PRACTICAL DEEP LEARNING AGLORITHMS USING ESTIMATION THEORY

ESTIMATION STRATEGIES FOR TRAINING OF DEEP LEARNING NEURAL NETWORKS

By
MAHMOUD ISMAIL, M.Sc, B.Sc.

A Thesis Submitted to the School of Graduate Studies in Partial Fulfilment of the Requirements for the Degree of Doctor of Philosophy

McMaster University

DOCTOR OF PHILOSOPHY (2018), McMaster University, Hamilton, Ontario (Mechanical Engineering)

TITLE:	Estimation Strategies for Training of Deep Learning Neural Networks
AUTHOR:	Mahmoud Ismail, M.Sc, B.Sc.
SUPERVISORS:	Dr. Saeid R. Habibi and Dr. Samir Ziada
NUMBER OF PAGES:	xxi, 20

Lay Abstract

There are two main ideas discussed in this thesis, both are related to Deep Learning (DL). The first investigates the use of estimation theory in DL network training. Training DL networks is challenging as it requires large amounts of data and it is computationally demanding. The thesis discusses the use of estimation theory for training of DL networks and its utility in information extraction. The thesis also presents the application of DL networks in an end-of-line Fault Detection and Diagnosis system for complex automotive components. Failure of appropriately testing automotive components can lead to shipping faulty components that can harm a manufacturer's reputation as well as potentially jeopardizing safety. In this thesis, DL is used to detect and analyze complex fault patterns of automotive starters, complemented by sound and vibration measurements.

Abstract

Deep Learning Networks (DLN) is a relatively new artificial intelligence algorithm that gained popularity quickly due to its unprecedented performance. One of the key elements for this success is DL's ability to extract a high-level of information from large amounts of raw data. This ability comes at the cost of high computational and memory requirements for the training process. Estimation algorithms such as the Extended Kalman Filter (EKF) and the Smooth Variable Structure Filter (SVSF) are used in literature to train small Neural Networks. However, they have failed to scale well with deep networks due to their excessive requirements for computation and memory size. In this thesis the concept of using EKF and SVSF for DLN training is revisited. A New family of filters that are efficient in memory and computational requirements are proposed and their performance is evaluated against the state-of-the-art algorithms. The new filters show competitive performance to existing algorithms and do not require fine tuning. These new findings change the scientific community's perception that estimation theory methods such as EKF and SVSF are not practical for their application to large networks.

A second contribution from this research is the application of DLN to Fault Detection and Diagnosis. The findings indicate that DL can analyze complex sound and vibration signals in testing of automotive starters to successfully detect and diagnose faults with 97.6% success rates. This proves that DLN can automate end-of-line testing of starters and replace operators who manually listen to sound signals to detect any deviation. Use of

DLN in end-of-line testing could lead to significant economic benefits in manufacturing operations.

In addition to starters, another application considered is the use of DLN in monitoring of the State-Of-Charge (SOC) of batteries in electric cars. The use of DLN for improving the SOC prediction accuracy is discussed.

Keywords:

Deep Learning, Fault Detection and Diagnosis, SVSF, RSVSF, EKF, REKF, RSVSF_MF, REKF_MF, Modified-SVSF, Battery SOC, Neural Networks.

Acknowledgement

I express my appreciation to my supervisors, Dr. Saeid R. Habibi and Dr. Samir Ziada for their continuous support and guidance that allowed me to delve into a productive research environment.

I would like thank Dr. Mina Attari for her great help with understanding the estimation theory and her friendship. I would like also to express my gratitude to my colleagues in the Centre for Mechatronics and Hybrid Technologies (CMHT) for their help and making the journey of pursuing a PhD degree a joyful one.

I convey special acknowledgement and gratitude to Natural Sciences and Engineering Research Council of Canada (NSERC) and our industrial partner D&V Electronics for the financial support provided and the industrial experience that allowed this work to be focused on industrial applications as well as the theoretical concepts.

The most special thanks of all goes to my family and their support through my academic career. I have been extremely lucky to have supportive parents, Ehsan and Ahmed who believed in me and encouraged me throughout my life. I would like also to express my gratefulness to have great siblings, Muhammed, Mona, and Mena.

Co-Authorship

This thesis has been prepared in accordance with the regulations for a sandwich thesis format or as a compilation of research papers stipulated by the faculty of graduate studies at McMaster University. This thesis consists of the following papers:

Paper I

Mahmoud Ismail , Mina Attari, Saeid Habibi, and Samir Ziada, "***Estimation Theory and Neural Networks Revisited: REKF and RSVSF as Optimization Techniques for Deep-Learning***" *Published in Neural Networks journal, Elsevier. Volume 108, 2018, Pages 509-526, ISSN 0893-6080. https://doi.org/10.1016/j.neunet.2018.09.012.*

Literature review, Design of Experiments, simulations and results analysis are conducted by Mahmoud Ismail. Paper I was written by Mahmoud Ismail. Mina Attari reviewed the paper and guided the paper writing. Saeid Habibi and Samir Ziada supervised the research work and reviewed and revised the journal paper.

Paper II

Mahmoud Ismail, Saeid Habibi, and Samir Ziada, "***A Study on REKF and RSVSF in Deep Learning: Sensitivity Analysis and Multi-Filter Implementation***" *Submitted to IEEE Transactions on Neural Networks and Learning Systems.*

Literature review, Design of Experiments, simulations and results analysis are conducted by Mahmoud Ismail. Paper II was written by Mahmoud Ismail. Saeid Habibi and Samir Ziada supervised the research work, reviewed and revised the journal paper.

Paper III

Mahmoud Ismail and Saeid Habibi, and Samir Ziada, "***Modified Smooth Variable Structure Filter***" *Submitted to Transactions of the Canadian Society for Mechanical Engineering.*

Literature review, Design of Experiments, simulations and results analysis are conducted by Mahmoud Ismail. Paper III was written by Mahmoud Ismail. Saeid Habibi and Samir Ziada supervised the research work, reviewed and revised the journal paper.

