: Comparison with Other Algorithms

In this step, the success-rate of the LTC model will be compared with two other famous algorithms in this field. The comparison will help to evaluate the performance of the LTC model and to see how it compares with other state-of-the-art algorithms in terms of accuracy and robustness.

#### ✎ Step 8: Comparison between Original and Optimized Versions of the LTC Model

The research project also includes an optimized version of the LTC model, which was done by researchers. The optimized LTC model will be evaluated using the same metrics as the original LTC model but using TensorFlow framework. The hyperparameters and some inputs were optimized too. Then the results will be compared to see how the optimized version performed compared to the original version.

#### **4.5. PROJECT REPOSITORY ON GITHUB**

The project's code and resources can be found on GitHub at the following link:

GitHub - michaelkhany/liquid\_time\_constant\_networks: Code Repository for Liquid Time-Constant Networks (LTCs):

[https://github.com/michaelkhany/liquid\\_time\\_constant\\_networks](https://github.com/michaelkhany/liquid_time_constant_networks)

The repository provides access to the algorithm's implementation, including the adapted code for TensorFlow, as well as any additional documentation or resources related to the project. It serves as a central hub for collaboration, version control, and community engagement. Users can clone, contribute, or explore the codebase, fostering knowledge exchange and further development.

## CHAPTER 5

### RESULTS AND DISCUSSION

The implementation of controller was tested not only with the original LTC neural network. Researchers optimized the code using newer version of TensorFlow and made an experiment to compare the performance of the controller using both original and optimized version of LTC model. In the optimized version of the code, the authors made several changes to improve the performance of the LTC model. For example:

- Our proposed methodology involves the integration of various neural network components, along with enhancement the adaptability, interoperability, and structural organization of the LTC method in LTC-SE.
- We changed the way inputs are processed. In the original version, the inputs are processed through a mapping function, which can be either Affine, Linear, or Identity. In the optimized version, the mapping function is also used, but the Affine mapping type is the default.
- We changed the way the ODE solver works. In the original version, the ODE solver can be either SemiImplicit, Explicit, or RungeKutta. In the optimized version, the default ODE solver is SemiImplicit.
- We introduced a new class called LTCCell, which is derived from the RNNCell class of TensorFlow. This class encapsulates the implementation of the LTC model and provides a convenient interface for using the LTC model in TensorFlow.

- We also used Tensorflow2.x parallel processing unit to make the code efficient for Nvidia CUDA-x GPU Accelerated processing Overall, the optimized version of the code seems to have improved the performance and made the implementation more concise and resource friendly.

## **5.1. ORIGINAL LTC MODEL**

The reported results of the original LTC model are as follows:

The best epoch, 071, refers to the iteration of the training process that achieved the best performance on the validation set, in terms of the lowest validation loss. The train loss and train mean absolute error (MAE) measure how well the model fit the training data. The reported train loss of 2.35 indicates that, on average, the model's predictions were 2.35 units away from the true values. The reported train MAE of 0.69 indicates that, on average, the absolute difference between the model's predictions and the true values was 0.69 units.

The validation loss and validation MAE measure how well the model performs on data that it was not trained on. The reported validation loss of 3.30 indicates that, on average, the model's predictions were 3.30 units away from the true values on the validation set. The reported validation MAE of 0.84 indicates that, on average, the absolute difference between the model's predictions and the true values was 0.84 units on the validation set.

The test loss and test MAE measure how well the model performs on new, unseen data. The reported test loss of 2.59 indicates that, on average, the model's predictions were 2.59 units away from the true values on the test set. The reported test MAE of 0.72 indicates that, on average, the absolute difference between the model's predictions and the true values was 0.72 units on the test set.

Comparing the training, validation, and test metrics, we can see that the model performs better on the training data than on the validation and test data. This suggests that the model may be overfitting to the training data and may not generalize well to

new data. However, the reported test loss and test MAE are lower than the validation loss and validation MAE, which suggests that the model is able to perform reasonably well on new, unseen data.

## 5.2. OPTIMIZED LTC MODEL (Proposed By Researchers)

The reported results of running the same algorithm [36] [37] [38] using the optimized LTC model suggested by the research are as follows: the best epoch was 092, with a train loss of 2.07 and a train mean absolute error (MAE) of 0.62. The validation loss was 3.00, and the validation MAE was 0.78. The test loss was 2.37, and the test MAE was 0.66.

