Deep Neural Network Modeling of Electric Motors

Sagar Verma

THESE DE DOCTORAT

NNT : 2023UPAST088

université PARIS-SACLAY

Deep Neural Network Modeling of Electric Motors

Modélisation de moteurs électriques à l'aide de réseaux de neurones profonds

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580, sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat: Sciences du traitement du signal et des images
Graduate School: Sciences de l'ingénierie et des systèmes
Référent: CentraleSupélec

Thèse préparée dans l'unité de recherche **Centre de Vision Numérique (Université Paris-Saclay, CentraleSupélec)** sous la direction de **Jean-Christophe Pesquet**, Professeur, CentraleSupélec, le co-encadrement de **Marc Castella**, Maître de Conférences, Télécom SudParis et- la supervision de **Nicolas Henwood**, Schneider Electric.

Thèse soutenue à Paris-Saclay, le 16 June 2023, par

Sagar VERMA

Composition du jury

Antoine BERTHET Professor, CentraleSupélec	Président
Ismail BEN AYED Professor, ÉTS Montréal	Examinateur
Camille COUPRIE Chargée de Recherche, Meta AI	Examinatrice
Chetan ARORA Professor, IIT Delhi	Rapporteur & Examinateur
Stefan DUFFNER Maitre de Conférences, Université de Lyon	Rapporteur & Examinateur

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Modélisation de moteurs électriques à l'aide de réseaux de neurones profonds
Mots clés: Apprentissage profond, moteur à induction, réseaux efficaces, sparsification

Résumé: Cette thèse traite de l'application des réseaux de neurones dans la résolution de problèmes liés aux moteurs électriques. Le chapitre 2 contribue à identifier une structure de réseau de neurones capable d'apprendre la relation multi-variée entre différents signaux d'un moteur électrique. La structure identifiée est ensuite utilisée pour l'estimation vitesse-couple à partir des courants et des tensions. Le chapitre 3 se concentre sur la détection et la correction de défauts de mesure. Notre méthode prend en compte les défauts de capteurs électriques, les défauts mécaniques et l'estimation de température.

Le chapitre 4 traite ensuite de la fiabilité de l'estimateur vitesse-couple en cas de courants et de tensions bruités. Nous présentons une méthode de débruitage permettant de rendre notre estimateur vitesse-couple applicable dans un contexte réaliste. Ensuite, une rapide analyse de la robustesse face à une attaque adverse est menée pour les réseaux neuronaux utilisés dans des applications des moteurs électriques. La capacité de généralisation de l'estimateur vitesse-couple est également brièvement analysée. Dans le chapitre 5, nous nous concentrons sur le dernier obstacle à la mise en oeuvre des réseaux de neurones: le coût de calcul. Nous présentons la méthode de sparsification par inclusion sous-différentielle (SIS) permettant de trouver le meilleur réseau parcimonieux à partir de poids pré-calculés, tout en conservant la précision d'origine.

Title: Modeling of Electric Motors using Deep Neural Networks
Keywords: Deep Learning, Induction Motor, Efficient Networks, Pruning

Abstract: This thesis deals with the application of neural networks in solving electrical motor problems. Chapter 2 contributes to identifying a neural network that can learn the multivariate relationship between different electrical motor signals. The identified network is then used for speed-torque estimation from currents and voltages. Chapter 3 focuses on detecting and recovering from faulty measurements. Our method encompasses electrical sensor faults, mechanical faults, and temperature estimation.

Chapter 4 then discusses the reliability of the speed-torque estimator in case of noisy currents and voltages. We present a denoising method which allows our speed-torque estimator to be applicable in a realistic context. This is followed by an analysis of the adversarial robustness of the neural networks used in electrical motor tasks. The generalization capability of the speed-torque estimator is also briefly considered. In Chapter 5, we focus on the final roadblock in achieving real-world application of neural networks: computational requirements. We present the Subdifferential Inclusion for Sparsity (SIS) method to find the best sparse network from pretrained weights while maintaining original accuracy.

Abstract

Electrical motors are so much a part of everyday life that we seldom give them a second thought. For example, when we switch on an electrical vehicle, we expect it to run rapidly up to the correct speed, provide acceleration, stop when brakes are applied, and casually predict faults to avoid future mishaps. Electrical motors have very complex dynamics and it is essential to have a controller that can provide robust control based on these dynamics. Current industrial controllers offer various solutions like fault detection, system observation, and predictive maintenance. Artificial neural networks have shown tremendous promising results in various vision, natural language processing, and robotic control tasks in the last decade. A similar revolution has just begun in the heavy industry under the umbrella of Industry 4.0. This thesis focuses on bringing such advances in deep neural networks to the electrical motor industry for system modeling, fault detection, and predictive maintenance.

The first contribution in Chapter 2 is about data-driven modeling of electrical motor dynamics. This is achieved by learning the input-output relationship between different quantities of induction motors using various neural networks. A new network architecture has been introduced that combines the benefits of convolutions and sequential layers for time-series regression tasks. Using this identified network, we set on the path for creating a neural speed-torque estimator. This contribution is towards applying the identified neural network and establishing electrical domain-aligned evaluation metrics to analyze the performance of neural networks. In Chapter 3, we leverage the same network to address practical applications by focusing on problems like sensor fault recovery, mechanical fault detection, and temperature modeling.

Having shown multiple applications of novel proposed neural networks, we move the needle toward the reliability of such methods in the real world. Chapter 4 proposes a real-world network pipeline by chaining a neural signal denoiser that takes real-world noisy currents and voltages, and denoises them as inputs for speed-torque estimation. This removes the need to collect a large, noisy dataset to train a good speed-torque estimator network. We study the robustness of this real-world speed-torque estimator pipeline by generating adversarial attacks, concluding that a better physics-aware attacker is essential. At last, this chapter contributes another insight into the reliability of the speed-torque estimator pipeline along the path of

domain generalization.

The mixed bag of scientific rewards attained in the contributions of Chapter 4 enables us to question more about the real-world usage of the speed-torque estimator and other such electrical motor networks introduced so far. For this, Chapter 5 leads to a new convex optimization-based method for sparsifying pre-trained neural networks. This opens the avenue of utilizing the speed-torque estimator in a real-time fashion on general-purpose computers.

Résumé

Les moteurs électriques font tellement partie de la vie quotidienne que nous ne leur prêtons plus guère attention. Par exemple, lorsque nous démarrons un véhicule électrique, nous nous attendons à ce qu'il atteigne rapidement la vitesse souhaitée, fournisse la bonne accélération, s'arrête lorsqu'on actionne les freins, et même qu'il prédise au passage les pannes pour éviter de futurs incidents. Les moteurs électriques ont une dynamique très complexe et il est essentiel de disposer d'un contrôleur capable de fournir un contrôle robuste basé sur cette dynamique. Les contrôleurs industriels offrent de nos jours diverses solutions telles que la détection de pannes, l'observation du système et la maintenance prédictive. Au cours de la dernière décennie, les réseaux de neurones artificiels ont montré des résultats extrêmement prometteurs dans diverses tâches de vision, de traitement du langage naturel et de contrôle robotique. Une révolution similaire vient de s'amorcer dans le domaine industriel sous l'égide de l'Industrie 4.0. Cette thèse s'applique à transférer de telles avancées en réseaux de neurones profonds vers l'industrie des moteurs électriques, en vue de la modélisation de systèmes, la détection de défauts et la maintenance prédictive.

La première contribution du chapitre 2 concerne la modélisation de la dynamique des moteurs électriques à l'aide de données. Celle-ci est réalisée en apprenant, grâce à divers réseaux de neurones, la relation entrée-sortie entre différentes quantités des moteurs à induction. Une nouvelle architecture de réseau est introduite qui combine les avantages des couches convolutives et séquentielles pour les tâches de régression de séries temporelles. À l'aide de ce réseau, nous avons conçu un nouvel estimateur neuronal vitesse-couple. Cette contribution vise à appliquer le réseau de neurones identifié et à établir des métriques d'évaluation spécifiques au domaine électrique pour analyser les performances des réseaux de neurones. Au chapitre 3, nous exploitons le même réseau pour traiter diverses applications pratiques, en nous concentrant sur des problèmes tels que la récupération lors de défauts des capteurs, la détection des défauts mécaniques et la modélisation de la température.

Après avoir montré de multiples applications des nouveaux réseaux de neurones proposés, nous examinons la fiabilité de ces méthodes dans un cadre réel. Le chapitre 4 contribue à une architecture de réseau applicable à des données réelles, employant en amont un débruiteur de signaux neuronal qui traite

des courants et des tensions bruités et les débruite afin de réaliser une estimation de couple et de vitesse. Le besoin de collecter un grand nombre de données bruitées est ainsi éliminé pour l'entrainement d'un bon réseau d'estimation vitesse-couple. Nous étudions la robustesse de cette architecture pratique d'estimation de couple-vitesse en générant des attaques adverses, concluant de l'importance d'une attaque inspirée de la physique du phénomène. Enfin, ce chapitre apporte un autre aperçu de la fiabilité de l'architecture d'estimateur vitesse-couple en vue d'une généralisation à d'autres domaines.

L'ensemble des aboutissements scientifiques et des contributions du chapitre 4 permet de nous interroger davantage sur l'utilisation réelle de l'estimateur de vitesse-couple et d'autres réseaux pour les moteurs électriques, tels qu'introduits jusqu'à présent. Dans ce but, le chapitre 5 nous mène à une nouvelle méthode basée sur l'optimisation convexe pour rendre parcimonieux des réseaux de neurones pré-entraînés. Ceci ouvre la voie à l'utilisation de l'estimateur vitesse-couple en temps réel sur des ordinateurs standards.

Acknowledgement

First and foremost, I express my deepest appreciation to my supervisors, Prof. Jean-Christophe Pesquet, Prof. Marc Castella, Dr. Nicolas Henwood, and Dr. Al Kassem Jebai, for their invaluable guidance and mentorship throughout this research. Their expertise, patience, and unwavering support have been instrumental in shaping the direction and quality of this thesis. I am truly fortunate to have had such a dedicated and knowledgeable supervisor. I am also indebted to the members of my doctoral committee, Prof. Chetan Arora, Prof. Stefan Duffner, Prof. Ismail Ben Ayed, Dr. Camille Couprie, and Prof. Antoine Berthet, for their insightful feedback, suggestions, and constructive criticism. Their expertise and academic rigor have significantly enhanced the depth and rigor of this work. I am grateful for their time and commitment to review and evaluate my research. I sincerely thank the staff and faculty of CVN, whose support and resources have been crucial in facilitating my research journey. The collaborative and intellectually stimulating environment provided a fertile ground for learning and exploration.

My heartfelt appreciation goes to my colleagues and fellow researchers, whose camaraderie and intellectual discussions have enriched my academic experience. Their support, encouragement, and willingness to share their expertise have been invaluable throughout this journey. To my parents, my wife, Kavya Gupta, and my friends, Siddharth Gupta, Aakaash Panigrahi, Suvakash Dey, Prerna Agarwal, Priyanka Gupta, and Richa Verma, I extend my deepest gratitude for their unwavering support, love, and encouragement. Their belief in my abilities and constant motivation has sustained me during moments of doubt and given me the strength to persevere.

Lastly, I express my gratitude to all the authors, scholars, and researchers whose work has laid the foundation for this study. Their contributions to the field have been instrumental in shaping the ideas and concepts presented in this thesis.

In conclusion, I am profoundly grateful to each and every individual who has played a role in this research endeavor. Your support and contributions have been invaluable, and I am honored to have had the opportunity to work with such exceptional individuals. Thank you for being part of this remarkable journey.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

The deep learning revolution has changed how humans interact with their surroundings. Computer vision applications, natural language processing, and robotics have touched every part of our lives. These dramatic change has been made possible by staggering growth in deep learning research and rapid commercialization of those research in areas like social media, urban organization, transportation, and medicine. The abundance of research can be attributed to two main factors: the vast amount of data and the accessibility to computing capable of processing large data. Such an adaptation of technologies based on deep learning is now taking place in heavy industrial sectors like steel manufacturing, automobile, mining, and space manufacturing to name a few. Within these sectors, there are numerous applications and open problems. One of the most crucial equipment used in these heavy industries is the electrical motor.

Electrical motors are so much a part of everyday life that we seldom give them a second thought. For example, when we switch on an electrical vehicle, we expect it to run rapidly up to the correct speed, provide acceleration, stop when brakes are applied, and casually predict faults to avoid future mishaps. Electrical motors have very complex dynamics and it is essential to have a controller that can provide robust control based on these dynamics. Electrical motor controllers also provide protection and supervision of the electro-mechanical system [3, 4]. These services are dependent on the dynamical physical models of electrical motors. Accurate dynamics are derived from the first principles of physics. These dynamical models depend on electrical motor physical quantities like currents, voltages, speed, fluxes, inductances, and resistances, which are measured directly or indirectly using sensors or estimated. Accurately measuring some of these quantities is hard due to noise. Operating conditions also affect some of these quantities, for example, the thermal evolution of resistances with time. Therefore, mathematical models backed by many simulations and human expert knowledge are required to develop robust controllers.

Modeling dynamics. The electrical drive industry has mastered the control aspect of electrical motors and has explored and utilized several mathematical models to deal with noise and faults. Much of the current research is focused

on finding machine learning solutions that can detect a wide array of faults in all types of motors used in all sorts of applications. The numerosity of the faults based on the motor type and its application makes deep learning an exciting and valuable path. If we can find a group of neural networks that are well suited to ingest and understand electrical motor signals then we can train those networks for different types of electrical motor tasks. Any kind of data-driven modeling using neural networks requires a large amount of data. One major challenge is the unavailability of such datasets and the time taken to create any meaningful dataset. A combination of a large amount of simulation data and small real-world data is a path forward. In this thesis, we propose advances to answer the following question:

> *How can neural networks learn electrical motor dynamics from simulated and real-world data?*

Reliability in the real world. When dealing with real-world data like electrical motor signals, we have to deal with measurement noise present in the signals. Due to the nature of the operation of the electrical motor, noisy data is generated all the time and neural networks are pretty susceptible to these noises. Another important factor at play is the generalization and robustness of neural networks when applied to such mission-critical systems. Collecting a large amount of data for all possible types of motors is economically prohibitive for even large industrial labs. Neural networks have to be robust to noise and concept drift (aging and thermal decay) to be a practically accounted for. The second challenge that we address in this thesis is the following one:

> *Can neural networks used in electrical motor tasks generalize and be robust towards noise?*

Applicability in the real world. Deep neural networks used in computer vision, automatic speech recognition, and natural language processing have different expectations when it comes to real-time usage than electrical motor applications. While their performance in various applications has matched and often exceeded human capabilities, neural networks may remain difficult to apply in real-world scenarios. Deep neural networks leverage the power of Graphical Processing Units (GPUs), which are power-hungry. Using GPUs to make billions of predictions per day, thus comes with a substantial energy cost. In addition, despite their quite fast response time, deep neural networks are not yet suitable for most real-time applications where memory-limited low-cost architectures need to be used. Given that neural networks for electrical motor tasks have to run on a low compute edge device with minimum delay, it is important that neural networks for electrical motor tasks are efficient and fast. The third problem that we explore is

> *How can we use neural networks for electrical motor tasks in real-time on inexpensive and low-resourced computing?*

1.1 . Previous Work

A short review of the electrical motor principles is followed by a recall of different approaches to the modeling of induction motor physical systems. Followed by literature on denoising time-series, generalization, and robustness of neural networks within the umbrella of reliability in the real world. Finally, making neural networks efficient and their real-time implementations are discussed.

1.1.1 . Electrical Motor

In a typical electrical motor, the following components can be found; rotor, stator, bearings, windings, rotor bars, and frame. The rotor is the moving part of an electrical motor that rotates the shaft to deliver mechanical power. The rotor usually has conductors on it to carry currents, which interact with the magnetic field of the stator. This interaction generates the forces required to rotate the shaft. Rotors are supported by bearings for smooth operation. The stator is the stationary part of a motor and contains either winding or permanent magnets. Winding is the wires that form coils to carry electrical currents and induce electromagnetic forces.

Electrical motors can be classified into different types based on power source type, internal construction, application, and type of motion output. Based on power source type, electrical motors are classified into Alternating Current (AC) and Direct Current (DC) motors. AC motors can be further classified into three types; induction, synchronous, and series-wound motors [5]. This thesis is mainly focused on induction motors.

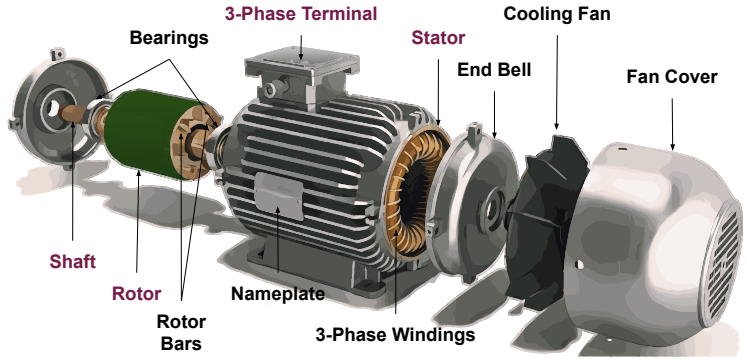


Figure 1.1: Induction Motor [1]

An induction motor is an induction machine in which the primary winding of the stator is connected to the power source, and the secondary winding of the rotor carries induced current. Figure 1.1 shows parts of an induction motor. Based on the secondary winding, induction motors can be classified into squirrel-cage induction motors and wound-rotor induction motors. In a squirrel-cage induction motor, the secondary winding contains several conducting bars having their extremities

connected by metal rings or plates at each end. In a wound-rotor induction motor, the secondary circuit contains a poly-phase winding or coils whose terminals are either short-circuited or closed through suitable circuits. Synchronous motors have a rotor spinning with coils passing magnets at the same rate as the AC which results in a magnetic field that drives it. Figure 1.2 shows mechanical and electromagnetic forces acting during an induction motor operation.

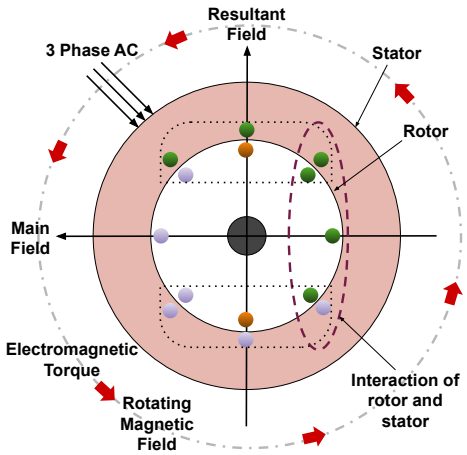


Figure 1.2: Operation of induction motor [2]

To control electrical motors, Variable Speed Drives (VSDs) can be used [3, 4]. VSDs are power converters, integrated with a power conversion system used to drive electrical motors. They are placed between the electrical grid and an electrical motor. VSDs provide controls for different use cases of the motor: a typical electrical motor can be used in centrifugal pumps, fans, conveyors, elevators, tower cranes, and electrical vehicles. The controller consists of three nested levels: control of the electrical part of the motor, control of the mechanical part of the motor, and control of the system application. The services provided by VSDs are control, protection, and supervision of the electromechanical system of an electrical motor. For these services, it is imperative to know the dynamical physical model of electrical motors. A brief introduction to reference frames and transformations of electrical motor quantities has been provided in the following subsections followed by a dynamical physical model of the induction motor.

1.1.1.1 . Reference Frames

The ubiquitous usage of induction motors across different types of industries and for different applications requires different mathematical models for analysis and development. There are three different reference frames used to represent electrical motor quantities. The three reference frames and their examples are

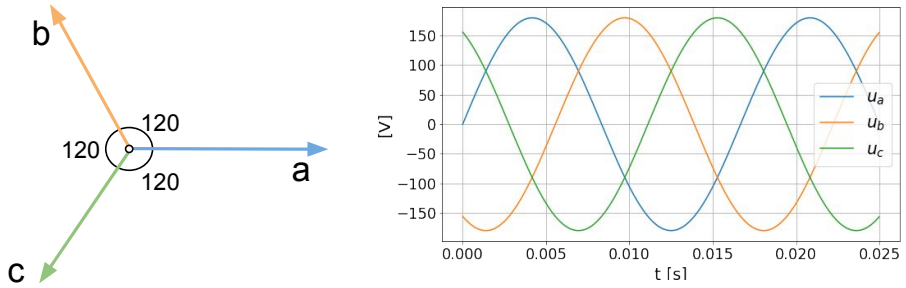


Figure 1.3: Three-phase reference frame

- **Three-phase stationary reference frame**, in which a, b, and c are co-planar axes at an angle of 120 degrees to each other.

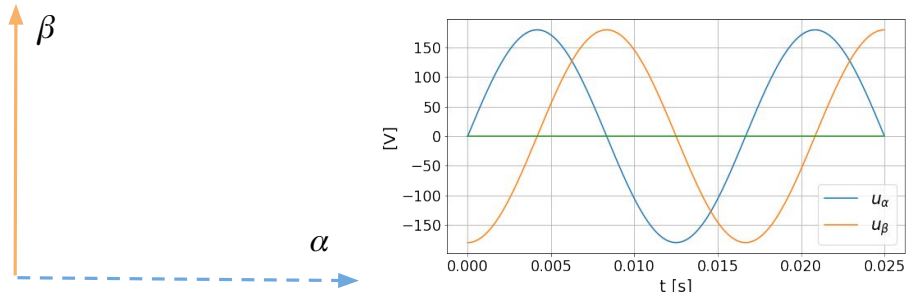


Figure 1.4: Orthogonal stationary reference frame

- **Orthogonal stationary reference frame**, in which α and β are perpendicular to each other in the same plane as the three-phase stationary reference frame.

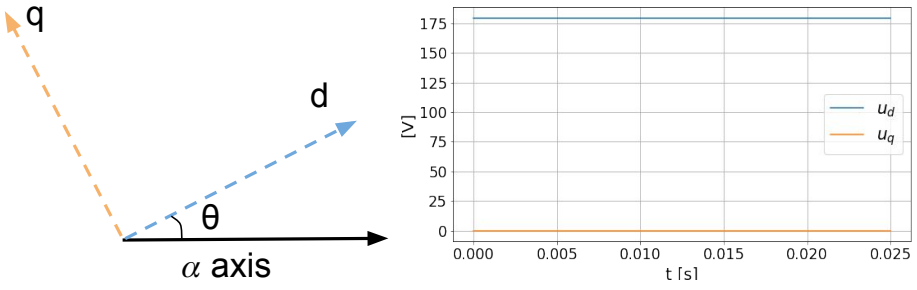


Figure 1.5: Orthogonal rotating reference frame

- **Orthogonal rotating reference frame**, in which d axis is at an angle θ (rotation angle) to the α axis and the q axis is perpendicular to the d axis.

The usage of a particular reference frame depends on the problem under consideration. One of the advantages of the dq frame is to be able to deal with constant quantities instead of sinusoidal quantities when the angle θ is well chosen. Conversion from a three-phase stationary reference frame to an orthogonal rotating reference frame via an orthogonal stationary reference frame is done using different transformations presented in the next section.

1.1.1.2 . Transformations

Concordia transformation converts the quantities from the three-phase stationary reference frame into quantities represented in the two-phase orthogonal stationary reference frame. The Concordia transformation is expressed by the following equations:

$$\begin{pmatrix} X_\alpha \\ X_\beta \end{pmatrix} = \sqrt{\frac{2}{3}} \begin{pmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \end{pmatrix} \begin{pmatrix} X_a \\ X_b \\ X_c \end{pmatrix} \tag{1.1}$$

where X_a, X_b, and X_c are the quantities represented in the three-phase stationary reference frame and X_α and X_β are the transformed quantities represented in the two-phase orthogonal stationary reference frame. Th fact that we generate only two components from three is related to the fact that the sum of components (X_a, X_b, X_c) is zero.

Inverse Concordia transformation converts quantities represented in the two-phase orthogonal stationary reference frame into the quantities represented in the three-phase stationary reference frame. The inverse Concordia transformation is expressed by the following equations:

$$\begin{pmatrix} X_a \\ X_b \\ X_c \end{pmatrix} = \sqrt{\frac{2}{3}} \begin{pmatrix} 1 & 0 \\ -1/2 & \sqrt{3}/2 \\ -1/2 & -\sqrt{3}/2 \end{pmatrix} \begin{pmatrix} X_\alpha \\ X_\beta \end{pmatrix}. \tag{1.2}$$

Rotation converts quantities represented in the two-phase orthogonal stationary reference frame into quantities represented in the two-phase orthogonal rotating reference frame. Rotation is achieved by using the following equations:

$$\begin{pmatrix} X_d \\ X_q \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} X_\alpha \\ X_\beta \end{pmatrix} \tag{1.3}$$

where X_d and X_q are the quantities represented in the two-phase orthogonal stationary reference frame, X_α and X_β are the transformed quantities represented in the two-phase orthogonal rotating reference frame quantities and θ is the rotation angle.

Inverse Rotation converts quantities represented in the two-phase orthogonal rotating reference frame into quantities represented in the two-phase orthogonal stationary reference frame. Inverse rotation is expressed by the following equations:

$$\begin{pmatrix} X_\alpha \\ X_\beta \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} X_d \\ X_q \end{pmatrix} \tag{1.4}$$

The combination of the Concordia transformation with a rotation is called the Park transformation.

1.1.1.3 . Dynamical Physical Model

The state-space model of an induction motor is presented in [6, 7, 8, 9, 10]. Modeling of electrical motors based on analytical mechanics and energy consumption is presented in [11]. This thesis uses a mathematical model of an induction motor presented in [10]. In this model, the quantities are represented in the rotating reference frame at pulsation ω_s ($d-q$ model). The fifth-order nonlinear state space model of the induction motor reads as follows:

$$\frac{J}{n_p}\frac{d}{dt}\omega_r = \frac{3}{2}n_p\Im(\psi_s^* i_s) - \tau_L \tag{1.5}$$

$$\frac{d}{dt}\psi_s = -j\omega_s\psi_s - R_s i_s + u_s \tag{1.6}$$

$$\frac{d}{dt}\psi_r = -j\omega_s\psi_r - R_r i_r + j\omega_r\psi_r \tag{1.7}$$

where $\Im(z)$ and z^* are respectively the imaginary part and conjugate of z. n_p denotes the number of pole pairs, J the motor shaft inertia, τ_L the load torque, and $u_s = u_{sd} + ju_{sq}$ the motor input voltage in the $(d-q)$ frame. ω_r (n_p times the mechanical speed), $\psi_s = \psi_{sd} + j\psi_{sq}$ (stator flux), and $\psi_r = \psi_{rd} + j\psi_{rq}$ (rotor flux) are the five state variables. The flux variables are linked to the current variables i_s (stator) and i_r (rotor) by nonlinear relationships [12] given by :

$$\begin{aligned} \psi_s &= L_{fs} i_s + \frac{L_m(i_s + i_r)}{1 + \gamma|i_s + i_r|} \\ \psi_r &= L_{fr} i_r + \frac{L_m(i_s + i_r)}{1 + \gamma|i_s + i_r|} \end{aligned} \tag{1.8}$$

The parameters are the stator and rotor resistances R_s and R_r, the stator and rotor leakage inductances L_{fs} and L_{fr}, and the magnetic saturation coupling parameters L_m and γ.

1.1.2 . Data-Driven Modeling of Physical Systems

Understanding physical processes from data without having the underlying physical model is very hard. The abundance of data in both natural and physical sciences has enabled the use of machine learning models to understand the governing dynamics of many complex processes. There are different ways of modeling physical systems. Existing research can be grouped into the following

areas; a) the problem of understanding the physical process from data, b) classifying or predicting complex physical processes, c) using physics to generate simulation data for machine learning models, d) using machine learning to control non-linear dynamical systems, and e) using machine learning to detect anomalies in dynamical systems. This thesis focuses on the data-driven modeling of electrical motor dynamics. The following subsections summarize existing research on different ways of modeling a time-series physical system.

1.1.2.1 . Machine Learning

Hidden Markov Model (HMM), or Kalman filter can learn linear dynamic models. For non-linear dynamics, accommodating non-linearity into HMM is non-trivial. In [6] a new method called sufficient posterior representation is presented which can be used to model non-linear dynamic behaviors using many non-linear supervised learning algorithms such as neural networks, boosting, and support vector machine (SVM) in a simple and unified fashion. Most of the methods of data-driven learning of dynamic systems deal with sequential data. A method has been presented in [8] to learn dynamics from non-sequential data. Another approach to modeling a physical system is to represent it in the form of Partial Differential Equations (PDEs). PDEs can describe complex phenomena. We do not always have a PDE model for a given problem, but we may have a large amount of data available. In [13] a data-driven method is proposed to learn the governing PDEs of a given system from time series data. Sparse regression is used to learn the coefficients and an iterative method is employed to get the most suitable coefficients.

1.1.2.2 . Black Box Modeling using Neural Networks

Neural networks to model physical phenomena in a black box approach was presented in [7]. The paper presents a way of modeling time-invariant non-linear systems. A multi-layered network architecture with a control input signal called Hidden Control Neural Network (HCNN) is presented which can model signals generated by non-linear dynamical systems with restricted time variability. Recently, deep neural networks have been used in learning physical dynamics from data in a range of applications e.g., calorimetry [14], drone landing [15], and nonlinear dynamics identification [16]. Karpatne et al. presents a physics-guided neural network (PGNN) that leverages the output of physics-based model simulations along with observational features to generate predictions using a neural network [17]. Furthermore, they present a novel framework for using physics-based loss functions in the learning objective of neural networks, in order to ensure that the model predictions not only show lower errors on the training set but are also scientifically consistent with the known physics on the unlabeled set. There are systems where dynamics change with time and some dynamics may not have been seen before. Identifying new dynamics is useful. [18] uses neural networks to identify new physics laws.

1.1.2.3 . Interpretable Modeling using Neural Networks

For some physical systems, we want weak governing dynamics in the form of equations. A neural network trained on physical system data does not provide a good representation in the form of equations. [19, 16] have presented methods that can be used to find weak governing dynamics in the form of equations or sparse matrices. Computing hidden system parameters from measurable quantities of complex physical systems using an Invertible Neural Network (INN) is presented in [19]. In [16] a data-driven approach for approximating nonlinear dynamics to a linear one using deep neural networks has been proposed. Koopman operators [20] are learned from data for the coordinate transformation of a nonlinear system to a linear one. Koopman operator is a linear operator C_ϕ defined by the rule $C_\phi(f) = f \circ \phi$, where $\circ$ denotes function composition. Other nonlinear to linear transformation methods are presented in [18, 7].

It is crucial to have a machine learning model consistent with the physics of the dynamic system. [21] has shown how physics can be used to make better data-driven discoveries. Theory-guided design, learning, and refinement of the machine learning model have been presented. In [22], a physics-guided neural network (PGNN) is presented, which leverages the output of physics-based model simulations and observational features to generate predictions using a neural network. The model predictions show lower errors in the training data and are consistent with the system dynamics. [23] uses machine learning to optimize physical dynamic systems.

1.1.3 . Reliability of Neural Networks

Neural networks are susceptible to noisy inputs. Denoising is a trivial task if the noise parameters of the system are known. In the case of electrical motors, such information can be collected by identifying the noise source or collecting some data and doing noise modeling. Then denoisers can be designed using these parameters. Another common problem in the case of neural networks is the robustness against input perturbations. This is an active research area with a lot of work on adversarial training and certifications of networks. From a reliability point of view, generalization capability of a network is a requirement. The following subsections summarize the literature on time-series denoising, robustness, and generalization of neural networks.

1.1.3.1 . Denoising Time-Series

Noise reduction in time signals is a very evolved field with a multitude of research involving various methodologies. Some of the techniques that can be easily applied are linear smoothing filters and non-linear filters [24, 25]. Kalman filter [26, 27, 28] is widely used in noisy observation where a state-space based estimation is done which takes the system model as input. Also, there are transform-based methods like those based on wavelet transforms [29, 30] which remove noisy components from transformed sensor data. Variational methods

often based on the total variation [31, 32, 33] have also been used in signal denoising and change detection, providing a robust and often more flexible solution over linear filter-based denoisers. Kalman filter, transform-based and variational methods require prior knowledge about the noise/signal statistics for efficient denoising. Deep learning methods, e.g. stacked autoencoders [34], are also used.

1.1.3.2 . Robustness of Neural Networks

Testing instability of neural networks is a well known and active area of research leading to a more explainable and trustable A.I. In [35], the concept of adversarial attacks was first proposed to fool neural networks. Adding a well-crafted subtle perturbation to the input of the neural network produces a misclassification. This scenario is possible even when the model has good clean accuracy. These attacks pose a huge threat to the performance of neural networks. There have been multitude of works introducing stronger adversarial attacks and their defenses. Goodfellow et al. [36] proposed Fast Gradient Sign Method (FGSM) to generate ℓ_∞ bounded adversarial attacks. This is a white box attack i.e it has access to network structure, parameter weights as well as all the related training details. The generated inputs are misclassified by adding perturbations and linearizing the cost function in the gradient direction. FGSM is a single step attack, Madry et al. [37] proposed a multi-step variant of FGSM called Projected Gradient Descent (PGD) attack.

1.1.3.3 . Generalization of Neural Networks

Deep neural networks often have far more trainable model parameters than the number of samples they are trained on. Some of these models exhibit small generalization errors, i.e., the difference between “training error" and “test error". But when such networks are used in the real world, there are several cases where they do not perform well. These cases can occur due to unseen distributions, concept drifts, etc. This leads to several interesting questions, a) how generalized a neural network is for the problem it is trained for, b) how can concept drift be detected, and c) how to generalize neural networks without training it exhaustively for all scenarios. To answer such questions, statistical learning theory has proposed several different complexity measures that are capable of controlling generalization errors. These include Vapnik–Chervonenkis (VC) dimension [38], Rademacher complexity [39], and uniform stability [40, 41, 42].

1.1.4 . Efficient Real Time Inference

Neural networks trained for real-time tasks like electrical motor fault detection have constraints around inference speed and available computing resouurces. Such constraints have been met in applications of neural networks in computer vision and natural language processing (NLP) problems. There are several works in pruning neural networks to bring down their excessive number of parameters. Some works

are more application-oriented, where all constraints like computing time, network type, accuracy, and speed are considered all at once to automatically search for the best network. These two areas of research are summarised in the following subsections.

1.1.4.1 . Pruning Neural Networks

Methods inducing sparsity in a pre-trained network involve multiple pruning and fine-tuning cycles till desired sparsity and accuracy are reached [43, 44, 45, 46, 47, 48, 49]. [50] proposed weight rewinding technique instead of vanilla fine-tuning post-pruning. Another popular approach has been to induce sparsity during training. This is achieved by modifying the loss function to consider sparsity as part of the optimization [51, 52, 53, 54]. [55] showed that it is possible to find sparse sub-networks that, when trained from scratch, were able to match or even outperform their dense counterparts. [56] presented single-shot network pruning (SNIP), a method to estimate, at initialization, the importance that each weight could have later during training. In [57] the authors perform a theoretical study of pruning at initialization from a signal propagation perspective, focusing on the initialization scheme.

1.1.4.2 . Searching for Efficient Networks

Hardware-aware network architecture search (NAS) methods [58, 59, 60, 61, 62, 63, 64] directly incorporate the hardware feedback into efficient neural architecture search. [65] proposes to learn a single network composed of a large number of subnetworks from which a hardware-aware subnetwork can be extracted in linear time. [66] proposes a similar approach wherein they identify subnetworks that can be run efficiently on microcontrollers (MCUs).

1.2 . Contributions of the Thesis

A list of contributions of this thesis follows. The first three entries below are contributions towards answering the first question on learning electrical motor dynamics using neural networks from data, followed by contributions on the reliability and applicability of these networks in the real world.

Input-Output relationship modeling. A principle contribution of this thesis is the data-driven modeling of electrical motor dynamics. This is established by learning the input-output relationship between various quantities of induction motors using different neural networks. A new network architecture is introduced that combines the benefits of convolutions and sequential layers for time-series regression tasks.

Speed-Torque estimator. One important application of identified neural network that can learn the relationship between different electrical motor

quantities is estimating speed and torque from currents and voltages. This contribution is towards the application of the identified neural network in the electrical motor and establishing a suitable pipeline to analyze the performance using proper electrical engineering metrics.

Utilizing Identified Model for Other Electrical Motor Applications. Input-output relationship model is utilized in several other practical tasks like sensor fault recovery, mechanical fault detection, and temperature modeling.

Denoising currents and voltages. This contribution enables us to use a speed-torque estimator trained on a large amount of simulated data directly in real-world settings. The outcome of this work is a neural network-based time-series signal denoiser that takes real-world noisy currents and voltages and denoises them for speed-torque estimator. This removes the need of collecting a large amount of real noisy data to train a good speed-torque estimator.

Robustness of speed-torque estimator. The behavior of neural networks to perturbations of its input can be done by generating adversarial attacks using FGSM [36] and PGD [37] attackers. These methods are not suitable for electrical motor quantities. This contribution shows why some of the existing methods are not good attackers for speed-torque estimators and pitches the requirement of a physical dynamics-based attacker.

Generalization of the speed-torque estimator. Much of the research on the generalization of neural networks is based on measuring the generalization bounds of trained networks. This contribution analyzes the generalization capability of the speed-torque estimator to different power motors.

Pruning of pre-trained neural networks. This contribution leads to a new convex optimization-based method for sparsifying pre-trained neural networks. This provides a sound approach offering convergence guarantees and opens the avenue of utilizing the speed-torque estimator in a real-time fashion on low compute devices.

1.3 . Organisation of the Thesis

This thesis is organised as follows.

i) In Chapter 2, we introduce the problem of modeling the input-output relationship between different quantities using neural networks as a way of modeling the system dynamics of electrical motors. We introduce a new encoder-decoder architecture that is well suited for electrical motor time-series signals. This encoder-decoder architecture is designed to create a speed-torque estimator with careful consideration of its performance from an electrical engineering perspective.

ii) In Chapter 3 we discuss three different applications of the proposed encoder-decoder architecture. A major focus in on the problem of sensor fault recovery. We show how GANs can be trained by using the pre-trained encoder-decoder networks as the backbone of generators and decoders to identify incipient faults present in sensor recording of electrical motor quantities. We demonstrate how the proposed architecture can be used for mechanical fault detection and temperature modeling in electrical motors.

iii) In Chapter 4, we focus on the reliability of the neural networks under different conditions for the applications above. We show that we can remove several data and real-time usage challenges by handling noise before speed-torque estimation. We also show the robust behavior of our proposed networks under different adversarial attacks. At last, we show the generalization capability of our speed-torque network and how the performance changes when we introduce different powered motor test data.

iv) In Chapter 5, we deal with the problem of utilizing our speed-torque network in the real world in a resource-limited setting. We propose a novel rigorous optimization approach for puning a pre-trained network. Our method takes advantage of recent advances in the mathematical understanding of neural networks which have shown that most of the activation functions used in neural networks are proximity operators of convex functions.

v) Finally, we conclude in Chapter 6 and present future work based on the contributions of this thesis.

1.4 . List of Publications

i) **S. Verma** and K. Gupta. *Robustness of Neural Networks used in Electrical Motor Time-Series*, NeurIPS Workshop 2022. [67]

ii) **S. Verma**, N. Henwood, M. Castella, JC Pesquet, and AK Jebai, *Can GANs Recover Faults in Electrical Motor Sensors?*, ICLR Workshop 2022. [68]

iii) **S. Verma**, N. Henwood, M. Castella, AK Jebai, and JC Pesquet, *Neural Speed-Torque Estimator for Induction Motors in the Presence of Measurement Noise*, IEEE Journal of Transactions on Industrial Electronics 2022. [69]

iv) **S. Verma** and JC Pesquet, *Sparsifying Networks via Subdifferential Inclusion*, ICML 2021. [70]

v) **S. Verma**, N. Henwood, M. Castella, AK Jebai, and JC Pesquet, *Neural Networks based Speed-Torque Estimators for Induction Motors and Performance Metrics*, IECON 2020. [71]

vi) **S. Verma**, N. Henwood, M. Castella, F. Malrait, and JC Pesquet, *Modeling Electrical Motor Dynamics using Encoder-Decoder with Recurrent Skip Connection*, AAAI 2020. [72]

vii) **Lassau et al.**, *Integrating deep learning CT-scan model, biological and clinical variables to predict severity of COVID-19 patients*, Nature Communications, 2021. [73]

Available codes:

i) Motor Dynamics Simulation: https://github.com/sagarverma/MotorSim

ii) Metrics Library for Motor: https://github.com/sagarverma/MotorMetrics

iii) Trajectory Generator: https://github.com/sagarverma/MotorRefGen

iv) Motor Dynamics: https://github.com/sagarverma/MotorDynamics

v) Robust Motor NN: https://github.com/sagarverma/robust-motor

Chapter 2

Data Driven Modeling of Input-Output Relationships

2.1 . Introduction

An electrical motor dynamics depends on physical quantities like currents, voltages, speed, fluxes, inductances, and resistances, measured directly or indirectly using sensors or estimators. Accurately measuring some of these quantities is challenging due to noise. Operating conditions also affect some of these quantities, for example, the thermal evolution of resistances with time. To better understand the behavior of electrical motor dynamics under different operating conditions, one can model the relationships between different electrical and mechanical quantities: currents (i_d, i_q), voltages (u_d, u_q), speed (ω_r), and estimated torque (τ_{em}). Figure 2.1 shows the first 40 seconds of a simulated sample collected using a Simulink model based on [10]. Figure 2.2 shows the first 40 seconds of a 4kW induction motor operation.

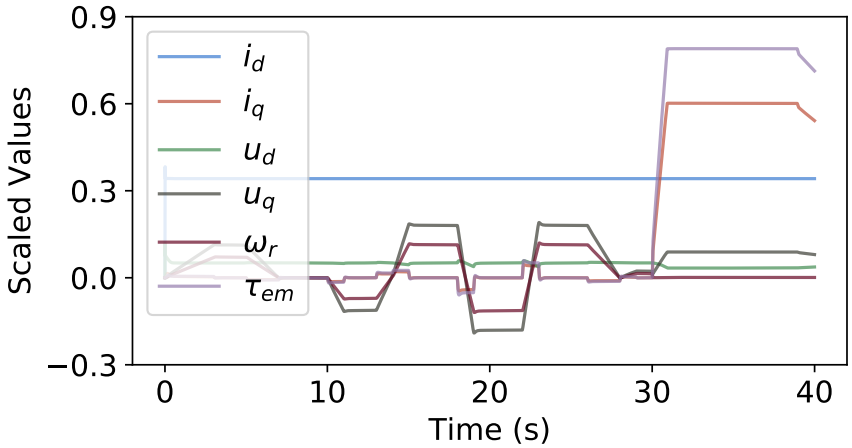


Figure 2.1: First 40 seconds of a simulated electrical motor operation.

Modeling time-varying systems can be treated as a sequence-to-sequence

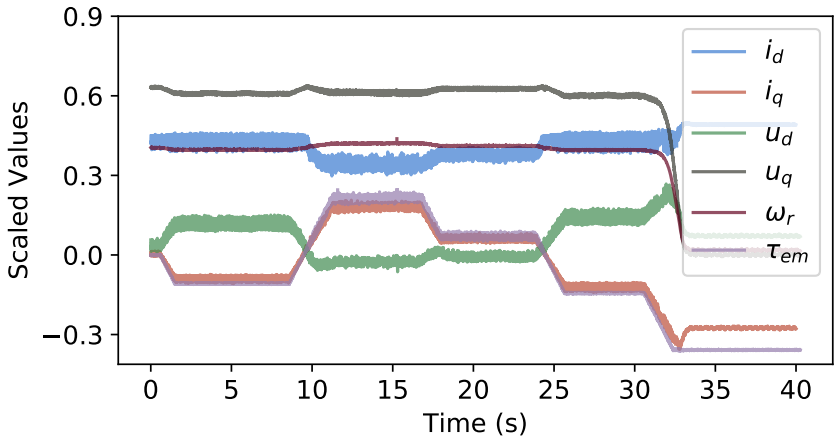


Figure 2.2: First 40 seconds of a real-world electrical motor operation.

prediction problem. For such problems, end-to-end learning of temporal dynamics from time-series data has been made easier thanks to methods like Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Long-Short Term Memory (LSTM) structures. It has been shown that RNNs and LSTMs can model complex nonlinear interactions by providing a large amount of multidimensional data. RNNs have been shown to perform hierarchical processing of time series with different layers tackling different time scales [74, 75]. Such sequential architectures can be utilized to learn electrical motor dynamics. Sequential networks like RNN and Gated Recurrent Unit (GRU) have fewer parameters than LSTM. There are several ways of obtaining fewer parameters in sequential networks. One such method is the diagonalization of weights in RNN [76]. Miller et al. [77] were the first to show that feed-forward and convolutional networks can also achieve stable sequential models.

In this chapter, we first explore the applicability of one-dimensional CNNs to our problem of modeling input-output relationships. This resulted in a new network architecture called "DiagBiRNN" published at AAAI 2020. This network is then utilized as a speed-torque estimator from currents and voltages in a subsequent work published at IECON 2021.

2.2 . Related Work

Neural Networks for Physics: The first use of neural networks to model physical phenomena was presented in [7]. This paper presents a multi-layered neural network for nonlinear prediction and system modeling from time-series data. Recently, deep neural networks have been used in learning physical dynamics from data in a range of applications, e.g., calorimetry [14], drone landing [15], and nonlinear dynamics identification [16]. Karpatne et al. [17] presents a physics-guided neural network that leverages the output of physics-based model simulations and observational

features to generate predictions using a neural network architecture. Furthermore, they present physics-based loss functions to ensure that the model predictions show lower errors on the training set and are scientifically consistent with the known physics.

Neural Networks for Time Series: RNN and LSTMs are very good at learning hidden temporal dynamics from data in various applications such as wind speed forecasting [78], estimating missing measurements in time series [79], and consumer event forecasting [80]. Convolutional architectures have recently been shown to be competitive on many sequence modeling tasks compared to the de-facto standard of recurrent neural networks (RNNs) while providing computational and modeling advantages related to inherent parallelism. In [81], the authors provide an empirical comparison between convolutional and recurrent networks in modeling time series. Aksan et al. [82] present a stochastic variant of the temporal convolutional network, which performs better than stochastic RNNs. Miller et al. have shown that in some cases, feed-forward networks are better at modeling temporal patterns than sequential networks [77]. In time-series prediction, different events often have different importance. This can be addressed by using an asymmetric loss function, which weights distinct parts of the signal differently, as shown in [83, 84, 85].

Pitfalls in using Neural Networks: The reasons why neural networks may fail are analyzed in [86]. Neural networks make fewer assumptions, have a large number of parameters, and have different modeling processes, creating more risks for inappropriate uses and unstable applications. Another major pitfall consists in treating neural networks as black boxes. In [87], a robustification technique is proposed to interpret neural network results regarding the input effects and interactions among input variables. Underfitting and overfitting are also well-known problems in machine learning methods. Neural networks are prone to these problems due to their data-hungry nature and their large number of parameters, as discussed in [88].

2.3 . Neural Network Architectures for Input-Output Modeling

	Feed-Forward	**RNN**	**LSTM**	**CNN**
Depth	3 Linear 4 Linear	1 RNN $\rightarrow$ 2 Linear 1 RNN $\rightarrow$ 3 Linear	1 LSTM $\rightarrow$ 2 Linear 1 LSTM $\rightarrow$ 3 Linear	3 Conv $\rightarrow$ 2 Linear 4 Conv $\rightarrow$ 2 Linear
Input	Flattened vector	Channelized	Channelized	Channelized
Output	Middle value	Input length	Input length	Middle value

Table 2.1: Architectural details of the benchmark models.

In this section, we discuss several benchmark methods that are derivatives of widely used standard neural networks. Broadly, feed-forward network, CNN, RNN, and LSTM structures are evaluated. Table 2.1 shows all the benchmark networks.

For each type of network, two variations exist namely, shallow and deep, to evaluate the effect of the network depth on their learning capability. The names "shallow" and "deep" are just for the namesake and do not depict a significant big change in the depth of the network.

2.3.1 . Fully Connected Networks

We use feed-forward neural networks (FNNs) to show that the proposed problem and dataset are quite difficult and that FNNs have limited learning capabilities. Column 1 in Table 2.1 provides the configuration details of the two experimented networks also shown in Figure 2.3.

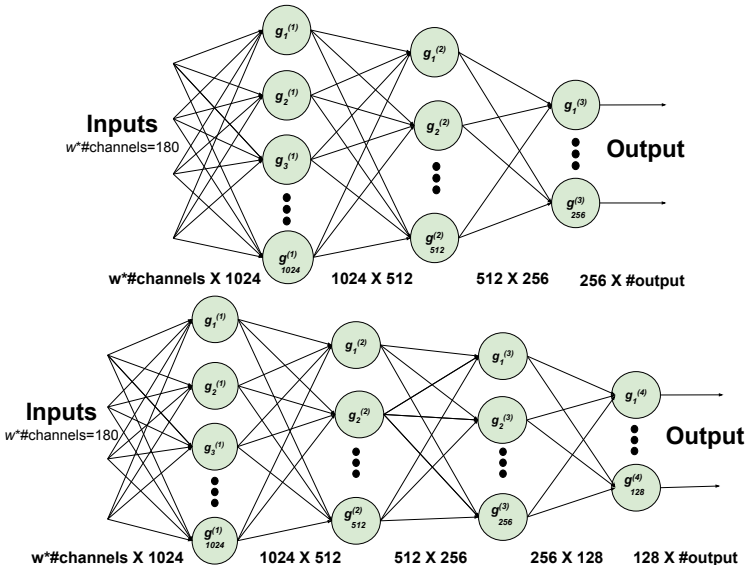


Figure 2.3: Three and four layered feed-forward networks.