Paper IV

Mahmoud Ismail and Saeid Habibi, and Samir Ziada, "***Quality Control Using Deep Learning for Automotive Starters***" *Submitted to ASME Journal of Manufacturing Science and Engineering.*

Literature review, Design of Experiments, simulations and results analysis are conducted by Mahmoud Ismail. Paper IV was written by Mahmoud Ismail. Saeid Habibi and Samir Ziada supervised the research work, reviewed and revised the journal paper.

Paper V

Mahmoud Ismail[a], Rioch Dlyma[a], Ahmed Elrakaybi[a], Ryan Ahmed[a], and Saeid Habibi[a], "***Battery State of Charge Estimation Using an Artificial Neural Network***" *Published In Transportation Electrification Conference and Expo (ITEC), 2017 IEEE (pp. 342-349). IEEE. DOI: 10.1109/ITEC.2017.7993295*

Literature review is done by Ahmed Elrakaybi, Design of Experiments, simulations and results analysis are conducted by Mahmoud Ismail. Paper III was written Rioch Dlyma.

Ryan Ahmed and Saeid Habibi supervised the research work, reviewed and revised the journal paper.[1]

[1] In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of McMaster's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

Permission to Use

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from McMaster University, I agree that the Libraries of this University may make it freely available for inspection. I further agree that the permission for copying this thesis in any manner, in whole or in part for scholarly purposes, may be granted by the professors who supervised my thesis work or, in their absence, by the Head of the Department or the Faculty Dean in which my thesis work was conducted. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and McMaster University in any scholarly use which may be made of any material in my thesis. Requests for permission to copy or to make other use of material in this thesis, in whole or part, should be addressed to:

Head of the Department of Mechanical Engineering
McMaster University
Faculty of Engineering
1280 Main Street West
Hamilton, Ontario L8S 4L6
Canada

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Deep Learning Training

Deep Learning (DL) is a relatively new Artificial Intelligence (AI) algorithm that has gained significant attention both in academia as well as in industry. The reason is that DL networks have been able to consistently outperform other AI algorithms across a range of applications, [1]. DL is a terminology that is used to describe a wide range of Neural Network architectures. A neuron (node) is the building cell for the network, the network architecture is defining how these neurons are connected and interact with each other. Deep-Learning in its simplest form is a deep feed-forward neural network with multi layers. In Figure 1-1, a simple one hidden layer neural network is shown. The output of each node is described in Figure 1-2 and Equation (1-1), where φ is the activation function, $w_{i,j}^{n}$ is the weight connecting $node\ (n,j)$ to $node\ (n+1,i)$, n is the layer number, and b_i^{n+1} is the bias (offset) for $node\ (n+1,i)$. In neural networks, there are several types of activation functions, linear and nonlinear. As an example, the most commonly used function is the sigmoid function, which is described in Equation (1-2). Feedforward networks take their name from how the output is calculated from the input. The input is first connected to the input layer and then each node in the subsequent layers uses Equation (1-1) to calculate its output. To evaluate the performance of the neural network, a loss/cost function is used. One of the simplest loss functions is the Mean Square Error (MSE), which is described in Equation (1-3).

$$Node_i^{n+1} = x_i^{n+1} = \varphi\left(\sum_{j=1}^{N_n} w_{i,j}^n x_j^n + b_i^{n+1}\right) \tag{1-1}$$

$$\varphi(z) = \frac{1}{1+e^{-z}} \tag{1-2}$$

$$Error_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - target)^2 \tag{1-3}$$