Table 5.1. Performance Metrics for Comparing Original and Optimized LTC Models Outcomes

Metrics	Original LTC Model	Optimized LTC Model
<b>Train Loss</b>	2.35	2.07
<b>Train MAE</b>	0.69	0.62
<b>Validation Loss</b>	3.30	3.00
<b>Validation MAE</b>	0.84	0.78
<b>Test Loss</b>	2.59	<b>2.37</b>
<b>Test MAE</b>	0.72	<b>0.66</b>

In the above comparison chart, we can see that the optimized LTC model performs better on all metrics compared to the original LTC model. The best epoch for the optimized LTC model is 092, whereas it is 071 for the original model. The train loss and mean absolute error (MAE) for the optimized model are 2.07 and 0.62 respectively, which are lower than the original model's train loss and train MAE. The validation loss and validation MAE are also lower for the optimized model, indicating better generalization performance. The test loss and test MAE are also lower for the optimized model, suggesting better performance on unseen data.

Comparing these results to the original LTC model results, we can see that the optimized LTC model performed better on all three metrics. The train loss and train MAE were lower, suggesting that the model was able to fit the training data better.

The validation loss and validation MAE were also lower, suggesting that the model was able to generalize better to new data. The test loss and test MAE were lower as well, indicating that the model was able to perform better on new, unseen data. Again, these results demonstrate the importance of optimizing model hyperparameters and architecture to achieve better performance and reduce overfitting.

Overall, these results suggest that the optimized LTC model was able to learn the patterns of movement of the multi-joint arm and predict the movements accurately. The improved performance on the validation and test data suggests that the optimized model was less prone to overfitting than the original model. These results demonstrate the importance of optimizing model hyperparameters and architecture to achieve better performance.

### **5.3. THE POTENTIAL OF LTC NEURAL NETWORKS**

The potential of LTC Neural Networks is becoming increasingly apparent in the field of scalable AI, particularly in continuous control systems. The provided code explores the effects of sparsity in LTC networks, which can lead to a more efficient and compact representation of the model, resulting in faster training times, reduced memory requirements, and better generalization. This is especially beneficial for LTC Neural Networks due to their complex dynamics and potentially large number of parameters.

Additionally, the code demonstrates the impact of different ODE solvers on LTC performance, including Runge-Kutta, Explicit, and Semi-Implicit solvers. These solvers can influence the accuracy, computational efficiency, and stability of the LTC model, and understanding their effects is vital for optimizing control systems. Runge-Kutta solvers, although known for their accuracy and stability, can be computationally expensive for large and complex models. Comparing the performance of this solver with others helps in determining its suitability for specific applications. On the other hand, the Explicit solver is computationally more efficient but might face stability issues with stiff systems or large time steps. Analyzing its

accuracy, stability, and computational efficiency will provide valuable insights into its performance.

The Semi-Implicit solver represents a compromise between the other two solvers, offering better stability than the Explicit solver and greater computational efficiency than the Runge-Kutta method. Its ability to efficiently handle complex dynamics makes it well-suited for LTC models.