2.3.2 . Sequential Networks

Sequential neural networks have been used widely to learn from sequential data. RNNs and LSTMs are two of the most commonly used sequential neural networks. Configuration details of RNN and LSTM networks are shown in columns 2 and 3 in Table 2.1, respectively. Figure 2.4 shows LSTM followed by two linear layers and Figure 2.5 shows LSTM followed by three linear layers.

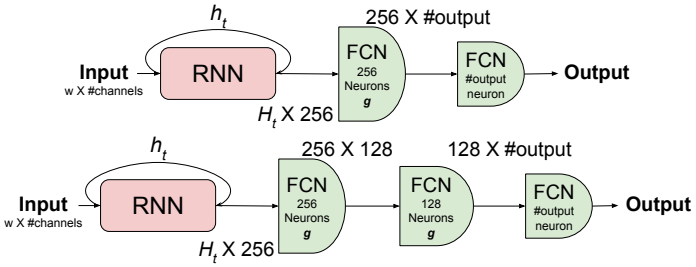


Figure 2.4: Recurrent layer followed by two and three linear layers.

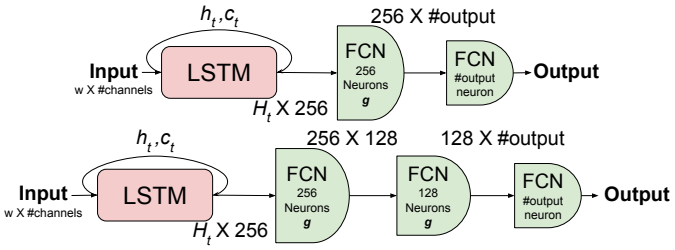


Figure 2.5: LSTM layer followed by two and three linear layers.

2.3.3 . 1D Convolutional Networks

FNNs have very limited learning capabilities when the input data is complex like sequential or multidimensional. Recently, CNNs have been shown to provide competitive performances on sequential data. The configuration of the benchmark for CNNs is shown in Column 4 of Table 2.1. Figure 2.6 shows two CNN variants.

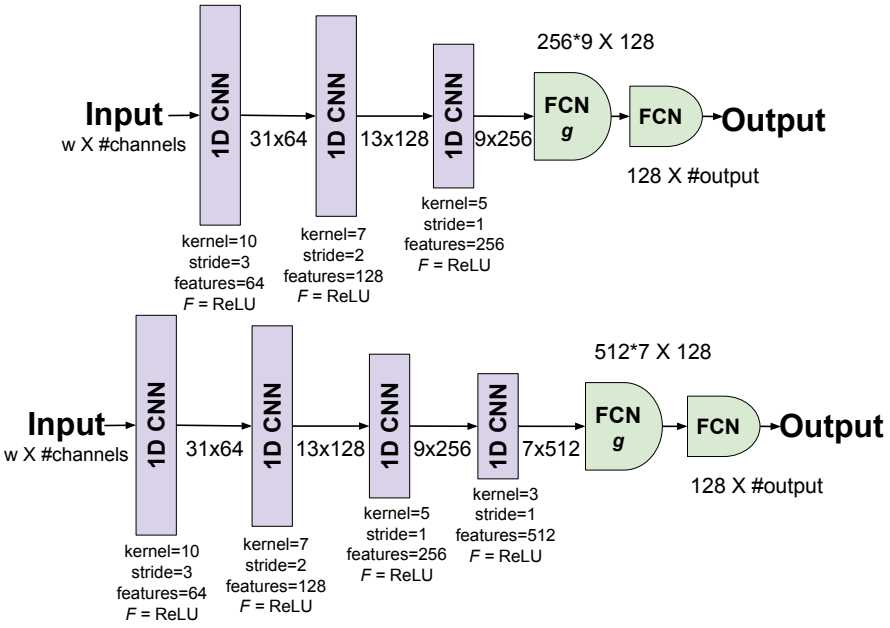


Figure 2.6: Three and four convolution layers followed by two linear layers.

2.3.4 . Encoder-Decoder Networks

Traditionally, sequential networks have been used to model temporal dynamics. In our experiments, we found that RNNs and LSTMs do not provide as good learning capability as one-dimensional CNNs. Since our task is to perform multivariate prediction over the same length as the input, we use an architecture where all layers are made of convolutions. We then carefully introduce several intuitive modifications to the encoder-decoder architecture which leads to a performing and efficient model.

Encoder-Decoder Network: To capture temporal dynamics from the complete input and output window, we introduce the encoder-decoder network shown in Figure 2.7. It consists of encoding and decoding blocks with convolutional and deconvolutional layers, respectively. The convolutional and deconvolutional blocks are followed by ReLU activations. We do not use pooling as in our experiments we found that they deteriorate the results.

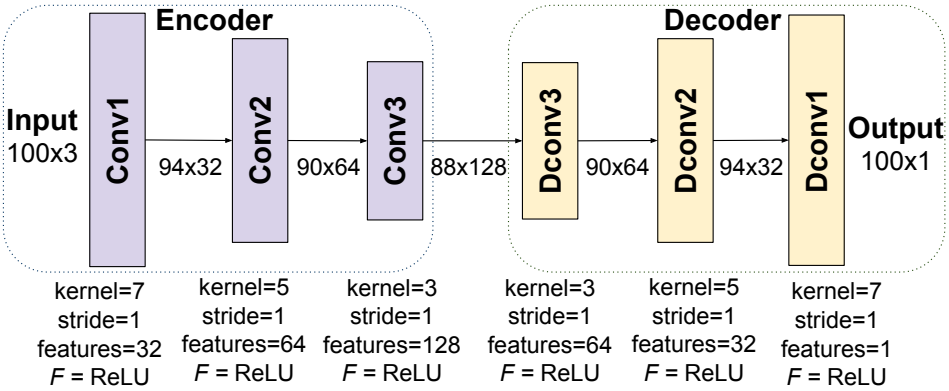


Figure 2.7: Three convolution layered encoder followed by three deconvolution layered decoder.

Encoder-Decoder Network with Skip Connection: It has been shown that adding a skip connection to the encoder-decoder helps in transferring high-level features directly from one encoding layer to its corresponding decoding layer [89]. We also introduce skip connections between encoding and decoding layers as shown in Figure 2.8.

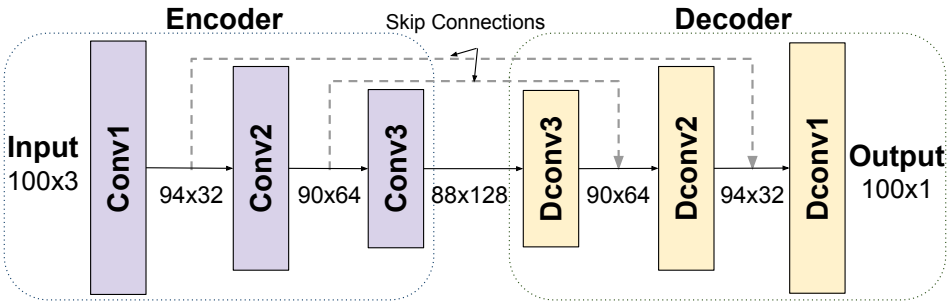


Figure 2.8: Encoder-decoder network with skip connections.

Encoder-Decoder Network with Recurrent Skip Connection: Convolution operations are windowed over the kernel size, this means that convolution cannot learn temporal relationships which are out of the kernel sized windows. Adding recurrent layer over convolutional features can overcome this issue. This also

helps in learning temporal patterns in the latent space. We add recurrent layers after every encoding layers as shown in Figure 2.9. The output of the recurrent layer is then sent to the corresponding decoding layers.

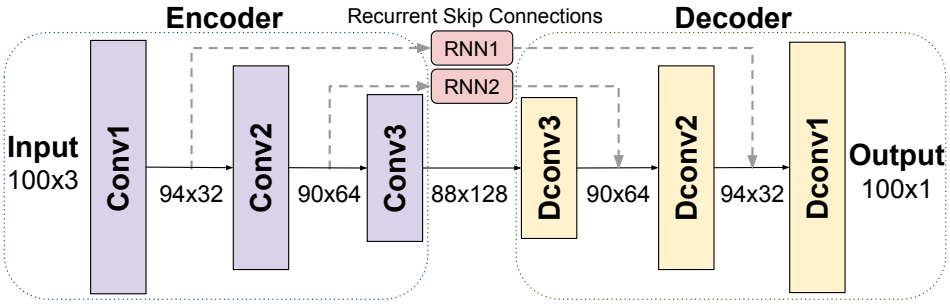


Figure 2.9: Encoder-decoder network with RNN as skip connections.

Encoder-Decoder Network with Bidirectional Recurrent Skip Connection: Bidirectional RNNs help to learn temporal patterns in both directions. For our use case, we want to predict each time step of the input window. Therefore, we also use bidirectional RNNs.

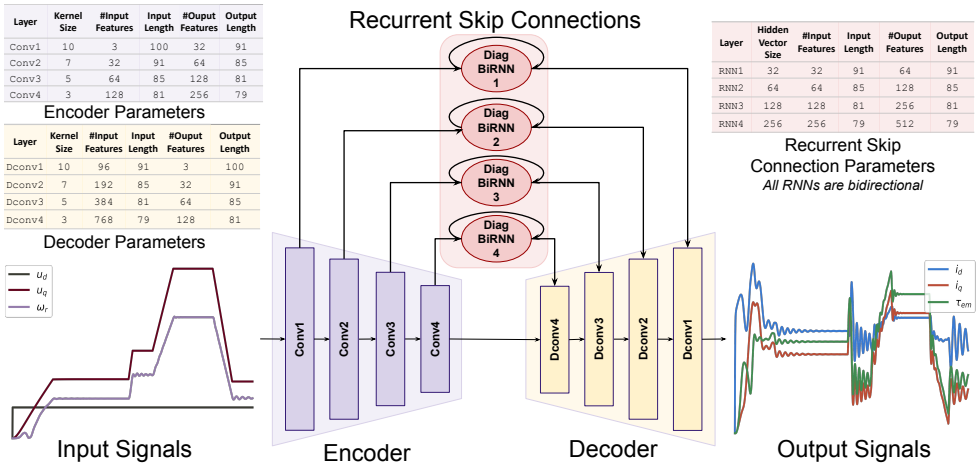


Figure 2.10: Proposed DiagBiRNN architecture.

Encoder-Decoder Network with Bidirectional Diagonalized Recurrent Skip Connection: Vanilla RNNs have a high number of parameters due to matrix multiplications between weights and features. Diagonalizing weights in the recurrent unit decreases the number of parameters. Figure 2.10 shows this architecture.

The hidden state update equation of an RNN is given by

$$h_t = \tanh(Wx_t + Uh_{t-1} + b) \tag{2.1}$$

where $x_t \in \mathbb{R}^M$ and $h_t \in \mathbb{R}^N$ are the input and hidden state at time t, respectively. $W \in \mathbb{R}^{N\times M}$, $U \in \mathbb{R}^{N\times N}$, and $b \in \mathbb{R}^N$ are the weights for the input and the hidden vector, and the bias of the neurons. We propose to impose diagonal structures in the weight matrices W and U by setting $N = M$ and impose a diagonal structure parameterized by diagonal weight vectors w and u, respectively. Practically, this amounts to replacing the matrix multiplication operations between W and U with Hadamard products $\odot$ between the vectors w and u. The diagonalized recurrent network is described as

$$h_t = \tanh(w \odot x_t + u \odot h_{t-1} + b) \tag{2.2}$$

where $w \in \mathbb{R}^M$ and $u \in \mathbb{R}^M$, are input weights, and $b \in \mathbb{R}^M$ is a bias vector.

2.3.5 . Total Variation Weighted Mean Square Loss

In real world usage of electrical motors, large variations in the signals occurs less often than small variations. We observe this effect in our dataset, which causes model bias toward small variations when trained with mean square loss. This is not a desirable behavior if the learned model is used in controllers. To avoid this problem, we propose a novel asymmetric loss function that takes into account the signal variations. The proposed loss function takes a signal $y_i = (y_t^i)_{1\leqslant t\leqslant T}$ of duration T and computes the mean squared error with the predicted signal $\hat{y}_i$, the MSE value is then multiplied with the total variation of signal y_i. The index of a single sample taken from the dataset of length N samples is denoted by i in the expression below:

$$\mathcal{L}_{\text{TV-MSE}} = \frac{1}{N}\sum_{i=1}^{N}\left(\underbrace{\left(\sum_{t=1}^{T-1}|y_t^i - y_{t+1}^i|\right)}_{\text{Total Variation}}\overbrace{\left(\frac{1}{T}\sum_{t=1}^{T}(y_t^i - \hat{y}_t^i)^2\right)}^{\text{MSE}}\right) \tag{2.3}$$

2.4 . Evaluation Procedure

There are standard metrics to judge machine learning methods for regression tasks. These metrics provide compelling global insights. Electrical engineering, especially in the induction motor field, has a different way of evaluating performance. In this section, we present both machine learning and electrical engineering metrics.

2.4.1 . Machine Learning Metrics

To evaluate the capability of the proposed method, we use different metrics allowing us to compare the performance at global and local scope of the input signal.

To analyse the learning capability at global scope, we report mean absolute error (MAE), symmetric mean absolute percentage error (SMAPE), and coefficient of determination R^2 [90]. Best MAE and SMAPE values are ideally 0 and R^2 should be 1. For a signal y of duration T, the model prediction is given by $\hat{y}$. y_t is the ground truth at time t and $\hat{y}_t$ is the predicted output of the model at time t. $\bar{y}$ denotes the mean of ground truth y. More precisely, consider the following definitions:

$$\text{MAE}(y, \hat{y}) = \frac{1}{T} \sum_{t=1}^{T} |y_t - \hat{y}_t| \tag{2.4}$$

$$\text{SMAPE}(y, \hat{y}) = \frac{100}{T} \sum_{t=1}^{T} \frac{|\hat{y}_t - y_t|}{|\hat{y}_t| + |y_t|} \tag{2.5}$$

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{t=1}^{T} (\hat{y}_t - \bar{y})^2}{\sum_{t=1}^{T} (y_t - \bar{y})^2} \tag{2.6}$$

$$\text{SC}_y = \sum_{t=1}^{W-1} |y_t - y_{t-1}| \tag{2.7}$$

MAE, SMAPE, and R^2 values do not provide enough information about the signal parts where the model is performing poorly or very well. Thus, we compute the signal complexity (SC) on sliding windows of length 100 over the ground truth signal and plot it against the corresponding window's SMAPE value computed between the ground truth and the predicted signal. All metrics are reported on the original range of the respective quantities after re-scaling.

2.4.2 . Electrical Engineering Performance Metrics

A more insightful way of evaluating the performance from an industrial standpoint is to compute widely used electrical engineering metrics. Figures 2.11 and 2.12 show where on the timeline of a signal performance metrics are calculated. The following metrics have been used for the response signal to a speed or torque reference ramp (whose amplitude is the absolute difference between the starting and target values):

- **2% response time ($t_{2\%}$)** is the time value at which the response signal has covered 2% of the ramp amplitude.
- **95% response time ($t_{95\%}$)** is the time value after which the response signal remains at less than 5% of the ramp amplitude from the target value.
- **Overshoot ($D\%$)** is the difference between the maximum peak value of the response signal and the final steady-state value. It is expressed in the percentage of the ramp amplitude.

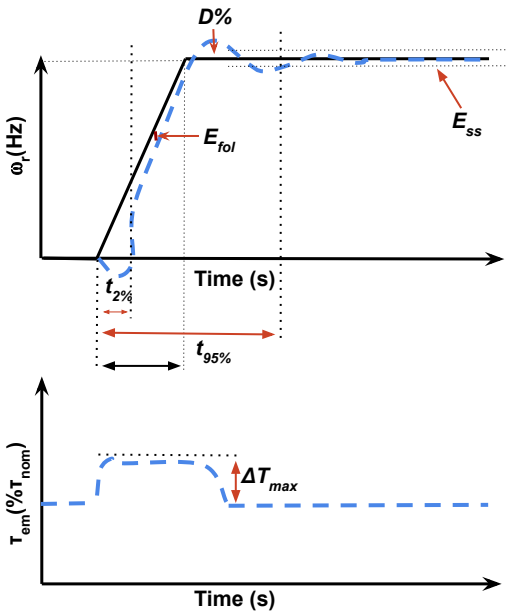


Figure 2.11: Performance metrics for speed ramp.

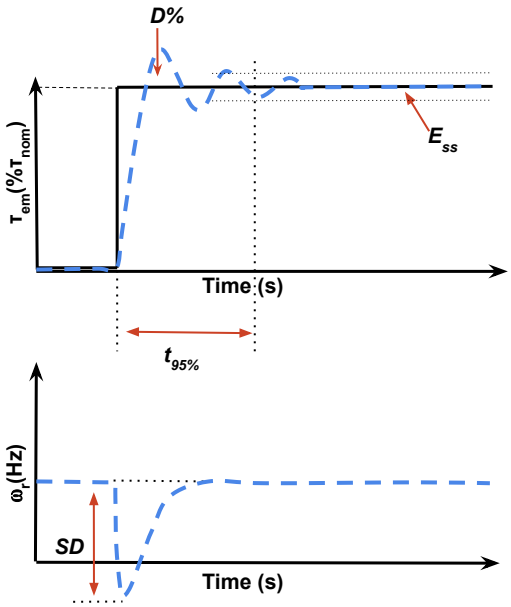


Figure 2.12: Performance metrics for torque step.

- **Steady-state error** (E_{ss}) is the difference between the response signal and target values once the steady-state has been reached.
- **Following error** (E_{fol}) is the difference between the reference and response signal values when the reference value has covered 50% of the ramp amplitude.

- **Maximum acceleration torque (speed ramp)** ($\Delta\tau_{max}$) is the max response torque deviation during the speed ramp.
- **Speed drop (torque ramp)** (SD) is the max response speed deviation during the torque ramp.
- **Maximum Absolute Error** (E_{max}) is the maximum absolute difference between the real speed and the model predicted speed.

2.5 . Dataset

To train deep neural networks, we need a large amount of data. It is laborious to capture real motor data. To overcome this issue, we use a large amount of simulated data and a small amount of real motor data. The following subsections describe how simulated and real data have been collected.

2.5.1 . Reference Trajectory Generator

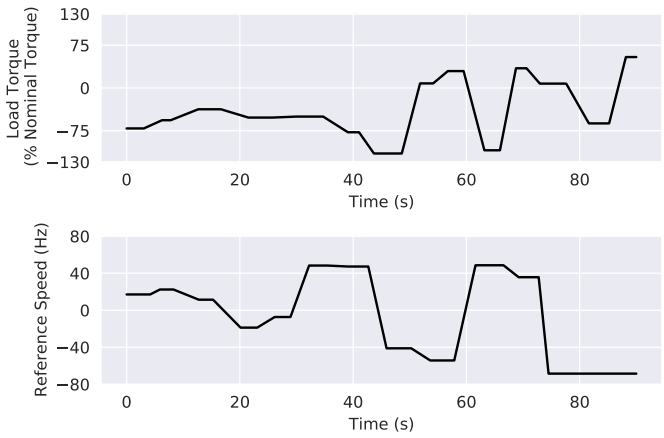
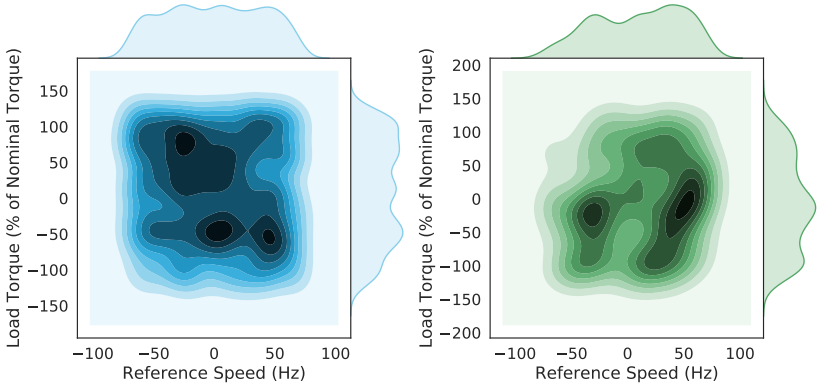


Figure 2.13: Reference speed and load torque trajectories from one of the training samples.

To generate simulated training and validation sets, we created a trajectory generator that generates realistic reference speed and load torque trajectories. For every simulation, the number of static states is drawn randomly from a uniform distribution between 5 and 15. The duration of a static state in a simulation is drawn from a uniform distribution between 1 and 5 seconds. Ramp duration between two consecutive static states is generated according to a shifted truncated exponential distribution between 4 and 2000 milliseconds to provide more frequent short-duration ramps. For each static state, speed and load torque values are generated according to a uniform distribution on [-70, 70] Hz and [-120, 120] % of nominal torque ($\%\tau_{nom}$), respectively. Figure 2.13 shows a sample reference trajectory from the training set.

2.5.2 . Training and Validation Set



(a) Training zone (b) Validation zone

Figure 2.14: Torque vs speed plans for all the simulations in training set and validation set. Density here shows the number of samples that belong to a zone.

We use our reference trajectory generator to generate 100 simulated paths totaling about 1000 speed and torque ramps in 150 minutes for the training set and 50 simulations totaling about 200 ramps in 30 minutes for the validation set. Torque-speed plan density plots for training and validation zones are shown in Figure 2.14. We then simulate these trajectories using our Simulink model of a 4kW induction motor and collect simulation data every 4ms. The simulation dataset consists of the following electrical quantities: currents i_{sd} and i_{sq}, voltages u_{sd} and u_{sq} acting as inputs, and rotor speed ω_r and electromagnetic torque τ_{em} acting as outputs.

2.5.3 . Test Set

The training and validation set are sufficient to train and evaluate neural network models on machine learning metrics (Eqs (2.4), (2.5), and (2.6). To properly evaluate neural network models on electrical engineering performance metrics, we generate five classical benchmark trajectories. These are divided into two categories:

Quasi-Static Benchmarks At constant torque, reference speed goes from 70 to -70Hz in 50 seconds. Two constant torques are tested: no-load and 50% of the nominal load. We name these benchmarks, **Quasi-Static1** and **Quasi-Static2**, respectively. In the case of the 50% nominal load torque, the torque has already reached the steady-state before the start of the benchmark.

Dynamic Benchmarks We generate three dynamic benchmarks to evaluate our neural network models:

(a) **Dynamic-Speed1:** Reference speed goes from 0 to 50Hz in 1 second at no load.

(b) **Dynamic-Speed2:** Reference speed goes from 0 to 50Hz in 1 second at 50% of nominal load.

(c) **Dynamic-Speed3:** Reference speed goes from 50 to -50Hz in 1 second at 50% of nominal load.

(d) **Dynamic-Torque:** Load torque goes from 0 to 100% of nominal torque in 4ms with a constant 25Hz reference speed.

2.5.4 . Real Motor Dataset

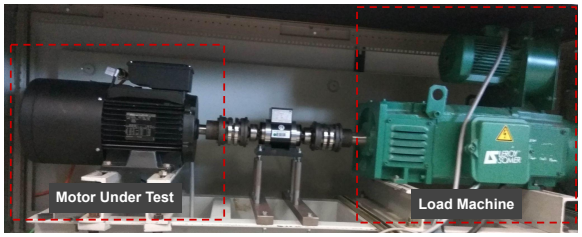


Figure 2.15: Experimental setup

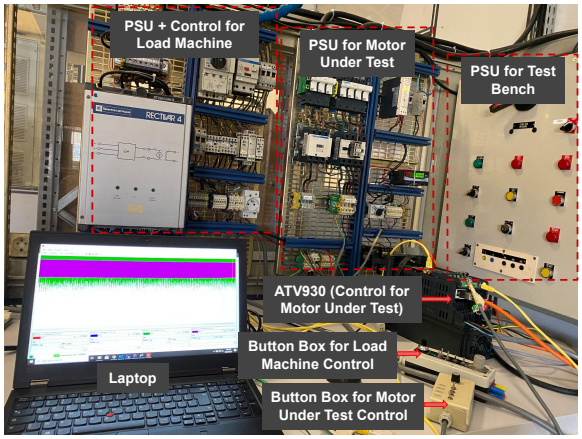


Figure 2.16: Test bench setup containing the data acquisition

Figure 2.15 shows a 4kW induction motor under test, and a direct current motor as load machine part of our experimental setup. Figure 2.16 shows our test bench setup which consists of the power supply units (PSU) for the test bench, the motor under test, the load machine, and an ATV930 VSD to control the induction motor. In addition, button boxes allow us to give run orders and manually set speed / torque references, both for the motor under test and the load machine.

In addition to the simulated data, 10 experiments were performed on 1.5kW and 4kW motors (50Hz nominal speed) shown in Figure 2.15 to collect real data. These experiments have different types of trajectories: the constant speed with torque variations in [-120, 120]% of the nominal torque, speed variations in [-70, 70] Hz at no load, torque steps under constant speed, and, in some cases, both speed and torque vary. All these types of trajectories have been considered to cover the majority of the use cases that may arise in the real world. This provides different kinds of dynamic behavior which makes denoising and speed-torque estimation quite challenging. Due to the non-availability of non-noisy trajectory in real data and the non-triviality of obtaining them, we only use three dynamic and one quasi-static benchmark to evaluate DiagBiRNN on real motor data:

(a) **RDynamic-Speed1:** Reference speed goes from 0 to 50Hz in 1 second at no load.

(b) **RDynamic-Speed2**: Reference speed goes through multiple inversions in both directions.

(c) **RDynamic-Torque:** Load torque goes from 0 to 100% of nominal torque in one time step at a constant 50Hz reference speed.

(d) **RQuasi-Static:** At no load, reference speed goes from 80 to -80Hz in 50 seconds.

2.5.5 . Overcoming Bias in Dataset

During our experiments, we found that all our models were biased toward long-duration ramps present in the training data. This is because when reference trajectories were generated, ramp durations were originally sampled from a uniform distribution. Expected motor speed and torque trajectories depend a lot on what ramp has been demanded. For short duration ramps, the speed and torque trajectories have high variance with slight change in the ramp duration. This is not the case for long duration ramps. Therefore we need to sample ramps with a non-uniform distribution focusing more on short duration ramps. To overcome this bias, generating data with ramps drawn from an exponential distribution plays a prominent role. This guarantees that our model can see more frequently short-duration ramps during training.

Figure 2.17 compares models trained on the two versions of data: Data V1, where we have ramps generated from a uniform distribution and Data V2, where we use exponential distribution (the number of samples decreases as ramp duration decrease). It can be seen that the model trained on Data V2 can accurately predict ramp overshoots for all values of ramp duration. In contrast, the model trained on Data V1 fails for short-duration ramps (0.0s to 1.5s). A maximum error of 23% can be observed for the ramp of duration 0.1s.

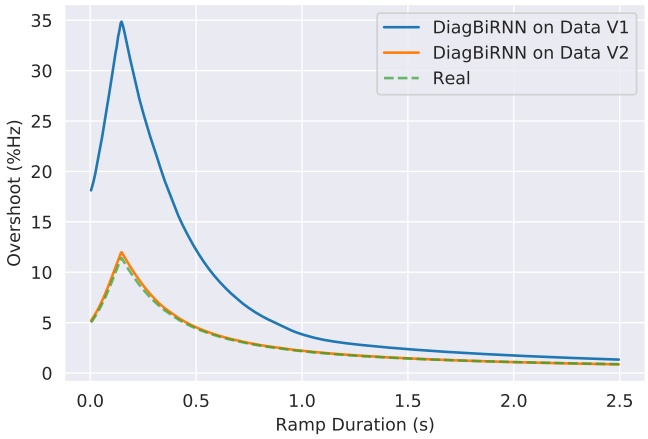


Figure 2.17: Overshoot vs. ramp for DiagBiRNN network trained on two versions of data.

2.6 . Input-Output Relationship Modeling Experiments

In this section, we try to model the relationship between motor quantities like currents, voltages, torque, and speed. To achieve this, we utilize several existing neural network architectures and arrive at a network that serves well for our use case.

2.6.1 . Derive Currents and Torque from Voltages and Speed

We vary our architecture by trying different input lengths, the number of layers, and RNN/LSTM hidden vector lengths. We try the following input lengths (5, 10, 15, 20, 25, 50, 100, 200) and find out that an input length greater than 100 is better at capturing the motor operation dynamics. Depending on the architecture, different input and output structures are required. Feed-forward networks take a flattened vector and predict a single output which is the middle value of the output signal. RNNs and LSTMs take channelized input and predict the output of the same length. CNNs take channelized input and predict the middle value of the output signal.

In encoder-decoder variations where an RNN is used, the hidden vector size is the same as the number of features in the input vector. In the encoder-decoder network, the input and output lengths are the same. We train all our models using the proposed TV-weighted mean square loss function in Eq. (2.3). To find the best architecture, we use the validation set of the simulated data. Then we re-train the best model on the training set of the raw data (fine-tuning) and test it on the raw data test set. We also train the best-performing model using mean square loss to compare it with the proposed loss function.

We provide results for the benchmark models in Table 2.2. The window size column shows the input length on which the best result was obtained. The hidden vector size for both RNN and LSTM is 32. The number of parameters is also

Model	Window Size	Parameters	MAE $(i_d + i_q + \tau_{em})/3$	SMAPE $(i_d + i_q + \tau_{em})/3$	R^2 $(i_d + i_q + \tau_{em})/3$
Shallow Feed-Forward	25	751K	**77.76**	9.79%	-0.59
Deep Feed-Forward	20	1118K	78.91	8.53%	-0.39
Shallow RNN	100	9K	77.97	8.5%	-0.3
Deep RNN	150	12K	78.26	7.76%	-0.35
Shallow LSTM	50	13K	79.39	6.41%	-0.26
Deep LSTM	100	21K	79.58	6.29%	**-0.11**
Shallow CNN	100	518K	79.51	6.22%	-0.13
Deep CNN	100	650K	79.69	**6.13%**	-0.14

Table 2.2: Results for the benchmark networks on the simulated validation set.

reported for all the models. For each of them, we report MAE, SMAPE, and R^2 values. For every metric, we report the average of current i_d, current i_q, and electromagnetic torque τ_{em}. All results were obtained on the validation set of the simulated data. Among benchmark models, we observe that MAE values are very close for all the models. But when we compare SMAPE and R^2 values, deep CNN and deep LSTM come out to be the best. In our experiments, we observe that the models perform better when the input length is 100 or more. For all the models, the performance gap between shallow and deep variants is small. This means that a deeper network provides little advantage in learning the nonlinear dynamics of electrical motors.

Model	Window Size	Parameters	MAE $(i_d + i_q + \tau_{em})/3$	SMAPE $(i_d + i_q + \tau_{em})/3$	R^2 $(i_d + i_q + \tau_{em})/3$
Shallow	100	309K	80.63	5.02%	0.08
Deep	100	1096K	81.21	4.57%	0.29
Skip	100	364K	28.96	3.71%	0.42
RNN-Skip	100	638K	28.18	3.42%	0.43
BiRNN-Skip	100	967K	27.96	3.31%	0.41
DiagBiRNN	100	618K	**26.88**	**1.09**%	**0.95**

Table 2.3: Results for the proposed networks on the simulated validation set.

Based on the results obtained from the benchmark methods, we set the input size to 100 for all our proposed model variants. Table 2.3 shows the results of the proposed model variants trained and validated on the simulated data. The first and second rows show the results of the shallow and deep variants of the encoder-decoder architecture. We see that MAE is still comparable to the benchmark models, but SMAPE and R^2 values improve. The third row shows the result of the model where skip connections have been added between encoder-decoder. MAE gets better in this case. The fourth and fifth rows correspond to recurrent skip connections with unidirectional and bidirectional recurrence, respectively. Having recurrence in skip connections improves MAE and SMAPE values but comes at the cost of an increased number of parameters.

It can be seen that bidirectionality has a positive effect on MAE and SMAPE. The last row shows the best version of our encoder-decoder model, where we replace RNNs in skip connections with diagonalized RNNs. This model outperforms all the methods and has fewer parameters when compared to other RNN variants.

2.6.2 . Ablation Study

An ablation study is required here to analyze different contributions made in this chapter. The first ablation is on why using TV-weighted MSE is better than MSE. The second ablation is on why fine-tuning is required to get good performance on the real dataset. It also explains how the domain shift from simulated to real data affects the model performance.

2.6.2.1 . Ablation on TV-Weighted MSE

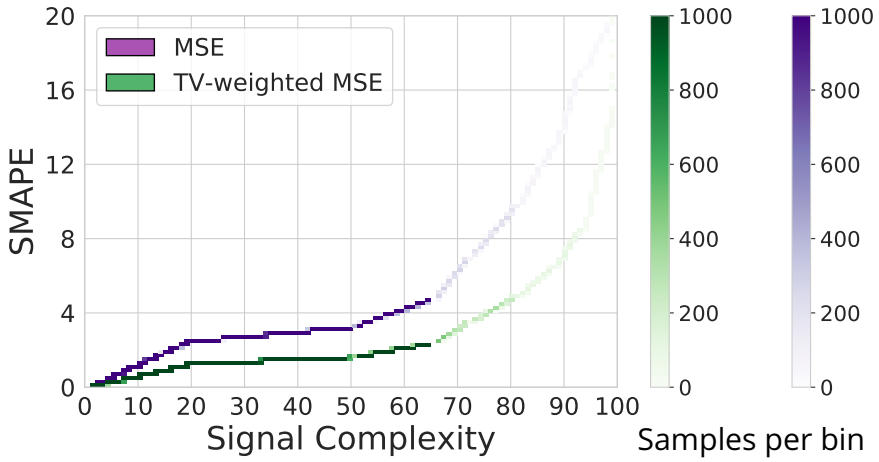


Figure 2.18: Comparison between the proposed TV-weighted MSE loss and MSE loss used to train the proposed network in case of current i_d.

Figure 2.18 shows the SMAPE vs signal complexity (SC) computed on the current i_d in the test set. SMAPE has been defined in Equation 2.5 and SC has been defined in Eq. (2.7). The SMAPE vs SC plots are 2D histograms where the color intensity of each box represents the number of samples that are in that bin. The comparison is drawn between the model trained with MSE loss and TV-weighted MSE loss. It can be observed that as the signal complexity value increases, the SMAPE of both models also increases. But the SMAPE of the TV-weighted MSE model predicted current i_d is consistently less than MSE trained model. As the signal complexity increases the difference in SMAPE between the two models also increases suggesting that TV-weighted MSE is not affected by high variations in the output signal.

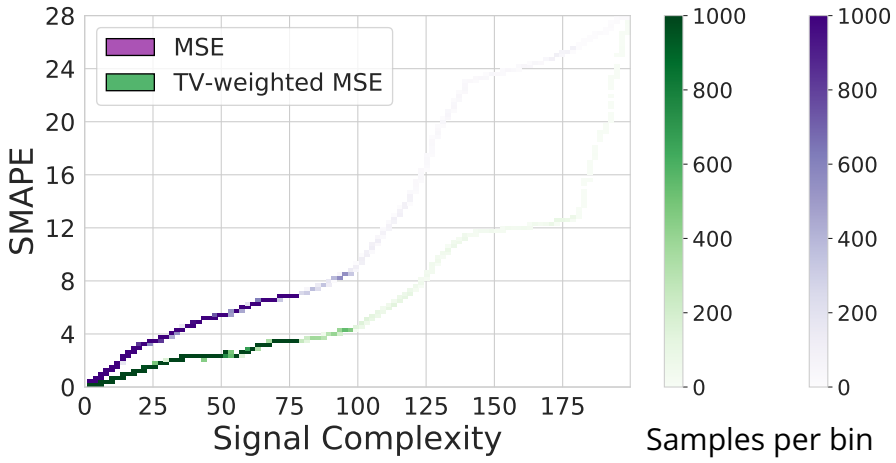


Figure 2.19: Comparison between the proposed TV-weighted MSE loss and MSE loss used to train the proposed network in case of current i_q.

A similar trend in SMAPE vs signal complexity is observed in the predicted current i_q from models trained with MSE and TV-weighted MSE. This can be seen in Figure 2.19. In case of the estimated torque τ_{em}, the same trend is observed as shown in the Figure 2.20. Average signal complexity of the current i_q is higher than the torque τ_{em} and the current i_d. We see that the signal parts with higher signal complexity are not frequent. Our model trained with MSE loss can predict more accurately parts of the signal with small signal complexity.

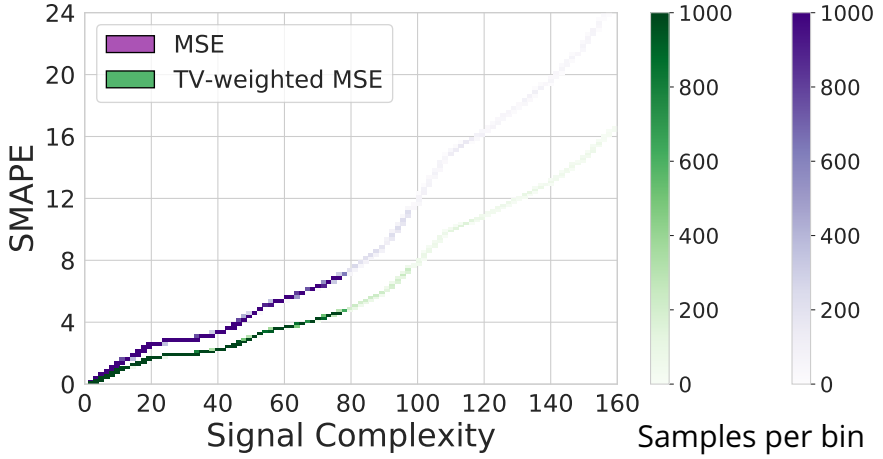


Figure 2.20: Comparison between the proposed TV-weighted MSE loss and MSE loss used to train the proposed network in case of torque τ_{em}.

Table 2.4 shows the results obtained by the proposed model when MSE loss and TV-weighted MSE loss were used in training. All three metrics for all three quantities improve when the proposed TV-weighted MSE loss is used in training

Quantity	MSE Loss			TV-weighted MSE Loss		
	MAE	SMAPE	R^2	MAE	SMAPE	R^2
i_d **(A)**	28.13	0.97%	0.65	**27.91**	**0.46%**	**0.92**
i_q **(A)**	26.89	2.39%	0.95	**26.52**	**1.90%**	**0.96**
τ_{em} **(Nm)**	26.23	1.58%	0.92	**26.19**	**0.92%**	**0.96**

Table 2.4: Performance of DiagBiRNN on the simulated validation set when trained using MSE loss and the proposed TV-weighted MSE loss.

the DiagBiRNN Encoder-Decoder network. We observe that the model trained using TV-weighted MSE loss overcomes this issue.

2.6.2.2 . Ablation on Fine-Tuning

Quantity	Simulated Model			Fine-tuned Model		
	MAE	SMAPE	R^2	MAE	SMAPE	R^2
i_d **(A)**	39.83	4.10%	0.38	**35.31**	**2.64%**	**0.56**
i_q **(A)**	47.38	6.37%	0.41	**42.94**	**5.28%**	**0.49**
τ_{em} **(Nm)**	38.56	3.81%	0.49	**32.38**	**2.38%**	**0.60**

Table 2.5: Results of DiagBiRNN on the real test set. One trained on the simulated data and the other fine-tuned on the real train set.

Table 2.5 shows the results of the simulated model and model fine-tuned on the raw data training set when tested on the raw data test set. It can be seen that the proposed model can learn the temporal dynamics of each of the quantities very well just from the simulated data. When the model is fine-tuned on the sensor data, it seems to be able to learn about the noise associated with the sensors and yields better results.

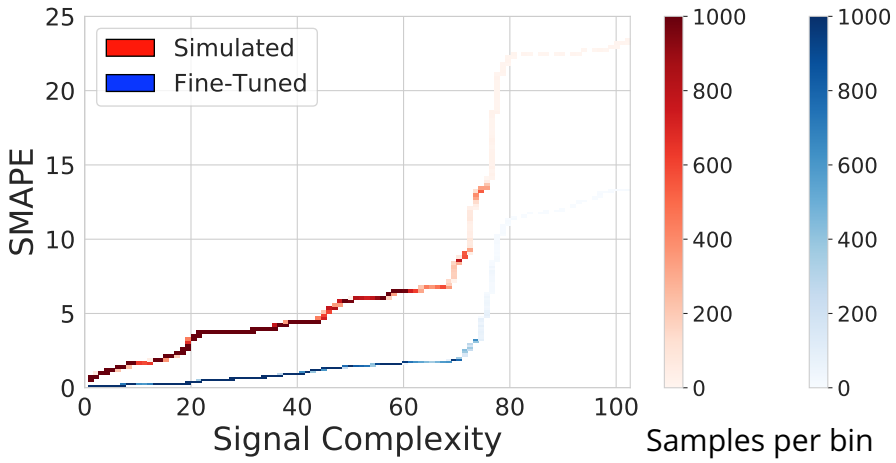


Figure 2.21: Comparison of simulated and fine-tuned model using SMAPE vs Signal Complexity graph for current i_d.

Figure 2.21 shows the signal complexity vs SMAPE computed on the current i_d in the test set. The comparison is drawn between models trained on simulated data and fine-tuned on real data. It can be observed that as the signal complexity value increases, the SMAPE of both models also increases. But the SMAPE of current i_d predicted from the model trained on simulated data is consistently larger than the one for the fine-tuned model. This is because the fine-tuned model learns about the variations present in the signal due to noise.

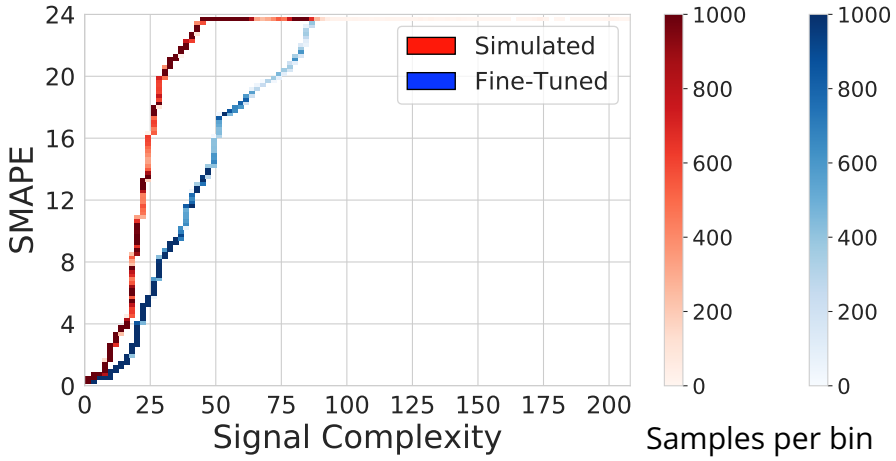


Figure 2.22: Comparison of simulated and fine-tuned model using SMAPE vs Signal Complexity graph for current i_q.

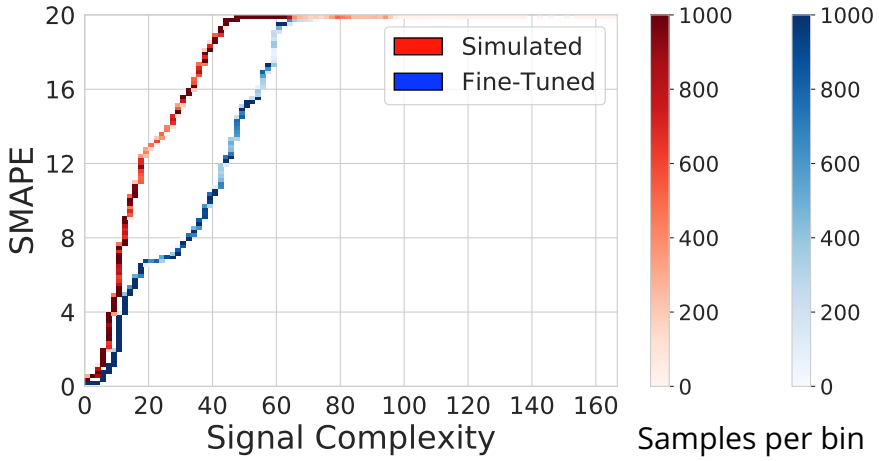


Figure 2.23: Comparison of simulated and fine-tuned model using SMAPE vs Signal Complexity graph for torque τ_{em}.

Figures 2.22 and 2.23 shows the SMAPE vs signal complexity graph for current i_q and torque τ_{em} predicted using models trained on simulated data and fine-tuned on real data. Both the predictions have very high SMAPE for slightly bigger signal

complexity which means that both predictions suffer a lot from the presence of noise, although this is somewhat taken care of by fine-tuning.

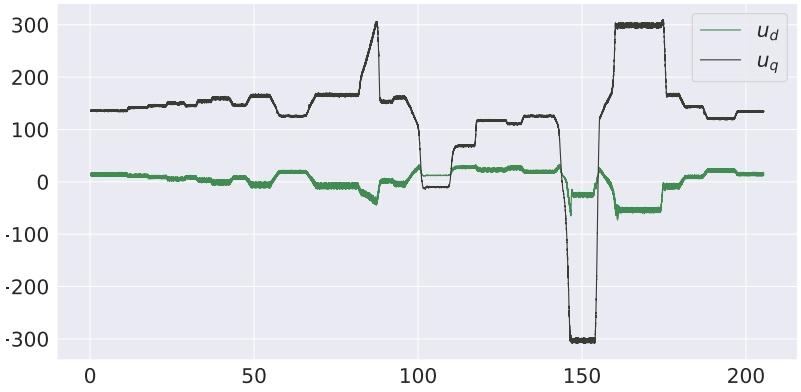


Figure 2.24: Input voltages u_d, u_q of one of the experiments from test set.

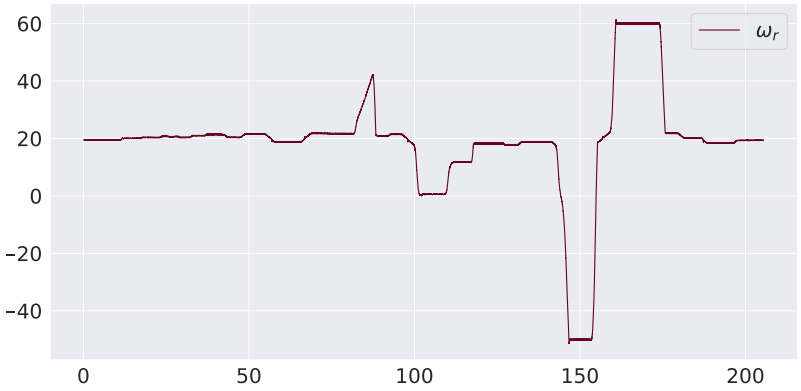


Figure 2.25: Input speed ω_r of one of the experiments from test set.

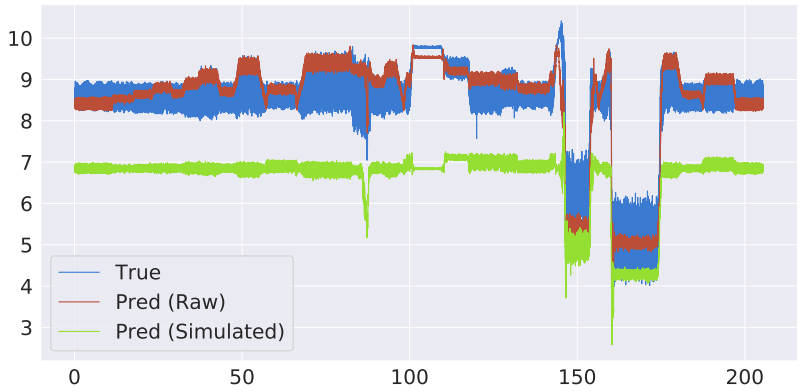


Figure 2.26: Predicted result for current i_d of one of the experiments from test set.

We plot the prediction results for a single real motor operation from the test set. Figure 2.24 shows the voltages u_d and u_q and Figure 2.25 shows the root speed ω_r for the real motor operation. Figure 2.26 shows the predicted trajectory of current i_d for the input voltages and rotor speed. It can be seen that the model trained on the simulated data has some offset in its prediction whereas the model fine-tuned on the sensor data is much closer to the ground truth, even if it is still not perfect.

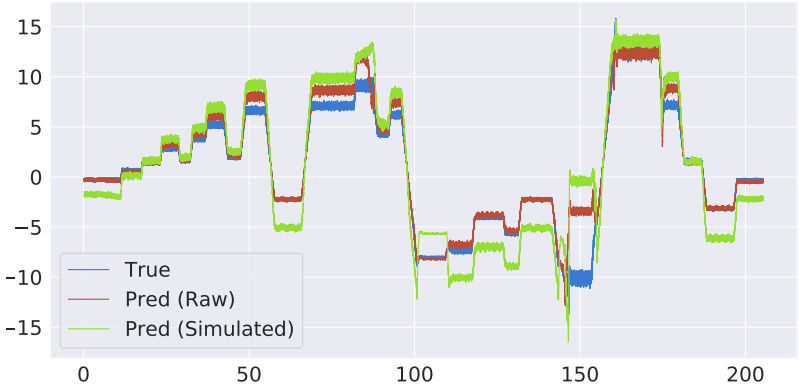


Figure 2.27: Predicted result for current i_q of one of the experiments from test set.