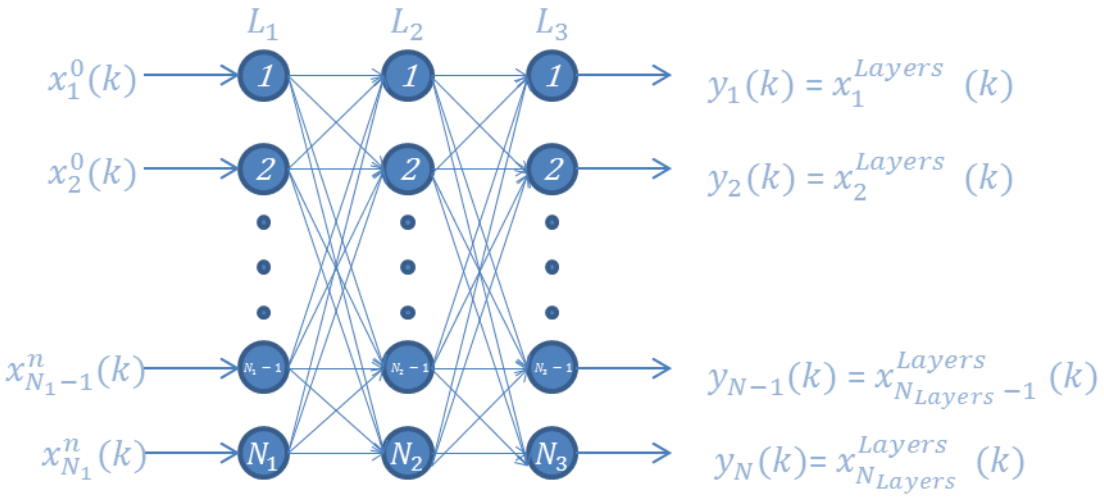


Figure 1-1 One hidden layer Feedforward Neural Network

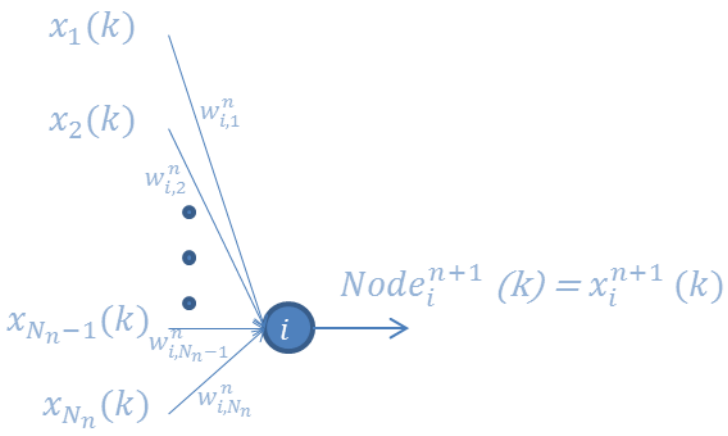


Figure 1-2 Output of a single node

There are different architectures for Deep Learning such as Deep Neural Network (DNN), Convolutional Neural Network (CNN), Long Shot Term Memory (LSTM), Recurrent Neural Network (RNN) and many others. The main difference between Neural Network (NN) and DL is the network size. DL as a term is used to describe deeper networks with larger number of layers and neurons. The concept of DL emerged in late 2000's and early 2010's after several world-recognized machine learning competitions were won by DL networks [1]. The key element that enabled the practical application of DL was the use of the Graphical Processing Units (GPU's) for computation. When it was discovered that a GPU is suitable for fast parallel calculations such as required by DL networks, their use advanced the training speeds to a new level. This discovery allowed researchers and engineers to explore much larger networks than what was the achievable without using a GPU. The data availability was the second key factor in this success. It was proven time after time that DL networks scale well with the size of the network and the amount of the training data. This allowed researchers to push their hardware to its limits by using

increasingly larger network architectures and larger amounts of training data. The main problem with DL lies in its training algorithm and its limitations related to:

1. computational complexity; and
2. the requirement for large computational memory.

These limitations became very clear quickly, as the training algorithms have to be efficient and simple to scale efficiently to large networks. Industry resorted to traditional simple methods such as the Stochastic Gradient Descent (SGD). An overview of the different training techniques is shown in Figure 1-3.

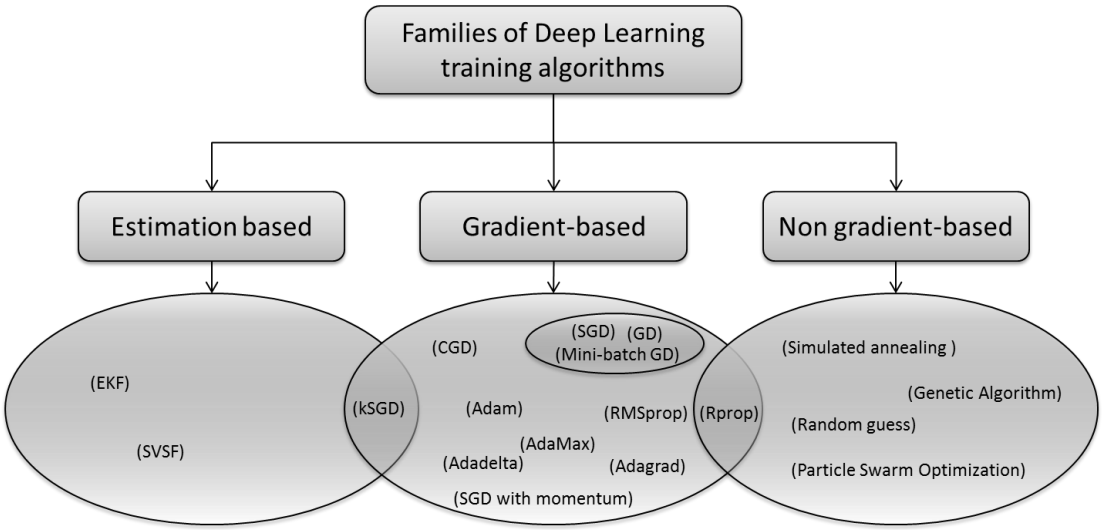


Figure 1-3 different training algorithms families [2-7].

The problem with the gradient descent training algorithms is their static learning rate; meaning that the learning rate is constant over the whole training period. This is not optimal. Generally speaking, in the beginning of the training a large learning rate is required to achieve good transient performance, and a small learning rate is needed to fine tune the

network weights at the end of the training. Another problem was guessing the right learning rate before the training process starts. The risk with a wrong guess jeopardizes:

1. Selecting too small learning rate, wasting long time in the beginning of the training, could be days or weeks for very large networks with large amounts of training data
2. Selecting too large learning rate, which will allow the network to capture the training data features very quickly in the beginning but the final result will not be adequate and the network will not be able to capture all the features in the training data

Academic researchers have had difficulties using dynamic learning rate algorithms that depend on the Hessian in DL due to their impractical computational complexity and memory requirements. Instead, ad-hoc algorithms that depend on the gradient were found that are based on the intuition of the learning process. An example of these algorithms is Nestrove's Acceleration Gradient (NAG) algorithm.

Besides second order algorithms, estimation methods failed to do the migration from small NN to large DL networks due to the same reasons. In literature estimation based methods such as the Extended Kalman Filter (EKF) and the Smooth Variable Structure Variable (SVSF) were applied on small scale NN [8, 9]. EKF and SVSF proved to be very powerful and, outperformed all first order methods and, were competitive with second order methods. However similar to second order methods, they suffered severe disadvantages in scaling to DL networks due to their computational complexity and their large memory requirements.

This thesis revisits the application of EKF and SVSF on DL networks by solving the scaling problems. This research also enables the use of other estimation methods in DL field. The hypothesis in this research is scaling up the estimation based methods can enhance their ability to extract and learn more information and features from the same amount of training data in less time compared with the traditional gradient based methods.

One of the critical characteristics in DL training algorithms is the amount of hyper parameters fine tuning required by the training algorithm. The less the amount of fine tuning without adding extra complexity, the more practical the algorithm would be in industry. Therefore, a comparative analysis is conducted in this research to study the fine tuning effort required for all the new training algorithms in this thesis.

The repetitive nature of DL training was found to be a key factor in simplifying one of the proposed algorithms, namely called the Reduced-SVSF (RSVSF). This simplification allowed for cutting the training time by one third while retaining the same success rate performance. The significance of this simplification inspired further investigation of the impact of such modifications on the SVSF performance outside the Neural Networks' scope of application. The investigation resulted in a new filter called the Modified-SVSF. The filter's stability is studied and the its performance is compared with SVSF using the Electro-Hydraulic Actuator (EHA) [10, 11].

