Communicating Neural Network Architectures for Resource Constrained Systems



Prince Abudu

Balliol College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary 2022

This thesis is dedicated to my parents
Anthony Aubi Abudu (late) and Ndomupeyi Madzana
for raising me to be the person I am today, and
my aunt Ellen Katsidzira (late), who inspired me to pursue science.

Acknowledgements

I am deeply grateful for the four years I have spent reading for a Doctor of Philosophy (DPhil) in Computer Science within the Cyberphysical systems group at the University of Oxford. These four years have been the most rewarding of my academic journey and personal growth.

I would like to sincerely thank my supervisor Andrew Markham for taking me as his student and consistently providing guidance and support throughout my research. Andrew has made my learning experience pleasant. In doing so, he has helped me appreciate the essence of academic research in the Computer Science. He is a great person and I am truly fortunate to be his student.

I would also like to acknowledge my lab mates Risqui Utama, Bo Yang and Changhao Chen who started their DPhil studies at the same time as me. We shared some defining experiences, and motivated each other from time to time over multiple dinners.

Moreover, I would like to thank the Department of Computer Science and Balliol College for providing conference funding for me over the years. I am fortunate to have attended some of the best Computer Science conferences in the world as a result of this support.

This thesis would not have been possible without The Rhodes Trust which provided full funding to pursue my studies at the University of Oxford. I am inspired now, more than ever to continue being an exemplary Rhodes Scholar who will 'stand up for the world' using the knowledge and skills I have gained throughout my academic journey.

Reading for a DPhil in Oxford comes with many challenges. I am thankful to many friends I have met over the years, who have made this journey fun and worth it: Ofentse, Edward, Ngoni, AJ, Noon, Stanley, Nelisa, Krishna, Mary and Tony.

I also want to thank my late aunt, Ellen Katsidzira for actively mentoring me and encouraging me to pursue academics to the level of a DPhil.

Most importantly, I am eternally indebted to my mother, Ndomupeyi Madzana for being the best mother in the world, and a star! Without her prayers and love I would not have got this far in my studies. I hope that this achievement can bring more joy to her.

Abstract

The deployment of millions of embedded sensors plagued by resource constraints in sophisticated, complex and dynamic Internet of Things (IoT) environments continues to inspire the need to build novel architectures and models for automated, efficient inference and communication. In such settings, practical challenges related to energy efficiency, computational power and reliability, tedious design implementation, effective communication, optimal sampling and accurate event classification, prediction and detection exist. Sensors operating in such environments must be capable of overcoming such challenges and enable scalable monitoring of dynamic phenomena whilst respecting limitations computational capabilities and battery energy. We continue to see a shift from reliance on cloud-centric to edge-centric architectures for data processing, inference and actuation. Distributed edge inference techniques address real-time, connectivity, network bandwidth and latency challenges in spatially distributed IoT applications. Achieving efficient, resource-aware communication in such systems is a longstanding challenge. Many current approaches require complex, hand-engineered communication protocols.

The development of Machine Learning (ML) continues to motivate a new wave of innovative solutions that intermarry embedded sensors, IoT, and ML to enable various applications. In this thesis, we explore the idea of maximizing the shared utility of multiple nodes (agents) seeking to achieve a sensing objective using a data-driven approach. Among other considerations, it is imperative that agents not only learn to communicate vital information, but also learn to know when to communicate.

Firstly, we describe an in-network processing computing paradigm in which modern distributed Deep Networks are employed for inference on sensed objectives. This computing approach unifies deep learning and decentralized computation.

Secondly, we propose a distributed communicating architecture based on Recurrent Neural Networks (RNNs) that can be instantiated on resource-constrained devices observing unique data and performing automated distributed inference via hidden-state communication. We prove that our framework is able to use a data-driven approach to collectively solve various distributed objectives using sensor-like data, as evidenced by a series of systematic analyses we present.

Thirdly, we present a novel scalable, data-driven and communication-efficient Convolutional Recurrent Neural Network *(C-RNN)* framework for distributed tasks. Our framework can extract relevant features on visual data and automatically learns communication protocols to solve tasks in distributed visual settings.

Finally, we demonstrate that our framework can be extended to visual tracking scenarios. In doing this, we introduce novel and scalable methods to setup, train, and deploy communication-efficient distributed nodes using Deep Networks.

Our empirical and systematic analyses of model convergence, node scalability, energy-efficiency, computational complexity and communication-cost on varying node topologies demonstrate that our proposed frameworks are able to employ Deep Networks to discover and optimize communication protocols on data with different modalities and can be extended to many distributed applications in resource-constrained systems.

Contents

1 Introduction **1**

1.1 Motivation 1
1.2 Challenges 4
1.3 Key Research Questions 7
1.4 Contributions 8
1.5 Publications 9
1.6 Thesis Structure 10

2 Background **11**

2.1 Resource Constrained Systems 12
2.1.1 Wireless Sensor Networks (WSNs) and IoT 13
2.1.2 Cloud IoT 14
2.1.3 Fog Computing 15
2.1.4 Mist Computing 15
2.1.5 End Devices 16
2.2 Deep Networks 16
2.2.1 Overview 18
2.2.2 Deep Network Architectures 20
2.2.2.1 Convolutional Neural Networks (CNNs) 20
2.2.2.2 Recurrent Neural Networks (RNNs) and Long Short-Term Memory Networks (LSTMs) 21
2.2.2.3 Hybrid CNN-RNN Architectures 21
2.2.3 Training Deep Networks 22
2.3 Discussion 23

3 Preliminaries and Related Work **25**

3.1 Deep Network-based In-network Processing (Machine Learning at the Network Edge) 25

3.1.1 Enablers . . . 27
3.1.2 Cooperative Deep Networks . . . 30
3.1.2.1 Distributed Event Detection . . . 30
3.1.2.2 Distributed Image Classification . . . 30
3.1.2.3 Distributed Target Tracking . . . 31
3.1.3 Communication in Deep Networks . . . 32
3.1.3.1 Distributed Machine Learning . . . 32
3.1.3.2 Multi-agent Communication . . . 32
3.2 Discussion . . . 32

4 Communicating Recurrent Neural Network (*C-RNN*) 34
4.1 System Overview . . . 34
4.2 System Model . . . 36
4.2.1 System UML Diagram . . . 38
4.3 Core Network Structure . . . 39
4.3.1 Node Topology . . . 39
4.3.2 RNNs and Weight Sharing . . . 40
4.4 Model Training using BackPropagation Through Time and Space (BPTTS) 42
4.5 Node Communication . . . 42
4.5.1 Learning Communication . . . 42
4.5.2 Communication Paradigm . . . 44
4.6 Discussion . . . 46

5 Learning Distributed Communication in Naive Settings 48
5.1 Introduction . . . 48
5.2 Problem Setting . . . 50
5.3 Our Approach: *C-RNN* . . . 50
5.3.1 System Architecture . . . 51
5.3.2 Training and Inference . . . 52
5.4 Evaluation . . . 53
5.4.1 Experimental Setup . . . 53
5.4.1.1 Sensed objectives . . . 54
5.4.1.2 Time . . . 55
5.4.1.3 Space . . . 55
5.4.2 Results . . . 55
5.4.2.1 2-node System Case . . . 55
5.4.2.2 3-node System Case . . . 58

5.5 Discussion . . . 59

6 Learning Distributed Communication in Visual Settings 61
6.1 Introduction . . . 61
6.2 Problem Setting . . . 64
6.3 Our Approach: $\boldsymbol{C^2RNN}$ . . . 64
6.3.1 System Architecture . . . 64
6.3.2 Feature Extraction . . . 65
6.3.3 Node Communication . . . 66
6.3.4 Training and Inference . . . 67
6.4 Evaluation . . . 68
6.4.1 Experimental Setup . . . 68
6.4.2 Results . . . 70
6.4.2.1 Distributed *Max* and *Max Parity* Tasks . . . 70
6.4.2.2 Model Generalization . . . 72
6.4.2.3 Communication-efficiency . . . 74
6.4.2.4 Computational Complexity . . . 76
6.4.2.5 Memory and Energy Consumption . . . 79
6.4.2.6 Network Robustness . . . 82
6.4.2.7 Comparison with Other Methods . . . 82
6.4.2.8 Architectural Considerations . . . 84
6.5 Discussion . . . 85

7 Learning Distributed Communication in Target Tracking Settings 88
7.1 Introduction . . . 88
7.2 Problem Setting . . . 90
7.3 Approach . . . 91
7.3.1 System Architecture . . . 91
7.3.2 Naive Setting . . . 91
7.3.3 Visual Setting . . . 92
7.4 Training and Inference . . . 93
7.4.1 Tracking Grids . . . 93
7.4.2 Model Training . . . 94
7.5 Evaluation . . . 96
7.5.1 Experimental Setup . . . 96
7.5.1.1 Input Features . . . 97
7.5.1.2 Tracking Steps . . . 97

7.5.1.3 Multi-objective Optimization . . . 97
7.5.1.4 Hyper-parameters . . . 98
7.5.2 Results . . . 98
7.5.2.1 Learning Distributed Target Tracking . . . 98
7.5.2.2 Learning Energy-efficient Optimization . . . 100
7.5.2.3 Learning Communication-efficient Optimization . . . 103
7.5.2.4 Node Scalability . . . 106
7.5.2.5 Extending $\boldsymbol{C^2RNN}$ to Naive Settings . . . 108
7.5.3 Computational Complexity . . . 110
7.5.4 Memory and Energy Consumption . . . 113
7.6 Discussion . . . 114

8 Conclusion and Future Work **116**
8.1 Summary of Contributions . . . 119
8.2 Future Directions . . . 121
8.2.1 Techniques For Hardware Implementation . . . 122
8.2.2 Complex People Tracking . . . 122
8.2.3 Distributed Inference On Multimodal Data . . . 123
8.2.4 Further Optimizations . . . 124
8.2.4.1 Quantized Feature Extractors . . . 124
8.2.4.2 Node Topologies . . . 124

Bibliography **125**

List of Figures

1.1 Pervasive computing model 1
1.2 A classification of energy-efficient mechanisms 4
1.3 Thesis structure outline 10

2.1 Characterization of Resource Constrained Systems 12

3.1 Deep Network-based In-network Processing paradigm 26

4.1 High-level *C-RNN* framework 36
4.2 Single Node Resources 37
4.3 *C-RNN* UML Activity diagram 38
4.4 Nodes in a *C-RNN* architecture 39
4.5 Node topologies 40
4.6 RNN computation over time 41
4.7 BackPropagation Through Time and Space (BPTTS) 43
4.8 Communication handler overview 44

5.1 An illustration of event detection 48
5.2 Nodes in a *C-RNN* architecture 51
5.3 Model training losses for all tasks *(T1,T2 and T3)* 56
5.4 Training time for all tasks *(T1,T2 and T3)* 56
5.5 Scatter plots of targets and model predictions 57
5.6 High level communication paradigm for the 3-node system 58

6.1 High level illustration for distributed MNIST classification 62
6.2 Overview of C^2RNN architecture 65
6.3 Training losses for *max* and *max-parity* tasks 71
6.4 RMSE and error percentages for *max* and *max-parity* tasks 71
6.5 Model training and generalization performance 72
6.6 C^2RNN network topologies 73
6.7 Communication patterns for a 16-node network 74

6.8 Communication-efficiency tradeoffs 75
6.9 Average number of messages communicated with training 76
6.10 Overall computational load with scaling nodes 78

7.1 Architectural overview of distributed target tracking framework . . . 92
7.2 Feature extractor architecture for each node 93
7.3 Convergence trends for the 16-node Visual Tracker model 99
7.4 Convergence trends and performance for optimized and *non*-optimized models 101
7.5 System-wide network energy usage 102
7.6 Tracking grids with varying energy optimization levels 103
7.7 Communication optimization for a 16-node system 104
7.8 Network energy and communication with *high* energy optimization . . 105
7.9 Network energy and communication with *medium* energy optimization 105
7.10 Network energy and communication with *low* energy optimization . . 105
7.11 Communication with training for *low* and *high*-optimized models . . . 106
7.12 System-wide network energy and communication in *non*-optimized and *high*-optimized models 107
7.13 Convergence trends for the 16-node Naive Tracker model. 108
7.14 System-wide network energy usage for 16-node and 25-node systems . 109
7.15 Overall computational overhead for 16-node and 25-node systems . . 112
7.16 Total computational overhead for 16-node and 25-node systems . . . 112

List of Tables

2.1 A summary of Resource Constrained Systems features 13
2.2 Classes of IoT end devices 17

5.1 Convergence trends for objectives (*T1, T2* and *T3*) 55
5.2 Model training results for tasks *(T1,T2 and T3)* 59

6.1 Model hyper-parameters 69
6.2 Summary of C^2RNN model performance results for 2-node, 5-node and 8-node systems 70
6.3 Model generalization with 6 - 40 nodes 72
6.4 C^2RNN performance results based on network topologies 73
6.5 Feature extractor layer by layer computational overhead 77
6.6 *C-RNN* Layer by layer computational overhead 77
6.7 Computational overhead for unified C^2RNN model 78
6.8 Energy and memory requirements 81
6.9 Model performance and average communication with link failure . . . 82
6.10 C^2RNN communication and compute footprints 84
6.11 Model performance with architectural considerations 85

7.1 Model hyper-parameters 98
7.2 Model performance for a 16-node *non*-optimized system 100
7.3 Model evaluation with varying optimization levels 102
7.4 Model evaluation on node scalability 107
7.5 Comparison of Naive and Visual Trackers 109
7.6 C^2RNN feature extractor computational overhead 110
7.7 *C-RNN* computational overhead per layer 111
7.8 Performance evaluation with computational overhead on 16-node and 25-node systems 113
7.9 Energy and memory Requirements 114

Chapter 1

Introduction

1.1 Motivation

People and objects are becoming increasingly interconnected. One contributor to this phenomenon is pervasive computing. Synonymously known as ubiquitous computing, this field of study has spurred interest amongst researchers, scientists and industry. The continual need to embed machines with inconspicuous and ambient intelligence towards the ultimate aim of aiding human experience and improving quality of life continues to motivate advancements in this field.

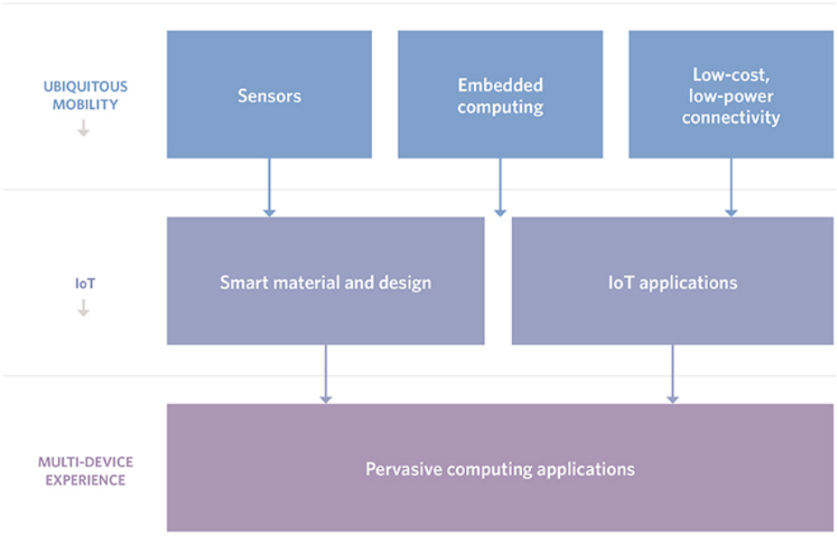


Figure 1.1: Pervasive computing model [180].

Rapid advancements in pervasive computing have enabled significant developments in technologies such as wireless communications and networking [51, 214], embedded and distributed systems [192], software agent systems [73], wearable computers and sensors [177]. These technologies have played crucial roles in Internet

of Things (IoT) infrastructure, thereby enabling many applications which include; smart parking [13], smart industries [119, 163, 232], smart traffic [25], smart homes [65,135,197], smart health [26,37,152], emergency response systems [52], construction site monitoring [103, 157], target tracking [102, 196], wildlife monitoring [63], surveillance and targeting in military systems [84, 181], and environmental monitoring [32]. Figure 1.1 shows a generic pervasive computing model as it relates to the IoT.

To meet their dedicated tasks, IoT systems have to collaboratively and autonomously collect, analyze, process and communicate massive amounts of data using infrastructure capable of adapting and reacting to the sensed phenomenon. In IoT systems, embedded sensing devices are typically resource constrained and need to meet operational requirements, such as sensing accuracy, whilst respecting limitations on battery energy and processing capability. Current IoT architectures are skewed in favor of cloud-centric analytics, with the widespread wireless sensors being 'dumb' and relatively limited in capability. In part, this is a natural consequence of the need for collaborative decision making, e.g., that sensor readings cannot be considered in isolation but must be analyzed as a collective spatio-temporal series. Moreover, the expansive nature of deployment, associated resource constraints on embedded edge devices, and massive data volumes, has further encouraged deployment of centralized Cloud IoT infrastructure. The rudimentary idea of Cloud computing proposed by John McCarthy in the 1960s, was originally coined as a paradigm that would provide computing facilities to the general public as a utility [161]. As an evolving technology, Cloud computing conceptually provides a virtually unlimited amount of computational and storage resources to IoT systems. This enables IoT systems to continuously send data to the Cloud for processing in a centralized paradigm that allows for elasticity, scalability, resource pooling and broad network access [142].

Within Wireless Sensor Networks (WSNs) and Cloud IoT infrastructure, devices entirely rely on the Cloud to provide analytics, insights and actuation for end users consumption. Although this centralized architecture takes advantage of virtually unlimited cloud computing capabilities, issues arise due to latency (data has to be sent up to the cloud, analyzed and then returned before actions can be taken); bandwidth (raw data is typically sent to the cloud, consuming limited network bandwidth in resource constrained, e.g., LORA or 6LowPAN networks [12]; and in battery energy (nodes have to send data to the cloud, regardless of whether it is useful or not). Furthermore, heavy dependency on the Cloud incurs high energy consumption and storage costs.

The pre-eminent need to overcome such challenges has led to newer decentralized edge-centric paradigms [117], such as Fog and Mist computing [33, 211]. In such settings, sensing devices are endowed with increased computing capabilities, thereby enabling inference at the edge of the network where data is collected. Consequently, the systems are equipped with robust real-time capabilities, minimal latency operations, reduced network bandwidth requirements, greater context-awareness and mobility. Nonetheless, significant gaps still exist in light of Fog and Mist computing within the general IoT picture. Instantiating energy-efficient communication techniques, robust and scalable systems remains a long-standing challenge. These limitations further motivate the need to augment distributed intelligence for inference on edge devices. Such capabilities can enable streamlined device to device communication, optimal resource management, fail-safe self-organization and adaptability to dynamic conditions in a data-driven manner with minimal execution effort.

Recent successes in the field of Machine Learning (ML) present a wealth of potential. This is mainly because ML-powered methods and techniques have inherent capabilities to overcome drawbacks that plague traditional methods habitually characterized by hand-designed features [126]. Machine Learning algorithms such as Deep Networks are finding their place in IoT and resource constrained settings. These models have achieved notable successes in domains characterized by massive amounts of data owing to their versatility, inference and predictive capacity [77]. In IoT applications, Deep Networks, particularly Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have achieved state-of-the-art performance on computer vision tasks, (i.e., image classification [39] and action recognition [236]) and time-series prediction tasks, (i.e., event detection [223] and target tracking [145]). These successes motivate the need for current and future resource constrained IoT systems to harness the predictive, inference and versatile power of ML algorithms.

However, to date, the majority of Deep Learning (DL) has focused on treating devices as isolated computational units, rather than as a distributed computational fabric. In this thesis, we investigate how to exploit recent advances in DL to design decentralized communicating architectures in the IoT. In this work, we show how DL algorithms can be employed to perform distributed inference in cooperative and collaborative IoT settings in the absence of Cloud support. We take a particular interest in designing a communication-efficient architecture that can be extended to various application domains.

We aim that our Deep Network-based communicating architecture is scalable, communication-efficient, computationally-efficient, robust, and generalizes well to

various distributed cooperative settings. Our proposed models use data-driven methods and automatically learn features to achieve efficient resource-aware communication. This avoids the problem of having to implement complex, hand-engineered algorithms to perform in-network computation.

1.2 Challenges

To fulfil their dedicated tasks, embedded networked systems must collaboratively and autonomously collect, analyze, process and communicate massive amounts of data using infrastructure capable of adapting to the sensed phenomenon. Those capabilities subject these systems to the following constraints;

- **Energy Consumption**

  Embedded sensors and devices operating in any given environment are expected to have long operating system lifetimes. The robustness of Wireless Sensor Networks (WSNs) rests in their ability to continuously and autonomously sense, detect and identify events, estimate locations, locally control actuators and transmit information, yet being powered by low-cost batteries or energy harvested from the environment.

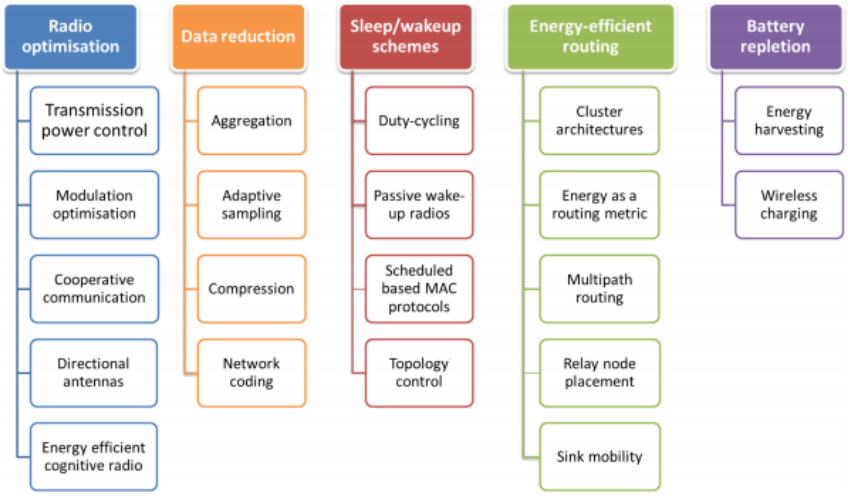


Figure 1.2: A classification of energy-efficient mechanisms [173].

This can only be accomplished by satisfying the low power consumption requirement at sensor nodes [10]. Although this requirement may sound facile, it is not, as it is virtually impossible to constantly replace and recharge thousands of

nodes. Ensuring a sustained source of power is difficult and cost-prohibitive especially in hostile environments. Those limitations have motivated work focused on energy conservation and efficiency in these systems. [15] proposes a comprehensive taxonomy of energy conversation schemes in WSNs. Further works by [9, 171, 201, 227] suggest some technique-specific approaches to energy efficiency through adaptive sleeping, data aggregation, approaches to routing protocols and energy harvesting. [173] provides a classification of energy-efficient mechanisms as shown in Figure 1.2. It can be seen that energy reduction can be achieved across the entire stack, from sources through to algorithmic improvements.

- **Computational Power and Reliability**

  WSNs have become an integral part of human life owing to their successes in delivering intelligence to end-users from environmental data, learning co-location patterns in wildlife monitoring, monitoring patient physiological health data, providing surveillance in security systems, facilitating targeting and command control in military applications augmenting industrial instrumentation in manufacturing [7, 26, 38, 59, 63, 163]. While it may be intriguing to see the effectiveness of WSNs in their application domains, these systems endure substantial constraints on their computing platforms due to power and cost constraints. Typically, these systems have constrained computing speed, memory size and communication bandwidth. In spite of that reality, the same systems have to collect, process and transport massive amounts of data. Equipping the systems with these abilities requires rigorous design and development efforts by experts. A sizeable amount of work [17, 45, 53, 58, 62, 68, 85, 151] has gone into research that focuses on efficient provisioning and management of communications in these systems. In systems where communication protocols are not optimized, performance bottlenecks resulting from the consumption of valuable bandwidth are not uncommon [62]. Such scenarios may lead to the system failing to achieve its intended task. It may also be worth noting that these systems have to constantly be assessed from a standpoint of the utility they attain and the computational cost that goes into achieving a specified objective. Furthermore, most embedded systems have real-time requirements, a circumstance which demands timely, reliable and accurate computation.

- **Heterogeneity**

Heterogeneity signifies diversity. In Wireless Sensor networks, the exploitation of heterogeneity augmented with the design of resource-aware protocols significantly affects system reliability and lifetime [226]. Moreover, careful consideration has to be rendered in the architectural design of nodes as these nodes can run various wireless communication technologies and/or operating systems. These nodes can also be composed of a diverse range of hardware, for instance, sensing tasks may be delegated to cheaper nodes whereas more expensive nodes provide computation capabilities for tasks such as data fusion, compression and transportation. In general, optimal deployment in WSNs is hard [226]. Nonetheless, when heterogeneity is properly considered and resources are carefully placed in a network, performance is improved. Since heterogeneity can be leveraged to optimize WSNs and inform the organization of sensing resources and embedded devices in a system, understanding how resource-aware protocols, energy and computationally efficient nodes can be positioned in a monitored space is a necessity.

- **Data Handling**

  Embedded systems are dedicated to specific tasks that seek to aid human experience. One way these systems provide value is by providing access to data from which humans may draw valuable insights. The development of IoT technologies has enabled more data collection in monitored environments and transmission to end-users. It is apparent that advances in WSNs and micro-electromechanical systems have facilitated the deployment of multiple heterogeneous low-cost sensing devices in various scenarios [22]. These sensing devices usually provide data of high temporal coherence with redundant data sequences in packets being transmitted, thus continuous data transmission is often unnecessary, computationally demanding and energy consuming [128]. Work by [155] proposes a prediction-based data fusion scheme that tries to reduce transmissions of raw data at sensor nodes. Although using more samples provides for better accuracy and resolution in obtaining information from sensed phenomena, it requires computation and uses significant amounts of energy which may not be readily available. Research by [139, 205, 221] proposes techniques for adaptive sampling that reduce energy consumption while retaining high accuracy. Another challenge worth noting is that raw sensed data is usually 'noisy'. In the case of applications monitoring physical phenomena, (i.e., wildlife tracking), data collected is usually incomplete, noisy and unreliable owing to sensor and

transmission errors and environmental conditions. How data is handled, from sampling, communication, pre-processing and transmission to end users affects the overall quality of delivery of a WSN in respect to its application.

- **Tedious System Design and Implementation**

  The constraints mentioned above make the implementation of Embedded systems, particularly WSNs a difficult task. Expertise in design and engineering has to go into the development of these systems owing to the need to develop low-cost devices that will operate autonomously for long periods of time. Ensuring scalability of these systems through accurate data gathering, processing and communication requires highly optimized algorithms capable of delivering tasks in a resource-aware manner. Work by [136] presents a novel approach that automatically generate codes for general-purpose parameter tuning in a sensor network as demonstrated on a target tracking application. Since the design of Embedded systems and WSNs takes into account product utility and resource constraints, work that enables more intelligent ways for these systems to adapt based on their observed environment with little reprogramming is key.

1.3 Key Research Questions

The constraints discussed in the previous section warrant the term, Resource Constrained Systems. In Chapter 2, Section 2.1, we provide a detailed definition of what we refer to as, Resource Constrained Systems.

In this thesis, we seek to address some of the challenges we have highlighted in Section 1.2 by harnessing the ability of Machine Learning algorithms, i.e., Recurrent Neural Networks capable of communicating and learning long term temporal dependencies within data. In doing so, we ask the main research question:

How can we exploit Deep Learning to provide automated distributed inference for Resource Constrained Settings?

1. **How can we adapt neural architectures which are typically centralized in a single computational unit (e.g., GPU) and instead use them to form a distributed neural network?** In asking this, we are interested in how we can instantiate a distributed architecture that is suitable for implementation in multi-node IoT scenarios. In such scenarios many resource constrained

devices are deployed to sustainably monitor dynamic phenomena. A key consideration we make is the approach to achieving latent-state communication focusing on the input to inference pipeline.

2. **How do we optimize our model across multiple problem domains?** Here we consider model optimization on naive and visual settings. In doing this, we are concerned with exploring how we can better learn dependencies in distributed problems while achieving communication-efficiency. We ask how model performance is impacted by varying hyper-parameters and model optimization techniques. Note that this extends traditional concerns in deep learning i.e., a core focus on accuracy, and instead considers multi-objective optimization trade-offs across dimensions of communication efficiency and accuracy.

3. **How do we achieve model robustness and scalability?** Here we seek to address the issue of model robustness. In resource constrained settings, link failure is a long-standing challenge. We seek to address how our model can be robust to failing nodes. Moreover, we explore how our model can generalize to unseen wireless network topologies during testing. This is an important component of achieving a robust model. We also navigate how we can make steps toward accuracy-preserving scalability.

1.4 Contributions

The major contributions of this work to date are as follows:

1. A fully functional communicating Recurrent Neural Network *(C-RNN)* framework that can be instantiated on multiple network nodes to cooperatively solve distributed tasks. We believe that this is the first example of deep neural networks being used to perform and mediate in-network processing.

2. A technique to selectively control latent state communication amongst multiple nodes based on node connections and an internal communication aggregator and gating function. The technique encompasses a communication loss and allows for further optimizations on communication channel sizes.

3. A dynamic graph for network nodes that is alterable during training. The method allows the distributed communication architecture to generalize to unseen wireless network topologies.

4. An end-to-end trainable framework to extract visual features on network nodes and solve distributed classification tasks via latent state communication.

5. Extensive experiments on distributed tasks showing the ability to train models well, evaluations of accuracy and communication cost trade-offs, network robustness and computational overhead. We demonstrate how this makes the proposed architectures viable for resource constrained distributed IoT settings.

1.5 Publications

Key contributions of this thesis have been published at the following international peer-reviewed conferences and journals:

1. **Prince M Abudu**, "CommNets: Communicating Neural Network Architectures for Resource Constrained Systems." Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), 2019.

2. **Prince Abudu** and Andrew Markham, "Distributed Communicating Neural Network Architecture for Smart Environments." Proceedings of the IEEE International Conference on Smart Computing (SMARTCOMP), 2019.

   In this poster and paper, we address the question of whether state-of-the-art Recurrent Neural Networks can observe different series of data on multiple nodes and communicate their hidden states to collectively solve an objective in a distributed fashion. In doing this, we propose a novel distributed communicating neural network architecture for Resource Constrained Systems. We present early results on small setup (2/3 nodes) that provide promise for automated distributed inference using RNNs via hidden-state communication. This work is presented in Chapter 5 of this thesis.

3. **Prince Abudu** and Andrew Markham, "Deep Emergent Communication for the IoT." Proceedings of the IEEE International Conference on Smart Computing (SMARTCOMP), 2020.

4. **Prince Abudu** and Andrew Markham, "Learning Distributed Communication and Computation for the IoT." Special Issue on Machine Learning approaches in IoT scenarios, Elsevier Computer Communications Journal, 2020.

In these two papers, we present a novel scalable, data-driven and communication-efficient Convolutional Recurrent Neural Network (C^2RNN) framework for distributed tasks. We show that our proposed model can achieve desirable results in a scalable, robust, and communication-efficient manner without any complex, hand-engineered communication protocols. Moreover, we show that our model can perform distributed inference on image classification tasks using limited computational resources available on embedded IoT devices. This work is presented in Chapter 6 of this thesis.

1.6 Thesis Structure

The rest of the thesis is structured as follows. Chapter 2 provides an overview of Resource Constrained Systems, Cloud IoT, Fog and Mist Computing, Machine Learning in IoT settings and state-of-the-art techniques on Cooperative Deep Networks in IoT settings and Communication in Deep Networks. Chapter 4 presents a generic Distributed Communicating Neural Network Architecture which can discover and optimize communication protocols across network nodes. Chapter 5 focuses on solving distributed problems in naive settings with sensor-like data using our *C-RNN* architecture. Chapter 6 presents a scalable, communication-efficient Convolutional Recurrent Neural Network (C^2RNN) framework for distributed inference on visual data. Chapter 7 presents a distributed grid-based target tracker for naive and visual scenarios. Chapter 8 concludes the thesis and provides a discussion on ideas for future work. Figure 1.3 provides an outline of how the thesis is organized with the proposed architectures.

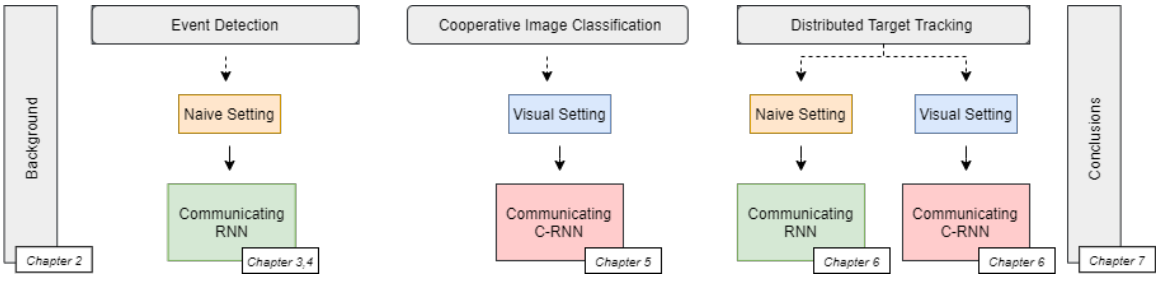


Figure 1.3: Thesis structure outline with proposed framework and problem settings.

Chapter 2

Background

The ever-growing number of interconnected, heterogeneous, highly distributed devices in Resource Constrained Systems continues to necessitate the need to develop computing paradigms that can operate these devices whilst meeting stringent application requirements in a resource-aware fashion. In mission critical systems and applications, latency sensitivity is an important factor. There are several methods that have been employed to achieve data reduction, energy-efficient communication, and dynamic and fault tolerance on sensing devices operating in IoT environments. Nonetheless, most methods in traditional Cloud IoT and Wireless Sensor Networks (WSNs) instantiate devices with complex hand-engineered communication protocols, limited intelligence and immense reliance on Cloud support. Consequently, issues associated with communication bandwidth, data transmission, connectivity and latency not uncommon. More detailed surveys on Cloud IoT and WSNs can be found in [10, 20, 54, 164, 174, 192, 198]. Recently, we have seen works on decentralized computing paradigms in which distributed collaboration and substantial data processing is pushed to edge devices in a system [97, 106]. Moreover, edge devices are endowed with ML-aided intelligence. This is ideal for many applications in which Cloud access is not guaranteed and up-link communication costs negatively impact system performance.

In this chapter we position our work in relation to literature on various computing paradigms in Resource Constrained Systems. Firstly, in Section 2.1, we characterize Resource Constrained Systems by categorizing them into Wireless Sensor Networks (WSNs), Cloud IoT, Fog and Mist Computing and End devices. We then provide a review of widely studied Deep Network architectures discussing their capabilities and instantiation in Resource Constrained Systems in Section 2.2. Lastly, we discuss Deep Network-based in-network processing. In doing so, we survey enablers and identify

opportunities for this newer research area that seeks to embed Deep learning-based intelligence at the network edge.

2.1 Resource Constrained Systems

Any system that operates under resource constraints, can effectively be referred to as a Resource Constrained System. The design of Resource Constrained Systems must address constraints on available resources and communication protocols while making a graceful trade-off between utility and execution cost. Figure 2.1 shows a high-level characterization of Resource Constrained Systems.

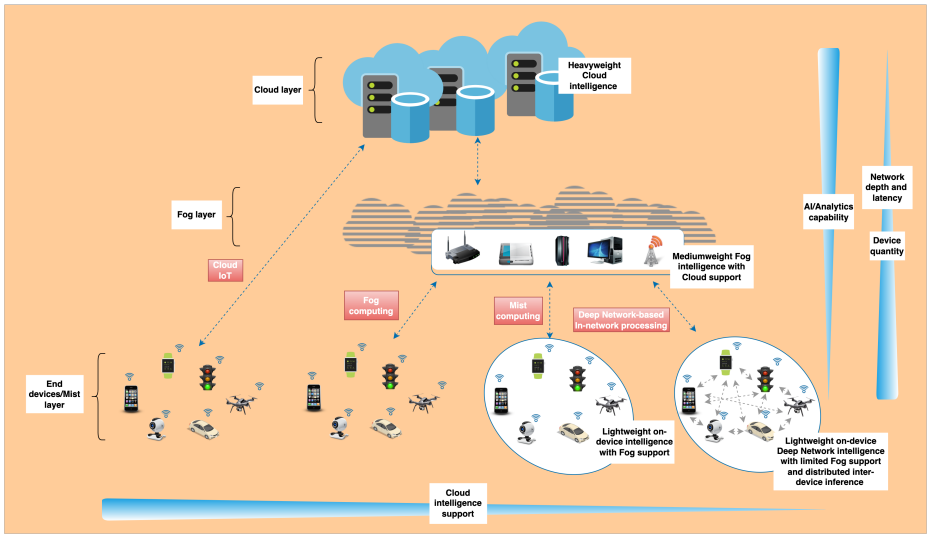


Figure 2.1: High level characterization of Resource Constrained Systems showing interacting components.

Table 2.1 shows a summary of features of specific Resource Constrained Systems. A comprehensive discussion of Deep Network-based In-network processing is provided in Section 3.1.

Below we offer some examples of Resource Constrained Systems with generic applications;

Wireless Sensor Networks in Environmental Monitoring: With the increase in levels of industrialization, new concerns over excessive water and air pollution have surfaced. These concerns have led to efforts to monitor environmental parameters so as to manage levels of atmospheric pollution and reduce consumption of natural resources. WSNs can be deployed to measure water pH levels, temperature and turbidity. Moreover, sensors can be deployed to monitor gas emissions through measuring levels of carbon monoxide, nitric oxide and sulfur oxide. It is imperative

Table 2.1: A summary of Cloud IoT, Fog, Mist and Deep Network-based In-network processing features.

Feature	Cloud IoT	Fog	Mist	Deep Network-based In-network processing
Latency	High	Relatively Low	Low	Low
Placement	Centralized, far from users	Decentralized, closer to users	Distributed and pervasive, close to users	Distributed and pervasive, close to users
Storage and Computational Power	Virtually unlimited	Limited	Limited	Limited
Intelligence Capability	Very High	High	Low	Relatively High
Service Availability and Use	Cloud service providers and end device users	Cloud and Fog service providers and end device users	Self-organized device users, with limited Cloud and Fog service	Self-organized device users, with limited Fog service, intelligent inference and training platform providers
Real time scenario support	Limited to none	High	High	High
Data Processing	Centralized	Decentralized, with centralized Cloud support	Localized, on-device with minimal Cloud support	Localized, on-device with very minimal Cloud support
Network Hardware and Devices	Data centers and virtualized platforms	Servers, routers and switches	Sensors, actuators, cameras, mobile phones	Sensors, actuators, cameras, mobile phones
Mobility	Low	Low	High	High
Typical and Emerging Applications	Large scale data storage, big data analytics and monitoring on health, environment, traffic and satellite data.	Smart cities, smart homes, smart energy management, health monitoring, intelligent transport systems	Unmanned aerial vehicles (UAVs), traffic monitoring, smart cities, smart health, emergency services, self-driving cars	UAVs, augmented reality, cooperative target tracking, distributed crowd monitoring and video surveillance, connected cars, mobile image recognition

that such a system operates autonomously for long periods of time. Since sensor nodes in these systems are typically powered by lithium batteries, this creates a constraint on the energy supply, thus creating a need for resource-aware techniques to maximize the life of the system and ensure the environment is constantly monitored to meet user needs.

2.1.1 Wireless Sensor Networks (WSNs) and IoT

The basic concept of the IoT is concerned with the instantiation of paradigms which transform physical objects around us that we use every day into an information ecosystem [19]. Although the original ideas around IoT may have sprung to life in the 1980s, Kevin Ashton, a British entrepreneur, asserted that all things characterized by RFID and internet connectivity for management and localization can be termed, "Internet of Things" [18]. Moreover, the term has also been widely attributed to Auto-ID Labs [1], a world-wide network of academic research laboratories in the field of networked RFID and emerging sensing technologies. [34] posits that the IoT paradigm is based on intelligent and self-configuring nodes (things) interconnected in a dynamic and global network infrastructure. Such an arrangement is commonly termed, Wireless Sensor Networks (WSNs). WSNs can be thought of as a precursor to IoT. In WSNs, IoT end devices, (i.e., sensors, mobile devices, actuators) detect, measure, sense, compute, localize, communicate, and in some cases, adapt to sensed phenomena. These devices play a significant role in acquiring data for analytics within the overall IoT paradigm, thereby, providing real-time application-specific insights to end users. Consider an example in home automation. Temperature sensors, water level sensors, pressure sensors, sound sensors and cameras are used to observe the home

environment. In some cases, these devices "things" are connected to the internet via a control system paradigm that allows for automatic control of appliances at home. Our understanding of WSNs and the capabilities we have discussed position us to define IoT as; *a paradigm in which "things" - these being devices and objects capable of computation, communication and pervasive operation, provide insights and actuations that add value to human life in real world environments.* Manifold definitions of IoT traceable within the research community and a survey of IoT architectures are presented by [20, 174]. WSNs have been widely employed in several application domains [26, 32, 64, 65, 84, 102, 157, 163, 196, 197]. IoT and WSN are, in many cases, synonymous.