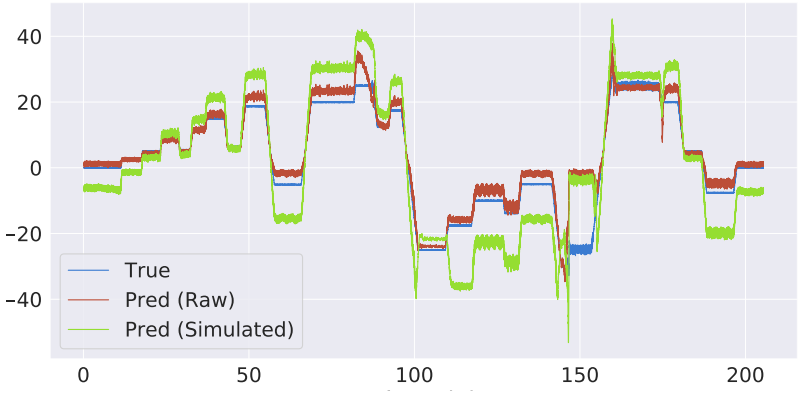


Figure 2.28: Predicted result for torque τ_{em} of one of the experiments from test set.

Figure 2.27 and 2.28 shows the predicted trajectory of current i_q and torque τ_{em}, respectively. These predictions are for the input voltages and rotor speed shown in Figures 2.24 and 2.25. The offset between the prediction from the model trained on simulated data and the ground truth is significantly larger than the difference between prediction results from the model fine-tuned on real data and ground truth.

2.7 . Speed-Torque Estimator Experiments

We utilize our proposed network for a more industry specific practical application of speed-torque estimation from currents and voltages. We report machine learning metrics like MAE, SMAPE, and R^2 to analyze how the models are performing from a global point of view. At the same time for deeper analysis, we use electrical engineering performance metrics and trajectory plots to understand the behavior of different networks at every time step of a motor operation.

2.7.1 . Machine Learning Benchmarks

Model	Speed (ω_r)		Torque (τ_{em})	
	MAE	**SMAPE**	**MAE**	**SMAPE**
FCN	0.79	21.77%	0.57	48.66%
LSTM	0.11	18.76%	0.21	43.01%
CNN	0.06	19.14%	0.09	38.91%
Vanilla	0.05	18.94%	0.10	39.91%
Skip	0.08	19.08%	0.12	43.23%
RNN	0.06	19.31%	0.08	41.81%
BiRNN	0.05	**18.67%**	0.09	42.82%
DiagBiRNN	**0.03**	18.76%	**0.04**	**38.46%**
R^2 is 0.99 for all the networks for both quantities.				

Table 2.6: ML metrics for all speed-torque estimator networks on the benchmark set.

ML metrics for the results obtained on the quasi-static and dynamic benchmarks are reported in Table 2.6. Smaller MAE and SMAPE values are desired for a good prediction model and R^2 closer to 1 is considered as perfect in terms of prediction. We observe that ML metrics do not allow a clear comparison between different networks. We can see that any evaluation based on SMAPE and R^2 is difficult.

2.7.2 . Electrical Engineering Benchmarks

ML metrics provide a global performance index on the benchmark set. Evaluating individual benchmarks using ML metrics does not yield a meaningful analysis. ML metrics provide good results because there are many static windows where signals remain almost constant. These static durations are easy to predict compared to fewer occurring dynamic windows. EE metrics focus on these dynamic parts of the signal. For readability, we only plot predictions of the worst performing network (FCN), the best performing standard neural network (CNN), and the overall best performing network (DiagBiRNN) along with reference trajectory and real output (given by Simulink) for each of the dynamic benchmark.

Model	$t_{2\%}$ (ms)	$t_{95\%}$ (ms)	E_{fol} (Hz)	$D\%$ (%)	E_{ss} (Hz)	$\Delta\tau_{max}$ ($\%\tau_{nom}$)
Real	**48**	**960**	**-0.02**	**2.16**	**0.00**	**32.69**
FCN	8	988	0.56	0.94	1.20	34.18
LSTM	44	933	-0.04	3.30	-0.13	33.49
CNN	40	**964**	-0.04	2.96	-0.04	32.57
Vanilla	44	968	-0.08	2.62	0.02	32.37
Skip	**48**	952	0.12	3.04	**0.01**	32.46
RNN	**48**	952	-0.04	2.28	0.02	32.82
BiRNN	44	944	-0.11	2.29	**0.01**	**32.67**
DiagBiRNN	44	952	**-0.01**	**2.21**	0.03	**32.67**

Table 2.7: EE performance metrics obtained on Dynamic-Speed1 benchmark.

Table 2.7 shows electrical engineering performance metrics of different network predictions on the Dynamic-Speed1 benchmark. Acceptable values (shown in green) are less than 0.1Hz / 10ms / 0.2 percent point from real values and unsatisfactory values (shown in red) are more than 0.25Hz / 25ms / 0.5 percent point from real values. 2% response time ($t_{2\%}$) is very accurate for Skip and RNN variants of encoder-decoder network. 95% response time ($t_{95\%}$) is the best for CNN. DiagBiRNN achieves the best following error (E_{fol}), overshoot ($D\%$), and maximum acceleration torque ($\%\tau_{nom}$). Skip and BiRNN variants of encoder-decoder achieve the best steady-state error (E_{ss}).

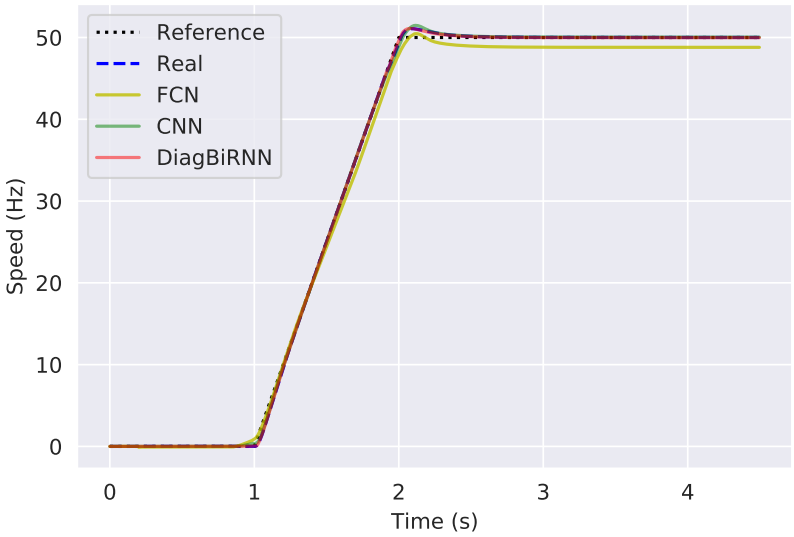


Figure 2.29: Results on Dynamic-Speed1 benchmark: speed trajectory.

Figure 2.29 shows plots for rotor speed predicted from different networks on Dynamic-Speed1 benchmark. The plot shows the complete benchmark trajectory of 5 seconds which contains the start-of-ramp, ramp, end-of-ramp, overshoot, and steady-state parts. FCN predicted trajectory after the start of ramp deviates from

the ground truth trajectory. The next two figures show what happens during the start and end of the ramp, respectively.

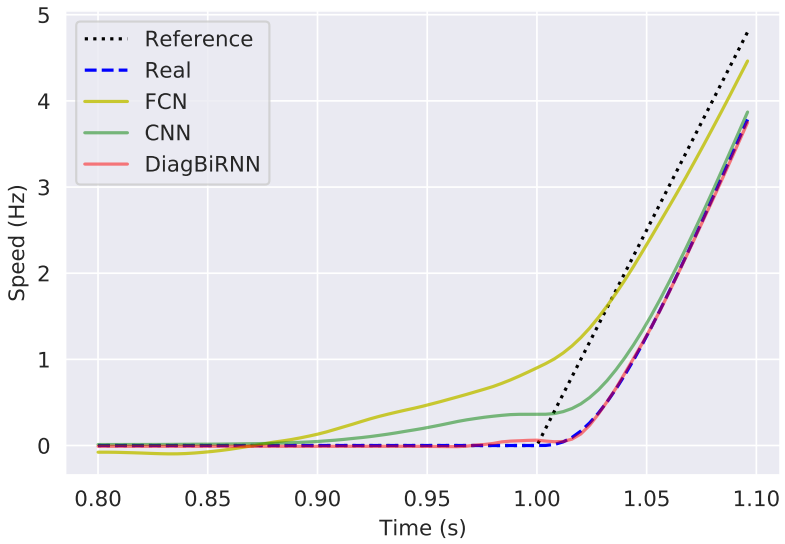


Figure 2.30: Results on Dynamic-Speed1 benchmark: start of the speed ramp.

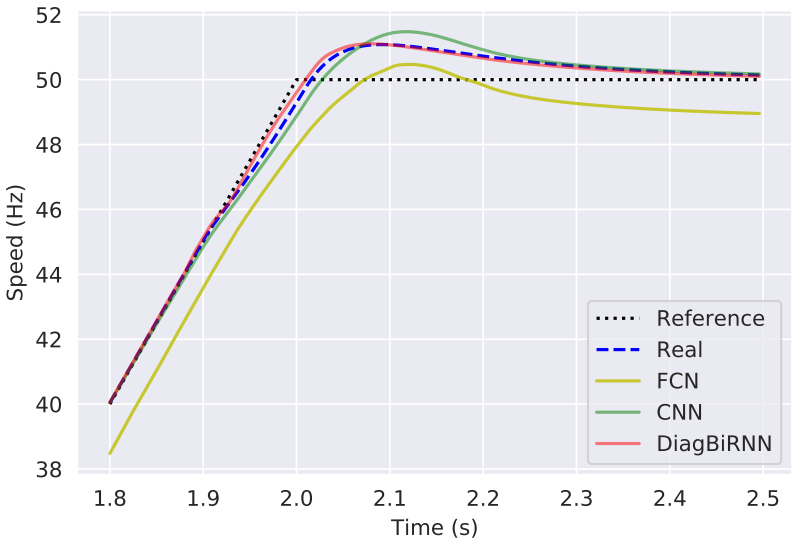


Figure 2.31: Results on Dynamic-Speed1 benchmark: end of the speed ramp.

Figures 2.30 and 2.31 show plots for start and end of ramp for different network speed predictions on Dynamic-Speed1 benchmark. DiagBiRNN predictions are very close to the ground truth trajectory. Among standard neural networks, FCN performs worst and CNN performs slightly better.

Figure 2.32 shows the plot for torque predicted from FNN, CNN, and DiagBiRNN on Dynamic-Speed1 benchmark. In this case, FCN has a large offset

during the acceleration period whereas CNN and DiagBiRNN are close to the ground truth torque trajectory.

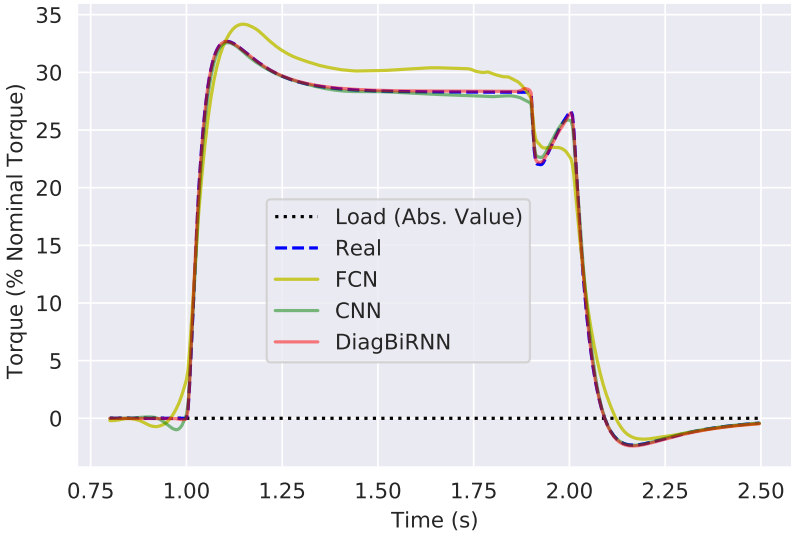


Figure 2.32: Results on Dynamic-Speed1 benchmark: torque trajectory.

Table 2.8 shows electrical engineering performance metrics of different network predictions on the Dynamic-Speed2 benchmark. 2% response time ($t_{2\%}$) is correct for Skip and DiagBiRNN variants of encoder-decoder network. 95% response time ($t_{95\%}$) is the best for CNN. DiagBiRNN achieves best following error (E_{fol}) and overshoot ($D\%$). BiRNN variant of encoder-decoder achieve the best steady-state error (E_{ss}) and maximum torque acceleration ($\%\tau_{nom}$).

Model	$t_{2\%}$ (ms)	$t_{95\%}$ (ms)	E_{fol} (Hz)	$D\%$ (%)	E_{ss} (Hz)	$\Delta\tau_{max}$ ($\%\tau_{nom}$)
Real	**48**	**968**	**-0.02**	**1.97**	**-0.06**	**29.40**
FCN	8	952	0.60	3.45	1.49	30.18
RNN	44	1024	-0.07	6.60	-0.10	29.84
CNN	40	**968**	-0.06	3.90	-0.10	29.25
Vanilla	44	980	-0.07	3.01	-0.13	29.19
Skip	**48**	940	0.02	4.02	-0.10	29.15
RNN	44	948	-0.06	2.69	-0.15	29.48
BiRNN	44	944	-0.09	2.42	**-0.06**	**29.37**
DiagBiRNN	**48**	948	**-0.03**	**2.04**	-0.08	29.31

Table 2.8: EE performance metrics obtained on Dynamic-Speed2 benchmark.

Figure 2.33 shows plots for rotor speed predicted from different networks on the Dynamic-Speed2 benchmark. The plot shows the part of the trajectory from the 5th second to the 8th second. First five seconds were used to bring the rotor speed to 5Hz. The plot shows from the start-of-ramp till steady-state, including

ramp, end-of-ramp, and overshoot. FCN predicted trajectory after the end of the ramp deviates from the ground truth trajectory and has a large steady-state error around the 7.5 second. The next two figures show what happens during the start and end of the ramps, respectively.

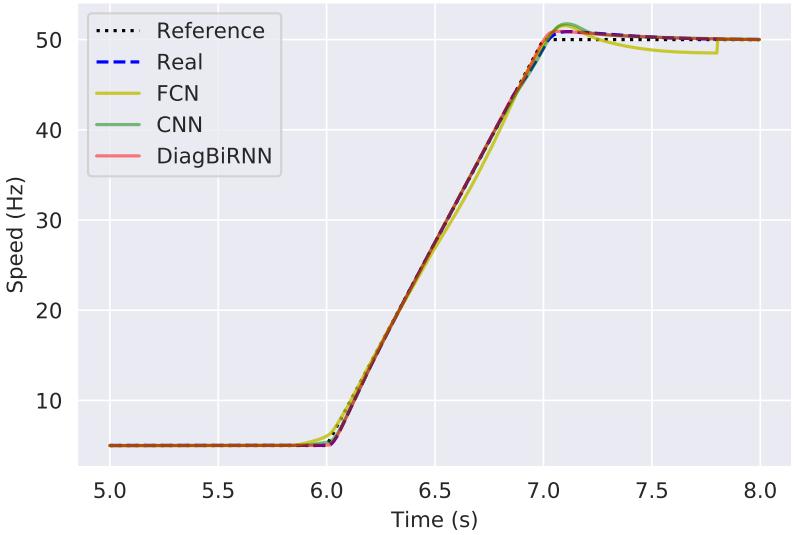


Figure 2.33: Results on Dynamic-Speed2 benchmark: speed trajectory.

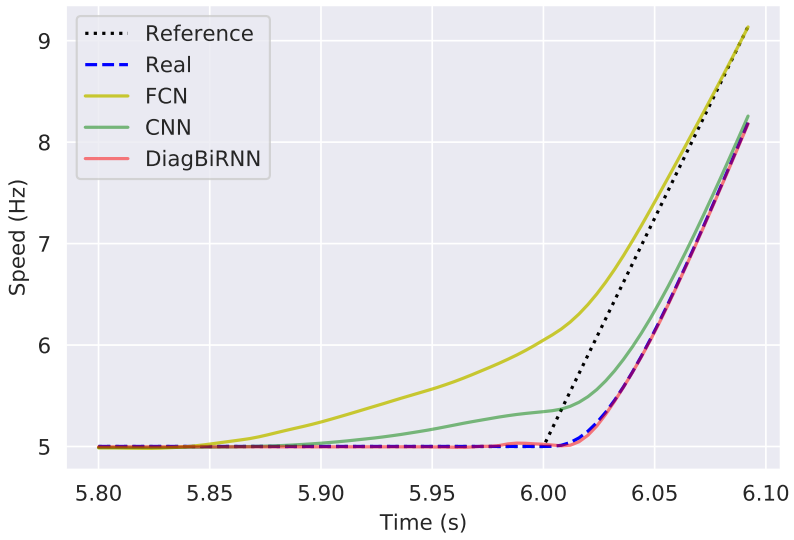


Figure 2.34: Results on Dynamic-Speed2 benchmark: start of the speed ramp.

Figures 2.34 and 2.35 show plots for start and end of ramps for different network speed predictions on Dynamic-Speed2 benchmark. It can be seen that DiagBiRNN predictions are very close to the ground truth trajectory. At the start of ramp both CNN and FCN give wrong results, the ramp cannot start before it was demanded,

as it is something physically impossible. After the ramp ends, both CNN and FCN have overshoots significantly larger than ground truth and DiagBiRNN.

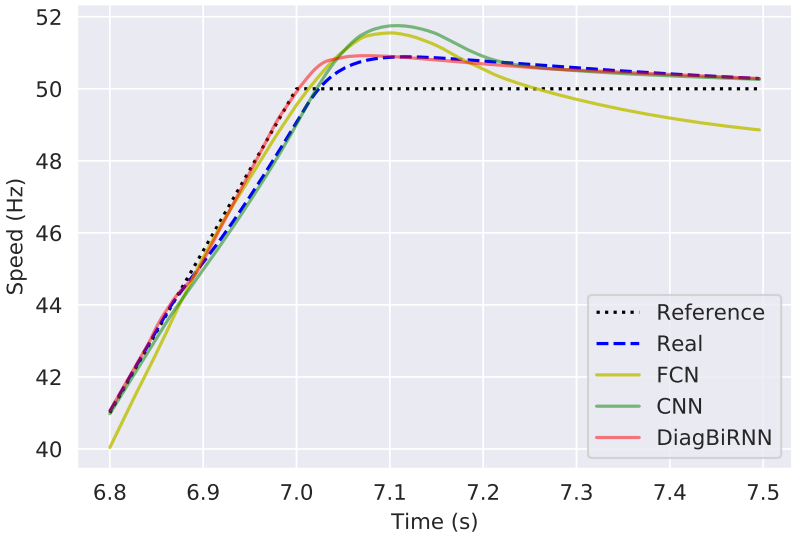


Figure 2.35: Results on Dynamic-Speed2 benchmark: end of the speed ramp.

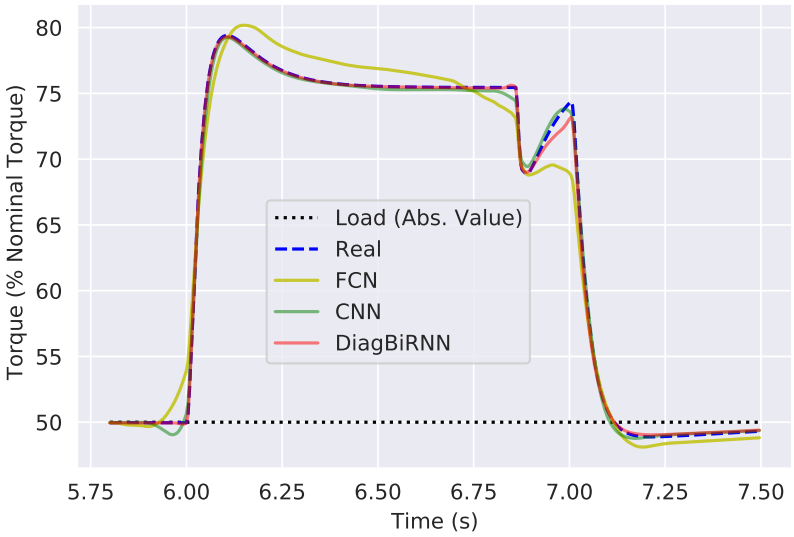


Figure 2.36: Results on Dynamic-Speed2 benchmark: torque trajectory.

Figure 2.36 shows the plot for torque predicted from FNN, CNN, and DiagBiRNN on Dynamic-Speed2 benchmark. In this case, FCN has a large offset around 6 seconds, 6.1 to 7 seconds and after 7.2 seconds. CNN and DiagBiRNN are close to the ground truth torque trajectory.

Table 2.9 shows electrical engineering performance metrics of different network predictions on the Dynamic-Speed3 benchmark. DiagBiRNN has precise

Model	$t_{2\%}$ (ms)	$t_{95\%}$ (ms)	E_{fol} (Hz)	$D\%$ (%)	E_{ss} (Hz)	$\Delta\tau_{max}$ ($\%\tau_{nom}$)
Real	**52**	**956**	**0.10**	**2.00**	**0.00**	**72.33**
FCN	-144	944	1.06	4.03	0.37	69.35
LSTM	32	1052	**0.10**	7.46	-0.33	73.30
CNN	44	960	0.42	3.21	-0.08	**72.30**
Vanilla	44	**956**	0.34	3.55	-0.11	72.29
Skip	48	1036	0.45	5.07	-0.16	71.28
RNN	48	936	0.26	3.89	**-0.06**	72.25
BiRNN	48	940	0.02	3.28	-0.13	72.45
DiagBiRNN	**52**	948	0.15	**2.39**	-0.12	72.15

Table 2.9: EE performance metrics obtained by different models on Dynamic-Speed3 benchmark.

2% response time ($t_{2\%}$). Vanilla encoder-decoder has precise 95% response time ($t_{95\%}$). LSTM has precise following error (E_{fol}). DiagBiRNN is closer to the real trajectory overshoot ($D\%$). RNN encoder-decoder achieves the closest steady-state error (E_{ss}) to the real trajectory. CNN has 0.03 error in maximum torque acceleration ($\%\tau_{nom}$) when compared to the real trajectory.

Figure 2.37 shows plots for rotor speed predicted from different networks on the Dynamic-Speed3 benchmark. The plot shows the benchmark trajectory of 3 seconds, starting just before the ramp and ending when the steady state has been achieved. In this benchmark, a speed inversion starts at 6th second and ends at 7th second. FCN predicted trajectory deviates from the ground truth trajectory way before the start of the ramp, and this is physically impossible. The next two figures show what happens during the start and end of the ramps, respectively.

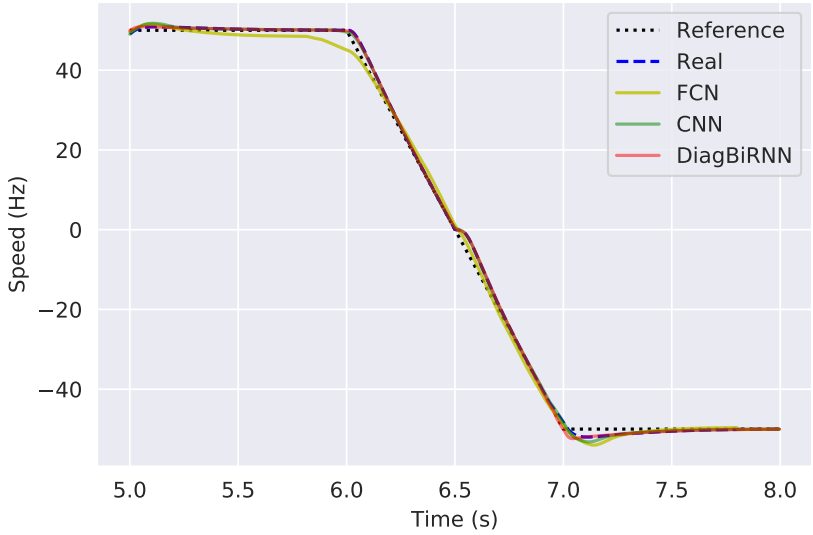


Figure 2.37: Results on Dynamic-Speed3 benchmark: speed trajectory.

Figures 2.38 and 2.39 show plots for start and end of ramps for different network speed predictions on Dynamic-Speed3 benchmark. DiagBiRNN predictions are very close to the ground truth trajectory but if looked closely it can be seen that it starts before the actual start of ramp. FCN can be outright rejected as a valid prediction. CNN also starts before actual start of ramp. When the ramp ends all predicted trajectories have close enough undershoot to the ground truth trajectory. This benchmark shows it is a bit hard to predict speed inversion for the learned networks.

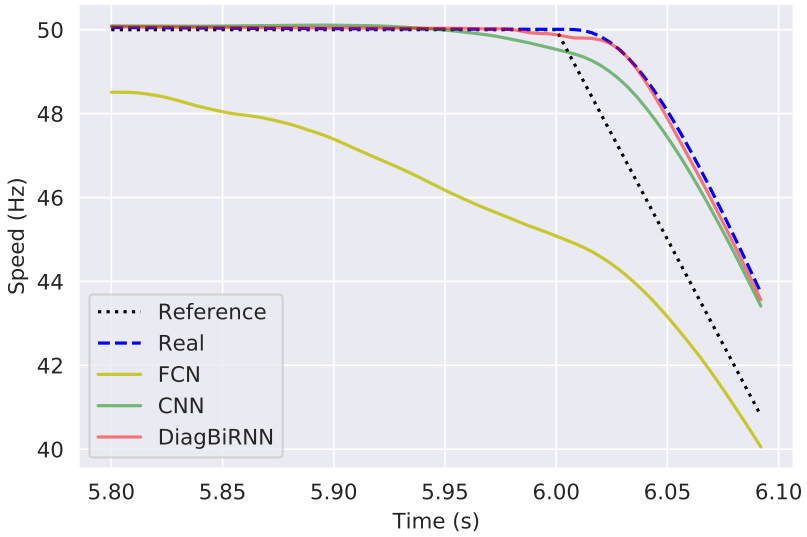


Figure 2.38: Results on Dynamic-Speed3 benchmark: start of the speed ramp.

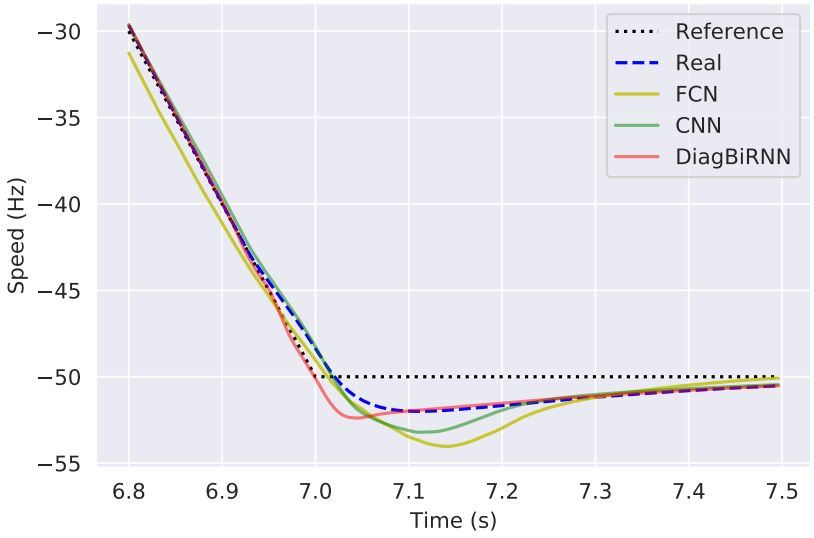


Figure 2.39: Results on Dynamic-Speed3 benchmark: end of the speed ramp.

Figure 2.40 shows the plot for torque predicted from FNN, CNN, and

DiagBiRNN on Dynamic-Speed3 benchmark. In this case, FCN has a large offset during the acceleration period whereas CNN and DiagBiRNN are close to the ground truth torque trajectory.

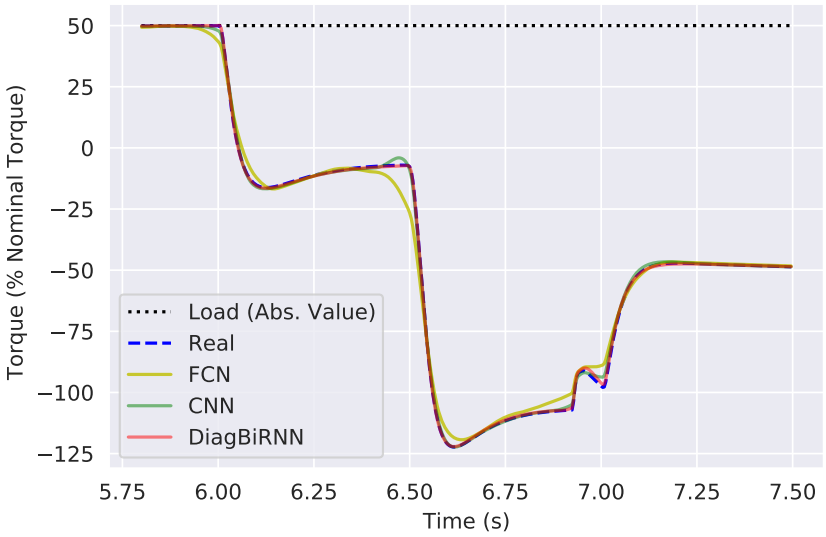


Figure 2.40: Results on Dynamic-Speed3 benchmark: torque trajectory.

Model	$t_{95\%}$ (ms)	$D\%$ (%)	E_{ss} ($\%\tau_{nom}$)	SD (Hz)
Real	**244**	**15.96**	**0.00**	**4.39**
FCN	252	11.19	-0.32	3.30
LSTM	**244**	**15.95**	-0.02	4.28
CNN	**244**	16.01	0.01	4.45
Vanilla	**244**	15.46	0.04	3.88
Skip	**244**	15.90	-0.02	**4.43**
RNN	**244**	15.87	**0.00**	4.03
BiRNN	**244**	16.01	0.01	4.23
DiagBiRNN	**244**	15.91	-0.02	4.31

Table 2.10: EE performance metrics obtained by different models on Dynamic-Torque benchmark.

Table 2.10 shows electrical engineering performance metrics of different network predictions on the Dynamic-Torque benchmark. 95% response time ($t_{95\%}$) is wrong only for FCN predicted trajectory. LSTM achieves the closest overshoot ($D\%$). RNN variant of encoder-decoder achieve the best steady-state error (E_{ss}). Skip variant of encoder-decoder is closest to ground truth trajectory in case of speed drop (SD).

Figure 2.41 shows plot for predictions and ground truth of torque of Dynamic-Torque benchmark. The benchmark starts at 2.5 seconds and ends at 4 seconds.

There is torque step at 3 seconds. It can be observed that FCN starts ramping up around 2.9 second which is physically impossible to do in a real motor. CNN has some undershoot effect just before 3 second. DiagBiRNN starts just after 3 second and follows the ground truth trajectory. Next two figures show the start of step and end of step parts.

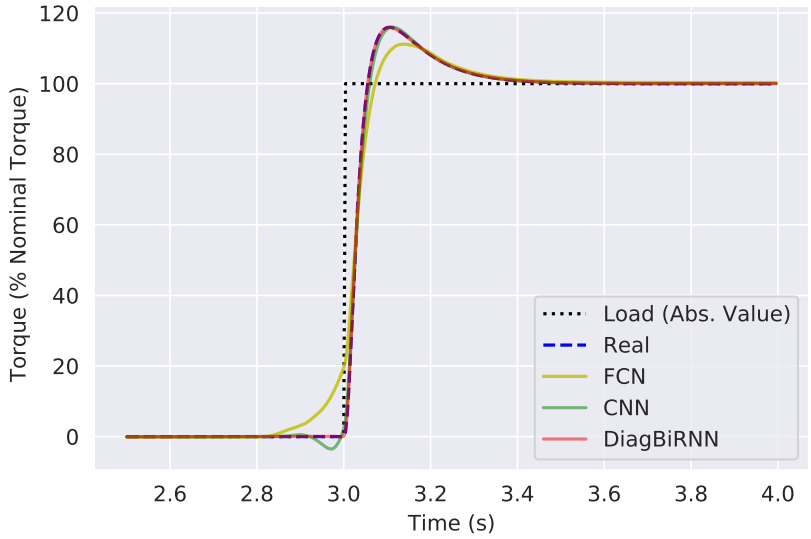


Figure 2.41: Results on Dynamic-Torque benchmark: torque trajectory.

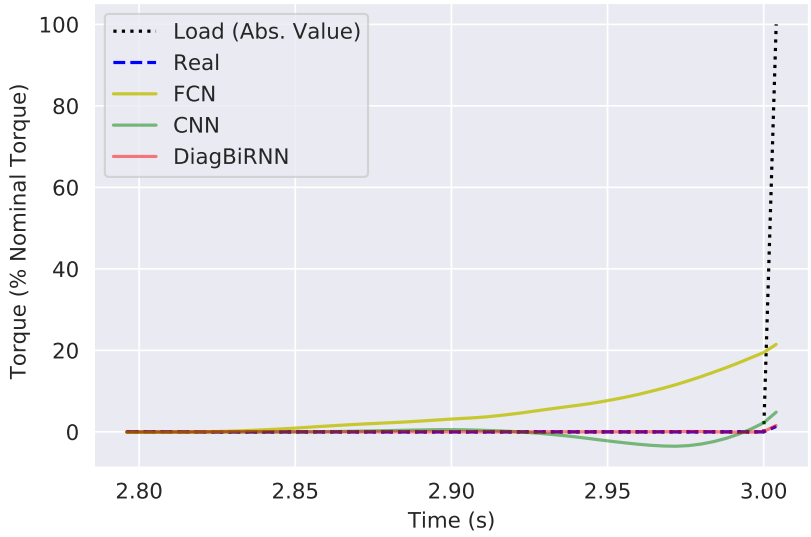


Figure 2.42: Results on Dynamic-Torque benchmark: start of torque step.

Figures 2.42 and 2.43 show start and end of torque step for Dynamic-Torque benchmark. It can be clearly seen that both FCN and CNN are predicting the start of step way before it should be. When it comes to the end of step, all

three networks are predicting correctly, CNN and DiagBiRNN are very close to the ground truth.

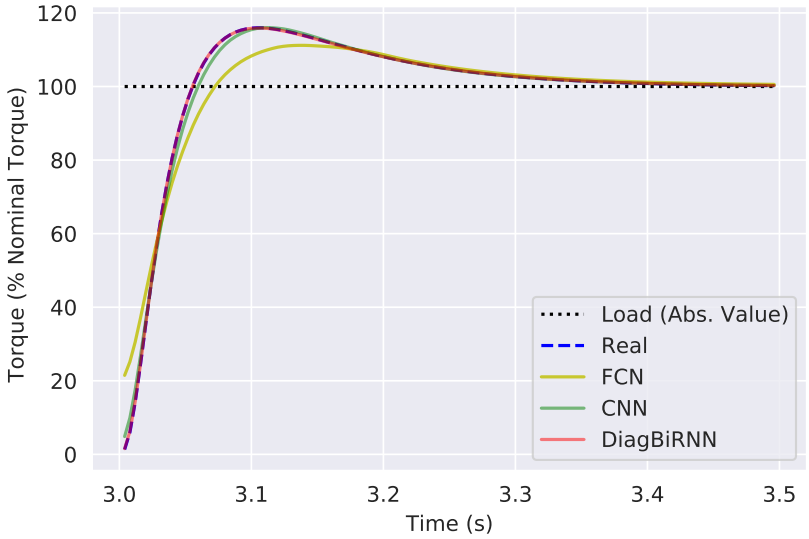


Figure 2.43: Results on Dynamic-Speed3 benchmark: end of torque step.

Figure 2.44 shows predicted speed trajectories for FCN, CNN, and DiagBiRNN. Compared with ground truth speed trajectory, DiagBiRNN performs very well, as does CNN. On the other hand, FCN predictions are absurdly wrong due to its limited capability in modeling temporal dynamics.

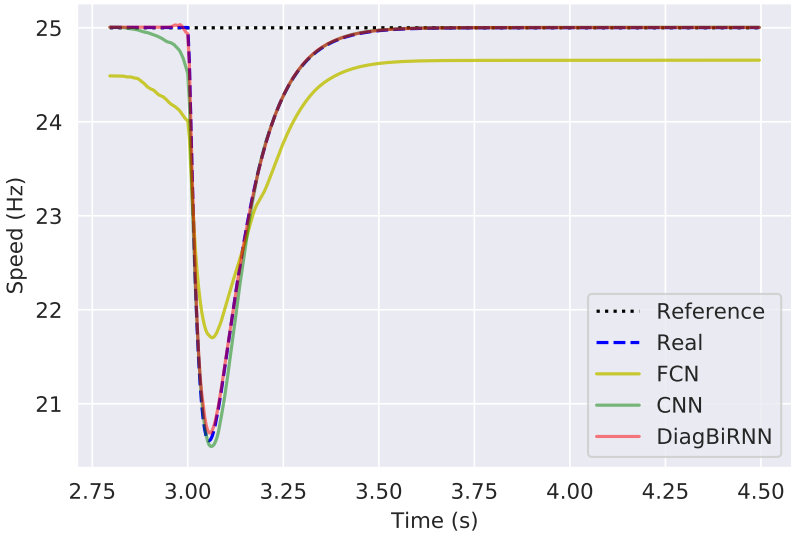


Figure 2.44: Results on Dynamic-Torque benchmark: speed trajectory.

The max absolute error for the predictions on quasi-static benchmarks is reported in Table 2.11. It can be seen that DiagBiRNN has the smallest error and

Model	FCN	LSTM	CNN
Quasi-Static1	3.66	0.992	0.261
Quasi-Static2	5.751	0.629	0.259

Model	Vanilla	Skip	RNN	BiRNN	DiagBiRNN
Quasi-Static1	0.178	0.549	0.341	0.236	**0.198**
Quasi-Static2	0.336	0.444	0.258	0.346	**0.171**

Table 2.11: Max absolute error (Hz) for static benchmarks.

therefore is again closest to the real output speed, whereas FCN leads to the largest error.

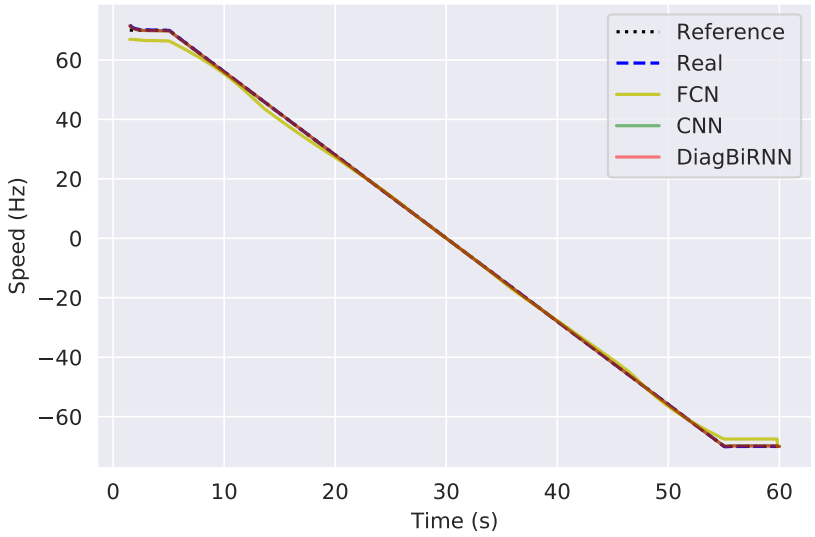


Figure 2.45: Results on Quasi-Static1 benchmark: speed trajectory.

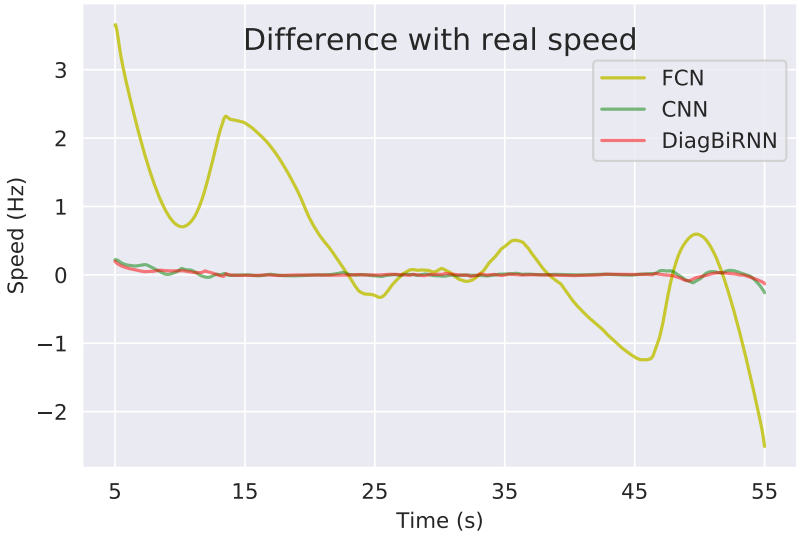


Figure 2.46: Results on Quasi-Static1 benchmark: error.

For quasi-static benchmarks, we plot the speed during the long ramp and the difference between neural network speed prediction and real output speed. Figures 2.45 and 2.46 show plots for predicted speed trajectories and their error with ground truth on Quasi-Static1 benchmark, respectively.

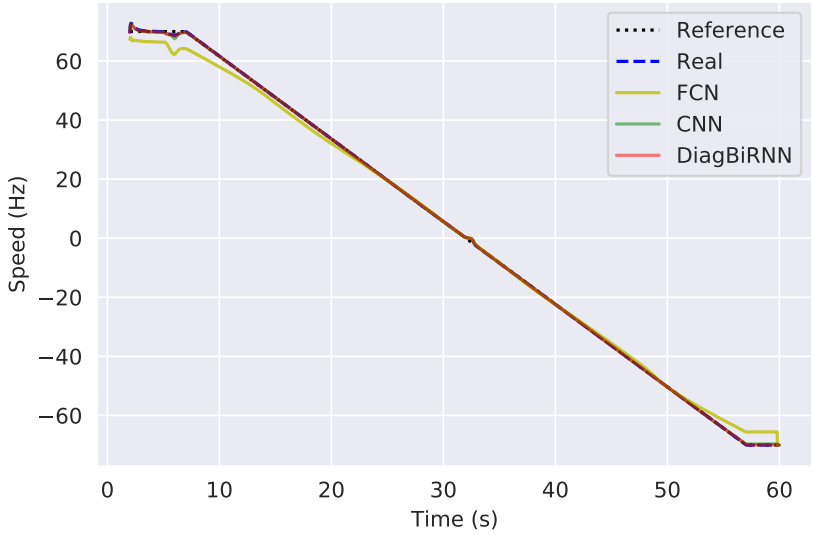


Figure 2.47: Results on Quasi-Static2 benchmark: speed trajectory.

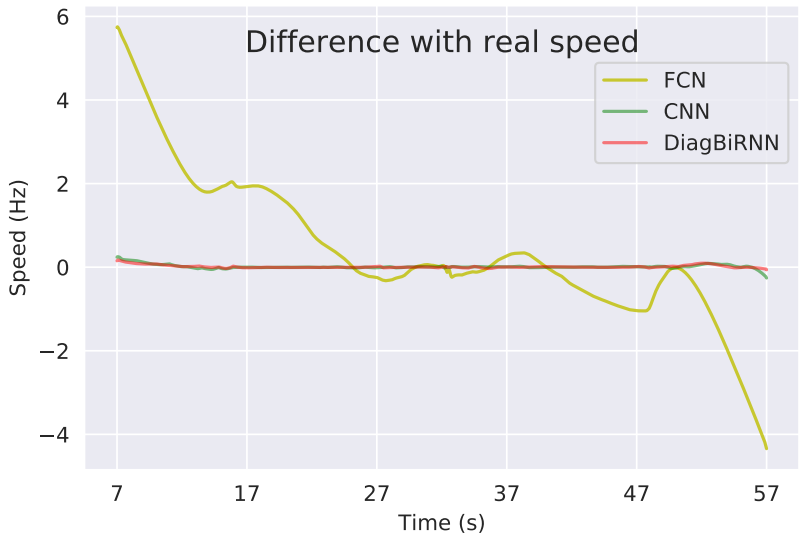


Figure 2.48: Results on Quasi-Static2 benchmark: error.

Figures 2.47 and 2.48 show plots for predicted speed trajectories and their error with ground truth on Quasi-Static2 benchmark, respectively. In both benchmarks it can be seen that FCN performs way worse than CNN and DiagBiRNN.

These benchmarks are a standard way of understanding various mathematical models used in the electrical motor drive control domain. All the plots and tables

we have presented so far establish a careful baseline for all the networks used as speed-torque estimators.

2.8 . Summary

In this chapter, we have investigated the problem of data-driven learning of electrical motor dynamics. We have also presented a novel encoder-decoder architecture called DiagBiRNN that uses diagonalized recurrent skip connections to learn the complex temporal dynamics. A novel loss function has been introduced to learn the model that avoids prediction bias. We have used transfer learning to fine-tune a model trained on large simulated data on a small raw sensor dataset. We have presented the first dataset that combines synthetic and real-world data for training deep neural networks for electrical motor tasks. We provided a realistic trajectory generator that allows us to generate realistic simulated data. Our experiments have shown the promising performance of the proposed method on a noisy sensor dataset collected in an industrial context. We have also conducted a detailed analysis of the global and local scope of the prediction performed on the test data.

We have used the proposed network and loss function to learn a model that can estimate an induction motor speed and torque from the currents and voltages. We show that, without care, these networks can be biased toward the patterns present in the data. We also emphasize the limitations of machine learning metrics in understanding neural network real performance. By using dynamic and quasi-static benchmarks, we show that electrical engineering metrics are better suited to evaluate the merits of different neural networks. Both types of metrics show that our proposed DiagBiRNN network performs better on benchmarks. Our results demonstrate the feasibility of AI solutions in modeling electrical motor dynamics, thus opening a new avenue of research in this area. In the next chapter, we extend the use of these networks in more applications beyond speed-torque estimation.

Chapter 3

Sensor Fault Recovery and Mechanical Fault Detection

3.1 . Introduction

Electrical motors undergo various stresses in harsh operating conditions, which may lead to failures. Many adjustable speed drives in industry and emerging applications such as automotive require high dynamic performance, robustness against motor parameter variations, and reliability. As the functions of electrical motors have become increasingly complex, continuous monitoring becomes progressively necessary. Various monitoring, observation, and fault detection techniques exist based on system dynamics or are data-driven using statistical machine learning and deep learning. Methods like mechanical fault detection and thermal anomaly detection rely on different types of sensor data to monitor and detect the operation of electrical motors. Various sensors are used to collect data for analytic purposes, such as current shunts, voltmeters, speed sensors, torque meters, inertial measurement units (IMUs), vibration sensors, temperature sensors, and acoustic sensors (microphones). Such sensors are rated for extreme operating conditions. Still, due to the nature of the operation, they may deliver erroneous data or, in some cases, might not be available in legacy systems.

A sensor breakdown happens if a failure occurs in any components, including the sensing device, transducer, signal processor, or data acquisition equipment. A power failure or corroded contacts can cause an abrupt failure in the sensor. In contrast, an incipient failure such as drift and precision degradation can be caused by deterioration in the transducer. Both an abrupt and an incipient failure can cause a non-permitted deviation from the characteristic property of a sensor. All these above-mentioned issues raise the challenge of detecting and isolating a faulty sensor. Three main types of faults can happen in a sensor: a) missing data at random, b) completely unavailable sensor, and c) erroneous sensor data. Many methods can deal with missing or unavailable data using a series of redundant sensors or estimators. Still, it is hard to deal with erroneous data as a recovery method must

first differentiate between erroneous and normal data. This chapter focuses on three different applications of neural networks in the realm of fault diagnosis in electrical motors. We focus on sensor breakdown, mechanical wear and tear, and excessive thermal operation. This work has resulted into a publication in ICLR Workshop on Deep Generative Models for Highly Structured Data 2022 and is also submitted as a patent.

3.2 . Related Work

The incipient fault detection in modern electrical motor drive systems has been approached in [91, 92, 93, 94]. For motor operation, current and voltage sensors are necessary for vector control algorithms [95, 96]. These sensors are very sensitive and can be broken [92, 93, 94]. Another important signal used in a motor drive system is the rotor speed. Mechanical sensors can be used to measure this variable [92]. Those elements are sensitive to the current drive and weather conditions [97] and can be dammaged.