1.2 Fault Detection and Diagnosis

The second contribution in this thesis is the adaptation of DL networks as a Fault Detection and Diagnosis (FDD) algorithm. Fault Detection and Diagnosis (FDD) systems are subject to ongoing research due to their significant economic value. They provide direct cost

savings by avoiding disruptions due to catastrophic failures, or can provide indirect benefits such as enhancing the reputation of a brand by making product more reliable, [12]. In the automotive sector, starter suppliers still use operators to listen to starters under end-of-line testing to detect faults and isolate the ones that do not sound healthy. This is very costly and subject to the operator's skill and ability to hear differences in the starter noise. FDD system can automate this process and substantively reduce cost and improve the reliability of end-of-line testing. FDD systems are generally categorized into signal-based or model-based systems [13-15]. Automotive electromechanical components such as motors, starters and alternators are relatively complex machines, and therefore building a mathematical model that can predict accurately their vibration and sound characteristics is difficult. For this case, signal-based FDD systems are preferred for such machines. FDD in electromechanical components can be done using different measurements, such as: current, voltage, torque, speed, temperature, sound, and vibration. Sound and vibration measurements are very unique compared to the other measurements; they serve two purposes: 1) confirm that the sound and vibration emissions are falling within the required limits, and therefore not producing annoying abnormal noise or vibration, and; 2) confirm there are no mechanical or electrical problems in the component. Most of the mechanical and electrical problems would appear as a change in the sound and vibration characteristics of the component. Assembly defects such as bearing problems can result in annoying sound and vibration without changing other performance parameters such as speed or torque. Despite the importance of sound and vibration measurements, they are not commonly used in industry because of the complexities associated with their processing. All available

analysis methods for sound and vibration result in complicated charts that requires a trained engineer or technician to look at them to decide if there are any faults. Operators would detect faulty parts by comparing baseline charts with the ones from a healthy part. This is a tedious job that takes time and requires trained personnel. To avoid such complexities, often simple thresholds that rely on the overall magnitude of signals are used when dealing with sound and vibration measurements. However, threshold based FDD strategies are not efficiently capable as many faults do not necessarily increase the overall level of the noise or the vibration. Some faults attenuate the noise level only within a specific frequency band or temporal range. Often faults do not change the level of the noise or vibration but change their characteristics; one such example is the broken bar fault in induction motors as discussed in [16]. For such problems, recently developed FDD systems use wavelets and Principle Components Analysis (PCA) such as used in the recently proposed Industrial Extended Multi-Scale Principle Components Analysis (IEMSPCA) in [17]. Another challenge in FDD systems resides in extracting and combining the information that exists in different sensors. When a fault occurs, usually different microphones and accelerometers capture it as a difference in sound and vibration characteristics. Combining the information collected from the different sensors can strengthen the FDD system's ability to detect faults. In literatures, there are two types of analyses for analyzing several measurements, Multi-Variate and Multi-Variable. The latter, Multi-Variable analysis, studies measurements from each sensor independently. On the other hand, Multi-Variate Analysis combines the information from all the measurements and hence, it is better for FDD applications.

Therefore any competitive FDD system to replace operators in industry has to have the following feature:

1. able to extract meaningful information about faults from measured signals from multiple sensors;
2. able to combine information from multiple sensors into one conclusion; and
3. use the characteristics of faults to diagnose and quantify the condition.

Deep Learning have been used extensively in classification tasks such as in speech and image recognition. However it is not yet well studied in other applications such as FDD. This thesis explores its application to the FDD field using sound and vibration measurements from a manufacturing environment. As far as automotive starters are concerned, a DL based FDD system that produces adequate performance can be used to replace operators who listen to starters' noise to isolate the defective ones. The FDD system would have higher repeatability and much lower operating cost than operators.

1.3 Battery SOC Level Estimation Using Neural Networks

Deep Learning application for batteries State-Of-Charge (SOC) is also studied. In literature Neural Networks is used to estimate battery SOC. However Neural Networks usually result in inaccurate noisy estimates as shown in Figure 1-4 [18], therefore the Neural Network estimation is usually followed by an enhancement stage, such as post application of the EKF or the Unscented Kalman Filter (UKF) as shown in Figure 1-5 [18-20].

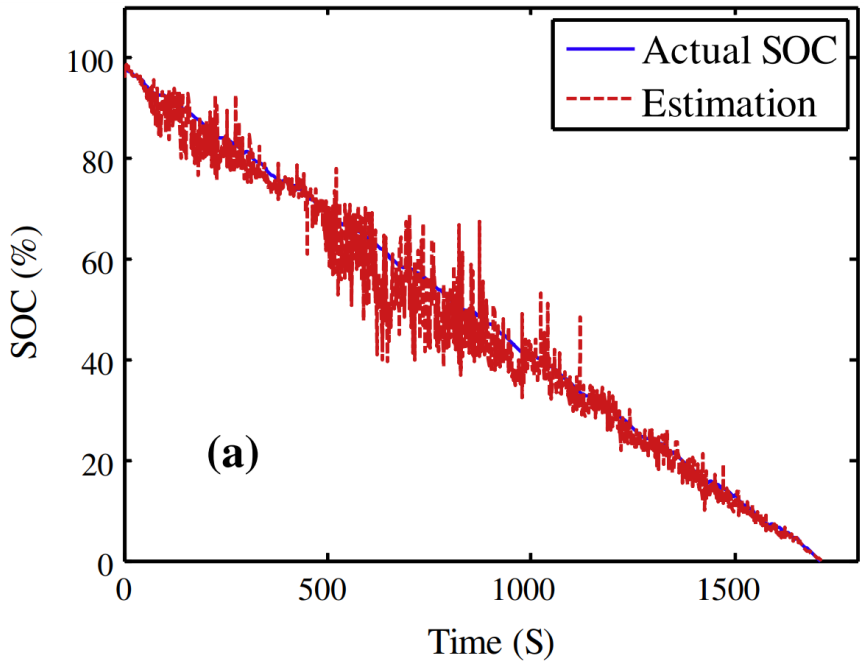


Figure 1-4 Neural network results for US06 data [18]