2.1.2 Cloud IoT

The pervasive presence of devices and objects that are perennially collecting and processing data continues to motivate the integration of Cloud computing [100, 142] and IoT. Moreover, IoT end devices are often plagued by resource constraints on their computing platforms, storage capacity and energy efficiency. Contrary to that, Cloud computing platforms are endowed with virtually unlimited computing capabilities and storage capacity. Such a reality suggests complementarity and an ostensible inseparability of Cloud computing with IoT. As means for compensation of its resource constraints, IoT can benefit from Cloud computing's virtually unlimited capabilities and resources [34]. Research by [54, 200] accentuates issues related with scalability, interoperability, reliability, availability, and flexibility of heterogeneous IoT end devices - depicting how Cloud IoT architectures can be successful in mitigating such limitations. [34] postulates that Cloud IoT is a paradigm in which Cloud and IoT are two complementary technologies merged together. In the quest to realize efficient IoT operation, [31] defines Cloud of Things (CoT), as a mapping of heterogeneous physical devices to the virtual domain of the Cloud. CoT and Cloud IoT are, in principle, akin. Going by [34, 199]'s work, we posit that Cloud IoT is a convergence of Cloud computing and the IoT, in which the two are intimately coupled and complementary. Such a convergence allows for IoT end devices and applications to leverage the agility, location independence, multi-tenancy, reliability, scalability and accessibility of Cloud computing.

2.1.3 Fog Computing

In geospatially distributed, large scale and latency sensitive applications, such as, disaster recovery systems, instant dynamics in environment conditions must be propagated to relevant applications without delay. In such scenarios, Cloud IoT systems fail [208]. To complement traditional Cloud IoT systems and address issues associated with guaranteeing continuous connectivity, mobility-support, efficient data traffic management and storage, and reliable latency, there has been a move to paradigms that offer decentralized, time-aware computations at the edge of network devices. That approach has been commonly termed Fog computing. In NIST [97]'s perspective, Fog computing is defined as; *a layered model for enabling ubiquitous access to a shared continuum of scalable computing resources.* NIST further describes this model as being an enabler of distributed and federated computing that facilitates latency-aware applications and bridges the gap between smart-end devices and centralized Cloud IoT services. In their definition, Cisco [33] acknowledges the need to have Cloud IoT platforms that can offer mobility support, low latency, location awareness and geospatial distribution and posits Fog computing as; *a highly virtualized platform providing compute, storage and networking services between end devices and traditional Cloud computing data centers, typically but not exclusively located at the edge of the network.* [228] provide an integrative definition of Fog computing that encompasses the technology's features and impacts. [231] give a characterization of Fog computing, while comparing it to other related edge computing paradigms, i.e., Mobile Edge and Cloud Computing (MEC, MCC). For this survey, it is imperative that we favor both NIST and Cisco's definitions for the avoidance of confusion, as this is an emerging technology and many definitions continue to be presented in literature.

2.1.4 Mist Computing

Put plainly simple, Mist computing can be thought of as a rudimentary and lightweight form of Fog computing, that is intrinsically enabled by micro-controllers existing at the very network edge [97]. Intriguingly, one author at IBM Developer [132], draws their explanation of Mist computing from day-to-day understanding of meteorological cloud and fog phenomena. In their explanation, Cloud computing is far removed from human activities whereas Fog computing happens beneath the 'clouds' and Mist computing happens on the ground. Drawing on that understanding and the NIST definition [97], we define Mist computing as; *a computing paradigm that provides sensors and actuators at network edges with increased autonomy, intelligence and local*

data processing capabilities. It is noteworthy that in the Mist computing paradigm, IoT end devices can essentially pre-process data, thereby reducing the amount of computationally intensive communication to central Cloud IoT systems supporting such devices. This reality augments latency sensitive applications and facilitates heterogeneous deployments of IoT end devices.

2.1.5 End Devices

In a Cloud IoT and Fog system, end devices are located within the Mist layer. Such devices consist of highly distributed and resource-constrained wearable, mobile and embedded devices at the edge of the network fabric, (i.e., sensors, actuator controllers, metering devices and security cameras). Typically end devices act as a primary source and or destination of data being transmitted over any network. Mist layer end devices are tasked with collecting, analyzing and processing massive amounts of data in real-time. To achieve this, these devices must have storage, networking and computing capabilities [33]. Moreover, they must conduct a communication efficient operation that meets latency, bandwidth and reliability requirements. In asset monitoring applications, i.e., infrastructure and environmental monitoring, drones could be equipped with end devices, such as, on-board navigation and sensors, cameras and actuators [212]. These end devices on such drones have to persistently perform real-time computations and actuations based on observed data. Fog computing endows end devices, as such with necessary capabilities that aid location awareness, geographical distribution, scalability, mobility, real-time interaction, heterogeneity, interoperability and interplay with the Cloud. Table 2.2 shows classes of IoT devices and expected minimal resources alongside applications.

2.2 Deep Networks

In the previous section, we characterized Resource Constrained Systems. Now we turn to investigating the promise of ML in these environments. Many applications continue to benefit tremendously from ML algorithms. That observation positions us to believe that ML has constituted and will play a more decisive role in Resource Constrained Systems. In traditional Cloud IoT systems, IoT end devices have been supported with ML capability for analytics and inference from the Cloud. Typically, massive amounts of data are collected at end devices and transmitted to the Cloud for processing by ML models. These complex models have harnessed elastic computational resources and virtually unlimited storage facilities in the Cloud. Nonetheless,

Table 2.2: Classes, minimal resources and applications of IoT end devices

Class	Description	Device	Resources	Applications
Class 0	Extremely constrained, compact, lightweight and low-power devices with very little memory and processing capacity.	Waspmote	**RAM:** 8 KB, **Clock Speed:** 14.74 MHz, **Flash:** 128 KB	Character Recognition [78]
		Airboard Light-Blue Bean+	**RAM:** 2.5 KB, **Clock Speed:** 8 MHz, **Flash:** 32 KB	Gesture Recognition [110]
		Arduino Uno	**RAM:** 2 KB, **Clock Speed:** 16 MHz, **Flash:** 32 KB	
Class 1	Constrained, low-power devices which cannot easily communicate with devices employing a full IoT stack, e.g., HTTP/TLS since they mostly use CoAP.	NXP Kinetics EA	**RAM:** 16 KB, **Clock Speed:** 48 MHz, **Flash:** 128 KB	Gesture Recogntion [110]
		NXP LPC 2000	**RAM:** 16 KB, **Clock Speed:** 60 MHz, **Flash:** 128 KB	
		EFM M32G	**RAM:** 16 KB, **Clock Speed:** 32 MHz, **Flash:** 128 KB	
		BBC Micro:Bit	**RAM:** 16 KB, **Clock Speed:** 16 MHz, **Flash:** 256 KB	
Class 2	Less constrained devices that can perform at par with mobile phones . They have increased interoperability, need to have energy-efficient protocols and must have less bandwidth consumption.	Arduino MKR	**RAM:** 32 KB, **Clock Speed:** 48 MHz, **Flash:** 256 KB	Gesture Recognition [165]
		Feather MO	**RAM:** 32 KB, **Clock Speed:** 48 MHz, **Flash:** 256 KB	Activity Recognition [111]
		ESP 8266	**RAM:** 32 KB, **Clock Speed:** 160 MHz, **Flash:** 1MB	
Class 3	Use existing protocols with minimal changes or modifications. can be affected by a limited power supply.	Intel Joule	**RAM:** 3GB, **Clock Speed:** 1.7 GHz, **Flash:** 16 GB	Keyword Spotting [235]
		Intel Edison	**RAM:** 1GB, **Clock Speed:** 512 MHz, **Flash:** 4 GB	Object Recognition [118, 123]
		Rasberry Pi 3	**RAM:** 1 GB, **Clock Speed:** 1.2 GHz, **Flash:** 16 GB	Image Classification [29]
		ESP32	**RAM:** 384 KB, **Clock Speed:** 160 MHz, **Flash:** 16MB	Pedestrian Detection [156]
		Dragon Board	**RAM:** 1 GB, **Clock Speed:** 1.2 GHz, **Flash:** 8 GB	Smart Industry [160]
		LG Nexus 5	**RAM:** 2 GB, **Clock Speed:** 2.8 GHz, **Flash:** 16 GB	

owing to the highly distributed nature of components and latency requirements of Fog environments, we envision an aggressive push of ML to the network edge, endowing end devices with more intelligence for inference, predictive, feature extraction, classi-

fication, pattern recognition, communication and cooperative tasks. Throughout the thesis, we use the umbrella term *Deep Network-based In-network processing* to encompass the use of resource constrained, distributed deep learning . In Section 3.1, we present timely ideas for Deep Network-based In-network processing. Before doing so, we provide a discussion of Deep Networks, offering a review of algorithms, architectures, prevalent application domains, current limitations, and recent advancements - thereby motivating their applicability to Resource Constrained Systems.

2.2.1 Overview

As the naming suggests, Deep Networks are a class of 'deep' (consisting of multiple layers) computational models capable of instantiating information processing and communication techniques that learn data representations and abstractions. To meet their objectives and depict expected behavior in any domain, Deep Networks must be trained from raw data. The training process enables these networks to automatically extract complex data representations at high levels of abstraction, thereby enabling domain-specific data analysis [154]. Nonetheless, this process is non-trivial and computationally intensive. Traditional methods for training multi-layer deep networks could not guarantee convergence, were time consuming, and feature learning and representation could not be achieved to meet performance requirements [233]. This motivated consequential research that saw Deep Networks experimentation gaining traction. Such works include [28,90]'s structured layer-by-layer greedy learning methods and the realization of pre-training and fine-tuning for efficiently learning deep models [86], thereby reducing data dimensionality and learning compact data representations. The end-to-end process of training Deep Networks to extract features, discover hierarchies of data representations is widely termed Deep Learning (DL). DL methods are generally categorized as *supervised learning*, *unsupervised learning*, *semi-supervised learning*, *reinforcement learning* and *transfer learning*, among other categorizations.

- **Supervised Learning:** In *supervised learning*, a model infers a function from labeled training data of input features and their corresponding outputs. A common *supervised learning* technique is, *classification* - a method to predict target classes or categories of data based on training data of acknowledged category membership. Another supervised technique is, *regression* - a method that learns relationships amongst variables with continuous output. [36] provide a comprehensive discussion of *supervised learning* algorithms. Contrary to *supervised*

learning, *unsupervised learning* infers a function to describe the underlying distribution and hidden structure of 'unlabeled data'. A common technique in *unsupervised learning* is, *clustering* - a method used to input data into classes on the basis of their statistical properties. Another technique is *association* - a rule-based method for discovering relationships and patterns amongst variables in data. [82] provides a detailed discussion of *unsupervised learning* and associated techniques.

- **Semi-supervised Learning:** This DL method is a hybrid of *unsupervised and supervised learning*. [237] survey *semi-supervised learning* and the techniques employed in this learning paradigm. In *reinforcement learning (RL)*, the model is expected to maximize a numerical reward signal from mapping situations to actions [204].

- **Reinforcement Learning:** In this DL method multiple agents are incentivized to communicate optimally and solve tasks specified by users in playing games, i.e., Atari from pixels [147] and controlling robot coordination [116].

- **Transfer Learning:** In *transfer learning*, learned knowledge in one task is stored and reused for other tasks. [159] provides an in-depth discussion of transfer learning.

These learning methods continue to inspire novel instantiations of Deep Networks being adopted to solve complex tasks in various application domains. Owing to their ability to automatically discover underlying patterns in observed data without being explicitly programmed, these networks have reported massive successes in literature. Widely celebrated successes have been demonstrated in the ImageNet classification challenge [108], playing expert-level Go game [194], handwritten digit recognition [114], speech recognition [40, 88], big data analytics [104, 154, 220], medical image analysis [122, 190], image recognition [83, 195], text understanding and natural language processing [71, 134, 234] and computer vision [98, 109]. In these domains, Deep Networks have surpassed human level performance. For instance, [194]'s AlphaGo displayed 99.8 % winning rate and defeated a European professional Go game player - making the first time an AI algorithm defeated a human in this game. Such a demonstrated success motivates the continual exploration of Deep Networks in various domains characterized by massive amounts of observed data. Resource Constrained Systems cannot be exempt from taking advantage of AI capabilities to empower current and future applications and visions.

A wide array of surveys [56, 122, 126, 233] worth noting in literature discuss Deep Networks from varying perspectives, i.e., algorithms, architectures, techniques, training, optimization, challenges and applications. [11,148] provide timely surveys of deep learning in Cloud IoT environments. Evidently in literature, several Deep Network architectures and algorithms have been proposed over the past decade. [56] categorizes these into generative or discriminative. Generative techniques learn the joint probability distribution of given inputs and outputs. On the other hand, discriminative techniques learn the conditional probability distribution.

2.2.2 Deep Network Architectures

Here, we briefly discuss three typical discriminative Deep Networks; CNNs, RNNs, LSTMs. We also summarize other popular techniques which have been widely employed in Cloud IoT to date, based on ML.

2.2.2.1 Convolutional Neural Networks (CNNs)

CNNs [114] enjoy a great deal of attention in computing research. This is mainly because they have achieved desirable performance on various tasks. CNNs stem from early research by [72, 95]. Motivated, by the organization of the animal visual cortex, CNNs can process two-dimensional data with grid-like topology, i.e., visual imagery [126]. Diverse application scenarios inspire architectural variations of CNNs, mainly employed to analyze visual imagery, extract features and classifies input images wholesomely. Popular architectures include 'shallow' networks, (i.e., LeNet [114], AlexNet [108]), and 'deep' architectures, (i.e., VGG [195], GoogLeNet [206], ResNet [83]). Owing to their capabilities, CNNs can be adopted for various tasks in Cloud IoT, and Fog environments. In such environments, end devices, (i.e., cameras and vision sensors) are constantly collecting visual data. CNNs can employ real-time processing techniques to enable video surveillance, object tracking, intrusion detection and industrial quality control among other tasks. In such applications, object detection is a crucial task. Several CNN architectures have been proposed in literature for general purpose and real time object detection tasks, i.e., Region proposals CNN (R-CNN) [75], Fast R-CNN [74], Faster R-CNN [178], Single Short Multibox Detector (SSD) [125], You Only Look Once (YOLO) [175], YOLO9000 [176].

2.2.2.2 Recurrent Neural Networks (RNNs) and Long Short-Term Memory Networks (LSTMs)

Unlike CNNs which have been successfully employed in modeling visual data, RNNs are widely used in modeling long sequences of data with temporal dependencies, in most cases, such sequences are non-visual. RNNs [182] were first developed in the 1980s. Owing to their prowess in modeling sequential data, RNNs have been applied to various problem domains. RNNs' successes in these applications have hugely been fueled by their ability to account for time series properties in data being processed. For instance, in speech recognition tasks, algorithms must identify phrases and words in spoken language and convert these into a format comprehensible by a machine. To achieve this, algorithms must incorporate contextual information in previous words and or phrases to successfully model following words. Despite their demonstrated successes, RNNs are difficult to train over longer term dependencies. Training methods suffer from issues to do with generalization degradation, owing to vanishing gradients and overfitting [140]. To address these issues, several variations of RNNs have been proposed in literature, i.e., Light RNN [120], Hierarchical Multiscale RNN [48], Deep RNNs [162], Relational RNNs [187], Reversible Recurrent Neural Networks [131] and Gated Recurrent Units [47]. Another widely known architecture is the Long Short-Term Memory Network (LSTM) [92]. The LSTM has been proven to solve the problem of gradient vanishing and gradient exploding during RNN training [76]. This has enabled successful modeling for longer term dependencies in language modeling [203] and speech recognition [185]. RNNs and LSTMs can be employed for these tasks and any task characterized by data with temporal dependencies. Cloud IoT and Fog environments are pregnant with such tasks. For instance, these models can be used in activity recognition and event detection from wearable sensor data, object tracking and monitoring patient health data among other application domains. [56, 122, 233] provide a more detailed discussion of RNNs.

2.2.2.3 Hybrid CNN-RNN Architectures

Deep Network architectures continue to evolve to suit a wide range of tasks in IoT settings. [113, 222] experiment with standalone CNN architectures on image sensor data considering resource constraints. [166] propose an RNN-based architecture that extracts features from raw sensor data and performs recognition tasks. These standalone architectures have proved to work in problem domains where feature extraction from image and/or sensor data is necessary before making predictions. Nonetheless,

hybrid architectures employing both CNNs and RNNs have reported further performance gains in the same domains, (i.e., recognition [94, 191, 236] and image classification tasks [217, 229].) Work by [94] shows that a hybrid CNN-RNN architecture for gesture recognition outperforms pure CNN architectures. Further work by [236] compares a pure RNN approach to a CNN-RNN approach to action recognition in 3-D videos. Their work reports a 13 % increase in accuracy in the hybrid approach compared to the pure RNN approach. [217, 229] propose an end-to-end trainable unified CNN-RNN framework for image classification.

2.2.3 Training Deep Networks

Neural Networks (NNs), i.e., DNNs, CNNs, RNNs and LSTMs have reported significant successes in various applications domains. Thanks to science, NNs do not work on magic! Neural Networks are able to effectively solve tasks and learn from dependencies in data without being explicitly programmed. Their information processing can be hugely attributed to training and optimization. [146] discusses NN training using an interesting analogy;

> *Imagine you are a mountain climber on top of a mountain, and night has fallen. You need to get to your base camp at the bottom of the mountain, but in the darkness with only your dinky flashlight, you can't see more than a few feet of the ground in front of you. So how do you get down? One strategy is to look in every direction to see which way the ground steeps downward the most, and then step forward in that direction. Repeat this process many times, and you will gradually go farther and farther downhill. You may sometimes get stuck in a small trough or valley; in which case you can follow your momentum for a bit longer to get out of it. Caveats aside, this strategy will eventually get you to the bottom of the mountain.*

As much as this analogy may seem detached from NNs in reality, in principle it is not. The ultimate goal of training a NN is to learn its features and the best set of weights that maximize the network's accuracy in relation to a given objective defined over some domain. A commonly used approach to training NNs is Stochastic Gradient Descent [182] which minimizes network errors by back-propagation. Other methods include using the Alternating Direction Method of Multipliers (ADMM) and Bregman iteration which decomposes network training into a sequence of sub-steps solved to global optimality [209]. Although both methods have shown advanced benefits, their

implementation require serious optimization efforts. Decades of research have gone into exploring problems associated with training NNs mainly caused by the highly volatile relationship between network parameters and the network's hidden states [138]. A classic manifestation of this volatility is the vanishing and exploding gradient problems [27,91], experienced in gradient-based methods of training. In the vanishing gradient problem, errors experience exponential decay through time leading to very small or no updates to the weights of the network. Exploding gradients are caused by large errors that grow exponentially over time, thereby affecting network learning.

Several efforts have gone into addressing vanishing/exploding gradient problems. Among these are Long Short-Term Memory Networks (LSTMs) [92] - capable of modeling long-term dependencies by instantiating memory cells to 'remember' information over time; Echo-state-Networks [99,130] - which use randomly assigned sparse connections, hidden neurons and weights; and Hessian Free learning [137] - which employs a second order approach to minimizing the local quadratic approximations of the objective function. In spite of these advances, training NNs remains a non-trivial task that requires efficient optimization. Most optimization techniques are designed in tune with the structure of the problem domains in question, often at the expense of performance in other problem domains. Popular optimization methods in literature include Rprop [179], RMSProp [207], ADAM [105] and Adagrad [57]. Much recent work by [16] casts the design of optimization algorithms as a learning problem. Further work by [215] employs a transfer learning approach to NN training. Although, all the approaches discussed have shown admirable results in many training cases, the area pertaining to effectively training and optimizing NNs for distributed tasks and Resource Constrained Systems remains unexplored.

2.3 Discussion

As people, objects and 'things' in the world become increasingly interconnected, we are experiencing a data revolution. This imminent reality has changed perceptions within computing research, leading to the upsurge and advancement of ML research. With the ultimate aim to use data to improve human lives, AI algorithms have had celebrated successes. Although such advancements are worth the attention, the field of AI has not yet reached its full maturity, let alone in Cloud IoT, Fog and Mist computing domains. This in turn, creates opportunities for research and industry to develop ML algorithms for tasks consummate to these environments. In full realization of this, we have provided a thorough discussion of key challenges in Cloud IoT,

Fog and Mist computing. We showed our acknowledgement of Fog and Mist computing, as emerging technologies by providing a comprehensive and timely survey of techniques that seek to address the challenges we discussed. We then introduced key AI algorithms, discussing various types of Deep Networks, (i.e., CNNs, RNNs, LSTMs, AEs, and GANs). In doing this, we demonstrated the value of these networks, and motivated their potential applicability to Fog and Mist environments.

Whilst there may have been individual works that have implemented Deep Networks in Cloud IoT and distributed Fog computing scenarios, we are interested in Deep Network-based In-network processing as an emerging paradigm. We believe future works will explore this paradigm to enable highly accurate, robust, latency-aware and resource-aware distributed learning and inference on end devices in holistically integrated Cloud IoT, Fog and Mist environments. In the next section, we discuss some works toward this direction and position our efforts within the context of these works.

Chapter 3

Preliminaries and Related Work

In this chapter, we position our research in the context of related work. Consequently, we discuss Deep Network-based In-network Processing, synonymously referred to as Machine Learning at the Network Edge. We also provide motivation and enablers of this timely paradigm within IoT-based systems. We also survey closely related Machine Learning systems deployed on end devices (as defined in Section 2.1). In doing this, we offer systemic and comparative analysis of recent works within the area of Deep Network-based In-network Processing. Our work primarily focuses on cooperative inference and communication aspects of distributed network nodes seeking to achieve a global sensing objective.

3.1 Deep Network-based In-network Processing (Machine Learning at the Network Edge)

Nodes in Wireless Sensor Networks (WSNs) are typically resource-constrained owing to limited computational power, reduced communication bandwidth and little storage capacity. To minimize energy consumption incurred from communication costs during data transmission, intermediate nodes are typically selected to perform complex and computationally expensive data transformation tasks, i.e., data aggregation, compression and in some cases inference on a defined network function [230]. When data processing and transformations are done within the networks, this is referred to as In-network processing. Deep Network-based in-network Processing, builds upon fundamentals of In-network processing by instantiating Deep Learning algorithms within a Wireless Sensor Network graph for data processing tasks, learning objective functions and inference on network nodes, closer to the data source. Figure 3.1

shows a high-level setup of components within a Deep Network In-network Processing paradigm.

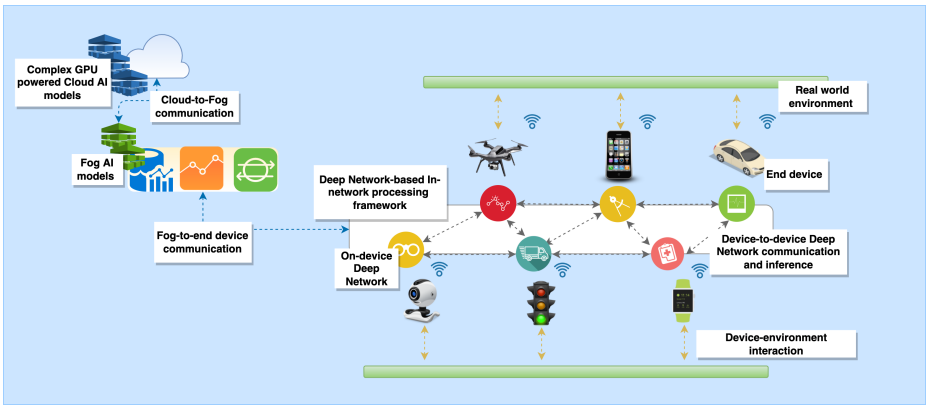


Figure 3.1: Deep Network-based In-network Processing paradigm showing inter-layer and inter-device interactions

In traditional IoT scenarios, complex and computationally intensive ML algorithms have been constrained to the Cloud, which has offered the illusion of infinite compute resources. Moreover, Cloud IoT systems have classically, implemented lightweight ML algorithms for simpler classification and regression tasks. Some of these techniques could be hosted on end devices, with online Cloud support for computational offloading. Nonetheless, these techniques have proven to be less accurate and fail to efficiently capture complex dependencies in massive heterogeneous datasets. Moreover, the ever-growing number of interconnected, heterogeneous, highly distributed Cloud IoT end devices necessitates the need to develop Deep Network-based computing paradigms that can adaptively operate these devices while meeting stringent application requirements in a resource-aware fashion.

Any implementation of Deep Networks in Mist computing can be thought of Deep Network-based in-network Processing. Advances in the development of hardware devices, frameworks, techniques, architectures and platforms suited for distributed AI in Resource Constrained Systems warrant myriad opportunities for research and instantiating of Deep Networks on end devices for large-scale distributed learning and inference. Predicated upon the inherent need to satisfy device heterogeneity criterion, achieve optimal device-to-device (D2D) communication, support mobility requirements, reduce network latency and provide in-built fault tolerance, Deep Network-based In-network Processing enables the realization of distributed intelligence in Cloud IoT end devices. This paradigm enables, both distributed training and inference on end devices, with minimal to no Cloud support. We continue to see

embedded processing of deep networks in end devices emerging as a trend. The goal of this approach is centered on enabling data collection, processing and analysis on local sensor nodes, thereby mitigating communication cost and latency challenges whilst imbuing the network with privacy preserving properties. Several works in literature have proposed approaches for instantiating Deep Network architectures on resource-constrained IoT devices. For example, [78] propose a k-Nearest Neighbor (KNN) algorithm that significantly reduces storage and prediction complexity on Deep Network models, enabling them to be deployed on resource-constrained IoT devices, e.g., Arduino UNO with 2kB RAM. With just 16kB model size, their approach can handle general supervised learning problems. [111] propose multiple instance formulation and an early prediction technique for fast and efficient classification of sequential data on Raspberry Pi and Arduino Uno devices. Further work by [110], propose a novel tree-based algorithm for prediction on a similar Arduino Uno device used in [78]. These works although helpful in enabling efficient inference and reducing storage requirements, they are limited to centralized, single node prediction settings. Similarly, we focus on developing methods for making local predictions on tiny IoT devices without any cloud support. Our work does not only consider efficient inference models on single IoT devices, but also proposes a communication-efficient framework that can work in a distributed setting with multiple cooperating nodes. A huge chunk of our work focuses on optimizing communication across multiple nodes within the Deep Network models for various supervised learning tasks. These works do not consider multi-node scenarios and the communication footprint for distributed inference on multiple IoT devices. Work by [29] considers efficient distributed inference on memory-constrained Raspberry Pi and Odroid boards. Although, their distributed Network of Neural Network paradigm considers communication cost amongst student modules on IoT edge devices, it is limited as students nodes cannot be heterogeneous devices and there must always be a teacher node. Unlike in this work, our proposed methods are hardware agnostic as distributed inference and communication is managed within the Deep Network graph and all nodes have the same capabilities.

For reference to additional research efforts in which Machine Learning systems have been deployed at the network edge, see [41, 143, 188].

3.1.1 Enablers

Training Deep Networks on massive datasets is a non-trivial task as it requires significant computational resources and learning model parameters is difficult. Advancements in distributed training techniques [55, 104] and Graphic Processing Units

(GPUs) have inspired the use of complex models that can achieve higher accuracy at hefty computational cost. This has in turn, made applicability to Mist devices impractical owing to the resource-constrained nature of such devices. In many cases, such complex algorithms have been confined to centralized Cloud infrastructure supporting IoT end devices. Nonetheless, we are beginning to see a push to embed Deep Networks on end devices for various tasks. This development is inspired by the celebrated inference capabilities and development of enabling frameworks, architectures and platforms for the instantiation of Deep Networks in embedded Fog and Mist devices. [30, 213] present a variety of promising hardware and algorithmic techniques that can enable Deep Network-based In-network Processing. We provide a brief discussion of some enablers of this paradigm;

- **Model Compression and Acceleration:** Training and inference with Deep Networks can be computationally intensive and prohibitive for resource-constrained devices. Moreover, a large number of parameters requires memory space that may not be easily available in mobile, Fog and Mist devices. Works in literature that reduce Deep Networks model sizes include; HashedNets [43], Binarized Neural Networks (BNNs) [49], and network wide pruning [80]. These works have resulted in smaller, memory efficient network models that can be extended to embedded devices. [46] surveys various methods for model compression and acceleration. In their work, they review methods for parameter pruning and sharing, low-rank factorization, and use of transferred convolutional filters. Further work by[87, 167] achieve model compression from a model training perspective by using knowledge distillation that results in smaller Deep Networks. Work by [112] proposes a software accelerator for efficient execution of Deep Networks on mobile platforms, offering efficient resource control and optimization algorithms coupled with run-time optimization.

- **Hardware Evolution:** Apart from the use of software accelerators for CNN training and inference as in [112] and in-model acceleration techniques [4,46], design is evolving to accommodate Deep Networks on various hardware platforms via hardware acceleration. [8] proposes a hardware accelerator that enables low-power and low-latency CNNs. Their accelerator processes up to 128 input and 128 output feature maps per layer in a single pass on an FGPA platform. [42] focus on memory impact and show the effectiveness of their accelerator design by achieving 117.87x speedup and 21.08x more energy-efficiency on average compared to a 128-bit SIMD core clocked at 2GHz on CNN inference

tasks. [44] incorporate model compression and data gating in a re-configurable accelerator that achieves 35 frames per second and 0.0029 Multiply and Accumulate (MAC) for the AlexNet [108] model at 278 mW, and 0.7 frames per second and and 0.0035 MAC for VGG-16 [195] at 236 mW. [172] propose the use of a programmable processor based on spintronic devices which achieves 14.4x reduction in energy consumption and 20.4x reduction in Energy-Delay-Product (EDP) as compared to a CMOS baseline. Other methods proposed in literature include the development of tiny motes (very small processors) suited for on-device, latency-aware Deep Network training and inference [23, 150].

- **Datasets Availability:** For Deep Networks to perform inference, labelled or unlabeled datasets are required. Pre-processing data for Deep Networks on Cloud IoT, Fog and Mist devices poses major challenges. The existence of readily available training datasets for Deep Networks in these domains is a key enabler of various embedded Deep Network inference tasks. Popular datasets that have been employed to train Deep Networks for embedded Cloud IoT devices include; ILSVRC [183], PASCAL VOC [66], Mobile Context Image dataset (MCID) [118], CIFAR [107], and Floating Car Data (FCD) [144] datasets. [148] provide a comprehensive list of IoT datasets that can be used to train Deep Networks in this domain. Some Fog and Mist applications require domain-specific datasets, which may not be easily available at the time of model design. Nonetheless, synthetic data generators have proved useful in such cases.

- **Emerging CNN Architectures:** Object detection, image and video classification are crucial to Fog and Mist applications. CNNs have achieved state-of-the-art performance on processing visual data. Nonetheless, the instantiation of these coveted Deep Networks does not come without a cost. CNNs are computationally intensive, and from an embedded perspective, require memory and energy resources that are usually unavailable. To enable, the instantiation of CNNs for object detection tasks in embedded devices, several exciting works have been proposed in literature [93, 96, 101, 125, 222]. [101] demonstrate that their flattened layers achieve speeds two times faster than a conventional CNN layer during a feed forward pass. [96] proposed a CNN architecture, they called SqueezeNet. Their architecture allows for computationally efficient data-parallel distributed training and can be deployed on FPGAs. [93] provide a class of efficient CNN architectures that account for memory and accuracy trade-offs on object detection, image classification and geo-localization tasks in embedded

mobile and vision applications. [125] report impressive accuracies i.e., 74.3 % mean Average Precision (mAP) and 59 frames per second (FPS) on VOC 2007 dataset [66], for object detection tasks using a model suitable for real-time applications in embedded devices. Further work by [222] accelerates computation on CNNs while reducing storage and memory overhead via quantization of network parameters. Their architecture is suited for CNN inference on mobile devices.

3.1.2 Cooperative Deep Networks

3.1.2.1 Distributed Event Detection

Several works [5, 35, 223] have shown that Deep Networks are well-suited for event detection and processing tasks in emerging IoT settings. This is of key importance because these models can fuse and process raw temporal data from distributed sensors over long periods of time to provide accurate inference [223]. In [35], Long Short Term Memory (LSTM) networks are proposed for predictive maintenance on distributed turbofan machines. Their model is able to capture faulty conditions based on the history of the systems being monitored from large sets of sensor data. Work by [223] characterizes *complex events*, often made up of a series of *primitive events* and employ a Deep Network approach that is coupled with a state-based event detector. [5] propose RNN-based architecture in which sensor nodes communicate to solve distributed event detection tasks. Although these works offer an exploration of model hyper-parameters as they impact model performance, they fail to provide discussions on the communication overhead incurred in distributively detecting events from multiple sensor nodes.

3.1.2.2 Distributed Image Classification

Devices in IoT settings, (i.e., cameras, mobile robots) enable computer vision applications powered by Deep Networks. Recent works have proposed Deep Network-based algorithms for distributed, cooperative IoT settings in which visual data from multiple cameras is analyzed in real-time for various tasks namely, mobile object recognition [118], military coalition networks [181] and image classification [158, 210]. [158] propose an agent-based cooperative learning algorithm in which feature summary vectors are passed to neighboring agents to achieve a global image classification task. Their work assumes unlimited communication capacity and does not optimize for communication cost. In [118], an edge-mediated, collaborative learning framework for mobile object recognition is proposed. [181] focus on exploiting the hierarchical nature of

learned representations from Deep Networks to adaptively distribute computation over edge devices in coalition networks. [210] propose a Deep Network architecture instantiated over distributed computing hierarchies, consisting of the Cloud, edge devices and various sensors. Although these works are of immense promise, they explore distributed image classification in scenarios supported by edge infrastructure. Instead of typically reasoning about how to offload computation to less resource-constrained devices, it is more interesting to navigate ways to utilize existing resources on sensing nodes through network energy and communication cost optimization among other considerations.

3.1.2.3 Distributed Target Tracking

The ability of Deep Networks to cooperatively solve event detection and image classification tasks in distributed IoT settings enables complex emergent applications such as distributed target tracking. With distributed target tracking, cooperative Deep Networks can be used to estimate the position of moving targets, (i.e., people, objects, etc.) for monitoring, surveillance and or real time control applications. Several works [133, 145, 168] in literature leverage Deep Networks for distributed target tracking. In their work, [168] propose a real-time approach for tracking multiple objects using multiple distributed cameras. Their work acknowledges challenges associated with communication and node failure whilst processing streams of visual data across multiple distributed cameras. [145] propose an end-to-end learning approach for online multi-target tracking. Their approach uses a unified Recurrent Neural Network (RNN) that combines data association, state update and prediction of targets. Further work by [133] propose a CNN-based multi-target tracking algorithm for estimating high-quality trajectories. They demonstrate their approach by identifying tracked targets in video sequences for more than 80 % of their presence. Although these works provide promise for implementation of Deep Networks in cooperative, distributed target tracking settings, using Deep Networks for this task in realistic scenarios remains a non-trivial task fraught with many challenges. Key questions on highly-accurate automated feature extraction, model training, energy-efficient and communication-efficient distributed inference, remain unaddressed.

3.1.3 Communication in Deep Networks

3.1.3.1 Distributed Machine Learning

In the field of Machine Learning, efficiently training and optimizing Deep Networks is an issue of research importance. This has been a consequence of the growing need to train and instantiate these algorithms in scenarios plagued with constrained computational budgets, (i.e., mobile devices). To alleviate this constraint while aiding applicability to large-scale data scenarios, (i.e., IoT applications), Distributed Learning [169] has been widely explored. In distributed, cooperative IoT settings, Deep Networks employed on sensing devices for inference tasks are often trained in a decentralized fashion. High latency and low-bandwidth limitations in such settings introduces communication bottlenecks on training nodes. Such bottlenecks have inspired significant research efforts that address communication-efficiency in Distributed Learning [6, 216]. Although these methods are instrumental to low-cost Distributed Learning, they purely focus on providing model training methods for optimizing gradient-based updates rather than task-aware distributed inference.

3.1.3.2 Multi-agent Communication

Communication has been considered across multiple domains, i.e., Cooperative IoT settings, (e.g., Wireless Sensor Networks) [127] and Multi-agent Systems [69]. In such settings, most works employ pre-defined communication protocols instead of automatically learned communication. Research by [202] considers agents with limited capability operating in partially observable environments. In their approach, cooperative agents individually controlled by deep feed-forward networks can learn task-specific communication via back-propagation. Further works by [70] use Reinforcement Learning and centralized learning approaches with decentralized execution. In these models, end-to-end back-propagation of error signals across agents enables learning automated communication using Deep Networks. Although these approaches are elegant in design, their scalability is limited owing to the need for specialized connectivity structures and more computational power to compute multiple agent actions at each time-step at inference time.

3.2 Discussion

In this Chapter, we characterized Deep Network-based In-network Processing, a paradigm which applies Deep Learning approaches to resource constrained IoT set-

tings for data processing, learning objective functions, and inference on network nodes. We also positioned our research in the context of recent related work, discussing distributed event detection, distributed image classification, distributed target tracking, distributed machine learning, and multi-agent communication. In doing this, we offered systematic and comparative analyses of these works within the area of Deep Network-based In-network Processing.

Chapter 4

Communicating Recurrent Neural Network (*C-RNN*)

In this chapter we describe our novel Communicating Recurrent Neural Network *(C-RNN)* framework for automated, efficient inference and communication in distributed settings. Section 4.1 provides a high-level overview of our *C-RNN* architecture. Here we detail issues in Resource Constrained Systems our *C-RNN* framework overcomes whilst provide motivation for our architecture. In Section 4.2 we layout the system model of our framework. Here we use a UML diagram to show a visual representation of interacting components in our system. We then highlight our core Machine Learning (ML) network structure and introduce the concept of weight-sharing in Section 4.3. In Section 4.4 we discuss the training method for our approach and in Section 4.5 we describe our communication paradigm before concluding the chapter with a brief discussion in Section 4.6.

4.1 System Overview

Recently, we have seen the deployment of distributed Internet of Things devices shift from novelty to necessity. IoT environments in which these devices are deployed employ paradigms where embedded, inconspicuous and ambient intelligence gainfully interacts with humans to deliver valuable insights pertaining to environments being monitored. In such environments, multiple decentralized devices are imbued with heterogeneity, ubiquity, and sensing capabilities. These devices must sustainably sense, efficiently communicate, dynamically analyze and cooperatively deliver actuation and or insights pertaining to dynamic and physical phenomena they are monitoring. Traditionally, Cloud systems have been the backbone of IoT systems and continue to play a pivotal role in offering virtually unlimited storage, elasticity,

scalability, resource pooling and compute power to resource-constrained IoT devices in a centralized fashion. However, owing to real-time, latency, connectivity, and network and communication bandwidth requirements, edge-centric paradigms [117] are gaining traction both in research and industry. These paradigms are, in many cases, unfettered by third party intervention, and equip IoT devices with mobility support, location-awareness, real-time capabilities, reduced network bandwidth requirements and latency sensitivity. Moreover, they are often intermarried with Artificial Intelligence and Machine Learning tools to achieve automatic data reduction, on-device inference, in-built fault tolerance and energy-efficient communication without Cloud support. This has motivated a new wave of applications in which Deep Networks are implemented on embedded, resource-constrained IoT devices. Moreover, notable advancements in hardware [8, 42, 44, 112, 172], model compression and acceleration [43,46,49,112], datasets availability [66,107,118,148,184] and emerging Deep Network architectures for embedded inference [93,96,101,222] motivates the rapid adoption of Deep Networks in resource, constrained IoT settings.