Traditional monitoring and fault detection methods like [98, 97, 99] rely on system mechanics, redundant sensors, and estimators to overcome such sensor faults. One useful piece of information these methods use is the correlation between multiple quantities. These methods can easily rely on other quantities if one correlated quantity fails. This is not directly possible in the case of neural network-based monitoring and fault detection methods. We can use such information to train multiple neural networks that can take different input sets and become sensor fault tolerant. But such an endeavor requires system knowledge and a large amount of data.

Three standard and widely-used methods for dealing with missing data are interpolation, imputation, and matrix completion. Interpolation methods [100, 101] attempt to reconstruct missing data by capturing the temporal relationship *within* each data stream, but not the relationships *across* streams. Imputation methods [102, 103, 104] attempt to reconstruct missing data by capturing the synchronous relationships *across* data streams, but not the temporal relationships *within* streams. Matrix completion methods [105, 106, 107] treat the data as static ignoring the temporal aspect. They assume a specific model of the data generation process and the pattern of missing data.

Recently, some researchers attempted to impute the missing values with recurrent neural networks [108, 109, 110, 111]. The recurrent components are trained together with the classification/regression component, which significantly boosts the accuracy. [109] proposed using Gated Recurrent Units (GRU), which imputes missing values in healthcare data in a smooth fashion. It assumes that a missing variable can be represented as the combination of its corresponding last observed value and the global mean. The method achieves remarkable performance on healthcare data.

Generative Adversarial Networks (GANs) have been trained on time-series data which learn relationships between variables across time [112]. The property of learning relationships between different variables in an input time series is then used to recover missing data [113, 114, 115]. GANs can also denoise corrupted time series data, as shown in [116], meaning they can understand and recover from noise.

3.3 . Sensor Faults

Missing and erroneous data in all types of sensors can occur due to electromagnetic interference, thermal aging, and lag in the sensor data processor. This is present at random for a short duration or can be present with some known characteristics for an extended period. Sensor faults can be classified into nine categories, as discussed below.

3.3.1 . Fault Types

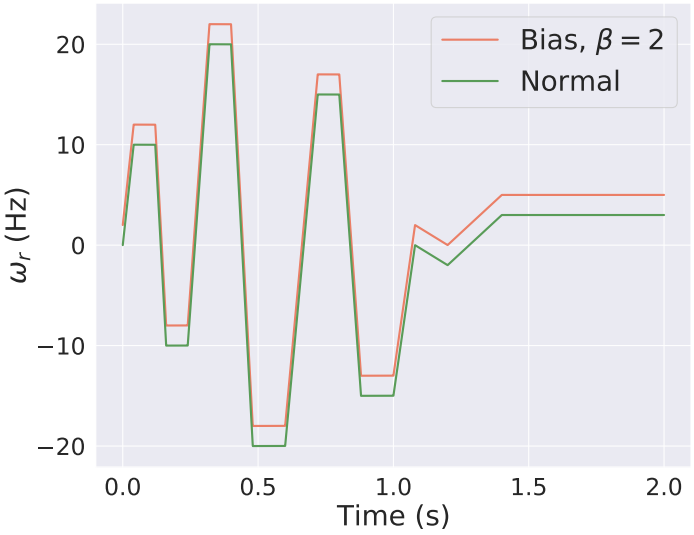


Figure 3.1: Bias

- **Bias (Figure 3.1)** : A constant offset from the nominal sensor signal statistics given by $Y_f = X + \beta +$ noise, where β is the constant offset value, X is the true value, Y_f is the faulty value, and noise is a disturbance within a tolerance range.[1]

[1] For simplicity, we drop the dependence on time t in the signal models.

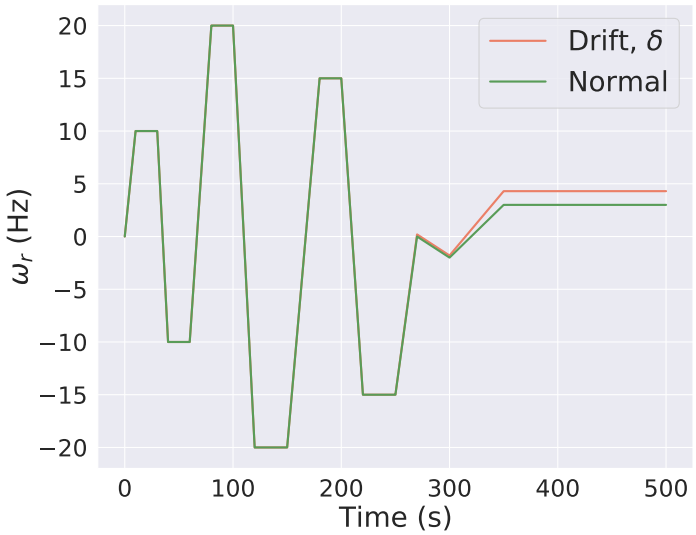


Figure 3.2: Drift

- **Drift (Figure 3.2) :** A time-varying offset from the nominal sensor statistics given by $Y_f = X + \delta +$ noise, where δ is the time-varying offset factor.

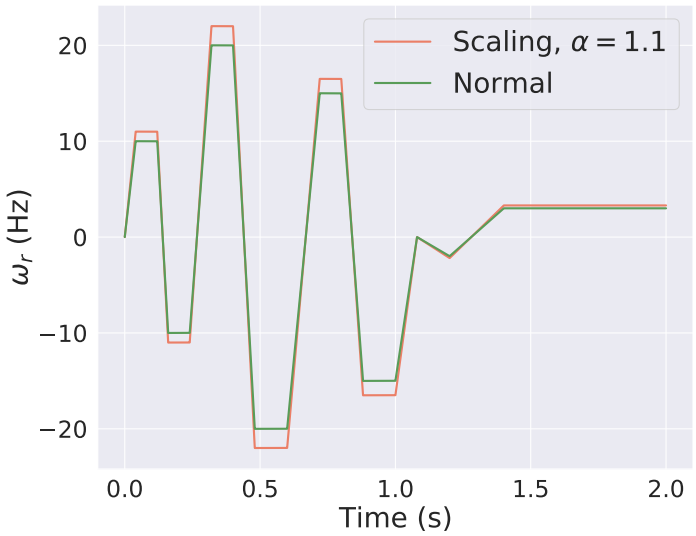


Figure 3.3: Scaling

- **Scaling (Figure 3.3):** Magnitudes are scaled by a factor, where the form of the waveform itself does not change. This is given by $Y_f = \alpha X +$ noise, where $\alpha > 0$ is a scaling constant that may be time-varying.

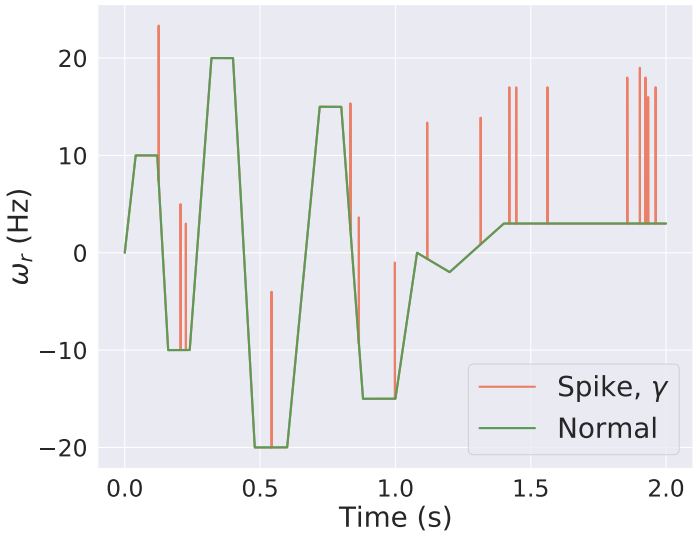


Figure 3.4: Spike

- **Spikes (Figure 3.4):** The sensor value is significantly above the true value for a single point. The density of spike faults within the signal can increase over time and is given by $Y_f = X + \gamma +$ noise where γ is an impulsive random signal.

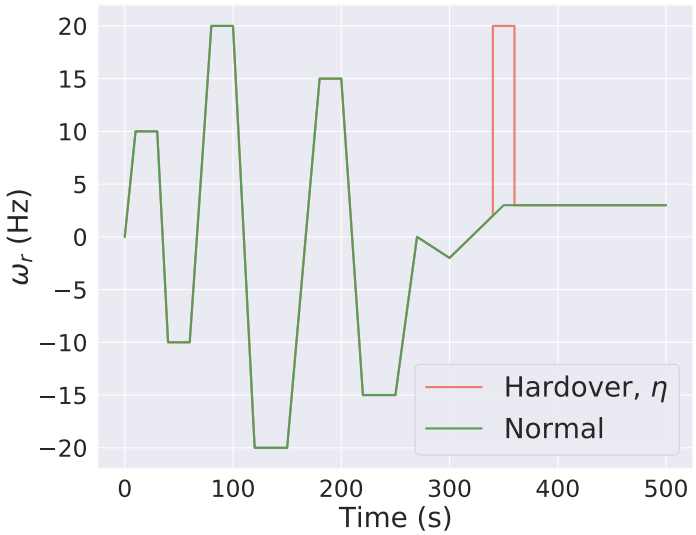


Figure 3.5: Hardover

- **Hardover (Figure 3.5):** The sensor value increases to the saturation point for a short period. This is given by $Y_f = X + \eta +$ noise where η is a rectangular pulse random signal.

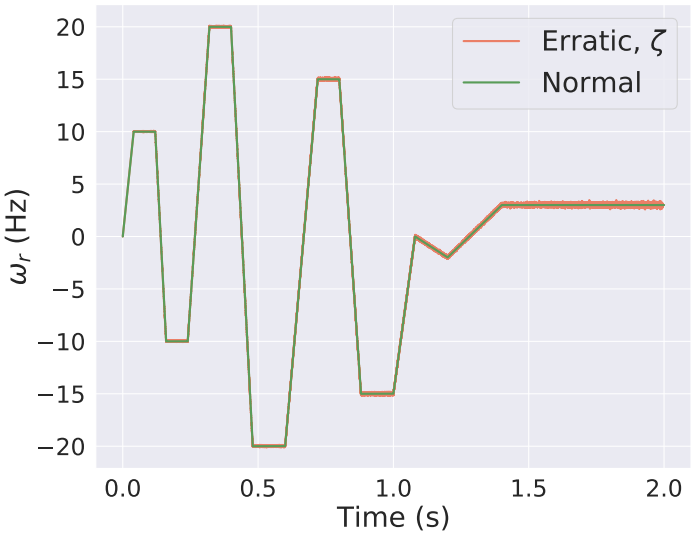


Figure 3.6: Erratic

- **Erratic (Figure 3.6):** The sensor value varies around the true value. The magnitude of this variance can increase over time and is given by $Y_f = X + \zeta +$ noise. ζ is a random perturbation whose variance tends to increase over time.

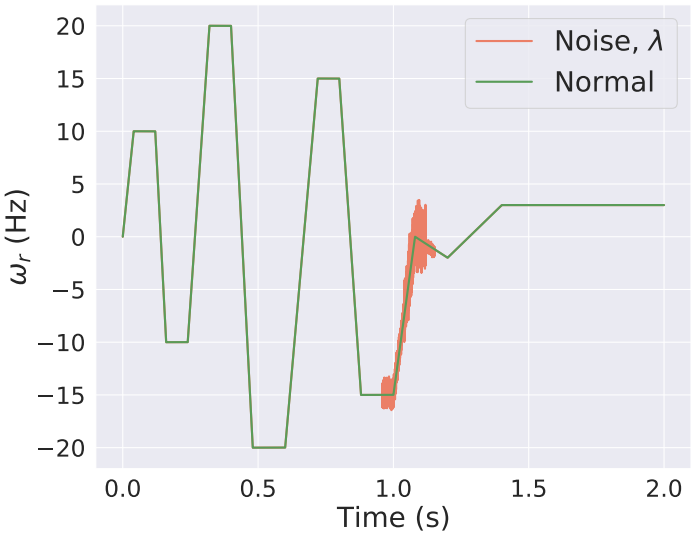


Figure 3.7: Noise

- **Heavy load noise (Figure 3.7):** A random time series is observed, $Y_f = X + \lambda$, where λ is time varying noise with short duration and often large standard deviation.

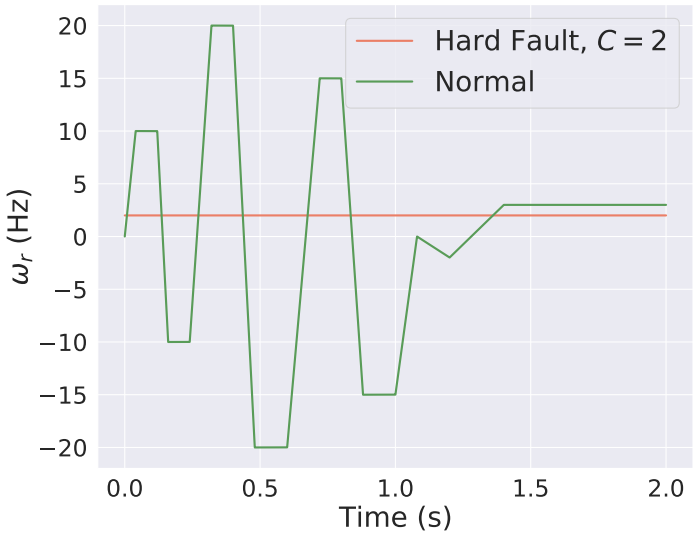


Figure 3.8: Hard Fault

- **Hard Fault (Figure 3.8):** The sensor output is stuck at a particular level expressed by $Y_f = C + \text{noise}$, where C is some non-zero constant. Hard fault can be due to loss of signal $(C = 0)$ or stuck sensor. Hard faults are usually treated as missing values.

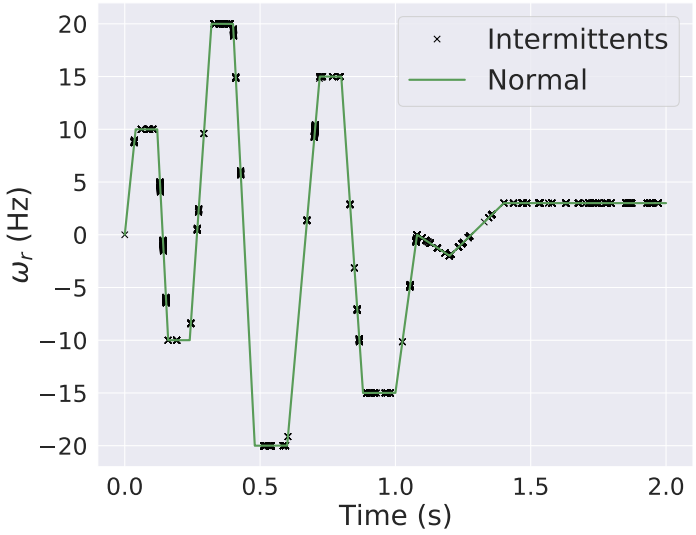


Figure 3.9: Intermittents

- **Intermittents (Figure 3.9):** Deviations from normal readings appear and disappear several times from the sensor signal. This results in some missing values (represented by black crosses in the figure). The occurence of such signatures is generally random.

3.3.2 . Fault Modeling

Here we define how to represent errors and missing values in multivariate signals. Consider a d-dimensional multivariate time series x, observed at $t = (t_0, ..., t_{n-1})$, denoted by $x = (x_0, ..., x_{n-1}) \in \mathbb{R}^{d \times n}$, where t is the observing timestamp, and $x_t = (x_t^j)_{1 \leqslant j \leqslant d} \in \mathbb{R}^d$ is the t-th observation. We can assume that the time-series has been recorded from d different sensors. Let $m \in \mathbb{R}^{d \times n}$ be a mask matrix that takes values in $\{0, 1\}$. The values signal whether the components of x exist or not, for example, if x_t^j exists then $m_t^j = 1$, otherwise it is equal to 0. $e \in \mathbb{R}^{d \times n}$ is an error matrix that contains value e_t^j at time t. The range of e_t^j is $[e_l^j, e_h^j]$, where e_l^j and e_h^j are the lowest and highest possible errors for sensor j. The purpose of multivariate time series imputation is to impute the missing values and correct the erroneous values in x as accurately as possible.

The following example provides an intuitive explanation of the formulation for an observed multivariate time series x, and its corresponding m and e variables, where "Na" designates a missing value:

$$x = \begin{bmatrix} 1 & 2 & 3 & \text{Na} & \\ \text{Na} & 2 & \underline{3.2} & 4 & \dots \\ 1 & \text{Na} & 3 & \underline{3.9} & \end{bmatrix}, m = \begin{bmatrix} 1 & 1 & 1 & 0 & \\ 0 & 1 & 1 & 1 & \dots \\ 1 & 0 & 1 & 1 & \end{bmatrix}, \tag{3.1}$$

$$e = \begin{bmatrix} 0 & 0 & 0 & 0 & \\ 0 & 0 & 0.2 & 0 & \dots \\ 0 & 0 & 0 & -0.1 & \end{bmatrix}, t = (0, 1, 2, 3, \dots). \tag{3.2}$$

Thus, in this example, the clean and complete time series could be

$$\hat{x} = \begin{bmatrix} 1 & 2 & 3 & 3.1 & \\ 1.5 & 2 & 3 & 4 & \dots \\ 1 & 2 & 3 & 4 & \end{bmatrix}. \tag{3.3}$$

Some methods use time differences as they have been designed for irregular time series. We define a matrix $\delta \in \mathbb{R}^{d \times n}$ that records the time lag between the current value and the last valid observed one. The components of δ are defined as follows:

$$\delta_{t_i}^j = \begin{cases} 0, & \text{if } i = 0 \\ t_i - t_{i-1}, & \text{if } i \neq 0 \text{ and } m_{t_{i-1}}^j (1 - e_{t_{i-1}}^j) = 1 \\ \delta_{t_{i-1}}^j + t_i - t_{i-1}, & \text{otherwise,} \end{cases} \tag{3.4}$$

where $m_{t_{i-1}}^j (1 - e_{t_{i-1}}^j) = 1$ means that the j^{th} component at time t_{i-1} has no error. In the previous example δ is given by

$$\delta = \begin{bmatrix} 0 & 1 & 1 & 1 & \\ 0 & 1 & 1 & 2 & \dots \\ 0 & 1 & 2 & 1 & \end{bmatrix}. \tag{3.5}$$

Our goal is to develop a method to estimate $\hat{x}$ from x. We will assume that m is known, but e has to be estimated.

3.4 . Dataset

Erroneous and missing values can be generated using nominal and abnormal statistics provided in the literature and sensor datasheets. Existing neural network methods for missing data imputation use datasets that have either missing data or can be synthetically generated during training. We first model the faults for different types of sensors using the process described in [117] to generate sensor faults in the training datasets. In our case, we consider the following types of electrical motor sensors: a) three current shunts measuring currents in three-phase (i_a, i_b, i_c), b) voltmeters measuring three phase voltages (u_a, u_b, u_c), c) encoder measuring speed ω_r, d) torque meter measuring torques τ, e) temperature sensors ϑ, and f) accelerometers measuring vibrations σ.

3.4.1 . Brushless Direct Current Sensor Faults Dataset

We collected a new dataset with faults in sensors that monitor Brushless Direct Current (BLDC) motors in Unmanned Aerial Vehicles (UAVs). The test bench setup is shown in Figure 3.10. Two different motors rated 880Kv and 1000Kv have been considered. Kv is number of revolutions per minute (rpm) that a motor turns when 1V (one volt) is applied with no load attached to that motor. Not to be confused with kV, the abbreviation for kilovolt. Six different sensors have been used to monitor thrust (T), torque (τ), voltage (u_{DC}), current (i_{DC}), angular speed (ω_r), and rotor shaft temperature (ϑ_r). Three flight modes, namely aerobatics, cruise, and racing, have been considered when running the motor. For each type of sensor listed above, we have used one faulty and one normal sensor to collect the dataset. Each motor is run for 15 minutes for each flight mode, with both normal and faulty sensors collecting data simultaneously. In total, we have 200 minutes of data collected at 80Hz.

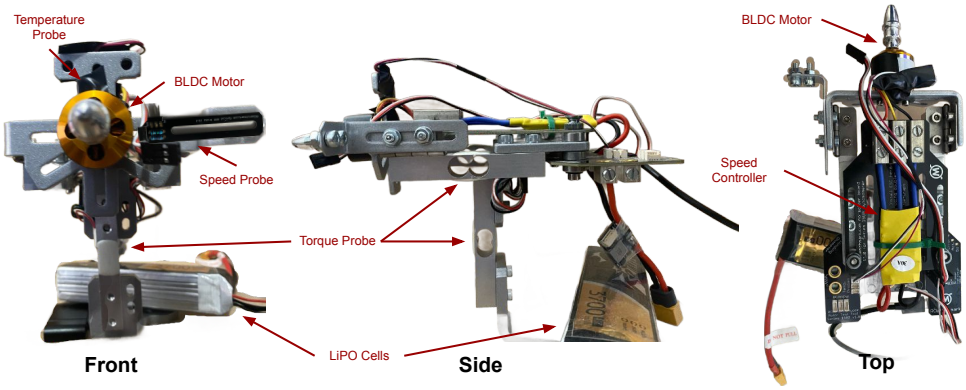


Figure 3.10: BLDC test bench to collect sensor data.

3.4.2 . Induction Motor Dataset

The Induction Motor dataset has been proposed earlier in Section 2.5.4 for designing neural speed-torque estimators which predicts speed (ω_r) and estimated torque (τ_{em}) from input currents (i_d, i_q) and voltages (u_d, u_q). This dataset has been collected at 250Hz.

3.4.3 . Permanent Magnet Synchronous Motor Temperature Dataset

Permanent Magnet Synchronous Motor (PMSM) Temperature dataset [118] is a publicly available dataset used to do thermal modeling to reduce the cost of placing thermal sensors deep inside moving parts of motors. The dataset consists of different experiments under real operating conditions. The following motor quantities are present in the dataset: currents (i_d, i_q), voltage (u_d, u_q), speed (ω_r), torque (τ_{em}). For the inference task, the following temperatures have been collected: permanent magnet (ϑ_{PM}), stator yoke (ϑ_{SY}), stator tooth (ϑ_{ST}), stator winding (ϑ_{SW}), ambient temperature outside of stator (ϑ_a), and coolant temperature (ϑ_c). The dataset has been collected at 2Hz.

3.4.4 . Broken Bar Detection Dataset

Broken bar detection dataset proposed in [119, 120] is a publicly available dataset that consists of electrical and mechanical signals recorded from 0.7457kW three-phase induction motor. The dataset consists of currents and voltages represented in abc frame, and we convert it to dq frame using Clarke-Park transformation as explained in 1.1.1.2. Drilling was done in the rotor to simulate the failure of the three-phase induction motor. The rupture rotor bars are generally adjacent to the first rotor bar; four rotors have been tested, the first with a break bar, the second with two adjacent broken bars, and so on rotor containing four adjacent broken bars. For all these conditions, the induction motor is operated at a constant nominal speed of 1715 rpm for 18 seconds, and (12.5, 25, 37.5, 50, 62.5, 75, 87.5, 100)% of nominal loads were applied. The nominal load of the load machine is 40Nm. The dataset has 40 experiments collected at 60Hz. Mechanical signals were collected using five axial accelerometers. These sensors capture vibration measurements in both the drive end (DE) and non-drive end (NDE) sides of the motor, axially or radially, in the horizontal or vertical directions. For the electrical signals, the currents were measured by alternating current probes, which correspond to precision meters. The voltages were measured directly at the induction terminals using the voltage points of the oscilloscope.

3.5 . Fault Detection and Recovery

In this section, we first show different existing time-series missing data imputation methods grouped into categories: statistical methods, recurrent

neural networks, and generative adversarial networks. Then we propose a method that can detect and recover errors along with missing data.

3.5.1 . Statistical Methods

Statistical methods use some kind of aggregation of nearby available values to find the missing values. Some of them are:

- **Mean** We replace the missing values with the mean value of the sensor calculated from the dataset.
- **KNN** [121] The missing values are replaced by using the k nearest neighbor samples. We try $k = 3,5,7$ when experimenting on the speed-torque dataset and choose $k = 5$ considering time and computation cost.
- **MF** [122] Matrix Factorization (MF) method is used to factorize the incomplete matrix into low-rank matrices and fill in the missing values. We use 100 epochs and a learning rate of 0.001 for MF.
- **MICE** [104] Multivariate Imputation by Chained Equations (MICE) fills the missing values by using an iterative regression model.

3.5.2 . Recurrent Neural Networks

Recurrent property of RNNs are utilized to interpolate missing values in time series data. Some of the proposed methods are:

- **GRUD** [109] is a recurrent neural network that uses a weighted combination of Gated Recurrent Units (GRU) output, last observation, and global mean to impute missing data.
- **M-RNN** [79] is a bi-directional RNN that uses hidden states in both directions of RNN to impute values.
- **BRITS** [123] This method uses bi-directional recurrent network to impute time series. It implicitly updates missing information and can be used directly for downstream tasks.

3.5.3 . Generative Adversarial Networks

The generative capability of GANs has been utilized to generate values for missing data using the following approaches:

- **GAIN** [113] is a GAN based imputation method that uses a hint vector that closely matches missing vector distribution to impute missing values.
- **2Stage GAN** [114] trains a GAN in two stages to impute missing data.
- **E^2-GAN** [115] is an end-to-end version that overcomes the inefficiency of 2stage training by using a single stage.

GRUD, M-RNN, BRITS, GAIN, 2Stage GAN, and E^2-GAN use δ defined in Eq. (3.4). Since there is no way of incorporating erroneous data, we treat them as missing for all these methods.

3.6 . Sceptic-GAN: To Recover From Error and Missing Data

This section describes the proposed method for generating imputed and corrected data motivated by Wasserstein GAN (WGAN) [124]. As we will see, both the generator and the discriminator in the designed method are grounded on the use of the DiagBiRNN structure (see Figure 3.11) introduced in Chapter 2. DiagBiRNN learns the multivariate and temporal relationships better than recurrent or feedforward networks used in [115].

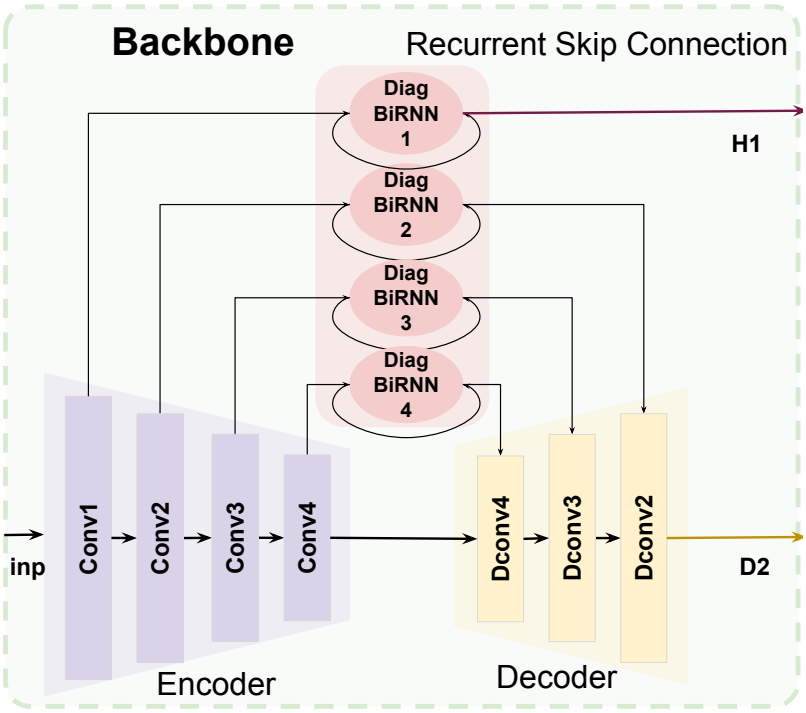


Figure 3.11: DiagBiRNN as backbone of generator and discriminator of proposed Sceptic-GAN.

3.6.1 . Generator

The generator uses DiagBiRNN layers as its backbone. We use all but the last deconvolutional layers to process the input time series, and the last layer is then replicated to have two heads, "Imp Deconv1" and "Err Deconv1". The "Imp Deconv1" layer outputs an imputed and corrected time series, whereas "Err Deconv1" outputs the error in the input time series. The structure of generator G is shown in Figure 3.12.

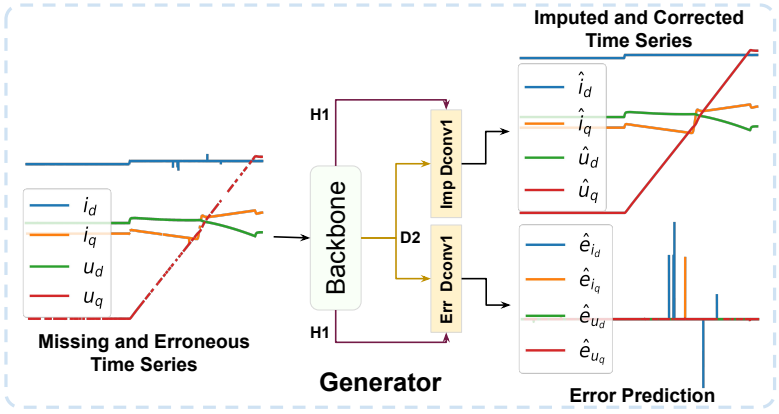


Figure 3.12: Generator network of Sceptic-GAN.

The generator uses input variables x, m, a noise variable z, and a sparse error variable e. During training, m and e are generated according to the fault models. z is sampled from a standard distribution $\mathcal{N}(\mu, \sigma^2)$ with mean $(\mu = 0)$ and standard deviation $(\sigma = 0.01)$. During testing, m is known and e is kept zero as test data already have erroneous values because of faults. The output of the generator G are the estimated variables given by

$$(\hat{x}, \hat{e}) = G(x, m, z, e) \tag{3.6}$$

In generator G, DiagBiRNN takes the following input signal

$$x_{\text{inp}} = m \odot x + (1 - m) \odot z + e \tag{3.7}$$

where $\odot$ designates the Hadamard product.

The proposed pipeline goal is to make the generator output probability distribution $P(\hat{x})$ close to the target distribution $P(x)$. We feed $(1 - m) \odot z$ as the noise into the generator autoencoder. The noise variable z plays a role analogous to the noise variable introduced in standard GANs.

The generator is trained to produce a new sample $\hat{x}$ that is most similar to x and sensor fault error $\hat{e}$ that is closest to e. The predicted error is then given as input to the fault classification head as shown in Figure 3.13. The fault classification head is a two layer network with a 1D convolution layer followed by a linear fully connected layer to predict fault logits.

The loss function for the generator is

$$\mathcal{L}_G = \lambda(\|x - \hat{x}\|_2^2) + \gamma(\|e - \hat{e}\|_2^2) + \mathcal{L}_{\text{CE}} - D(\hat{x}) \tag{3.8}$$

where $\lambda > 0$ and $\gamma > 0$ are hyper-parameters that control the weight of the time series reconstruction error, noise prediction error, and the output of the discriminator (D) loss. $\mathcal{L}_{\text{CE}}$ is the cross-entropy loss between fault classification logits and true fault types. In our case, we have 10 classes: 9 different types of

faults and no-fault. Note that the terms $\|x - \hat{x}\|$ and $-D(\hat{x})$ have a similar purpose. Since the GAN here is doing a pure regression task we can use this as an extra loss term just to help in learning. In turn, $-D(\hat{x})$ gives very global level information on the estimated signal compared to the former term.

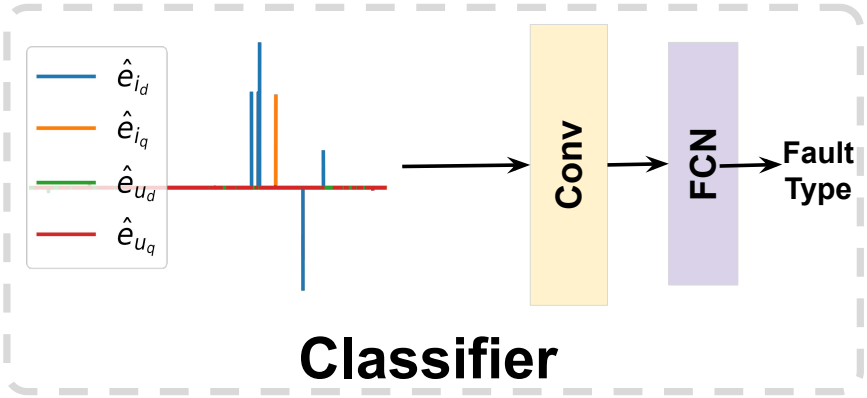


Figure 3.13: Fault classifier network of Sceptic-GAN.

3.6.2 . Discriminator

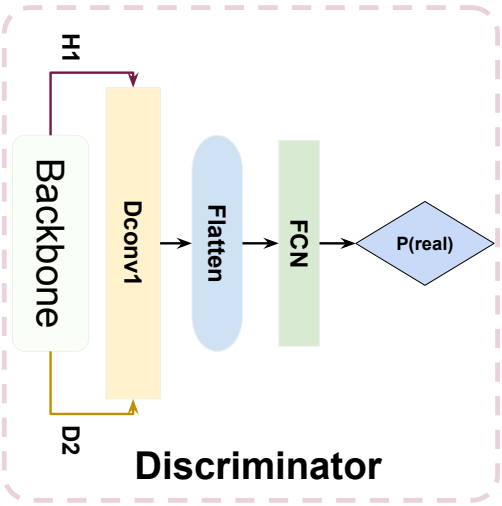


Figure 3.14: Discriminator network of Scpetic-GAN

The network structure of the discriminator D shown in Figure 3.14 also uses DiagBiRNN with one extra layer that takes the flattened output and feeds to a fully connected layer with a single output. The task of the discriminator is to distinguish between fake sample $\hat{x}$ and true sample x. The output of the discriminator is a probability that indicates the degree of authenticity. We try to find a set of parameters that can produce a high probability when we feed a true sample x and

a low probability when we feed a fake sample $\hat{x}$. Therefore, the loss function of the discriminator can be designed as follows:

$$\mathcal{L}_D = D(\hat{x}) - D(x). \tag{3.9}$$

We feed incomplete time series x or complete time series $\hat{x}$ into the discriminator. With the help of the autoencoder, the time series can be successfully handled. The autoencoder output is then passed to the fully connected layer that outputs the probability of being true. Minimizing $\mathcal{L}_D$ aims at increasing the discrimination between true signals (x), for which $D(x)$ should be maximal, and synthetically generated ones ($\hat{x}$) for which $D(\hat{x})$ should be minimal.

3.6.3 . Training

Sceptic-GAN is trained by alternating between steps where $\mathcal{L}_G$ is minimized and steps where $\mathcal{L}_D$ is minimized. For each step, we use Adam optimizer with learning rate 10^{-3}, inertial parameters $(0.9, 0.999)$, and weight decay of 10^{-5}. This alternating optimization technique can be viewed as a pratical way of solving the underlying minimax problem which is classically encountered when training GANs. To get the best performance, we update the generator k times and the discriminator once at every iteration ($k = 3$ in our experiments).

The complete pipeline is summarized in Figure 3.15.

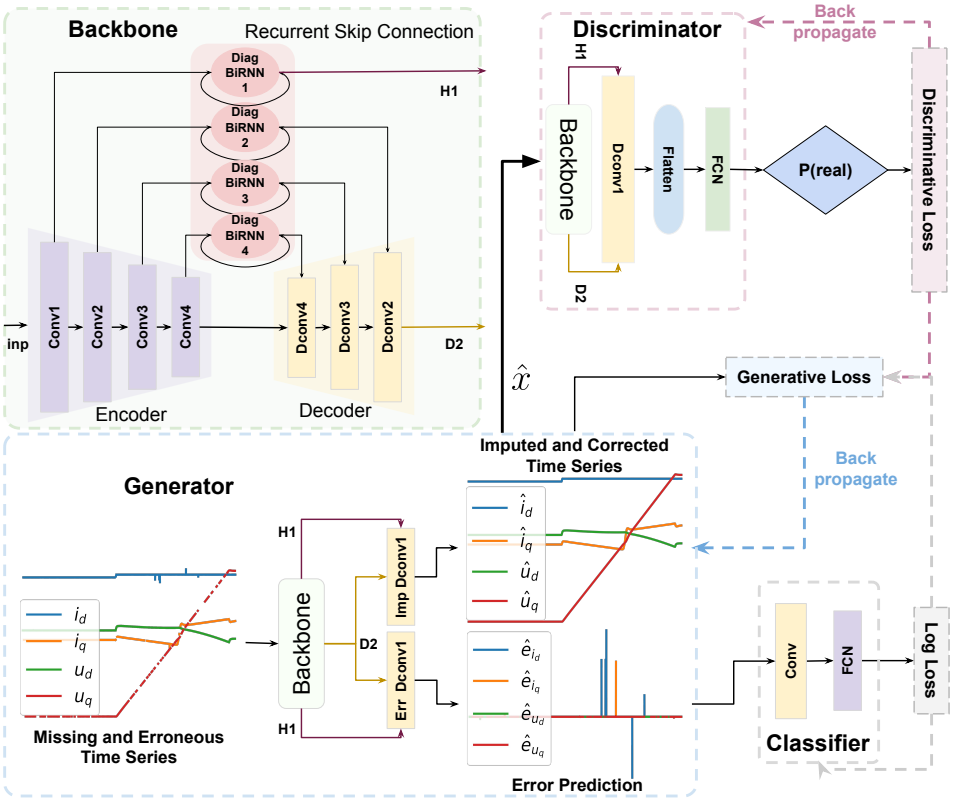


Figure 3.15: Proposed model architecture : Sceptic-GAN

3.6.4 . Sceptic Score

Using our proposed generator network, we can overcome many difficulties associated with missing data. One major challenge in using a neural network to impute and correct faulty sensor data is not knowing when the sensors cannot be trusted, so as to take corrective measures like shutting the drive system. It is easy to know when input data are missing; however, knowing erroneous values is hard. By aggregating the generator error prediction e, we can easily score each sensor reliability and decide when to trust the imputed and corrected values. Sceptic score of a sensor j recording for duration T is given by

$$\mathcal{SS}_j = \frac{100}{T} \sum_{t=1}^{T} \frac{\hat{e}_t^j}{\hat{x}_t^j + \epsilon} \tag{3.10}$$

where ϵ is a small positive constant.

3.7 . Experiments

In this section, we first show the experiments conducted for fault detection in the aforementioned datasets. We compare our proposed fault detection and recovery method with other state of the art methods discussed in Section 3.5.

Fault	Range	Median	Sensors and References
Bias	1.2% to 60% 0.1% to 0.15%	20% 0.12%	ϑ and ω [125] τ [126]
Scaling	2.5 to 4.8 0.3 to 0.7	3.28 0.45	σ [127] T [128]
Drift	6% to 75% 0.1% to 0.2%	29% 0.17%	T [129] ϑ [125] τ [126]
Noise / Spike / Erratic	2.5% to 250% 0.1% to 2% 1% to 6%	20% 0.17% 2.4%	T, τ and ω [130] σ [131] τ [126] i and u [132]
Hard Fault	0% to 100%	NA	All Sensors
Intermittent	2 to 10 drops	8 drops	All sensors [129]

Table 3.1: Faults leading to missing and erroneous values in different types of sensors. The indicated percentage values in the second column correspond to the minimum and maximum allowed change of nominal range when a specific type of fault is present.

Once we have recovered signals, we use them as input in downstream tasks like speed-torque estimation, broken-bar detection, and thermal modeling. This shows how useful recovered signals are as input to other machine learning objectives.

Table 3.1 shows the statistics of different types of faults that can happen in different types of sensors. The sources from which these values have been obtained has been cited in the last column along with the sensor names. These values have been used to simulate faults in the four datasets for training and testing of different types of fault detection and recovery methods.

3.7.1 . Fault Detection

Table 3.2 shows MSE between true and imputed values for BLDC, Induction, PMSM, and Broken Bar datasets. We can see that our proposed method provides the best imputation and corrected values. Sceptic-GAN can utilize erroneous values to recover both missing and erroneous ones, whereas other methods treat them as missing ones. This is one reason why our method performs better. Another significant advantage of Sceptic-GAN is a good pretrained backbone that has been shown to be efficient for learning electrical motor dynamics from sensor data. Such a network can utilize multivariate intra-component and temporal relationships better than simple, fully connected, or recurrent networks.

To demonstrate the usability of the sceptic score, we purposefully destroy a temperature sensor by heating it at $200°C$. This is above the max operating

Method (↓)	BLDC	Induction	PMSM	Broken Bar
		Non NN		
Mean	0.518	0.632	0.672	0.757
KNN	0.282	0.392	0.432	0.517
MF	0.334	0.483	0.523	0.608
MICE	0.314	0.400	0.440	0.525
		RNN		
GRUD	0.325	0.486	0.526	0.611
M-RNN	0.411	0.491	0.528	0.603
BRITS	0.264	0.396	0.436	0.521
		GAN		
GAIN	0.287	0.399	0.416	0.523
2-Stage	0.231	0.373	0.413	0.498
E^2	0.219	0.352	**0.372**	0.477
Sceptic-GAN	**0.192**	**0.329**	0379	**0.458**

Table 3.2: MSE results for Sceptic-GAN and other imputation methods.

temperature of $125°C$, causing a very high drift. This sensor and other sensors are then used to record the BLDC motor for 5 minutes. Sceptic GAN is then used to recover these values. We then compute the sceptic score of the temperature sensor, which comes out to be 33.9%, above the 29% median value of drift from Table 3.1.

3.7.2 . Fault Classification

The fault classifier head has a 10-class output where we want to predict the type of faults (1 out of 9) or no-fault. The macro f1-score obtained on the four datasets are BLDC (91.8%), Induction Motor (96.1%), PMSM (89.2%), and Broken Bar (90.5%). It should be noted that we only consider that there is one fault present in the input signal at any given time, making it a multi-class problem. In reality, there is a small likelihood that multiple faults can be present simultaneously, but this is a more complicated problem to solve beyond the scope of this study.

3.7.3 . Performance of Downstream Tasks

Once we have corrected and recovered faulty signals using different methods, we use those signals as input to models trained for speed-torque estimation, broken bar detection, and thermal modeling tasks.

3.7.3.1 . Speed-Torque Estimation

We also compare our proposed method with baseline methods by using them to impute value in test data in downstream tasks. For speed-torque estimation, we use real motor test set from Section 2.5.4 and introduce missing and erroneous values using Table 3.1. Then we use all imputation methods to resolve the test set and feed it to the speed-torque estimator trained in Chapter 2. Table 3.3 reports

Method	Speed ω_r		Torque τ_{em}	
	MAE	**SMAPE**	**MAE**	**SMAPE**
DiagBiRNN	**0.03**	**18.76%**	**0.04**	**38.46%**
Mean	0.29	19.01%	0.28	38.64%
KNN	0.22	18.93%	0.21	38.58%
MF	0.18	18.81%	0.37	38.62%
MICE	0.18	18.74%	0.35	38.59%
GRUD	0.19	18.75%	0.36	38.60%
M-RNN	0.21	18.75%	0.30	38.58%
BRITS	0.17	18.77%	0.24	38.56%
GAIN	0.16	18.79%	0.25	38.60%
2-Stage GAN	0.12	**18.72%**	0.22	38.55%
E^2-GAN	0.11	18.76%	0.14	**38.49%**
Sceptic-GAN (Ours)	**0.03**	18.76%	**0.12**	38.50%

Table 3.3: Metrics for speed-torque estimation with imputed data.

MAE and SMAPE of estimated speed (ω_r) and torque (τ_{em}). The first row shows the results obtained when the input data is without any fault. We can see that Sceptic-GAN imputed values provide better MAE for speed (ω_r) and torque (τ_{em}). One interesting observation is that SMAPE for speed (ω_r) is even better for 2-Stage GAN compared to the baseline. This could be due to the fact that some of the recovered faulty values are closer to the training set leading to an overfitted prediction. Overall, Sceptic-GAN is better at imputation of missing values and the only method to identify and correct errors in the recorded signals.

3.7.3.2 . Broken-Bar Detection

Model	**Baseline**	**Mean**	**KNN**	**MF**	**MICE**
Accuracy	**92.57**	72.16	79.13	78.86	83.17

Model	**GRUD**	**M-RNN**	**BRITS**	**GAIN**	**2-Stage GAN**	**E^2-GAN**	**Sceptic-GAN (Ours)**
Accuracy	85.92	86.49	86.12	91.07	90.67	90.14	**91.39**

Table 3.4: Accuracy of the bearing fault prediction task using imputed and corrected values fed as input to the time-series ResNet.

The bearing fault detection dataset is a classification dataset where different sensor quantities are used to classify motor-broken bearing faults into one of five classes. This is a challenging dataset since, when we introduce sensor faults, the input quantities end up having two sources of faults present in them. Since we control sensor error generation, we can train our GAN to detect and use the imputed values to classify broken faults. Table 3.4 shows the accuracy of ResNet

when applied to imputed values from all imputation networks. From the results of downstream task of broken bar detection, we can say that the first stage of fault recovery by Sceptic-GAN has performed quite well when compared to other methods.

3.7.3.3 . Thermal Modeling

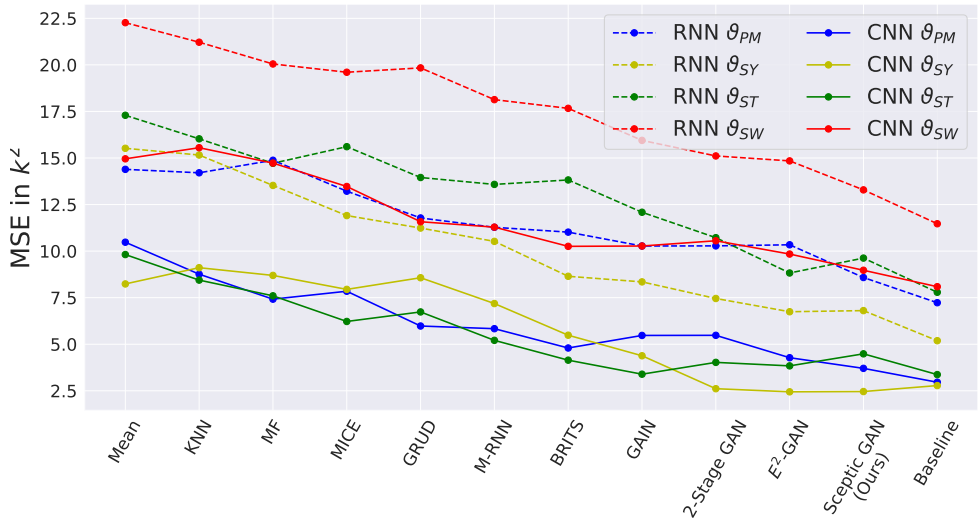


Figure 3.16: MSE of RNN and CNN outputs predicted from imputed inputs in thermal modeling task.

Thermal modeling dataset provides a unique analysis point as the imputation problem is harder. This is because temporal relationships between ϑ_a, ϑ_c and other non-thermal quantities persist over a long time horizon. Given that our GAN operates on a window of size 100, such relationships will be tough to capture, and imputation will be done based on local information only. Figure 3.16 shows results for $\vartheta_{PM}, \vartheta_{SY}, \vartheta_{ST}, \vartheta_{SW}$ using RNN and CNN networks proposed in [118]. The x-axis shows all imputation methods with MSE in the y-axis. We can see that our method is the closest to the baseline. E^2-GAN also shows good results by sometimes performing better than Sceptic-GAN as well as the baseline.

3.8 . Summary

This chapter investigates three applications where our proposed fault detection and recovery methods provides effective solutions. We focused majorly on the problem of sensor faults. We presented nine types of sensor faults and built a model based on literature and sensor datasheets. This fault model is for sensors that are explicitly used in the electrical industry. We show how the fault model can generate faults in the existing datasets. We also presented a new Brushless DC motor dataset where faults were generated by deteriorating the sensors in the

system. We discussed several existing state-of-the-art missing data imputation methods to solve the proposed sensor fault detection and recovery problem. To overcome the limitations of these methods, we introduced Sceptic-GAN, which can recover missing and correct erroneous values. We show the good performance of Sceptic-GAN on four datasets and demonstrate the usability of recovered signals in downstream tasks of speed-torque estimation, broken-bar detection, and thermal modeling.

Chapter 4
Noise Processing, Robustness, and Generalization

4.1 . Introduction

In this chapter, we introduce one adaptation and two *litmus tests* for the neural networks trained for applicative purposes such as input-output relationship modeling, speed-torque estimator, thermal modeling, and fault detection. The objective is to identify if the neural networks are suitable for real-world electrical motor operations.

4.1.1 . Noise in Real World Data

DiagBiRNN experiments conducted in Chapter 2 for speed-torque estimation were limited to simulated data. In the real world, accurately measuring currents and voltages is challenging due to temperature variations and noise. The primary source of measurement noise is the inherent inaccuracies in sensors that measure quantities like currents and speed. We show that it is possible to learn electrical motor dynamics in the presence of noisy observations. To better characterize the model identification process from noisy observations and to ensure its reliability, we collect, analyze, and characterize real-world motor data. Noise properties derived from the raw data allow us to design different denoisers, which are then used to remove noise. This work has been published in IEEE Transactions on Industrial Electronics in 2022.