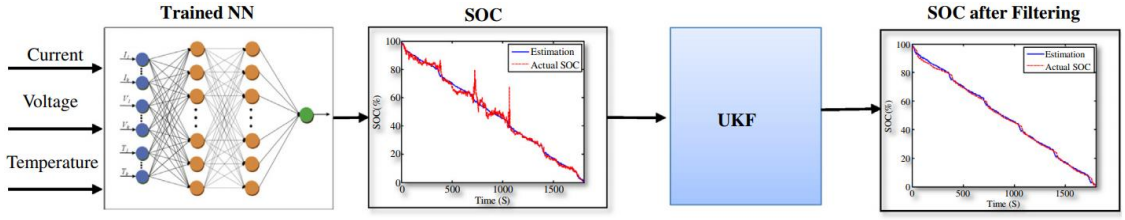


Figure 1-5 depicts an example of filtering NN SOC prediction using Unscented Kalman Filter (UKF) [18]

There is a tremendous demand for optimizing SOC estimation in: 1) Hybrid Electric Vehicles (HEVs), 2) Plug-in Hybrid Electric Vehicles (PHEVs), and 3) Battery Electric Vehicles (BEVs). Batteries are the core of electric vehicles and one of the most complex systems to monitor. With the development of different battery chemistries, SOC levels for some types are very hard to estimate. This is very important as the SOC level translates into the remaining mileage range for the vehicle. Not knowing the remaining available mileage range is inconvenient and introduces what is called as the range anxiety. In

literature, there are several battery models such as 1) Equivalent Circuit-Based Models, 2) Electrochemical-Based Models, and 3) Behavioral Battery Models [21]. The purpose of these models is for accurately predicting the SOC and State-Of-Health (SOH) of batteries. Due to noise in the prediction of the SOC prediction, NN is not the favored choice for SOC estimation in industry. Chapter 6 of this thesis discusses how NN SOC prediction can be improved by using appropriate training data.

1.4 Research Contribution and Novelty

This thesis presents three hypothesis that are discussed below in details and followed by a list of contributions.

1.4.1 Hypotheses

1.4.1.1 Hypothesis 1: New estimation based training methods shall be able to outperform the current state-of-the-art Deep-Learning gradient-based training techniques.

- Estimation methods and SVSF in particular are believed to be able to be deployed as a Deep-Learning optimization tool that can outperform other optimization methods for the following reasons:

1- SVSF is robust against modeling uncertainties as shown in [22]

2- SVSF benefits from the filtering feature inherent in model-based estimation methods and suppress noise in training samples that can hinder the training performance

3- Estimation methods are adaptive and therefore they can adequately update the learning rate of the network in an adaptive manner that allows for maximum training performance.

4- The research in [8, 23] show that SVSF and EKF have a very high potential in outperforming other optimization methods. However, their main problem is their computational complexity when scaling up to bigger network.

1.4.1.2 Hypothesis 2: Deep Learning based Fault Detection and Diagnosis shall be able to outperform the current state-of-the-art FDD systems.

Deep learning is believed to be able to be deployed as a Fault Detection and Diagnosis tool that can outperform other FDD systems for the following reasons:

1. It is a signal based algorithm
2. It does not require reading complex charts
3. Deep Learning is a relatively new algorithm and in the last few years, it has outperformed other AI strategies when applied to complex problems such as in:
    i. image recognition [1]; and
    ii. speech recognition [24, 25]

    These examples show the potential of Deep Learning in extracting information from large amount of data.
4. Deep Learning has shown robustness against signal noise in both speech and image recognition

1.4.1.3 Hypothesis 3: Deep Learning networks shall be able to predict batteries State-Of-Charge (SOC) levels very accurately if they are properly trained

Deep learning is very suitable to the application of SOC estimation due to several reasons including:

1. Deep Learning excels in complex classification tasks such as image and speech recognition, which are much more complex in nature than predicting the SOC levels of batteries.

- Deep Learning has performed well in regression tasks, and subsequently being used heavily in finance [26, 27]. SOC level prediction is a regression problem and is comparatively simpler as it uses fewer input parameters (such as battery voltage and current).

1.4.2 Contributions

The above mentioned hypotheses were the main driving force for this work and several primary and secondary contributions resulted from researching the answers for these hypotheses. The following sections summarize the primary and the secondary contributions and they are summarized in Figure 1-6.