Inspired by the successes of Deep Networks in achieving automated feature extraction, automatically learning communication protocols and enabling distributed, cooperative intelligence on embedded, resource-constrained IoT devices, we propose a Communicating Recurrent Neural Network (*C-RNN*) framework. Our framework is novel, scalable, data-driven and achieves efficient communication without the need for any expert designed, complex, pre-determined, specialized communication protocols. Our approach presents many opportunities for distributed, cooperative inference on embedded, resource-constrained IoT devices. More specifically, we make the following contributions:

- We propose a distributed, end-to-end trainable Deep Network framework that is purely data-driven and can solve distributed tasks cooperatively.
- We design a communicating Deep Network-based architecture that implements homologous and uni-directional Recurrent Neural Networks (RNNs). Our architecture learns dependencies in temporal data across multiple, cooperating nodes via latent-state communication and weight-sharing.
- We consider a scalable node topology instantiation method that defines node connections using an undirected network adjacency graph. Node connections can be arbitrary, user-defined or task-specific.

- We design a communication paradigm that enables distributed system nodes to solve tasks cooperatively. Our communication paradigm efficiently aggregates communicated data and maps latent states based on node connections.
- We show how our framework generalizes to multiple problem scenarios given its versatility and can perform inference on data with multiple modalities with minor modifications.

4.2 System Model

In this section we discuss key aspects of our end-to-end trainable system model. We also provide a high-level outline of our proposed *C-RNN* framework. We show the system model as a Neural Network graph that can be superimposed on a Wireless Sensor Network (WSN) graph. Unlike in multi-agent communication systems [69, 70, 202] where the focus is on the Neural Network graph, our work closely considers both the Wireless Sensor Network and Neural Network graphs.

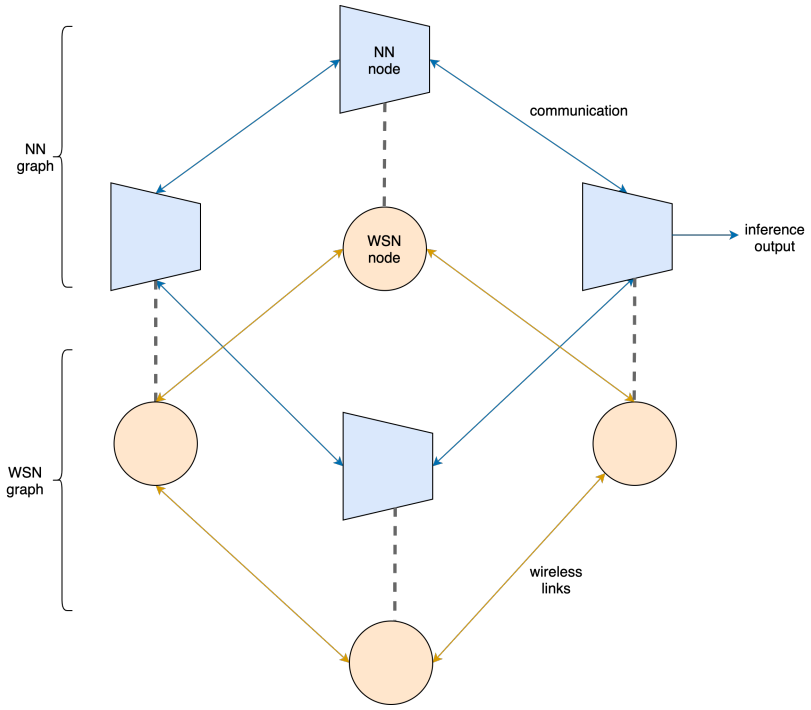


Figure 4.1: *C-RNN* framework network arrangement showing WSN and NN graphs.

Consider the high-level framework for our Communicating Recurrent Neural showing an arrangement of layers within the overall network, in Fig 4.1. The bottom WSN layer encases the WSN graph and is characterized with distributed sensing nodes with

a specified topological arrangement These nodes are tasked with monitoring dynamic phenomena in an environment. In top Neural Network layer, Neural Network nodes perform node communication and distributed inference on the objective task of the overall network.

A Neural Network can be mounted on an embedded, resource-constrained IoT device, which can be abstracted as a single Neural Network node in the system. This system abstraction can be augmented with a User Interface Layer which provides an interface to the system to specify node topology, define and initialize network parameters, set monitored events and define classification or prediction tasks. The overall system must achieve high accuracy on objective tasks whilst executing a communication and computationally efficient operation. The framework is designed to be robust to link failure as nodes in the Neural Network layer perform a data-driven computation which learns from data signals from the WSN layer. Moreover, the system allows for streamlined node scalability as adding additional nodes, means simply extending the Neural Network graph with an additional data input signal. In Fig 4.2, we look at the computation, communication, memory and energy dynamics for a single node within our system model.

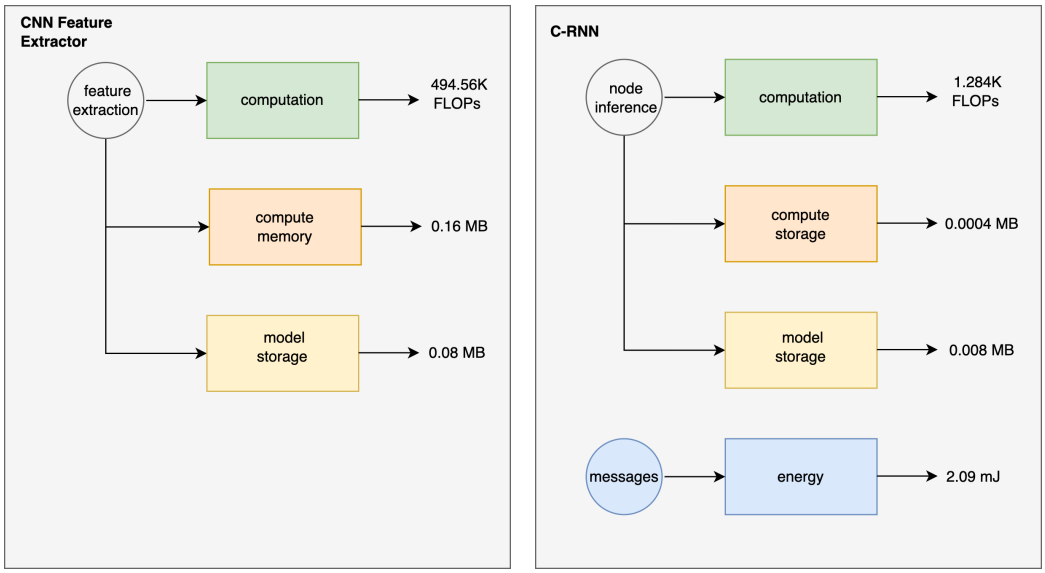


Figure 4.2: Characterization of resources on a single node within the system model. Values are estimated using the PyTorch platform.

The computation is estimated from the actual Multiply-Add operations, hence a single FLOP is considered to be half a Multiply-Add operation during inference. The compute memory is considered for inference in a single forward pass for a single image and estimated by the PyTorch platform. The model storage is estimated from a

summation of the model parameters and buffers multiplied by the datatype factor (in our case float32). Compute storage considers the memory footprint of storing weights and activations in single forward inference. For energy estimation, we consider both computation and memory accesses on a single node (refer to section 6.4.2.5). In our setting, each node concurrently communicates a single 1 byte message during a communication time-step. The packet size of the communicated message is constant and takes the form of a 1-dimensional tensor.

4.2.1 System UML Diagram

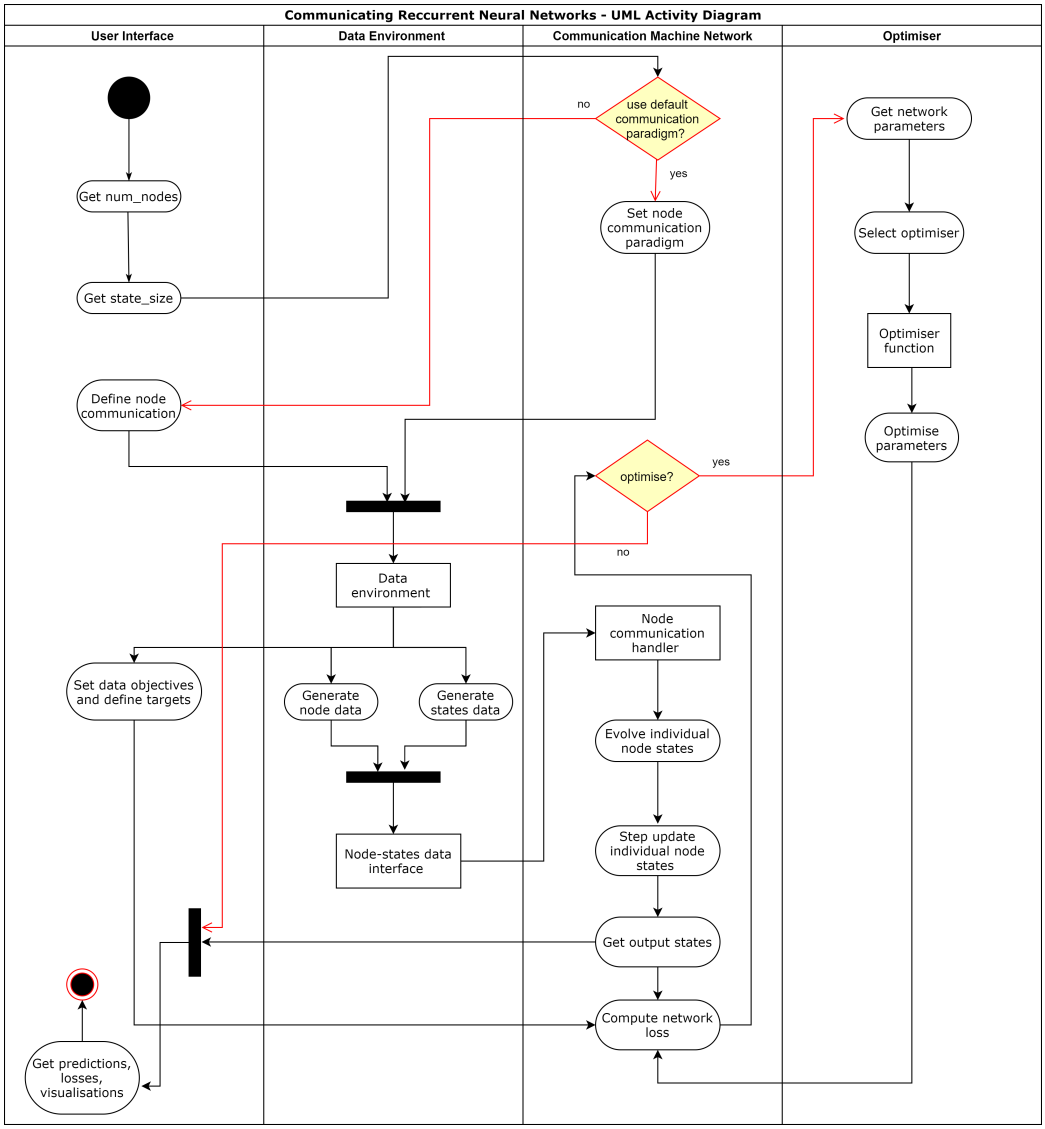


Figure 4.3: Communicating Recurrent Neural Network (*C-RNN*) UML Activity diagram.

Now that we have a general idea of the modules in our proposed system, we can visually present how components in the different modules interact in Figure 4.3.

4.3 Core Network Structure

4.3.1 Node Topology

Our proposed *C-RNN* architecture employs nodes observing unique data over defined periods of time, and at each processing step sharing their hidden states with neighboring nodes to ultimately learn the dependencies in the data and the observed function(s).

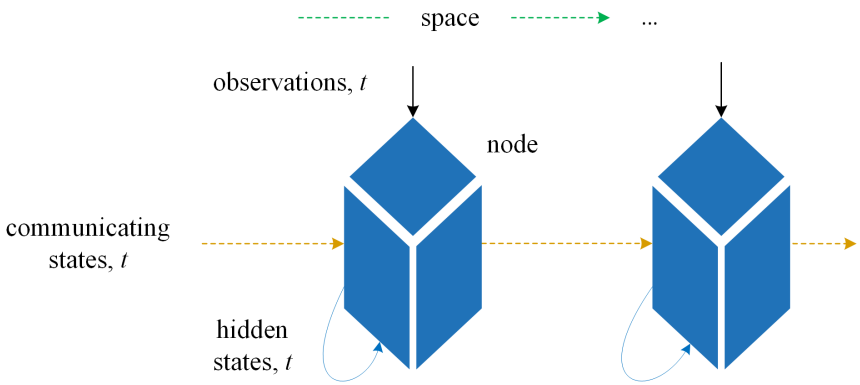


Figure 4.4: Nodes in a *C-RNN* architecture.

The nodes in the system are organized into a decentralized, distributed and cooperative network that allows them to collectively solve distributed tasks. The *C-RNN* architecture can be extended to edge devices, (i.e., cameras and sensors) cooperatively operating in a smart environment, (i.e., parking occupancy detection, surveillance). To achieve their intended tasks, distributed nodes must collect data from their environment, communicate and perform inference locally. Figure 4.4 shows a high-level arrangement of nodes in our architecture.

Our *C-RNN* architecture can be extended to an arbitrary number of nodes, N. We represent the topology with an undirected network adjacency graph $G = (\mathrm{V}, \xi)$ where $V = \{1,..., N\}$ are the nodes in the network and $\xi \subset V$ x V represents the presence or absence of a link between any two nodes $(i, j) \in \xi$. Based on the network graph G, each given node has a set of neighbors with which it can directly communicate.

The network graph allows for setting up node topologies with varying complexities. Figure 4.5 shows common topologies with decentralized and centralized arrangements.

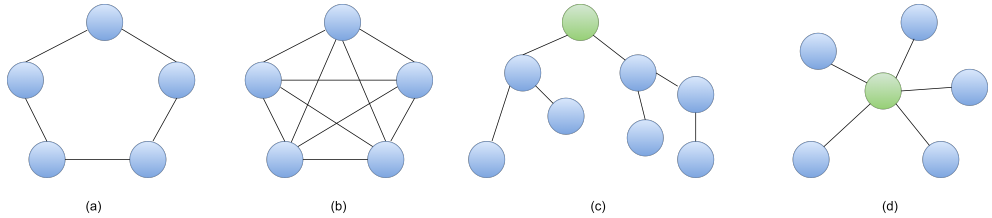


Figure 4.5: Decentralized (a,b) and centralized (c,d) node topologies. The different topologies shown are, (a) Ring topology, (b) Fully Connected Mesh topology, (c) Tree topology, and (d) Star topology. In (a) each node connects to exactly two other nodes whilst in (b), every node is connected to every other node in the network. In (c) a single 'root' node at the highest level of the tree connects to one or many nodes in levels below whilst in (d), every node in the network connects to a 'central node.'

A more detailed discussion on node connections is given in Section 4.5. The distributed node topology our model employs allows for self-organization. Resultantly, we notice emergent behaviors in which network nodes learn to discover and optimize communication patterns autonomously and devoid of any central coordination.

4.3.2 RNNs and Weight Sharing

We implement our *C-RNN* architecture using homologous and unidirectional RNNs with shared weights on each node. RNNs form a directed cycle along a sequence and treat input data like a sequence whilst performing the same tasks for all elements in that given sequence. This enables RNNs to successfully model temporal dynamics in data by considering network states in the previous timestep *t-1* at current timestep *t*. More specifically, given an input sequence of length T, $\mathbf{x} = (x_1, ..., x_T)$, a standard RNN maps this to a sequence of hidden vectors $\mathbf{s} = (s_1, ..., s_T)$ and produces output activations, $\mathbf{o} = (o_1, ..., o_T)$ through recursive computation as follows:

$$s_t = \sigma(W_{xs}x_t + s_{t-1}W_{ss} + b_s) \tag{3.1}$$

$$o_t = \sigma(W_{so}s_t + b_o) \tag{3.2}$$

where σ is an element-wise non-linear activation function (e.g. Sigmoid, Tanh or ReLU), W_{xs} is the input activation weight matrix, W_{ss} and W_{ds}, are the hidden

activation weight matrices, W_{so} is the output activation weight matrix, and b_s and b_o are bias vectors. Figure 4.6 shows a diagram of RNN computation over time.

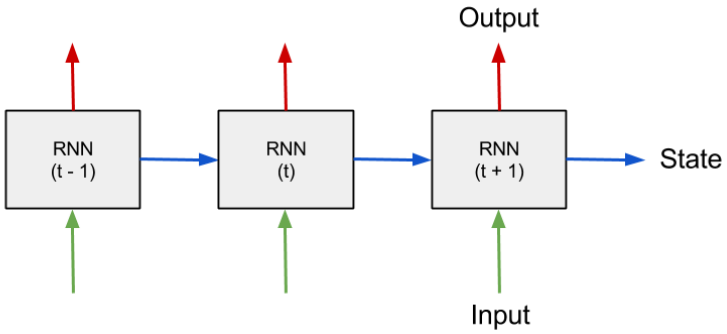


Figure 4.6: RNN computation over time. A different input sequence is fed into the network at each time-step, t-1, t and t+1, producing an output and a hidden state which is fed back into the network at the next time-step. The hidden state constitutes the 'memory' of the network.

This computation is useful for our *C-RNN* as it enables us to evolve the latent states of all the nodes in the system with communication over time. We unroll the RNN for a given number of communication steps, this can be determined by the number of nodes in the network or the structure of the connections of each network node. To evolve the communicated latent states, we feed the same input at communication timestep t. The principle here is to evolve the latent states, taking advantage of the temporal component built in the RNN to perform a sequence of communication steps within the network while holding the input. It is safe to establish that each RNN produces similar output as it instantiates the same network. This is achieved via weight sharing and the temporal component of the RNN in which the model's communication capability maps the latent states of all nodes observing their connections.

Our *C-RNN* architecture benefits from shared weights as parameters are significantly reduced across all nodes during learning. Resultantly, all nodes across the network share a single set of weights. The computation for the hidden states representing shared weights can be stated as:

$$s(x) = (\sigma({\mathbf{w}_0}^T x_1), ..., \sigma({\mathbf{w}_0}^T x_N)) \tag{3.3}$$

Where $\mathbf{w}_0$ is a weight vector that can be learned for inputs across all nodes in the network $\mathbf{x} = (x_1, ..., x_N)$ at any given time $t = (1,...,T)$. The resulting reduction in network parameters through weight sharing makes our network amenable to instantiation on resource-constrained IoT devices. Any node in the network will have sufficient

information for inference at the end of the inference pipeline, thereby guaranteeing a seamless distributed operation.

4.4 Model Training using BackPropagation Through Time and Space (BPTTS)

Recurrent Neural Networks (RNNs) can be trained through BackPropagation Through Time (BPTT) [182, 219]. This technique allows RNNs to iteratively update network weights whilst minimizing the error of the network's outputs with respect to some input. In turn, the network can learn feature representations of a domain task through hidden states propagated through the network over time.

Our *C-RNN* implements a simple extension of BPPT which trains the network through time (unrolled over t timesteps) and space (network nodes). As such, we implement BackPropagation Through Time and Space (BPTTS), enabling us to train our network on multiple nodes (> 2) with tied-weights. In Figure 4.7 we can observe that network nodes depend on latent states from their neighbors over time for recurrent computation. For instance, let ${}^{1}s_t$ be the current state for node *1* at time t. The computation of ${}^{1}s_t$ depends on that node's current input, ${}^{1}x_t$, its previous hidden state ${}^{1}s_{t-1}$ and node *2*'s hidden state communication ${}^{2}s_t^*$. The same principle can be applied to all network nodes based on node connections. We can then evaluate discrepancies between model predictions and ground truth using an objective function across all timesteps and network nodes. Thus, we can effectively learn the objective function across time and space using BPTTS.

4.5 Node Communication

4.5.1 Learning Communication

Traditional communication systems typically employ complex, hand-engineered protocols to achieve communication across network nodes. These systems face serious performance limitations owing to dynamic network conditions, change in network topologies and end-to-end communication system optimization. Deep Networks can learn patterns and representations in a data-driven fashion across network nodes. These patterns can be specific to node communication in which communication optimization can be achieved end-to-end. We leverage Deep Networks to design a deep learning-based communication paradigm in which nodes can learn to optimize their

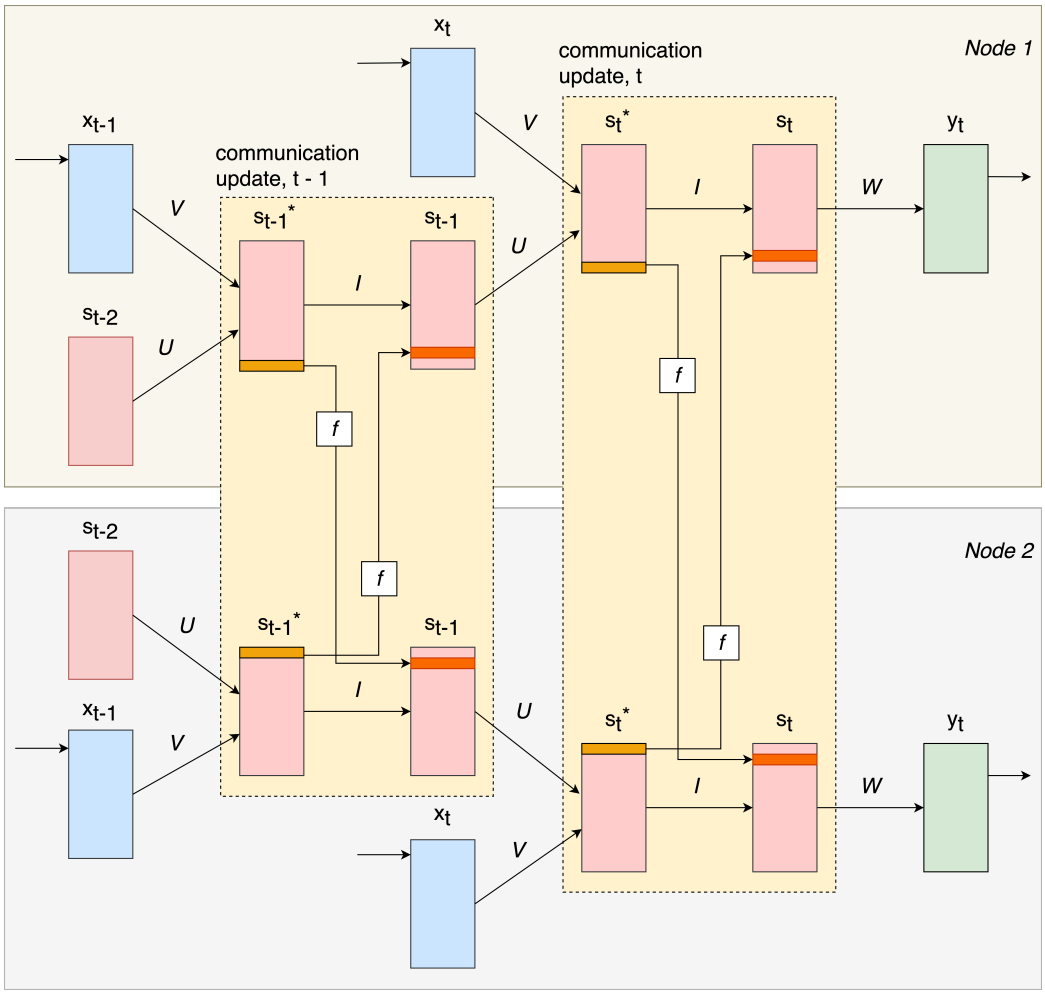


Figure 4.7: BackPropagation Through Time and Space in our *C-RNN* architecture with two nodes unrolled over two timesteps. At timestep t the model input is denoted by x_t, and output by y_t. The intermediate hidden state without node communication within the model, f at time t is indicated by s_t^* and s_t with node communication. We use U, V, W and I to depict weight matrices for input, hidden and output layers. I represents the identity matrix mapping the latent states. The weights are tied across network nodes.

communication by choosing when to communicate and how many messages to communicate. Consequently, our *C-RNN* model can learn a communication system and automatically optimize network parameters as they relate to the communication objective and task at hand. In this approach, nodes can learn to coordinate their communication behavior by intelligently discovering communication protocols within the Deep Network framework. Tuning hyper-parameters for the communication system does not require any communication expert knowledge. Practical implementation of deep learning based communication provides opportunities for further exploration

as the learned communication system must be trained to optimize across variable channel sizes and quantities. Learning communication enables modern communication technologies that harness deep learning successes in representation learning and pattern recognition.

4.5.2 Communication Paradigm

In our *C-RNN* model, multiple nodes connected via an arbitrary graph must communicate to distributively solve various tasks. The arrangement of nodes in a network is an important consideration in network design as it has direct impact on overall network resources, (i.e., energy and computational power). Resource-constrained nodes must typically communicate the least amount of information using the shortest paths to achieve an efficient operation [189].

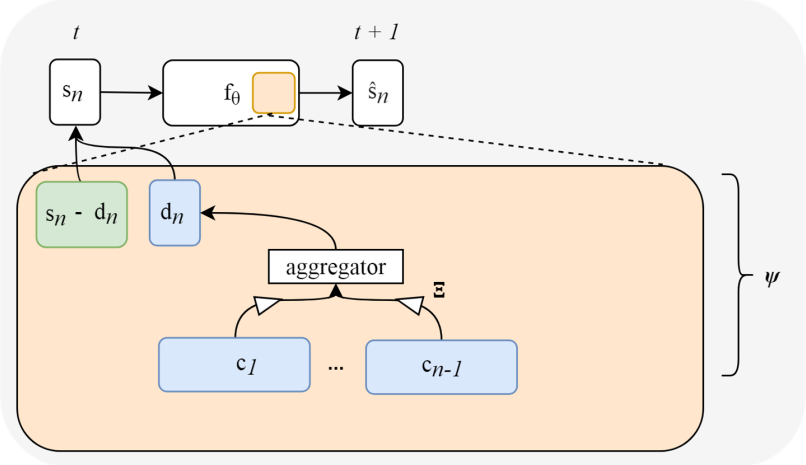


Figure 4.8: Overview of the communication handler ψ in the overall *C-RNN* architecture for the n^{th} node in the network. At communication time-step t, the node evolves a latent state through the RNN model f_θ. The latent state s_n is then taken by ψ as input to achieve communication. In the case above, a gating function Ξ, is applied to allow nodes to choose to communicate or not at each time-step t. The remaining nodes $1 \dots n-1$ pass their communication c_1 and c_{n-1} as specified by the adjacency graph G to the n^{th} node via an aggregation/pooling function. The aggregated communication to this node d_n, is concatenated to the n^{th} node's latent states s_n to produce updated states $\hat{s}_n$ at communication time-step *t+1*.

We design a novel communication handler that takes as input, at any given communication time-step t, the latent state space $\mathbf{s} = \{s_1, ..., s_n\}$ of all nodes in the network, where s_n is the latent state of the n^{th} node in the network. The communication handler ψ performs a mapping operation $\hat{\mathbf{s}} = \psi(\mathbf{s}, G)$ in which $\hat{\mathbf{s}} = \{\hat{s}_1, ..., \hat{s}_n\}$

is the original latent state updated with node communication as specified by the network graph, G. Note that this allows nodes to have arbitrary numbers of neighbors in the graph. Figure 4.8 shows in more detail how the mapping, $\hat{\mathbf{s}} = \psi(\mathbf{s})$ is achieved.

At each communication time-step t, the latent state s_n (of an arbitrary dimension z, corresponding to the number of neurons in the RNN) of the n^{th} node is passed to the communication handler ψ which aggregates communicated data from nodes which are connected via the graph G. A subset of the latent state for the n^{th} node $c_n \subset s_n$ is used to communicate partial state to neighbors. Rather than communicating the entire latent state of dimension z, we instead send a smaller state subset of size $v < z$ for reasons of bandwidth efficiency. The network thus needs to learn what state representation to send.

To further enhance efficiency, we explore whether nodes can learn to decide when to communicate or not. We do so through a simple gating function Ξ which controls if data is actually transmitted over the wireless channel or not. As a simple example of a gating function, using a RELU activation and only transmitting information if the output is non-zero will limit communication to strictly positive values, with negative or zero values not being sent. Without loss of generality, it is clear that this could be extended to arbitrary functions. Thus, this describes how nodes send data to one another.

Through the adjacency graph G and modulated by each node's gated output, a node must collect communicated states from its neighbors. As the network graph is arbitrary and the number of nodes is not predetermined, it is clear that the number of communicated states is variable. To handle this, an aggregation or pooling function is used to aggregate the variable input into a single vector d, also of size v. This aggregation function can be learned or pre-specified. In our implementation, where only a single message is communicated, we use $max()$ to aggregate the communication for each node. Mathematically, this is achieved as:

$$\hat{d} = \underset{1 \leq i \leq m}{max} \; d_{ij} \tag{3.4}$$

where m is the number of communicated messages resulting from the node connections for any given node, and d_{ij} is the j^{th} communication state and $\hat{d}$ is the j^{th} single aggregated communication. In other cases where the size of the communication for each node $v > 1$, a simple concatenation of the communication is done and mapped to the latent state of each node in the network. Using the concatenation aggregation method expands the size of the resulting vector, although it is useful in retaining all

the information across communicating nodes. Another aggregation method to consider is Average pooling. In Average pooling, aggregation is performed by taking the average of all communication states from all nodes in the network. Mathematically, we can present this as:

$$\hat{d} = \sum_{i=1}^{n} \frac{d_{ij}}{n} \tag{3.5}$$

where m is the number of communicated messages given as inputs to the aggregator, and d_{ij} is the j^{th} communication state and $\hat{d}$ is the j^{th} single aggregated communication.

After aggregation, the vector d, which represents the communicated and gated states from peers is then concatenated with the latent state s. Note that each node runs its own aggregation, based on received communicated states and is hence fully distributed. However, for maximizing training efficiency, the communication handler ψ is built inside a homologous and unidirectional RNN and performs the mapping $\hat{\mathbf{s}} = \psi(\mathbf{s})$ simultaneously for all nodes at each communication time-step t within the *C-RNN* model. This communication paradigm can be employed for a diverse set of distributed tasks with multiple, cooperating nodes.

4.6 Discussion

In this chapter, we have presented our *C-RNN* architecture. In doing this, we discussed our system model highlighting the arrangement of layers in our network showing a Neural Network layer superimposed on a WSN graph. We characterized minimal resources required for a single image inference on a single node, describing the packet form and size for a communicated message, compute and model memory and energy requirement for the model.

Our framework can model temporal dependencies in data being observed across multiple distributed nodes, automatically learn communication protocols and enable distributed, cooperative intelligence on embedded, resource-constrained IoT devices. We take advantage of the self-organizing nature of Neural Networks to instantiate RNNs on nodes observing unique data sequences and employing latent state communication. Our framework is end-to-end trainable and achieves efficient emergent inter-node communication without the need for any expert designed, complex, predetermined, specialized communication protocols. This presents many opportunities

for distributed, cooperative inference on embedded, resource-constrained IoT devices. Using shared, weights we can realize a smaller network with reduced parameters resulting in a computationally efficient, light-weight network capable of executing a resource-aware operation. Our framework provides necessary flexibility to dynamically add and remove nodes from the network.

In the following chapters, we employ this general framework, and in some cases, a variation thereof, to a diverse set of distributed tasks. We show that our approach enables distributed system nodes to solve tasks cooperatively whilst learning communication. Moreover, we show how our framework generalizes to multiple problem scenarios given its versatility and that it performs inference on data with multiple modalities with minor modifications.

Chapter 5

Learning Distributed Communication in Naive Settings

5.1 Introduction

In this chapter, we extend the Communicating RNN architecture *(C-RNN)* comprehensively discussed in the previous chapter to a naive setting, in which nodes observe unique data over defined periods of time and communicate to learn dependencies in sensor-like data. In doing this, nodes must maximize their shared utility whilst solving distributed tasks collectively via hidden-state communication.

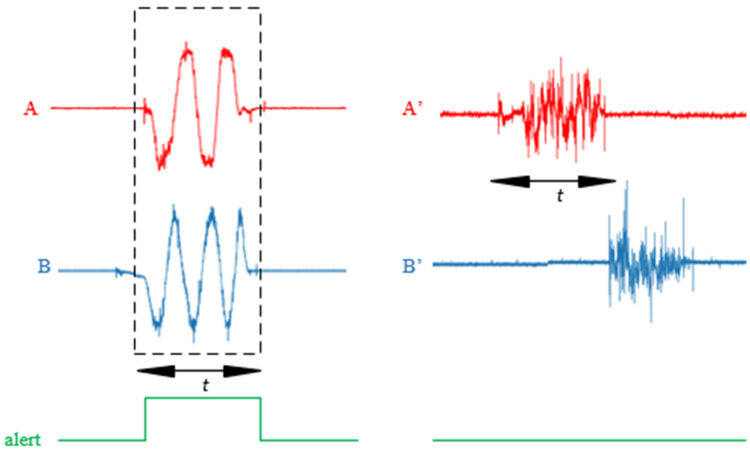


Figure 5.1: An illustration of event detection.

We leverage the knowledge that RNNs can achieve significant results in learning temporal dependencies in sequences of data to theoretically propose a distributed communicating RNN architecture that can be instantiated on distributed smart devices in smart environments. In such environments, hundreds of embedded sensors plagued by resources constraints must constantly monitor dynamic phenomena and

contribute near real-time actuation. That reality makes the task of developing novel architectures and models for automated inference and communication in such distributed settings salient.

Instead of trying to come up with algorithms and or complex communication protocols for distributed and cooperative inference in smart settings, we model distributed problems synonymous to event detection tasks and show that communication protocols can be discovered using Deep Networks. We show that our *C-RNN* architecture can solve sensing objectives in a distributed setting using automatically learned communication. Our framework can learn to maximize accuracy on sensed objectives whilst reducing computational overhead in the entire system. Figure 5.1 shows a scenario in which two sensors cooperatively detect an event within their data stream. In the scenario, two sets of sensor signals *A* and *B*, *A'* and *B'* depicting temporally varying phenomena are shown. An alert signal is generated for each set based on the occurrence of an episode of interest. In this chapter, we demonstrate that distributed nodes observing different signals characterized by dynamic phenomena can communicate - a capability which enables event detection applications in smart environments. Although, we demonstrate this on a small setup (2/3) nodes, our proposed framework sets out a new direction for automatically discovering communication protocols to solve tasks in distributed settings. This first step opens opportunities for exploration on node scalability, network robustness (in the case of faulty nodes), and further exploration on communication and computational efficiency. In summary our contributions in this chapter are as follows:

- We present a distributed, end-to-end trainable Deep Network framework that is purely data-driven and can solve distributed tasks cooperatively.
- We employ our communicating RNN architecture using homologous and unidirectional Recurrent Neural Networks (RNNs) and show that our architecture learns dependencies in temporal data across multiple, cooperating nodes via latent-state communication and weight-sharing.
- We formulate the problem of node communication by enabling communicating RNNs on 2/3 nodes to share their hidden state information with neighbors during training and inference.
- We evaluate our proposed system on sensor-like data and assess computational and predictive performance across various event detection sensed objectives; *Value*, *Range*, *Compare* tasks.

- We show that our framework automatically learns to discover communication patterns and solve distributed tasks cooperatively, thereby averting the need to design complex communication algorithms.

For the rest of the chapter, in Section 5.2, we provide a formulation of the problems considered in this chapter. In Section 5.3 we present our proposed approach discussing the system architecture and model training and inference. Section 5.4 presents our experimental setup and results whilst Section 5.5 provides a concluding discussion for the chapter.

5.2 Problem Setting

In Wireless Sensor Networks, many real-world events depict specific patterns detectable by applying Machine Learning [21]. Such patterns often characterize events of interest, e.g., impending disasters, emergency responses, and industrial machine failure, etc. Typically, to monitor an environment, sensor nodes are distributed over a large area. We consider a setting in which N nodes observe sensor-like data, communicate with their neighbors to collaboratively detect application-specific and or user defined events. To achieve this, sensor nodes operate based on pre-defined threshold values. When observed values exceed or go below the threshold values, an event is detected and a response, (e.g., an alarm or actuation procedure) is triggered. We model the problem of learning distributed communication across cooperating nodes to ultimately detect an event. Given two sets of temporal input sequences of observations $X = \{x_a, x_b\}$ where $x_a, x_b = (x_0, ..., x_T)$, node a takes sequence x_a as input and node b takes sequence x_b, over time T, we want to learn an objective function, f (x_a, x_b). The function maps to the presence or absence of an event of interest. In this case, the 2 nodes in the system have a different view of the scenario being monitored but must distributively communicate to make a collective decision. We evaluate our framework on 3 objective tasks; *Value*, *Range* and *Compare*. These objectives are discussed in more detail in Section 5.4.1. Figure 5.1 shows a high-level schema of the problem setting.

5.3 Our Approach: *C-RNN*

In this section, we present our approach and how we solve problem of discovering communication protocols using RNNs in a distributed setting.

5.3.1 System Architecture

Our decentralized approach to communication employs nodes observing unique data over defined periods of time, and at each processing step sharing their hidden states with neighboring nodes to ultimately learn the dependencies in the data and the observed function. This allows the event detection pipeline to automatically learn patterns in the data without any need for complex hand-designed heuristics to model the sensed objective. Figure 5.2 shows an arrangement of nodes in our architecture.

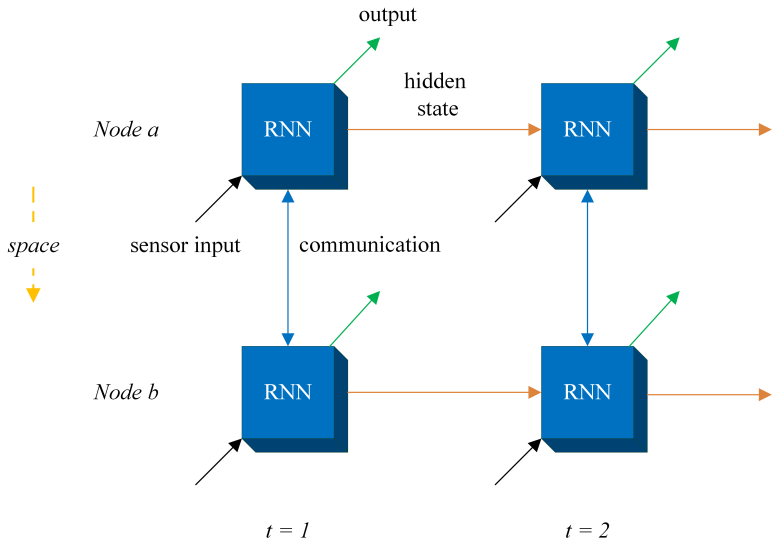


Figure 5.2: Nodes in a communicating RNN architecture. Color variations on arrows respectively represent sensor input, communication, hidden states and output across nodes.

In our *C-RNN* architecture, each node processes input over T time-steps. The node takes input observations at times $t = 0,...,T$. At each time-step, the node takes as input, its hidden state from the previous time-step and a communicating state from a communicating node. At the next time-step, the node then outputs the computation of its hidden state and a communicating state. The node uses ReLu activation across its layers. Nodes can represent a smart device onto which a communicating RNN is mounted. Our architecture implements homologous and unidirectional RNNs with shared weights and is instantiated to empirically assess performance in distributed communication. The system we propose is characterized by nodes denoted by n, observing some data X, unrolled over a defined number of time-steps t, has communicating states denoted by d, and hidden states denoted by s. It follows that for a network with N number of nodes, unrolled over T time-steps, $n = 1,...,N$ and t = 1,...,T. For distributed problems, it should always be the case that $N > 1$.

Our implementation extends the novel *C-RNN* architecture extensively discussed in Chapter 4 to a naive setting with distributed nodes observing sensor-like data.

5.3.2 Training and Inference

For our training process, we consider a case where $N = 2$, having nodes a and b. Given two sets of temporal input sequences of observations $X = \{x_a, x_b\}$ where $x_a, x_b = (x_0, ..., x_T)$, node a takes sequence x_a as input and node b takes sequence x_b, and computes its unit activations as high dimensional distributed representations of the observed data over time, giving the hidden states, $S_a = (s_0^a, ..., s_T^a)$ and $S_b = (s_0^b, ..., s_T^b)$ and communicating states $D_a = (d_0^a, ..., d_T^a)$ and $D_b = (d_0^b, ..., d_T^b)$. At each time-step t, a target value y_t is defined. The hidden states are updated using the equations:

$$s_t^a = \sigma(W_{xs}x_t^a + s_{t-1}^a W_{ss} + d_{t-1}^b W_{ds} + b_s^a) \tag{5.1}$$

$$s_t^b = \sigma(W_{xs}x_t^b + s_{t-1}^b W_{ss} + d_{t-1}^a W_{ds} + b_s^b) \tag{5.2}$$

The communicating states, d_a and d_b are cloned from the hidden states s_a and s_b respectively, using the operation, $(d_t^a,\ d_t^b) \leftarrow (s_t^a,\ s_t^b)$ at each time-step. Output predictions, $O_a = (o_0^a, ..., o_T^a)$ and $O_b = (o_0^b, ..., o_T^b)$ are computed as activations of the hidden states S_a and S_b recursively as defined by:

$$o_t^a = \sigma(W_{so}s_t^a + b_o^a) \tag{5.3}$$

$$o_t^b = \sigma(W_{so}s_t^b + b_o^b) \tag{5.4}$$

where σ is an element-wise non-linear function i.e., Rectified Linear Units (ReLU), W_{xs} is the input activation weight matrix, W_{ss} and W_{ds}, are the hidden activation weight matrices, W_{so} is the output activation weight matrix, and b_s and b_o are bias vectors. Equations 5.1 and 5.3 provide the update for the states in node a whereas Equations 5.2 and 5.4 update node b's states.