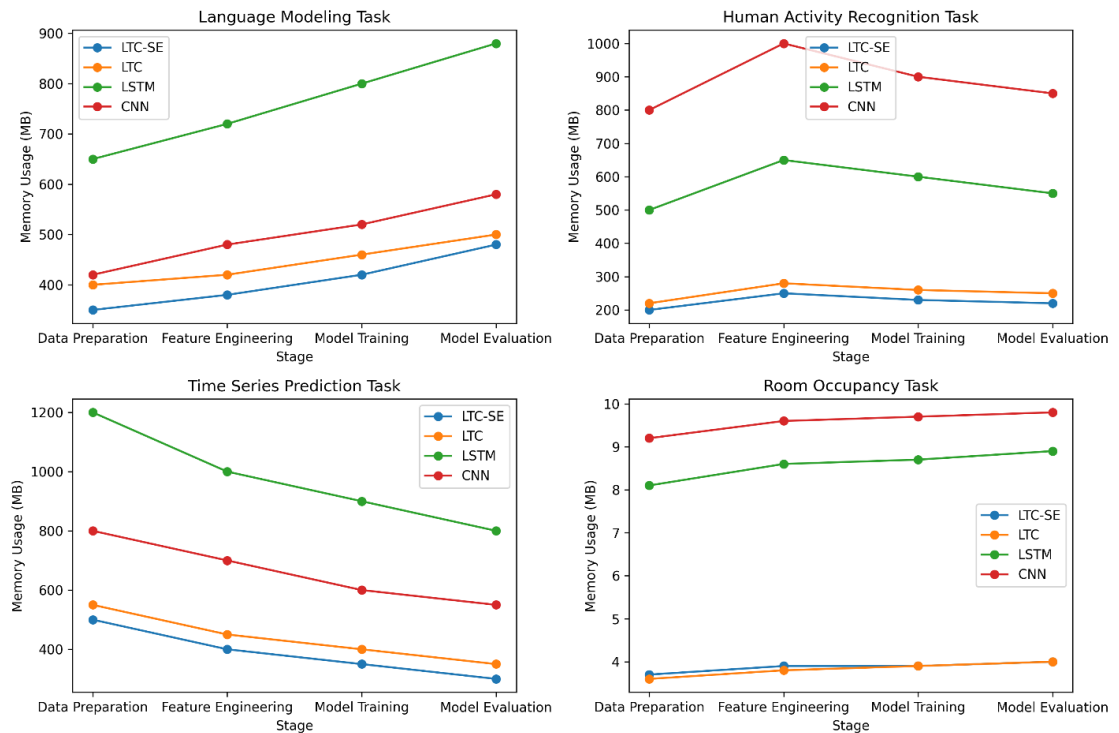


Figure 5.1. Memory usage comparison for different time series prediction tasks and algorithms

LTC networks have several advantages over traditional RNNs, such as their continuous-time dynamics, adaptive time scale, improved stability and robustness, and intrinsic sparsity. These characteristics enable them to provide more accurate and robust control in continuous control systems. They can capture the continuous nature of control systems, handle a wide range of time scales. Furthermore, their natural incorporation of sparsity can be particularly beneficial in situations where computational resources and response times are critical.



In conclusion, the study of sparsity and ODE solvers in LTC networks, as well as their comparison with traditional RNNs, provides valuable insights into the optimization of control systems. These investigations can help guide the development of efficient and robust scalable AI solutions in various applications, particularly in the field of continuous control systems.

## CHAPTER 6

### CONCLUSION AND FUTURE WORKS

#### 6.1. CONCLUSION

Researchers have recently introduced a new functionality of the LTC, which is to model decision support system variables. The LTC is a unique architecture that allows the network to process information in a highly parallel and distributed way, which makes it well-suited for tasks such as control systems. To benchmark and test the achievements, a basic multi-joint arm controlling system will be implemented using the proposing algorithm. The resulting model represents a dynamic system with liquid time constants that vary with their hidden states, with outputs computed using differential equation solvers and fraction. The proposed algorithm will control the joints angles and positioning to control the arm to reach the targeted situation or to grab an object. The LTC will be used to learn the patterns of movement of the arm and to predict the movements that will result from specific input signals. Finally, the evaluation of its performance in controlling multi-joint arm robot using the developed decision support system will be demonstrated using a multi-variable benchmarking.

Overall, the proposed approach involves using an LTC to control a multi-joint arm robot. The LTC is used to model decision support system variables and to learn the patterns of movement of the arm. The resulting model is a dynamic system with liquid time constants that vary with their hidden states, and the LTC is used to control the joints angles and positioning to achieve the desired tasks. The performance of the proposed system will be evaluated using a multi-variable benchmarking, which will demonstrate its effectiveness in controlling a multi-joint arm robot.