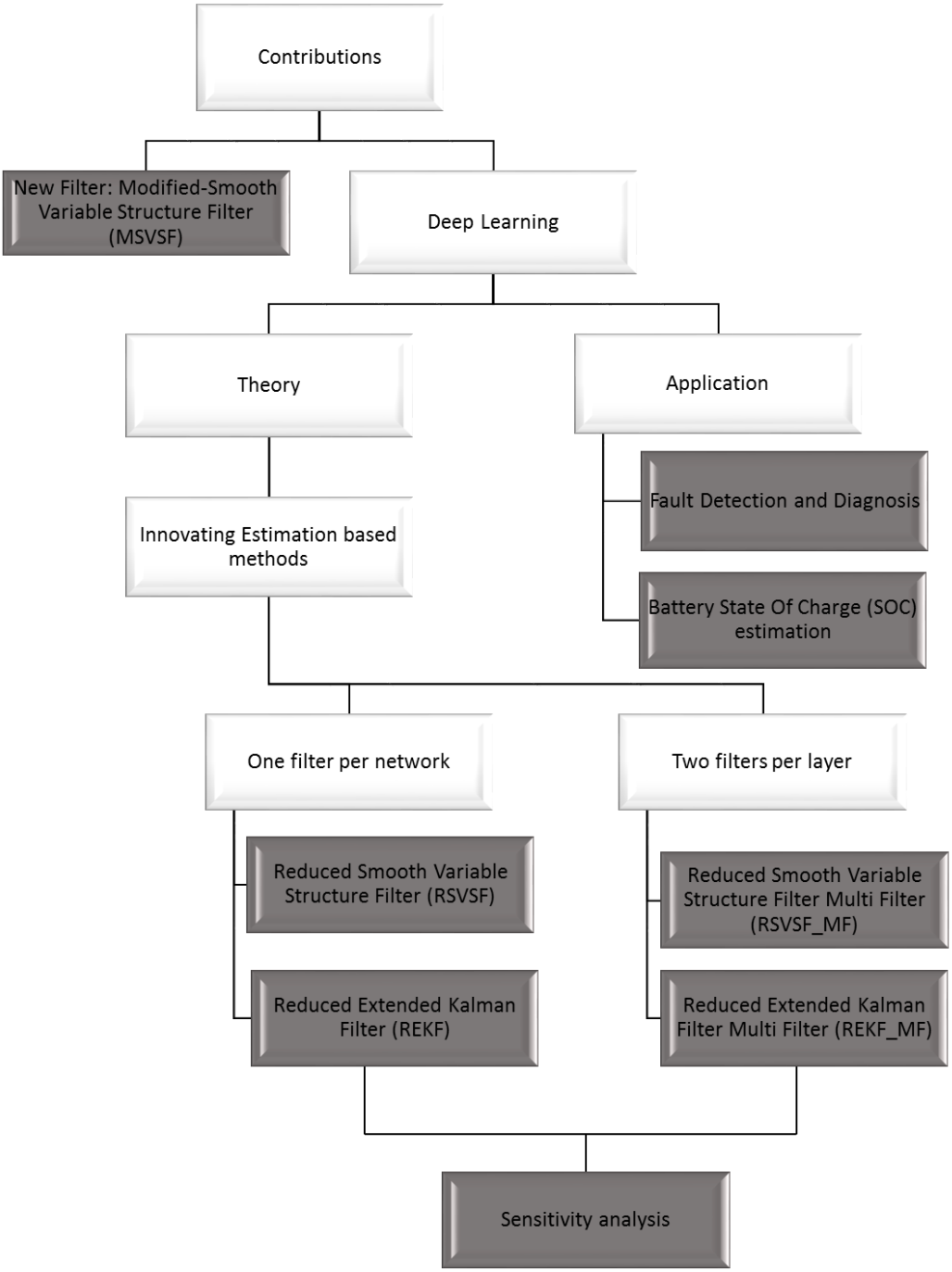


Figure 1-6 Research Contribution Flowchart

1.4.2.1 Primary Contributions

The main contributions are as follows:

1. Developed a new category of optimization techniques for Deep Learning, referred to Reduced Extended Kalman Filter (REKF), Reduced Smooth Variable Extended Filter (REKF), REKF_MultiFilter (REKF_MF) and RSVSF_MultiFilter (RSVSF_MF) (chapters 2 and 3).
2. The above mentioned new filters were implemented both on CPU and GPU platforms (chapters 2 and 3).
3. Evaluated the performance of RSVSF and REKF (chapter 2). They proved to be superior to state-of-art training algorithms in terms of:
    a. transient performance;
    b. steady state performance;
    c. required memory; and
    d. computational complexity.
4. Conducted a comparative sensitivity analysis for RSVSF, REKF RSVSF_MF and REKF_MF against their tuning parameters (chapter 3).
5. Developed a new state estimation filter called Modified-SVSF that is influenced by the high performance of RSVSF. This filter adds a new dimension to the original SVSF filter (chapter 4). Associated with this contribution were the following:
    a. a proof of stability for the Modified-SVSF; and
    b. performance comparison with the original form of the SVSF filter.

6. Developed a Deep Learning based FDD system that is able to analyze sound and vibration measurements. The performance of this system in evaluated using measurements for starters from production environments (chapter 5).
7. Developed a Neural Network system that is able to estimate accurately the SOC levels of batteries. Network training was found to be the key element determining the prediction accurately for SOC levels (chapter 6). In the training the following are the most important factors:
    a. providing the network with the correct inputs (voltage and current) and their history to capture the transient effect; and
    b. including transient battery behavior in the current profile that is used to train the network.

1.4.2.2 Secondary Contributions

Further contributions from the research are as following:

1. Conducted a benchmark study for commonly used DL training algorithms and compared their performance to estimation based methods training algorithms using benchmarking datasets (chapter 2).
2. Compared the proposed new filters with state-of-art algorithms. A benchmark study was conducted using different standard benchmark problems on a large network scale (chapter 2).
3. Conducted a benchmark study to compare the performance of both DL network architectures, Convolution Neural Networks (CNN) and Deep Neural Networks

(DNN) in analyzing faults in starters using their sound and vibration characteristics (chapter 5).

4. Studied the importance of combining different sensor types to accurately capture different faults. Fault detection algorithms were tested using sound only, vibration only, and both measurements (chapter 5).

1.5 Organization of the thesis

This thesis presents a combination of analytical and experimental research used to propose advancements in Deep Learning algorithms and Deep Learning applications. The thesis is organized as follows.

Chapter 2 introduces two novel Deep learning training algorithms. The chapter provides a literature review on Deep Learning training algorithms and introduces different benchmark problems that are used later for comparative performance evaluations of the proposed training methods. The new estimation based training algorithms (namely REKF and RSVSF) are discussed and their stability is studied. The new algorithms are compared with the state-of-art algorithms both on small and large scale networks using CPU and GPU respectively.

Chapter 3 proposes two new algorithms based on REKF and RSVSF. The new algorithms deploy two filters per network layer instead of having one filter that trains the whole network. The new algorithms are called the Reduced Extended Kalman Filter_Multi Filter (REKF_MF) and the Reduced Smooth Variable Struction Filter_Multi Filter (RSVSF_MF).

A sensitivity analysis is conducted for the four algorithms and the most important tuning parameters are identified. A comparison of the algorithms is presented that provides a guideline for determining which algorithm is more suitable under which circumstances.

Chapter 4 introduces a new general estimation method called Modified-SVSF based on RSVSF. This new filter expands SVSF tuning parameters to three dimensions instead of two. This allows for better fine tuning of the filter for better performance. The stability of the new filter is studied, complemented by a mathematical proof. The performance of this filter is evaluated and compared with the original SVSF filter. In this evaluation the Electro-Hydraulic Actuator (EHA) [10, 11] is used. The stability of the new filter is experimentally compared with the theoretical stability and was shown to match.