We can notice that Equations 5.1 and 5.2 show how the hidden states are shared across the different nodes at each time-step. We enable communication by effectively using shared weights across nodes a and b. The weight matrices of our system constitute the model parameters estimated using the stochastic gradient descent (SGD) algorithm and the Backpropagation through time algorithm [182] and *Adam* optimization [105]. We evaluate our model for z predictions simply in terms of Mean Squared Error (MSE) as in Equation 5.5:

$$MSE^N = \frac{1}{z}\sum_{i=1}^{z}(O_i - Y_i)^2 \tag{5.5}$$

where $Y = (y_0, ..., y_T)$ is the target output vector.

Our system enables N nodes parameterized by shared weights to share the outputs of their hidden states at each time-step t while employing our communication paradigm to solve for an objective function in a distributed setting. We take a streamlined approach to modelling problems in distributed settings, using simple functions and objectives to empirically assess the functionality and effectiveness of our proposed system. We utilize the inherent hidden state communicating nature of an RNN architecture to model distributed problems and show our approach's effectiveness without the need to implement any complex communication algorithm to meet defined objectives.

5.4 Evaluation

In this section we evaluate the efficacy of our architecture in employing hidden state communication using RNNs instantiated across distributed nodes.

5.4.1 Experimental Setup

Our experimental setup is designed to assess the computational efficiency and performance across sensed objectives, varying number of training samples, M and *space* (N). Our experimental setting is composed of a network with two (2-node case) and three (3-node case) nodes unrolled for $T = 2$ and $T = 3$, respectively and implemented in PyTorch. All layers are ReLu activated and parameters are optimized using SDG [182] with a learning rate of 0.0005. The model uses the default *Adam* [105] optimizer in Pytorch. We denote the number of training samples by M. The input is composed of sensor-like data sequences with a uniform distribution on the interval [0,1]. Parameter values are initialized with samples from a uniform distribution. In all experimental cases, communicating nodes instantiate RNN layers with hidden dimensions of size 8 and 144 parameters. We use a small setup (2/3 nodes) in order to demonstrate that nodes can learn to communicate with each other in a distributed setting.

5.4.1.1 Sensed objectives

Inspired by tasks in Sensor Networks, we model three objectives varying in application significance:

- **Value Task *(T1)***: For this objective, we investigate a case in which given sequence of sensor-values and pre-defined threshold values for two values x_t^a and x_t^b, the model should output a signal value, either *0* or *1* based on whether the observed value is above or below the threshold value. The nodes in our system each observe unique sets of data and are tasked with solving the function in a distributed manner. The target of the network y_t, is given by

$$y_t = \begin{cases} 1 & \text{if } x_t^a > p \quad AND \quad x_t^b < q \\ 0 & \text{otherwise.} \end{cases} \tag{5.6}$$

  where p and q are pre-defined threshold values.

- **Range Task *(T2)***: To assess model performance on this objective, we consider a sensing scheme in which given a sequence of sensor values and a pre-defined range, the model should be tasked with producing a signal value, either *0* or *1* when the incoming value falls within or outside the defined range. As in the scenarios above, the nodes in the system observe different sequences of data but must communicate to learn the objective function from the data. The target of the network y_t, is given by

$$y_t = \begin{cases} 1 & \text{if } (x_t^a < u \quad AND \quad x_t^b < u) \\ & OR \quad (x_t^a > v \quad AND \quad x_t^b > v) \\ 0 & \text{otherwise.} \end{cases} \tag{5.7}$$

  where u and v are pre-defined range values.

- **Compare Task *(T3)***: For the compare task, we look at a case in which given two values x_t^a and x_t^b in different sequences, the model should make a comparison and output a signal value either *0* or *1*, if either value is greater or less than the other, as per the user-defined objective. The two nodes in the system communicate to determine which node observes the smaller or larger value. The target of the network y_t, is given by

$$y_t = \begin{cases} 1 & \text{if } x_t^a > x_t^b \\ 0 & \text{otherwise.} \end{cases} \tag{5.8}$$

5.4.1.2 Time

We also assess our model's performance across time-steps. In our approach, we unroll our RNNs across varying time-steps, $t = 0,...,T$. Doing this enables us to heuristically determine the impact of feeding datasets of different sizes and the number of nodes, as we vary the time-steps of the RNNs in our system.

5.4.1.3 Space

In our proposed system, the number of nodes, $n = 1,...N$ correspond to *space*. We experiment on this measure to evaluate how the effectiveness of our communication paradigm scales with increasing nodes. For instance, in any system in which you have $N > 2$, the communicating strategy will obviously result in architectural complexity - a reality that can enable us to solve more complex objective functions and distributed problems while respecting a trade-off on model performance.

5.4.2 Results

We run experiments with different objective tasks and evaluate the performance in terms of training time and losses for a 2-node system. We also provide results for an ablation experiment on a 3-node system.

5.4.2.1 2-node System Case

Our results are promising as we are able to demonstrate the effectiveness of recurrent computation in a distributed sense for our defined objectives. The model finds optimal parameter values and achieves great performance on the defined tasks; *T1 (Value task)*, *T2 (Range task)* and *T3 (Compare task)*. We evaluate our architecture for training and predictive performance using the same number of parameters across all tasks. In Table 5.1, we summarize training losses of our model for varying samples in the 2-node system case.

	Losses (train)			
Task	$M = 10000$	$M = 15000$	$M = 25000$	$M = 50000$
T1	0.0716	0.0348	0.0138	0.0079
T2	0.1015	0.0681	0.0287	0.0184
T3	0.0478	0.0160	0.0068	0.0038

Table 5.1: Lowest training losses reached for objectives *T1, T2* and *T3* for $M =$ 10000, 15000, 25000 and 50000 in the 2-node case.

To investigate our model's training performance, we use training samples of sizes $M = 10000, 15000, 25000, 50000$. Notably, increasing M leads to a significant performance improvement for all tasks. Nonetheless, we observe that our model is able to satisfactorily learn the dependencies in task *T3* with relatively few samples, $M < 15000$. For this task, large samples seem to yield very insignificant performance results so far as learning is concerned. Not so surprisingly, tasks *T1* and *T2* require more samples, above $M > 25000$ for the model to learn the underlying pattern in the data. This is in fact a result of the objectives in tasks *T1* and *T2* being a slightly more complex to model.

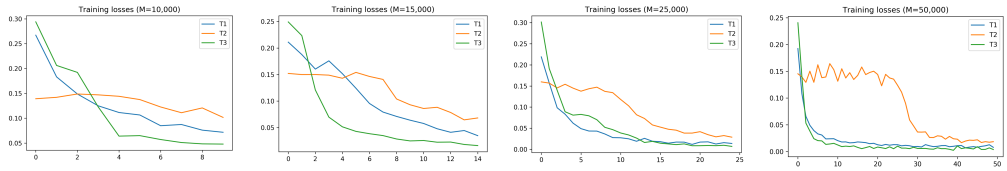


Figure 5.3: Training losses for all tasks *(T1, T2 and T3)* at every 1000 iterations, for training samples, $M = 10000, 15000, 25000, 50000$. The y-axis shows training losses for the model, whereas the x-axis represents training epochs for which the losses are calculated.

From Figure 5.3 it can be seen that the model takes longer to converge for task *T2* compared to other tasks - thereby proving the difficulty of training the model for this task. For all tasks, values of $M > 50000$, cause the model to converge, making it virtually impossible to gain any more information from the longer sequences. This is a key finding as the architecture design must be optimized and fit with the right model, considering number of parameters and model size in various sensing objectives.

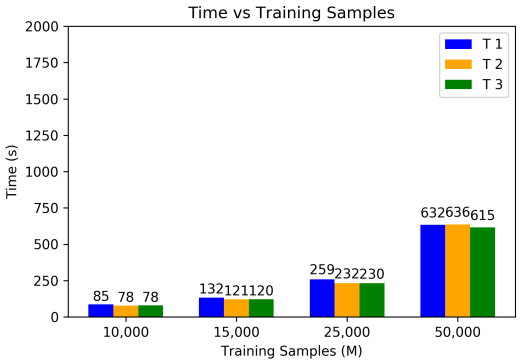


Figure 5.4: Training time for each task *T1, T2* and *T3* evaluated for $M = 10000, 15000, 25000, 50000$

In Figure 5.4, it can be seen that the amount of training time for all the tasks, when $M = 10000$ and $M = 15000$ is comparable, while losses show that training performance varies. Although training is harder for tasks with more complex dependencies, our results are applaudably encouraging as they prove that the nodes in our system communicate in their hidden states to solve distributed tasks. The model trains fairly fast and achieves highly accurate inferences for all tasks.

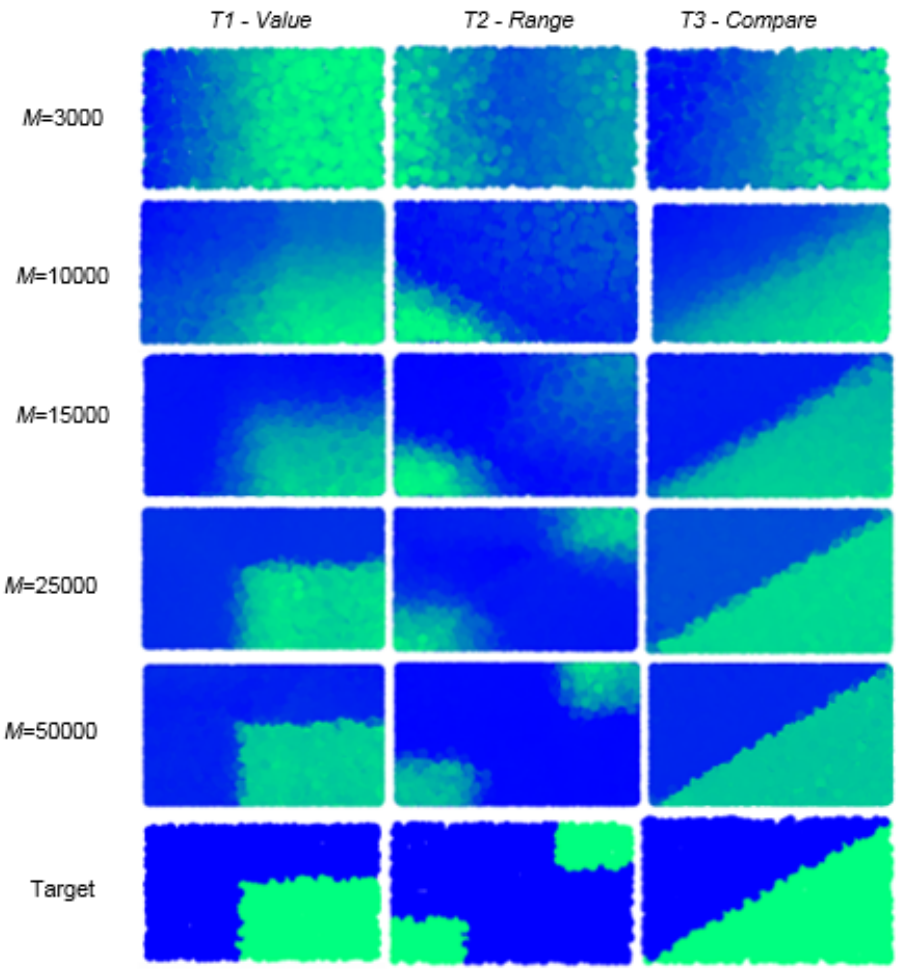


Figure 5.5: Scatter plots of targets and model predictions showing color-valued dependencies between x_t^a and x_t^b and predictions.

In Figure 5.5 we visualize model predictions and targets to show how well our model learns the objective function on the given sensed objectives *Value (T1)*, *Range (T2)* and *Compare (T3)* tasks.

5.4.2.2 3-node System Case

3-node case: To assess model performance across *space* as discussed in our Experimental Setup section, we also investigated on the impact of increasing the number of nodes in our system. Using a *C-RNN* setup with three (3) nodes unrolled for $T = 3$, we ran our model on tasks *T1, T2* and *T3*. We use the same number of hidden layers and parameters as our 2-node model. This setup takes in three input sequences x_a, x_b and x_c, with each sequence being observed by a different node. The sequences map to a single target objective, y. We test our architecture on tasks *T1*, *T2* and *T3* for $M = 10000$, 15000 and 25000. Figure 5.6 shows the communication paradigm for our 3-node system.

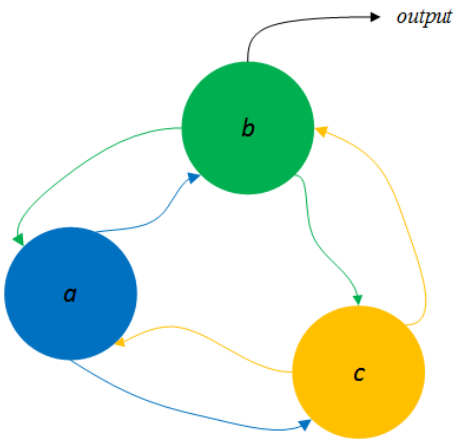


Figure 5.6: High level communication paradigm for the 3-node system. At each time-step, the nodes communicate based on their cyclic connection. The colored arrows depict node a, b and c's communication signals. There are multiple ways to establish connecting architectures for distributed models where $N > 2$. In this work, we chose the cyclic connections for their simplicity and intuitiveness. In a muti-node scenario, node connections can be defined in the form of a dynamic graph, thereby enabling users to select devices which must perform distributed inference in smart environments based on the communication architecture we present.

An analysis of our results shows that the 3-node model takes almost 1.5 times longer to train for $M = 10000$, 15000 and 25000 samples compared to the 2-node model. The model trains reasonably well for tasks *T1* and *T3*. The model fails to converge for task *T2* given $M = 10000$, 15000 and 25000. This could be because the model is too simple to learn the complex dependencies in the training sequences. To improve predictive performance, while avoiding overfitting for this task, we would have to increase the number of model parameters and training samples. This would,

nonetheless, make the model more complex and will require more training time. Moreover, it is worthwhile to note that the amount of hidden state communication scales drastically with increasing M and number of nodes, thereby creating need for more computation. In this case, an optimal trade-off must be achieved between accuracy and model complexity for the sensed objectives in the distributed tasks. Table 5.2 summarizes training losses for varying samples in the 3-node case.

	Losses (train)		
Task	$M = 10000$	$M = 15000$	$M = 25000$
T1	0.0922	0.0929	0.0955
T2	0.2368	0.2117	0.2153
T3	0.0930	0.1037	0.0876

Table 5.2: Lowest training losses reached for objectives *T1, T2* and *T3* for $M =$ 10000, 15000 and 25000 in the 3-node case.

5.5 Discussion

In this chapter, we proposed a distributed, communicating architecture based on state-of-the-art ML algorithms, precisely, RNNs. We presented results on varying sensed objectives, number of nodes, data samples and model training time. Our empirical evaluations show that we can solve objective tasks in a distributed fashion using our architecture through hidden state communication. We show that our *C-RNN* system can employ Deep Networks to discover communication patterns based on a sound theoretical framework that maximizes accuracy on the predictions of the sensed objective while reducing computation overhead in each node in the system.

The data-driven nature of our proposed distributed communicating architecture makes it scalable as adding a node is simply abstracted as adding a data source. This means that no architectural changes are required to the model. Since weights are tied, scaling nodes will not have any impact on the number of network parameters or the size of the model. As a generic architecture, our model can be extended to sensing devices of varying heterogeneity in smart environments, in which these devices observe unique sequences of data to cooperatively and distributively infer on an event of interest. For instance, in monitoring applications, (i.e., traffic and health monitoring), raw domain data is typically time-series and has multiple modalities. Distributed, communicating nodes can perform event detection and classification while using hidden-state communication to achieve resource optimization.

In Chapter 6 we investigate how distributed communication is affected in the case of faulty nodes within our architecture. We also provide a more in-depth analysis of how our *C-RNN* model generalizes to unseen network structures, scales to more nodes, optimizes for communication and manages computational complexity, albeit in a visual setting.

Chapter 6

Learning Distributed Communication in Visual Settings

6.1 Introduction

In Chapter 5, we implemented our proposed *C-RNN* architecture in a distributed naive setting in which nodes in a small setup (2/3), learned to discover communication patterns to cooperatively solve pre-defined sensed objectives. To this end, we showed that distributed nodes can share their hidden state information and learn distributed tasks without any specialized, hand-engineered communication protocol. The small setup we employed in Chapter 5 navigates node scalability in a limited way. In resource, constrained smart environments, scalability remains a pressing issue. This is because as the number of interconnected, embedded sensing devices grows in an IoT system, efficient resource management becomes a challenge. Moreover, meeting operational requirements in a larger network characterized by multitudinous nodes succumbs to interoperability and elasticity issues. Another pressing challenge in resource constrained IoT systems, is network robustness and fault tolerance.

The architecture we present in Chapter 5 assumes that distributed nodes will always participate in the inference pipeline and communicate to their neighbors at each communication timestep. In many realistic scenarios, sensing nodes are susceptible to permanent or intermittent failure due to multipath and or obstructions among other factors. In mission critical or latency sensitive applications, node failure could be disastrous. This warrants the need for newer approaches that guarantee self-organization, reliability and fault tolerance for embedded sensing devices operating at the network edge as locating failures and enabling sensing reliability under dynamic conditions is extremely challenging [193].

In distributed, cooperative IoT settings, embedded sensing devices must communicate in a resource-aware fashion whilst achieving a diverse set of tasks, (i.e., event detection, image classification, etc.). The goal is for the IoT perform data collection, processing, analysis and inference to deliver key insights. Current IoT architectures are skewed in favor of cloud-centric analytics, with the widespread sensors being 'dumb' and relatively limited in capability. In part, this is a natural consequence of the need for collaborative decision making, e.g., that sensor readings cannot be considered in isolation but must be analyzed as a collective spatio-temporal series. Instead, we employ a decentralized, edge-centric paradigm in which cooperating nodes have intelligence to make decisions locally without the need for Cloud support. For instance, in distributed, cooperative scene monitoring, multiple sensing devices, (e.g., cameras) must observe part of the environment and video data from all devices must be fused to reach a collective insight. Transmitting individual sensing device data to a central node for processing is not scalable owing to bandwidth constraints. Moreover, the approach is susceptible to overall network failure in the event of a failing central node.

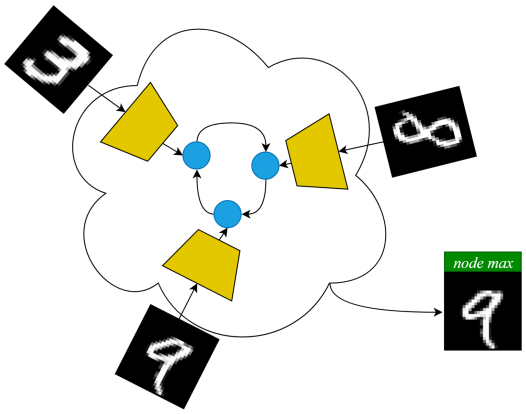


Figure 6.1: High level illustration for distributed MNIST classification. Multiple network nodes observe unique MNIST handwritten digits and cooperatively solve a distributed task - finding the *max* in this case. In this work we show that nodes can communicate through their latent states to achieve this goal.

Motivated by this, in this Chapter, we propose a distributed, cooperative approach in which sensing devices can perform local computations on observed visual data and cooperate with neighboring devices to perform image classification, recognition or detection tasks. In the previous chapter, we explored the efficacy of our proposed *C-RNN* architecture on a naive setting, with sensor-like data. In order to perform inference using visual data, we make significant architectural changes to

enable feature extraction and make further optimizations for computational and communication efficiency. We consider a hybrid scheme which combines CNNs and RNNs. Recently, it has been shown that hybrid architectures in which CNNs are employed for feature extraction and RNNs for temporal modelling and prediction achieve better results when compared to other architectures on many tasks [61, 191, 217, 236]. To achieve distributed image classification tasks, devices must communicate efficiently in a manner that respects constraints on their battery energy and bandwidth budgets. Handcrafted protocols and algorithms can be used to achieve these goals but are tightly coupled to the particular problem and wireless network topology.

We extensively explore the problem of observing visual data to solve distributed tasks collectively, as illustrated in Figure 6.1. Our approach does not require any specialized communication protocols and automatically learns to minimize communication and handle link failure, whilst achieving high accuracies on defined objectives. In summary our contributions in this chapter are as follows:

- We propose a novel, data-driven and communication-efficient Communicating Convolutional Recurrent Neural Network (C^2RNN) framework for distributed inference on visual tasks.

- We integrate feature extractors with our *C-RNN* framework into an end-to-end trainable hybrid architecture for distributed image classification.

- We employ a scalable node communication paradigm to enable nodes to share their hidden state information with neighbors during training and inference. Our node communication paradigm presents several opportunities for optimization, (i.e., communication channel capacity, communication loss.)

- We evaluate our proposed system on visual data and assess model convergence, node scalability, communication and computational cost on distributed MNIST classification *max* and *max-parity* tasks. Our proposed C^2RNN architecture automatically learns to discover and optimize communication whilst achieving high accuracy on objective tasks. Our model generalizes to unseen network structures and is robust to node failure.

- We show that our approach is computationally and communication-efficient when compared to *centralized, semi centralized, tree-based* and *energy-unaware distributed* C^2RNN baselines.

For the rest of the chapter, in Section 6.2, we introduce a problem setting motivating our proposed framework. In Section 6.3 we provide an overview of our system architecture, feature extraction process, node communication and model training and inference. Section 6.4 presents our experimental setup and results whilst Section 6.5 provides a concluding discussion for the chapter.

6.2 Problem Setting

We consider a scenario in which N nodes observe visual data and must communicate with neighboring nodes over potentially lossy wireless links to solve a given image classification task collectively. Our scenario is motivated by surveillance and computer vision-based applications in the IoT. In such applications, widely distributed cameras are deployed, and insights must be derived with minimal processing effort owing to resource constraints on the devices. To aid scalability whilst avoiding bandwidth constraints associated with performing analytics on a centralized Cloud infrastructure, each camera must perform local inference on the observed data and communicate with other devices in the network to achieve a global inference task. In our approach, we explore the problem of whether each node in the network can observe a randomized handwritten MNIST digit, perform feature extraction, then pass the features as input to an RNN architecture that can communicate a subset of latent states to neighboring nodes to solve various tasks. In this problem setting, tasks include detecting which node is observing the *max* MNIST digit based on automatically discovered communication protocols. Further to this, we also consider a distributed *parity* problem in which the network must produce a signal value based on the *parity* of the max of the MNIST digits being observed by all nodes in the network.

6.3 Our Approach: C^2RNN

In this section, we provide an overview of our proposed C^2RNN framework for distributed image classification. In doing this, we discuss how network nodes observe visual data then achieve feature extraction, communication, training and inference distributively.

6.3.1 System Architecture

Our C^2RNN architecture models distributed nodes observing visual MNIST handwritten digits and collectively solving distributed tasks based on each node's unique

observations and automatically discovered communication patterns. System nodes are organized in a decentralized, distributed and cooperative network. Over a sampling time-step, distributed nodes perform sensing and feature extraction before executing a series of communication time-steps to share their latent states with neighboring nodes, with respect to the node topology of the network (see. Section 4.3.1). Figure 6.2 provides an overview of our C^2RNN architecture.

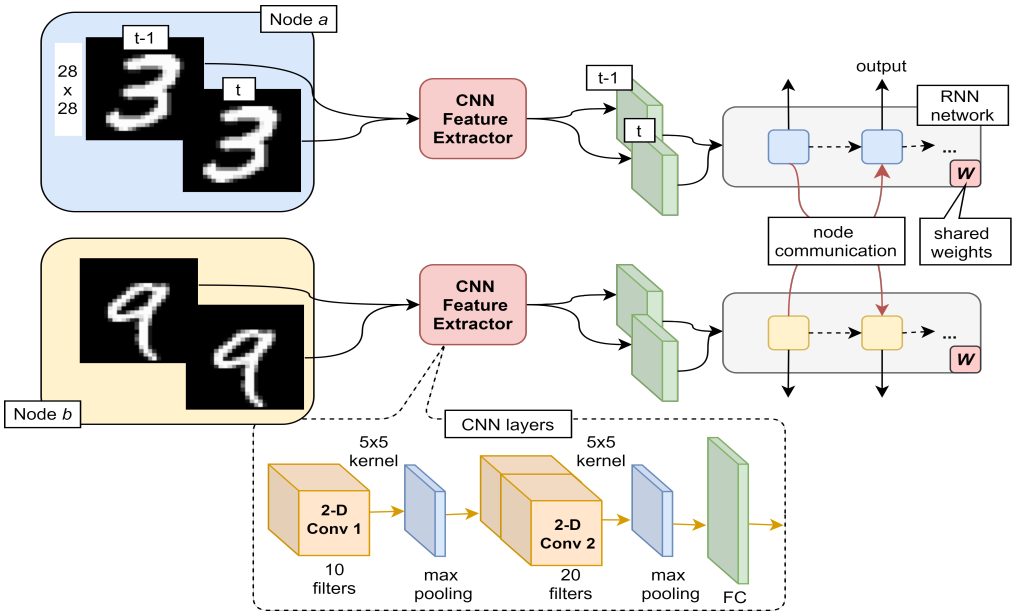


Figure 6.2: Overview of C^2RNN architecture, showing two nodes interacting over two sampling time-steps. At any given time-step t, each node observes visual data (in our case a single MNIST handwritten digit) then passes it through a CNN. The resulting image features are passed as input into a communicating RNN that shares a subset of its latent states with its neighbor. Each node evolves its own latent states and an output but has identical weights. The temporal component built in the RNN is used to evolve the latent states via recurrent computation. A sequence of communication steps is performed whilst holding the input at a given sampling time-step. Consequently, the sampling time-step is slower than the communication time-step.

6.3.2 Feature Extraction

To achieve distributed image classification, system nodes must first extract relevant features from input images before passing these features as input into the *C-RNN* framework for inference. Achieving efficient feature extraction under resource constraints is paramount as convolutional layers are typically time consuming and computationally intensive whilst fully connected layers are characterized by massive numbers

of network parameters [222]. We design a distributed CNN-based feature extractor that encodes the image into a reduced set of features. At each sampling time-step t and for each node, the feature extractor takes as input a 28 x 28 image in gray-scale then passes it into two 2-D ReLu-activated convolutional layers with max pooling for down sampling and dropout [89]. The first convolutional layer filters the image with 10 kernels of size 5 x 5. The second convolutional layer has 20 kernels of size 5 x 5. The resulting features are then passed into a fully connected layer with f_{in} units. The number of units f_{in} for the fully connected layer corresponds to the number of features that are input to the *C-RNN* framework. The weights of the feature extractor are initialized from a pre-trained MNIST handwritten digits classifier. The feature extractors across all the nodes in the network are identical in architecture and have the same number of weights.

6.3.3 Node Communication

In order to cooperatively solve tasks in a resource constrained distributed setting, network nodes must communicate optimally. In such a setting, all nodes have similar computational capabilities and must efficiently coordinate to contribute to an overall objective. Typically, communication protocols are pre-determined and hand-engineered to suit specific applications. As a result, this approach suffers from a scalability standpoint. Moreover, limited versatility makes it hard to model dynamic phenomena in way that respects resource constraints on network nodes.

In our system, distributed cooperating node instantiate a homologous and uni-directional RNN which takes as input image features and communicate latent state representations of these features to neighboring nodes based on the network topology. As in [202], what, how much and when nodes communicate is not specified a-priori. The goal is to only communicate information necessary to learn objective tasks with high predictive accuracy. To achieve this, we employ a communication loss, as given in Section 6.3.4. Network nodes automatically discover and optimize communication protocols with respect to the task at hand. Our node communication paradigm implements a communication handler ψ as presented in Figure 4.8, Section 4.5. In turn, our system is applicable to a wide range of distributed tasks employing Deep Networks.

6.3.4 Training and Inference

As in [5,210], the nodes in our network distributively observe unique sequences of data and must collectively perform an objective task. Let x_n be the n^{th} node's sequence of input features. At each sampling time-step t, input features, $\mathbf{x} = \{x_1, ..., x_n\}$ are passed from distributed feature extractors attached to each node. At each given sampling time-step t, each node evolves its individual latent states with a series of updates being performed iteratively as:

$$\hat{h}_n^t = f_\theta(x_n, \hat{s}_n^t) \tag{6.1}$$

where $\hat{h}_n^t$ is the model updated latent state of the n^{th} node in the network. The communication handler ψ performs the mapping $\hat{s}_n^t = \psi(s_n^t)$, resulting in the latent states, $\hat{s}_n^t$ with node communication. The latent states, $\hat{s}_n^t$ are then passed into the model f_θ together with the input features x_n (see Figure 4.8, Section 4.5). The model f_θ takes the form of a fully connected, lightweight RNN with model parameters and biases θ. The model's parameters are tied amongst all the nodes in the network and optimized using Adam optimizer [105]. Using tied weights significantly reduces the number of network parameters and enables our model to train faster. This also ensures that all nodes in the network are identical in operation, i.e., they all run the same network. The layers in f_θ are activated using a non-linearity, ReLu. At each sampling time-step t, each node computes an output $\hat{y}_n^t$ as given by:

+

$$\hat{y}_n^t = \sigma(\hat{h}_n^t) \tag{6.2}$$

where σ is a non-linearity. We evaluate our model with joint optimization considering both model accuracy and communication cost. For the network's accuracy loss, we use Mean Squared Error loss (MSE) as in Equation 6.3:

$$L_{acc}(\hat{y}, y; \theta) = \frac{1}{z} \sum_{i=1}^{z} (\hat{y}_i - y_i)^2 \tag{6.3}$$

where $\hat{y}$ is the overall network prediction and y is the network target. For joint optimization, we implement a combined loss $L_{combined}$ that enables our network to learn both the objective tasks and optimize for node communication:

$$L_{comm}(\hat{a}, a; \theta) = \frac{1}{z} \sum_{i=1}^{z} (\hat{a}_i - a_i)^2 \tag{6.4}$$

$$L_{combined} = w_{acc}\, L_{acc} + w_{comm}\, L_{comm} \tag{6.5}$$

In Equation 6.4, $\hat{a}$ is the number of messages communicated across the nodes at inference time and a is the target number of communicated messages for the network based on a communication budget. In Equation 6.5, the combined loss, $L_{combined}$ for each sample passed to the network during training with weights w_{acc} and w_{comm} is given. Our distributed C^2RNN model is end-to-end trainable. A single network is learned across all nodes, although each node evolves its own latent states based on unique observations.

For inference, a single target y_t is defined for each input sample, x_t at any given sampling time-step t. The nodes in the network communicate based on the connections defined in the network graph G. At the end of the inference pipeline, any node in the network must have a unified view of the environment to perform inference on the objective tasks based on the communicate information from other nodes in the network. In [210] a local aggregator is used to determine if a combined summary of information from all the end devices is sufficient for the classification task. Our approach does not employ any central coordination for the nodes thereby enabling any node in the network to have sufficient information for inference at the end of the inference pipeline. This guarantees a seamless distributed operation based on automatically learned communication patterns. It is worthwhile to note that model inference is dependent on the node connections in the network and must be modelled based on the objective task.

6.4 Evaluation

In this section, we evaluate the efficacy of our C^2RNN model quantitatively using models trained on the popular MNIST handwritten digits dataset [114]. In our experiments, we quantitatively assess model convergence, node scalability, and computation-cost and communication-cost on the *max and max-parity* tasks.

6.4.1 Experimental Setup

To assess whether our proposed C^2RNN model can learn to communicate, we consider a setup where we have a varying number of network nodes, N between 2 and 40. In our model, parameters of the feature extractors on each node are initialized with weights from a pre-trained MNIST digits classifiers. The classifiers are pre-trained over 4 epochs and use a training batch size set to 64. The classifiers have parameters optimized using SGD [182] with a learning rate of 0.01 and a momentum of 0.5. The size of FC units is varied based on the the number of input features f_{in}, passed to

the *C-RNN* model. The *C-RNN* model is trained with $f_{in} = 10$ and 25. Our setup uses [105] for optimization with a learning rate $l_r = 0.005$. The *C-RNN* model has 4 hidden layers, the first two with 50 units and the other 2 with 8 units. Table 6.1 highlights hyper-parameter choices.

Description	**Quality/Quantity**
Platform	PyTorch
Pre-trained CNN layers	$l_r = 0.01$, $momentum = 0.5$
CNN training	batch size = 64, epochs = 4
Input features	$f_{in} = 10$, 25
RNN layers	$l_r = 0.005$, act = Linear, Relu
Weights	Shared
Optimizer	*Adam*
Training method	SDG
Loss	L2
Model parameters	$\alpha = 1284$, 2169
Number of nodes	$N = 2$ to 40

Table 6.1: Model hyper-parameters.

The network parameters (which vary by f_{in}) are initialized with samples from a uniform distribution. Depending on the size of f_{in}, the number of parameters in the RNN α is either 1284 ($f_{in} = 10$) or 2169 ($f_{in} = 25$), making it amenable to deployment on low-end microcontrollers with only a few kbytes of memory. Each node is trained on randomly shuffled training samples $M_{train} = \{m_1, ..., m_i\}$ from the MNIST dataset, where m_i is the i^{th} training sample. We evaluate our model in terms of Root Mean Squared Error (RMSE). Further to this, we employ an error index similar to the measure used in [35] defining our model's mean percentage error. This % error augments RMSE by estimating the impact of the error made on each sample at inference time as RMSE only provides a generic error estimation based on mean 'distance' between model predictions and targets. The error rate is calculated as:

$$Error\ \% = \frac{1}{z} \sum_{i=0}^{z} \frac{|\hat{y}_i - y_i|}{y_i} \tag{6.6}$$

where y_i, $\hat{y}_i$ and z are same as values defined in Equation 6.3.

Using RMSE and error percentages, we are able to assess our model's predictive performance at inference time continuously by learning how much the model 'misses' the targets on both the *max* and the *max-parity* tasks.

6.4.2 Results

We provide empirical and systematic analyses of model convergence, node scalability, and computation-cost and communication-cost on distributed MNIST *max* and *max-parity* tasks. We also perform comparisons of our proposed C^2RNN framework with other methods.

6.4.2.1 Distributed *Max* and *Max Parity* Tasks

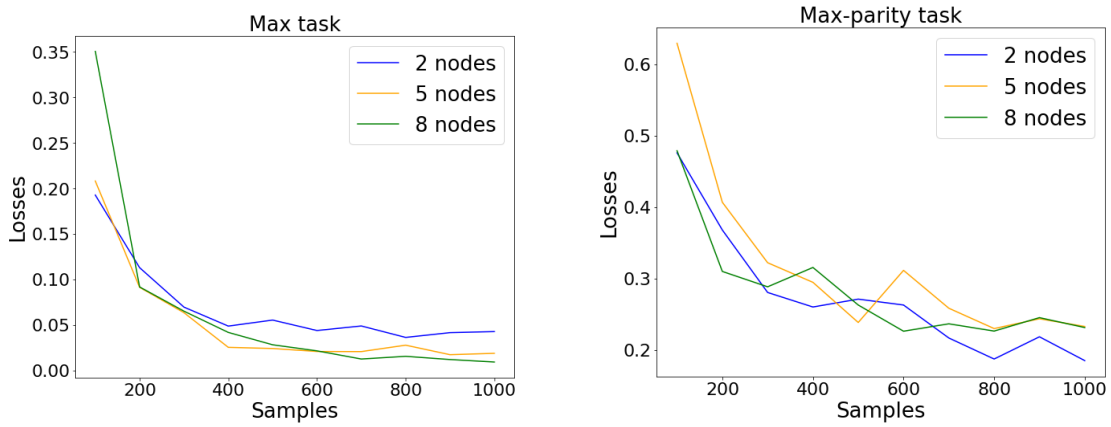
First, we consider a *max* task setup in which nodes must learn the maximum MNIST digit being observed across all nodes. Further to this, we consider a parity problem (*max-parity* task) in which distributed nodes must first cooperatively work out the maximum MNIST digit observed, then determine parity of that digit. All the nodes in the network show convergence (with $M_{train} = 1000$, $M_{test} = 500$) on the overall objective and achieve low error percentages on test data.

		max		*max-parity*	
N	α / f_{in}	Error %	RMSE	Error %	RMSE
2	1284 / 10	1.03 (±0.021)	0.195 (±0.019)	2.24 (±0.172)	0.454 (±0.017)
	2169 / 25	0.87 (±0.013)	0.190 (±0.019)	1.46 (±0.312)	0.421 (±0.035)
5	1284 / 10	0.18 (±0.009)	0.150 (±0.011)	1.79 (±0.145)	0.461 (±0.012)
	2169 / 25	0.17 (±0.009)	0.156 (±0.008)	2.40 (±0.189)	0.488 (±0.005)
8	1284 / 10	0.11 (±0.009)	0.113 (±0.007)	1.83 (±0.240)	0.456 (±0.006)
	2169 / 25	0.09 (±0.009)	0.104 (±0.006)	1.74 (±0.222)	0.462 (±0.018)

Table 6.2: Summary of model performance results for $N = 2, 5, 8$, with communication channel size $v = 1$ and $\alpha = 1284, 2169$ for $f_{in} = 10, 25$ respectively. We provide the standard deviation over 5 runs.

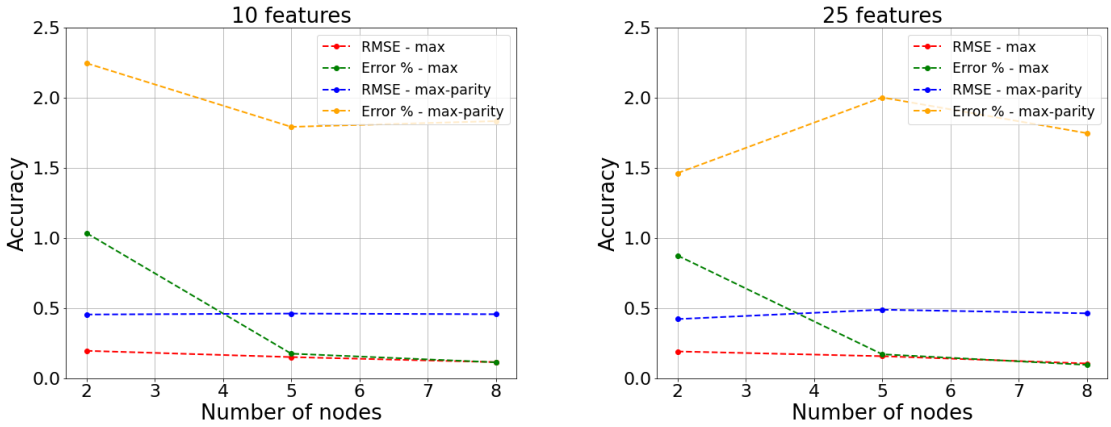
Table 6.2 provides a summary of results showing model performance on the distributed node *max* and *max-parity* tasks as the number of nodes (N) and number of image features f_{in} are varied. We can notice that the model is stable as, on average, C^2RNN has a standard deviation of 0.011 on RMSE for the *max task* and 0.015 for the *max-parity* task for $N = 2, 5, 8$.

Our model achieves the lowest error percentages when $N = 8$ and RMSE improves with increasing nodes. This shows that our model can scale while retaining high inference accuracy on given tasks. It is not surprising that the *max-parity* task is much harder to learn compared to the *max* task. Efficiently learning *parity* in distributed systems is typically difficult. The network nodes learn to integrate information over



(a) (b)

Figure 6.3: Training losses for (a) *max* and (b) *max-parity* tasks for $M_{train} = 1000$.