4.1.2 . Robustness in Real World Operations

Depending on where electrical motors are used, heavy industry or household appliances, a different degree of operating reliability is required from the motors. A brushless DC motor inside a rotational drive has a consistent operational paradigm. In contrast, an induction motor inside a tunnel boring machine has to face extreme heat, humidity, and dust while having a very inconsistent operation cycle. This operating environment often leads to unforeseen situations that can be treated as perturbations to the inputs of the neural networks used in various applications.

Therefore, neural networks used in speed-torque estimation, sensor fault recovery, broken bar detection, and thermal modeling require a thorough robustness analysis. This work has been published in NeurIPS RobustSeq Workshop 2022.

4.1.3 . Generalized Neural Networks for Real World Applications

Electrical motors come in different shapes and sizes for different types of applications. We discuss shortly the generalization of neural networks trained for speed-torque estimation for different motor powers. In Chapter 1, we started with the notion that training speed-torque estimator requires large simulated and small real datasets. In this context, we experimented with a 4kW powered electrical motor in Chapter 2. We conduct a generalization study to understand how this speed-torque estimator behaves when a new powered motor is used during inference.

4.2 . Related Work

4.2.1 . Noise Handling

Noise reduction in time-series is a very evolved field with a multitude of research involving various methodologies. Some of the techniques that can be easily applied are linear smoothing filters and non-linear filters [24, 25]. Kalman filter [26, 27, 28, 133, 134] is widely used in noisy observation where a state-space based estimation is done which takes the system model as input. Also, there are transform based methods like those based on wavelet transforms [29, 30, 135] which remove noisy components from transformed sensor data. Variational methods often based on the total variation [31, 32, 33] have also been used in signal denoising and change detection, providing a robust and often more flexible solution over linear filter based denoisers. Kalman filter, transform-based and variational methods require prior knowledge about the noise/signal statistics for efficient denoising. Deep learning based method like stacked autoencoders [34] have been shown to be promising in learning to remove noise. However, neural network denoisers require a large amount of data with non-noisy ground truth, which is not possible in the case of electrical motors.

4.2.2 . Neural Network Robustness

Goodfellow et al. [36] proposed the Fast Gradient Sign Method (FGSM) to generate ℓ_∞ bounded adversarial attacks. This is a white box attack, i.e., it has access to network structure, parameter weights, and all the related training details. FGSM is a single-step attack, Madry et al. [37] proposed a multi-step variant of FGSM called Projected Gradient Descent (PGD) attack. Kurakin et al. [136] proposed an optimized FGSM, Iterative Gradient Sign Method (IGSM), which adds perturbations in multiple smaller steps and clips the results after each iteration ensuring that the perturbations are restricted in the neighborhood. Dong et al. [137] added momentum to IGSM attacks. Moosavi et al. [138] proposed Deepfool

as a non-targeted attack that tries to find the decision boundary closest to the sample in the input space and then uses the classification boundary to fool the classifier. Moosavi et al. [139] proposed a universal image-agnostic perturbation attack method that fools the classifier by adding a single perturbation to all images in the dataset. In [140], the authors propose a general framework for the generation of adversarial examples in both classification and regression tasks for applications in the image domain. Robustness of networks applied on tabular data and regression task has been investigated in [141]. A better attacker for regression problems has been proposed in [142]. This work has been extended in [143] to understand effect of different input variables in the stability of neural networks and has been applied in designing robust fault detection neural networks for UAVs [144].

4.2.3 . Generalization of Deep Neural Networks

In [145], authors derived generalization bounds for various learning algorithms based on their robustness. A number of theoretically oriented works have focused on characterizing the generalization properties of neural networks by offering bounds on their generalization error [146, 147, 148]. Detecting distribution shifts is another sub-area of domain generalization research. In [149], authors try to solve the problem of attributing performance differences to the type of distribution shift (covariate or concept) based on the underlying data generating mechanism. Most of the existing work focus on deep learning classifiers and image domain. Less attention has been paid to the performance of deep neural networks for regression problems. In [150], authors build upon the robustness and generalization framework proposed in [145]. They present a new method of computing generalization errors and introduce a new regularizer for regression networks.

4.3 . Denoising Currents and Voltages

In this section, we first present the measurement noise modeling process. Then various strategies for denoising currents and voltages are discussed with the proposed denoising method.

4.3.1 . Noise Modeling

The noise of currents (i_d,i_q) and voltages (u_d, u_q) have been modeled in a way similar to [151]. To that extent, static parts of the experimental data have been collected, i.e. the periods during which both speed and torque are constant. For each static part, assuming that the non-noisy "true" currents and voltages signals coincide with their mean values on the static part, the temporal noise signals for the two currents and the two voltages have been determined. Gathering all the static parts of our experimental data, the temporal distributions of the noise corrupting each signal have been obtained. Figure 4.1 shows the distributions of noise in currents (i_d,i_q) and voltages (u_d,u_q).

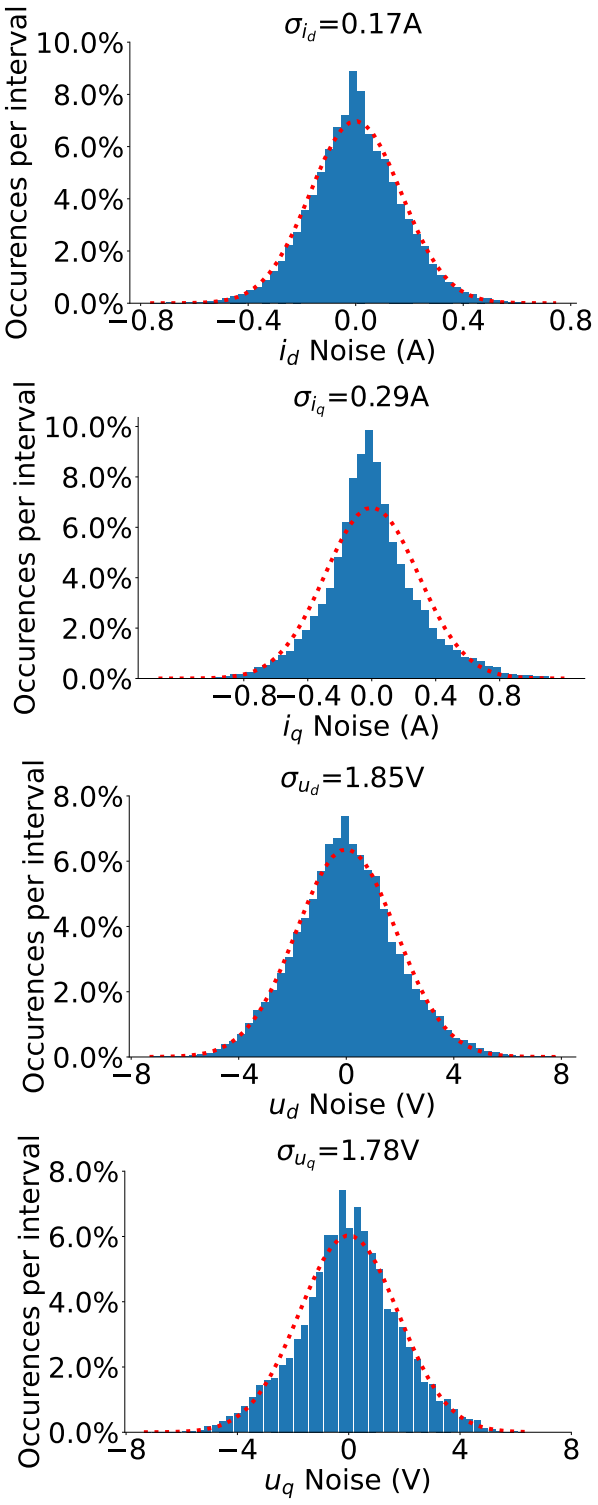


Figure 4.1: Noise distributions of currents (i_d,i_q) and voltages (u_d,u_q) from real data.

Relying on ergodicity and making the assumption that the real distributions can be approximated by Gaussian ones, the statistical characteristics of the noises have been deduced. The Gaussian approximations are shown by the red lines in figures. These show that the noises approximately follow normal distributions with zero mean and standard deviations equal to $\sigma_{i_d} = 0.17A$, $\sigma_{i_q} = 0.29A$, $\sigma_{u_d} = 1.85V$, and $\sigma_{u_q} = 1.78V$, respectively.

4.3.2 . Standard Denoisers

Extended Kalman filter (EKF) [27] is a state-space based non-linear filtering approach. A diagonal measurement noise covariance matrix is chosen using the noise variances estimated in Section 4.3.1 and the state transition matrix is an identity matrix I_2. Wavelet transform (WT) denoising is a non-linear estimation method operating on each wavelet coefficient separately. The adaptive Bayes Shrink algorithm [152] has been used to soft threshold wavelet coefficients using the noise standard deviations identified in Section 4.3.1. Minmax-concave total variation (MCTV) [33] is a non-linear variational method. It has been used with $K = 100$ maximum iterations, a root mean square error of 10^{-3} as convergence criteria, a regularization constant $\lambda = \sqrt{\sigma T}/5$ with σ the standard deviation defined in Section 4.3.1 and T the duration of the signal, and the non-convexity parameter $\alpha_{nc} = 0.3/\lambda$, after having also tried other values for the numerator. Unlike EKF, WT and MCTV, denoising auto-encoder (DAE) is a neural network based technique. It consists of 3 layers of 1-D convolutions and 3 layers of 1-D deconvolutions with a number of channels equal to 1-32-64-128-64-32-1.

These denoisers do not work properly on real motor data. EKF often fails when there is a sudden transition in noise amplitude which could be resolved using Adaptive EKF [153]. The denoised outputs of WT and MCTV exhibit staircase effects in ramp parts of the signal (with less error for MCTV). DAE solves these problems and gives the smoothest denoised output, but leads to incorrect predictions at the start and the end of a ramp.

4.3.3 . Meta-Denoiser

To overcome the problems of measurement noise, we introduce the Meta-Denoiser (MD) shown in Figure 4.2. MD is an extension of DAE where the last deconvolution layer gives the same number of feature channels as its input. It is then fed to the segmentation head and the denoiser head to predict segmentation and denoised outputs. By training it to identify dynamic and static parts, the model is able to identify the correct start and end of a ramp, thereby overcoming the problem of DAE.

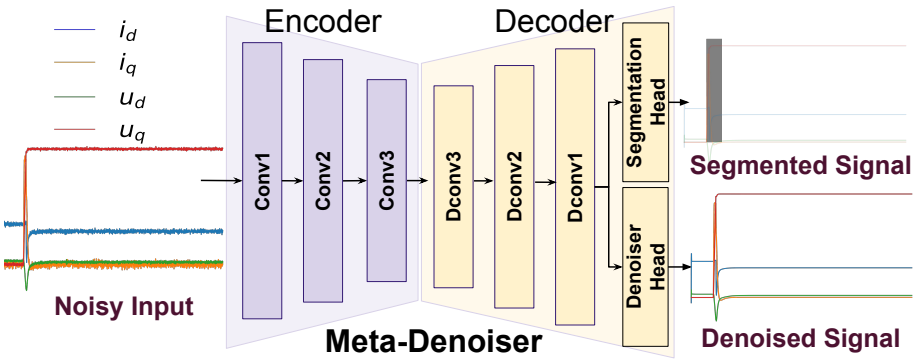


Figure 4.2: Meta-Denoiser architecture.

MD is trained using the MD-Joint loss function:

$$\mathcal{L}_{\text{MD-Joint}} = \alpha \mathcal{L}_{\text{BCE}} + (1-\alpha)\mathcal{L}_{\text{MSE}} \quad \begin{cases} \alpha = 0.5 & \text{if } y^i \in \text{Simulated Data} \\ \alpha = 1 & \text{if } y^i \in \text{Real Data} \end{cases} \tag{4.1}$$

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{T}\sum_{t=1}^{T}(y_t^i - \hat{y}_t^i)^2\right) \tag{4.2}$$

$$\mathcal{L}_{\text{BCE}} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{T}\sum_{t=1}^{T}(-z_t^i \log(\hat{z}_t^i) - (1-z_t^i)\log(1-\hat{z}_t^i))\right) \tag{4.3}$$

where y_t^i and $\hat{y}_t^i$ are the respective values of output and predicted sample i at time-step t for the denoising task, z_t^i and $\hat{z}_t^i$ are the respective classification probabilities of output and predicted sample i at time-step t for the segmentation task, and N is the number of training samples where each sample is of duration T. Eq. (4.1) is made of a MSE part expressed in Eq. (4.2), accounting for the denoising (regression) task, and a widely used binary cross entropy (BCE) loss [154] in Eq. (4.3). In case of simulated data, as the ground truth is available for both denoising and segmentation tasks, we use $\alpha = 0.5$ to give equal weights to the two loss function parts. For real motor data, since the denoised trajectory is not available, the denoising loss cannot be computed and we set $\alpha = 1$. Dynamic and static parts is obtained using a simple change detection algorithm [155].

4.4 . Denoising Experiments

To train MD and DiagBiRNN, the dataset has been partitioned into four parts: 70% (resp. 30%) of the simulation data are used for training (resp. validation), whereas 20% (resp. 80%) of the real motor data for fine-tuning (resp. testing) purpose. The fact that the majority of the real data has been used for testing is in line with real industrial needs. DAE and MD have been trained with mean square error (MSE) loss function and a stochastic gradient descent optimizer with the following hyperparameters: 100 epochs, learning rate of 0.01, and batch size

of 128. DiagBiRNN uses the training strategy and hyperparameters described in Section 2.6. EKF, WT, MCTV, DAE and MD have been used to denoise the currents (i_d, i_q) and voltages (u_d, u_q). The denoised currents and voltages are then used to predict the speed ω_r and the torque τ_{em} using DiagBiRNN trained on non-noisy simulated data from Chapter 2.

4.4.1 . Simulated Benchmarks

To analyze the effect of noise on DiagBiRNN, a set of experiments are performed on simulated data from Section 2.5.3. The absence and presence of noise have been considered. Real motor data from Section 2.5.4 have been used for experiments with real noise. Based on different combinations of test data and training conditions of DiagBiRNN, the following cases are studied:

i) **Case A:** NN estimator trained on non-noisy simulated data, applied on non noisy simulated data.

ii) **Case B:** Estimator from Case A applied to noisy simulated data.

iii) **Case C:** Estimator trained on simulated data with noisy currents and voltages and non-noisy speed and torque, applied to noisy simulated data.

iv) **Case D:** Transfer learning presented in Chapter 2 is used to train and predict on real motor data.

Method	**Speed ω_r**		**Torque τ_{em}**	
	MAE	**SMAPE**	**MAE**	**SMAPE**
CASE A	0.03	18.7%	0.04	38.5%
CASE B	0.05	20.1%	0.13	41.3%
CASE C	0.05	19.7%	0.13	41.0%
EKF[27]	0.05	19.4%	0.13	41.0%
WT[30]	0.05	19.4%	0.12	41.0%
MCTV[33]	0.05	19.4%	0.12	40.1%
DAE[34]	0.04	19.0%	0.09	39.7%
MD (Ours)	**0.04**	**18.8%**	**0.05**	**38.9%**

Table 4.1: Aggregated ML metrics for the predictions done on all simulated benchmarks.

Table 4.1 shows the aggregated ML metrics obtained by DiagBiRNN on the simulated benchmarks with and without denoising using various methods. Small MAE and SMAPE values are desirable for a good prediction model. It can be observed that Case B performs worse than Case A since DiagBiRNN alone has no way of cancelling the noise. Case C performs relatively better than Case B since training is performed on noisy simulated data. Amongst standard denoiser, minor

improvements with MCTV and DAE are observed. However, the proposed MD offers the best performance among all the evaluated methods, with quantitative results approaching the ones obtained in the non-noisy Case A.

Method	$t_{2\%}$ (ms)	$t_{95\%}$ (ms)	E_{fol} (Hz)	$D\%$ (%)	E_{ss} (Hz)	$\Delta\tau_{max}$ (%τ_{nom})
"Real"	**44.0**	**960**	**-0.02**	**2.16**	**0.00**	**32.7**
CASE A	44.1	956	0.01	2.32	0.03	32.8
CASE B	44.2	**950**	**-0.01**	2.26	**-0.01**	32.8
CASE C	44.2	955	**0.03**	**2.39**	**0.04**	32.8
EKF[27]	**44.1**	952	**0.02**	**2.48**	**0.04**	32.6
WT[30]	44.2	955	**0.03**	**2.37**	**0.04**	**32.7**
MCTV[33]	**44.1**	956	0.02	2.34	**0.04**	**32.7**
DAE[34]	43.8	**950**	0.00	2.33	0.02	**32.7**
MD (Ours)	44.2	**958**	**-0.01**	**2.23**	**0.01**	**32.7**

Table 4.2: Performance metrics for the predictions performed on simulated Dynamic-Speed1 benchmark

Table 4.2 shows performance metrics for Dynamic-Speed1 benchmark. Our objective is to be as close as possible to "Real" values given in the first row, "Real" standing here for "non-noisy (simulated)". It can be seen that MD-DiagBiRNN is the closest one to the "Real" for almost all metrics, again led to estimate speed and torque.

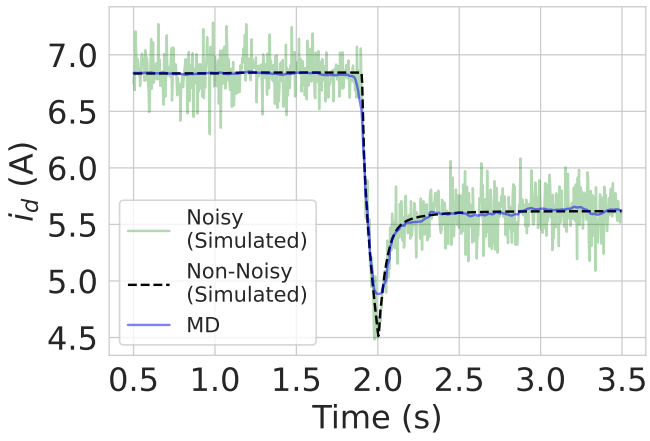


Figure 4.3: Meta-Denoiser on current i_d from Dynamic-Speed1 benchmark.

Figures 4.3 show MD output when noisy current i_d is given as input. It can be observed that the denoised output is very closely to the non-noisy simulated current i_d. There are two small problems: the first one is that at around 1.9 seconds the MD output starts the ramp before the simulated one does, and the second problem is that the end of ramp is not perfect at 2 second. Figure 4.4 shows denoised output when noisy current i_q is given as input. At 1.1 second it can be observed that the meta-denoiser underestimates the signal and a similar error can be observed at 1.8 second.

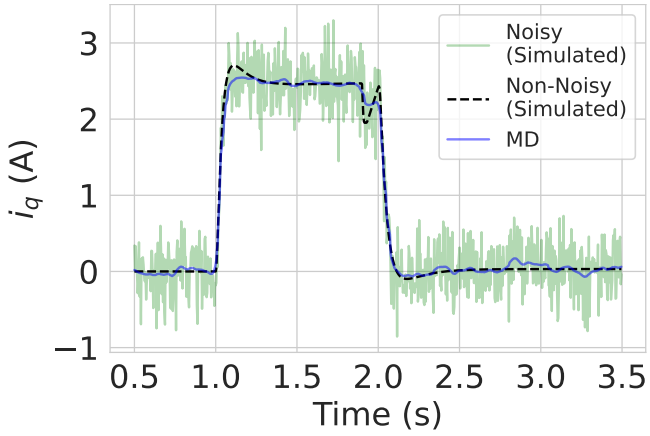


Figure 4.4: Meta-Denoiser on current i_q from Dynamic-Speed1 benchmark.

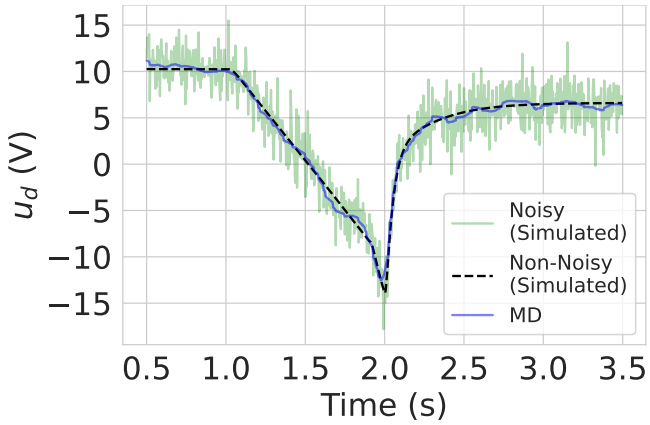


Figure 4.5: Meta-Denoiser on voltage u_d from Dynamic-Speed1 benchmark.

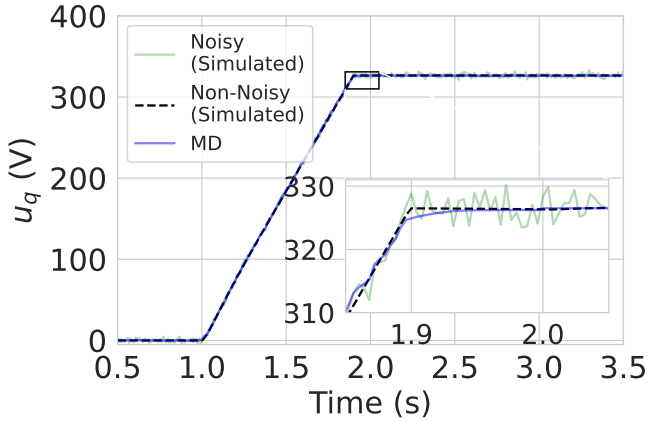


Figure 4.6: Meta-Denoiser on voltage u_q from Dynamic-Speed1 benchmark.

Figure 4.5 shows MD output when noisy voltage u_d is given as input. In this case, no artifacts are present in the prediction when there is a change in the signal, for example, at 1 and 2 seconds. Figure 4.6 shows results when denoising

voltage u_q with meta-denoiser. Around 1.8 second (in the zoomed sub-plot) we can see that the denoised output under-predicts the ramp end. From the above plots of non-noisy / noisy simulated currents and voltages, and MD denoised signals, we get good insights into the MD performance. We observed that MD denoised trajectories are close to non-noisy simulated trajectories, demonstrating the acceptable denoising performance of MD.

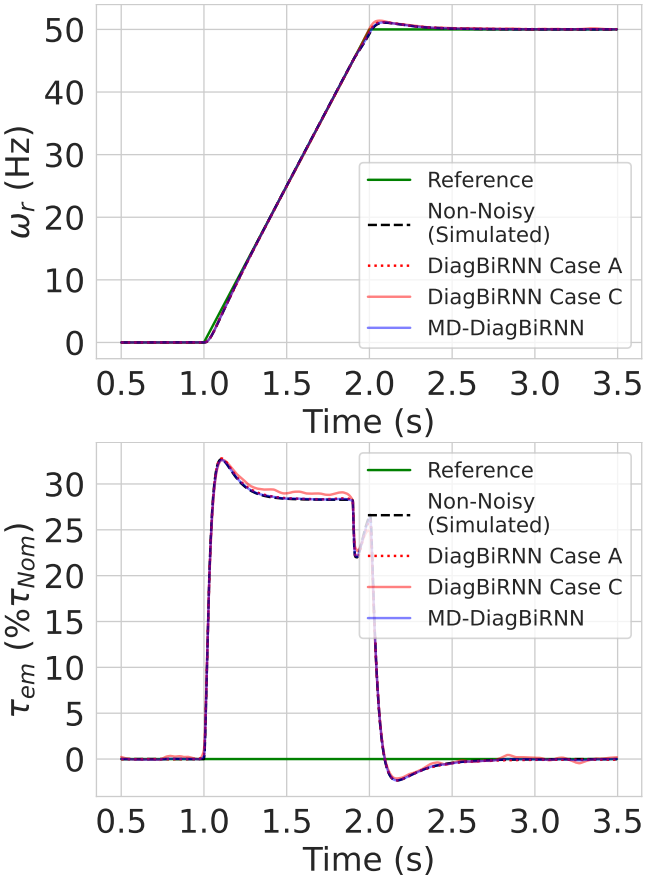


Figure 4.7: Speed and torque estimation from noisy and denoised currents and voltages in Dynamic-Speed1 benchmark.

Figure 4.7 shows predicted speed (top) and torque (bottom), respectively. Results from Case A (applied to non-noisy simulated), Case C (applied to noisy simulated), and MD-DiagBiRNN (applied to MD output currents and voltages) have been shown. For better evaluation, Figure 4.8 shows the difference between the generated outputs and non-noisy simulated. The top sub-figure shows speed errors and the bottom one shows torque errors. While Case C presents substantial errors, it can be seen that MD-DiagBiRNN has errors of the same order of magnitude as in non-noisy Case A. Thus, combining MD and DiagBiRNN allows us to perform a high-quality speed-torque estimation.

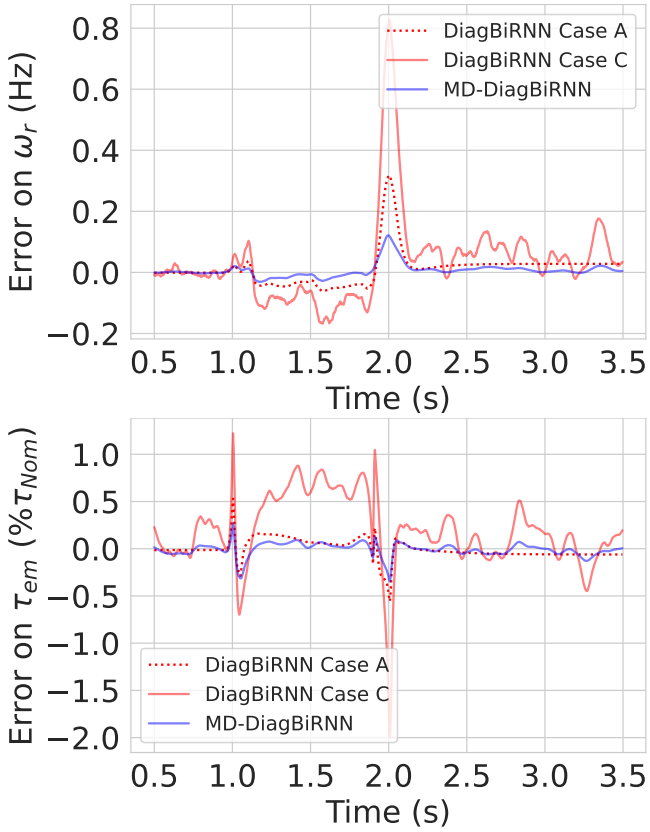


Figure 4.8: Error between estimated and real speed-torque in Dynamic-Speed1 benchmark.

Method	$t_{2\%}$ (ms)	$t_{95\%}$ (ms)	E_{fol} (Hz)	$D\%$ (%)	E_{ss} (Hz)	$\Delta\tau_{max}$ (%τ_{nom})
"Real"	**32.0**	**492**	**0.32**	**3.86**	**0.00**	**65.7**
CASE A	**32.1**	**588**	0.24	5.77	**0.05**	**65.6**
CASE B	32.2	**596**	**0.30**	6.28	**0.18**	65.9
CASE C	32.2	**604**	**0.20**	**7.33**	0.11	**69.8**
EKF[27]	32.2	**616**	**0.19**	**7.23**	0.12	**65.8**
WT[30]	32.2	**604**	**0.20**	**7.27**	0.12	**65.8**
MCTV[33]	**32.1**	**600**	0.22	6.24	0.07	65.5
DAE[34]	**32.9**	**608**	**0.19**	6.21	0.10	66.3
MD (Ours)	**32.1**	**484**	0.26	**4.45**	**0.05**	65.9

Table 4.3: Performance metrics for the predictions performed on simulated Dynamic-Speed2 benchmark

Table 4.3 shows performance metrics for Dynamic-Speed2 benchmark. It can be seen that MD-DiagBiRNN is the closest one to the "Real" for almost all metrics, when estimating speed and torque. Figure 4.9 shows denoised currents (i_d, i_q) from meta-denoiser for Dynamic-Speed2 benchmark.

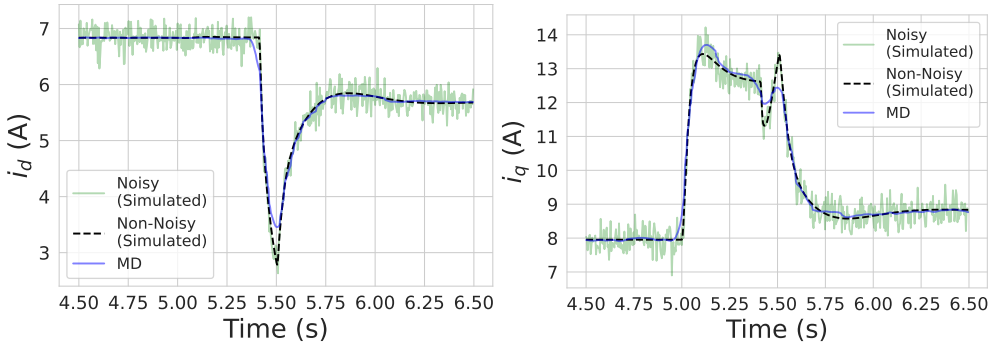


Figure 4.9: Meta-Denoiser on currents (i_d, i_q) from Dynamic-Speed2 benchmark.

In case of current i_d, it can be seen that around 5.4 seconds, at the start of the ramp, the meta-denoised output starts the ramp before the simulated trajectory. Meta-denoiser also cannot reach the same current as the non-noisy simulated one at 5.5 seconds. In case of current i_q, there are three places where meta-denoiser input is degraded, at 5 second just before the start of ramp, around 5.1 second at the first overshoot, around 5.4 second at first undershoot, and at 5.5 second at the second overshoot.

Figure 4.10 shows denoised voltages (u_d, u_q) from meta-denoiser fro Dynamics-Speed2 benchmark. In the case of voltage u_d, we can see no problem at the start of the ramp (5 second) and the overshoot (around 5.8 second). But there is a problem in predicting voltage at 5.5 second. There is a slight averaging effect when there is a sudden change in the voltage. The predictions are very good for all other time stamps and close to the non-noisy simulated voltage u_d. In the case of voltage u_q, we observe that the meta-denoised voltage is very close to the non-noisy simulated voltage. It should be noted that the noise variance is also very low, so it can be considered that the task of denoising is trivial for this particular sample.

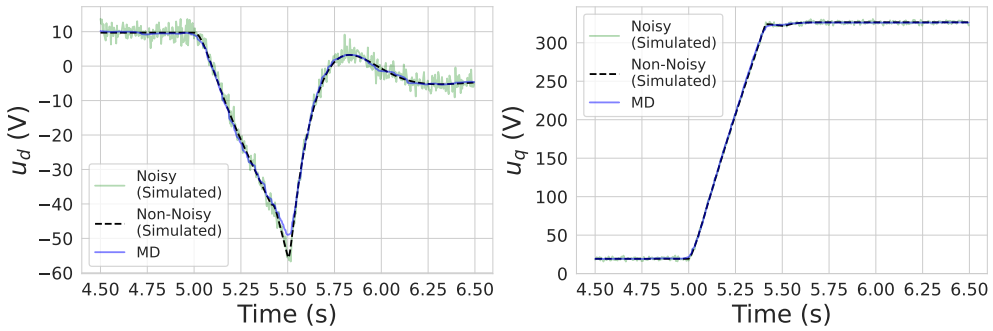


Figure 4.10: Meta-Denoiser on voltages (u_d, u_q) from Dynamic-Speed1 benchmark.

In Figure 4.11, the left sub-figure shows the estimated speed and the right sub-figure shows the estimated torque from noisy and different non-noisy currents

and voltages including meta-denoiser predicted currents and voltages. It can be seen that the prediction of MD DiagBiRNN is very close to non-noisy simulated speed and torque. DiagBiRNN Cases A and C are both performing poorly in the case of speed. DiagBiRNN Case C performs worse in the case of torque, whereas Case A is close to the non-noisy simulated torque.

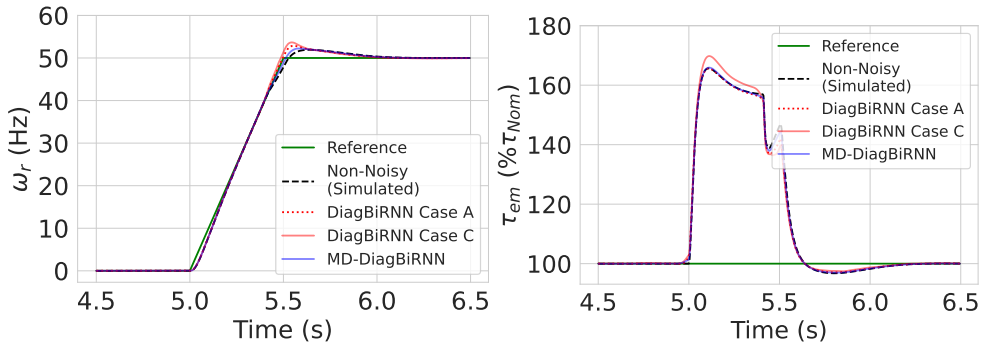


Figure 4.11: Speed-torque estimation from noisy and denoised currents and voltages in Dynamic-Speed2 benchmark.

In Figure 4.12, the left sub-figure shows the error between estimated speeds and the real one, and the right sub-figure shows the error between estimated torque and the real one. The speed error plot shows that at 5.5 second the MD-DiagBiRNN is realitvely better than Cases A and C. In the torque error plot, there is a large error in Case C prediction at the overshoot (5.1 second).

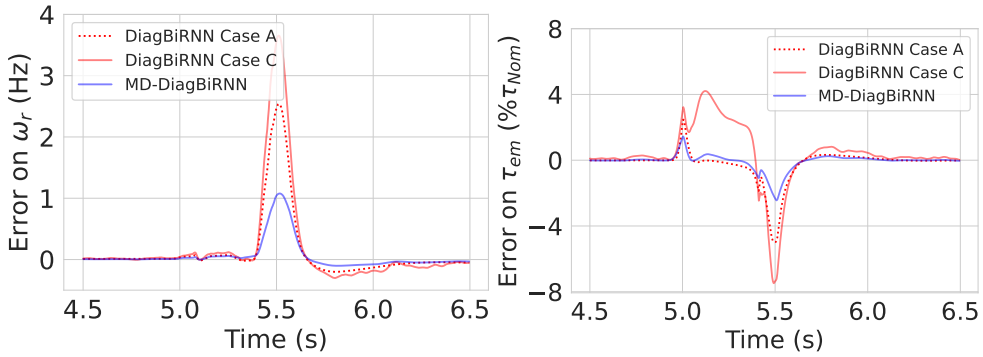


Figure 4.12: Error between estimated and real speed-torque in Dynamic-Speed2 benchmark.

Table 4.4 reports the maximum absolute error between the model predicted speed and the non-noisy simulated speed obtained with the different methods on two quasi-static benchmarks. The lowest error values are reached for non-noisy Case A and for DiagBiRNN associated with NN denoisers, namely DAE, and once again MD.

Method	CASE A	CASE B	CASE C
Quasi-Static1	0.198	0.213	0.227
Quasi-Static2	0.171	0.199	0.204

Method	EKF	WT	MCTV	DAE	MD (Ours)
Quasi-Static1	0.196	0.201	0.198	0.198	**0.197**
Quasi-Static2	0.184	0.189	0.181	0.174	**0.173**

Table 4.4: Maximum absolute error (Hz) for the predictions done on simulated Quasi-Static benchmarks.

Globally, all the results obtained on the simulated benchmarks show that MD-DiagBiRNN outperforms other methods in estimating non-noisy speed and torque from noisy currents and voltages.

4.4.2 . Real Data Benchmarks

The main requirement for the speed-torque estimator is that it operates in real-time with minimum possible delay. The processing time for DiagBiRNN on Nvidia Quadro P620 GPU is 40ms. To avoid adding too much extra delay, it is recommended that denoisers operate in less than 40ms. This is possible in the case of EKF, DAE, and MD. WT and MCTV require larger delay to get acceptable performance, making them hardly usable for real-time usage. Since EKF and DAE do not perform as well as MD on simulated benchmarks, real benchmark experiments are limited to Case D and MD.

Method	Speed ω_r		Torque τ_{em}	
	MAE	**SMAPE**	**MAE**	**SMAPE**
CASE D	0.92	23.7%	1.12	39.2%
MD (Ours)	**0.61**	**13.7%**	**1.09**	**35.2%**

Table 4.5: Aggregated ML metrics for the predictions done on all real data benchmarks.

Table 4.5 shows the aggregated ML metrics on the dynamic and quasi-static real benchmarks obtained by DiagBiRNN with MD and in Case D. It confirms that DiagBiRNN-MD performs very well compared to Case D owing to its denoising capability.

Method	$t_{2\%}$ (ms)	$t_{95\%}$ (ms)	E_{fol} (Hz)	$D\%$ (%)	E_{ss} (Hz)	$\Delta\tau_{max}$ (%τ_{nom})
"Motor Real"	**31**	**1118**	**0.49**	**1.44**	**0.00**	**359.0**
CASE D	36	1134	0.83	0.78	0.48	362.7
MD (Ours)	30	1133	0.69	0.52	0.45	355.4

Table 4.6: Performance metrics for the online inference on RDynamic-Speed1 benchmark for 1.5kW motor.

Table 4.6 shows the electrical engineering metrics for the real-time inference on RDynamic-Speed1 benchmark. It should be pointed out that, for consistency, electrical engineering metrics for Case D are computed on smooth reconstruction of Case D predictions. Moreover, "Motor Real" stands for "non-noisy (reconstructed)", which has been obtained manually by doing an a posteriori approximation of the non-noisy trajectory. It can be seen that electrical engineering metrics on both Case D and DiagBiRNN-MD predictions are close to real data metrics, DiagBiRNN-MD being even a little closer.

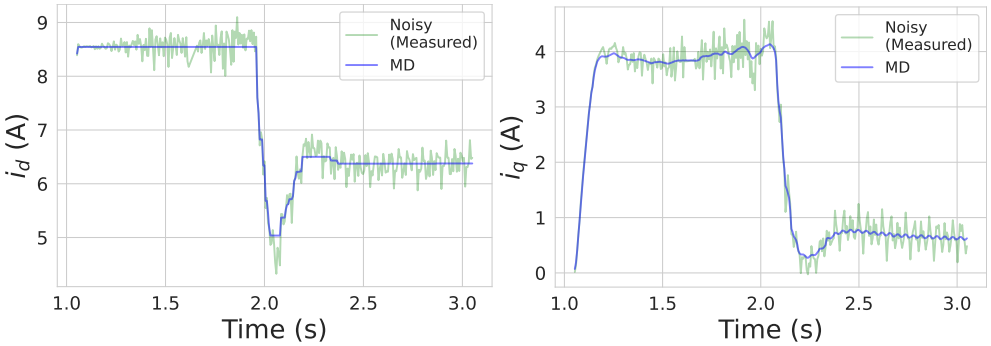


Figure 4.13: Meta-Denoiser on currents (i_d, i_q) for RDynamic-Speed1 benchmark.

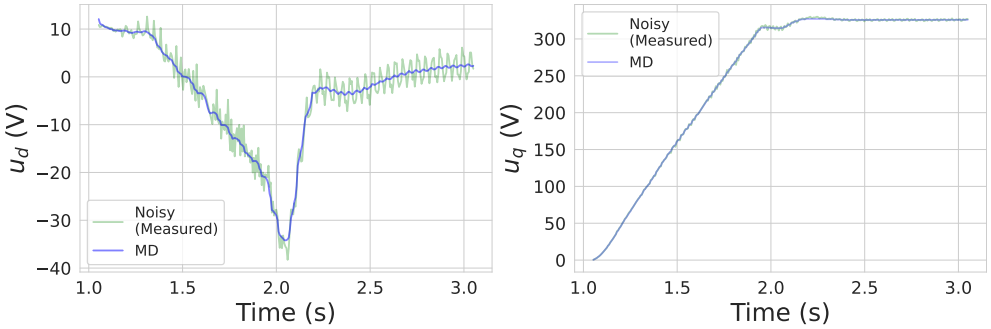


Figure 4.14: Meta-Denoiser (MD) denoised u_q for RDynamic-Speed1 benchmark.

Figure 4.13 shows denoised currents (i_d, i_q) for RDynamic-Speed1 benchmark. Figure 4.14 shows denoised voltages (u_d, u_q). Real noisy currents and voltages as well as MD denoised signals have been plotted. It can be observed that MD denoised trajectories are close to noisy trajectories, demonstrating the good denoising performance of MD.

Figure 4.15 shows speed-torque estimation from Case D and MD-DiagBiRNN, while Figure 4.16 shows speed and torque prediction errors with respect to non-noisy (reconstructed) real speed and torque. Unlike DiagBiRNN-MD predictions, both speed and torque predictions for Case D are corrupted with a level of noise comparable to the noisy (measured) real data. This is confirmed by Figure 4.15,

which shows that case D predictions are pretty good in average but highly varying, while DiagBiRNN-MD predictions are not only clean but also close to the non-noisy (reconstructed) real signals.

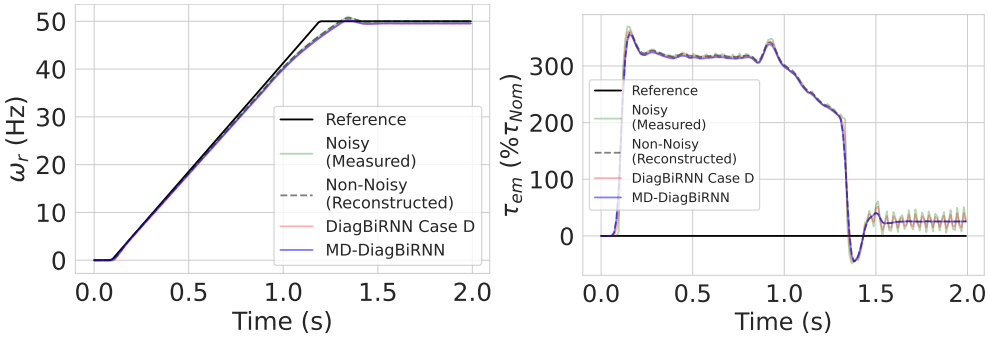


Figure 4.15: Online estimated speed and torque from noisy and denoised currents and voltages from RDynamic-Speed1 real data benchmark (on 1.5kW motor).

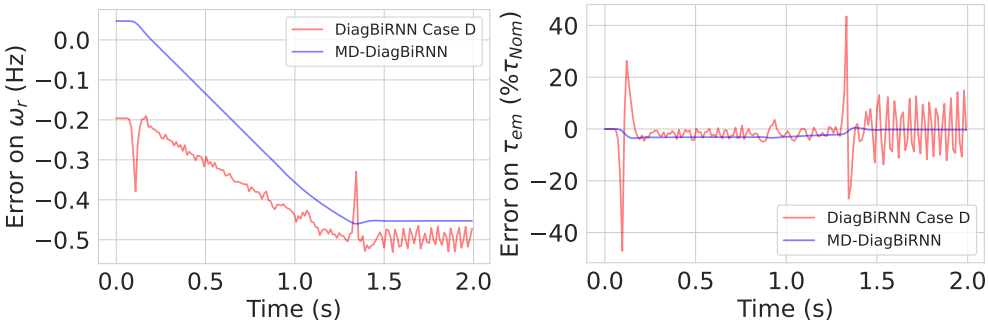


Figure 4.16: Error between estimated and real speed-torque from RDynamic-Speed1 real data benchmark (on 1.5kW motor).

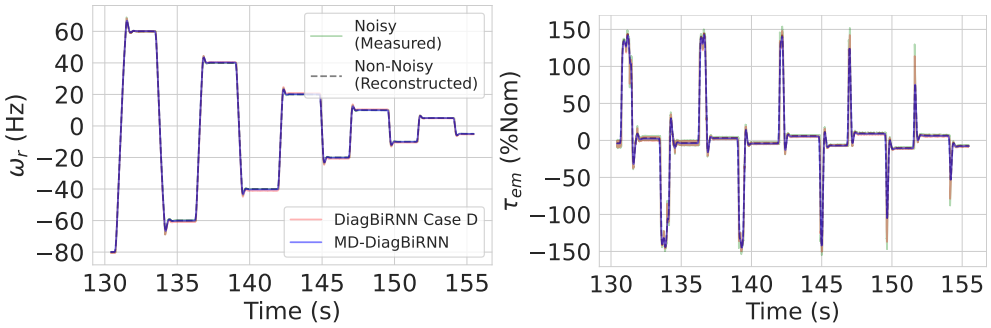


Figure 4.17: Online estimated speed-torque from noisy and denoised currents and voltages RDynamic-Speed2 real data benchmark.

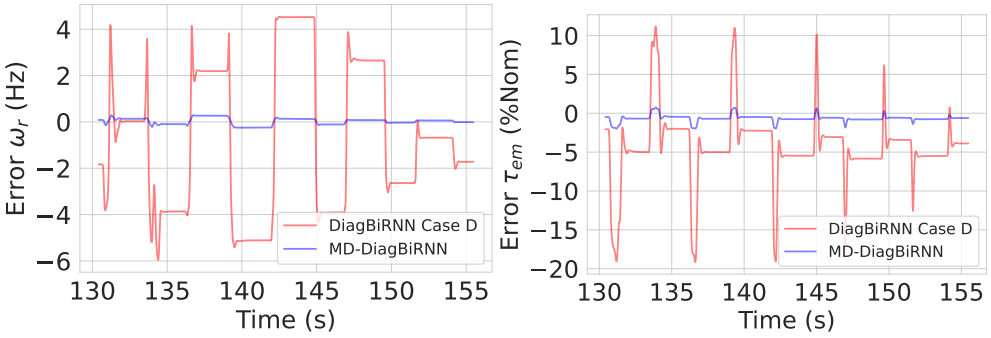


Figure 4.18: Error between estimated and real speed-torque from RDynamic-Speed2 benchmark.

Figure 4.17 shows speed-torque estimation from Case D and MD-DiagBiRNN, while Figure 4.18 shows speed and torque prediction errors with respect to non-noisy (reconstructed) real speed and torque. It can be observed that the MD-DiagBiRNN performs significantly better than case D predictions on such a challenging benchmark.

Method	$t_{95\%}$ (ms)	$D\%$ (%)	E_{ss} (%τ_{nom})	SD (Hz)
"Motor Real"	**370**	**12.8**	**-0.4**	**3.41**
CASE D	373	13.2	-0.2	3.56
MD (Ours)	372	12.9	-0.3	3.45

Table 4.7: Performance metrics for the predictions done on real data RDynamic-Torque benchmark

In Table 4.7, we report electrical engineering metrics for RDynamic-Torque benchmark. It can be observed that, as for the previous benchmark, electrical engineering metrics of both DiagBiRNN-MD and smoothed Case D predictions are very close to real data metrics. Figure 4.19 shows currents and Figure 4.20 shows voltages for RDynamic-Torque benchmark.

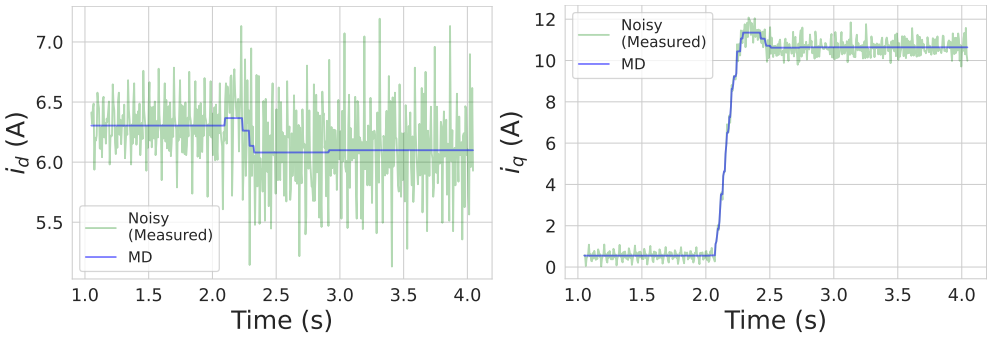


Figure 4.19: Meta-Denoiser on currents (i_d, i_q) for RDynamic-Torque benchmark.

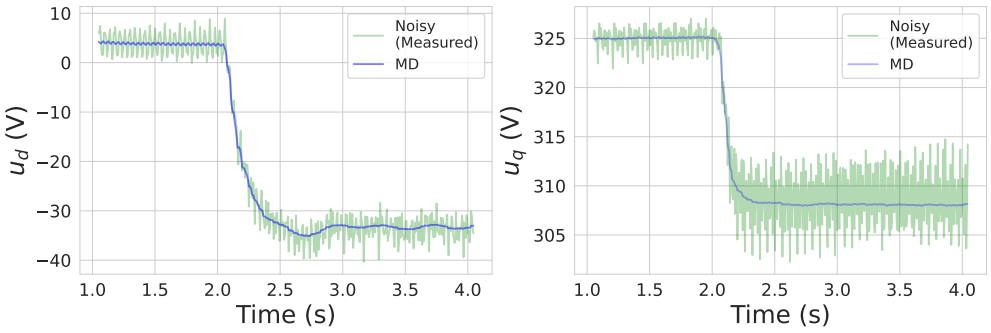


Figure 4.20: Meta-Denoiser on voltages (u_d, u_q) for RDynamic-Torque benchmark.