Chapter 5 proposes the use of Deep Learning as a Fault Detection and Diagnosis (FDD) algorithm. In this chapter two DL architectures were studied, Convolutional Neural Networks (CNN) and Deep Neural Networks (DNN). They were tested using sound and vibration measurements from automotive starters. The measurements were acquired using an end-of-line tester from a production environment. The results show the ability of the DL to accurately detect and diagnose faults even with the existence of background noise. In this study different variations of inputs were tested, using sound measurements only, using vibration measurements only or using both measurements. The purpose of these studies is to 1) find the optimum network architecture and type of sensors to capture faults 2) show Deep Learning's ability to combine information from different sensors to achieve higher fault detection and diagnosis performance.

Chapter 6 introduces a novel training method for neural networks that are used for estimating Batteries State Of Charge (SOC) levels. This training method makes use of the history of the voltage and the current of the batteries during operation. In this method, the current profile includes different cycles that use constant current segments as well as normally distributed current profiles with shifted mean to have a complete charge/discharge cycles. This current profile assures that the network is introduced to the transient behavior of batteries across different SOC levels through the training phase. This exposure to battery transient behavior in the training phase improves significantly the SOC prediction performance.

Chapter 7 provides the major conclusions and the recommendations future research.

Chapter 2: Estimation Theory and Neural Networks Revisited: REKF and RSVSF as Optimization Techniques for Deep-Learning

Mahmoud Ismail[a,*], Mina Attari[a], Saeid Habibi[a], and Samir Ziada[a]

[a] Department of Mechanical Engineering, McMaster University, Hamilton, Canada;

[*]**Corresponding author**: Mahmoud Ismail, Department of Mechanical Engineering, McMaster University, Hamilton, Canada.

This paper is published in Neural Networks journal, Elsevier. Volume 108, 2018, Pages 509-526, ISSN 0893-6080. https://doi.org/10.1016/j.neunet.2018.09.012. This paper is republished here with permission.

Abstract

Deep-Learning has become a leading strategy for artificial intelligence and is being applied in many fields due to its excellent performance that has surpassed human cognitive abilities in a number of classification and control problems [1, 2]. However, the training process of Deep-Learning is usually slow and requires high-performance computing, capable of handling large datasets. The optimization of the training method can improve the learning rate of the Deep-Learning networks and result in a higher performance while using the same number of training epochs (cycles). This paper considers the use of estimation theory for training of large neural networks and in particular Deep-Learning networks. Two estimation strategies namely the Extended Kalman Filter (EKF) and the Smooth Variable Structure Filter (SVSF) have been revised (subsequently referred to as RSVSF and REKF)

and used for network training. They are applied to several benchmark datasets and comparatively evaluated.

Keywords

Kalman filter, smooth variable structure filter, Deep-Learning, neural networks, REKF, RSVSF.

2.1 Introduction

Deep-Learning (DL) is being increasingly used in academic and industrial applications. DL is based on self-learning which means that the network's performance depends mainly on the training process. The quality of the acquired knowledge through the training process depends on many factors. The most important ones are:

1) the complexity and the quantity of the training data;
2) the architecture of the DL network; and
3) the training algorithm.

For DL networks, fast and efficient training algorithms are required to maximize information extraction in the least amount of time. Training algorithms are evaluated mainly based on their (1) convergence speed and (2) generalization performance that is how the algorithm generalizes across different architectures and datasets. Any improvement in convergence speed is critical, as it can shorten the training time and enable the network to achieve results that are more accurate. The reduction in the training time can be significant. Therefore, training algorithms of DL networks is an open research topic.

Early research on DL using a Feedforward Multi-layer Perceptron was introduced in [3-6]. DL attracted attention starting in late 2000's when it won a number of official international competitions in pattern recognition [7-9]. In [9], a detailed review of DL history can be found. There are different DL structural architectures with each excelling in a different application. Generally, the increased number of layers or depth allows for capturing high-level features from low-level/raw features resulting in better clustering, regression, and classification performance. For example, in [10, 11], it is shown that DL achieves the same performance as shallow NNs and kernel methods, however, DL needs only about one-tenth of the units to achieve that performance. In [12], a discussion of the theoretical advantages of deep architectures can be found; however, these improvements come at a cost. The first problem is the increase of the computational complexity and the lower training speed due to the extensive calculation steps in each training cycle, particularly, for a large number of tunable parameters. Any minor added computation accumulates quickly over the number of parameters and increases the total computation time drastically. The second problem in DL is the memory limitation. The memory requirement limits (1) the number of input samples that can be processed in parallel, and (2) the calculations and the number of network parameters. For the first issue, mini-batches are used. Memory limitation is restrictive to the implementation of the training algorithms as they need to be memory efficient. These challenges are among the reasons why DL networks despite their history were not able to practically achieve broad meaningful and remarkable results due to computational constraints until modern hardware components and optimized codes are used [9, 13, 14]. The significant increase in the number of network parameters created a

motivation for simple, fast and memory efficient training algorithms to speed up the training process. Therefore algorithms such as Adam [15], RMSprop [16] and Adadelta [17] and other techniques became very competitive.

In literature, two state estimation strategies, namely, the Extended Kalman Filter (EKF) [18-20], and the Smooth Variable Structure Filter (SVSF) [21, 22] were applied on Neural Networks (NN) as training algorithms. Using estimation methods such as EKF and SVSF for NN training is a well-suited application for these algorithms as the training involves using empirical data that is noisy. However, EKF and SVSF have been used only in the training of shallow NN structures. They have not to date been scaled up to larger DL structures. This paper considers proposing two new training algorithms derived from EKF and SVSF for the training of DL networks.

In Section 2.2 of this paper, a review of network training strategies including the application of the EKF and the SVSF concepts for NN training is presented. Afterward, two new algorithms based on the EKF and the SVSF are proposed in Section 2.3. The performance evaluation of the proposed algorithms is presented in Section 2.4 and concerns three case studies. The first case compares between adaptive first-order algorithms to select a representative of this family for the tests in the third case. The second case shows the advantage of the proposed revised algorithms over their original forms of EKF and SVSF, and the third case evaluates the generalization of the optimization performance of the proposed algorithms on large-scale networks across different datasets and architectures. Section 2.5 is the conclusion.