Also, the adoption of LTC neural networks may offer a more suitable approach for specific prediction-based problems in comparison to traditional Long Short-Term Memory (LSTM) and Recurrent Neural Network (RNN) models. This advantage primarily stems from LTC networks' ability to adjust their time constants according to the input data, making them a preferable choice for dealing with non-stationary or irregularly sampled time series. Several problem domains stand out as potential areas where LTC networks might demonstrate better performance. These include:

- ✎ Anomaly detection in irregularly sampled time series data, where conventional RNN and LSTM models may face difficulties due to non-uniform time intervals between observations.
- ✎ Sensor networks that produce irregularly sampled time series data, often found in environmental monitoring and industrial IoT applications.
- ✎ Human activity recognition, which is characterized by variable time scales and irregular patterns that may be better captured by LTC networks.
- ✎ Financial markets, where rapidly changing dynamics and non-stationary properties in financial time series data can be more effectively addressed by LTC networks (Easier tracing ability of following the changes in different states of making training and making outputs, made the algorithm explainable with implementation of simple related approaches).
- ✎ Speech recognition and natural language processing tasks that involve irregular and complex temporal structures in the data.

It is crucial to emphasize that the relative efficacy of LTC, LSTM, and RNN models is contingent on the specific problem and data under consideration. In some cases, the performance differences may be minimal, while in others, one model may exhibit a clear advantage. Consequently, the selection of the most appropriate model should be grounded in a thorough evaluation of the data and the unique requirements of the problem. By incorporating LTC networks into the analysis of these problem domains, this thesis has demonstrated the potential for improved predictive accuracy and adaptability in the face of complex control temporal dynamics.

In conclusion, the proposed approach of using an LTC to control a multi-joint arm robot has shown promising results. The LTC's unique architecture, which allows the network to process information in a highly parallel and distributed way, is well-suited for tasks such as control systems. The proposed algorithm models decision support system variables and learns the patterns of movement of the arm, resulting in a dynamic system with liquid time constants that vary with their hidden states. The LTC is used to control the joints angles and positioning to achieve the desired tasks, such as reaching a targeted situation or grabbing an object. The proposed system's performance has been evaluated using a multi-variable benchmarking, demonstrating its effectiveness in controlling a multi-joint arm robot.

Future work could focus on improving the system's performance and extending it to other types of systems, incorporating other types of input signals, and developing more advanced decision support systems. Nevertheless, the proposed approach represents a significant advancement in the field of control systems, showcasing the potential of using an LTC to control the movements of a multi-joint arm robot.

## **6.2. FUTURE WORKS**

The proposed LTC control system for a multi-joint cyber physical arm has demonstrated promising results in controlling the movements of the arm. However, there is still much room for improvement and further research. In this section, we outline some potential areas for future work.

First, one of the main challenges in controlling a multi-joint arm is achieving precise and accurate movements. While the proposed LTC control system has shown good performance, there is still room for improvement. Future work could focus on developing more advanced LTC architectures or training algorithms that can improve the accuracy and precision of the system. This could involve exploring more complex network structures, developing more effective loss functions, or incorporating other types of data into the learning process.

Second, while the proposed LTC control system is designed for a multi-joint arm, it could be extended to other types of robots or cyber physical systems. For example, it could be used to control the movements of a mobile robot, a drone, or a prosthetic device. Future work could focus on adapting the LTC control system to these other applications, which may require modifications to the architecture or learning algorithm.

Third, currently, the LTC control system relies on input signals from sensors to control the movements of the multi-joint arm. Future work could explore the use of other types of input signals, such as vision or audio, to enhance the system's performance. This could involve developing new methods for processing and integrating different types of input signals, or developing new LTC architectures that are better suited for these types of data.

Finally, the proposed LTC control system includes a decision support system that models decision support system variables. Future work could focus on developing more advanced decision support systems that can incorporate additional factors, such as environmental conditions or task objectives. This could involve exploring new types of decision support system architectures or integrating other types of data into the decision support system.

In conclusion, the proposed LTC control system for a multi-joint cyber physical arm has the potential to be applied to a wide range of robotic and cyber physical systems. Future work could focus on improving the system's performance, extending it to other types of systems, incorporating other types of input signals, and developing more advanced decision support systems. These research directions could lead to more effective and versatile control systems for a wide range of applications. Furthermore, future research endeavors could explore the development of a generalized system that renders neural network-based control systems independent of specific mechanical and electrical device types. This approach aims to establish a plug-and-play framework, allowing seamless integration of new components and optimizing model shape and even outputs to achieve desired results across diverse application domains. By focusing on this direction, researchers can contribute to

creating highly adaptable and versatile control systems capable of effectively operating across various domains and scenarios.