(a) (b)

Figure 6.4: RMSE and error percentages for *max* and *max-parity* tasks for $M_{train} = 1000$ with (a) $f_{in} = 10$ and (b) $f_{in} = 25$.

multiple time-steps ($T = 2, 5, 8$) via recurrent computation. For both the *(max and max parity)* tasks, the error percentages are slightly higher with $f_{in} = 10$, but this does not drastically affect model performance. We also compare RMSE on training data and RMSE on test data. A significantly lesser train RMSE compared to test RMSE means a model is over-fitting, whilst a greater means a model under-fits. Our model neither under-fits nor over-fits. Our results illustrate our model's ability to instantiate latent state communication with a *C-RNN* model to solve distributed objectives scalably. Figures 6.3 and 6.4 show training losses, and model performance (RMSE and error percentages) for $N = 2, 5, 8$.

6.4.2.2 Model Generalization

The success of any Neural Network model heavily depends on its ability to generalize. This remains a difficult challenge for most Neural Network architectures. To show the effectiveness of our model on adapting to unseen network structures, (i.e., networks with varying number of nodes) we conduct experiments in which training is done on $N = 2, 3, 5$ network nodes and inference is done on $N = 6, 8, 12, 16, 20$ and 40 nodes.

N	Error %	RMSE	Avg. Test Loss
6	0.15 ($\pm$ 0.008)	0.137 ($\pm$ 0.006)	0.018
8	0.14 ($\pm$ 0.010)	0.141 ($\pm$ 0.010)	0.019
12	0.11 ($\pm$ 0.018)	0.121 ($\pm$ 0.014)	0.013
16	0.12 ($\pm$ 0.016)	0.134 ($\pm$ 0.014)	0.014
20	0.11 ($\pm$ 0.021)	0.120 ($\pm$ 0.018)	0.012
40	0.11 ($\pm$ 0.013)	0.121 ($\pm$ 0.016)	0.011

Table 6.3: Model performance results for inference on $N = 6, 8, 10, 12, 16, 20, 40$ with communication channel size $v = 1$ and $f_{in} = 25$ for the *max* task. Results are averaged over 5 runs and standard deviation values given.

The results shown in Table 6.3 are comparable to cases when only data from a single network structure is used for training. At training time, we vary N for batches of training samples $M_{train} = 400$ as shown in Figure 6.5.

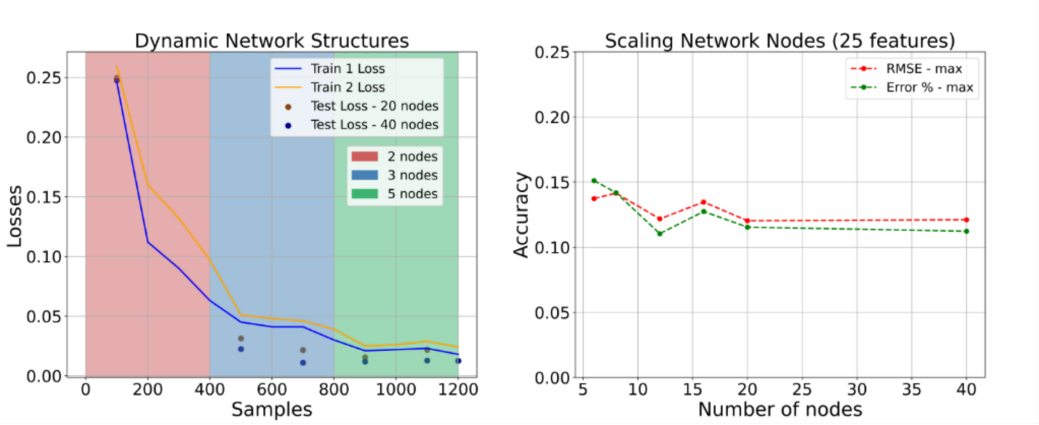


Figure 6.5: Left: Train and test losses on the *max* task. The network uses dynamic network structures with N= 2,3,5 for $M_{train} = 400$ for each network structure at training time. Right: Model performance for $N = 6, 8, 12, 16, 20, 40$.

Results on the test data show that our model generalizes well and does not encounter any performance degradation when tested on 20 and 40 nodes. Our model is robust to a dynamic number of nodes, N during inference and generalizes well to learn the dependencies on the defined tasks in spite of varying network structures. This is of key importance for IoT systems which can face dynamic changes in the number of nodes due to node loss or replacement.

Now that we have assessed model generalization from the perspective of different network structures (varying number of nodes), we turn to looking at our model performs on different network node topologies.

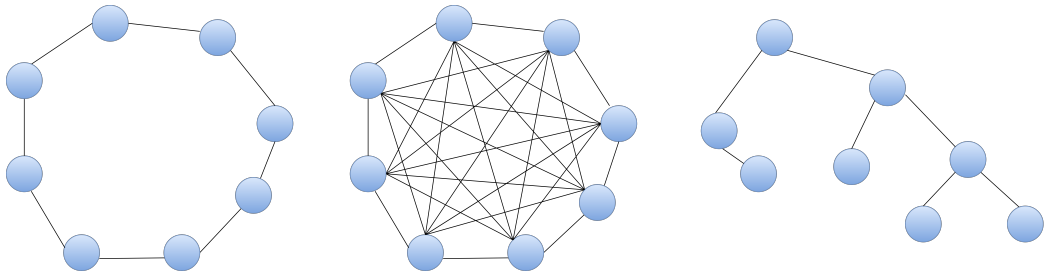


Figure 6.6: C^2RNN network topologies showing different node connectivity structures. In (a) we show a ring topology in which each node connects to exactly two other neighboring nodes in the network. In (b), we show a full mesh topology in which each node is connected to every other node in the network. In (c) we show a tree topology in which a 'root' node at the highest level connects to many other nodes in the levels below. In our case, we choose 1 to be the root node.

Our network is modeled as a graph with node connections determining which nodes can communicate with each other. We implement three different network topologies shown in Figure 6.6. For the sake of simplification, we assume that the location of a node is not important for our distributed tasks. In Table 6.4 we provide a summary of model performance results on the *max* task for the different network topologies.

Topology	Error %	RMSE
Ring topology	0.14 ($\pm$ 0.010)	0.141 ($\pm$ 0.010)
Full mesh topology	0.09 ($\pm$ 0.008)	0.093 ($\pm$ 0.008)
Tree topology	0.13 ($\pm$ 0.012)	0.128 ($\pm$ 0.009)

Table 6.4: C^2RNN performance results for inference on an 8-node system with f_{in} = 10 and $v = 1$ on the *max* task for Ring, Full mesh and Tree topologies. Results for the different topologies are averaged over 5 runs and standard deviation values given.

The Full mesh topology achieves 5 % better performance when compared to the Ring topology and is 4 % more accurate than the Tree topology on the *max* task. We can observe that there is no significant performance difference between Tree and Ring topologies. It is not surprising that the Full mesh topology generally performs better as the network can integrate latent states across all nodes faster, thereby enabling all nodes to have richer network-wide information on distributed task at each communication step. Nonetheless, since more messages are shared in a Full mesh network during inference, communication overhead issues are not uncommon. For instance, at each communication step, in the Full mesh topology, each node sends N-1 messages, whilst in the Ring topology it only would send only 2 messages.

6.4.2.3 Communication-efficiency

In addition to demonstrating that our model can learn to cooperatively solve distributed tasks via latent state communication, we quantitatively assess communication-efficiency at inference time.

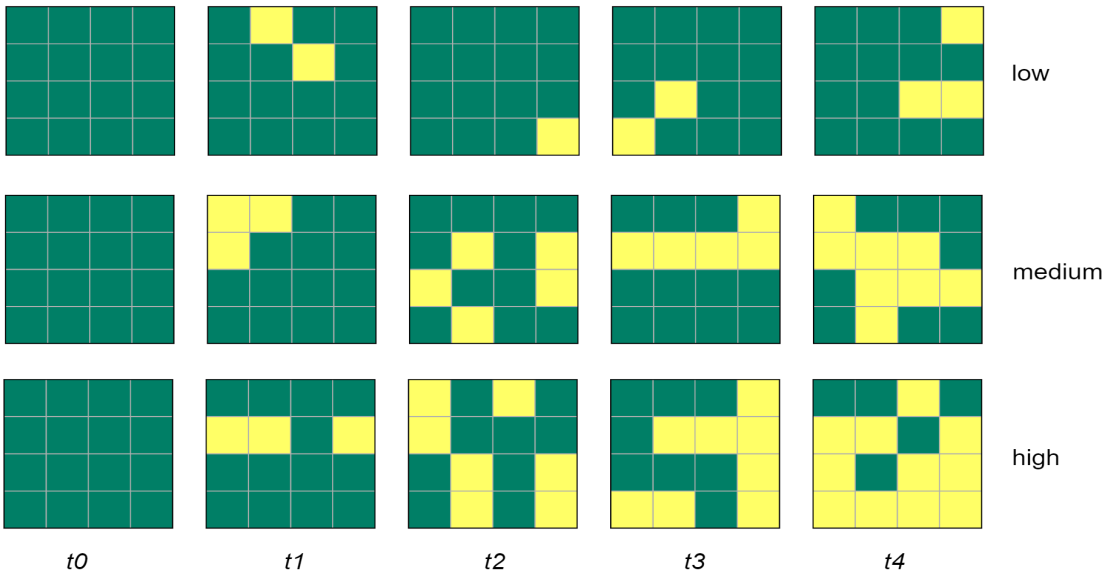


Figure 6.7: Communication patterns for $N = 16$ nodes across 5 sampling time-steps with low (factor: 0.25), medium (factor: 0.5) and high (factor: 0.8) penalization on communication cost optimization. Nodes in ON state (communicating nodes) are indicated by the green color whilst nodes in OFF state (non-communicating nodes) are indicated by the yellow color. We can see that our model optimizes for communication cost (more or less aggressively) depending on the impacting factor on the communication cost function.

In resource constrained systems, the total number of messages sent to solve a

problem must be minimized to conserve node energy and limit network bandwidth. In our experimental setup, we consider the total number of messages communicated t_{comm}, during inference. The communicated messages t_{comm} are determined purely by the model which optimizes a communication loss function L_{comm} by penalizing the total number of messages sent.

Figure 6.7 shows communication patterns in a 16-node network when we vary the penalization on communication cost optimization. Our model is able to automatically adapt to the impact on penalization. As an architectural consideration, we set the size of the node output communication channel $v = 1$. As such, each node can only communicate a single message at a time. Our results show that our model jointly optimizes for communication cost whilst maintaining high predictive accuracy at inference time.

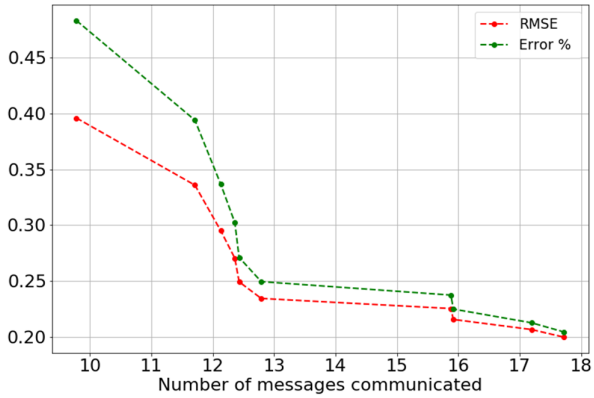


Figure 6.8: Model performance with varying number of messages communicated t_{comm} for $N = 5$ on the *max* task. This demonstrates the tradeoff between task accuracy and network resources. An operator could choose to focus on efficient communication at the cost of a small reduction in accuracy. Network nodes can only communicate for a total maximum of $t_{comm} = 25$ for each forward push.

In Figure 6.8, we show the trade-off between model performance and total number of communicated messages in t_{comm} for $N = 5$. By taking the total number of messages communicated across all nodes, t_{comm} and dividing it by the maximum limit (the denominator) of the number of messages that can be communicated during inference, we learn that only 51% - 56% communication is necessary to reach a good accuracy on the *max* task. This intuitively makes sense, as nodes observing small digits can stop participating in the process as it is clear that they do not satisfy the *max* operator. In Figure 6.9, we show how the network learns to communicate less

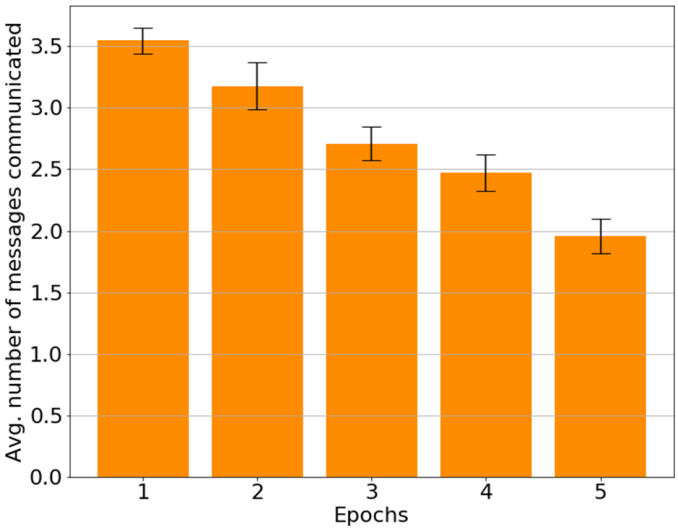


Figure 6.9: Average number of messages communicated t_{comm} for $N = 5$ as training progresses on the *max* task. Note how the network learns to slowly reduce the amount of communication needed.

with increasing number of training epochs. This shows that nodes first learn how to achieve good accuracy and then can reduce communication cost through subsequent optimization.

6.4.2.4 Computational Complexity

Intelligent spatially distributed IoT applications must execute a resource-aware operation that effectively utilizes limited computational resources available on embedded IoT devices. Although Deep Networks, (i.e., CNNs) are powerful for inference tasks in many IoT settings, they typically take up a significant amount of memory. In our experiment, we quantify the computational overhead on both the feature extractors and the *C-RNN* model.

Table 6.5 shows the number of parameters and FLOPs[1] (Floating-point Operations) per layer in the models. We can clearly see that most of the model's computation (30.28 % in the first convolutional layer and 64.7 % in the second convolutional layer) is incurred at the individual nodes' feature extraction kernels during convolution operations. Although the linear layer only accounts for 3 % of the computation, it involves 3× more parameters when compared to the convolutional layers. In total, the feature extractor takes approximately 0.16 MB of memory for inference on a

[1]We consider a FLOP as half a Multiply-Add operation.
Thus, a single Multiply-Add is 2 FLOPs

Op	Params, α (K)	FLOPs (K)/ %
Conv2d-1	0.26	149.76 / 30.28
MaxPool	-	4.32 / 0.87
ReLu	-	2.88 / 0.58
Dropout/Conv2d-2	5.02	320 / 64.7
MaxPool	-	0.96 / 0.19
ReLu	-	0.64 / 0.13
Linear-1	16.05	16.00 / 3.24
Totals	21.33	494.56 / 100

Table 6.5: Layer by layer computational overhead of a feature extractor applied on a single node at inference time. Model parameters, α are given and FLOPs are estimated for extracting features for a single image passed as a 4-D tensor on a single node. Feature extraction kernels in the convolutional layers take up more computational resources as compared to the fully connected layers.

single input image considering the forward/backward pass and parameters.

Table 6.6 shows the computational overhead for our *C-RNN* model.

	$f_{in} = 10$		$f_{in} = 25$	
Op	α (K)	FLOPs (K)	α (K)	FLOPs (K)
Linear-1	0.75	0.700	1.5	1.450
ReLu	-	0.050	-	0.050
Linear-2	0.408	0.400	0.408	0.400
ReLu	-	0.008	-	0.008
Linear-3	0.126	0.112	0.261	0.232
ReLu	-	0.014	-	0.029
Totals	1.284	1.284	2.169	2.169

Table 6.6: Layer by layer computational overhead for the *C-RNN* model. Model parameters, α are given for $f_{in} = 10$, 25 respectively and FLOPs are estimated for a single forward push at timestep t during inference on a single node. Using ReLu for layer activation saves on computation cost as other activation functions, i.e., Sigmoid are more computationally costly as they use more FLOPs.

FLOPs and model parameters scale with the dimensionality of the input. On an 8-node system, we realize a 5 % performance difference on the *max task* when

comparing $f_{in} = 10$ and $f_{in} = 25$, although the model performs 51 % more FLOPs when $f_{in} = 25$.

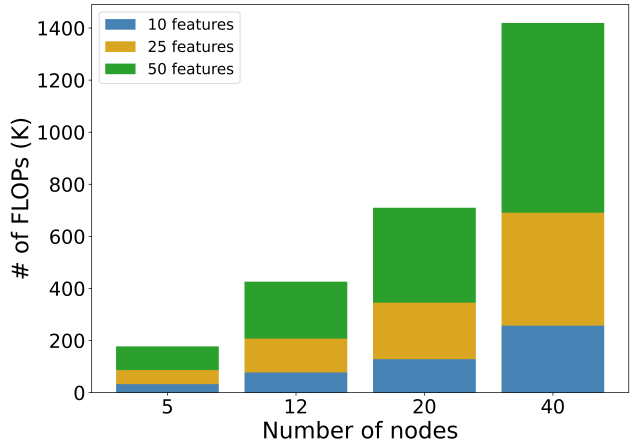


Figure 6.10: Total (sum of all nodes) computational load with increasing $N = 5$, 12, 20, 40 at inference time on the *max* task. Model parameters scale with increasing f_{in}. FLOPs are estimated as a product of N, t and number of FLOPs per forward push, where t is the number of timesteps for which we hold the model input.

Nodes in our model learn to distributively and cooperatively solve tasks via latent-state communication. To achieve this, the model performs a series of latent state updates across all nodes in the network at inference time.

f_{in}	N	Params (K)	FLOPS (K)	Error %
10	12	22.617	571.60	0.13 ($\pm$ 0.015)
	40	22.617	751.36	0.15 ($\pm$ 0.011)
25	12	23.499	624.70	0.11 ($\pm$ 0.018)
	40	23.499	928.36	0.11 ($\pm$ 0.013)
50	12	24.974	713.20	0.10 ($\pm$ 0.009)
	40	24.974	1,223.36	0.11 ($\pm$ 0.012)

Table 6.7: Summary of computational overhead and model performance for the unified C^2RNN model on the *max* task with $N = 12{,}40$ and $\alpha = 1284$, 2169, 3644 for $f_{in} =$ 10, 25, 50 respectively. FLOPs are given for inference on a single image. Results are averaged over 5 runs and standard deviation values given.

In Figure 6.10, we show the computational overhead across $N = 5$, 12, 20, 40. It is worth noting how doubling the nodes in the network results in 100 % increase in FLOPs (the computational overhead scales linearly). This is due to the model

performing more latent state updates with additional nodes in the network. Our results show that the computational overhead of the latent state updates is negligible with respect to the overall model.

As can be seen from Table 6.7, our unified model with CNN-based feature extractors integrated with the *C-RNN* model at each single node is suited for embedded, resource constrained devices. This is considering our model size ($<$ 25K parameters) and the number of FLOPs needed for inference in a N node network on a single image. Further studies are necessary to explore opportunities for optimization on feature extractors through quantization and model compression.

6.4.2.5 Memory and Energy Consumption

Energy and memory usage are key concerns when considering deployment of Deep Networks on resource-constrained embedded IoT devices. The number of network parameters and Floating Point Operations does not give a good estimation of the actual energy consumed when conducting inference using Deep Networks on these devices. That is mainly because different weights consume varying amounts of energy and the energy consumed is tied to the inference platform. For instance, in Convolutional Neural Networks (CNNs), the energy consumption of weights in the convolutional layers is similar to that of all fully-connected layers although convolutional layers have fewer weights than fully connected layers, so each weight in the convolutional layers is used more frequently than that in fully connected layers [225]. To estimate energy consumed by Deep Network models, we can consider the computation and memory accesses. For our C^2RNN model characterized by distributed communicating nodes, we must also factor in the energy cost of communication in addition to computation energy. We formulate the overall network energy during inference as:

$$E_{TOTAL} = E_{COMP} + E_{COMM} \tag{6.7}$$

where E_{COMP} is the energy cost of computation and E_{COMM} corresponds to the energy cost of communication.

In our case, we assume that our model can fully rely on on-chip memory and avoids communication to larger off-chip memory, i.e., DRAM accesses which are more costly. We estimate the relative energy consumption of SRAM fetches and MAC (Multiply-Accumulate) operations according to a generic model proposed in [81, 149]. Using this approach, the energy consumption of memory accesses on a small SRAM on-chip memory is considered to be equivalent to the energy of a single Multiply-Accumulate (MAC) operation E_{MAC}. In our case, a MAC is 2 $\times$ 1 FLOP. Drawing from the

energy modelling in [81], we take E_{MAC} as 5 pJ $\times$ $(32/Q)^{1.25}$ for a float32 full precision network with Q fixed at 32 bits. Computation energy consumption for inference on a single test sample, E_{TOTAL} can be calculated as:

$$E_{COMP} = E_{COMPUTE} + E_{ACCESSES} \tag{6.8}$$

Where $E_{ACCESSES}$ is the sum of weight and activation accesses for the Deep Network and $E_{COMPUTE}$ is the computation energy. To obtain $E_{COMPUTE}$, we use the following equation:

$$E_{COMPUTE} = E_{MAC} \times C + (3 \times A) \tag{6.9}$$

In this equation, C is the complexity of the network in terms of number of MAC operations and A is the total number of activations in the entire Deep Network. Since biasing, batch-normalization and activation is done on all activations, we multiply activations, A by 3. To compute $E_{ACCESSES}$, we use the following equation:

$$E_{ACCESSES} = (2 \times E_{MAC} \times P + 2 \times E_{LOCAL} \times C) + (2 \times E_{MAC} \times A) \tag{6.10}$$

Where P is the number of parameters (weights and biases) in the network, E_{LOCAL} is the energy per read and write from local storage. Memory access typically ranges from 5pJ for 32 bit coefficients in on-chip SRAM [80]. We could realize further reduction in memory energy due to activation and weight-level parallelism and weight reuse in the case of CNNs which employ shared weights. For the sake of simplicity, we will not reduce the energy consumption for our model by a parallelism factor as this requires further assumptions.

Now that we have provided an approach to estimate the energy cost of computation, we can consider the energy cost of communication. In our model, each node transmits a 1-byte message in the form of a fixed sized tensor. The channel size can be adjusted to transmit more than 1 message per node, same for receiving channel. Since communication is done within the Deep Network, we assume maximum channel gain and no background noise in the communication channel.

Estimating energy consumption during communication within Deep Networks model is difficult since a significant portion of the communication energy is consumed by data movement and there is no direct way to extract this from the models [224]. Using the approaches used in [115,124], we estimate the energy cost of communication E_{COMM}, using the following equation:

$$E_{COMM} = p \times T_{COMM} \tag{6.11}$$

where p is the system transmission power and T_{COMM} is the time complexity of communication for the system. As in [115], we can estimate the time complexity as:

$$T_{COMM} = c\delta + \omega B \tag{6.12}$$

where c is the number of communicated messages during forward inference, ω is the aggregate message size, δ is the estimated message latency (1-N) and B is the system bandwidth. We can see that the energy cost of communication depends on the size of the communicated message and is scaled by the platform cost of communicating that message. T_{COMM} is be obtained directly from the Pytorch platform and we assume p to be 0.5 W. Table 6.8 provides overall network energy consumption and model memory for our C^2RNN compares them to state-of-the-art models trained for MNIST image classification and inference on resource-constrained IoT devices.

Model	Params, α (K)	Model Size (kB)	Total Energy (mJ)
C^2RNN-1	22.617	84.82	2.09
C^2RNN-2	24.974	93.66	4.42
Bonsai [110]	21.000	84.00	1.27
ProtoNN [78]	15.957	63.83	2.00

Table 6.8: Energy Consumption and memory requirements for our C^2RNN models compared with state-of-the-art implementations of Deep Networks for inference on a single image on resource-constrained IoT devices. C^2RNN-1 is a lighter model with $f_{in} = 10$, and 12 nodes whereas C^2RNN-2 is a heavier model with $f_{in} = 50$ and 50 nodes.

The energy cost of computation is 83.5 % (1.74 mJ) and % 71.7 (3.17 mJ), whereas the energy cost of communication is 16.5 % (0.345 mJ) and 28.3 % (1.25 mJ) for the C^2RNN-1 and C^2RNN-1 models, respectively.

In a typical federated computing setting, we would expect the energy cost of communication to be significantly more than that of the energy cost of computation. This is because moving messages between nodes constitutes data movement and this is more expensive than computation. Nonetheless in our case since all communication is in the latent states of the Deep Network model, the actual realized energy cost of communication is less than the computation cost. This is expected. We estimate

the energy cost of communication assuming that all nodes communicate at the communication step. Since we optimize communication, accounting for the optimization on communication would mean that the energy cost of communication would be less than stated. We learn that the latent-state communication has negligible impact on the overall energy required for the model. Our model is small enough, making it amenable for storage on-chip and is energy-efficient for Class 2-3 IoT devices (refer to 2.2).

6.4.2.6 Network Robustness

In distributed multi-node settings, network robustness is of paramount importance. This is because lossy links could potentially cause a system to be unreliable, thereby adversely impacting its ability to deliver on its intended application and or service. We model lossy-links in which network nodes may fail to communicate their latent states due to obstructions or multi-path.

Link failure	0 %	20 %	40 %	60 %
Error %	0.170 (± 0.009)	0.176 (± 0.010)	0.188 (± 0.019)	0.194 (± 0.017)
t_{comm}	7.0	10.0	13.0	13.0

Table 6.9: Model performance and average t_{comm} with link failure. Error % is averaged over 5 runs and standard deviation values given.

Table 6.9 shows a summary of our results with various percentages of link failure with $N = 5$ and $v = 1$. Our C^2RNN model's performance degrades by only 3.4 %, 13.6 % and 10.05 %, respectively for 20%, 40% and 60% link failure on both *max* and *max-parity* tasks. Moreover, our model automatically learns to increase the total number of messages, t_{comm} sent amongst nodes with increasing link failure. This shows that our model is robust to failing nodes.

6.4.2.7 Comparison with Other Methods

When considering distributed inference on local, resource-constrained devices, communication cost is an important concern. We evaluate our model's computational footprint by comparing it with 4 baseline methods.

- **Energy-Unaware Distributed C^2RNN Baseline** In this case, we setup an energy unaware variation of the C^2RNN model. This model does not optimize for communication cost as in the case with our C^2RNN model. As in the

C^2RNN model, each node observes a unique 28 x 28 MNIST digits in gray-scale, extracts 25 features and feeds features into the *C-RNN*, which distributively and cooperatively solve the objective functions (*max and max-parity* tasks) locally.

- **Centralized Baseline:** In this model, each node observes a unique 28 x 28 MNIST digit in gray-scale and transmits it to a central compute point that computes the objective function.

- **Semi-centralized Baseline:** Each node observes a unique 28 x 28 MNIST digit in gray-scale, extracts 25 features from each image before transmitting them to a central point that computes the objective function.

- **Tree-based Baseline:** This approach considers a classic Wireless Sensor Network arrangement as in [60]. We model a tree structure with $N = 12, 40$. At inference time, the nodes observe a unique 28 x 28 MNIST digit in gray-scale, extracts 25 features and communicates features via links to neighboring nodes to an aggregating node. The aggregating node then passes features to a central point for computation of the function from features across all nodes.

We evaluate performance based on the communication and compute footprints of the baseline models compared to our C^2RNN model. The communication footprint measures the actual size of the messages being transmitted/communicated whilst the compute footprint gives a measure of the distance from the very edge of the network to a central compute point. In our case we model the distance from nodes at the edge of the network to a central compute point as 4 units. All baselines use the same architecture for feature extraction as given in Section 6.3.2. In turn, the memory requirement (1.8 MB for a 12-node system and 6 MB for a 40-node system) for feature extraction on a single image is similar across all baselines.

From results in Table 6.10, we can clearly see that the communication and compute footprints for the centralized baselines scales quadratically when compared to our C^2RNN model. Transmitting images to a central compute point is 8× more expensive when compared to transmitting features from individual nodes. Although the communication and compute footprints of the centralized approaches seem hefty, the models achieve slightly higher accuracy on the objective task. Optimizing for communication cost, as in our C^2RNN model reduces the communication footprint, on average, by 1.7× when compared to the energy-unaware variation of the same model. For the two models, there is a marginal difference (10 %) in model performance. Our C^2RNN model uses 25× less communication and 1.9× less compute resources when

Baseline	N	Comms	Compute	Error %
Energy-Unaware C^2RNN	12	240	12.0	0.09 (± 0.009)
	40	800	40.0	0.10 (± 0.013)
Semi-centralized	12	1200	48.0	0.08 (± 0.008)
	40	4000	160.0	0.09 (± 0.017)
Centralized	12	9408	48.0	0.09 (± 0.013)
	40	31360	160.0	0.09 (± 0.015)
Tree-based	12	3400	23.0	0.10 (± 0.015)
	40	8100.16	160	0.11 (± 0.009)
C^2RNN	12	137.6	12.0	0.11 (± 0.018)
	40	440	40.0	0.11 (± 0.013)

Table 6.10: Communication (bytes) and compute (units) footprints for the different baselines considered in our experiment on the *max* task with $N = 12$, 40 for $f_{in} = 25$. The communication and compute footprints are given for inference on a single image across all nodes in the network. Results are averaged over 5 runs and standard deviation values given.

compared to the Tree-based baseline in a 12-node system. These results demonstrate that our C^2RNN achieves an encouraging balance on communication and compute resource utilization whilst automatically learning to optimize communication to solve distributed tasks locally. Moreover, our approach automatically discovers complex communication patterns without compromising on performance.

6.4.2.8 Architectural Considerations

Our C^2RNN model presents opportunities for further optimization across the architectural considerations. We manually vary the sizes of the node output communication channel using $v = 1$, 5. The node output communication channel size, v corresponds to the number of latent states each node can communicate at the communication timestep.

As can be seen from Table 6.11, for $v = 5$, a model with $f_{in} = 10$ achieves an error rate 2 % lower than when $v = 1$ on the *max* task. When we change v from 1 to 5 in a model with $f_{in} = 25$ we realize a 2.6 % performance gain. In turn, we can note that a slightly higher value of v improves model performance as more latent state information across nodes is retained for learning the objective within the model layers. From these results, we can also notice a small increment in the model's performance

f_{in}	v	α	RMSE	Error %
10	1	1284	0.150 ($\pm$ 0.011)	0.18 ($\pm$ 0.009)
	5	2228	0.146 ($\pm$ 0.010)	0.16 ($\pm$ 0.016)
25	1	2169	0.156 ($\pm$ 0.008)	0.17 ($\pm$ 0.009)
	5	3113	0.140 ($\pm$ 0.003)	0.14 ($\pm$ 0.018)

Table 6.11: Model performance for $N = 5$, with communication channel sizes $v = 1$, 5 and parameters α for models with $f_{in} = 10$, 25 on the *max* task. We select v such that the maximum value is 50 % or less than the size of f_{in}. Results are averaged over 5 runs and standard deviation values given.

when we use more input features in a setup with $f_{in} = 25$ when compared to a setup with $f_{in} = 10$. It is worthwhile to note that the model's parameters scale with increasing f_{in} and v values considering that these parameters directly affect the size of the latent states across nodes in the model. In spite of this, we can instead choose to optimize on the model's memory and communication footprint by minimizing values of f_{in} and v. Nonetheless, more complex tasks could require nodes to communicate more than a single message at each time, in which case the operator can easily adjust the v accordingly to suit a task at hand.

6.5 Discussion

In this chapter, we proposed a novel, scalable, data-driven and communication-efficient Communicating Convolutional Recurrent Neural Network (C^2RNN) framework for distributed visual settings. Our model enables seamless latent state communication without the need for pre-determined and hand-engineered communication protocols. We showed through an array of experiments that our model learns to discover communication protocols scalably and can be extended to cooperative, distributed IoT scenarios with visual data. Our model achieves high accuracy with low computational and communication costs on target objectives. All experiments were performed with repetition with results being averaged over 5 runs. It is interesting to notice the relatively high repeatability of the results as reflected by very small values of standard deviation across all experiments.

Our model is capable of performing multi-objective optimization on prediction accuracy for (*max* and *max-parity*) tasks and communication by means of a communication loss. As shown in Section 6.4.2.3, it is possible to trade-off prediction accuracy for a reduction in communication cost. With only 56 % communication, we can

achieve desirable accuracy (0.104 RMSE on an 8-node network on the *max* task) on the max task. These kinds of multi-objective optimization problems are challenging because they require careful design and weighting of the loss functions for the learned objectives to avoid conflicts. Most real life tasks are intrinsically multi-objective, making our approach amenable for resource-constrained IoT settings which require optimization on computation, communication, energy costs and prediction accuracy amongst other factors.

We estimated the memory and energy consumption of our C^2RNN framework considering memory accesses, communication and computation cost. The realized energy cost of communication is small and does not significantly impact the energy requirement for the model. Our model can fit and operate on Class 2-3 IoT devices as defined in Table 2.2.

Further to this, in Section 6.4.2.6 we show that our C^2RNN framework harbors self-organization capabilities that make it robust to lossy links and tolerant to faulty nodes. We modelled dropped packets (i.e., nodes fail to contribute to inference) and learned that the model adapts by increasing communication amongst active nodes and achieve minimal performance degradation (only 3.4 % for 20 % dropped packets). This result, although indicative of our model's robustness, could be explored further by asking whether the network could be adaptive over time to changing traffic conditions (i.e., if the network suddenly degrades performance based on various forms of network node interference). It would be interesting to navigate if the network could ultimately generate bias toward more frequent communication as a means to counter lossy links, leading to broadcast storms. We show that our framework can generalize to unseen network structures and can run over varying network sizes and topologies. Realizing efficient communication in resource constrained environments whilst meeting required accuracies on target objectives is a key challenge. Our approach makes strides toward achieving this using intelligent Machine Learning algorithms that can sit on distributed, cooperating, resource-constrained end devices in IoT settings, albeit demonstrated on simpler tasks (*max* and *max-parity*). This framework presents opportunities for experimentation on more challenging problems, (e.g., did the green car turn down the left or right at the intersection?) that would require nodes to be aware of their location, whether relative or absolute. Moreover, this work has concentrated on homogeneous data sources, i.e., either real numbers (Naive setting) or images (visual setting). As the communication is decoupled from sensing, it would be interesting to explore our our framework in multimodal settings,

i.e., audio-visual challenges. That would simply require, that the front-end network be altered to according to the to the sensed phenomena.

We can also note some challenges associated with training cooperative deep networks in distributed settings. When nodes in the distributed network scale, (e.g., to 1000 nodes), training becomes more difficult and without modification, would need a supercomputer. This is because the model must be uniformly updated across all nodes to perform distributed inference and achieve high accuracy on objective tasks. Careful design is necessary to consider how to scalably train distributed, cooperative models and constantly update them without introducing unnecessary network overhead during parameter updating. Another notable issue is consistency across nodes. In this framework, we assume that sensing is largely error free and not subject to adversarial attacks. Invalid input data could easily compromise the ability of the distributed network to operate correctly. We highlight this as a particular area to explore further.

In the next chapter, we extend our C^2RNN architecture to a more complex problem. We experiment with more sophisticated multiple node connection architectures in distributed target tracking applications (i.e., naive and visual settings). In doing this, we are interested in achieving robust, accurate and communication-efficient target tracking via automatically learned latent state communication with multiple distributed, cooperating nodes.

Chapter 7

Learning Distributed Communication in Target Tracking Settings

7.1 Introduction

In Chapter 6, we proposed a hybrid Communicating Convolutional Recurrent Neural Network C^2RNN framework for distributed visual settings. In doing this, we integrated feature extractors with our communicating RNN framework into an end-to-end trainable architecture capable of solving distributed tasks through seamless latent state communication. We showed that our framework cooperatively solves distributed image classification tasks with high accuracy, optimizes computational-cost, communication-cost, generalizes well to unseen network structures, scales to multiple nodes and is robust to link failure. Moreover, we showed that our proposed system outperforms *centralized*, *semi-centralized*, *tree-based* and *energy-unaware distributed C-RNN* baselines when evaluated for communication-efficiency and computational-efficiency. Although, our experiments on distributed *max* and *max-parity* tasks in Chapter 6 prove the efficacy of our proposed framework, they focus on distributed image classification without regard to visual target tracking, i.e., detection and prediction over time and space.

In many real-world applications, the problem of distributed classification is often tied to tracking applications. For instance, in Wireless Visual Sensor Networks (WVSNs) [14, 218], low-cost, battery-powered, camera-equipped sensor nodes communicate to cooperatively perform target detection, tracking and recognition tasks. In turn, processing visual data enables surveillance, industrial monitoring and environmental monitoring applications through richer sensing. In contrast to more

traditional Wireless Sensor Network (WSNs) settings which collect scalar information (typically easier to process), WVSNs present unique challenges. To successfully track visual targets, important factors, such as, camera node field of view must be considered. This is because, camera-equipped sensing nodes cannot capture a tracked target out of the sensing range or if any occlusions exist in the field of view. Sensing coverage can be reduced by faulty nodes warranting optimal deployment approaches. Moreover, processing visual data introduces significant strains on a system's memory consumption and energy-efficiency. Huge bandwidth demands in such a setting, can impact network performance and affect latency sensitive applications.

In this chapter, motivated by these challenges, we propose a cooperative, grid-based, distributed target tracking framework based on our *C-RNN* and C^2RNN frameworks for naive settings and visual settings. Grid-based topologies have been successfully explored in similar settings using multi-agent deep Reinforcement Learning to code agent motion with inspiration drawn from developmental studies of Entorhinal Cortex (EC) in neural systems [24, 50]. As a natural consequence, our framework instantiates intricate node connection architectures based on sensing node grids. Conventionally, target tracking and target classification are considered separate problems [170]. Our framework jointly performs target tracking and classification in a unified manner.

In the previous chapter, we implemented a communication loss to optimize for communication-cost as a single objective with an accuracy tradeoff. In this chapter, we implement multiple objective optimization on tracking grid reconstruction, network energy and communication-cost. Our approach can automatically discover and optimize distributed communication, accurately track targets both in naive and visual settings in an energy efficient and computationally efficient way, and scales to multiple nodes. In summary, our contributions for this chapter are as follows:

- We propose novel, data-driven, energy-efficient and communication-efficient *C-RNN* and C^2RNN architectures for distributed target tracking in naive and visual settings.
- We present an end-to-end trainable grid-based scheme for cooperative, distributed target tracking based on Deep Networks.
- We employ a node communication paradigm that enables sensing nodes in the grid-based scheme to communicate feature representations in their latent states from observed visual data.

- We implement multiple objective optimization on tracking grid reconstruction (predicting network grid with tracked target), network energy (activation of only relevant sensing nodes) and communication-cost (learning to communicate only relevant information for the tracking objective).
- We evaluate our proposed approach on various tasks (i.e., naive tracking, and visual MNIST tracking) whilst assessing grid reconstruction and tracking performance, network energy optimization (based on *low*, *medium* and *high* optimization schemes), communication-cost optimization, and node scalability.
- We show that our approach can employ latent state communication and learns to cooperatively and distributively track targets in a resource-aware manner, considering computation-cost and communication-cost.

For the rest of the chapter, in Section 7.2, we provide a formulation of the problem setting being addressed. In Section 7.3 we present our distributed tracking system architectures for naive and visual settings, and training and inference. Section 7.5 presents our experimental setup and results whilst Section 7.6 provides a concluding discussion for the chapter.

7.2 Problem Setting

In distributed tracking, multiple sensing nodes cooperate to obtain an accurate representation of an environment with a moving target. The idea is that these multiple nodes can exchange information to realize a representation of the sensed environment that is more accurate than a representation obtained independently [168]. We consider a grid setting similar to the setting in [136] in some respects. In our setting, multiple sensing nodes $N = \{n_1,...,n_N\}$ are positioned arbitrarily in fixed positions. Each node has at least 2 neighbors based on its position in the grid. The overall network can be represented as an undirected graph $G = (\mathrm{N}, \xi)$ where $N =$ is the set of nodes in the network and $\xi \subset N$ x N represents the presence or absence of a connection between any two nodes $(i, j) \in \xi$. At any given timestep t, a single target, τ can move in 4 cardinal directions (North, West, South, East) continuously for the duration of the tracking period, P. For simplicity reasons, τ is incapable of executing diagonal motion. Network nodes observe unique naive (scalar values) and/or visual (MNIST digits) data and automatically learn to communicate latent state representations of image features or observed scalar values to locate the position of the tracked

target based on their connections as defined by the network graph, G. In doing this, network nodes distributively classify tracked target τ at time t. The entire network is assumed to have a limited energy budget and nodes can only communicate with respect to confined ranges. To achieve distributed target tracking whilst respecting resource constraints, sensing nodes must learn activation patterns based on the position of the tracked target. This allows the network to optimize for network energy by turning sensing nodes *on/off* whilst maximizing prediction tracking accuracy. Moreover, if sensor nodes are in *on* state, preemptively detecting a target, they can choose to communicate only vital information relevant to the tracked objective. Overall, the goal is to achieve sustainable distributed target tracking in resource-constrained IoT settings without the need for camera-equipped, battery-powered sensing nodes to instantiate complex, pre-determined and specialized communication protocols.