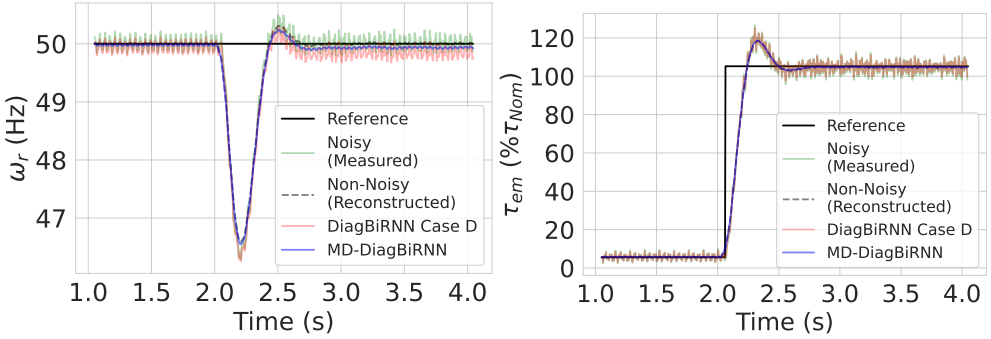


Figure 4.21: Online estimated speed-torque from noisy and denoised currents and voltages from RDynamic-Torque benchmark.

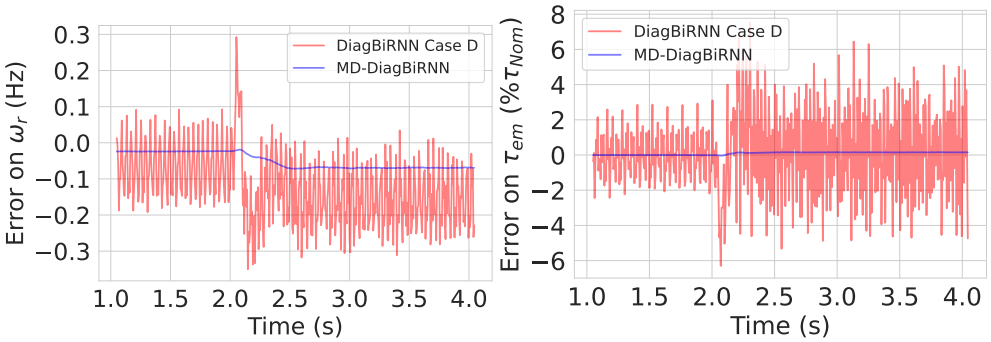


Figure 4.22: Error between estimated and real speed-torque from RDynamic-Torque benchmark.

Figure 4.21 shows speed-torque estimation from Case D and MD-DiagBiRNN, while Figure 4.22 shows speed and torque prediction errors with respect to non-noisy (reconstructed) real speed and torque. Case D predictions are still corrupted with a significant amount of noise, whereas DiagBiRNN-MD prediction error is close to 0.

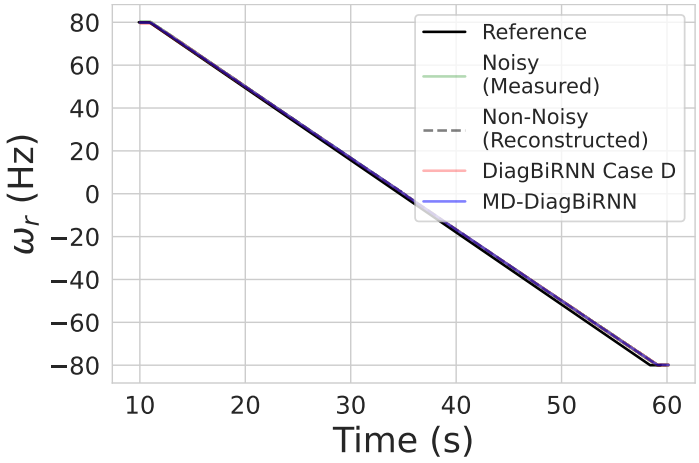


Figure 4.23: Real noisy signal, non-noisy reconstructed, and desnoised rotor speeds from RQuasi-Static benchmark.

Figures 4.23 and 4.24 display results on RQuasi-Static benchmark for Case D and DiagBiRNN-MD. As for the dynamic benchmarks, both methods perform well in average but with some noticeable noise for Case D prediction. The difference between prediction with DiagBiRNN-MD and non-noisy (reconstructed) real speed is never greater than 0.1Hz, which is very good.

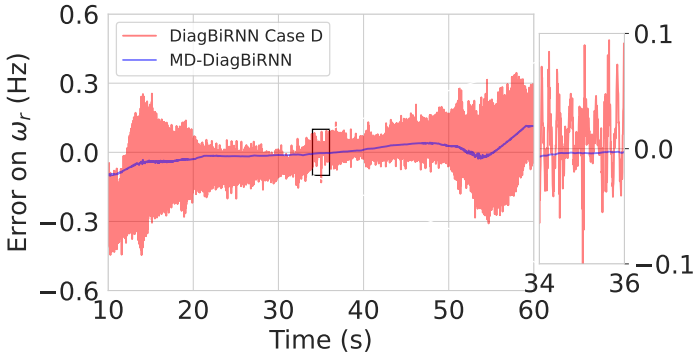


Figure 4.24: Error between denoised and real speeds from RQuasi-Static benchmark.

All the results obtained on the simulated and real benchmarks show that MD-DiagBiRNN outperforms other methods in estimating non-noisy speed and torque from noisy currents and voltages. Results on the four real data benchmarks show that both Case D and DiagBiRNN-MD provide significantly better speed and torque prediction. Case D predicts noisy speed and torque thanks to the fact that it is fine-tuned on train set of real data. DiagBiRNN-MD predicts speed and torque very close to the ideal (reconstructed) signals owing to the good denoising performance of the proposed MD technique. MD outperforms all other methods since it is not biased to static behaviours in the input noisy signals. Other methods perform relatively poorly in dynamic parts. This justifies the need for MD to be trained to

perform segmentation and denoising simultaneously.

4.5 . Robustness of Neural Networks for Electrical Motor Tasks

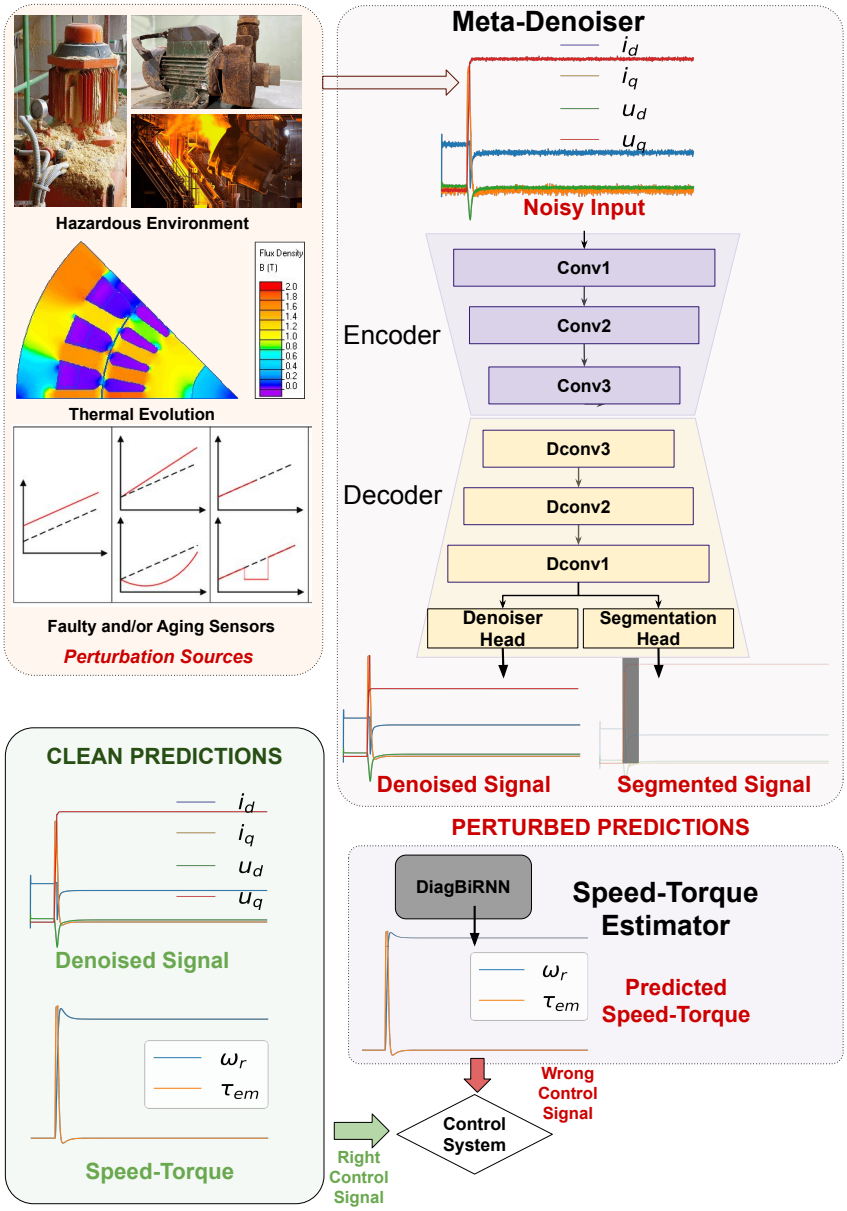


Figure 4.25: Different sources of perturbations in signals that are input to neural networks used in electrical motor tasks.

Figure 4.25 shows how naturally occurring perturbation in inputs can cause a neural network to deliver the wrong output. When such neural networks are cascaded to drive a control system, it can lead to catastrophic failures. In this case, the meta-denoiser and speed-torque estimator together take denoised currents and

voltages and estimate speed and torque, which can then be used to drive the control system. Currents and voltages are measured using sensors that can be affected by extreme heat, water, dust, or material faults. Sensors can also perform poorly due to aging. Motors can age and get affected due to their environment and will behave differently. Given that meta-denoiser and DiagBiRNN have not been trained on a dataset that considers such varied inputs and motor states, the networks are bound to give wrong predictions, which can affect the downstream control task.

4.5.1 . Datasets

Datasets proposed in Chapter 3 for **motor dynamics** input-output modeling, **denoise**, **speed-torque** estimation, **temperature** modeling, and **broken bar** tasks have been used. In case of **motor dynamics**, inputs are voltages (u_d, u_q) and rotor speed (ω_r). The quantities that have to be predicted are currents (i_d, i_q) and mechanical torque (τ_{em}). The **denoise** problem deals with denoising noisy currents ($\hat{i_d}$, $\hat{i_q}$), noisy voltages ($\hat{u_d}$, $\hat{u_q}$). For the **speed-torque** estimation, the goal is to predict rotor speed (ω_r), and mechanical torque (τ_{em}) from currents (i_d, i_q) and voltages (u_d, u_q).

In **temperature** dataset, currents (i_d, i_q), voltages (u_d, u_q), speed (ω_r), and torque (τ_{em}) are the electrical motor quantities. Permanent magnet (ϑ_{PM}), stator yoke (ϑ_{SY}), stator tooth (ϑ_{ST}), stator winding (ϑ_{SW}), ambient temperature outside of stator (ϑ_a), and coolant temperature (ϑ_c) are the recorded temperatures. The objective is to predict permanent magnet (ϑ_{PM}) temperature from all other quantities, which makes this a regression task.

Broken bars dataset consists of currents, voltages, and torque as electrical signals. Accelerometers placed in 5 different parts of motor is used to collect vibrations/mechanical signals. In total 400 experiments of each lasting 20 seconds is performed. The objective is to predict how many broken bars are in the motor from its electrical and mechanical signals.

4.5.2 . Model Architectures

The architectures proposed in this thesis have been used for **motor dynamics learning**, **denoise**, and **speed-torque estimation** tasks. Networks named FNN, RNN, LSTM, and CNN are used for baselines. Encoder-decoder variants named Deep, Skip, RNN-Skip, BiRNN-Skip, and DiagBiRNN are used to understand robustness behavior with respect to network complexity. For the **temperature** task DiagBiRNN and FedFormer [156] are used. For the **broken bars** task, 1D variant of three classification networks namely CRNN [157], ResNet-18 [158], and RegNet-20 [159] are used.

For generating adversarial attacks, we use FGSM and DeepFool. The value of ℓ_2 perturbations within an ϵ-neighborhood of 0.01 and 0.1 is taken. In the case of DeepFool, the number of iterations used is 100. For the **broken bars** task, we report clean accuracy, and FGSM and DeepFool attack accuracy. For all

the regression tasks, we report clean and FGSM attack values of Mean Absolute Error (MAE), Symmetric Mean Absolute Percentage Error (SMAPE), coefficient of determination (R^2) [90], and Root Mean Squared Error (RMSE).

4.5.3 . Motor Dynamics Learning Networks

Network	Attack	MAE	SMAPE(%)	R^2	RMSE
FNN	**Clean**	0.03	5.97	0.72	0.06
	FGSM $\epsilon = 0.01$	0.12	24.06	-1.57	0.17
	FGSM $\epsilon = 0.1$	0.82	66.11	-130.99	1.23
CNN	**Clean**	**0.02**	**4.86**	**0.76**	**0.05**
	FGSM $\epsilon = 0.01$	0.11	22.02	-1.05	0.15
	FGSM$\epsilon = 0.1$	0.56	94.63	-45.14	0.73
RNN	**Clean**	0.03	6.33	0.73	0.06
	FGSM $\epsilon = 0.01$	**0.11**	**21.91**	**-0.77**	**0.14**
	FGSM $\epsilon = 0.1$	**0.22**	**45.53**	**-6.99**	**0.30**
LSTM	**Clean**	0.03	5.05	0.75	0.05
	FGSM $\epsilon = 0.01$	0.12	24.73	-1.90	0.18
	FGSM $\epsilon = 0.1$	0.39	65.19	-32.92	0.63
Deep	**Clean**	0.026	5.15	0.73	0.06
	FGSM $\epsilon = 0.01$	0.11	22.89	-1.02	0.15
	FGSM $\epsilon = 0.1$	**0.15**	**32.11**	**-2.90**	**0.21**
Skip	**Clean**	**0.02**	**4.62**	**0.77**	**0.05**
	FGSM $\epsilon = 0.01$	0.11	24.94	-1.14	0.16
	FGSM $\epsilon = 0.1$	0.39	94.92	-21.99	0.52
RNN-Skip	**Clean**	0.03	6.73	0.72	0.06
	FGSM $\epsilon = 0.01$	**0.09**	**20.36**	**-0.54**	**0.13**
	FGSM $\epsilon = 0.1$	0.29	73.5	-10.87	0.37
BiRNN-Skip	**Clean**	0.03	6.82	0.72	0.06
	FGSM $\epsilon = 0.01$	0.13	26.09	-1.85	0.18
	FGSM $\epsilon = 0.1$	0.69	76.03	-84.77	0.99
DiagBiRNN	**Clean**	0.02	4.70	0.76	0.05
	FGSM $\epsilon = 0.01$	0.10	21.31	-0.83	0.15
	FGSM $\epsilon = 0.1$	0.32	58.40	-16.23	0.44

Table 4.8: Metrics of clean and adversarial predictions from all the networks trained for **motor dynamics** task.

Table 4.8 shows results obtained by networks for **motor dynamics learning** task. It shows MAE, SMAPE, R^2, and RMSE for clean predictions and FGSM predictions at $\epsilon = 0.01$ and $\epsilon = 0.1$. Key observation is that all networks predictions degrades when attacked. CNN variant achieves the best clean predictions MAE (0.02), SMAPE (4.86%), and R^2 (0.76). When attacked, RNN is most robust at both epsilon values. Skip encoder-decoder variant achieves the best clean predictions SMAPE (4.62%) and R^2 (0.77). When attacked, RNN-Skip is most robust at $\epsilon = 0.01$ with SMAPE (20.36%), followed by DiagBiRNN with SMAPE (21.31%). Skip achieves the second worst performance when attacked.

Figure 4.26 shows an example trajectory from the validation set of the **motor dynamics** dataset. In the top sub-figure normalized values of u_d, u_q, and ω_r are showed as normal lines. FGSM is used with $\epsilon = 0.01$ to attack the DiagBiRNN

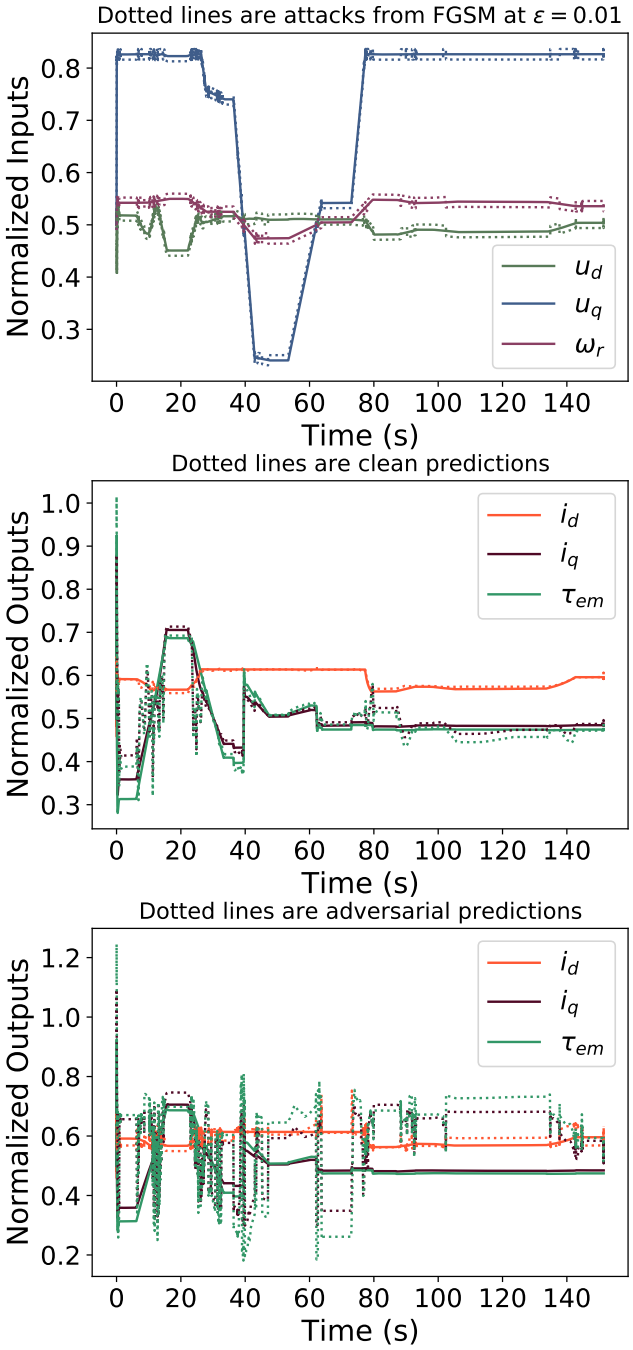


Figure 4.26: A sample from validation set of **motor dynamics** task showing clean input, clean output, DiagBiRNN clean prediction, FGSM generated adversarial example, and adversarial prediction.

network by generating an adversarial example shown using the dotted lines in the same sub-figure. It can be seen that the adversarial example has an offset when compared to the clean input, but this offset can be positive or negative. There

is also intermittent noise like big offsets, for example, one at 90 second. The middle sub-figure shows the ground truth of output signals i_d, i_q, and τ_{em} using normal lines and DiagBiRNN predictions using dotted lines. It can be seen that the network has poor predictions during the first ramp between 5s and 20s. The bottom figure shows ground truth as normal lines and dotted lines are the predictions of DiagBiRNN when the generated adversarial example is given as the input. It can be seen that the adversarial predictions are very noisy compared to clean predictions when the perturbations in the input are minimal. This shows that the attacks on the networks are strong and establishes that a robustness study of such networks is important.

4.5.4 . Motor Denoising Networks

Network	Attack	MAE	SMAPE(%)	R^2	RMSE
FNN	**Clean**	0.01	1.16	0.99	0.01
	FGSM $\epsilon = 0.01$	0.01	2.52	0.99	0.02
	FGSM $\epsilon = 0.1$	0.07	13.98	0.55	0.09
CNN	**Clean**	0.00	0.11	0.99	0.00
	FGSM $\epsilon = 0.01$	0.01	1.86	0.99	0.01
	FGSM $\epsilon = 0.1$	0.07	13.88	0.62	0.09
RNN	**Clean**	0.00	0.26	0.99	0.00
	FGSM $\epsilon = 0.01$	0.01	1.61	1.00	0.01
	FGSM $\epsilon = 0.1$	0.07	15.11	0.71	0.08
LSTM	**Clean**	0.00	0.26	0.99	0.00
	FGSM $\epsilon = 0.01$	0.01	1.71	1.00	0.01
	FGSM $\epsilon = 0.1$	0.07	15.10	0.71	0.08
Deep	**Clean**	0.00	0.22	0.99	0.00
	FGSM $\epsilon = 0.01$	0.01	1.63	0.99	0.01
	FGSM $\epsilon = 0.1$	0.06	13.95	0.58	0.09
Skip	**Clean**	0.00	0.18	0.99	0.00
	FGSM $\epsilon = 0.01$	0.01	1.59	1.00	0.01
	FGSM $\epsilon = 0.1$	0.07	15.25	0.67	0.08
RNN-Skip	**Clean**	**0.00**	**0.14**	**0.99**	**0.00**
	FGSM $\epsilon = 0.01$	0.01	1.76	1.00	0.01
	FGSM $\epsilon = 0.1$	0.08	16.16	0.66	0.08
BiRNN-Skip	**Clean**	0.00	0.15	0.99	0.00
	FGSM $\epsilon = 0.01$	0.01	1.59	1.00	0.01
	FGSM $\epsilon = 0.1$	0.06	14.04	0.71	0.08
DiagBiRNN	**Clean**	0.00	0.18	0.99	0.00
	FGSM $\epsilon = 0.01$	**0.01**	**1.53**	**1.00**	**0.01**
	FGSM $\epsilon = 0.1$	**0.07**	**13.6**	**0.72**	**0.08**

Table 4.9: Metrics of clean and adversarial predictions from all the networks trained for motor **denoise** task.

Table 4.9 shows results of the networks trained for **denoise** task. RNN-Skip shows best SMAPE (0.14%) among all the variants. R^2 is 0.99 for all the networks. But when attacked with FGSM at $\epsilon = 0.01$ DiagBiRNN outperforms every other network with lowest SMAPE (1.53%). With a more aggressive attack of $\epsilon = 0.1$ DiagBiRNN still outperforms every other network with SMAPE (13.6%) and R^2 (0.72).

4.5.5 . Speed-Torque Estimation Networks

Network	Attack	MAE	SMAPE(%)	R^2	RMSE
FNN	**Clean**	0.00	0.76	0.99	0.01
	FGSM $\epsilon = 0.01$	0.02	3.46	0.95	0.02
	FGSM $\epsilon = 0.1$	0.11	22.65	-1.55	0.15
CNN	**Clean**	0.00	0.17	0.99	0.00
	FGSM $\epsilon = 0.01$	0.02	3.67	0.94	0.02
	FGSM $\epsilon = 0.1$	0.12	27.10	-2.09	0.17
RNN	**Clean**	0.00	0.25	0.99	0.00
	FGSM $\epsilon = 0.01$	0.01	2.67	0.97	0.02
	FGSM $\epsilon = 0.1$	0.10	24.26	-0.76	0.13
LSTM	**Clean**	0.00	0.22	0.99	0.00
	FGSM $\epsilon = 0.01$	0.01	2.55	0.97	0.02
	FGSM $\epsilon = 0.1$	0.09	21.74	-0.97	0.14
Deep	**Clean**	0.00	0.3	0.99	0.00
	FGSM $\epsilon = 0.01$	0.01	2.94	0.96	0.02
	FGSM $\epsilon = 0.1$	0.096	24.42	-0.87	0.13
Skip	**Clean**	**0.00**	**0.18**	**0.99**	**0.00**
	FGSM $\epsilon = 0.01$	0.01	2.83	0.97	0.02
	FGSM $\epsilon = 0.1$	0.10	23.99	-0.80	0.13
RNN-Skip	**Clean**	0.00	0.82	0.99	0.00
	FGSM $\epsilon = 0.01$	0.01	3.13	0.96	0.02
	FGSM $\epsilon = 0.1$	**0.10**	**17.96**	**-0.56**	**0.12**
BiRNN-Skip	**Clean**	0.00	0.68	0.99	0.00
	FGSM $\epsilon = 0.01$	0.01	2.91	0.97	0.02
	FGSM $\epsilon = 0.1$	0.10	24.84	-0.67	0.12
DiagBiRNN	**Clean**	0.00	0.26	0.99	0.00
	FGSM $\epsilon = 0.01$	**0.01**	**2.70**	**0.97**	**0.02**
	FGSM $\epsilon = 0.1$	0.10	20.11	-0.49	0.12

Table 4.10: Metrics of clean and adversarial predictions from all the networks trained for **speed-torque** estimation task.

Table 4.10 shows results of the networks trained for **speed-torque** estimation task. Encoder-decoder variant Skip shows best SMAPE (0.18%) among all the variants. R^2 is 0.99 for all the networks. But when attacked with FGSM at $\epsilon = 0.01$ DiagBiRNN outperforms every other network with lowest SMAPE (2.70%). With a more aggressive attack of $\epsilon = 0.1$ RNN-Skip outperforms every other network with SMAPE (17.96%) and R^2 (-0.56). DiagBiRNN achieves best R^2 of -0.49.

4.5.6 . Temperature Estimation Networks

Table 4.11 shows results for FedFormer and DiagBiRNN trained to do permanent magnet **temperature** prediction task. In this case, it can be seen that FedFormer has performed very poorly when compared to DiagBiRNN in terms of clean predictions. However, when attacked with FGSM at $\epsilon = 0.01$, DiagBiRNN gets degraded substantially while FedFormer shows slight degradation with respect to its poor performance.

Network	Attack	MAE	SMAPE(%)	R^2	RMSE
FedFormer	**Clean**	0.03	6.47	0.96	0.04
	FGSM	0.03	**7.53**	0.95	0.05
DiagBiRNN	**Clean**	**0.00**	**0.82**	**1.00**	**0.01**
	FGSM	0.03	7.73	**0.97**	**0.04**

Table 4.11: Metrics for clean and adversarial predictions for the two networks trained for permanent magnet **temperature** prediction task.

4.5.7 . Fault Detection Networks

ϵ	Method	Clean(%)	FGSM(%)	DeepFool(%)
0.01	**CRNN**	80.0	79.0	79.0
	ResNet	90.0	89.0	89.0
	RegNet	**93.0**	**92.0**	**92.0**
0.1	**CRNN**	79.4	61.4	60.1
	ResNet	88.9	80.0	79.6
	RegNet	**92.3**	**86.8**	**86.6**

Table 4.12: Classification results for **broken bars** task.

Table 4.12 shows results obtained by CRNN, ResNet, and RegNet on **broken bars** task. In this case RegNet gives best clean accuracy. When attacked with FGSM and DeepFool at $\epsilon = 0.01$ and $\epsilon = 0.1$ the accuracy of all three networks decrease but the order remains same. At $\epsilon = 0.1$, RegNet is still more robust compared to other two networks.

4.6 . Generalization of Speed-Torque Estimator

One of the experiments in Section 4.4.1 uses the speed-torque estimator trained on non-noisy simulated data. This network has been trained on 4kW simulated data generated in Section 2.5.2. We use Simulink model of 90kW motor to generate exactly the same set of five benchmarks: **Dynamic-Speed1**, **Dynamic-Speed2**, **Dynamic-Torque**, **Quasi-Static1**, and **Quasi-Static2**. It should be noted that the nominal torque of 4kW and 90kW motors are quite different (25Nm and 580Nm, respectively). We use the network trained on 4kW motor simulated data to estimate speed-torque for the 90kW benchmarks.

In Table 4.6, we saw that the estimator results are very good when the speed-torque estimator trained on 4kW motor data is applied on a 1.5kW powered motor data. Table 4.13 shows the machine learning metrics obtained for 4kW (from Table 2.6) and 90kW motor benchmarks. The table shows that the speed-torque estimator predicts torque for the 90kW motor with a similar MAE and SMAPE as the 4kW motor for any given benchmark. This is not the case for

Power	4kW				90kW			
Model	Speed (ω_r)		Torque (τ_{em})		Speed (ω_r)		Torque (τ_{em})	
	MAE	SMAPE	MAE	SMAPE	MAE	SMAPE	MAE	SMAPE
Dynamic-Speed1	0.05	40.12%	0.74	151.13%	1.49	83.63%	0.57	95.08%
Dynamic-Speed2	0.09	2.05%	2.03	4.61%	7.95	36.33%	1.64	2.74%
Dynamic-Torque	0.03	0.11%	3.18	70.32%	1.88	7.87%	0.62	67.82%
Quasi-Static1	0.08	0.16%	0.70	67.20%	1.82	6.29%	0.81	32.28%
Quasi-Static2	0.04	0.31%	1.78	3.66%	1.81	7.61%	1.05	4.21%

Table 4.13: ML metrics for DiagBiRNN speed-torque estimator networks on 4kW and 90kW benchmarks.

speed estimations. The 90kW motor results clearly show a big degradation in the estimation performance. This evidences that care must be taken in order to generalize the network so as to handle such a high-powered motor.

4.7 . Summary

We have developed a data-driven approach for estimating an induction motor speed and torque from measured currents and voltages. This method allows us to bridge between data simulated from a physical model and real-world ones. We showed, however, that standard techniques for learning the underlying dynamical model are prone to errors in the presence of measurement noise. To overcome this, we proposed a novel meta-denoiser (MD) method that removes noise from currents and voltages before feeding them to our speed-torque estimator (DiagBiRNN). We showed that the proposed approach performs very well on real data benchmarks.

A robustness analysis of neural networks used in five different electrical motor tasks has been performed. A wide array of networks are trained and perturbed by adversarial attackers to evaluate their stability. The experiments show that networks are somewhat unstable to the input perturbations. It would be interesting to generate more sophisticated perturbations that consider domain knowledge of electrical motors. It is also important to understand the sensitivity of the individual inputs with respect to the neural network stability.

Ultimately, we briefly conduct a generalization study of the speed-torque estimator trained on the 4kW motor data by performing prediction tasks on 90kW motor data. We observe a significant error in the speed estimation, leading to the conclusion that further investigation of a proper generalization mechanism is required.

Chapter 5
Neural Network Compression

5.1 . Introduction

Deep neural networks have evolved to the state-of-the-art techniques in a wide array of applications: computer vision [160, 161, 162], automatic speech recognition [163, 164, 165, 166, 167, 168], natural language processing [169, 170, 171, 172], and time series forecasting [173]. While their performance in various applications has matched and often exceeded human capabilities, neural networks may remain difficult to apply in real-world scenarios. Deep neural networks leverage the power of Graphical Processing Units (GPUs), which are power-hungry. Using GPUs to make billions of predictions daily thus comes with a substantial energy cost. In addition, despite their fast response time, deep neural networks are not yet suitable for most real-time applications where memory-limited low-cost architectures need to be used. Compression and efficiency have become a topic of high interest in the deep learning community for all those reasons.

Sparsity in DNNs has been an active research topic generating numerous approaches. DNNs achieving state-of-the-art in a given problem usually have many layers with non-uniform parameter distribution across layers. Most sparsification methods are based on a global approach, which may result in sub-optimal compression for reduced accuracy. This may occur because layers with smaller parameters may remain dense, although they may contribute more to computational complexity (e.g., for convolutional layers). Some methods, known as magnitude pruning, use hard or soft thresholding to remove less significant parameters. Soft thresholding techniques achieve a good sparsity-accuracy trade-off at the cost of additional parameters and increased computation time during training [174]. Searching for a hardware-efficient network is another area that has been proven quite useful, but it requires massive computational resources. Convex optimization techniques such as those used in [175] often rely upon fixed point iterations that use the proximity operator [176]. The related concepts are fundamental for tackling nonlinear problems and have recently come into play in analyzing neural networks [177] and nonlinear systems [178].

5.2 . Related Work

5.2.1 . Inducing sparsity post training

Methods operating on a pre-trained network take multiple pruning and fine-tuning cycles to achieve desired sparsity and accuracy are reached [43, 44, 45, 46, 47, 48, 49]. [50] proposed weight rewinding technique instead of vanilla fine-tuning post-pruning. Net-Trim algorithm [175] removes connections at each layer of a trained network by convex programming. The proposed method works for networks using rectified linear units (ReLUs). Lowering rank of parameter tensors [179, 180, 181], removing channels, filters and inducing group sparsity [182, 183, 184, 185, 186] are some methods that take network structure into account. All these methods rely on pruning and fine-tuning cycle(s), often from full training data.

5.2.2 . Inducing sparsity during training

Another popular approach has been to induce sparsity during training. This is achieved by modifying the loss function to consider sparsity as part of the optimization [51, 52, 53, 54]. Bayesian priors [187], L_0, L_1 regularization [188], and variational dropout [47] get accuracy comparable to [189] but at a cost of $2\times$ memory and $4\times$ computations during training. [190, 191, 174, 192, 193, 194] have proposed learnable sparsity methods through training of the sparse masks and weights simultaneously with minimal heuristics. Although these methods are cheaper than pruning after training, they need at least the same computational effort as training a dense network to find a sparse sub-network. This makes them expensive when compressing big networks where the number of parameters ranges from hundreds of millions to billions [171, 165, 172]. Methods like [189, 195, 196, 197, 198] can be sub-classified as methods where dynamic pruning is performed during training by observing the network flow. [199, 200, 201] computes weight magnitude and reallocates weights at every step of model training.

5.2.3 . Training sparsely initialized networks

[55] showed that it is possible to find sparse sub-networks that, when trained from scratch, were able to match or even outperform their dense counterparts. [56] presented SNIP, a method to estimate, at initialization, the importance that each weight could have later during training. In [57] the authors perform a theoretical study of pruning at initialization from a signal propagation perspective, focusing on the initialization scheme. Recently, [202] proposed GraSP, a different method based on the gradient norm after pruning, and showed a significant improvement for moderate levels of sparsity. [203] starts with a small subnetwork and progressively grow it to a subnetwork that is as accurate as its dense counterpart. [204] proposes SynFlow, which avoids a pruned network's flow collapse during training. [205] proposed FORCE, an iterative pruning method

that progressively removes a small number of weights. This method can achieve extreme sparsity at little accuracy expense. These methods are not usable for big pre-trained networks and are expensive as multiple training rounds are required for different sparse models depending on deployment scenarios (computing devices).

5.2.4 . Neural Architecture Search and Auto ML

Many works on searching models with reinforcement learning and genetic algorithms [206, 207, 208, 209] greatly improve the performance of neural networks. Neural Architecture Search (NAS) [210] aims to search the transferable network blocks, and its performance surpasses many manually designed architectures. Cai et al. [211] proposed to speed up the exploration via network transformation [212]. Inspired by them, N2N [213] integrated reinforcement learning into channel selection. [214] presents an Auto ML pipeline for model compression.

5.2.5 . Efficient Networks for IoT

Hardware-aware NAS methods [58, 59, 60, 61, 62, 63, 64] directly incorporate the hardware feedback into efficient neural architecture search. [215] proposes learning a single network composed of many subnetworks from which a hardware-aware subnetwork can be extracted in linear time. [66] proposes a similar approach wherein they identify subnetworks that can be run efficiently on microcontrollers. Quantization methods can be classified into two categories: quantization-aware training [216, 217, 218] and post-training quantization [219, 220, 221, 222].

5.3 . Neural Network Pruning

In this section, we present our convex optimization based sparsification method called subdifferential inclusion for sparsity (SIS) that utilizes activation function properties. We then demonstrate the proposed method usefulness by sparsifying some widely used pre-trained networks from vision, NLP, and speech tasks.

5.3.1 . Variational Principles

A basic neural network layer can be described by the relation:

$$y = R(Wx + b) \tag{5.1}$$

where $x \in \mathbb{R}^M$ is the input, $y \in \mathbb{R}^N$ the output, $W \in \mathbb{R}^{N\times M}$ is the weight matrix, $b \in \mathbb{R}^N$ the bias vector, and R is a nonlinear activation operator from $\mathbb{R}^N$ to $\mathbb{R}^N$. A key observation is that most of the activation operators currently used in neural networks are proximity operators of convex functions [177, 223]. We will therefore assume that there exists a proper lower-semicontinuous convex function f from $\mathbb{R}^N$ to $\mathbb{R} \cup \{+\infty\}$ such that $R = \operatorname{prox}_f$. We recall that f is a proper lower-semicontinuous convex function if the area overs its graph, its epigraph $\big\{(y,\xi) \in \mathbb{R}^N \times \mathbb{R} \;\big|\; f(y) \leqslant \xi\big\}$, is a nonempty closed convex set. For

such a function the proximity operator of f at $z \in \mathbb{R}^N$ [176] is the unique point defined as

$$\operatorname{prox}_f(z) = \underset{p \in \mathbb{R}^N}{\operatorname{argmin}} \ \frac{1}{2}\|z-p\|^2 + f(p). \tag{5.2}$$

It follows from standard subdifferential calculus that Eq. (5.1) can be re-expressed as the following inclusion relation:

$$Wx + b - y \in \partial f(y), \tag{5.3}$$

where $\partial f(y)$ is the Moreau subdifferential of f at y defined as

$$\partial f(y) = \big\{ t \in \mathbb{R}^N \ \big| \ (\forall z \in \mathbb{R}^N) \quad f(z) \geqslant f(y) + \langle t \mid z - y \rangle \big\}. \tag{5.4}$$

The subdifferential constitutes a useful extension of the notion of differential, which is applicable to nonsmooth functions. The set $\partial f(y)$ is closed and convex and, if y satisfies Eq. (5.1), it is nonempty. The distance to this set of a point $z \in \mathbb{R}^N$ is given by

$$d_{\partial f(y)}(z) = \inf_{t \in \partial f(y)} \|z - t\|. \tag{5.5}$$

We thus see that the subdifferential inclusion in Eq. (5.3) is also equivalent to

$$d_{\partial f(y)}(Wx + b - y) = 0. \tag{5.6}$$

Therefore, a suitable accuracy measure for approximated values of the layer parameters (W, b) is $d_{\partial f(y)}(Wx + b - y)$.

5.3.2 . Optimization problem

Compressing a network consists of a sparsification of its parameters while keeping a satisfactory accuracy. Let us assume that, for a given layer, a training sequence of input/output pairs is available which results from a forward pass performed on the original network for some input dataset of length K. The training sequence is split in J minibatches of size T so that $K = JT$. The j-th minibatch with $j \in \{1, \ldots, J\}$ is denoted by $(x_{j,t}, y_{j,t})_{1 \leqslant t \leqslant T}$. In order to compress the network, we propose to solve the following constrained optimization problem.

Problem 1 We want to

$$\underset{(W,b) \in C}{\text{minimize}} \ \ g(W, b) \tag{5.7}$$

with

$$C = \Big\{ (W, b) \in \mathbb{R}^{N \times M} \times \mathbb{R}^N \mid (\forall j \in \{1, \ldots, J\}) \\ \sum_{t=1}^{T} d^2_{\partial f(y_{j,t})}(Wx_{j,t} + b - y_{j,t}) \leqslant T\eta \Big\}, \tag{5.8}$$

where g is a sparsity measure defined on $\mathbb{R}^{N\times M}\times\mathbb{R}^N$ and $\eta\in[0,+\infty[$ is some accuracy tolerance.

Since, for every $j\in\{1,\dots,J\}$, the function $(W,b)\mapsto\sum_{t=1}^{T} d^2_{\partial f(y_{j,t})}(Wx_{j,t}+b-y_{j,t})$ is continuous and convex, C is a closed and convex subset of $\mathbb{R}^{N\times M}\times\mathbb{R}^N$. In addition, this set is nonempty when there exist $\overline{W}\in\mathbb{R}^{N\times M}$ and $\overline{b}\in\mathbb{R}^N$ such that, for every $j\in\{1,\dots,J\}$ and $t\in\{1,\dots,T\}$,

$$d^2_{\partial f(y_{j,t})}(\overline{W}x_{j,t}+\overline{b}-y_{j,t})=0. \tag{5.9}$$

This condition is satisfied when $(\overline{W},\overline{b})$ are the parameters of the uncompressed layer. Often, the sparsity of the weight matrix is the determining factor whereas the bias vector represents a small number of parameters, so that we can make the following assumption ($\|\cdot\|_{\mathrm{F}}$ denotes here the Frobenius norm).

Assumption 2 For every $W\in\mathbb{R}^{N\times M}$ and $b\in\mathbb{R}^N$, $g(W,b)=h(W)$ where h is a function from $\mathbb{R}^{N\times M}$ to $\mathbb{R}\cup\{+\infty\}$, which is lower-semicontinuous, convex, and coercive (i.e. $\lim_{\|W\|_{\mathrm{F}}\to+\infty}h(W)=+\infty$). In addition, there exists $(\overline{W},\overline{b})\in C$ such that $h(\overline{W})<+\infty$ and there exists $(j^*,t^*)\in\{1,\dots,J\}\times\{1,\dots,T\}$ such that y_{j^*,t^*} lies in the interior of the range of R.

Under this assumption, the existence of a solution to Problem 1 is guaranteed. A standard choice for such a function is the ℓ_1-norm of the matrix elements, $h=\|\cdot\|_1$, but other convex sparsity measures could also be easily incorporated within this framework, e.g. group sparsity measures. Another point worth being noticed is that constraints other than (5.8) could be imposed. For example, one could make the following alternative choice for the constraint set

$$C=\Big\{(W,b)\in\mathbb{R}^{N\times M}\times\mathbb{R}^N \mid \sup_{j\in\{1,\dots,J\},t\in\{1,\dots,T\}} d_{\partial f(y_{j,t})}(Wx_{j,t}+b-y_{j,t})\leqslant\sqrt{\eta}\Big\}. \tag{5.10}$$

Although the resulting optimization problem could be tackled by the same kind of algorithm as the one we will propose, Constraint (5.8) leads to a simpler implementation.

5.3.3 . Optimization algorithm

A standard proximal method for solving Problem 1 is the Douglas-Rachford algorithm [224, 225]. This algorithm alternates between a proximal step aiming at sparsifying the weight matrix and a projection step allowing a given accuracy to be reached. This algorithm reads as shown below.

The Douglas-Rachford algorithm uses positive parameters γ and $(\lambda_n)_{n\in\mathbb{N}}$. Throughout this chapter, proj_S denotes the projection onto a nonempty closed convex set S. The convergence of Algorithm 1 is guaranteed by the following result (see illustrations in Section 5.4.3).

Algorithm 1: Douglas-Rachford algorithm for network compression

Initialize: $\widehat{W}_0 \in \mathbb{R}^{N\times M}$ and $b_0 \in \mathbb{R}^N$
for $n = 0, 1, \ldots$ **do**
$W_n = \mathrm{prox}_{\gamma h}(\widehat{W}_n)$
$(\widetilde{W}_n, \widetilde{b}_n) = \mathrm{proj}_C(2W_n - \widehat{W}_n, b_n)$
$\widehat{W}_{n+1} = \widehat{W}_n + \lambda_n(\widetilde{W}_n - W_n)$
$b_{n+1} = b_n + \lambda_n(\widetilde{b}_n - b_n)$.

Proposition 3 *[225] Assume that Problem 1 has a solution and that there exists $(\overline{W}, \overline{b}) \in C$ such $\overline{W}$ is a point in the interior of the domain of h. Assume that $\gamma \in]0, +\infty[$ and $(\lambda_n)_{n\in\mathbb{N}}$ in $]0, 2[$ is such that $\sum_{n\in\mathbb{N}} \lambda_n(2-\lambda_n) = +\infty$. Then the sequence $(W_n, b_n)_{n\in\mathbb{N}}$ generated by Algorithm 1 converges to a solution to Problem 1.*

The proximity operator of function γh has a closed-form for standard choices of sparsity measures[1]. For example, when $h = \|\cdot\|_1$, this operator reduces to a soft-thresholding (with threshold value γ) of the input matrix elements. In turn, since the convex set C has an intricate form, an explicit expression of proj_C does not exist. Finding an efficient method for computing this projection for large datasets thus constitutes the main challenge in the use of the above Douglas-Rachford strategy, which we will discuss next.

5.3.4 . Computation of the projection onto the constraint set

For every mini-batch index $j \in \{1, \ldots, J\}$, let us define the following convex function:

$$(\forall (W, b) \in \mathbb{R}^{N\times M} \times \mathbb{R}^N) \quad c_j(W, b) = \sum_{t=1}^{T} d^2_{\partial f(y_{j,t})}(Wx_{j,t} + b - y_{j,t}) - T\eta \tag{5.11}$$

Note that, for every $j \in \{1, \ldots, J\}$, function c_j is differentiable and its gradient at $(W, b) \in \mathbb{R}^{N\times M} \times \mathbb{R}^N$ is given by

$$\nabla c_j(W, b) = (\nabla_{\mathsf{W}} c_j(W, b), \nabla_{\mathsf{b}} c_j(W, b)), \tag{5.12}$$

where

$$\nabla_{\mathsf{W}} c_j(W, b) = 2\sum_{t=1}^{T} e_{j,t} x_{j,t}^{\top}, \quad \nabla_{\mathsf{b}} c_j(W, b) = 2\sum_{t=1}^{T} e_{j,t} \tag{5.13}$$

[1] http://proximity-operator.net

with, for every $t \in \{1, \dots, T\}$,

$$e_{j,t} = Wx_{j,t} + b - y_{j,t} - \mathrm{proj}_{\partial f(y_{j,t})}(Wx_{j,t} + b - y_{j,t}). \tag{5.14}$$

A pair of weight/bias parameters belongs to C if and only if it lies in the intersection of the 0-lower level sets of the functions $(c_j)_{1 \leqslant j \leqslant J}$. To compute the projection of some $(W, b) \in \mathbb{R}^{N \times M} \times \mathbb{R}^N$ onto this intersection, we use Algorithm 2.

Algorithm 2: Minibatch algorithm for computing $\mathrm{proj}_C(W, b)$

Initialize: $W_0 = W$ and $b_0 = b$
for $n = 0, 1, \dots$ **do**
  Select a batch of index $j_n \in \{1, \dots, J\}$
  if $c_{j_n}(W_n, b_n) > 0$ **then**
    Compute $\nabla_{\mathsf{W}} c_{j_n}(W_n, b_n)$ and $\nabla_{\mathsf{b}} c_{j_n}(W_n, b_n)$ by using Eqs. (5.13) and (5.14)
    $\delta W_n = \frac{c_{j_n}(W_n, b_n)\, \nabla_{\mathsf{W}} c_{j_n}(W_n, b_n)}{\|\nabla_{\mathsf{W}} c_{j_n,n}(W_n, b_n)\|_{\mathrm{F}}^2 + \|\nabla_{\mathsf{b}} c_{j_n}(W_n, b_n)\|^2}$
    $\delta b_n = \frac{c_{j_n}(W_n, b_n)\, \nabla_{\mathsf{b}} c_{j_n}(W_n, b_n)}{\|\nabla_{\mathsf{W}} c_{j_n,n}(W_n, b_n)\|_{\mathrm{F}}^2 + \|\nabla_{\mathsf{b}} c_{j_n}(W_n, b_n)\|^2}$
    $\pi_n = \mathrm{tr}((W_0 - W_n)^\top \delta W_n) + (b_0 - b_n)^\top \delta b_n$
    $\mu_n = \|W_0 - W_n\|_{\mathrm{F}}^2 + \|b_0 - b_n\|^2$
    $\nu_n = \|\delta W_n\|_{\mathrm{F}}^2 + \|\delta b_n\|^2$
    $\zeta_n = \mu_n \nu_n - \pi_n^2$
    if $\zeta_n = 0$ *and* $\pi_n \geqslant 0$ **then**
      $W_{n+1} = W_n - \delta W_n$
      $b_{n+1} = b_n - \delta b_n$
    else if $\zeta_n > 0$ *and* $\pi_n \nu_n \geqslant \zeta_n$ **then**
      $W_{n+1} = W_0 - (1 + \frac{\pi_n}{\nu_n}) \delta W_n$
      $b_{n+1} = b_0 - (1 + \frac{\pi_n}{\nu_n}) \delta b_n$
    else
      $W_{n+1} = W_n + \frac{\nu_n}{\zeta_n}(\pi_n (W_0 - W_n) - \mu_n \delta W_n)$
      $b_{n+1} = b_n + \frac{\nu_n}{\zeta_n}(\pi_n (b_0 - b_n) - \mu_n \delta b_n)$
  else
    $W_{n+1} = W_n$
    $b_{n+1} = b_n$

This iterative algorithm has the advantage of proceeding in a minibatch manner. It allows us to choose the mini-batch index j_n at iteration n in a quasi-cyclic manner. The simplest rule is to activate each minibatch once within J successive iterations of the algorithm so that they correspond to an epoch. The proposed algorithm

belongs to the family of block-iterative outer approximation schemes for solving constrained quadratic problems, which was introduced in [226]. The convergence of the sequence $(W_n, b_n)_{n\in\mathbb{N}}$ generated by Algorithm 2 to $\mathrm{proj}_C(W, b)$ is thus guaranteed. One of the main features of the algorithm is that it does not require to perform any projection onto the 0-lower level sets of the functions c_j, which would be intractable due to their expressions. Instead, these projections are implicitly replaced by subgradient projections, which are much easier to compute in our context.