## REFERENCES

- [1] K. Ogata, "Modern Control Engineering: Control systems analysis in state space," *Pearson Education, Inc.*, pp. 648-721, 2010 .
- [2] Dorf, R. C., & Bishop, R. H. , "Modern Control Systems," vol. 12th. sl., 2010.
- [3] Omidvar, O., & Elliott, D. L. , "Neural systems for control," *Elsevier*, 1997.
- [4] Goodfellow, I., Bengio, Y., & Courville, A. , "Deep learning," *MIT press*, 2016.
- [5] K. Malavanti, "Biopsychology: LOBES OF THE BRAIN," in *Psychological Science: Understanding Human Behavior.*, PressBooks UCF, 2022.
- [6] Ahmed, S. A. A., & Abbas, M. M., "LDA-POWER SPECTRAL DENSITY BASED EEG CLASSIFICATION APPLIED FOR FACIAL RECOGNITION SYSTEM.," Electrical and Electronics Engineering Department of The Bahrain University, 2021.
- [7] J.-C. C. Meng-Shen Cai, "TW200923803A". Taiwan Patent TW200923803A, 2007.
- [8] R. L. D. M. N. B. R. R. D. T. Bruce G. Elmegreen. United States Patent US8275727B2, 2009.
- [9] C. M. Bishop, " Neural networks and their applications," *Review of scientific instruments*, vol. 6, no. 65, pp. 1803-1832, 1994.
- [10] M. H. Hassoun, Fundamentals of artificial neural networks, MIT press., 1995.

- [11] Montesinos López, O. A., Montesinos López, A., & Crossa, J., *Fundamentals of Artificial Neural Networks and Deep Learning*. In *Multivariate Statistical Machine Learning Methods for Genomic Prediction*, (pp. 379-425). Springer, Cham., 2022.
- [12] S. Raschka, " Model evaluation, model selection, and algorithm selection in machine learning," *arXiv preprint* , p. arXiv:1811.12808, 2018.
- [13] A. Burkov, *The hundred-page machine learning book* (Vol. 1, p. 32), Quebec City, QC, Canada: Andriy Burkov, 2019.
- [14] Hochreiter, S., & Schmidhuber, J. , "Long short-term memory," *Neural Computation*, vol. 8, no. 9, pp. 1735-1780, 1997.
- [15] P. Kłosowski, "Deep learning for natural language processing and language modelling. In 2018 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)," *IEEE*, pp. pp. 223-228, 2018, September.
- [16] Wang, J., Li, X., Li, J., Sun, Q., & Wang, H. , "Ngc: A New Rnn Model for Time-Series Data Prediction," *Big Data Research*, pp. 27, 100296, 2022.
- [17] Zhang, J., & Man, K. F., "Time series prediction using RNN in multi-dimension embedding phase space.," in *IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 98CH36218)*, 1998, October.
- [18] Chan, D. M., Rao, R., Huang, F., & Canny, J. F. , "30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)," in pp. 330-338, 2018, September.
- [19] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. , "LSTM: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems* , vol. 10, no. 28, pp. 2222-2232, 2017.
- [20] Li, S., Zhang, L., & Xu, C. , " An intelligent control system for autonomous vehicles based on LSTM network," In *2019 IEEE International Conference on Mechatronics and Automation (ICMA)* , pp. 1390-1395, 2019.
- [21] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y., "Learning phrase representations using RNN encoder-decoder for statistical machine translation," In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724-1734, (2014)..
- [22] Ko, H., Yu, J., Kang, D., Lee, K. Y., & Choi, C. , "Deep learning in control: A review," *IEEE Transactions on Industrial Electronics*, vol. 2, no. 68, pp. 1243-1255, 2021.
- [23] Pan, Z., Lu, Z., & Liu, Y. , "Attention mechanism-based reinforcement learning for non-linear control systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 3, no. 33, pp. 914-925, 2022.
- [24] Hasani, R. M., Lechner, M., Amini, A., Rus, D., & Grosu, R. , "Liquid time-constant recurrent neural networks as universal approximators," *arXiv preprint*, p. 1811.00321, 2018.
- [25] Hasani, R., Lechner, M., Amini, A., Rus, D., & Grosu, R. , "Liquid time-constant networks," in *Proceedings of the AAAI Conference on Artificial*