7.3 Approach

In this section, we provide an overview of our framework for distributed target tracking. In doing this, we discuss how we implement and train Deep Networks involving latent state communication and multiple objective optimization to distributed tracking tasks.

7.3.1 System Architecture

The cooperative, grid-based, distributed target tracking framework extends our *C-RNN* and C^2RNN architectures as respectively given in sections 5.3 and 6.3. In turn, we employ homologous and unidirectional RNNs with weight sharing instantiated on distributed network nodes N, to communicate latent state representations with pre-determined neighbors based on a grid-based topology for distributed tracking tasks over defined periods of tracking steps, P. Figure 7.1 provides an architectural overview of our proposed system in naive and visual settings.

7.3.2 Naive Setting

In the naive setting, we consider a simplified objective in which cooperating nodes observe real-valued data, and communicate distributively to determine the position of a tracked target on a square tracking grid. Consider, a single node a in a network of N nodes. At any given timestep t, a node a makes a series of observations denoted by $X^a = \{x_1^a, ..., x_P^a\}$ and communicates latent space representations of its observations

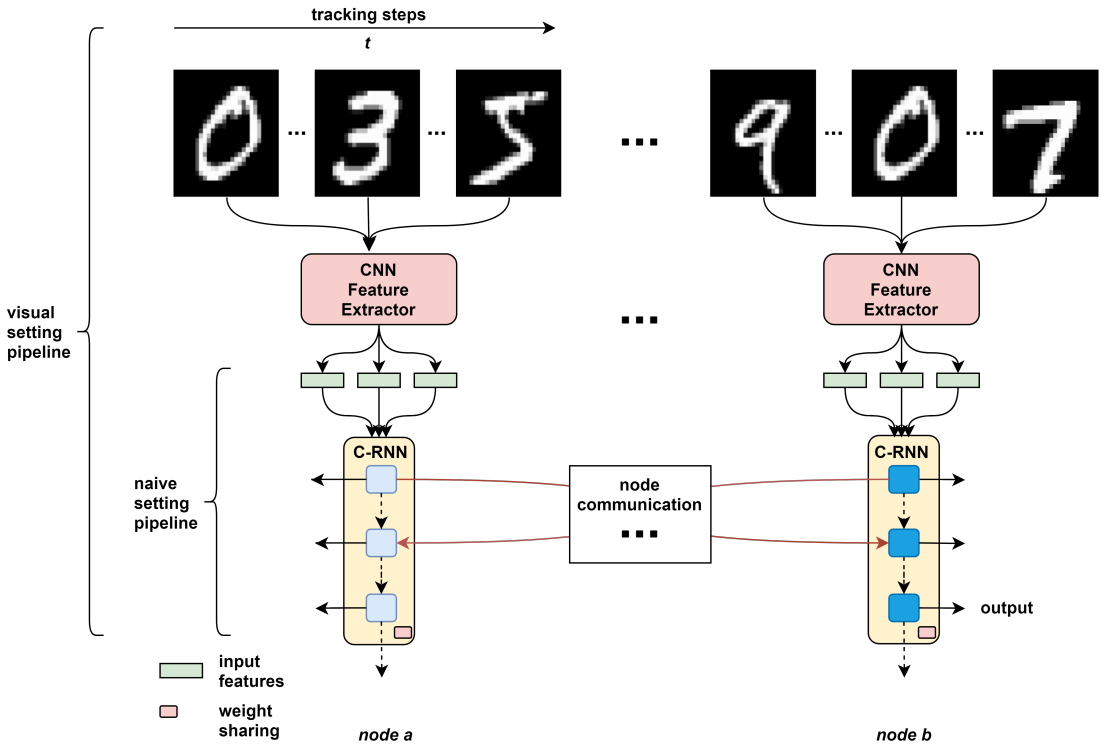


Figure 7.1: Architectural overview of distributed target tracking framework. At any given tracking step p = 1,...P, nodes a and b observe MNIST handwritten digits in gray-scale and extract features that are fed into the *C-RNN* network. Each inference timestep t the model takes as input a series of features for P MNIST digits. The input features to the *C-RNN* model encode temporal dynamics with movement for the tracked digit in 4 cardinal directions (North, West, South, East). In the naive setting, node observations are passed directly into the *C-RNN* network bypassing the feature extraction process.

$S^a = \{s_1^a, ..., s_P^a\}$ with its neighbors . Note that, nodes at the 4 corners of the tracking grid have 2 neighbors whilst edge nodes have 3 neighbors and the rest of the inner nodes have 4 neighbors . The communication across all network nodes is performed by a communication handler ψ which performs a mapping operation $\hat{\mathbf{s}} = \psi(\mathbf{s}, G)$. In this case, $\hat{\mathbf{s}} = \{\hat{s}_1, ..., \hat{s}_n\}$ is the original latent state updated with node communication as specified by the network graph, G. Our communication paradigm is described in more detail in Section 4.5. It is worthwhile to note that observations are passed directly into the *C-RNN* framework without any pre-processing for naive settings.

7.3.3 Visual Setting

Each node implements a CNN-based feature extractor that encodes an MNIST digit into a set of features. At any given tracking step p, a 28 x 28 MNIST digit in gray-

scale is passed into the feature extractors. We employ the same feature extractors presented in 6.3.2. In turn, the feature extractors across all network nodes have an identical architecture and are initialized with the same weights from a pre-trained MNIST handwritten digits classifier. Figure 7.2 shows a high-level diagram of a feature extractor showing two convolutional layers (the first one with 10, 5×5 kernels and the second one with 20, 5×5 kernels), max pooling and dropout layers and a final fully connected layer, FC.

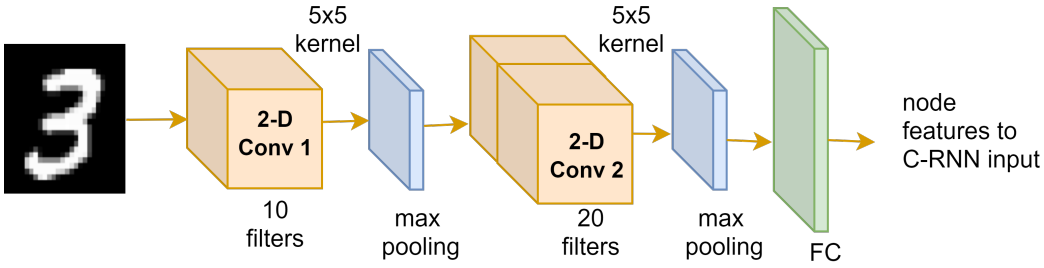


Figure 7.2: Feature extractor architecture for each node. The output of the FC constitutes the input to the *C-RNN* model.

Considering a single node a, the feature extraction process produces a set of features denoted by $X^a = \{x_1^a, ..., x_P^a\}$, which are passed into the *C-RNN* framework. Here we can note that the dimensionality of the input features fed into the *C-RNN* model is larger when compared to the naive setting. Similar, to the naive setting, at any given timestep t, where a tracking period, $p = 1,...,P$ is defined, latent space representations of node a's observations $S^a = \{s_1^a, ..., s_P^a\}$ are communicated with its neighbors . Communication is necessary for the distributed image tracking task as any single node is limited to the observations at its fixed position on the grid. We employ the same communication paradigm as in the naive setting.

7.4 Training and Inference

In this section, we discuss the setup for distributed tracking grids and our model training method for naive and visual distributed tracking settings.

7.4.1 Tracking Grids

Distributed network nodes are tasked with observing real-valued data in the naive setting and MNIST images in the visual setting over a defined tracking period, P and cooperatively determine the position of a tracked target on the grid via latent

state communication. Network nodes, N are arbitrarily placed in fixed positions and arranged in a a X $a = N$ tracking grid, where N is the size of the tracking grid and overall number of nodes in the network. For instance, a 4 $\times$ 4 grid, has 16 nodes. Each grid cell *(i,j)* for $1 \leq i \leq a$, $1 \leq i \leq a$ provides a position on the overall grid. A node observing a target can execute motion for the target τ in its immediate position based on a pre-defined action $\boldsymbol{m} \in A$, where A is the motion space with 4 cardinal directions (North, East, South, West) continuously for P tracking steps. At any given tracking step $p \in t$, a target can either move or maintain its current position on a specific node whilst the rest of the nodes in the grid keep their observations fixed. Move or wait actions both acquire unit costs. The network f predicts the overall state of the tracking grid s through a reconstruction based on predictions of $m_{i,j}$ for each grid cell (node) regardless of the availability of a target at grid cell *(i,j)*. Along with the reconstruction, the network decides which nodes to turn in *on / off* position at any given time-step *t*. For nodes in *off* position, we fill their input vectors with zeros.

7.4.2 Model Training

We follow a similar network training approach as presented in Section 6.3.4 with some modifications. In our framework, distributed nodes arranged in a grid-based topology observe unique sequences of data and communicate recurrently to track a target moving on the grid. Consider a single node $a \in N$. At each given tracking step p, node a determines its activation as *on / off*. Note that a time-step, t is composed of a series of tracking steps (the tracking period), hence $t = P$. Let x_n be the n^{th} node sequence of input features. It follows that,

$$x_n^p = \begin{cases} x_n^p & \text{if } Node\ \ State\ \ ON \\ \mathbf{0} & \text{otherwise.} \end{cases} \tag{7.1}$$

So $\{\mathbf{x}_1^t = \{\mathbf{x}_n^1, ..., \mathbf{x}_n^P\}$. Input features from all nodes in the grid $\mathbf{x} = \{\mathbf{x}_1^t, ..., \mathbf{x}_N^t\}$ are iteratively passed into the network to produce model updated latent states for all network nodes at timestep step t, $\hat{\mathbf{h}}^t = \{\mathbf{h}_1^t, ..., \mathbf{h}_N^t\}$:

$$\hat{\mathbf{h}}^t = f_\theta(\mathbf{x}^t, \hat{\mathbf{s}}^t) \tag{7.2}$$

Where $\hat{\mathbf{s}}^t = \{\mathbf{s}_1^t, ..., \mathbf{s}_N^t\}$ are resulting latent states with node communication performed via the operation $\hat{\mathbf{s}}^t = \psi(\mathbf{s}^t, G)$ as given in Section 4.5. G represents the network graph which determines node connections used to produce a mapping encompassing the node communication. Once the communication step is complete,

the model f_θ, with parameters and biases θ computes outputs for all nodes $\mathbf{o}^t = \{\mathbf{o}_1^t, ..., \mathbf{o}_N^t\}$ at time-step t as:

$$\hat{\mathbf{o}}^t = \sigma(\hat{\mathbf{h}}^t) \tag{7.3}$$

where σ is a ReLu non-linearity function. The model f_θ implements shared weights optimized using Adam optimizer [105]. In turn, all nodes implement the same network. The network outputs predictions as reconstructions of original input features as $\mathbf{x}^t = \{x_1^{t+1}, ..., x_N^{t+P}\}$ after P tracking steps. We train and evaluate our model with multi-objective optimization on prediction/reconstruction accuracy, network energy cost and communication-cost. For tracking grid reconstruction, we minimize the Mean Squared Error (MSE) loss between model outputs and targets as given in equation 7.4.

$$L_{acc}(\hat{y}, y; \theta) = \frac{1}{z}\sum_{i=1}^{z}(\hat{y}_i - y_i)^2 \tag{7.4}$$

where $\hat{y}$ is the network's prediction with reconstructed digits on the tracking grid and y is the network target. Let y_n be the n^{th} node's target. The node's target, $y_n \equiv x_n^{t+1}$. We also optimize network energy as given in equation 7.5 and node communication as given in equation 7.6. In turn, we implement a combined loss as given in 7.7, allowing us to simultaneously optimize on prediction accuracy for tracking grid reconstruction, network energy (nodes ON/OFF) and node communication.

$$L_{energy}(e, N; \theta) = \frac{e}{N} \tag{7.5}$$

$$L_{comm}(\hat{a}, a; \theta) = \frac{1}{z}\sum_{i=1}^{z}(\hat{a}_i - a_i)^2 \tag{7.6}$$

$$L_{overall} = w_{acc}L_{acc} + w_{energy}L_{energy} + w_{comm}L_{comm} \tag{7.7}$$

In equation 7.5, e is the number of nodes in ON position at inference time and N is the total number of nodes on the tracking grid. In 7.6, $\hat{a}$ is the total number of messages communicated across communicating nodes during inference and a is the target number of communicated messages based on a communication budget. The target communication, a is a product of an optimization factor o and the total number of nodes on, e at inference time. The optimization factor, o is impacted by the operator's choice of *high*, *medium* or *low* optimization. Note that $0 < o < 1$, so *low* optimization takes a value close to 1. In this case, the model learns to spend most

of its communication budget. Equation 7.7 represents how we calculate an overall loss, $L_{overall}$ with variable weightings w_{acc}, w_{energy} and w_{comm} for each respective loss function. This enables to control the impact of a specific loss function to our model for our multi-objective optimization process. Moreover, we can realize necessary tradeoffs amongst network's predictive accuracy and the different objectives subject to network energy and communication constraints.

We train our model for the distributed target tracking objective in an end-to-end fashion. Consequently, a single network is learned across all N nodes in the tracking grid. The network dynamically fuses node observations with temporal information encoding tracked target digit motion and overall tracking grid state changes. In our approach, multiple nodes on the tracking grid observe a MNIST handwritten digit (visual setting) or real-valued data (naive setting), evolve their individual latent states and communicate with their immediate neighbors to cooperatively track a moving digit on the grid. At inference time, the network decides which nodes to turn *on / off* based on the position of the tracked target digit and overall state of the tracking grid learned via distributed latent state communication. In turn, nodes in *off* position do not contribute to the inference pipeline as they are incapable of making observations. In doing this, the network can optimize for tracking grid energy whilst achieving the distributed target tracking objective. There is no central coordination for the tracking task as the network learns to communicate optimally in an energy-efficient distributed operation.

7.5 Evaluation

In this section, we focus on evaluating our data-driven C^2RNN architecture on distributed target tracking tasks. For distributed target tracking in visual settings (using Visual Tracker), we perform extensive experiments using the MNIST handwritten digits dataset. We also extend our approach to naive settings (using Naive Tracker) with real-valued data. Moreover, we provide an analysis of computational overhead in both visual and naive settings showing how our architecture is memory-efficient and achieves its intended tasks with low computational cost.

7.5.1 Experimental Setup

Our experimental setup is intentionally designed to assess whether our approach can discover distributed, emergent latent state communication protocols to cooperatively and distributively track targets in a grid-based scheme with varying network node

structures and optimization on network energy and communication. We setup 16-node (4×4) and 25-node (5×5) grids with each node trained on training samples M_{train}. Each training sample is composed of P 4×4 grids (in a 16-node network) encoding the motion of the tracked target digit.

7.5.1.1 Input Features

For the visual setting, all network nodes in the model instantiate CNN-based feature extractors initialized with pre-trained MNIST digit classifiers. The feature extractors transform MNIST handwritten digits into low dimensional features for input into the *C-RNN* network. We vary the sizes of FC units in the pre-trained classifiers based on the number of input features, f_{in} passed to the *C-RNN* network. The number of model parameters depends on the size of f_{in}. In our case, when $f_{in} = 10$, the number of model parameters $\alpha = 1402$, whereas when $f_{in} = 25$, $\alpha = 2287$. The number of model parameters, α is not impacted by the number of nodes in the network, N.

7.5.1.2 Tracking Steps

The distributed target tracking process has an in-built temporal component that considers a time dimension on how many steps to track a target motion on the tracking grid. During inference, our communicating RNN incorporates all the motion within a series of tracking steps, P. We evaluate the impact of varying the number of tracking steps setting $P = 8$, 12, and 16.

7.5.1.3 Multi-objective Optimization

Our approach is designed to implement multi-objective optimization on the prediction accuracy (position of digits on the tracking grid), network energy (activation of only relevant sensor nodes) and communication-cost (learning to communicate only relevant information for the tracking objective). At inference time, the model must learn to turn nodes into *on* or *off* position based on the position of the tracked target. The network must also optimize for communication by choosing when to communicate and what information to communicate as it improves its performance on the distributed target tracking objective. This approach to optimization allows the operator to choose amongst *low* (0 - 39 %), *medium* (40 - 69 %) or *high* (70 - 99 %) optimization on network energy and communication-cost based on available resources.

7.5.1.4 Hyper-parameters

We make a series of hyper-parameter choices for our model for evaluation. We do not necessarily tune our hyper-parameters as we are able to achieve desired performance with manual search based on our knowledge of the distributed target tracking within our architecture. In table 7.1, we highlight our model parameter choices.

Description	**Quality/Quantity**
Platform	PyTorch
Input Data	MNIST handwritten digits (1-9)
Input features	$f_{in} = 10, 25$
RNN layers	$l_r = 0.0005$, act = Linear, Relu
Net architecture	(,50), (50,8), (8,)
Hidden	4 layers
Weights	Shared
Optimizer	*Adam*
Training method	SDG
Loss	L2
Model parameters	$\alpha = 1402, 2287$
Number of nodes	$N = 16, 25$
Tracking steps	$P = 8, 12, 16$
Energy optimization	*low, medium, high*

Table 7.1: Model hyper-parameters

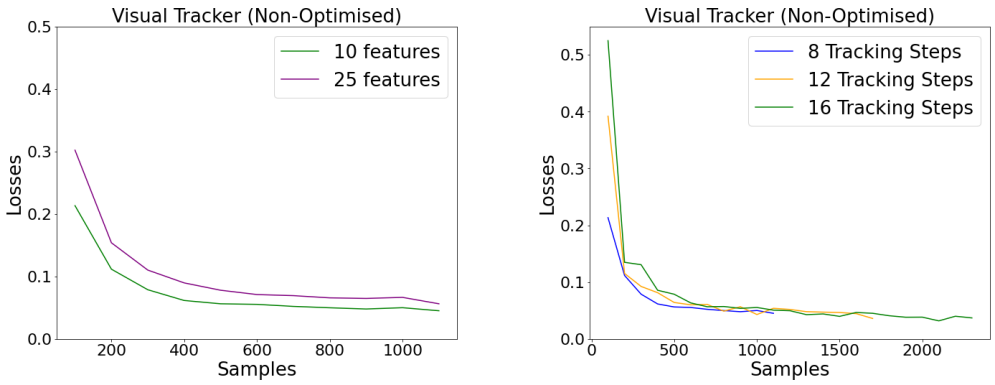
7.5.2 Results

Here we provide results on the empirical performance of our proposed approach on distributed target tracking. We also provide detailed analyses on the impact of network energy and optimization on overall predictive accuracy of the model. Further to this, our results demonstrate our framework's ability to scale to multi-node settings and the versatility of our architecture in different settings, whether visual or naive.

7.5.2.1 Learning Distributed Target Tracking

We first evaluate overall C^2RNN performance on distributed target tracking in a visual setting. In this setting, multiple nodes observe MNIST handwritten digits, extract features, decide when to activate (turn *on* or *off*) and communicate to cooperatively tracked the target digit in a resource-aware fashion. To evaluate predictive accuracy on the tracked objective, we consider two Visual Tracker models:

Non-Optimized Model: In this model, the network is assumed to have an unlimited network budget. In turn, it utilizes 99.9 % of its energy budget, (i.e., all nodes are set to ON position throughout the entire training and inference process).
Optimized Model: For the optimized model, the operator can tune parameters for the optimization factor, o that enables the network to optimize network energy and communication aggressively or non-aggressively. In this case the optimized model can achieve either one of three magnitudes of optimization, (i.e., *low, medium* and *high*. Figure 7.3 shows model convergence varying number of input features, f_{in} to the communicating RNN model and tracking steps, P.



(a) (b)

Figure 7.3: Training losses for the 16-node Visual Tracker model. In (a) we show model convergence when number of features fed into the communicating RNN model are varied with f_{in} = 10, 25 for tracking steps P = 8 with training samples, M_{train} = 150. In (b) the we show model convergence trends with varying tracking steps, P = 8, 12, 16.

We can see that both models converge to a good set of parameters quickly for the distributed target tracking task in a *non*-optimized scenario. In this scenario, the models have a complete view of the entire tracking grid at each time-step in the training process as nodes are constantly activated to *on* position. As such, the models achieve high predictive accuracy and is able to recurrently capture the motion of the tracked digit. Using more input features seems to perform slightly better with an accuracy of 0.292 (RMSE) for a model with f_{in} = 10 when compared to a model with f_{in} = 25 which achieves 0.303 (RMSE) in a system that implements P = 8 tracking steps. Consequently, a small sacrifice (3.6 %) in model accuracy means we can implement C^2RNN models with 38.7 % less parameters for the distributed target tracking task. By using RMSE for evaluation, we can assess model performance continuously and capture the underlying regression nature of learning emergent communication to solve

distributed target tracking cooperatively. We consider how much the model 'misses' the targets on the distributed tracking grid for evaluation. Table 7.2 summarizes performance results for the *non*-optimized model across varying tracking steps.

P	Energy used	α / f_{in}	Train RMSE	Test RMSE
8	19200	1402 / 10	0.268 (± 0.027)	0.303 (± 0.015)
		2287 / 25	0.228 (± 0.015)	0.292 (± 0.018)
12	28800	1402 / 10	0.276 (± 0.031)	0.329 (± 0.015)
		2287 / 25	0.247 (± 0.024)	0.321 (± 0.010)
16	38400	1402 / 10	0.275 (± 0.025)	0.319 (± 0.005)
		2287 / 25	0.230 (± 0.012)	0.315 (± 0.009)

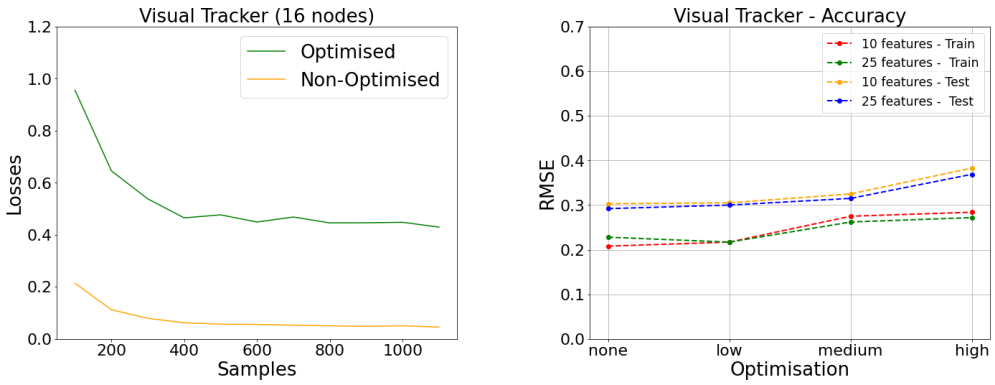
Table 7.2: Model performance results for a 16-node *non*-optimized system utilizing 100 % energy (all nodes in *on* position 99.9 % of the time). Energy Used is calculated as tracking steps, $P \times$ number of training samples, $M_{train} \times$ number of nodes, N in the system. When $P = 8$, maximum energy/energy totals = 19200. Model performance is given for $P = 8$, 12, 16 with $M_{train} = 150$ samples and input features, $f_{in} = 10$, 25. Results are averaged over 5 runs and standard deviation values for train and test RMSE given.

Across all tracking steps, on average, we note standard deviation of 0.022 on train RMSE and 0.012 on test RMSE results. The results show low variability proving relatively good model stability.

7.5.2.2 Learning Energy-efficient Optimization

Energy-efficient distributed inference is a critical aspect of cooperating nodes deployed in IoT settings. This is mainly because resource constrained (limited battery energy) IoT devices instantiated at the network edge must deliver high accuracy on predictive tasks over prolonged periods of time. We evaluate model performance in an experimental setup that actively optimizes for network energy whilst maintaining accuracy on the distributed target tracking objective. To assess energy impact in this section, we consider node activation patterns and in Section 7.5.4, we estimate the actual memory and energy consumption realised for inference on a single image on resource-constrained IoT devices using established energy metrics.

In our setup, we consider a measure of energy used during inference per node and across the overall system. The maximum energy totals for the system are is calculated as, number of tracking steps, $P \times$ number of training samples, $M_{train} \times$ number of nodes on the tracking grid, N. Figure 7.4 shows convergence and performance



(a) (b)

Figure 7.4: Training losses and model performance for the Visual Tracker in a 16-node system. In (a) we show training convergence for a *non*-optimized and an optimized model. The optimized model uses *high* optimization (39.9 % energy usage) with tracking steps, $P = 8$ and $f_{in} = 10$. In (b) model accuracy is plotted in terms of RMSE trends for $P = 8$, and varying levels of optimization. We compare train and test accuracy for $f_{in} = 10, 25$ on *non*-optimized (99.9 % energy usage), *low*-optimized (70 - 99 % energy usage), *medium*-optimized (40 - 69 % energy usage), *high*-optimized (0 - 39 %) models with $M_{train} = 150$.

results for *non*-optimized and optimized models. Evidently, *non*-optimized and *low*-optimized models outperform *medium* and *high*-optimized models. This is because the network makes a tradeoff on predictive accuracy and energy usage. Compared to the *non*-optimized model (the baseline in our case), the *high*-optimized model uses 2.5× less energy overall with only 8 % loss in accuracy. A *medium*-optimized model uses 1.5× less energy with a 2 % loss in accuracy for the same comparison. Not surprisingly, a *low*-optimized model achieves roughly similar performance (0.3 % difference) as a *non*-optimized model.

Table 7.3 presents results for our framework with models trained and performing distributed inference and activating nodes based on automatically learned energy schemes (*low, medium*, and *high* energy optimization). These results are encouraging as they show that our C^2RNN framework can automatically discover distributed communication patterns and accurately tracks a target in an energy-efficient manner. Hence, our data-driven neural network based approach can be distributed across multiple cooperating embedded IoT edge devices for distributed target tracking tasks without the need for specialized energy management schemes, as it can model and learn to automatically optimize for energy usage based on objective tasks being modeled. The varying levels of automatically learned network energy optimization for the models are shown in Figure 7.5.

Optimization	α / f_{in}	Energy Usage	Energy %	Train RMSE	Test RMSE
low	1402 / 10	16152	84.1 %	0.217 (± 0.021)	0.305 (± 0.013)
	2287 / 25	13473	70.1 %	0.217 (± 0.018)	0.300 (± 0.013)
medium	1402 / 10	12708	66.1 %	0.275 (± 0.020)	0.325 (± 0.013)
	2287 / 25	10225	53.2 %	0.262 (± 0.028)	0.315 (± 0.008)
high	1402 / 10	7667	39.9 %	0.284 (± 0.018)	0.383 (± 0.018)
	2287 / 25	5776	30.0 %	0.272 (± 0.019)	0.369 (± 0.014)
none	1402 / 10	19184	99.9 %	0.268 (± 0.027)	0.303 (± 0.015)
	2287 / 25	19184	99.9 %	0.228 (± 0.015)	0.292 (± 0.018)

Table 7.3: Model performance results for a 16-node optimized system based on *high* (0-39 % energy usage), *medium* (40-69 % energy usage) and *low* (70-99 % energy usage). Energy Usage is calculated as tracking steps, $P \times$ number of training samples, $M_{train} \times$ number of nodes, N in the system. In this case, $P = 8$ and maximum energy/energy totals = 19200 units. Model performance is compared with $M_{train} = 150$ samples for input features, $f_{in} = 10, 25$. Results are averaged over 5 runs and standard deviation values given.

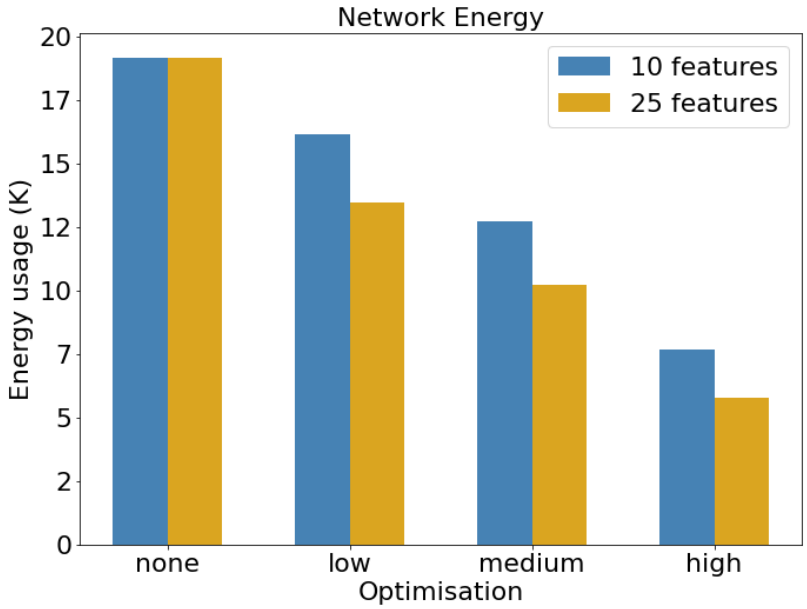


Figure 7.5: System-wide network energy usage totals showing models with none, low, high and medium energy optimization. The maximum energy usage limits in a 16-node system when $P = 8$ is 19200.

We can observe that the models with $f_{in} = 10$ (1402 parameters) and $f_{in} = 25$ (2287 parameters) are light-weight and can be deployed for inference on a resource constrained IoT edge device with 500kB memory as in [30]. In Figure 7.6, we

show activation patterns for nodes on tracking grids based on *low*, *medium* and *high* optimization. Evidently, we can see how the network adapts to the varying network energy optimization levels as it learns to distributively track the moving target digit. In *medium*-optimized and *high*-optimized model, we can observe grids in which the model fails to detect the target as it turns nodes *on / off* during inference.

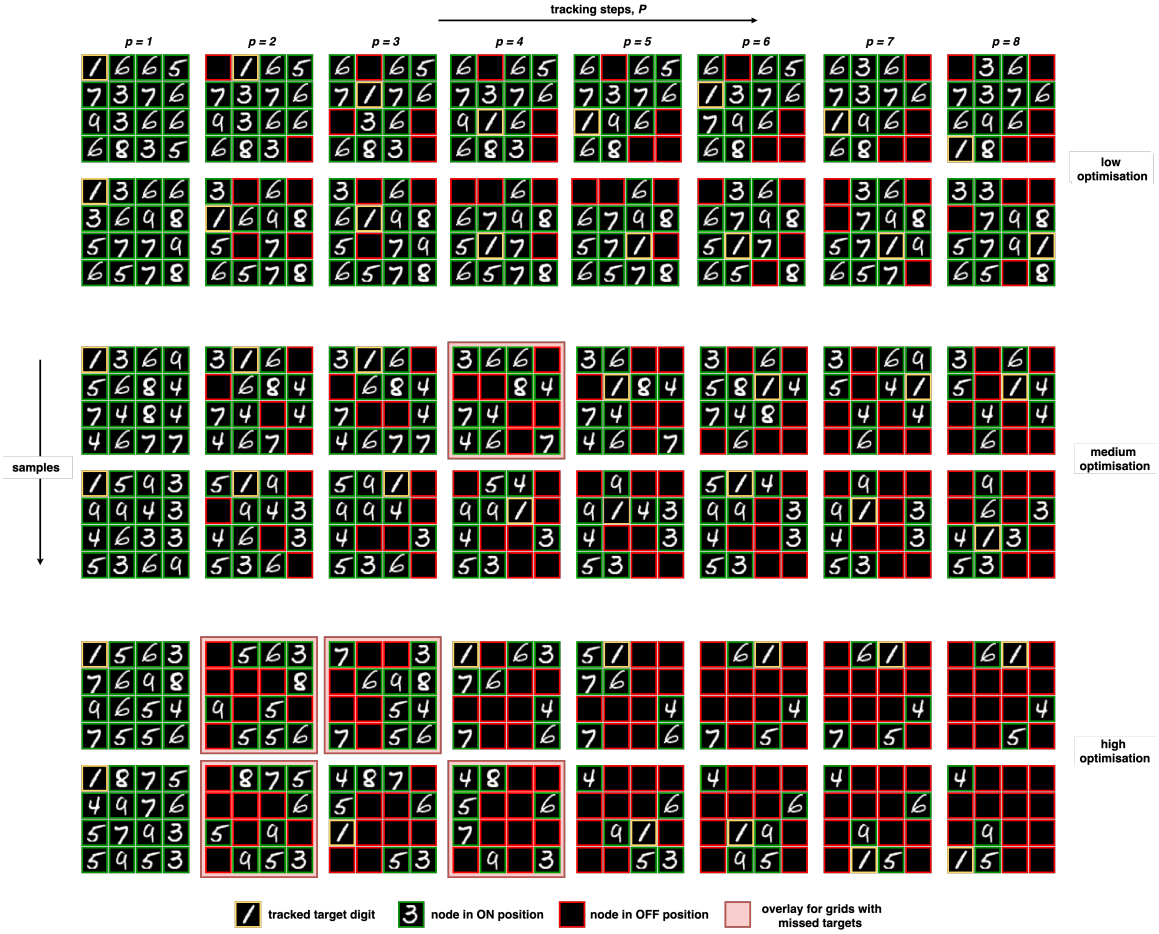


Figure 7.6: Tracking grids for *low*-optimized (84.1 % energy usage), *medium*-optimized (66.1 % energy usage) and *high*-optimized (39.9 % energy usage) models with $f_{in} = 10$. We show a tracked MNIST hand-written digit for tracking steps, $P = 8$ and two samples per model. All nodes in the tracking grids are initialized with the *on* position at $p = 1$. Over time, the network learns to turn nodes *on / off* based on the position of the tracked target and optimization on the network's energy.

7.5.2.3 Learning Communication-efficient Optimization

We have shown that our C^2RNN framework can achieves desirable performance on the distributed target tracking task and learns to activate relevant nodes during inference, thereby optimizing energy with respect to the objective task. To perform

distributed inference for the target tracking task, multiple nodes in our framework must learn when and what to communicate in a manner that respects constraints on their energy usage. We evaluate our model for communication efficiency by penalizing nodes for unnecessary communication based on number of communicated messages, n_{comm} during inference. Further to this, we constrain the communication channel size, $v = 1$. Thus, each node can only communicate a single message at any given communication timestep. Our C^2RNN is able to minimize communication, thereby achieving a tradeoff between the number of messages communicated across all nodes and predictive accuracy. Figure 7.7 compares performance in *low* and *high* energy optimized models. Only activated nodes can choose whether to communicate or not.

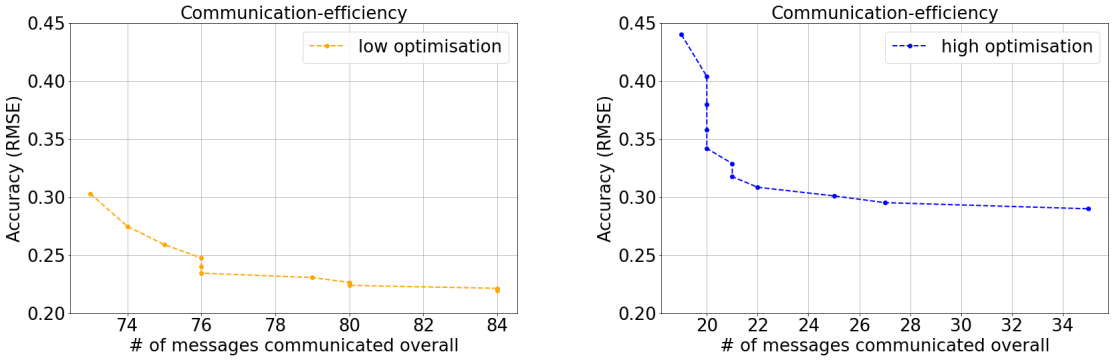
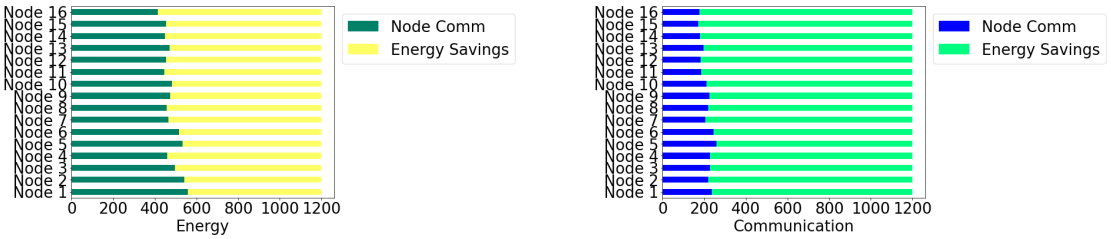


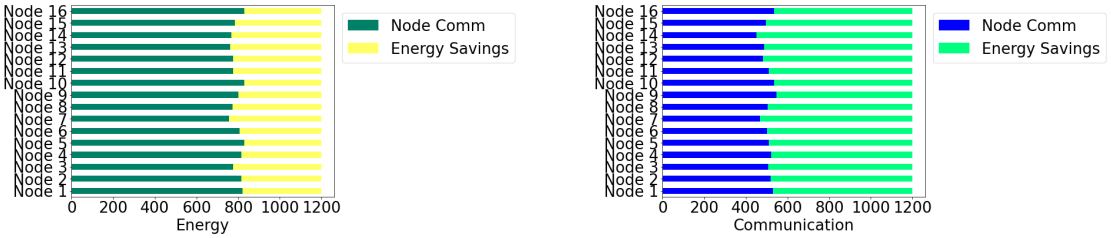
Figure 7.7: Model performance for a 16-node system with $f_{in} = 10$ showing number of messages communicated overall across all nodes on the distributed tracking task with (a) *high* and (b) *low* energy optimization. With each forward push, the system's maximum number of messages communicated is pegged at number of nodes N $\times$ tracking steps, P. In this case the model's maximum communication, is $(P = 8) \times (N = 16) = 128$.

The *low*-optimized model achieves 72.4 % communication whereas the *high*-optimized model achieves 43.2 % communication. Compared to the *non*-optimized baseline model, the *low*-optimized model uses 1.4× less communication with only 0.3 % reduction in accuracy. Making similar comparison, the *high*-optimized uses 5× less communication with 8 % reduction in accuracy. Our results demonstrate C^2RNN's ability to make communication and accuracy tradeoffs, thereby presenting opportunities for the operator to choose amongst varying degrees of communication-efficiency needed whilst bench marking against accuracy needed for the application task. In Figures 7.8, 7.9, 7.10 we can visualize system wide communication in relation to network energy usage for varying levels of optimization. The model learns to optimize multiple objectives (network energy and communication) simultaneously.



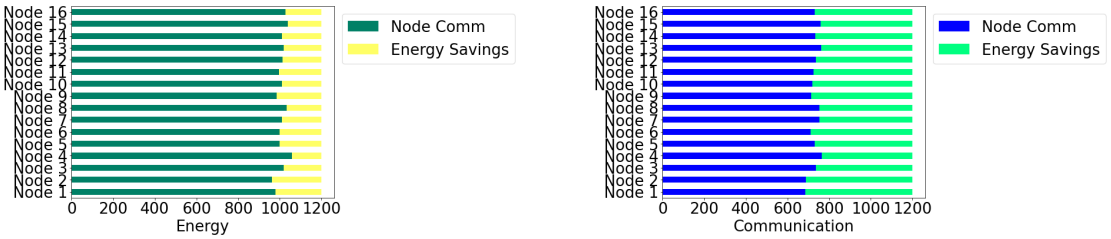
(a) Energy (b) Comms

Figure 7.8: In (a) we show overall system node energy usage. In (b), we show overall system communication. The model is trained with $P = 8$, $f_{in} = 10$, uses *high* energy optimization with 39.9 % energy usage and achieves a 0.383 test RMSE.