5.3.5 . Dealing with various nonlinearities

Name	$\rho(\zeta)$	$\mathrm{proj}_{\partial\varphi(v)}(\zeta)$
Sigmoid	$(1+e^{-\zeta})^{-1} - \frac{1}{2}$	$\ln(v+1/2) - \ln(v-1/2) - v$
Arctangent	$(2/\pi)\arctan(\zeta)$	$\tan(\pi v/2) - v$
ReLU	$\max\{\zeta, 0\}$	$\begin{cases} 0 & \text{if } v > 0 \text{ or } \zeta \geqslant 0 \\ \zeta & \text{otherwise} \end{cases}$
Leaky ReLU	$\begin{cases} \zeta & \text{if } \zeta > 0 \\ \alpha\zeta & \text{otherwise} \end{cases}$	$\begin{cases} 0 & \text{if } v > 0 \\ (1/\alpha - 1)v & \text{otherwise} \end{cases}$
Capped ReLU	$\mathrm{ReLU}_\alpha(\zeta) = \min\{\max\{\zeta, 0\}, \alpha\}$	$\begin{cases} \zeta & \text{if } (v = 0 \text{ and } \zeta < 0) \\ & \text{or } (v = \alpha \text{ and } \zeta > 0) \\ 0 & \text{otherwise} \end{cases}$
ELU	$\begin{cases} \zeta & \text{if } \zeta \geqslant 0 \\ \alpha(\exp(\zeta) - 1) & \text{otherwise} \end{cases}$	$\begin{cases} 0 & \text{if } v > 0 \\ \ln\left(\frac{v+\alpha}{\alpha}\right) - v & \text{otherwise} \end{cases}$
QuadReLU	$\frac{(\zeta+\alpha)\mathrm{ReLU}_{2\alpha}(\zeta+\alpha)}{4\alpha}$	$\begin{cases} v & \text{if } v = 0 \text{ and } \zeta \leqslant -\alpha \\ -v + 2\sqrt{\alpha v} - \alpha & \text{if } v \in]0, \alpha] \\ & \text{or } (v = 0 \text{ and } \zeta > -\alpha) \\ v - \alpha & \text{otherwise} \end{cases}$
Softmax	$\left(\frac{\exp(\zeta^{(k)})}{\sum_{k'=1}^N \exp(\zeta^{(k')})}\right)_{1\leqslant k\leqslant N}$	$Q(y) + \frac{\mathbf{1}^\top(z - Q(y))}{N}\mathbf{1}$

Table 5.1: Expression of $\mathrm{proj}_{\partial\varphi(v)}(\zeta)$ for $\zeta \in \mathbb{R}$ and v in the range of ρ, for standard activation functions ρ. α is a positive constant.

For any choice of activation operator R, we have to calculate the projection onto $\partial f(y)$ for every vector y satisfying Eq. (5.1). This projection is indeed required in the computation of the gradients of functions $(c_j)_{1\leqslant j\leqslant J}$, as shown by Eq. (5.14). Two properties may facilitate this calculation. First, if f is differentiable at y, then $\partial f(y)$ reduces to a singleton containing the gradient $\nabla f(y)$ of f at y, so that, for every $z \in \mathbb{R}^N$, $\mathrm{proj}_{\partial f(y)}(z) = \nabla f(y)$. Second, R is often separable, i.e. consists of the application of a scalar activation function $\rho\colon \mathbb{R} \to \mathbb{R}$ to each component of its input argument. According to our assumptions, there thus exists a proper lower-semicontinuous convex function φ from $\mathbb{R}$ to $\mathbb{R} \cup \{+\infty\}$ such that $\rho = \mathrm{prox}_\varphi$ and, for every $z = (\zeta^{(k)})_{1\leqslant k\leqslant N} \in \mathbb{R}^N$, $f(z) = \sum_{k=1}^N \varphi(\zeta^{(k)})$. This implies that, for every $z = (\zeta^{(k)})_{1\leqslant k\leqslant N} \in \mathbb{R}^N$, $\mathrm{proj}_{\partial f(y)}(z) = (\mathrm{proj}_{\partial\varphi(v^{(k)})}(\zeta^{(k)}))_{1\leqslant k\leqslant N}$, where

the components of y are denoted by $(v^{(k)})_{1\leqslant k\leqslant N}$. Based on these properties, a list of standard activation functions ρ is given in Table 5.1, for which we provide the associated expressions of the projection onto $\partial\varphi$. As shown next, these results are derived from the expression of the convex function φ associated with each activation function ρ [177, Section 2.1] [223, Section 3.2].

- **Sigmoid**

$$(\forall \zeta \in \mathbb{R}) \quad \varphi(\zeta) = \begin{cases} (\zeta + 1/2)\ln(\zeta + 1/2) + (1/2 - \zeta) \\ \qquad \ln(1/2 - \zeta) - \dfrac{1}{2}(\zeta^2 + 1/4) & \text{if } |\zeta| < 1/2 \\ -1/4 & \text{if } |\zeta| = 1/2 \\ +\infty & \text{if } |\zeta| > 1/2. \end{cases} \tag{5.15}$$

The range of the Sigmoid function is $]-1/2, 1/2[$ and the above function is differentiable on this interval and its derivative at every $v \in]-1/2, 1/2[$ is

$$\varphi'(v) = \ln(v + 1/2) - \ln(v - 1/2) - v. \tag{5.16}$$

We deduce that, for every $\zeta \in \mathbb{R}$, $\mathrm{proj}_{\partial\varphi(v)}(\zeta) = \varphi'(v)$.

- **Arctangent**

$$(\forall \zeta \in \mathbb{R}) \quad \varphi(\zeta) = \begin{cases} -\frac{2}{\pi}\ln\left(\cos\left(\frac{\pi\zeta}{2}\right)\right) - \frac{1}{2}\zeta^2, & \text{if } |\zeta| < 1 \\ +\infty, & \text{if } |\zeta| \geqslant 1. \end{cases} \tag{5.17}$$

By proceeding for this function similarly to the Sigmoid function, we have, for every $v \in \rho(\mathbb{R}) =]-1, 1[$,

$$(\forall \zeta \in \mathbb{R}) \ \ \mathrm{proj}_{\partial\varphi(v)}(\zeta) = \varphi'(v) = \tan(\pi v/2) - v. \tag{5.18}$$

- **ReLU**

$$(\forall \zeta \in \mathbb{R}) \quad \varphi(\zeta) = \begin{cases} 0 & \text{if } \zeta \geqslant 0 \\ +\infty & \text{otherwise.} \end{cases} \tag{5.19}$$

For every $v \in \rho(\mathbb{R}) = [0, +\infty[$, we have

$$\partial\varphi(v) = \begin{cases} \{0\} & \text{if } v > 0 \\ ]-\infty, 0] & \text{if } v = 0. \end{cases} \tag{5.20}$$

We deduce that

$$(\forall \zeta \in \mathbb{R}) \quad \mathrm{proj}_{\partial\varphi(v)}(\zeta) = \begin{cases} 0 & \text{if } v > 0 \text{ or } \zeta \geqslant 0 \\ \zeta & \text{otherwise.} \end{cases} \tag{5.21}$$

- **Leaky ReLU**

$$(\forall \zeta \in \mathbb{R}) \qquad \varphi(\zeta) = \begin{cases} 0, & \text{if } \zeta > 0 \\ (1/\alpha - 1)\zeta^2/2 & \text{if } \zeta \leqslant 0. \end{cases} \tag{5.22}$$

Since this function is differentiable on $\mathbb{R}$, for every $v \in \mathbb{R}$,

$$\begin{aligned} (\forall \zeta \in \mathbb{R}) \quad \operatorname{proj}_{\partial\varphi(v)}(\zeta) &= \varphi'(v) \\ &= \begin{cases} 0 & \text{if } v > 0 \\ (1/\alpha - 1)v & \text{otherwise.} \end{cases} \end{aligned} \tag{5.23}$$

- **Capped ReLU**

$$(\forall \zeta \in \mathbb{R}) \qquad \varphi(\zeta) = \begin{cases} 0 & \text{if } \zeta \in [0, \alpha] \\ +\infty & \text{otherwise.} \end{cases} \tag{5.24}$$

We have thus, for every $v \in [0, \alpha]$,

$$\partial\varphi(v) = \begin{cases} \{0\} & \text{if } v \in]0, \alpha[\\ ]-\infty, 0] & \text{if } v = 0 \\ [0, +\infty[& \text{if } v = \alpha. \end{cases} \tag{5.25}$$

This leads to

$$(\forall \zeta \in \mathbb{R}) \quad \operatorname{proj}_{\partial\varphi(v)}(\zeta) = \begin{cases} \zeta & \text{if } (v = 0 \text{ and } \zeta < 0) \\ & \text{or } (v = \alpha \text{ and } \zeta > 0) \\ 0 & \text{otherwise.} \end{cases} \tag{5.26}$$

- **ELU**

$$(\forall \zeta \in \mathbb{R}) \qquad \varphi(\zeta) = \begin{cases} 0 & \text{if } \zeta \geqslant 0; \\ (\zeta + \alpha)\ln\left(\frac{\zeta+\alpha}{\alpha}\right) - \zeta - \frac{\zeta^2}{2}, & \text{if } -\alpha < \zeta < 0 \\ \alpha - \frac{\alpha^2}{2}, & \text{if } \zeta = -\alpha \\ +\infty, & \text{if } \zeta < -\alpha. \end{cases} \tag{5.27}$$

This function being differentiable on $\rho(\mathbb{R}) =]-\alpha, +\infty[$, we have for every $v \in]-\alpha, +\infty[$,

$$\begin{aligned} (\forall \zeta \in \mathbb{R}) \quad \operatorname{proj}_{\partial\varphi(v)}(\zeta) &= \varphi'(v) \\ &= \begin{cases} 0 & \text{if } v > 0 \\ \ln\left(\frac{v+\alpha}{\alpha}\right) - v & \text{otherwise.} \end{cases} \end{aligned} \tag{5.28}$$

- **QuadReLU** Unlike the previous ones, this function does not seem to have been investigated before. It can be seen as a surrogate to the hard swish activation function, which is not a proximal activation function. Let us define

$$(\forall \zeta \in \mathbb{R}) \qquad \varphi(\zeta) = \begin{cases} +\infty & \text{if } \zeta < 0 \\ -\frac{\zeta^2}{2} + \frac{4}{3}\sqrt{\alpha}\zeta^{3/2} - \alpha\zeta & \text{if } \zeta \in [0, \alpha] \\ \frac{\zeta^2}{2} - \alpha\zeta + \frac{\alpha^2}{3} & \text{if } \zeta > \alpha. \end{cases} \tag{5.29}$$

φ is a lower-semicontinuous convex function whose subdifferential is

$$(\forall v \in [0, +\infty[) \quad \partial\varphi(v) = \begin{cases}]-\infty, -\alpha] & \text{if } v = 0 \\ \{-v + 2\sqrt{\alpha v} - \alpha\} & \text{if } v \in]0, \alpha] \\ \{v - \alpha\} & \text{if } v > \alpha. \end{cases} \tag{5.30}$$

From the definition of the proximity operator, for every $(v, \zeta) \in \mathbb{R}^2$, we have $v = \operatorname{prox}_{\varphi}(\zeta)$ if and only if

$$\begin{aligned} \zeta \in v + \partial\varphi(v) \quad &\Leftrightarrow \quad \begin{cases} \zeta \in]-\infty, -\alpha] & \text{if } v = 0 \\ \zeta = 2\sqrt{\alpha v} - \alpha & \text{if } v \in]0, \alpha] \\ \zeta = 2v - \alpha & \text{if } v > \alpha. \end{cases} \\ &\Leftrightarrow \quad v = \begin{cases} 0 & \text{if } \zeta \in]-\infty, -\alpha] \\ \frac{(\zeta+\alpha)^2}{4\alpha} & \text{if } \zeta \in]-\alpha, \alpha] \\ \frac{\zeta+\alpha}{2} & \text{if } \zeta > \alpha. \end{cases} \end{aligned} \tag{5.31}$$

This shows that

$$\operatorname{prox}_{\varphi}(\zeta) = (4\alpha)^{-1}(\zeta + \alpha)\,\mathrm{ReLU}_{2\alpha}(\zeta + \alpha). \tag{5.32}$$

In addition, for every $v \in [0, +\infty[$, it follows from Eq. (5.30) that the projection onto $\partial f(v)$ is

$$(\forall \zeta \in \mathbb{R}) \quad \operatorname{proj}_{\partial f(v)}(\zeta) = \begin{cases} v & \text{if } v = 0 \text{ and } \zeta \leqslant -\alpha \\ -v + 2\sqrt{\alpha v} - \alpha & \text{if } v \in]0, \alpha] \\ & \text{or } (v = 0 \text{ and } \zeta > -\alpha) \\ v - \alpha & \text{if } v > \alpha. \end{cases} \tag{5.33}$$

- **Softmax Activation** Softmax is a non-separable activation operator frequently employed in neural network architectures, it is defined as

$$(\forall z = (\zeta^{(k)})_{1 \leqslant k \leqslant N} \in \mathbb{R}^N) \qquad R(z) = \left(\frac{\exp(\zeta^{(k)})}{\sum_{k'=1}^{N} \exp(\zeta^{(k')})} \right)_{1 \leqslant k \leqslant N}. \tag{5.34}$$

Let C denote the closed hypercube $[0,1]^N$, let V be the vector hyperplane defined as

$$V = \{z = (\zeta^{(k)})_{1\leqslant k\leqslant N} \in \mathbb{R}^N \mid \sum_{k=1}^{N} \zeta^{(k)} = 0\}, \tag{5.35}$$

and let A be the affine hyperplane defined as

$$A = \{z = (\zeta^{(k)})_{1\leqslant k\leqslant N} \in \mathbb{R}^N \mid \sum_{k=1}^{N} \zeta^{(k)} = 1\} = V + u, \tag{5.36}$$

where $u = [1,\dots,1]^\top/N = \mathbf{1}/N \in \mathbb{R}^N$. If R is the Softmax activation operator, the convex function f such that $\operatorname{prox}_f = R$ is [177, Example 2.23]:

$$(\forall z = (\zeta^{(k)})_{1\leqslant k\leqslant N}) \quad f(z) = \begin{cases} \sum_{k=1}^{N} \varphi(\zeta^{(k)}) & \text{if } z \in C \cap A \\ +\infty & \text{otherwise,} \end{cases} \tag{5.37}$$

where

$$(\forall \zeta \in [0,+\infty[) \quad \varphi(\zeta) = \zeta \ln \zeta - \frac{\zeta^2}{2} \tag{5.38}$$

(with the convention $0 \ln 0 = 0$). The latter function is differentiable on $]0,+\infty[$. It then follows from standard subdifferential calculus rules that, for every $y = (v^{(k)})_{1\leqslant k\leqslant N} \in]0,+\infty[^N$,

$$\partial f(y) = (\varphi'(v^{(k)}))_{1\leqslant k\leqslant N} + \partial \iota_{C\cap A}(y), \tag{5.39}$$

where φ' is the derivative of φ on $]0,+\infty[$ and $\iota_{C\cap A}$ denotes the indicator function of the intersection of C and A (equal to 0 on this set and $+\infty$ elsewhere). It can be deduced from Eq. (5.39) that, for every $y = (v^{(k)})_{1\leqslant k\leqslant N} \in]0,+\infty[^N$,

$$\partial f(y) = (\varphi'(v^{(k)}))_{1\leqslant k\leqslant N} + N_C(y) + N_A(y), \tag{5.40}$$

where N_D denotes the normal cone to a nonempty closed convex set D, which is defined as

$$(\forall y \in D) \;\; N_D(y) = \{t \in \mathbb{R}^N \mid (\forall z \in D) \; \langle t \mid z - y\rangle \leqslant 0\}. \tag{5.41}$$

Thus, for every $y \in A$, $N_A(y) = N_V(y-u)$ is the orthogonal space $V^\perp$ of V.

Let us now assume that $y \in R(\mathbb{R}^N) =]0,1[^N \cap A$. Then, since y is an interior point of C, $N_C(y) = \{0\}$. We then deduce from Eq. (5.40) that

$$\partial f(y) = Q(y) + V^\perp, \tag{5.42}$$

where

$$Q(y) = (\varphi'(v^{(k)}))_{1\leqslant k\leqslant N} = (\ln v^{(k)} + 1 - v^{(k)})_{1\leqslant k\leqslant N}. \tag{5.43}$$

It follows that, for every $z \in \mathbb{R}^N$,

$$\mathrm{proj}_{\partial f(y)}(z) = Q(y) + \mathrm{proj}_{V^\perp}(z - Q(y)). \tag{5.44}$$

By using the expression of the projection $\mathrm{proj}_V = \mathrm{Id} - \mathrm{proj}_{V^\perp}$ onto hyperplane V, we finally obtain

$$\mathrm{proj}_{\partial f(y)}(z) = Q(y) + \frac{\mathbf{1}^\top (z - Q(y))}{N} \mathbf{1}. \tag{5.45}$$

5.3.6 . SIS on multi-layered networks

Algorithm 3: Parallel SIS for multi-layered network

Input: input sequence $X \in \mathbb{R}^{M \times K}$, compression parameter $\eta > 0$, weight matrices $W^{(1)}, \ldots, W^{(L)}$, and bias vectors $b^{(1)}, \ldots, b^{(L)}$

$Y^{(0)} \leftarrow X$

for $l = 1, \ldots, L$ **do**

  $Y^{(l)} = R_l(W^{(l)\top} Y^{(l-1)} + b^{(l)})$

  $\widehat{W}^{(l)}, \widehat{b}^{(l)} \leftarrow \mathrm{SIS}(\eta, W^{(l)}, b^{(l)}, Y^{(l)}, Y^{(l-1)})$

Output: $\widehat{W}^{(1)}, \ldots, \widehat{W}^{(L)}$ and $\widehat{b}^{(1)}, \ldots, \widehat{b}^{(L)}$

Algorithm 3 describes how we make use of SIS for a multi-layered neural network. We use a pre-trained network and part of the training sequence to extract layer-wise input-output features. Then we apply SIS on each individual layer l by passing η, layer parameters $(W^{(l)}, b^{(l)})$ and extracted input-output features $(Y^{(l-1)}, Y^{(l)})$ to Algorithm 1. The benefit of applying SIS to each layer independently is that we can run SIS on all the layers of a network in parallel. This reduces the time required to process the whole network and compute resources are optimally utilized.

5.4 . Experiments

In this section, we conduct various experiments to validate the effectiveness of SIS in terms of test accuracy vs. sparsity and inference time FLOPs vs. sparsity by comparing against RigL [201]. We also include SNIP [56], GraSP [202], SynFlow [204], STR [174], and FORCE [205]. These methods start training from a sparse network and have some limitations when compared to methods that prune a pre-trained network [227]. For a fair comparison we also include LRR [50] which uses a pre-trained network and multiple rounds of pruning and retraining by leveraging learning rate rewinding.

PyTorch is employed to implement our method. We use and extend SNIP and

RigL code available here[2], LRR[3], GraSP[4], SynFlow[5], STR[6], and FORCE[7]. In order to manage our experiments we use Polyaxon[8] on a Kubernetes[9] cluster and use five computing nodes with eight V100 GPUs each. Floating point operations per second (FLOPs) is calculated as equal to one multiply-add accumulator using the code[10].

SIS has the following parameters: number of iterations of Algorithm 1, number of iterations of Algorithm 2, step size parameter γ in Algorithm 1, constraint bound parameter η used to control the sparsity, and relaxation parameter $\lambda_n \equiv \lambda$ of Algorithm 1. In our experiments, the maximum numbers of iterations of Algorithms 1 and 2 are set to 2000 and 1000, respectively. λ is set to 1.5 and γ is set to 0.1 for all the SIS experiments. η value depends on the network and dataset. With few experiments, we search for a good η value that gives suitable sparsity and accuracy.

5.4.1 . Pruning of Convolutional Networks

5.4.1.1 . VGG19 and ResNet50 on CIFAR-10/100.

We train VGG19 on CIFAR-10 for 160 epochs with a batch size of 128, learning rate of 0.1 and weight decay of 5×10^{-4} applied at epochs 81 and 122. A momentum of 0.9 is used with stochastic gradient descent (SGD). We make use of 1000 images per training class when using SIS. We fine-tune the identified sparse subnetwork for 10 epochs at a learning rate of 10^{-3}. For CIFAR-100 we keep the same training hyperparameters as for CIFAR-10. When applying SIS to the dense network, we use 300 images per class from the training samples. We fine-tune the identified sparse subnetwork for 40 epochs on the training set with a learning rate of 10^{-3}. ResNet50 employs the same hyperparameters as VGG19, except the weight decay that we set to 10^{-4}. When applying SIS to train dense ResNet50, we use the same partial training set and the same hyperparameters during fine-tuning. In case of VGG19 for CIFAR-10 and CIFAR-100, we found that η values in range $(1.5, 2)$ works best for sparsity range $(90\%, 98\%)$. In case of ResNet50, η values in range $(1, 2)$ is used.

Table 5.2 shows SIS and competitive baselines on CIFAR-10/100 for three different sparsity regimes 90%, 95%, 98%. It can be observed that LRR, RigL and SIS are able to maintain high accuracy with increasing sparsity. LRR performs

[2] https://github.com/google-research/rigl

[3] https://github.com/lottery-ticket/rewinding-iclr20-public/tree/master/vision/gpu-src/official

[4] https://github.com/alecwangcq/GraSP

[5] https://github.com/ganguli-lab/Synaptic-Flow

[6] https://github.com/RAIVNLab/STR

[7] https://github.com/naver/force

[8] https://github.com/polyaxon/polyaxon

[9] https://kubernetes.io/

[10] https://github.com/Lyken17/pytorch-OpCounter

Dataset	CIFAR-10			CIFAR-100		
Pruning ratio	90%	95%	98%	90%	95%	98%
VGG19 (Baseline)	94.23	-	-	74.16	-	-
SNIP [56]	93.63	93.43	92.05	**72.84**	71.83	58.46
GraSP [202]	93.30	93.04	92.19	71.95	71.23	68.90
SynFlow [204]	93.35	93.45	92.24	71.77	71.72	70.94
STR [174]	93.73	93.27	92.21	71.93	71.14	69.89
FORCE [205]	93.87	93.30	92.25	71.9	71.73	70.96
LRR [50]	**94.03**	**93.53**	91.73	72.12	71.36	70.39
RigL [201]	93.47	93.35	93.14	71.82	71.53	70.71
SIS (Ours)	93.99	93.31	**93.16**	72.06	**71.85**	**71.17**
ResNet50 (Baseline)	94.62	-	-	77.39	-	-
SNIP [56]	92.65	90.86	87.21	73.14	69.25	58.43
GraSP [202]	92.47	91.32	88.77	73.28	70.29	62.12
SynFlow [204]	92.49	91.22	88.82	73.37	70.37	62.17
STR [174]	92.59	91.35	88.75	73.45	70.45	62.34
FORCE [205]	92.56	91.46	88.88	73.54	70.37	62.39
LRR [50]	92.62	91.27	89.11	**74.13**	70.38	62.47
RigL [201]	92.55	91.42	89.03	73.77	70.49	62.33
SIS (Ours)	**92.81**	**91.69**	**90.11**	73.81	**70.62**	**62.75**

Table 5.2: Test accuracy of sparse VGG19 and ResNet50 on CIFAR-10 and CIFAR-100 datasets.

better than both RigL and SIS for VGG19 on CIFAR-10 at 90% and 95% sparsity. When compared to SNIP, our method achieves impressive performance for VGG19 on CIFAR-100 (58.46 $\rightarrow$ 71.17). In the case of ResNet50, SIS outperforms all the other methods for CIFAR-10/100 except for CIFAR-100 at 90%.

5.4.1.2 . ResNet50 on ImageNet

Due to its small size and controlled nature, CIFAR-10/100 may not appear sufficient to draw solid conclusions. We thus conduct further experiments on ImageNet using ResNet50 and MobileNets. For ResNet50 on ImageNet experiment, we adapt SNIP [192], GraSP [202], SynFlow [204], STR [174], FORCE [205], SpraseVD [47], Bayesian Compression [187], and L0 regularization [188] methods to use pre-trained weights. We also include results from NetTrim [175] which is another convex optimization based pruning method.

We use the weights of ResNet50 pre-trained on ImageNet available at PyTorch hub[11]. When applying SIS to the dense pre-trained network we use 20% samples per class from the training set. We fine-tune the identified sparse subnetwork for 40 epochs on the training set with a learning rate of 10^{-4}. We use different η values in range $(0.7, 1.5)$ for sparsity range $(60\%, 90\%)$. We found that $\eta = 2.3$ achieves 96.5% sparsity.

Table 5.3 shows that, in the case of ResNet50, STR performs marginally

[11] https://pytorch.org/hub/pytorch_vision_resnet/

Sparsity	60%			80%		
	Train/Prune FLOPs (×e16)	Top-1 Acc(%)	Infer FLOPs	Train/Prune FLOPs (×e16)	Top-1 Acc(%)	Infer FLOPs
SNIP	0.978	74.06	1.88G	0.696	72.34	941M
GraSP	0.903	75.95	1.63G	0.650	74.21	786M
SynFlow	0.898	76.54	1.61G	0.665	74.14	776M
FORCE	**0.833**	75.47	1.39G	0.619	73.42	685M
SparseVD	1.827	76.75	1.71G	1.737	74.68	811M
BC-GHS.	1.825	76.45	1.69G	1.737	74.15	813M
$L_{0_{hc}}, \lambda = e-5$	1.825	76.98	1.69G	1.736	76.67	802M
STR	0.891	**77.75**	1.59G	0.625	76.11	704M
NetTrim	1.148	74.52	1.71G	0.866	72.88	842M
SIS (Ours)	0.923	77.05	**1.34G**	**0.435**	**76.96**	**647M**
Sparsity	90%			96.5%		
SNIP	0.537	66.97	409M	0.502	59.16	292M
GraSP	0.555	70.71	470M	0.501	69.55	290M
SynFlow	0.553	71.01	465M	0.500	70.10	288M
FORCE	0.550	72.59	455M	0.497	72.04	276M
SparseVD	1.702	69.73	461M	1.685	67.13	286M
BC-GHS.	1.701	71.33	454M	1.684	68.54	282M
$L_{0_{hc}}, \lambda = e-5$	1.702	71.61	459M	1.684	68.61	276M
STR	0.516	75.72	341M	0.449	71.87	117M
NetTrim	0.465	67.62	461M	0.283	62.01	281M
SIS (Ours)	**0.351**	**76.31**	**298M**	**0.102**	**73.11**	**101M**

Table 5.3: Pruning phase compute cost, test Top-1 accuracy and single image inference FLOPs of sparse ResNet50 on ImageNet where baseline accuracy and inference FLOPs are 77.37% and 4.14G, respectively. All methods were applied on same pre-trained "dense" ResNet50. 20% samples per class were used during pruning phase of all the methods and were run for 40 epochs.

better than SIS at 60% sparsity. At 80%, 90%, and 96.5% sparsity SIS outperforms all other methods. For all sparsity regimes, SIS achieves least inference FLOPs. Training FLOPs is best for SIS in 80%, 90%, and 96.5% regimes, FORCE achieves best training FLOPs in 60% regime. MobileNets are compact architectures designed specifically for resource-constrained devices.

5.4.1.3 . MobileNets on ImageNet

We use MobileNetV1 dense pre-trained model from here[12] and MobileNetV2 from PyTorch hub[13]. In case of MobileNetV3, we replace the hard swish activation

[12] https://github.com/RAIVNLab/STR

[13] https://pytorch.org/hub/pytorch_vision_mobilenet_v2/

Sparsity	75%			90%		
	LRR	RigL	SIS (Ours)	LRR	RigL	SIS (Ours)
V1 (70.90)	68.79	69.97	**70.11**	66.59	67.10	**67.15**
FLOPs (569M)	498M	461M	**367M**	401M	331M	**284M**
V2 (71.88)	68.83	69.60	**69.83**	64.17	**65.23**	65.11
FLOPs (300M)	267M	211M	**182M**	192M	174M	**162M**
V3 (72.80)	68.97	70.21	**70.47**	64.32	65.13	**66.07**
FLOPs (226M)	187M	198M	**172M**	185M	167M	**151M**

Table 5.4: Test accuracy and inference FLOPs of sparse MobileNet versions using RigL and SIS on ImageNet, baseline accuracy and inference FLOPs shown in brackets.

function used in [64] with our QuadReLU function (see the last row of Table 5.1). We use hyperparameters provided in the original paper to train MobileNetV3. When applying SIS to the dense pre-trained MobileNets, we use 20% samples per class from the training set. We fine-tune the identified sparse subnetwork for 30 epochs on the training set with a learning rate of 10^{-4}. For MobileNets, we search η values in range $(0.6, 1.75)$ for sparsity range $(75, 90)$.

Table 5.4 shows results for RigL and SIS on MobileNets. We observe that SIS outperforms all MobileNet versions at 75% sparsity level. For a 90% sparsity level, SIS outperforms RigL for MobileNet V1 and V3 whereas, for MobileNetV2, RigL performs slightly better than SIS at 90% sparsity level. In all the cases, we can see that the resulting SIS sparse network uses fewer FLOPs than RigL. A possible explanation for this fact is that SIS leverages activation function properties during the sparsification process.

5.4.2 . Sequential Tasks

Network	JASPER		Transformer-XL		N-BEATS	
	WER	FLOPs	PPL	FLOPs	SMAPE	FLOPs
Dense	12.2	4.53G	18.6	927.73G	8.3	41.26M
SNIP [56]	14.3	2.74G	24.6	398.92G	10.1	21.45M
LRR [50]	13.7	2.61G	23.1	339.21G	**9.3**	14.47M
RigL [201]	13.9	2.69G	22.4	326.56G	10.2	15.13M
SIS (Ours)	**13.1**	**2.34G**	**21.1**	**290.38G**	9.7	**14.21M**

Table 5.5: Test accuracy and inference FLOPs of JASPER, Transformer-XL, and N-BEATS at 70% sparsity.

5.4.2.1 . Jasper on LibriSpeech.

Jasper is a speech recognition model that uses 1D convolutions. A BxR Jasper network has B blocks, each consisting of R repeating sub-blocks. Each sub-block consists of 1D-Convolution, Batch Normalization, ReLU activation, and Dropout. The kernel size of convolutions increases with depth. The network has

one convolution block at the beginning and three at the end. We train a network of 13 encoding blocks and one decoding block, having 54 1D-Convolution layers on the LibriSpeech dataset. Jasper network is trained on train-clean-100, train-clean-360, and train-other-500 splits of the LibriSpeech dataset [228]. The training configuration can be found here[14]. The trained network is a 333 million parameter model and has a word error rate (WER) of 12.2 on the test set.

We use train-clean-100 when using SIS and compare it with RigL and SNIP in terms of sparsity. We fine-tune the identified sparse sub-network on the completed training set for ten epochs with a learning rate of 10^{-4}. We use η values in range $(0.6, 1.75)$ for sparsity range $(70, 90)$. Table 5.5 reports WER and inference FLOPs for all three methods. SIS marginally performs better than LRR on this task in terms of WER and FLOPs for 70% sparsity. The main advantage of our approach lies in the fact that we can use a single pre-trained Jasper network and achieve different sparsity level for different types of deployment scenarios with less computational resources than RigL.

5.4.2.2 . Transformer-XL on WikiText-103.

Transformer-XL is a language model with 246 million parameters. We train the Transformer-XL network [171] on the base version of WikiText-103 [229]. We use the training configuration available here[15]. The trained network on WikiText-103 has a perplexity score (PPL) of 18.6.

We use 10% of the training set articles when using SIS. We use η values in range $(0.5, 0.75)$ for sparsity range $(40, 70)$. In Table 5.5, we see that SIS performs better than SNIP and RigL in terms of PPL and has 68% fewer inference FLOPs. This is due to the fact that large language models can be efficiently trained and then compressed easily, but training a sparse sub-network from scratch is hard [230], as is the case with SNIP and RigL. SNIP uses one-shot pruning to obtain a random sparse sub-network, whereas RigL is able to change its structure during training, which allows it to perform better than SNIP.

5.4.2.3 . N-BEATS on M4.

N-BEATS is a very deep residual fully-connected network to perform forecasting in univariate time-series problems. We train the interpretable architecture network of N-BEATS on the M4 dataset. The trained network has six residual blocks. Each block consists of four fully-connected layers and two linear projection layers. With 24 fully-connected layers, this network has 14 million trainable parameters. To compare different methods, we only train a single network on a 48-hour window instead of 180 networks on different timescales. We use the training configuration available here[16]. The training set has 50K time-series samples. The SMAPE of the dense network on the test set is 8.3%. We use 10K training samples to generate

[14] https://tinyurl.com/yrp33w72
[15] https://tinyurl.com/ym3hzmrd
[16] https://tinyurl.com/2p8nk7d3

a sparse sub-network using SIS. We use η values in range $(0.75, 1.5)$ for sparsity range $(70, 90)$. As shown Table 5.5, SIS performs better than both methods and results in 65% fewer inference FLOPs.

5.4.3 . Empirical Convergence Analysis

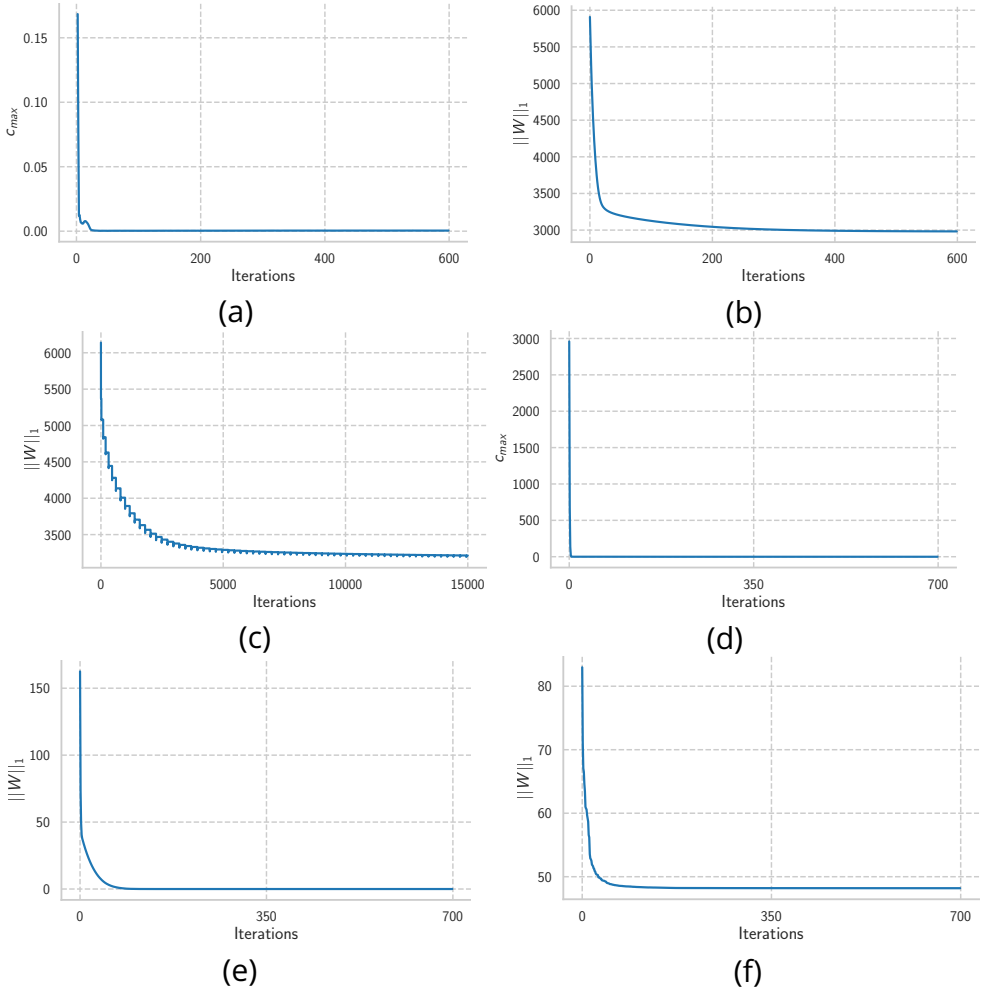


Figure 5.1: Convergence of SLIC: Top row coresponds to the first layer (ReLU activated) and bottom row corresponds to the last one (softmaxed) in LeNet-FCN. (a) and (d) show the evolution of the maximum value $c_{\max}$ of the constraint functions $(c_j)_{1 \leqslant j \leqslant J}$, (b) and (e) show the evolution of $\|W\|_1$ along Algorithm 1 iterations. (c) and (f) show $\|W\|_1$ evolution in Algorithm 2.

We illustrate the convergence of our method on LeNet-FCN trained on MNIST. LeNet-FCN is a fully-connected network having four layers with 784-300-1000-300-10 nodes (two 300 nodes and one 1000 node hidden layers). Figure 5.1 shows the convergence of SIS when applied to dense LeNet-FCN. We observe that the convergence is smooth and SIS finds a global solution for the first (ReLU activated) and last (softmax) layer cases. This fact is in agreement with our theoretical claims.

SIS attains a sparsity of 99.21% at an error of 1.86%. The trained dense network has an error of 1.65%. This result is obtained at $\eta = 2$.

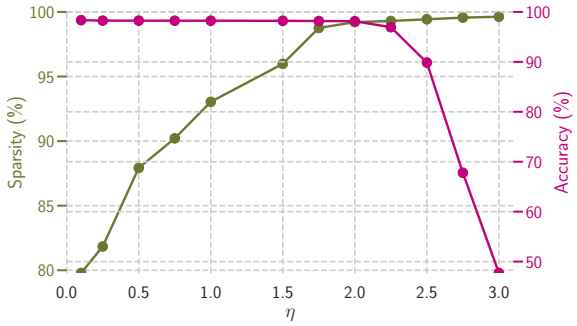


Figure 5.2: Effect of η on LeNet-FCN

The η parameter in our algorithm controls the accuracy tolerance. The higher, the more tolerant we are on the loss of precision and the sparser the network is. Thus, this parameter also controls the network sparsity. The choice of this parameter should be the result of an accuracy-sparsity trade-off. This is illustrated in Figure 5.2.

5.5 . Summary

In this chapter, we have introduced a novel method for sparsifying neural networks. The compression problem for each layer has been recast as the minimization of a sparsity measure under accuracy constraints. This constrained optimization problem has been solved using advanced convex optimization tools. Our method SIS is reliable in terms of iteration convergence guarantees, applicable to a wide range of activation operators, and able to deal with large datasets. Our experiments demonstrate that the approach is appealing from a theoretical viewpoint and practically efficient.

Chapter 6
Conclusion and Future Work

This thesis examined the feasibility of applying neural networks to various electrical motor problems. In this thesis, we describe and introduce several methods that can be used to better align current neural network research with Industry 4.0. The contributions of this thesis are summarized below in more detail, along with recommendations for future work.

6.1 . Conclusion

This thesis started with a general introduction to electrical motor mechanics by explaining its mathematical modelling. We presented transformations representing electrical motor quantities in a different reference frame for more straightforward calculations. We then discussed existing neural network research used for electrical motor problems. After this, we presented the following eight contributions

- Data-driven modeling of electrical motor physics is a key contribution of this thesis. The input-output relationship between induction motor quantities is established using different neural networks. To perform time-series regression, a new network architecture named DiagBiRNN has been introduced that combines the benefits of convolutions and sequential layers.
- An important application of identified neural networks is estimating speed and torque based on currents and voltages. In this contribution, we addressed the application of DiagBiRNN to electrical motors and established a suitable pipeline for analyzing performance using electrical engineering metrics.
- DiagBiRNN can be utilized in several other electrical motor tasks like sensor fault recovery, mechanical fault detection, and temperature modeling. Another significant contribution of this thesis consists of using DiagBiRNN as a backbone in a GAN called Sceptic-GAN to identify sensor faults and recover from them.

- This contribution enables us to use the speed-torque estimator trained on a large amount of simulated data directly in real-world settings. This led to the introduction of a neural network-based time-series signal denoiser called meta-denoiser. Meta-denoiser takes real-world noisy currents and voltages and denoises them for the speed-torque estimator. This removes the necessity of collecting a large amount of real noisy data to train the speed-torque estimator network for real-world adaptation.
- DiagBiRNN based applications like speed-torque estimator, Scpetic-GAN, thermal modeling network, etc., are vulnerable to unforeseen variations in their input. This contribution is on the robustness analysis of such application networks using the modified versions of standard adversarial attackers. We showed why some existing methods are not realistic attackers for speed-torque estimation networks and established the requirement of a physical dynamics-based attacker.
- After establishing the need for a better adversarial attacker to understand the robustness of DiagBiRN-Skip applications, we briefly discussed its generalization. This contribution analyzes the generalization of the speed-torque estimator to a 90kW power motor. We found that more in-depth analysis is required to understand why speed estimation fails while torque works for this particular motor.
- We need to implement the speed-torque estimator and fault detection and recovery methods on computational devices that are low in computing and memory. To achieve this, we introduced subdifferential inclusion for sparsity (SIS), a convex optimization-based method to obtain a sparse representation of pre-trained neural networks. We demonstrate its effectiveness in three domains: vision, natural language processing, and speech.
- The proposed the SIS method designs sparse networks from a pre-trained version. Unlike many methods existing for network compression, our proposed approach is grounded on a sound optimization approach offering theoretical guarantees of convergence. In addition, we showed that it is very competitive with existing approaches.

6.2 . Future Work

The contributions of the thesis being summarised, a few possible future research directions based on the contributions and new concepts related to the work are mentioned in this section.

6.2.1 . Better Synthetic Data

This thesis presented a two-stage pipeline to leverage large simulated and small real-world data to train a speed-torque estimator for real-world applications. Such

dual step pipeline of transfer learning has been used widely in all domains of deep learning research. Better sophisticated methods have emerged which generate more realistic data to train deep neural networks [231]. The trained networks do not require fine-tuning on real-world data. Such methods have risen due to the complexity of obtaining labeling data in fields like particle physics [232] and medical science [233, 234]. Such techniques should be leveraged for problems like fault detection, where we have around a thousand different faults. There is no easy way of obtaining labeled data for all types of faults. Researchers have well-understood and established mathematical models of the fault and the system where they occur. These mathematical models can be leveraged for synthetic data generation.

6.2.2 . Transformers for System Fault Detection

Time series communities have also been captivated by the success of transformers in natural language processing, computer vision, and optimal control tasks. Time series modeling benefits from the ability of transformers to capture long-range dependencies and interactions, resulting in exciting advances in various time series applications. This thesis used a transformer variant called FedFormer [156] for the thermal modeling task. Similarly, we can modify existing time-series transformer architectures for electrical motor tasks like fault detection and recovery. For example, we can achieve better positional encoding using [235] to represent the temporal aspect of data. AutoFormer [236] and FedFormer [156] use timestamp encoding to achieve the same. It is also possible to modify attention modules to achieve efficient architectures since they are the bottleneck in the whole network and are dependent on the input size, which can be problematic in cases where long time series are involved. We can then train such transformers on large synthetic data containing all types of faults to obtain a foundational model [237] for fault detection in electrical motors.

6.2.3 . Reliability of the Neural Networks for Electrical Motor Tasks

In Chapter 4 we performed a robustness analysis of neural networks used in five different electrical motor tasks. We demonstrated that the existing attackers are not the most suitable for regression tasks like speed-torque estimation. In the future, it would be interesting to consider domain knowledge and sequential dynamics of motor data to generate better attacks and improve the robustness of neural networks using recent methods like CertViT [238] and Shrink & Cert [239]. It is also essential to understand the sensitivity of the individual inputs for neural network stability. In the case of multivariate regression, the partial Lipschitz constant has been used to analyze the contribution of individual input perturbations [143]. Such an analysis is essential in the context of electrical motors where different inputs might have different entropy and/or certain inputs might be more vulnerable to perturbations than others.

6.2.4 . Tensorized Pruning of Neural Networks

Transformers have garnered immense interest lately due to their effectiveness across various domains like language, vision, and reinforcement learning. However, these model training and inference costs have snowballed and become prohibitively expensive. The multi-head attention mechanism, as a critical component of the transformer, limits the effective deployment of the model to a resource-limited setting. It has been shown that tensor decomposition and parameter sharing can be used to obtain an efficient version of the self-attention model (multi-linear attention) [240]. We can extend sub-differential inclusion for sparsity (SIS) to get a low-rank tensor approximation of neural networks. Such a method will be very appropriate for finding efficient representations of transformers like AutoFormer [236] and FedFormer [156].

6.2.5 . Optimal Neural Networks for Industrial Devices

Although SIS obtains a sparse network from a pre-trained one, we may need more steps to realize our goal of running networks on very low compute devices. Network architecture search methods take a massive amount of experiments, long-tailed heuristics, and sophisticated engineering to obtain a network that meets the constraints of an industrial device. A possible direction for future improvements consist of extending the Once-for-All network proposed in [215] for neural networks and IoT devices used for electrical motor applications. Preliminary results obtained with this approach are presented in Appendix A

Appendix A
Optimal Neural Networks for IoT Devices

This appendix is about extending the once-for-all method allowing us to find efficient sub-networks. These sub-networks are obtained relatively cheaply and meet the criteria of a real-time processor and other application requirements like latency, energy, storage, precision, etc. More precisely, we introduce a practical way of obtaining an efficient version of DiagBiRNN for speed-torque estimation. The goal is to identify a sub-structure of DiagBiRNN that can maintain its original performance while running on a limited resource IoT device. We demonstrate results on six IoT devices with microcontrollers and microprocessors.

A.0.1 . Once-for-All Networks

A once-for-all (OFA) network supports diverse architectural settings by decoupling network training and efficient sub-network search, to reduce the overall cost of obtaining an efficient sub-network suitable for a targeted computational device. Assuming the weights of the once-for-all network as W_0 and the architectural configurations as $arch_i$, [215] formalize the problem as

$$\min_{W_0} \sum_{arch_i} \mathcal{L}_{val}(C(W_0, arch_i)), \tag{A.1}$$

where $C(W_0, arch_i)$ denotes a selection scheme that selects part of the model from the once-for-all networks W_0 to form a sub-network with architectural configuration $arch_i$. The overall training objective is to optimize W_0 to make each supported sub-networks maintain the same level of validation accuracy (or loss $\mathcal{L}_{val}$) as independently training a network with the same architectural configuration.

There are several ways of solving the above optimization problem. The best one is **progressive shrinking**. This method uses a training order where large sub-networks are trained first and then small sub-networks are trained progressively. For example, in DiagBiRNN, we start with training the largest network with the maximum kernel sizes for the convolutions in the encoder layer (e.g., conv1: 11,

conv2: 7, conv3: 5, and conv4: 3) (Figure 2.10). Then we progressively fine-tune the network to support smaller sub-networks by gradually adding them into the sampling space. Specifically, after training the largest network, we first support smaller kernel sizes {3, 5, 7, 9} depending on the convolutional layer. The details of the progressive shrinking for DiagBiRNN network used for speed-torque estimation are as follows:

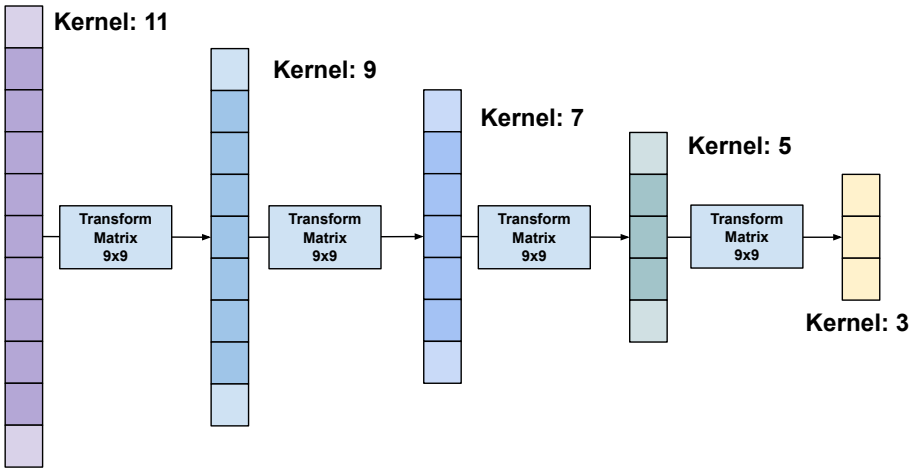


Figure A.1: Kernel transformation matrix for elastic kernel size in the first convolutional layer of DiagBiRNN.

- **Elastic Kernel Size** (Figure A.1): The center of the 1D convolution kernel of size 11 can also serve as a 9-sized kernel, the center of which can also be a 7-sized kernel, and so on. Therefore, the kernel size becomes *elastic*. The challenge is that the centering sub-kernels are shared and must play multiple roles (independent kernel and part of a large kernel). The weights of centered sub-kernels may need different distributions or magnitudes as different roles. Forcing them to be the same degrades the performance of some sub-networks. A kernel transformation matrix can convert one kernel to another sub-kernel. Within each 1D convolutional layer, the kernel transformation matrices are shared among different channels. We only need $9 \times 9 + 7 \times 7 + 5 \times 5 + 3 \times 3 = 164$ extra parameters to store the kernel transformation matrices in the first convolutional and decovlutional layers of DiagBiRNN.