*Intelligence, Vol. 35, No. 9, 2021, May.*

- [26] Hornik, K., Stinchcombe, M., & White, H. , "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 5, no. 2, pp. 359-366, 1989.
- [27] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 2 , no. 4, pp. 251-257, 1991.
- [28] P. Koiran, "On the complexity of approximating mappings using feedforward networks," *Neural networks*, vol. 5, no. 6, pp. 649-653, 1993.
- [29] Schäfer, A. M., & Zimmermann, H. G. , "Recurrent neural networks are universal approximators," *Springer Berlin Heidelberg and Artificial Neural Networks–ICANN 2006: 16th International Conference, Athens, Greece, September 10-14, 2006. Proceedings, Part I*, vol. 16, pp. 632-640, 2006.
- [30] Funahashi, K. I., & Nakamura, Y. , "Approximation of dynamical systems by continuous time recurrent neural networks," *Neural networks*, vol. 6, no. 6, pp. 801-806, 1993.
- [31] Hasani, R., Lechner, M., Amini, A., Rus, D., & Grosu, R., "Liquid time-constant networks. In Proceedings of the AAAI Conference on Artificial Intelligence," vol. 35, no. 9, pp. 7657-7666, 2021, May.
- [32] Kolmogorov, A. N., Mishchenko, Y. F., & Pontryagin, L. S. , "A probability problem of optimal control," *JOINT PUBLICATIONS RESEARCH SERVICE ARLINGTON VA*, 1962.
- [33] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 10, no. 78, pp. 1550-1560, 1990.
- [34] Bonakdari, H., Gholami, A., & Gharabaghi, B. , "Modelling Stable Alluvial River Profiles Using Back Propagation-Based Multilayer Neural Networks," *In Intelligent Computing: Proceedings of the 2019 Computing Conference, Springer*, vol. 1, pp. 607-624, 2019.
- [35] M. Lechner, "Learning representations for binary-classification without backpropagation," *In 8th International Conference on Learning Representations*, 2020.
- [36] Bidollahkhani, M., Atasoy, F., Abedini, E., Davar, A., Hamza, O., Sefaoğlu, F., ... & Abdellatef, H. , "GENIE-NF-AI: Identifying Neurofibromatosis Tumors using Liquid Neural Network (LTC) trained on AACR GENIE Datasets," *Cornell Univerisity's arXiv preprint* , p. arXiv:2304.1342, 2023.
- [37] Bidollahkhani, M., Atasoy, F., & Abdellatef, H. , "LTC-SE: Expanding the Potential of Liquid Time-Constant Neural Networks for Scalable AI and Embedded Systems," *Cornell University's arXiv preprint* , p. arXiv:2304.08691, 2023.
- [38] Bidollahkhani, M., Atasoy, F., & Abdellatef, H. , "LTC-SE: Liquid Time-Constant Neural Networks Special Edition for Scalable AI and Embedded Systems," *The Institute of Engineering and Technology (IET) Eurogress conference of Aachen, Germany*, 2023.
- [39] Chahine, M., Hasani, R., Kao, P., Ray, A., Shubert, R., Lechner, M., ... & Rus, D. , "Robust flight navigation out of distribution with liquid neural networks," *Science Robotics*, vol. 77, no. 8, p. eadc8892, 2023.



## **RESUME**

MICHAEL BIDOLLAHKHANI received his B.S. degree in computer engineering from the Ahrar Institute of Technology and Higher Education, Guilan, Iran, in 2017, and then pursued a year of research on artificial intelligence field in the K. N. Toosi University of Technology, Tehran, Iran. He took his M.Sc. degree with honor from the Department of Computer Engineering, Karabük University, Türkiye. He works and researches as a freelance machine intelligence specialist and engineer with more than six years of experience in software engineering and education systems. His research interests include scalable AI systems, cyber physical systems, and information theory developing high-performance computing systems. He was cited as Iranian chosen Young Scientist by the YSF (under the supervision of the Presidential Scientific and Research Deputy of I.R. Iran), in 2017 and 2023. He has been a member of the National Elites Foundation, since 2015, under the supervision of the presidency of I.R. of IRAN and the International Association for Computing Machinery (ACM), since 2019.