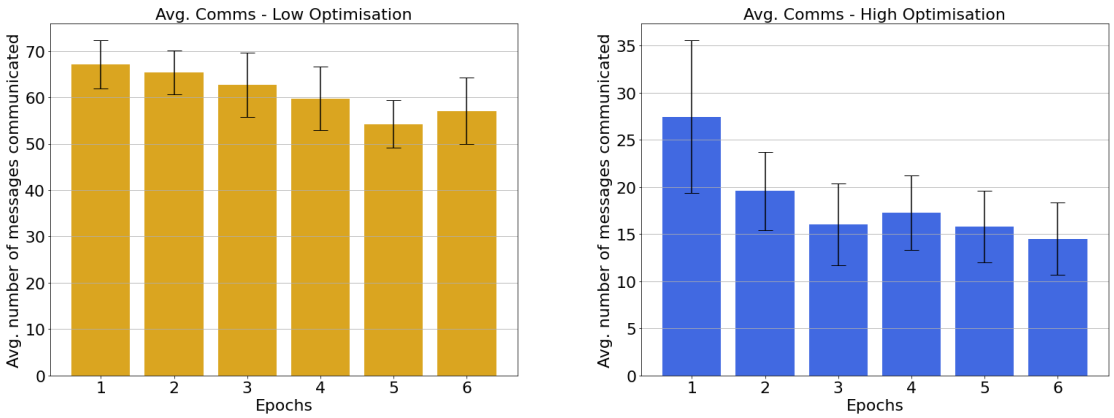
(a) Energy (b) Comms

Figure 7.9: In (a) we show overall system node energy usage. In (b), we show overall system communication. The model is trained with $P = 8$, $f_{in} = 10$, uses *medium* energy optimization with 66.1 % energy usage and achieves a 0.325 test RMSE.



(a) Energy (b) Comms

Figure 7.10: In (a) we show overall system node energy usage. In (b), we show overall system communication. The model is trained with $P = 8$, $f_{in} = 10$, uses *low* energy optimization with 84.1 % energy usage and achieves a 0.305 test RMSE.



(a) (b)

Figure 7.11: Average number of messages communicated with increasing training epochs for models with (a) *low* and (b) *high* network energy optimization. Results are given for $N = 16$, $P = 8$ and $f_{in} = 10$.)

In Figure 7.11, we show our C^2RNN's ability to iteratively communicate less with training progression. Our results shows how the model intuitively deactivates nodes in the tracking grid that are not in good proximity to the tracked target. For activated nodes, our approach presents opportunities to optimize for communication-cost, thereby respecting preset communication budgets. As such, we avert the need for complex pre-defined, hand-engineered communication protocols as the model can learn emergent communication based on the distributed task.

7.5.2.4 Node Scalability

Scaling nodes in distributed settings can have significant impact on network resources due to inefficient energy optimization, computational complexity and communication overhead. Typically, distributed target tracking applications provide coverage for a wide geographical area. Such a scenario could have hundreds of sensing nodes (i.e., cameras) cooperatively performing distributed tracking tasks using on-device intelligent algorithms.

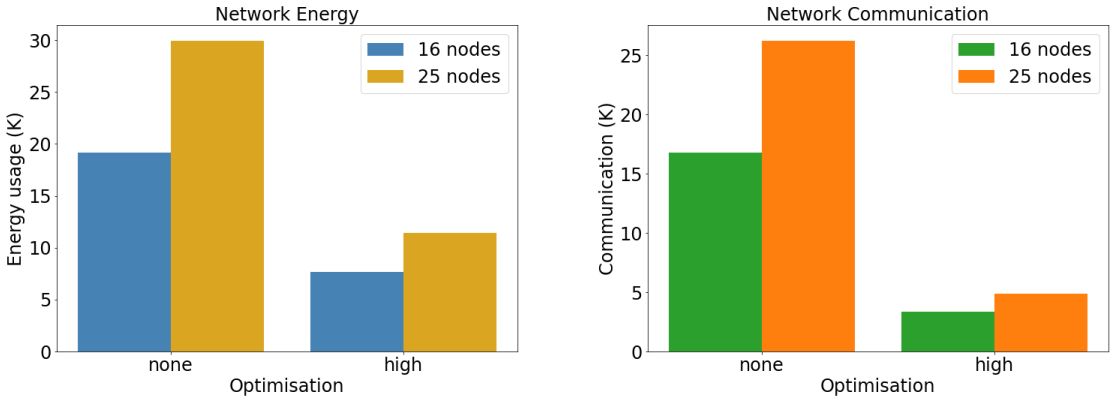
Training multiple nodes for this task is hard and requires massive amounts of computer resources. In order to explore scalability across multiple nodes, we setup an experiment using $N = 16$, 25 for the distributed target tracking task.

Our C^2RNN framework is able to achieve good accuracy on both 16-node (4×4 tracking grid) and 25-node (5×5 tracking grid) systems. Table 7.4 shows results for *non*-optimized and *high*-optimized models in 16-node and 25-node systems. The 25-node system achieves better accuracy by 0.2 % for *non*-optimized model and 2.8

N	Energy	Energy %	Comms	Comms %	Train RMSE	Test RMSE
16	7667	39.9 %	3374	44 %	0.284 ($\pm$ 0.018)	0.383 ($\pm$ 0.018)
	19184	99.9 %	16784	87.4 %	0.268 ($\pm$ 0.027)	0.303 ($\pm$ 0.015)
25	11418	38.0 %	4837	42.3 %	0.276 ($\pm$ 0.023)	0.355 ($\pm$ 0.018)
	29975	99.9 %	26225	87.4 %	0.258 ($\pm$ 0.026)	0.301 ($\pm$ 0.006)

Table 7.4: Summary of model performance results for $N = 16, 25$, with $P = 8$, $f_{in} = 10$ and $M_{train} = 150$. For both $N = 16, 25$, the model is evaluated with high energy optimization. The *non*-optimized models have 99.9 % energy usage whereas the *high*-optimized models achieve 39.9 % energy usage (when $N = 16$) and 38 % energy usage (when $N = 25$). Results are averaged over 5 runs and standard deviation values given.

% for the *high*-optimized model. Notice that the performance difference is marginal. This suggests that our *C-RNN* can easily scale to multiple nodes.



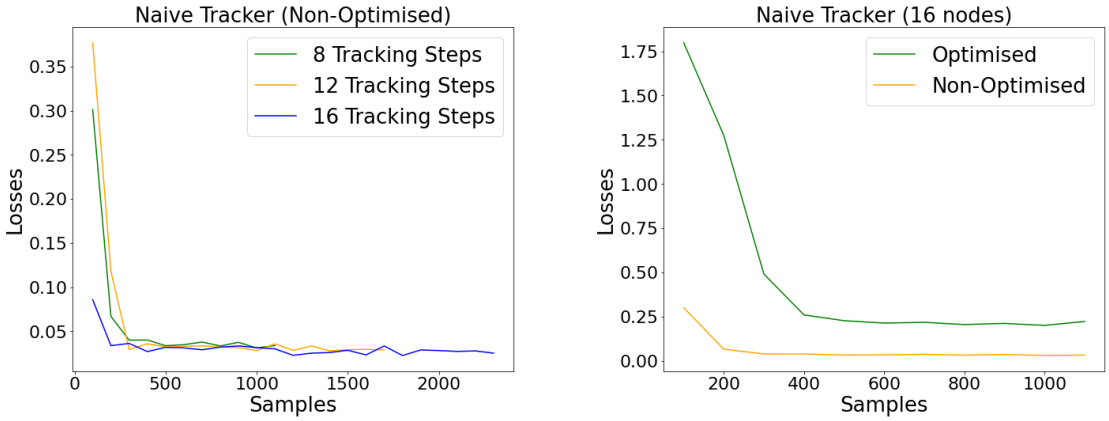
(a) (b)

Figure 7.12: System-wide network (a) energy usage and (b) communication totals showing *non*-optimized and *high*-optimized models with $P = 8$ and $f_{in} = 10$. Maximum energy usage is 19200 units (16-node system) and 30000 units (25-node system).

In Figure 7.12, we show system-wide energy usage and communication on *non*-optimized and *high*-optimized models with $N = 16, 25$. For our distributed target tracking task, the *non*-optimized always activates all nodes whilst the *high*-optimized model aggressively decides which nodes to set to *off* position based on its understanding of the position of the tracked target. Evidently, the amount of overall system-wide energy used, and number of messages communicated scales linearly with increasing nodes. Nonetheless, enforcing varying levels of network energy optimization enables the operator to influence degrees at which network energy usage (and communication) scales with increasing network nodes.

7.5.2.5 Extending C^2RNN to Naive Settings

We have shown that our cooperative, grid-based, distributed target tracking C^2RNN framework can be used to extract features from MNIST digits, employ a communicating neural network architecture to discover emergent communication patterns amongst distributed nodes and achieve high accuracy on the distributed target tracking task albeit in a visual setting. Next, we extend our framework to a naive setting (see section 7.3) and assess its efficacy when we use non-visual inputs to the model. In this setting, our input data is composed of real-valued (integers in the range 1-9) directly fed to the communicating RNN model. Unlike in the visual setting, we do not need to use CNN-based feature extractors on our input.



(a) (b)

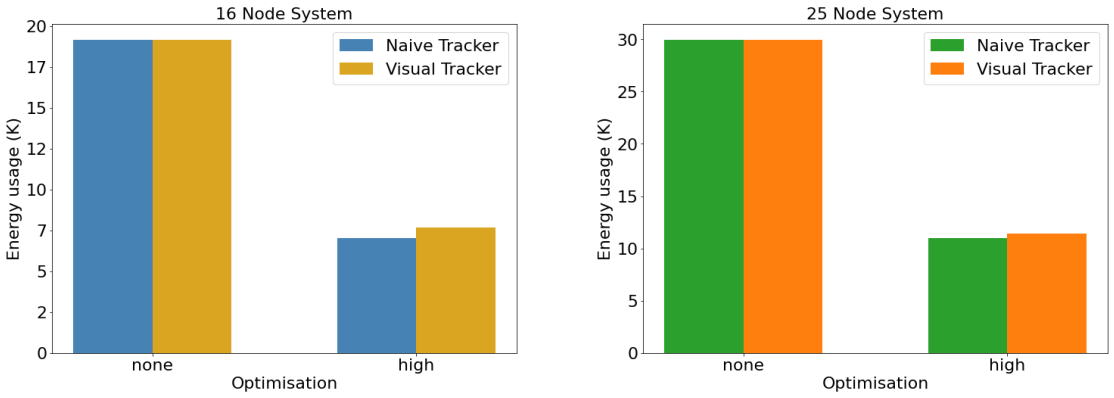
Figure 7.13: Training losses for the 16-node Naive Tracker model. In (a), we show model convergence trends with varying tracking steps, $P = 8, 12, 16$. In (a) we show training convergence for a *non*-optimized and an optimized model. The optimized model uses *high* optimization (36.6 % energy usage) with tracking steps, $P = 8$ and $M_{train} = 150$.

From Figure 7.13, we can see convergence trends for the distributed target tracking task in the *non*-optimized model with different values of P. The model incorporates temporal dynamics for the tracked target over P tracking steps and quickly converges for all values of P. We can also observe training convergence comparing a *non*-optimized model and a *high*-optimized model (with 36.6 % energy usage). As expected the optimized model takes longer to converge as lesser nodes are activated during training. Although, *high* optimization drastically conserves network energy, predicting the tracking grid with the location of the tracked target is generally harder. In Table 7.5, we present model performance results for the Naive Tracker and compare it to the Visual Tracker.

Tracker	Optimization	N	Energy %	Comms %	Train RMSE	Test RMSE
Naive	*high*	16	36.6 %	52 %	0.303 ($\pm$ 0.017)	0.229 ($\pm$ 0.014)
		25	36.7 %	36.2 %	0.292 ($\pm$ 0.022)	0.205 ($\pm$ 0.010)
	none	16	99.9 %	87.4 %	0.203 ($\pm$ 0.013)	0.183 ($\pm$ 0.001)
		25	99.9 %	87.4 %	0.222 ($\pm$ 0.012)	0.158 ($\pm$ 0.002)
Visual	*high*	16	39.9 %	44 %	0.284 ($\pm$ 0.018)	0.383 ($\pm$ 0.018)
		25	38.0 %	42.3 %	0.276 ($\pm$ 0.023)	0.355 ($\pm$ 0.018)
	none	16	99.9 %	87.4 %	0.268 ($\pm$ 0.027)	0.303 ($\pm$ 0.015)
		25	99.9 %	87.4 %	0.258 ($\pm$ 0.026)	0.301 ($\pm$ 0.006)

Table 7.5: Summary of model performance for $N = 16, 25$ in naive and visual tracking settings with $P = 8$ and $M_{train} = 150$. The Naive Tracker achieves 36.6 % (16-node system) and 36.7 % (25-node system) energy usage whereas the Visual Tracker uses $f_{in} = 10$ and achieves 39.9 % (16-node system) and 38 % (25-node system) energy usage for the *high*-optimized case. Results are shown for *high*-optimized and *non*-optimized models. Results are averaged over 5 runs and standard deviation values given.

In Figure 7.14 we show the varying network energy optimization levels achieved by both the Naive Tracker and the Visual Tracker. As can be seen, both trackers achieve similar optimization levels. Thus we can fairly compare model accuracy for both trackers.



(a) (b)

Figure 7.14: System-wide network energy usage in (a) 16-node system and (b) 25-node system showing *non*-optimized and *high*-optimized models with $P = 8$ and f_{in} 10 (for Visual Tracker). Maximum energy usage is 19200 units (16-node system) and 30000 units (25-node system).

Our framework achieves higher accuracy for the naive setting using 16-node and

25-node systems. This intuitively makes sense as the Naive Tracker is solving a simpler problem. From these results, we can conclude that our framework can be extended to various problem domains.

7.5.3 Computational Complexity

Most embedded IoT devices have significant constraints on available memory and computational resources. Instantiating deep networks on these devices can imbue them with intelligence for distributed inference, albeit at a cost.

Op	Params, α (K)	FLOPs (K)/ %
Conv2d-1	0.26	1198.08 / 30.28
MaxPool	-	34.56 / 0.87
ReLu	-	23.04 / 0.58
Dropout/Conv2d-2	5.02	2560.0 / 64.7
MaxPool	-	7.68 / 0.19
ReLu	-	5.12 / 0.13
Linear-1	16.05	128.0 / 3.24
Totals	21.33	3956.48 / 100

Table 7.6: Computational overhead of a single feature extractor applied on a single node at inference time over tracking steps, $P = 8$. Model parameters, α are given for extracting features for a single image passed as a 4-D tensor on a single node. FLOPs are estimated for a sequence of 8 images in a single inference step.

We quantify computational overhead for our C^2RNN framework for the distributed target tracking task. We estimate the size of our distributed *C-RNN* framework using the number of parameters α and required computations for inference in terms of FLoating-point OPerations (FLOPs)[1].

Our distributed target tracking framework employs feature extractors similar in architecture as in 6.3.2. Table 7.6 quantifies the computational cost incurred on a single feature extractor per forward pass, over P tracking steps.

We can observe that feature extraction kernels in the convolutional layers take up more computational resources (94.98 %) as compared to the fully connected layer (3.24 %). Nonetheless, we can also see that the convolutional layers use $3\times$ less parameters making them memory efficient as they only take up 0.16 MB of memory. Increasing P

[1]We consider a FLOP as half a Multiply-Add operation. Thus, a single Multiply-Add is 2 FLOPs

does not affect model parameters, but leads to an increase in FLOPs used for inference in a single forward pass. In Table 7.7 we show the layer-by-layer computational overhead for the communicating RNN model independent of the feature extractors. Model parameters and FLOPs scale with the dimensionality of the input. In our case, a model using $f_{in} = 10$ has 1.6× less parameters than a model with $f_{in} = 25$. FLOPs scale with tracking steps, P.

	$f_{in} = 10$		$f_{in} = 25$	
Op	α (K)	FLOPs (K)	α (K)	FLOPs (K)
Linear-1	0.85	6.4	1.6	12.4
ReLu	-	0.4	-	0.4
Linear-2	0.408	3.2	0.408	3.2
ReLu	-	0.064	-	0.064
Linear-3	0.144	1.024	0.279	1.984
ReLu	-	0.128	-	0.248
Totals	1.402	11.216	2.287	18.296

Table 7.7: Computational overhead for the *C-RNN* model per layer. Model parameters, α are given for $f_{in} = 10$, 25 respectively and FLOPs are estimated for a single forward push over tracking steps, $P = 8$ at inference time on a single node in a 16-node system.

To provide a holistic, system-wide view of computational overhead, we quantify α and FLOPs for the unified *C-RNN* (in naive settings) and C^2RNN models (in visual settings). Figure 7.15 shows FLOPs for 16-node and 25-node systems for the Visual Tracker. We observe that the 16-node and 25-node models with varying f_{in} use FLOPs in the same magnitude, with small difference in the computational overhead. Scaling from 16 to 25 network nodes results with $f_{in} = 10$ yields a 2.4 % increase in computational overhead and a 0.2 % gain in model accuracy.

In Figure 7.16 we can see that the Visual Tracker incurs significantly more computational cost (150×) compared to the Naive Tracker. This is mainly because of the cost of feature extraction in the visual setting.

These computational costs can be avoided by outsourcing feature extraction to a centralized compute resource (i.e., a Cloud) or employing quantization and model compression techniques [30, 222]. Nonetheless, we can see that the unified C^2RNN network uses less than 24 K parameters. Moreover, it is important to note that

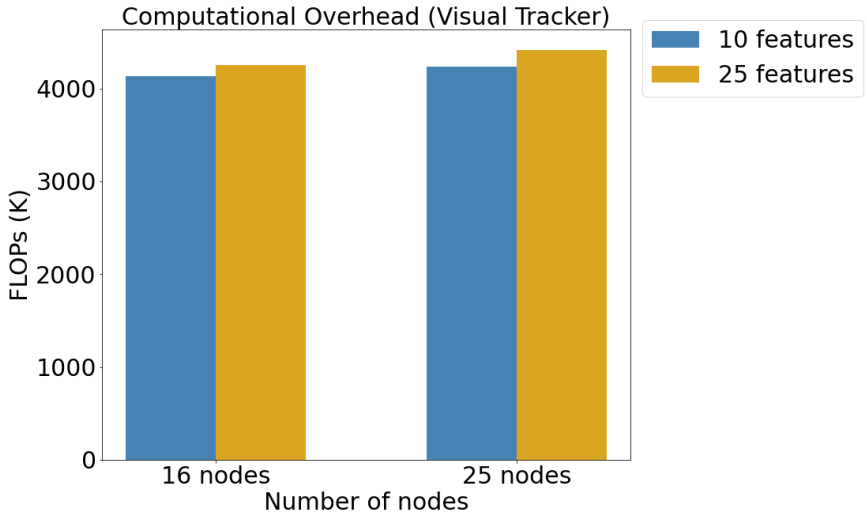


Figure 7.15: Total computational overhead across all system nodes in 16-node and 25-node systems at inference time with $f_{in} = 10, 25$. FLOPs are estimated as a sum of FLOPs necessary for feature extraction during a single forward push and the product of N and FLOPs per forward push in the communicating RNN.

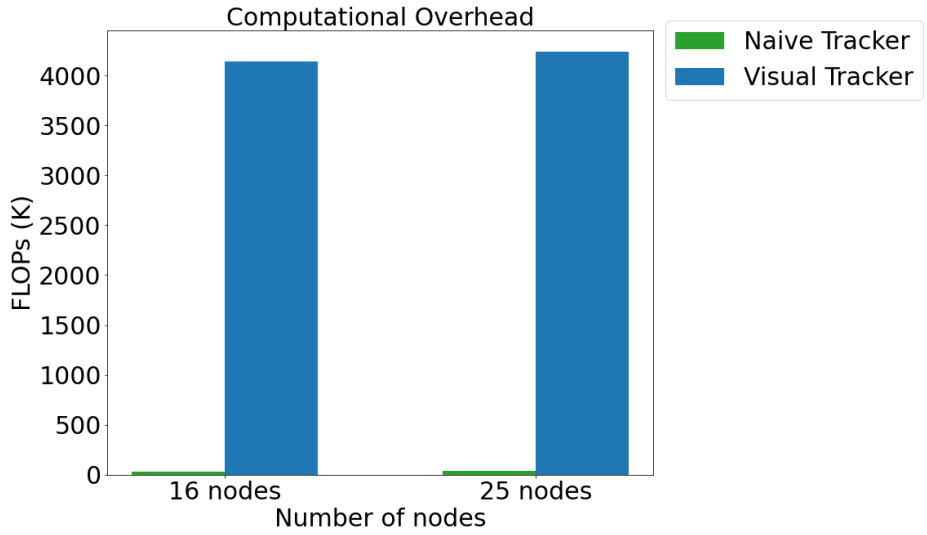


Figure 7.16: Total computational overhead across all system nodes in 16-node and 25-node systems at inference time with $f_{in} = 10$ (for the Visual Tracker) and $f_{in} = 10$ (for the Naive tracker). Feature extractors in the visual setting account for high computational overhead.

the computational overhead as given in Table 7.8 assumes a *non*-optimized model. Clearly, with network energy optimization (*low*, *medium* or *high*) we can further reduce computational overhead across the entire system. Quantifying computational overhead with this assumption (a *non*-optimized model) is sufficient to evaluate our C^2RNN framework for computational efficiency as we consider maximum possible

Tracker	f_{in}	N	Params (K)	FLOPS (K)	Test RMSE
Naive Tracker	6	16	0.212	27.136	0.183 ($\pm$ 0.001)
		25	0.212	42.4	0.158 ($\pm$ 0.002)
Visual Tracker	10	16	22.732	4135.936	0.303 ($\pm$ 0.015)
		25	22.732	4236.88	0.301 ($\pm$ 0.006)
	25	16	23.617	4249.216	0.292 ($\pm$ 0.018)
		25	23.617	4413.88	0.284 ($\pm$ 0.010)

Table 7.8: Computational overhead and model performance for the unified C^2RNN model on the distributed target tracking task with $N = 16$, 25 in 4×4 and 5×5 grids respectively. FLOPs are given for inference over tracking steps, $p = 8$. The number of FLOPs scales with increasing nodes N and tracking steps,P. Results are averaged over 5 runs and standard deviation values given.

FLOPs at inference time.

7.5.4 Memory and Energy Consumption

In our problem setting, network node activation patterns have a direct impact on the overall system energy used. Actual memory and energy used depends on various factors that we have discussed in previous sections, i.e., cost of computation (FLOPs and memory accesses), cost of intra-node and inter-node communication and provisions of the hardware platform in question. In Chapter 6, Section 6.4.2.5, we outlined our approach for estimating actual energy and memory consumed by our C^2RNN framework. Our approach formulates the overall network energy during inference as a sum of the energy cost of computation and energy cost of communication. Table , provides overall network energy consumption and model memory for our C^2RNN compares them to state-of-the-art models trained for MNIST image classification and inference on resource-constrained IoT devices. We assume that the system has 100 % utilisation of its energy budget (all nodes are activated) for the estimates.

Similar to the lessons in Chapter 6, Section 6.4.2.5, the energy cost of communication only constitutes a small fraction (5 % in $Visual - C^2RNN - 1$) of the overall energy consumption as expected. Compared to our C^2RNN framework for learning distributed communication in visual settings, the target tracking task described in this chapter consumes $4\times$ more energy. The same comparison is true for the state-of-the-art approaches [78, 110] in Table 7.9. This is a result of the target tracking component which requires inference over a number of tracking steps, so naturally the computational cost is $\approx 5\times$ more. Looking at the memory requirement, we can

Model	Params, α (K)	Model Size (kB)	Total Energy (mJ)
$Visual - C^2RNN - 1$	22.732	85.25	8.31
$Visual - C^2RNN - 2$	23.617	88.57	9.66
Bonsai [110]	21.000	84.00	1.27
ProtoNN [78]	15.957	63.83	2.00

Table 7.9: Energy Consumption and memory requirements for our C^2RNN models compared with state-of-the-art implementations of Deep Networks for inference on a single image on resource-constrained IoT devices over $p = 8$ tracking steps. $VisualC^2RNN - 1$ is a lighter model with $f_{in} = 10$, and 16 nodes whereas $VisualC^2RNN - 2$ is a heavier model with $f_{in} = 25$ and 25 nodes.

notice that our C^2RNN models are small enough for storage on-chip and they do not require significant amounts of energy for inference, making them amenable for implementation on Class 2-3 IoT devices (refer to Table 2.2).

7.6 Discussion

In this chapter, we proposed a cooperative, grid-based, distributed target tracking framework that extends our C^2RNN architecture. Distributed target tracking in multi-node scenarios is a challenging task as the network must accurately track the position of a moving target and perform multi-objective optimization concurrently. Our framework is able to jointly perform target tracking and image classification in visual settings with MNIST handwritten digits. Further to this, we showed how our proposed framework can be extended to naive settings with sensor-like data. We have shown that in both visual and naive settings, our framework can achieve high accuracy on the tracking objective whilst optimizing for network energy and communication-costs with low standard deviation on train and test RMSE results. This is encouraging as our network can stably and automatically learn to adapt to motion patterns of tracked objective via distributed communication in a manner that respects resource constraints on its energy budget and computational resources. In many real-world scenarios, operational budgets on network energy are constantly changing. In our approach, the operator can choose varying levels of network-wide energy optimization, (i.e., *high*, *medium* and *low*). This allows the owner to effect necessary control as network energy optimization could be impacted by specific application scenarios and tracking accuracy goals. Nonetheless, a notable limitation with our approach is

that it employs multi-objective optimization based on loss gradients. Although this method is simple to implement, it is likely to scale poorly as the number of nodes in the network increases. This is because the dimensionality of the gradients increases which leads to sparse events (like true target detection) being swamped by common events. This issue can be resolved to an extent through network segmentation or clustering to limit the maximum effective network size. Alternatively, it does suggest rich opportunities for exploring how to perform multi-objective optimization in a robust and scalable way for these distributed cooperative inference tasks.

We considered network energy optimization from the perspective of node activation during inference and thoroughly quantified network energy based on various levels of optimization. In addition to this, we estimated energy consumption using the approach described in the previous Section 6.4.2.5 considering our inference platform. The memory and energy requirement of our model is comparable, in terms of magnitude, to state-of-the-art models implemented on resource-constrained IoT devices.

Although we have discussed offline, centralized training, it is also interesting to consider in-network training. However, training the model on multiple nodes, (e.g., a few hundreds) in resource constrained IoT settings is challenging as embedded IoT devices typically have limited memory (i.e., less than 500kB). As such, training methods must consider both the computational overhead (FLOPs) of gradient evaluation and the memory (SRAM) for storing parameters. Deploying highly accurate deep network models in a resource-aware fashion warrants constant design considerations and tradeoffs that make it challenging to train models directly on end devices to solve complex tasks, (e.g., distributed, cooperative multi-object tracking). We highlight this as another rich avenue for future exploration.

In the next chapter, we conclude this thesis by providing a summary of its contributions and a discussion of future work.

Chapter 8

Conclusion and Future Work

Deep Networks have proven their efficacy across various problem domains, i.e., natural language processing, computer vision, speech recognition, and time-series modelling and signal processing. Nonetheless, these networks have huge computational demands amongst other issues, which pose a critical bottleneck for implementation on embedded IoT devices at the network edge. In this section. we summarize the key challenges identified in this thesis and research questions we sought to address:

- **Hardware Constraints** Most embedded IoT end devices have stringent limits on memory capacity i.e., a standard Class 2 IoT device running on an ARM Cortex M4F processor only has about 256Kb of memory. Flash memory limitations affect the amount of data that can be read stored on a chip whereas ROM memory impacts the size and complexity of the transformation function that can be stored on a chip. In turn, this affects the classes of problems some IoT end devices can handle, for instance, some chips are capable of housing algorithms that can perform image classification on-device whereas other can only perform basic functions like, linear regression on provided data. Moreover, embedded-networked systems are expected to have long system lifetimes, yet their most precious resource, i.e., battery capacity is constrained. For these devices, we must consider transmission and computation energy as they directly impact system lifetime. Since these systems, are characterized with sensors collecting large amounts of data, the data size and bandwidth at each node also impacts transmission and and computational energy spent in handling a workload. Resultantly, these devices must operate at low frequencies (8-100 MHz) to preserve energy. Energy consumption is important as it is directly tied to cost. Replacing batteries on sensor nodes is in most cases not sustainable or difficult, especially in hostile environments. Programmability of embedded, networked

IoT systems is another challenge as these systems are tedious to implement and design. This is mainly because network design must always take into account the resource constraints on end devises. For instance, efficient communication protocols can realize notable energy efficiency gains. The optimal design of complex, hand-engineered communication protocols is a tedious exercise and such protocols often fail to adapt to dynamic network conditions. Consequently, this remains a longstanding challenge. Some IoT end devices have serious limitations on available communication technologies which result in connectivity, bandwidth and latency issues and underlying communication protocols often fail to deliver system-wide reliability due to both intermittent and continuous link failure. [193] provide a more detailed survey of challenges limiting end devices within embedded IoT networked systems. These challenges motivate our choice to explore the implementation of Deep Networks in distributed IoT scenarios. We asked, *How can we exploit Deep Learning to provide automated distributed inference for Resource Constrained Settings?* These networks can extract meaningful information from huge amounts of data collected on sensor nodes and require minimal coding as they can learn dependencies from data. This approach also mitigates security and privacy risks associated with over-reliance on Cloud infrastructure. For communication cost, which is a major focus of this thesis, having communication-aware Deep Networks immediately alleviates the need for complex, non-adaptive, hand-engineered protocols. Moreover, Deep Networks can learn to optimize communication in a task-agnostic way.

- **Modelling Challenges** Since [30] survey some challenges associated with instantiating Deep Networks on IoT devices in depth, we will only provide a summary focusing on the components we address in this work. The quality of a Deep Network is evaluated by how accurate it is on a prediction task. For implementation in IoT settings, developers must achieve a trade-off between computational and energy costs, and model accuracy. Highly accurate models typically require huge datasets to train and have millions of parameters to learn the underlying patterns in data being pushed into the models. Most of these models are not suitable for IoT settings and several tricks must be employed to make them suitable for instantiation on end devices with resource constraints on their memory sizes and energy budgets. These methods, e.g., model compression, quantization, and weight sparsification, aim to drastically

reduce the size of these models without sacrificing accuracy on prediction tasks. Processors on IoT end devices, must be capable to support different Deep Network architectures with varying layers and parameters. As the dimensionality of data use increases, so does the computation at inference time. To achieve distributed inference on multiple, interconnected, networked nodes, communication is necessary. Distributed nodes must communicate efficiently to achieve their modelling task whilst respecting computational and energy constraints on end devices. Connected IoT devices e.g., wearables, smarthome cameras etc., collect personal data, making data at the network edge increasingly personal. Transmitting this data to the cloud for processing often presents privacy and security risks motivating the need to analyze and process these data locally. Deep Networks operating at the network edge, must be designed for streamlined training, weights updating and robust inference locally on end devices whilst preserving privacy. Motivated by these challenges, we considered the research questions, *(1) How best can we design a Deep Network-based architecture suitable for distributed inference and extendable to resource-constrained IoT devices?*, *(2) How do we optimize our model across multiple problem domains, in a way that makes the model task-agnostic?*, *(3) How do we achieve model robustness and scalability in the case of multi-node scenarios and failing nodes?*.

To address some of these challenges, we studied the problem of achieving automatically learned communication for distributed inference in resource-constrained systems using Deep Networks. Achieving energy-efficient communication is a long-standing challenge in resource-constrained IoT systems (e.g., Wireless Sensor Networks, Smart environments etc.) that continually pervade our physical world. This issue is further exacerbated by sophisticated deployments of heterogeneous, embedded sensing devices which must perennially communicate via wireless links to deliver quality services whilst meeting stringent operational targets. Machine Learning and Artificial intelligence have celebrated successes in providing self-organization, versatility, predictive, and inference capabilities thereby providing promise for instantiation in resource-constrained IoT settings. Our work shows that communication amongst multiple intelligent distributed nodes in a network can be modelled as a learning problem. In Section 8.1, we detail our contributions motivated by the challenges and research questions we have so far discussed. We then provide a discussion of future research directions in Section 8.2

8.1 Summary of Contributions

In this research we explore whether we can exploit Machine Learning to achieve automated distributed inference via automatically learned communication protocols in resource constrained systems. In doing this, are able to summarize the novel contributions of this thesis:

1. We explored a Deep Network-based In-network processing paradigm in which Deep Networks are implemented in Mist Computing in Chapter 2. Traditionally, computationally intensive ML algorithms are constrained to Cloud systems as embedded sensing devices are characterized by limited to none intelligence. In the Deep Network-based In-network processing paradigm, large scale distributed learning and inference can be conducted in the absence of a Cloud system. We show that this paradigm presents newer benefits that capacitate resource-constrained IoT end devices to instantiate Deep Networks in an energy-efficient, computationally viable, scalable, and robust way. Deep Network-based In-network processing can realize immediate benefits through automatic data reduction, reduced latency, streamlined resource optimization, dynamic self-organization, and privacy preservation.

2. In Chapter 4, we proposed a novel, end-to-end trainable Communicating Recurrent Neural Network *(C-RNN)* framework that can be instantiated on multiple network nodes to cooperatively solve distributed tasks via automatically learned latent state communication patterns. We provided a system model which delineates the WSN network graph and the Deep Network graph in our proposed system. In doing this we provided insights into the core network structure, system model, inter-node and intra-node communication paradigm and message structure and properties. We also showed how we were able to train a distributed Deep Network framework using a modified version of BackPropagation Through Time (BPTT) [182, 219], which we refer to as BackPropagation Through Time and Space (BPTTS). Using this approach, we can train our framework with tied-weights across multiple nodes. In Chapter 5 we show that our network can solve various objective tasks, synonymous to event detection pipelines, with high accuracy in a distributed fashion using our deep learning based communication paradigm. The data-driven approach to our enables easy scalability as adding additional nodes is abstracted as adding a data source

removing for the need for any architectural changes to the framework. Our *(C-RNN)* framework is generic and scaling nodes does not increase the network's parameters.

3. We proposed a Deep Network-based paradigm to handle latent state communication amongst multiple nodes based on node topology, an internal communication aggregator and gating function. Our communication paradigm, as discussed in Chapter 4, is adaptable to multiple problem scenarios, energy budgets and node topologies. Moreover, it offers flexible scalability and does not require any specialized hand-engineered algorithmic alterations when network-wide conditions change. Our communication paradigm can optimize for node communication using a communication loss and allows for further optimizations on communication channel sizes.

4. We proposed a Communicating Convolutional Recurrent Neural Network (C^2RNN) framework for cooperative distributed inference on visual tasks in Chapter 6. Our framework employs CNNs for feature extraction and RNNs for node communication and distributed inference in a unified end-to-end trainable hybrid architecture. Extensive evaluation on model convergence, node scalability, computation-cost, communication-cost, and network robustness show that our framework is suitable for resource-constrained IoT settings. Our framework learns to iteratively optimize communication and when tested on distributed inference tasks (details given in 6.4), only 51-56% node communication is necessary to reach a good accuracy. We showed that Our C^2RNN framework is robust to link failure by simulating link failure on different objective tasks. The model only degrades by 3.4%, 13.6% and 10.05% for 20%, 40% and 60% respectively. We also evaluated our model's ability to learn to discover and optimize communication protocols scalably (with experiments done up to 40 nodes). The model generalizes to unseen node topologies with varying numbers of nodes. For our model to achieve distributed inference, a series of latent-state updates across nodes is necessary. We show that the computational overhead of such updates is negligible with respect to the overall system. We compared our framework's computation and communication footprints to an Energy-Unaware baseline, Semi-centralized, Centralized and Tree-based model through extensive simulations. Our framework achieves 68x less communication footprint and utilizes 4× less compute resources in a 12-node system when compared to a fully centralized baseline. Our model whose feature extraction memory requirement

(0.16MB) and computational footprint (571.64 FLOPS) per single inferred image on a 12-node system is amenable for extension to resource-constrained Class 2-3 IoT devices, refer to for IoT end device classes.

5. Finally, in Chapter 7, we showed that our *C-RNN* and C^2RNN frameworks can be extended to a more complex problem of distributed target tracking. Distributed target tracking is challenging as evaluation based on ground truth could be plagued by ambiguity and learning motion patterns is computationally intensive. We show that our framework distributedly learns motion patterns and achieves good accuracies whilst optimizing for communication. We evaluate the trade-offs amongst varying levels of optimization on energy usage and node communication as both factors relate to tracking accuracy. Our framework is able to jointly optimize prediction accuracy (position of digits on a tracking grid), network energy (activation of only relevant sensor nodes) and communication-cost (learning to communicate only the relevant information for the tracking objective). With our approach, the operator can choose an optimization regime based on available computation, communication and energy resources. We showed how different optimization regimes perform in terms of predictive accuracy and realize optimal trade-off configurations. A model with 500K parameters can fit on an end device with 500KB memory when quantized to 8-bits [30]. Our models with parameter sizes 1402 and 2287 are lightweight and can be deployed on Class 2-3 resource constrained IoT devices.

8.2 Future Directions

So far, we have shown that we can learn complex communication patterns automatically using distributed Deep Networks instantiated on multiple nodes in a cooperative network. In turn, we can achieve distributed inference on various tasks in resource-constrained IoT settings where multiple embedded sensing devices continuously monitor dynamic phenomena and observe real-valued sensor readings and/or visual data. We have modelled several scenarios to demonstrate the efficacy of our proposed approach. Nonetheless, our work is limited in scope and opens up exciting directions for future research.

8.2.1 Techniques For Hardware Implementation

The deployment of deep network architectures on the IoT edge devices is hindered by limitations on compute and memory resources on these devices. Designing computation-aware and communication-efficient deep networks that can fit on these devices for distributed inference on various tasks enables the realization of truly intelligent IoT network edge. The work presented in this thesis makes strides in this direction. A natural extension of our approach would be to setup *C-RNN* and C^2RNN on real micro-controllers, (e.g., Raspberry-Pi, ARM Cortex M4) to represent an IoT edge setting. New techniques have been proposed for deploying deep networks for inference on microcontrollers, (i.e., EdgeML [2], X-CUBE-AI [3], MCUNet [121], ELM [186] et cetera). Among other considerations, some of these techniques employ model compression techniques, (i.e., pruning and quantization) as part of edge-centric memory and compute-optimized inference libraries. [30] distribute a deep network model for an image classification task on 8 Raspberry-Pi's and achieve 167M FLOPs, 430K parameters per Raspberry-Pi with overall parameters for network taking 0.8 M memory. Compared to the models considered in [30] for image classification tasks, our C^2RNN model is 17 $\times$ smaller and 38 $\times$ faster. This is without any model compression techniques applied to C^2RNN. In turn, we can extend C^2RNN to real hardware and realize even more computational and memory gains by using some of the newer techniques we have highlighted. Implementing C^2RNN on real hardware will require to careful design considerations on model training and parameter updating, (i.e., desktop training versus on-device training).

8.2.2 Complex People Tracking

A key direction for future work would be to extend our system to model complex human motion within a people tracking scenario employing multiple video streams. This setup is common for video surveillance applications in public spaces. Multi-view setups enable multiple people tracking in complex environments. People tracking using multiple cameras is challenging as approaches must simultaneously exploit information from multiple camera views whilst overcoming occlusion [67]. Moreover, the unpredictability of human motion introduces complexities. Using our approach, we could employ multiple distributed nodes to track individual trajectories and fuse the tracking information using our communicating Recurrent Neural Network model. In turn, we could estimate trajectories for multiple people and achieve a global view

of the objective cooperatively in a distributed paradigm with latent state communication. Using a Deep Network-based approaches in a data-driven manner avoids the need for unnecessary future engineering on distributed camera nodes. Nonetheless, learning optimized trajectories whilst addressing issues associated with human motion warrants further model architectural considerations. For instance, probabilistic models, such as Hidden Markov Models (HMMs) achieve better tracking accuracy by combining coarse, low-level sensory data with the prior knowledge of the data's statistical characteristics thereby avoiding the need to compute high-level representations of the data using expensive image processing algorithms [153]. Moreover, this direction is hampered by a lack of multi-view multi-camera datasets necessary for this task. Adapting available datasets for our distributed, communicating RNN approach requires significant and costly pre-processing.

8.2.3 Distributed Inference On Multimodal Data

We presented frameworks for distributed inference in 3 main settings; (1) a naive setting with real-valued sensor-like data, (2) a visual setting with non-temporal visual data and (3) a tracking setting (with temporal real-valued sensor-like data and temporal visual data). We have not tackled more complex scenarios in which multiple nodes may have heterogeneous characteristics, observe multi-modal data to solve distributed tasks cooperatively via automatically learned communication. An exciting direction would be to extend our *C-RNN* and C^2RNN frameworks into more generic and versatile architectures capable of handling distributed, cooperating nodes observing multimodal data. One way to achieve this would be to incorporate Graph Neural Networks [79, 129]. In this approach, each node is characterized by a set features which can be passed into the overall network communication pipeline as embeddings. In turn, nodes observing multimodal data can be effectively added to network for distributed inference on more complex distributed problems in resource-constrained IoT settings. For instance, consider a health monitoring application. In these scenarios, raw domain data collected from wearables and sensors can take the form of time-series physiologic and biomedical signals, clinical image data, spatial and longitudinal sensor data etc. Distributed, communicating nodes can cooperatively perform event detection and classification of monitored health objectives via latent-state communication.