- **Elastic Feature Width**: Feature width means the number of channels coming out of a convolutional and deconvlutional layer in the DiagBiRNN. We give each layer the flexibility to choose different channel expansion ratios. Following the progressive shrinking scheme, we first train a full-width network. Then we introduce a channel sorting operation to support partial widths. It reorganizes the channels according to their

importance, calculated based on the L1 norm of a channel's weight. For example, when shrinking from a 4-channel-layer to a 3-channel-layer, we select the most significant 3 channels, whose weights are shared with the 4-channel-layer. Smaller sub-networks are initialized with the most important channels on the once-for-all network, which is already well-trained. This channel sorting operation preserves the accuracy of larger sub-networks.

For training the full network, we use Adam optimizer [241]. The initial learning rate is 1e-3, and we use the cosine schedule for learning rate decay. The full network is trained for 150 epochs with batch size 2048. Training the once-for-all version of DiagBiRNN took around 200 GPU hours on 8 80GB A100 GPUs. Next, we show how one-time training costs can be amortized for many deployment scenarios.

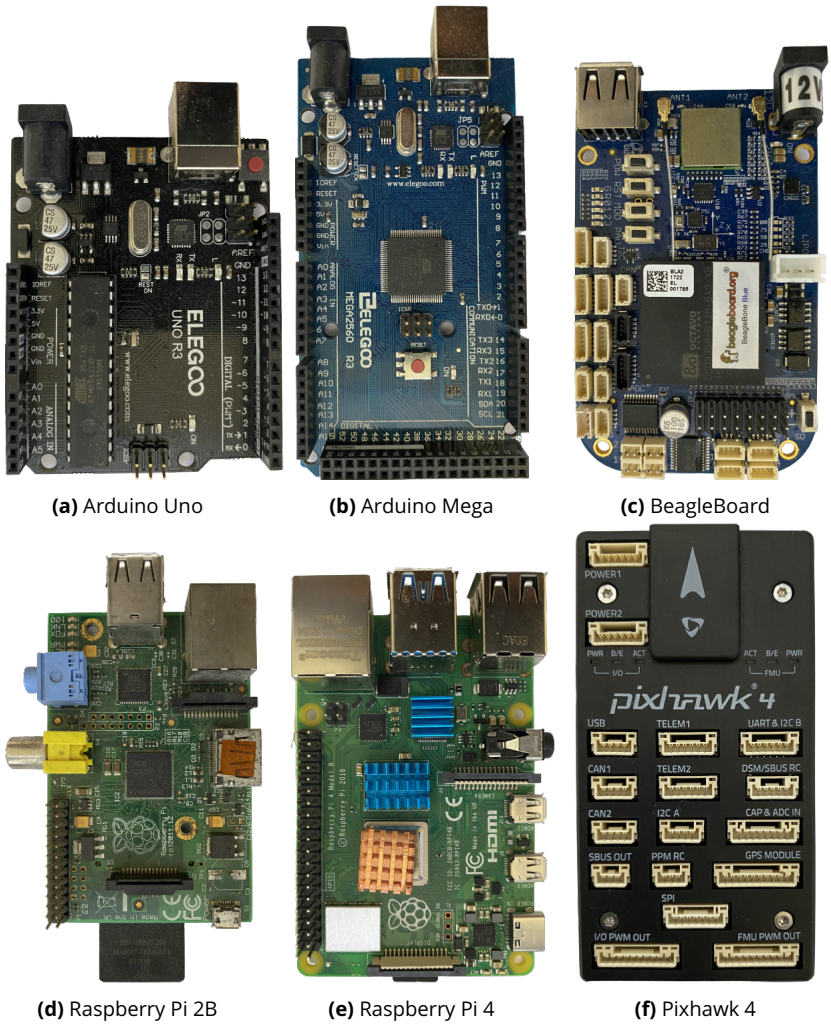
A.0.2 . Finding Best Fit Networks

Once a once-for-all DiagBiRNN has been trained using the progressive shrinking method, we can derive a sub-network for a given computational device and deployment constraints. This can be achieved by searching a neural network that satisfies the efficiency (e.g., latency, energy, RAM, CPU cycles) constraints on the target hardware while maintaining the original accuracy. We experiment with different types of IoT devices widely used as development kits in industrial and academic labs and hobby electronics. Table A.1 shows the processor type, CPU cycles, RAM, storage, and max power available in these IoT devices.

Board	Processor	CPU	RAM	Storage	Power
Arduino Uno	ATmega328P	16MHz	2KB	32KB	240 mW
Arduino Mega	ATmega2560	16MHz	8KB	256KB	240 mW
Pixhawk 4	Cortex-M7	216MHz	512KB	2MB	2.25W
BeagleBoard	Cortex-A8	1GHz	512MB	4GB	2W
Raspberry Pi 2B	Cortex-A7	900MHz	1GB	8GB	6.25W
Raspberry Pi 4	Cortex-A72	1.5GHz	8GB	32GB	6.4W

Table A.1: IoT devices computational and power specifications.

Figure A.2 shows all six IoT devices. All of them are powered using a standard 5V-2A power adapter. We convert DiagBiRNN weights from PyTorch to ONNX weights. We then use Tensorflow Lite to export it to run on microcontrollers. This is only required for Adruino Uno, Arduino Mega, BeagleBoard, and Pixhawk 4, as they have microcontrollers. In the case of Raspberry Pi 2B and Pi 4, we use executable object code running on the Linux kernel. We start sampling sub-networks by iteratively decreasing the elastic kernel and width to obtain a network that fits a given hardware device. For each sampled sub-network, we measure the accuracy on the test set from Section 2.5.3 and inference latency for a sample of duration 100 milliseconds. We stop the search when we have achieved a minimum



(a) Arduino Uno **(b)** Arduino Mega **(c)** BeagleBoard

(d) Raspberry Pi 2B **(e)** Raspberry Pi 4 **(f)** Pixhawk 4

Figure A.2: IoT devices used in the edge inference experiments.

latency of 20 milliseconds or crossed the mean absolute error (speed + torque) of 0.5 on the test set.

Board	Search Time (GPU Hours)	Latency (ms)	Params (x1000)	Speed (ω_r) MAE	Speed (ω_r) Smape	Torque (τ_{em}) MAE	Torque (τ_{em}) SMAPE
Arduino Uno	2.5	67	8	0.23	25.72%	0.21	43.72%
Arduino Mega	2.0	53	8	0.21	22.93%	0.19	41.83%
Pixhawk 4	1.7	23	8.7	0.19	21.28%	0.17	40.56%
BeagleBoard	1.5	21	9.2	0.11	19.45%	0.12	39.35%
Raspberry Pi 2B	0.5	12	19.2	0.09	25.72%	0.10	39.72%
Raspberry Pi 4	0.5	8	19.2	0.09	25.72%	0.10	39.72%

Table A.2: Efficient DiagBiRNN sub-networks sampled from once-for-all version.

Table A.2 shows test set results and latency of one inference sample for best

sub-networks obtained for different IoT devices. It should be noted that a single network was obtained for both types of Raspberry Pi. Microcontrollers like Arduino Uno and Mega get latency upwards of 53 ms with degradation in MAE and SMAPE. Pixhawk 4 and BeagleBoard can achieve a target latency of 20ms. Raspberry Pi variants being more compute-rich than other devices, go for minimal latency while achieving excellent speed-torque estimation performance. This shows that the once-for-all trained network can obtain hardware-targeted efficient versions with good original performance. This small experiment can help us deliver real-world deployable and usable applications of neural networks for electrical motor use cases.

Bibliography

[1] F. Mansour, "Induction motors: Construction, principle of operation, power and torque calculations, characteristics and speed control," 2020. (Cited on 13, 21)

[2] D. Kumar, "Operation of Induction Motor," https://engineeringlearn.com/operation-of-induction-motor/, 2021. (Cited on 13, 22)

[3] S. J. Campbell, *Solid-State AC Motor Controls*, 1987. (Cited on 19, 22)

[4] C. S. Sisking, *Electrial Control Systems in Industry*, 1978. (Cited on 19, 22)

[5] A. Hughes and B. Drury, *Electric Motors and Drives*, Newnes, 2013. (Cited on 21)

[6] J. Langford, R. Salakhutdinov, and T. Zhang, "Learning nonlinear dynamic models," in *International Conference on Machine Learning*, 2009. (Cited on 25, 26)

[7] E. Levin, "Modeling time varying systems using hidden control neural architecture," in *Neural Information Processing Systems*. 1991. (Cited on 25, 26, 27, 34)

[8] T. Huang, L. Song, and J. Schneider, "Learning nonlinear dynamic models from non-sequenced data," in *International Conference on Artificial Intelligence and Statistics*, 2010. (Cited on 25, 26)

[9] O. P. Ogunmolu, X. Gu, S. B. Jiang, and N. R. Gans, "Nonlinear systems identification using deep dynamic neural networks," *arXiv:1610.01439*, 2016. (Cited on 25)

[10] F. Jadot, F. Malrait, J. Moreno-Valenzuela, and R. Sepulchre, "Adaptive regulation of vector-controlled induction motors," *IEEE Transactions on Control Systems Technology*, vol. 17, pp. 646–657, 2009. (Cited on 25, 33)

[11] A. K. Jebai, P. Combes, F. Malrait, et al., "Energy-based modeling of electric motors," in *Conference on Decision and Control*, 2014. (Cited on 25)

[12] F. Malrait, A. K. Jebai, and K. Ejjabraoui, "Power conversion optimization for hydraulic systems controlled by variable speed drives," *Journal of Process Control*, vol. 74, pp. 133–146, 2019. (Cited on 25)

[13] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Data-driven discovery of partial differential equations," *Science Advances*, vol. 3, 2017. (Cited on 26)

[14] F. Carminati, G. K. M. Pierini, A. Farbin, et al., "Calorimetry with deep learning: Particle classification, energy regression, and simulation for high-energy physics," in *Deep Learning for Physical Sciences Workshop (NeurIPS)*, 2017. (Cited on 26, 34)

[15] G. Shi, X. Shi, M. O'Connell, et al., "Neural lander: Stable drone landing control using learned dynamics," in *International Conference on Robotics and Automation*, 2019. (Cited on 26, 34)

[16] B. Lusch, J. N. Kutz, and S. L. Brunton, "Deep learning for universal linear embeddings of nonlinear dynamics," *Nature Communications*, vol. 9, 2018. (Cited on 26, 27, 34)

[17] A. Karpatne, W. Watkins, J. Read, and V. Kumar, "How can physics inform deep learning methods in scientific problems?: Recent progress and future prospects," in *Deep Learning for Physical Sciences Workshop (NeurIPS)*, 2017. (Cited on 26, 34)

[18] R. T. D'Agnolo and A. Wulzer, "Learning new physics from a machine," *Physical Review*, 2019. (Cited on 26, 27)

[19] L. Ardizzone, J. Kruse, S. J. Wirkert, et al., "Analyzing inverse problems with invertible neural networks," *arXiv:1808.04730*, 2019. (Cited on 27)

[20] B. O. Koopman, "Hamiltonian systems and transformation in hilbert space," *National Academy of Sciences*, vol. 17, pp. 315–318, 1931. (Cited on 27)

[21] A. Karpatne, G. Atluri, J. H. Faghmous, et al., "Theory-guided data science: A new paradigm for scientific discovery from data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, pp. 2318–2331, 2017. (Cited on 27)

[22] A. Karpatne, W. Watkins, J. S. Read, and V. Kumar, "Physics-guided neural networks (pgnn): An application in lake temperature modeling," *arXiv:1710.11431*, 2017. (Cited on 27)

[23] M. Hermans, B. Schrauwen, P. Bienstman, and J. Dambre, "Automated design of complex dynamic systems," *PLoS ONE*, vol. 9, pp. e86696, 2014. (Cited on 27)

[24] J. Astola, P. Haavisto, and Y. Neuvo, "Vector median filters," *IEEE Proceedings*, vol. 78, pp. 678–689, 1990. (Cited on 27, 92)

[25] Y. Liu, "Noise reduction by vector median filtering," *GEOPHYSICS*, vol. 78, pp. 79–87, 2013. (Cited on 27, 92)

[26] B. Anderson and J. Moore, *Optimal Filtering*, Prentice-Hall, 1979. (Cited on 27, 92)

[27] K. R. Muske and T. F. Edgar, *Nonlinear State Estimation*, pp. 311–370, Prentice-Hall, 1997. (Cited on 27, 92, 95, 97, 98, 101)

[28] L. Ralaivola and F. d'Alche Buc, "Time series filtering, smoothing and learning using the kernel Kalman filter," in *International Joint Conference on Neural Networks*, 2005. (Cited on 27, 92)

[29] S. Banerjee and M. Mitra, "Application of cross wavelet transform for ecg pattern analysis and classification," *IEEE Transactions on Instrumentation and Measurement*, vol. 63, pp. 326–333, 2014. (Cited on 27, 92)

[30] Ç. P. Dautov and M. S. Özerdem, "Wavelet transform and signal denoising using wavelet method," in *Signal Processing and Communications Applications Conference*, 2018, pp. 1–4. (Cited on 27, 92, 97, 98, 101)

[31] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: Nonlinear Phenomena*, vol. 60, pp. 259–268, 1992. (Cited on 28, 92)

[32] I. Selesnick, "Total variation denoising via the Moreau envelope," *IEEE Signal Processing Letters*, vol. 24, pp. 216–220, 2017. (Cited on 28, 92)

[33] H. Du and Y. Liu, "Minmax-concave total variation denoising," *Signal, Image and Video Processing*, vol. 12, 2018. (Cited on 28, 92, 95, 97, 98, 101)

[34] P. Vincent, H. Larochelle, I. Lajoie, et al., "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010. (Cited on 28, 92, 97, 98, 101)

[35] C. Szegedy, W. Zaremba, I. Sutskever, et al., "Intriguing properties of neural networks," *arXiv:1312.6199*, 2013. (Cited on 28)

[36] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv:1412.6572*, 2014. (Cited on 28, 30, 92)

[37] A. Madry, A. Makelov, L. Schmidt, et al., "Towards deep learning models resistant to adversarial attacks," *arXiv:1706.06083*, 2017. (Cited on 28, 30, 92)

[38] V. Vapnik, *The Support Vector Method of Function Estimation*, pp. 55–85, 1998. (Cited on 28)

[39] P. L. Bartlett and S. Mendelson, "Rademacher and gaussian complexities: Risk bounds and structural results," in *Journal of Machine Learning Research*, 2001. (Cited on 28)

[40] S. Mukherjee, P. Niyogi, T. Poggio, and R. Rifkin, "Statistical learning: Stability is sufficient for generalization and necessary and sufficient for consistency of empirical risk minimization," 2004. (Cited on 28)

[41] O. Bousquet and A. Elisseeff, "Stability and generalization," *Journal of Machine Learning Research*, vol. 2, pp. 499–526, 2002. (Cited on 28)

[42] H. Mhaskar and T. Poggio, "Deep vs. shallow networks : An approximation theory perspective," *Analysis and Applications*, vol. 14, 2016. (Cited on 28)

[43] M. C. Mozer and P. Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment," in *Neural Information Processing Systems*, 1989. (Cited on 29, 120)

[44] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Neural Information Processing Systems*, 1990. (Cited on 29, 120)

[45] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *International Conference on Neural Networks*, 1993. (Cited on 29, 120)

[46] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Neural Information Processing Systems*, 2015. (Cited on 29, 120)

[47] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *International Conference on Machine Learning*, 2017. (Cited on 29, 120, 133)

[48] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Neural Information Processing Systems*, 2016. (Cited on 29, 120)

[49] S. Park, J. Lee, S. Mo, and J. Shin, "Lookahead: A far-sighted alternative of magnitude-based pruning," in *International Conference on Learning Representations*, 2020. (Cited on 29, 120)

[50] A. Renda, J. Frankle, and M. Carbin, "Comparing rewinding and fine-tuning in neural network pruning," in *International Conference on Learning Representations*, 2020. (Cited on 29, 120, 131, 133, 135)

[51] Y. Chauvin, "A back-propagation algorithm with optimal use of hidden units," in *Neural Information Processing Systems*, 1989. (Cited on 29, 120)

[52] M. A. Carreira-Perpiñán and Y. Idelbayev, ""learning-compression" algorithms for neural net pruning," in *Computer Vision and Pattern Recognition*, 2018. (Cited on 29, 120)

[53] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," in *International Conference on Learning Representations*, 2017. (Cited on 29, 120)

[54] K. Neklyudov, D. Molchanov, A. Ashukha, and D. Vetrov, "Structured bayesian pruning via log-normal multiplicative noise," in *Neural Information Processing Systems*, 2017. (Cited on 29, 120)

[55] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations*, 2019. (Cited on 29, 120)

[56] N. Lee, T. Ajanthan, and P. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," in *International Conference on Learning Representations*, 2019. (Cited on 29, 120, 131, 133, 135)

[57] N. Lee, T. Ajanthan, S. Gould, and P. H. S. Torr, "A signal propagation perspective for pruning neural networks at initialization," in *International Conference on Learning Representations*, 2020. (Cited on 29, 120)

[58] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Computer Vision and Pattern Recognition*, 2018. (Cited on 29, 121)

[59] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Association for the Advancement of Artificial Intelligence*, 2019. (Cited on 29, 121)

[60] H. Cai, J. Yang, W. Zhang, et al., "Path-level network transformation for efficient architecture search," *arXiv:1806.02639*, 2018. (Cited on 29, 121)

[61] B. Wu, X. Dai, P. Zhang, et al., "FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Computer Vision and Pattern Recognition*, 2019. (Cited on 29, 121)

[62] M. Tan, B. Chen, R. Pang, et al., "Mnasnet: Platform-aware neural architecture search for mobile," in *Computer Vision and Pattern Recognition*, 2019. (Cited on 29, 121)

[63] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *International Conference on Learning Representations*, 2019. (Cited on 29, 121)

[64] A. Howard, M. Sandler, G. Chu, et al., "Searching for mobilenetv3," in *International Conference on Computer Vision*, 2019. (Cited on 29, 121, 135)

[65] H. Cai, C. Gan, T. Wang, et al., "Once-for-all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, 2020. (Cited on 29)

[66] J. Lin, W.-M. Chen, Y. Lin, et al., "Mcunet: Tiny deep learning on iot devices," in *Neural Information Processing Systems*, 2020. (Cited on 29, 121)

[67] S. Verma and K. Gupta, "Robustness of neural networks used in electrical motor time-series," in *NeurIPS Workshop on Robustness in Sequence Modeling*, 2022. (Cited on 31)

[68] S. Verma, N. Henwood, M. Castella, et al., "Can GANs recover faults in electrical motor sensors?," in *ICLR Workshop on Deep Generative Models for Highly Structured Data*, 2022. (Cited on 31)

[69] S. Verma, N. Henwood, M. Castella, et al., "Neural speed-torque estimator for induction motors in the presence of measurement noise," *IEEE Transactions on Industrial Electronics*, 2022. (Cited on 32)

[70] S. Verma and J.-C. Pesquet, "Sparsifying networks via subdifferential inclusion," in *International Conference on Machine Learning*, 2021. (Cited on 32)

[71] S. Verma, N. Henwood, M. Castella, et al., "Neural networks based speed-torque estimators for induction motors and performance metrics," in *IEEE Industrial Electronics Society*, 2020. (Cited on 32)

[72] S. Verma, N. Henwood, M. Castella, et al., "Modeling electrical motor dynamics using encoder-decoder with recurrent skip connection," in *Association for the Advancement of Artificial Intelligence*, 2020. (Cited on 32)

[73] N. Lassau, S. Ammari, E. Chouzenoux, et al., "Integrating deep learning CT-scan model, biological and clinical variables to predict severity of COVID-19 patients," *Nature Communications*, vol. 12, 2021. (Cited on 32)

[74] M. Hermans and B. Schrauwen, "Training and analysing deep recurrent neural networks," in *Neural Information Processing Systems*, 2013. (Cited on 34)

[75] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *European Symposium on Artificial Neural Networks*, 2015. (Cited on 34)

[76] S. Li, W. Li, C. Cook, et al., "Independently recurrent neural network (IndRNN): Building a longer and deeper RNN," in *Computer Vision and Pattern Recognition*, 2018. (Cited on 34)

[77] J. Miller and M. Hardt, "Stable recurrent models," in *International Conference on Learning Representations*, 2019. (Cited on 34, 35)

[78] A. Ghaderi, B. Sanandaji, and F. Ghaderi, "Deep forecast: Deep learning-based spatio-temporal forecasting," in *Time Series Workshop in International Conference on Machine Learning*, 2017. (Cited on 35)

[79] J. Yoon, W. Zame, and M. Schaar, "Multi-directional recurrent neural networks: A novel method for estimating missing data," in *Time Series Workshop in International Conference on Machine Learning*, 2017. (Cited on 35, 79)

[80] N. Laptev, J. Yosinski, L. Li, and S. Smyl, "Time-series extreme event forecasting with neural networks at uber," in *Time Series Workshop in International Conference on Machine Learning*, 2017. (Cited on 35)

[81] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv:1803.01271*, 2018. (Cited on 35)

[82] E. Aksan and O. Hilliges, "STCN: Stochastic temporal convolutional networks," in *International Conference on Learning Representations*, 2019. (Cited on 35)

[83] P. F. Christoffersen and F. X. Diebold, "Optimal prediction under asymmetric loss," *Econometric Theory*, vol. 13, pp. 808–817, 1997. (Cited on 35)

[84] J. H. Stock and M. W. Watson, "A comparison of linear and nonlinear univariate models for forecasting macroeconomic time series," Tech. Rep., National Bureau of Economic Research, 1998. (Cited on 35)

[85] M. Iacopini, F. Ravazzolo, and L. Rossini, "Proper scoring rules for evaluating density forecasts with asymmetric loss functions," *Journal of Business & Economic Statistics*, pp. 1–15, 2022. (Cited on 35)

[86] G. P. Zhang, "Avoiding pitfalls in neural network research," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 37, pp. 3–16, 2007. (Cited on 35)

[87] O. Intrator and N. Intrator, "Interpreting neural-network results: a simulation study," *Computational Statistics & Data Analysis*, vol. 37, pp. 373–393, 2001. (Cited on 35)

[88] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, pp. 1–58, 1992. (Cited on 35)

[89] X. Mao, C. Shen, and Y.-B. Yang, "Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections," in *Neural Information Processing Systems*, 2016. (Cited on 38)

[90] A. C. Cameron and F. A. Windmeijer, "An R-squared measure of goodness of fit for some common nonlinear regression models," *Journal of Econometrics*, vol. 77, pp. 329–342, 1997. (Cited on 41, 112)

[91] K.-S. Lee and J.-S. Ryu, "Instrument fault detection and compensation scheme for direct torque controlled induction motor drives," *Control Theory and Applications*, vol. 150, pp. 376–382, 2003. (Cited on 70)

[92] S. Fan and J. Zou, "Sensor fault detection and fault tolerant control of induction motor drivers for electric vehicles," in *International Power Electronics and Motion Control Conference*, 2012. (Cited on 70)

[93] L. Jiang, "Sensor fault detection and isolation using system dynamics identification techniques.," 2011. (Cited on 70)

[94] R. Isermann, *Fault-Diagnosis Systems From Fault Detection to Fault Tolerance*, vol. 28, 2006. (Cited on 70)

[95] G. Buja and M. Kazmierkowski, "Direct torque control of PWM inverter-fed AC motors - a survey," *IEEE Transactions on Industrial Electronics*, vol. 51, pp. 744–757, 2004. (Cited on 70)

[96] T. Orlowska-Kowalska and M. Dybkowski, "Stator-current-based MRAS estimator for a wide range speed-sensorless induction-motor drive," *IEEE Transactions on Industrial Electronics*, vol. 57, pp. 1296–1308, 2010. (Cited on 70)

[97] K. S. Gaeid, H. W. Ping, M. Khalid, and A. Masaoud, "Sensor and sensorless fault tolerant control for induction motors using a wavelet index," *Sensors*, vol. 12, pp. 4031–4050, 2012. (Cited on 70)

[98] H. Ben Zina, M. Allouche, M. Chaabane, and M. Souissi, "Sensor fault tolerant control for induction motor," in *Sciences and Techniques of Automatic Control and Computer Engineering*, 2015, pp. 252–256. (Cited on 70)

[99] D. Diallo, M. Benbouzid, and A. Makouf, "A fault-tolerant control architecture for induction motor drives in automotive applications," *IEEE Transactions on Vehicular Technology*, vol. 53, pp. 1847–1855, 2004. (Cited on 70)

[100] D. Kreindler and C. J. Lumsden, "The effects of the irregular sample and missing data in time series analysis," *Nonlinear dynamics, psychology, and life sciences*, vol. 10, pp. 187–214, 2006. (Cited on 70)

[101] D. Mondal and D. Percival, "Wavelet variance analysis for gappy time series," *Annals of the Institute of Statistical Mathematics*, vol. 62, pp. 943–966, 2008. (Cited on 70)

[102] D. Rubin, *Multiple imputation for nonresponse in surverys*, vol. 81, John Wiley & Sons, 2004. (Cited on 70)

[103] P. J. García-Laencina, J.-L. Sancho-Gómez, and A. R. Figueiras-Vidal, "Pattern classification with missing data: a review," *Neural Computing and Applications*, vol. 19, pp. 263–282, 2009. (Cited on 70)

[104] I. White, P. Royston, and A. Wood, "Multiple imputation using chained equations: Issues and guidance for practice.," *Statistics in medicine*, vol. 30 4, pp. 377–99, 2011. (Cited on 70, 79)

[105] R. Mazumder, T. Hastie, and R. Tibshirani, "Spectral regularization algorithms for learning large incomplete matrices," *Journal of Machine Learning Research*, vol. 11, pp. 2287–2322, 2010. (Cited on 70)

[106] H.-F. Yu, N. Rao, and I. S. Dhillon, "Temporal regularized matrix factorization for high-dimensional time series prediction," in *Neural Information Processing Systems*, 2016. (Cited on 70)

[107] T. Schnabel, A. Swaminathan, A. Singh, et al., "Recommendations as treatments: Debiasing learning and evaluation," in *International Conference on Machine Learning*, 2016. (Cited on 70)

[108] M. Berglund, T. Raiko, M. Honkala, et al., "Bidirectional recurrent neural networks as generative models," in *Neural Information Processing Systems*, 2015. (Cited on 70)

[109] Z. Che, S. Purushotham, K. Cho, et al., "Recurrent neural networks for multivariate time series with missing values," *Scientific Reports*, vol. 8, 2018. (Cited on 70, 79)

[110] Z. C. Lipton, D. Kale, and R. Wetzel, "Directly modeling missing data in sequences with RNNs: Improved classification of clinical time series," in *Machine Learning for Healthcare Conference*, 2016. (Cited on 70)

[111] E. Choi, T. Bahadori, and J. Sun, "Doctor AI: Predicting clinical events via recurrent neural networks," *JMLR workshop and conference proceedings*, vol. 56, 2015. (Cited on 70)

[112] J. Yoon, D. Jarrett, and M. van der Schaar, "Time-series generative adversarial networks," in *Neural Information Processing Systems*, 2019. (Cited on 71)

[113] J. Yoon, J. Jordon, and M. van der Schaar, "GAIN: Missing data imputation using generative adversarial nets," in *International Conference on Machine Learning*, 2018. (Cited on 71, 79)

[114] Y. Luo, X. Cai, Y. Zhang, et al., "Multivariate time series imputation with generative adversarial networks," in *Neural Information Processing Systems*, 2018. (Cited on 71, 79)

[115] Y. Luo, Y. Zhang, X. Cai, and X. Yuan, "E^2gan: End-to-end generative adversarial network for multivariate time series imputation," in *International Joint Conference on Artificial Intelligence*, 2019. (Cited on 71, 79, 80)

[116] S. Gandhi, T. Oates, T. Mohsenin, and D. Hairston, "Denoising time series data using asymmetric generative adversarial networks," in *Advances in Knowledge Discovery and Data Mining*, 2018, pp. 285–296. (Cited on 71)

[117] E. Balaban, A. Saxena, P. Bansal, et al., "Modeling, detection, and disambiguation of sensor faults for aerospace applications," *IEEE Sensors*, vol. 9, pp. 1907–1917, 2009. (Cited on 77)

[118] W. Kirchgässner, O. Wallscheid, and J. Böcker, "Estimating electric motor temperatures with deep residual machine learning," *IEEE Transactions on Power Electronics*, vol. 36, pp. 7480–7488, 2021. (Cited on 78, 88)

[119] A. Elly Treml, R. Andrade Flauzino, M. Suetake, and N. A. Ravazzoli Maciejewski, "Experimental database for detecting and diagnosing rotor broken bar in a three-phase induction motor.," 2020. (Cited on 78)

[120] N. A. R. Maciejewski, A. E. Treml, and R. A. Flauzino, "A systematic review of fault detection and diagnosis methods for induction motors," in *International Conference on Electrical Engineering*, 2020. (Cited on 78)

[121] A. T. Hudak, N. L. Crookston, J. S. Evans, et al., "Nearest neighbor imputation of species-level, plot-scale forest structure attributes from lidar data," *Remote Sensing of Environment*, vol. 112, pp. 2232–2245, 2008. (Cited on 79)

[122] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Mørup, "Scalable tensor factorizations with missing data," in *International Conference on Data Mining*, 2010. (Cited on 79)

[123] W. Cao, D. Wang, J. Li, et al., "BRITS: Bidirectional recurrent imputation for time series," in *Neural Information Processing Systems*, 2018. (Cited on 79)

[124] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *arXiv:1701.07875*, 2017. (Cited on 80)

[125] S. Qin and W. Li, "Detection and identification of faulty sensors with maximized sensitivity," in *American Control Conference*, 1999. (Cited on 85)

[126] Magtrol, "TM series in-line torque transducers," 2021. (Cited on 85)

[127] D. Zimmerman and T. Lyde, "Sensor failure detection and isolation in flexible structures using the eigensystem realization algorithm," in *Structural Dynamics and Materials Conference*, 1992. (Cited on 85)

[128] D. Wang, Y. Fotinich, and G. P. Carman, "Influence of temperature on the electromechanical and fatigue behavior of piezoelectric ceramics," *Journal of Applied Physics*, vol. 83, pp. 5342–5350, 1998. (Cited on 85)

[129] K. Goebel and W. Yan, "Correcting sensor drift and intermittency faults with data fusion and automated learning," *IEEE Systems Journal*, vol. 2, pp. 189–197, 2008. (Cited on 85)

[130] P.-J. Lu and T.-C. Hsu, "Application of autoassociative neural network on gas-path sensor data validation," *Journal of Propulsion and Power*, vol. 18, pp. 879–888, 2002. (Cited on 85)

[131] D. C. Zimmerman and T. L. Lyde, "Sensor failure detection and isolation in flexible structures using system realization redundancy," *Journal of Guidance, Control, and Dynamics*, vol. 16, pp. 490–497, 1993. (Cited on 85)

[132] X. Zhang, G. Foo, M. Don Vilathgamuwa, et al., "Sensor fault detection, isolation and system reconfiguration based on extended Kalman filter for induction motor drives," *IET Electric Power Applications*, vol. 7, pp. 607–617, 2013. (Cited on 85)

[133] Z. Q. Zhu, X. Zhu, P. D. Sun, and D. Howe, "Estimation of winding resistance and pm flux-linkage in brushless ac machines by reduced-order extended Kalman filter," in *Conference on Networking, Sensing and Control*, 2007. (Cited on 92)

[134] G. H. B. Foo, X. Zhang, and D. M. Vilathgamuwa, "A sensor fault detection and isolation method in interior permanent-magnet synchronous motor drives based on an extended Kalman filter," *IEEE Transactions on Industrial Electronics*, vol. 60, pp. 3485–3495, 2013. (Cited on 92)

[135] K. Naveed, B. Shaukat, and N. U. Rehman, "Dual tree complex wavelet transform-based signal denoising method exploiting neighbourhood dependencies and goodness-of-fit test," *Royal Society Open Science*, vol. 5, 2018. (Cited on 92)

[136] A. Kurakin, I. Goodfellow, S. Bengio, et al., "Adversarial examples in the physical world," 2016. (Cited on 92)

[137] Y. Dong, F. Liao, T. Pang, et al., "Boosting adversarial attacks with momentum," in *Computer Vision and Pattern Recognition*, 2018. (Cited on 92)

[138] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Computer Vision and Pattern Recognition*, 2016. (Cited on 92)

[139] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Computer Vision and Pattern Recognition*, 2017. (Cited on 93)

[140] E. R. Balda, A. Behboodi, and R. Mathar, "Perturbation analysis of learning algorithms: A unifying perspective on generation of adversarial examples," *arXiv:1812.07385*, 2018. (Cited on 93)

[141] K. Gupta, B. Pesquet-Popescu, F. Kaakai, and J.-C. Pesquet, "A quantitative analysis of the robustness of neural networks for tabular data," in *International Conference on Acoustics, Speech and Signal Processing*, 2021, pp. 8057–8061. (Cited on 93)

[142] K. Gupta, J.-C. Pesquet, B. Pesquet-Popescu, et al., "An adversarial attacker for neural networks in regression problems," in *IJCAI Workshop on Artificial Intelligence Safety (AI Safety)*, 2021. (Cited on 93)

[143] K. Gupta, F. Kaakai, B. Pesquet-Popescu, et al., "Multivariate lipschitz analysis of the stability of neural networks," *Frontiers in Signal Processing*, 2022. (Cited on 93, 141)

[144] K. Gupta, F. Kaakai, B. Pesquet-Popescu, and J.-C. Pesquet, "Safe design of stable neural networks for fault detection in small uavs," in *Computer Safety, Reliability, and Security Workshop*, 2022, pp. 263–275. (Cited on 93)

[145] H. Xu and S. Mannor, "Robustness and generalization," *Machine Learning*, vol. 86, pp. 391–423, 2011. (Cited on 93)

[146] G. K. Dziugaite and D. M. Roy, "Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data," *arXiv:1703.11008*, 2017. (Cited on 93)

[147] B. Neyshabur, Z. Li, S. Bhojanapalli, et al., "Towards understanding the role of over-parametrization in generalization of neural networks," *arXiv:1805.12076*, 2019. (Cited on 93)

[148] Y. Cao and Q. Gu, "Generalization error bounds of gradient descent for learning over-parameterized deep relu networks," in *Association for the Advancement of Artificial Intelligence*, 2020. (Cited on 93)

[149] H. Zhang, H. Singh, M. Ghassemi, and S. Joshi, ""why did the model fail?": Attributing model performance changes to distribution shifts," *arXiv:2210.10769*, 2022. (Cited on 93)

[150] J. Amjad, Z. Lyu, and M. R. Rodrigues, "Regression with deep neural networks: Generalization error guarantees, learning algorithms, and regularizers," in *European Signal Processing Conference*, 2021. (Cited on 93)

[151] N. Henwood, J. Malaizé, and L. Praly, "PMSM identification for automotive applications: Cancellation of position sensor errors," in *IEEE Industrial Electronics Society*, 2011, pp. 687–692. (Cited on 93)

[152] H. A. Chipman, E. D. Kolaczyk, and R. E. McCulloch, "Adaptive Bayesian wavelet shrinkage," *Journal of the American Statistical Association*, vol. 92, pp. 1413–1421, 1997. (Cited on 95)

[153] E. Zerdali, "Adaptive extended kalman filter for speed-sensorless control of induction motors," *IEEE Transactions on Energy Conversion*, vol. 34, pp. 789–800, 2019. (Cited on 95)

[154] K. P. Murphy, *Probabilistic Machine Learning: An introduction*, MIT Press, 2021. (Cited on 96)

[155] M. Lavielle and E. Moulines, "Least-squares estimation of an unknown number of shifts in a time series," *Journal of Time Series Analysis*, vol. 21, pp. 33–59, 2000. (Cited on 96)

[156] T. Zhou, Z. Ma, Q. Wen, et al., "Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting," *arXiv:2201.12740*, 2022. (Cited on 111, 141, 142)

[157] B. Shi, X. Bai, and C. 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*, vol. 39, pp. 2298–2304, 2017. (Cited on 111)

[158] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Computer Vision and Pattern Recognition*, 2016. (Cited on 111)

[159] I. Radosavovic, R. P. Kosaraju, R. Girshick, et al., "Designing network design spaces," in *Computer Vision and Pattern Recognition*, 2020. (Cited on 111)

[160] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015. (Cited on 119)

[161] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Computer Vision and Pattern Recognition*, 2016. (Cited on 119)

[162] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," in *Computer Vision and Pattern Recognition*, 2017. (Cited on 119)

[163] A. Hannun, C. Case, J. Casper, et al., "Deepspeech: Scaling up end-to-end speech recognition," *arXiv:1412.5567*, 2014. (Cited on 119)

[164] L. Dong, S. Xu, and B. Xu, "Speech-transformer: A no-recurrence sequence-to-sequence model for speech recognition," in *International Conference on Acoustics, Speech, and Signal Processing*, 2018. (Cited on 119)

[165] J. Li, V. Lavrukhin, B. Ginsburg, et al., "Jasper: An end-to-end convolutional neural acoustic model," in *Interspeech*, 2019. (Cited on 119, 120)

[166] S. Watanabe, T. Hori, S. Karita, et al., "Espnet: End-to-end speech processing toolkit," in *Interspeech*, 2018. (Cited on 119)

[167] T. Hayashi, R. Yamamoto, K. Inoue, et al., "ESPnet-TTS: Unified, reproducible, and integratable open source end-to-end text-to-speech toolkit," *arXiv:1910.10909*, 2019. (Cited on 119)

[168] H. Inaguma, S. Kiyono, K. Duh, et al., "ESPnet-ST: All-in-one speech translation toolkit," *arXiv:2004.10234*, 2020. (Cited on 119)

[169] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova, "Well-read students learn better: On the importance of pre-training compact models," *arXiv:1908.08962v2*, 2019. (Cited on 119)

[170] A. Radford, J. Wu, R. Child, et al., "Language models are unsupervised multitask learners," 2019. (Cited on 119)

[171] Z. Dai, Z. Yang, Y. Yang, et al., "Transformer-XL: Attentive language models beyond a fixed-length context," in *ACL*, 2019. (Cited on 119, 120, 136)

[172] T. B. Brown, B. P. Mann, N. Ryder, et al., "Language models are few-shot learners," *arXiv:2005.14165*, 2020. (Cited on 119, 120)

[173] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting," in *International Conference on Learning Representations*, 2020. (Cited on 119)

[174] A. Kusupati, V. Ramanujan, R. Somani, et al., "Soft threshold weight reparameterization for learnable sparsity," in *International Conference on Machine Learning*, 2020. (Cited on 119, 120, 131, 133)

[175] A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg, "Net-trim: Convex pruning of deep neural networks with performance guarantee," in *NeuIPS*, 2017. (Cited on 119, 120, 133)

[176] J.-J. Moreau, "Fonctions convexes duales et points proximaux dans un espace hilbertien," *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, 1962. (Cited on 119, 122)

[177] P. L. Combettes and J.-C. Pesquet, "Deep neural network structures solving variational inequalities," *SVVA*, 2020. (Cited on 119, 121, 127, 130)

[178] P. L. Combettes and Z. C. Woodstock, "A fixed point framework for recovering signals from nonlinear transformations," in *European Conference on Signal Processing*, 2021. (Cited on 119)

[179] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *British Machine Vision Conference*, 2014. (Cited on 120)

[180] K. A. vahid, A. Prabhu, A. Farhadi, and M. Rastegari, "Butterfly transform: An efficient fft based neural architecture design," in *Computer Vision and Pattern Recognition*, 2020. (Cited on 120)

[181] Z. Lu, V. Sindhwani, and T. N. Sainath, "Learning compact recurrent neural networks," in *International Conference on Acoustics, Speech, and Signal Processing*, 2016. (Cited on 120)

[182] W. Wen, C. Wu, Y. Wang, et al., "Learning structured sparsity in deep neural networks," in *Neural Information Processing Systems*, 2016. (Cited on 120)

[183] H. Li, A. Kadav, I. Durdanovic, et al., "Pruning filters for efficient convnets," *arXiv:1608.08710*, 2017. (Cited on 120)

[184] J. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *International Conference on Computer Vision*, 2017. (Cited on 120)

[185] J. Yu, L. Yang, N. Xu, et al., "Slimmable neural networks," in *International Conference on Learning Representations*, 2019. (Cited on 120)

[186] L. Liebenwein, C. Baykal, H. Lang, et al., "Provable filter pruning for efficient neural networks," in *International Conference on Learning Representations*, 2020. (Cited on 120)

[187] C. Louizos, K. Ullrich, and M. Welling, "Bayesian compression for deep learning," in *Neural Information Processing Systems*, 2017, pp. 3290–3300. (Cited on 120, 133)

[188] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through l_0 regularization," in *International Conference on Learning Representations*, 2018. (Cited on 120, 133)

[189] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," *arXiv:1710.01878*, 2018. (Cited on 120)

[190] Z. Liu, M. Sun, T. Zhou, et al., "Rethinking the value of network pruning," in *International Conference on Learning Representations*, 2019. (Cited on 120)

[191] P. Savarese, H. Silva, and M. Maire, "Winning the lottery with continuous sparsification," *arXiv:arXiv:1912.04427*, 2020. (Cited on 120)

[192] Y. Lee, "Differentiable sparsification for deep neural networks," *arXiv:1910.03201*, 2019. (Cited on 120, 133)

[193] X. Xiao, Z. Wang, and S. Rajasekaran, "Autoprune: Automatic network pruning by regularizing auxiliary parameters," in *Neural Information Processing Systems*, 2019. (Cited on 120)

[194] K. Azarian, Y. Bhalgat, J. Lee, and T. Blankevoort, "Learned threshold pruning," *arXiv:2003.00075*, 2020. (Cited on 120)

[195] G. Bellec, D. Kappel, W. Maass, and R. Legenstein, "Deep rewiring: Training very sparse deep networks," in *International Conference on Learning Representations*, 2018. (Cited on 120)

[196] D. C. Mocanu, E. Mocanu, P. Stone, et al., "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nature Communications*, 2018. (Cited on 120)

[197] X. Dai, H. Yin, and N. K. Jha, "NeST: A neural network synthesis tool based on a grow-and-prune paradigm," *IEEE TC*, 2019. (Cited on 120)

[198] T. Lin, S. U. Stich, L. Barba, et al., "Dynamic model pruning with feedback," in *International Conference on Learning Representations*, 2020. (Cited on 120)

[199] H. Mostafa and X. Wang, "Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization," in *International Conference on Machine Learning*, 2019. (Cited on 120)

[200] T. Dettmers and L. Zettlemoyer, "Sparse networks from scratch: Faster training without losing performance," *arXiv:1907.04840*, 2020. (Cited on 120)

[201] U. Evci, E. Elsen, P. Castro, and T. Gale, "Rigging the lottery: Making all tickets winners," in *International Conference on Machine Learning*, 2020. (Cited on 120, 131, 133, 135)

[202] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," in *International Conference on Learning Representations*, 2020. (Cited on 120, 131, 133)

[203] M. Ye, C. Gong, L. Nie, et al., "Good subnetworks provably exist: Pruning via greedy forward selection," in *International Conference on Machine Learning*, 2020. (Cited on 120)

[204] H. Tanaka, D. Kunin, D. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," *arXiv:2006.05467*, 2020. (Cited on 120, 131, 133)

[205] P. Jorge, A. Sanyal, H. Behl, et al., "Progressive skeletonization: Trimming more fat from a network at initialization," *arXiv:2006.09081*, 2020. (Cited on 120, 131, 133)

[206] A. Brock, T. Lim, J. Ritchie, and N. Weston, "SMASH: One-shot model architecture search through hypernetworks," in *International Conference on Learning Representations*, 2018. (Cited on 121)

[207] R. Miikkulainen, J. Z. Liang, E. Meyerson, et al., "Evolving deep neural networks," *arXiv:1703.00548*, 2017. (Cited on 121)

[208] E. Real, S. Moore, A. Selle, et al., "Large-scale evolution of image classifiers," in *International Conference on Machine Learning*, 2017. (Cited on 121)

[209] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, pp. 99–127, 2002. (Cited on 121)

[210] C. Szegedy, W. Liu, Y. Jia, et al., "Going deeper with convolutions," in *Computer Vision and Pattern Recognition*, 2015. (Cited on 121)

[211] H. Cai, T. Chen, W. Zhang, et al., "Reinforcement learning for architecture search by network transformation," *arXiv:1707.04873*, 2017. (Cited on 121)

[212] T. Chen, I. J. Goodfellow, and J. Shlens, "Net2net: Accelerating learning via knowledge transfer," *arXiv:1511.05641*, 2016. (Cited on 121)

[213] A. Ashok, N. Rhinehart, F. Beainy, and K. M. Kitani, "N2n learning: Network to network compression via policy gradient reinforcement learning," in *International Conference on Learning Representations*, 2018. (Cited on 121)

[214] Y. He and S. Han, "ADC: automated deep compression and acceleration with reinforcement learning," *arXiv:1802.03494*, 2018. (Cited on 121)

[215] H. Cai, C. Gan, T. Wang, et al., "Once for all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, 2020. (Cited on 121, 142, 143)

[216] S. K. Esser, J. L. McKinstry, D. Bablani, et al., "Learned step size quantization," in *International Conference on Learning Representations*, 2020. (Cited on 121)

[217] C. Xu, J. Yao, Z. Lin, et al., "Alternating multi-bit quantization for recurrent neural networks," in *International Conference on Learning Representations*, 2018. (Cited on 121)

[218] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *European Conference on Computer Vision*, 2018. (Cited on 121)

[219] Y. Li, R. Gong, X. Tan, et al., "{BRECQ}: Pushing the limit of post-training quantization by block reconstruction," in *International Conference on Learning Representations*, 2021. (Cited on 121)

[220] M. Nagel, R. A. Amjad, M. Van Baalen, et al., "Up or down? Adaptive rounding for post-training quantization," in *International Conference on Machine Learning*, 2020. (Cited on 121)

[221] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*, 2016. (Cited on 121)

[222] P. Stock, A. Joulin, R. Gribonval, et al., "And the bit goes down: Revisiting the quantization of neural networks," in *International Conference on Learning Representations*, 2020. (Cited on 121)

[223] P. L. Combettes and J.-C. Pesquet, "Lipschitz Certificates for Layered Network Structures Driven by Averaged Activation Operators," *SIAM Journal on Mathematics of Data Science*, 2020. (Cited on 121, 127)

[224] P.-L. Lions and B. Mercier, "Splitting algorithms for the sum of two nonlinear operators," *SIAM Journal on Numerical Analysis*, 1979. (Cited on 123)

[225] P. L. Combettes and J.-C. Pesquet, "A Douglas–Rachford splitting approach to nonsmooth convex variational signal recovery," *IEEE Journal of Selected Topics in Signal Processing*, 2007. (Cited on 123, 124)

[226] P. L. Combettes, "A block-iterative surrogate constraint splitting method for quadratic signal recovery," *IEEE Transactions on Signal Processing*, 2003. (Cited on 126)

[227] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 129–146, 2020. (Cited on 131)

[228] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *International Conference on Acoustics, Speech, and Signal Processing*, 2015. (Cited on 136)

[229] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," in *International Conference on Learning Representations*, 2017. (Cited on 136)

[230] Z. Li, E. Wallace, S. Shen, et al., "Train large, then compress: Rethinking model size for efficient training and inference of transformers," *arXiv:2002.11794*, 2020. (Cited on 136)

[231] S. I. Nikolenko, "Synthetic data for deep learning," *Synthetic Data for Deep Learning*, 2021. (Cited on 141)

[232] M. D. Schwartz, "Modern machine learning and particle physics," *Harvard Data Science Review*, 2021. (Cited on 141)

[233] R. Chen, M. Lu, T. Chen, et al., "Synthetic data in machine learning for medicine and healthcare," *Nature Biomedical Engineering*, vol. 5, pp. 1–5, 2021. (Cited on 141)

[234] A. Zhang, L. Xing, J. Zou, and J. Wu, "Shifting machine learning for healthcare from development to deployment and from models to data," *Nature Biomedical Engineering*, pp. 1–16, 2022. (Cited on 141)

[235] G. Zerveas, S. Jayaraman, D. Patel, et al., "A transformer-based framework for multivariate time series representation learning," *Knowledge Discovery & Data Mining*, 2021. (Cited on 141)

[236] H. Wu, J. Xu, J. Wang, and M. Long, "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting," in *Neural Information Processing Systems*, 2021. (Cited on 141, 142)

[237] R. Bommasani, D. A. Hudson, E. Adeli, et al., "On the opportunities and risks of foundation models," *ArXiv*, vol. abs/2108.07258, 2021. (Cited on 141)

[238] K. Gupta and S. Verma, "CertViT: Certified Robustness of Pre-Trained Vision Transformers," in *ICML Workshop on New Frontiers in Adversarial Machine Learning*, 2023. (Cited on 141)

[239] K. Gupta and S. Verma, "Shrink & Cert: Bi-level Optimization for Certified Robustness," in *ICML Workshop on New Frontiers in Adversarial Machine Learning*, 2023. (Cited on 141)

[240] X. Ma, P. Zhang, S. Zhang, et al., "A tensorized transformer for language modeling," *arXiv:1906.09777*, 2019. (Cited on 142)

[241] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015. (Cited on 145)