8.2.4 Further Optimizations

Our proposed frameworks present several opportunities for optimization. Further research is necessary to realize the utility of optimizations on node topologies and feature extractors as they may have impact on the overall system's predictive accuracy.

8.2.4.1 Quantized Feature Extractors

Our C^2RNN can perform cooperative, distributed inference on visual tasks. From a computational cost perspective, our model is viable for resource constrained IoT settings. Nonetheless, feature extractors are canonically computationally intensive owing to complex operations in convolutional layers. Moreover, fully-connected layers are memory intensive as they contain a massive number of network parameters. Further research is necessary to navigate opportunities for further optimization on feature extractors through inference speed improvements, reduction in memory consumption and model compression without significant loss in inference accuracy as in [222].

8.2.4.2 Node Topologies

To perform inference on a distributed task, nodes are typically arranged in a cooperative system. In some cases, the arrangement is closely tied to the application. In our proposed approaches (as described in Sections 5.3, 6.3, 7.3) the exact location of nodes has no impact on overall system performance and inference on the sensed objectives and distributed visual classification tasks. It is also worthwhile to note that node communication patterns evolve based on the node topology, although at inference time these patterns are impacted by the communication loss (as described in Section 6.3.4). Our framework assumes that the transmission range between distributed nodes is equal and all nodes are similar in function and capability. In many real-world deployments, distributed cooperating nodes could have varied transmission ranges and heterogeneous characteristics and functions. Further research could navigate model performance when instantiated with more sophisticated node topologies with non-uniform transmission ranges across distributed cooperating nodes. [141] provide some ideas for node connections from a Wireless Sensor Network perspective.

Bibliography

[1] Auto-ID Labs.

[2] EdgeML: Edge Machine Learning.

[3] X-CUBE-AI - AI expansion pack for STM32CubeMX - STMicroelectronics.

[4] Kamel Abdelouahab, Maxime Pelcat, Jocelyn Sérot, and François Berry. Accelerating CNN inference on FPGAs: A Survey. Technical report, 2018.

[5] Prince Abudu and Andrew Markham. Distributed Communicating Neural Network Architecture for Smart Environments. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 233–240. IEEE, 6 2019.

[6] Naman Agarwal, Ananda Theertha Suresh, Felix Yu, Sanjiv Kumar, and H. Brendan Mcmahan. cpSGD: Communication-efficient and differentially-private distributed SGD. *Advances in Neural Information Processing Systems*, 2018-December:7564–7575, 5 2018.

[7] Bilal Ahmad Khan, Muhammad Sharif, Mudassar Raza, Tariq Umer, Khalid Hussain, and Aman Ullah Khan. An Approach for Surveillance Using Wireless Sensor Networks (WSN). *Journal of Information & Communication Technology*, 1:35–42, 3 2007.

[8] Alessandro Aimar, Hesham Mostafa, Enrico Calabrese, Antonio Rios-Navarro, Ricardo Tapiador-Morales, Iulia-Alexandra Lungu, Moritz B Milde, Federico Corradi, Alejandro Linares-Barranco, Shih-Chii Liu, and Tobi Delbruck. NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps. Technical report.

[9] Kemal Akkaya and Mohamed Younis. A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, 3(3):325–349, 5 2005.

[10] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 3 2002.

[11] Mohammed Ali Al-Garadi, Amr Mohamed, Abdulla Al-Ali, Xiaojiang Du, and Mohsen Guizani. A Survey of Machine and Deep Learning Methods for Internet of Things (IoT) Security. 7 2018.

[12] H. A.A. Al-Kashoash and Andrew H. Kemp. Comparison of 6LoWPAN and LPWAN for the Internet of Things. *Australian Journal of Electrical and Electronics Engineering*, 13(4):268–274, 10 2016.

[13] Giuseppe Amato, Fabio Carrara, Fabrizio Falchi, Claudio Gennaro, and Claudio Vairo. Car parking occupancy detection using smart camera networks and Deep Learning. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 1212–1217. IEEE, 6 2016.

[14] J. Amudha and P. Arpita. Multi-Camera Activation Scheme for Target Tracking with Dynamic Active Camera Group and Virtual Grid-Based Target Recovery. In *Procedia Computer Science*, volume 58, pages 241–248. Elsevier, 2015.

[15] Giuseppe Anastasi, Marco Conti, Mario Di Francesco, and Andrea Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537–568, 5 2009.

[16] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. 6 2016.

[17] Karl-Erik Årzén, Anton Cervin, and Dan Henriksson. Resource-Constrained Embedded Control Systems: Possibilities and Research Issues.

[18] Kevin Ashton. That 'Internet of Things' Thing - 2009-06-22 - Page 1 - RFID Journal.

[19] T. Reuban Gnana Asir and Hansa Lysander Manohar. Key Challenges and Success Factors in IoT - A Study on Impact of Data. In *2018 International Conference on Computer, Communication, and Signal Processing (ICCCSP)*, pages 1–5. IEEE, 2 2018.

[20] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 10 2010.

[21] Majid Bahrepour, Nirvana Meratnia, Mannes Poel, Zahra Taghikhaki, and Paul J.M. Havinga. Distributed event detection in wireless sensor networks for disaster management. In *Proceedings - 2nd International Conference on Intelligent Networking and Collaborative Systems, INCOS 2010*, pages 507–512, 2010.

[22] Seema Bandyopadhyay and Edward J Coyle. Spatio-Temporal Sampling Rates and Energy Efficiency in Wireless Sensor Networks.

[23] Suyoung Bang, Jingcheng Wang, Ziyun Li, Cao Gao, Yejoong Kim, Qing Dong, Yen-Po Chen, Laura Fick, Xun Sun, Ron Dreslinski, Trevor Mudge, Hun Seok Kim, David Blaauw, and Dennis Sylvester. A programmable deep-learning processor with 270KB on-chip weight storage using non-uniform memory hierarchy for mobile intelligence. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 250–251. IEEE, 2 2017.

[24] Andrea Banino, Caswell Barry, Benigno Uria, Charles Blundell, Timothy Lillicrap, Piotr Mirowski, Alexander Pritzel, Martin J. Chadwick, Thomas Degris, Joseph Modayil, Greg Wayne, Hubert Soyer, Fabio Viola, Brian Zhang, Ross Goroshin, Neil Rabinowitz, Razvan Pascanu, Charlie Beattie, Stig Petersen, Amir Sadik, Stephen Gaffney, Helen King, Koray Kavukcuoglu, Demis Hassabis, Raia Hadsell, and Dharshan Kumaran. Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557(7705):429–433, 5 2018.

[25] Johan Barthélemy, Nicolas Verstaevel, Hugh Forehead, and Pascal Perez. Edge-Computing Video Analytics for Real-Time Traffic Monitoring in a Smart City. *Sensors*, 19(9):2048, 5 2019.

[26] P. Bauer, M. Sichitiu, R. Istepanian, and K. Premaratne. The mobile patient: wireless distributed sensor networks for patient monitoring and care. In *Proceedings 2000 IEEE EMBS International Conference on Information Technology Applications in Biomedicine. ITAB-ITIS 2000. Joint Meeting Third IEEE EMBS International Conference on Information Technology Applications in Biomedicine (ITAB'00). Third Works*, pages 17–21. IEEE.

[27] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 3 1994.

[28] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy Layer-Wise Training of Deep Networks. Technical report.

[29] Kartikeya Bhardwaj, Ching-yi Lin, Anderson Sartor, Radu Marculescu, and Ching-Yi Lin. Memory-and Communication-Aware Model Compression for Distributed Deep Learning Inference on IoT. *ACM Trans. Embed. Comput. Syst*, 18, 2019.

[30] Kartikeya Bhardwaj, Naveen Suda, and Radu Marculescu. EdgeAI: A Vision for Deep Learning in IoT Era. *IEEE Design and Test*, 10 2019.

[31] Tapalina Bhattasali, Rituparna Chaki, and Nabendu Chaki. Secure and trusted cloud of things. In *2013 Annual IEEE India Conference (INDICON)*, pages 1–6. IEEE, 12 2013.

[32] P. Bonnet, J. Gehrke, and P. Seshadri. Querying the physical world. *IEEE Personal Communications*, 7(5):10–15, 2000.

[33] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC '12*, page 13, New York, New York, USA, 2012. ACM Press.

[34] Alessio Botta, Walter de Donato, Valerio Persico, and Antonio Pescapé. Integration of Cloud computing and Internet of Things: A survey. *Future Generation Computer Systems*, 56:684–700, 3 2016.

[35] Dario Bruneo and Fabrizio De Vita. On the Use of LSTM Networks for Predictive Maintenance in Smart Industries. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 241–248. IEEE, 6 2019.

[36] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning - ICML '06*, pages 161–168, New York, New York, USA, 2006. ACM Press.

[37] B.G. Celler, T. Hesketh, W. Earnshaw, and E. Ilsar. An instrumentation system for the remote monitoring of changes in functional health status of the elderly at home. In *Proceedings of 16th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 908–909. IEEE.

[38] Alberto Cerpa, Jeremy Elson, Deborah Estrin, Lewis Girod, Michael Hamilton, and Jerry Zhao. Habitat Monitoring: Application Driver for Wireless Communications Technology *.

[39] Tatjana Chavdarova and François Fleuret. Deep Multi-camera People Detection. 2 2017.

[40] Dongpeng Chen and Brian Mak. Multi-task Learning of Deep Neural Networks for Low-resource Speech Recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, pages 1–1, 2015.

[41] Jiasi Chen and Xukan Ran. Deep Learning with Edge Computing: A Review.

[42] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems - ASPLOS '14*, volume 49, pages 269–284, New York, New York, USA, 2014. ACM Press.

[43] Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing Neural Networks with the Hashing Trick. 4 2015.

[44] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 1 2017.

[45] Zhenping Chen, Dequan Li, Yourui Huang, and Chaoli Tang. Event-triggered communication for time synchronization in WSNs. *Neurocomputing*, 177(C):416–426, 2 2016.

[46] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A Survey of Model Compression and Acceleration for Deep Neural Networks. 10 2017.

[47] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. 6 2014.

[48] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical Multiscale Recurrent Neural Networks. *CoRR*, abs/1609.01704, 9 2016.

[49] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. 2 2016.

[50] Christopher J Cueva and Xue-Xin Wei. EMERGENCE OF GRID-LIKE REPRESENTATIONS BY TRAINING RECURRENT NEURAL NETWORKS TO PERFORM SPATIAL LOCALIZATION. Technical report.

[51] Daquan Feng, Chenzi Jiang, Gubong Lim, L.J. Cimini, Gang Feng, and G.Y. Li. A survey of energy-efficient wireless communications. *IEEE Communications Surveys & Tutorials*, 15(1):167–178, 21 2013.

[52] Bilal Khalid Dar, Muanm Ali Shah, Huniya Shahid, Fizzah Fizzah, and Zunaira Amjad. An Architecture for Fog Computing Enabled Emergency Response and Disaster Management System (ERDMS). In *2018 24th International Conference on Automation and Computing (ICAC)*, pages 1–6. IEEE, 9 2018.

[53] B. Das and V. Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *Proceedings of ICC'97 - International Conference on Communications*, volume 1, pages 376–380. IEEE.

[54] Sanjit Kumar Dash, Subasish Mohapatra, and Prasant Kumar Pattnaik. A Survey on Applications of Wireless Sensor Network Using Cloud Computing. Technical Report 4, 2010.

[55] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks, 2012.

[56] Li Deng. A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing*, 3:e2, 1 2014.

[57] John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.

[58] S.O. Dulman, L.F.W. van Hoesel, T. Nieberg, and Paul J.M. Havinga. Collaborative communication protocols for wireless sensor networks, 2003.

[59] Milica Pejanovic Durisic, Zhilbert Tafa, Goran Dimic, and Veljko Milutinovic. A survey of military applications of wireless sensor networks.

[60] Ozlem Durmaz Incel, Amitabha Ghosh, Bhaskar Krishnamachari, and Krishna Chintalapudi. Fast data collection in tree-based wireless sensor networks. *IEEE Transactions on Mobile Computing*, 11(1):86–99, 1 2012.

[61] Kartik Dutta, Praveen Krishnan, Minesh Mathew, and C. V. Jawahar. Improving CNN-RNN hybrid networks for handwriting recognition. In *Proceedings of International Conference on Frontiers in Handwriting Recognition, ICFHR*, volume 2018-August, pages 80–85. Institute of Electrical and Electronics Engineers Inc., 12 2018.

[62] Partha S Dutta, Claudia V Goldman, and Nicholas R Jennings. Communicating Effectively in Resource-Constrained Multi-Agent Systems.

[63] Vladimir Dyo, Stephen A Ellwood, David W Macdonald, Andrew Markham, Niki Trigoni, Ricklef Wohlers, Cecilia Mascolo, Bence Asztor, Salvatore Scellato, and Kharsim Yousef. WILDSENSING: Design and Deployment of a Sustainable Sensor Network for Wildlife Monitoring.

[64] Vladimir Dyo, Kharsim Yousef, Stephen A. Ellwood, David W. Macdonald, Andrew Markham, Niki Trigoni, Ricklef Wohlers, Cecilia Mascolo, Bence Pásztor, and Salvatore Scellato. WILDSENSING. *ACM Transactions on Sensor Networks*, 8(4):1–33, 9 2012.

[65] Irfan A Essa. Ubiquitous Sensing for Smart and Aware Environments: Technologies towards the building of an Aware Home.

[66] Mark Everingham, S M Ali Eslami, Luc Van Gool, Christopher K I Williams, John Winn, Andrew Zisserman, M Everingham, S M A Eslami, J Winn, L KU Van Gool Leuven, Belgium L Van Gool ETH, Switzerland C K I Williams, and A Zisserman. The PASCAL Visual Object Classes Challenge: A Retrospective. *Int J Comput Vis*, 111:98–136, 2015.

[67] François Fleuret, Jérôme Berclaz, Richard Lengagne, and Pascal Fua. Multicamera people tracking with a probabilistic occupancy map. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):267–282, 2 2008.

[68] Franck Fleurey, Brice Morin, Arnor Solberg, and Olivier Barais. MDE to Manage Communications with and between Resource-Constrained Systems.

[69] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to Communicate to Solve Riddles with Deep Distributed Recurrent Q-Networks. *arXiv*, 2 2016.

[70] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to Communicate with Deep Multi-Agent Reinforcement Learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, page 2145–2153, 5 2016.

[71] Xianghua Fu, Wangwang Liu, Yingying Xu, Chong Yu, Ting Wang, Robert J Durrant, and Kee-Eung Kim. Long Short-term Memory Network over Rhetorical Structure Theory for Sentence-level Sentiment Analysis. 63:17–32, 2016.

[72] Kunihiko Fukushima. Biological Cybernetics Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biol. Cybernetics*, 36(193), 1980.

[73] Michael R. Genesereth and Steven P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–ff., 7 1994.

[74] Ross Girshick. Fast R-CNN. 4 2015.

[75] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. 11 2013.

[76] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. Technical report.

[77] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. Technical report, MIT Press, 2015.

[78] Chirag Gupta, Arun Sai Suggala, Ankit Goyal, Harsha Vardhan Simhadri, Bhargavi Paranjape, Ashish Kumar, Saurabh Goyal, Raghavendra Udupa, Manik Varma, and Prateek Jain. ProtoNN: Compressed and Accurate kNN for Resource-scarce Devices. 2017.

[79] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, volume 2017-December, pages 1025–1035. Neural information processing systems foundation, 6 2017.

[80] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both Weights and Connections for Efficient Neural Networks. Technical report.

[81] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both Weights and Connections for Efficient Neural Networks. Technical report.

[82] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Unsupervised Learning. pages 485–585. Springer, New York, NY, 2009.

[83] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. 12 2015.

[84] Tian He, Sudha Krishnamurthy, John A Stankovic, Tarek Abdelzaher, Liqian Luo, Radu Stoleru, Ting Yan, Lin Gu, Jonathan Hui, and Bruce Krogh. Energy-Efficient Surveillance System Using Wireless Sensor Networks *.

[85] W.R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, volume vol.1, page 10. IEEE Comput. Soc.

[86] G E Hinton and R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science (New York, N.Y.)*, 313(5786):504–7, 7 2006.

[87] Geoffrey Hinton and Jeff Dean. Distilling the Knowledge in a Neural Network. Technical report, 2015.

[88] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-Rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. Deep Neural Networks for Acoustic Modeling in Speech Recognition. Technical report.

[89] Geoffrey E. Hinton. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[90] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets *.

[91] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. 3 1991.

[92] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[93] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, and Marco Andreetto. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. Technical report.

[94] Yu Hu, Yongkang Wong, Wentao Wei, Yu Du, Mohan Kankanhalli, and Weidong Geng. A novel attention-based hybrid CNN-RNN architecture for sEMG-based gesture recognition. *PLOS ONE*, 13(10):e0206049, 10 2018.

[95] D H HUBEL and T N WIESEL. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160:106–54, 1 1962.

[96] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and. 2 2016.

[97] Michaela Iorga, Larry Feldman, Robert Barton, Michael J Martin, Nedim Goren, and Charif Mahmoudi. Fog Computing Conceptual Model: Recommendations of the National Institute of Standards and Technology. *NIST*, 2018.

[98] S.M. Sofiqul Islam, Shanto Rahman, Md. Mostafijur Rahman, Emon Kumar Dey, and Mohammad Shoyaib. Application of deep learning to computer vision: A comprehensive study. In *2016 5th International Conference on Informatics, Electronics and Vision (ICIEV)*, pages 592–597. IEEE, 5 2016.

[99] Herbert Jaeger. Adaptive Nonlinear System Identification with Echo State Networks.

[100] Jeremy Geelan. Twenty One Experts Define Cloud Computing, 2008.

[101] Jonghoon Jin, Aysegul Dundar, and Eugenio Culurciello. Flattened Convolutional Neural Networks for Feedforward Acceleration. 12 2014.

[102] B. Kalpana and R. Sangeetha. A Novel Framework for Target Tracking and Data Fusion in Wireless Sensor Networks Using Kernel Based Learning Algorithm. pages 170–174. 2011.

[103] Riad Kanan, Obaidallah Elhassan, and Rofaida Bensalem. An IoT-based autonomous system for workers' safety in construction sites with real-time alarming, monitoring, and positioning strategies. *Automation in Construction*, 88:73–86, 4 2018.

[104] Akhmedov Khumoyun, Yun Cui, and Lee Hanku. Spark based distributed Deep Learning framework for Big Data applications. In *2016 International Conference on Information Science and Communications Technologies (ICISCT)*, pages 1–5. IEEE, 11 2016.

[105] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980, 12 2014.

[106] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated Learning: Strategies for Improving Communication Efficiency. 10 2016.

[107] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, 2009.

[108] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks.

[109] N. Kruger, P. Janssen, S. Kalkan, M. Lappe, A. Leonardis, J. Piater, A. J. Rodriguez-Sanchez, and L. Wiskott. Deep Hierarchies in the Primate Visual Cortex: What Can We Learn for Computer Vision? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1847–1871, 8 2013.

[110] Ashish Kumar, Saurabh Goyal, and Manik Varma. Resource-efficient Machine Learning in 2 KB RAM for the Internet of Things. 2017.

[111] Don Kurian, Dennis Chirag, Pabbaraju Harsha, Vardhan Simhadri, and Prateek Jain. Multiple Instance Learning for Efficient Sequential Data Classification on Resource-constrained Devices.

[112] Nicholas D. Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 1–12. IEEE, 4 2016.

[113] Nicholas D. Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. An Early Resource Characterization of Deep Learning on Wearables, Smartphones and Internet-of-Things Devices. In *Proceedings of the 2015 International Workshop on Internet of Things towards Applications - IoT-App '15*, pages 7–12, New York, New York, USA, 2015. ACM Press.

[114] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998.

[115] Sunwoo Lee, Ankit Agrawal, Prasanna Balaprakash, Alok Choudhary, and Wei Keng Liao. Communication-Efficient Parallelization Strategy for Deep Convolutional Neural Network Training. *Proceedings of MLHPC 2018: Machine Learning in HPC Environments, Held in conjunction with SC 2018: The International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 47–56, 2 2019.

[116] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection.

[117] Chao Li, Yushu Xue, Jing Wang, Weigong Zhang, and Tao Li. Edge-Oriented Computing Paradigms: A Survey on Architecture Design and System Management. *ACM Comput. Surv*, 51, 2018.

[118] Dawei Li, Theodoros Salonidis, Nirmit V. Desai, and Mooi Choo Chuah. DeepCham: Collaborative Edge-Mediated Adaptive Deep Learning for Mobile Object Recognition. In *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 64–76. IEEE, 10 2016.

[119] Liangzhi Li, Kaoru Ota, and Mianxiong Dong. Deep Learning for Smart Industry: Efficient Manufacture Inspection System With Fog Computing. *IEEE Transactions on Industrial Informatics*, 14(10):4665–4673, 10 2018.

[120] Xiang Li, Tao Qin, Jian Yang, and Tie-Yan Liu. LightRNN: Memory and Computation-Efficient Recurrent Neural Networks. *CoRR*, abs/1610.09893, 10 2016.

[121] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. MCUNet: Tiny Deep Learning on IoT Devices. Technical report.

[122] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A.W.M. van der Laak, Bram van Ginneken, and Clara I. Sánchez. A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42:60–88, 12 2017.

[123] Chang Liu, Yu Cao, Yan Luo, Guanling Chen, Vinod Vokkarane, Ma Yunsheng, Songqing Chen, and Peng Hou. A New Deep Learning-Based Food Recognition System for Dietary Assessment on An Edge Computing Service Infrastructure. *IEEE Transactions on Services Computing*, 11(2):249–261, 3 2018.

[124] Lumin Liu, Jun Zhang, S H Song, and Khaled B Letaief. Client-Edge-Cloud Hierarchical Federated Learning.

[125] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single Shot MultiBox Detector. Technical report.

[126] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 4 2017.

[127] Tie Luo and Sai G. Nagarajan. Distributed Anomaly Detection Using Autoencoder Neural Networks in WSN for IoT. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 5 2018.

[128] Xiong Luo and Xiaohui Chang. A novel data fusion scheme using grey model and extreme learning machine in wireless sensor networks. *International Journal of Control, Automation and Systems*, 13(3):539–546, 6 2015.

[129] Yao Ma, Suhang Wang, Charu C. Aggarwal, Dawei Yin, and Jiliang Tang. Multi-dimensional graph convolutional networks. In *SIAM International Conference on Data Mining, SDM 2019*, pages 657–665. Society for Industrial and Applied Mathematics Publications, 8 2019.

[130] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations. *Neural Computation Massachusetts Institute of Technology*, 14:2531–2560, 2002.

[131] Matthew Mackay, Paul Vicol, Jimmy Ba, and Roger Grosse. Reversible Recurrent Neural Networks. Technical report.

[132] Raka Mahesa. How cloud, fog, and mist computing can work together, 2018.

[133] Nima Mahmoudi, Seyed Mohammad Ahadi, and Mohammad Rahmati. Multi-target tracking using CNN-based features: CNNMTT. *Multimedia Tools and Applications*, 78(6):7077–7096, 3 2019.

[134] Navonil Majumder, Soujanya Poria, Alexander Gelbukh, and Erik Cambria. Deep Learning-Based Document Modeling for Personality Detection from Text. *IEEE Intelligent Systems*, 32(2):74–79, 3 2017.

[135] Milos Manic, Kasun Amarasinghe, Juan J. Rodriguez-Andina, and Craig Rieger. Intelligent Buildings of the Future: Cyberaware, Deep Learning Powered, and Human Interacting. *IEEE Industrial Electronics Magazine*, 10(4):32–49, 24 2016.

[136] Andrew Markham and Niki Trigoni. Discrete Gene Regulatory Networks (dGRNs): A Novel Approach to Configuring Sensor Networks. In *2010 Proceedings IEEE INFOCOM*, pages 1–9. IEEE, 3 2010.

[137] James Martens. Deep learning via Hessian-free optimization.

[138] James Martens and Ilya Sutskever. Learning Recurrent Neural Networks with Hessian-Free Optimization.

[139] Alireza Masoum, Nirvana Meratnia, and Paul J M Havinga. An Energy-Efficient Adaptive Sampling Scheme for Wireless Sensor Networks. 2013.

[140] César Lincoln C. Mattos, Zhenwen Dai, Andreas Damianou, Jeremy Forth, Guilherme A. Barreto, and Neil D. Lawrence. Recurrent Gaussian Processes. 11 2015.

[141] Michael J. McGrath, Cliodhna Ní Scanaill, Michael J. McGrath, and Cliodhna Ní Scanaill. Sensor Network Topologies and Design Considerations. In *Sensor Technologies*, pages 79–95. Apress, 2013.

[142] P M Mell and T Grance. The NIST definition of cloud computing. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, 2011.

[143] Massimo Merenda, Carlo Porcaro, and Demetrio Iero. Edge Machine Learning for AI-Enabled IoT Devices: A Review. *Sensors 2020, Vol. 20, Page 2533*, 20(9):2533, 4 2020.

[144] Stefano Messelodi, Carla M. Modena, Michele Zanin, Francesco G.B. De Natale, Fabrizio Granelli, Enrico Betterle, and Andrea Guarise. Intelligent extended floating car data collection. *Expert Systems with Applications*, 36(3):4213–4227, 4 2009.

[145] Anton Milan, S Hamid Rezatofighi, Anthony Dick, Ian Reid, and Konrad Schindler. Online Multi-Target Tracking Using Recurrent Neural Networks. Technical report, Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17), 2017.

[146] ml4a. How neural networks are trained, 2012.

[147] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning.

[148] Mehdi Mohammadi, Ala Al-Fuqaha, Sameh Sorour, and Mohsen Guizani. Deep Learning for IoT Big Data and Streaming Analytics: A Survey. 12 2017.

[149] Bert Moons, Koen Goetschalckx, Nick Van Berckelaer, and Marian Verhelst. Minimum Energy Quantized Neural Networks. *Conference Record of 51st Asilomar Conference on Signals, Systems and Computers, ACSSC 2017*, 2017-October:1921–1925, 11 2017.

[150] Bert Moons and Marian Verhelst. A 0.3-2.6 TOPS/W Precision-Scalable Processor for Real-Time Large-Scale ConvNets. Technical report, arXiv, 2016.

[151] Conor Muldoon, Gregory M.P. O'Hare, Michael J. O'Grady, and Richard Tynan. Agent Migration and Communication in WSNs. In *2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 425–430. IEEE, 2008.

[152] Lekha R. Nair, Sujala D. Shetty, and Siddhanth D. Shetty. Applying spark based machine learning model on streaming big data for health status prediction. *Computers & Electrical Engineering*, 65:393–399, 1 2018.

[153] Vinod Nair and James J Clark. Automated Visual Surveillance Using Hidden Markov Models. Technical report.

[154] Maryam M Najafabadi, Flavio Villanustre, Taghi M Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. Deep learning applications and challenges in big data analytics. *Journal of Big Data*, 2(1):1, 12 2015.

[155] Eduardo F Nakamura, Antonio A F Loureiro, Alejandro C Frery, E F Nakamura, and A A F Loureiro. Information Fusion for Wireless Sensor Networks: Methods, Models, and Classifications ACM Reference Format. *ACM Computing Surveys*, 39(9), 2007.

[156] Pedro J. Navarro, Carlos Fernández, Raúl Borraz, and Diego Alonso. A Machine Learning Approach to Pedestrian Detection for Autonomous Vehicles Using High-Definition 3D Range Data. *Sensors 2017, Vol. 17, Page 18*, 17(1):18, 12 2016.

[157] Sarfraz Nawaz, Xiaomin Xu, David Rodenas-Herráiz, Paul Fidler, Kenichi Soga, and Cecilia Mascolo. Monitoring A Large Construction Site Using Wireless Sensor Networks.

[158] Nicholas Nordlund, Heesung Kwon, and Leandros Tassiulas. Cooperative Learning for Multi-perspective Image Classification. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 75–80. IEEE, 6 2019.

[159] Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 10 2010.

[160] Donghyun Park, Seulgi Kim, Yelin An, and Jae Yoon Jung. LiReD: A Light-Weight Real-Time Fault Detection System for Edge Computing Using LSTM Recurrent Neural Networks. *Sensors 2018, Vol. 18, Page 2110*, 18(7):2110, 6 2018.

[161] Douglass Parkhill. *The challenge of the computer utility*. Addison-Wesley, 1966.

[162] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to Construct Deep Recurrent Neural Networks. *CoRR*, abs/1312.6026, 12 2013.

[163] Madhav Patil and Kishan Gutta. Wireless Sensors in Industrial Instrumentation A Survey.

[164] Madhav Patil and Kishan Gutta. Wireless Sensors in Industrial Instrumentation A Survey.

[165] Shishir G Patil, Don Kurian, Dennis Chirag, Pabbaraju Rajanikant, Deshmukh Harsha, Vardhan Simhadri, Manik Varma, and Prateek Jain. GesturePod: Programmable Gesture Recognition for Augmenting Assistive Devices.

[166] Schalk Wilhelm Pienaar and Reza Malekian. Human Activity Recognition Using LSTM-RNN Deep Neural Network Architecture. *2019 IEEE 2nd Wireless Africa Conference, WAC 2019 - Proceedings*, 5 2019.

[167] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. 2 2018.

[168] Fabio Previtali, Domenico D. Bloisi, and Luca Iocchi. A distributed approach for real-time multi-camera multiple object tracking. *Machine Vision and Applications*, 28(3-4):421–430, 5 2017.

[169] Foster J. Provost. Scaling Up: Distributed Machine Learning with Cooperation. In *AAAI'96: Proceedings of the thirteenth national conference on Artificial intelligence*, pages 74–79, 1996.

[170] Hongwei Quan, Anke Xue, and Dongliang Peng. Joint target tracking and classification with heterogeneous sensors. *Elektronika ir Elektrotechnika*, (4):61–64, 2012.

[171] Ramesh Rajagopalan and Pramod Varshney. Data-aggregation techniques in sensor networks: a survey. *IEEE Communications Surveys & Tutorials*, 8(4):48–63, 2006.

[172] Shankar Ganesh Ramasubramanian, Rangharajan Venkatesan, Mrigank Sharad, Kaushik Roy, and Anand Raghunathan. SPINDLE. In *Proceedings of the 2014 international symposium on Low power electronics and design - ISLPED '14*, pages 15–20, New York, New York, USA, 2014. ACM Press.

[173] Tifenn Rault, Abdelmadjid Bouabdallah, and Yacine Challal. Energy efficiency in wireless sensor networks: A top-down survey. *Computer Networks*, 67:104–122, 7 2014.

[174] P.P. Ray. A survey on Internet of Things architectures. *Journal of King Saud University - Computer and Information Sciences*, 30(3):291–319, 7 2018.

[175] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. Technical report.

[176] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. Technical report.

[177] A. Rehman, M. Mustafa, N. Javaid, U. Qasim, and Z.A. Khan. Analytical Survey of Wearable Sensors. In *2012 Seventh International Conference on Broadband, Wireless Computing, Communication and Applications*, pages 408–413. IEEE, 11 2012.

[178] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. 6 2015.

[179] Martin Riedmiller and Heinrich Braun. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm.

[180] Margaret Rouse. What is pervasive computing (ubiquitous computing)? - Definition from WhatIs.com, 2016.

[181] Deboleena Roy, Gopalakrishnan Srinivasan, Priyadarshini Panda, Richard Tomsett, Nirmit Desai, Raghu Ganti, and Kaushik Roy. Neural Networks at the Edge. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 45–50. IEEE, 6 2019.

[182] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 10 1986.

[183] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 12 2015.

[184] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 12 2015.

[185] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. 2 2014.

[186] Fouad Sakr, Francesco Bellotti, Riccardo Berta, and Alessandro De Gloria. Machine Learning on Mainstream Microcontrollers. *Sensors*, 20(9):2638, 5 2020.

[187] Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. Relational recurrent neural networks. *arXiv*, 6 2018.

[188] MG Sarwar Murshed, Christopher Murphy, Nazar Khan, and Pakistan Ganesh Ananthanarayanan. Machine Learning at the Network Edge: A Survey; Machine Learning at the Network Edge: A Survey.

[189] Divya Sharma, Sandeep Verma, and Kanika Sharma. Network Topologies in Wireless Sensor Networks: A Review 1. 4.

[190] Dinggang Shen, Guorong Wu, and Heung-Il Suk. Deep Learning in Medical Image Analysis. *Annual Review of Biomedical Engineering*, 19(1):221–248, 6 2017.

[191] Baoguang Shi, Xiang Bai, and Cong Yao. An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(11):2298–2304, 11 2017.

[192] Jianhua Shi, Jiafu Wan, Hehua Yan, and Hui Suo. A survey of Cyber-Physical Systems. In *2011 International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6. IEEE, 11 2011.

[193] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 10 2016.

[194] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the Game of Go with Deep Neural Networks and Tree Search.

[195] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. 9 2014.

[196] Philipp Sommer, Jiajun Liu, Kun Zhao, Branislav Kusy, Raja Jurdak, Adam Mckeown, and David Westcott. Information Bang for the Energy Buck: Towards Energy-and Mobility-Aware Tracking.

[197] S.V. Srikanth, Pramod P.J., Dileep K.P., Tapas S., Mahesh U. Patil, and Sarat Chandra Babu N. Design and Implementation of a Prototype Smart PARKing (SPARK) System Using Wireless Sensor Networks. In *2009 International Conference on Advanced Information Networking and Applications Workshops*, pages 401–406. IEEE, 5 2009.

[198] John A. Stankovic, Anthony D. Wood, and Tian He. Realistic Applications for Wireless Sensor Networks. pages 835–863. Springer, Berlin, Heidelberg, 2011.

[199] Christos Stergiou, Kostas E. Psannis, Byung-Gyu Kim, and Brij Gupta. Secure integration of IoT and Cloud Computing. *Future Generation Computer Systems*, 78:964–975, 1 2018.

[200] George Suciu, Alexandru Vulpe, Simona Halunga, Octavian Fratu, Gyorgy Todoran, and Victor Suciu. Smart Cities Built on Resilient Cloud Computing and Secure Internet of Things. In *2013 19th International Conference on Control Systems and Computer Science*, pages 513–518. IEEE, 5 2013.

[201] Sujesha Sudevalayam and Purushottam Kulkarni. Energy Harvesting Sensor Nodes: Survey and Implications. 2008.

[202] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning Multiagent Communication with Backpropagation. *arXiv*, 2016.

[203] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM Neural Networks for Language Modeling. Technical report.

[204] Richard S Sutton, Andrew G Barto, and A Bradford Book. Reinforcement Learning: An Introduction.

[205] Piotr Szczytowski, Abdelmajid Khelil, and Neeraj Suri. ASample: Adaptive Spatial Sampling in Wireless Sensor Networks. In *2010 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, pages 35–42. IEEE, 2010.

[206] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. 9 2014.

[207] T. Tieleman and G. Hinton. Rmsprop: Divide the gradient by a running average of its recent magnitude - Optimization: How to make the learning go faster — Coursera.

[208] Amir Taherkordi and Frank Eliassen. Data-Centric IoT Services Provisioning in Fog-Cloud Computing Systems. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation - IoTDI '17*, pages 317–318, New York, New York, USA, 2017. ACM Press.

[209] Gavin Taylor, Taylor@usna Edu, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, Tom Goldstein, and Tomg@cs Umd Edu. Training Neural Networks Without Gradients: A Scalable ADMM Approach.

[210] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 328–339. IEEE, 6 2017.

[211] Minoru Uehara. Mist Computing: Linking Cloudlet to Fogs. pages 201–213. Springer, Cham, 2018.

[212] Prateeksha Varshney and Yogesh Simmhan. Demystifying Fog Computing: Characterizing Architectures, Applications and Abstractions. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 115–124. IEEE, 5 2017.

[213] Marian Verhelst and Bert Moons. Embedded Deep Neural Network Processing: Algorithmic and Processor Techniques Bring Deep Learning to IoT and Edge Devices. *IEEE Solid-State Circuits Magazine*, 9(4):55–65, 9 2017.

[214] Cheng-Xiang Wang, Fourat Haider, Xiqi Gao, Xiao-Hu You, Yang Yang, Dongfeng Yuan, Hadi Aggoune, Harald Haas, Simon Fletcher, and Erol Hepsaydir. Cellular architecture and key technologies for 5G wireless communication networks. *IEEE Communications Magazine*, 52(2):122–130, 2 2014.

[215] Dong Wang, Chao Liu, Zhiyuan Tang, Zhiyong Zhang, and Mengyuan Zhao. Recurrent Neural Network Training with Dark Knowledge Transfer. 2015.

[216] Hongyi Wang, Scott Sievert, Zachary Charles, Shengchao Liu, Stephen Wright, and Dimitris Papailiopoulos. ATOMO: Communication-efficient Learning via Atomic Sparsification. *Advances in Neural Information Processing Systems*, 2018-December:9850–9861, 6 2018.

[217] Jiang Wang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. CNN-RNN: A Unified Framework for Multi-label Image Classification. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-December, pages 2285–2294. IEEE Computer Society, 12 2016.

[218] Zhonghui Wang and Feng Wang. Wireless Visual Sensor Networks: Applications, Challenges, and Recent Advances. In *Conference Proceedings - IEEE SOUTHEASTCON*, volume 2019-April. Institute of Electrical and Electronics Engineers Inc., 4 2019.

[219] Paul J. Werbos. Backpropagation Through Time: What It Does and How to Do It. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[220] Bogdan. M. Wilamowski, Bo Wu, and Janusz Korniak. Big data and deep learning. *undefined*, 2016.

[221] Rebecca Willett, Aline Martin, and Robert Nowak. Backcasting: Adaptive Sampling for Sensor Networks.

[222] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-December, pages 4820–4828. IEEE Computer Society, 12 2016.

[223] Tianwei Xing, Marc Roig Vilamala, Luis Garcia, Federico Cerutti, Lance Kaplan, Alun Preece, and Mani Srivastava. DeepCEP: Deep Complex Event Processing Using Distributed Multimodal Information. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 87–92. IEEE, 6 2019.

[224] Tien Ju Yang, Yu Hsin Chen, Joel Emer, and Vivienne Sze. A method to estimate the energy consumption of deep neural networks. *Conference Record of 51st Asilomar Conference on Signals, Systems and Computers, ACSSC 2017*, 2017-October:1916–1920, 4 2018.

[225] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning.

[226] Mark Yarvis, Nandakishore Kushalnagar, Harkirat Singh, Anand Rangarajan, York Liu, and Suresh Singh. Exploiting Heterogeneity in Sensor Networks.

[227] W. Ye, J. Heidemann, and D. Estrin. Medium Access Control With Coordinated Adaptive Sleeping for Wireless Sensor Networks. *IEEE/ACM Transactions on Networking*, 12(3):493–506, 6 2004.

[228] Shanhe Yi, Cheng Li, and Qun Li. A Survey of Fog Computing: Concepts, Applications and Issues.

[229] Qiwei Yin, Ruixun Zhang, and XiuLi Shao. CNN and RNN mixed model for image classification. *MATEC Web of Conferences*, 277:02001, 2019.

[230] Chen Yingwen, Va Leong Hong, Xu Ming, Cao Jiannong, Keith C.C. Chan, and Alvin T.S. Chan. In-network data processing for wireless sensor networks. *Proceedings - IEEE International Conference on Mobile Data Management*, 2006, 2006.

[231] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. All One Needs to Know about Fog Computing and Related Edge Computing Paradigms: A Complete Survey. 8 2018.

[232] Ömer Yüksel, Jerry den Hartog, and Sandro Etalle. Reading between the fields. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing - SAC '16*, pages 2063–2070, New York, New York, USA, 2016. ACM Press.

[233] Qingchen Zhang, Laurence T. Yang, Zhikui Chen, and Peng Li. A survey on deep learning for big data. *Information Fusion*, 42:146–157, 7 2018.

[234] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level Convolutional Networks for Text Classification, 2015.

[235] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. Hello Edge: Keyword Spotting on Microcontrollers.

[236] Rui Zhao, Haider Ali, and Patrick van der Smagt. Two-Stream RNN/CNN for Action Recognition in 3D Videos. *IEEE International Conference on Intelligent Robots and Systems*, 2017-September:4260–4267, 3 2017.

[237] Xiaojin Zhu and Xiaojin Zhu. Semi-Supervised Learning Literature Survey. 2006.