2.2 Deep Learning and Neural Networks training methods

2.2.1 Deep Neural Networks training methods

In Deep-Learning, there are different training methods that minimize the network error iteratively. The training methods can be categorized as:

- gradient-based methods, and
- non-gradient methods

Most Neural Networks (NN) are trained using the Back Propagation (BP) concept [23], with an optimization algorithms such as the Gradient Descent or one of its variants [24]. Gradient-based methods provide guided optimization, direction and magnitude wise. They are relatively fast and can be calculated in a parallel manner. In this section, gradient-based methods are presented. In the second non-gradient based category, the gradient is not used. Examples of these methods include random guess, evolutionary and genetic algorithms. These are not reviewed here. In [24], a review of different gradient-based methods is provided. The following table provides a brief summary of the most common gradient-based methods and their governing equations for updating network weights. The nomenclature is provided in

Appendix A and as follows.

Table 2-I Current training optimization algorithms for DL

Training algorithms	Equation	
Batch Gradient Descent (GD)	$w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) - \eta \times \frac{\partial Error(w,x,y)}{\partial w_{i,j}^{n_l}}$	(2-4)
Stochastic Gradient Descent (SGD)	$w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) - \eta \times \frac{\partial Error(w,x^q,y^q)}{\partial w_{i,j}^{n_l}}$	(2-5)
Mini-batch Gradient Descent	$w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) - \eta \times \frac{\partial Error(w,x^{q\to q+B},y^{q\to q+B})}{\partial w_{i,j}^{n_l}}$	(2-6)
Mini-batch GD with momentum	$v_{k+1} = \gamma\, v_k - \eta \times \frac{\partial Error(w,x^{q\to q+B},y^{q\to q+B})}{\partial w_{i,j}^{n_l}}$	(2-7)
	$w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) + v_{k+1}$	(2-8)
Mini-batch GD with Nesterov acceleration	$v_{k+1} = \gamma\, v_k - \eta \times \frac{\partial Error((w+\gamma\, v_k),x^{q\to q+B},y^{q\to q+B})}{\partial w_{i,j}^{n_l}}$	(2-9)
	$w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) + v_{k+1}$	(2-10)
Regularised Update Descent	$v_{k+1} = \gamma\, v_k - \eta \times \frac{\partial Error((w+v_k),x^{q\to q+B},y^{q\to q+B})}{\partial w_{i,j}^{n_l}}$	(2-11)
	$w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) + v_{k+1}$	(2-12)
Adagrad	$w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) - \frac{\eta}{\sqrt{G_{k,ij}+\epsilon}} \times \frac{\partial Error(w,x^{q\to q+B},y^{q\to q+B})}{\partial w_{i,j}^{n_l}}$	(2-13)
	$G_{k,ij} = \sum_{\tau=1}^{k}\left(\frac{\partial Error(w_\tau,x^{q\to q+B},y^{q\to q+B})}{\partial w_{\tau,i,j}^{n_l}}\right)^2$	(2-14)
Adadelta	$w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) - \frac{RMS[\Delta w]_k}{RMS[g]_{k+1}} \times g_{k+1}$	(2-15)
	$RMS[\Delta w]_k = \sqrt{E[\Delta w^2]_{k+1}+\epsilon}$	(2-16)
	$E[\Delta w^2]_{k+1} = \gamma\, E[\Delta w^2]_{k+1} + (1-\gamma)\,[\Delta w^2]_{k+1}$	(2-17)
	$RMS[g]_{k+1} = \sqrt{E[g^2]_{k+1}+\epsilon}$	(2-18)

	$E[g^2]_{k+1} = \gamma\, E[g^2]_{k+1} + (1-\gamma)\,[g^2]_{k+1}$	(2-19)
	$g_{k+1} = \dfrac{\partial Error(w, x^{q\to q+B}, y^{q\to q+B})}{\partial w_{i,j}^{n_l}}$	(2-20)
RMSProp	$v_{k+1} = \gamma\, v_k + (1-\gamma)\left[\dfrac{\partial Error(w, x^{q\to q+B}, y^{q\to q+B})}{\partial w_{i,j}^{n_l}}\right]^2_{k+1}$	(2-21)
	$w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) - \dfrac{\eta}{\sqrt{v_{k+1}+\epsilon}} \times \dfrac{\partial Error(w, x^{q\to q+B}, y^{q\to q+B})}{\partial w_{i,j}^{n_l}}$	(2-22)
Adam	$v_{k+1} = \beta_2\, v_k + (1-\beta_2)\left[\dfrac{\partial Error(w, x^{q\to q+B}, y^{q\to q+B})}{\partial w_{i,j}^{n_l}}\right]^2_{k+1}$	(2-23)
	$m_{k+1} = \beta_1\, m_k + (1-\beta_1)\left[\dfrac{\partial Error(w, x^{q\to q+B}, y^{q\to q+B})}{\partial w_{i,j}^{n_l}}\right]_{k+1}$	(2-24)
	$\hat{m}_{k+1} = \dfrac{m_{k+1}}{1-\beta_1^t}$	(2-25)
	$\hat{v}_{k+1} = \dfrac{v_{k+1}}{1-\beta_2^t}$	(2-26)
	$w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) - \dfrac{\eta \cdot \hat{m}_{k+1}}{\sqrt{\hat{v}_{k+1}}+\epsilon}$	(2-27)
AdaMax	$u_{k+1} = Max\left(\beta_2\, u_k\,, \left\lvert\dfrac{\partial Error(w, x^{q\to q+B}, y^{q\to q+B})}{\partial w_{i,j}^{n_l}}\right\rvert_{k+1}\right)$	(2-28)
	$m_{k+1} = \beta_1\, m_k + (1-\beta_1)\left[\dfrac{\partial Error(w, x^{q\to q+B}, y^{q\to q+B})}{\partial w_{i,j}^{n_l}}\right]_{k+1}$	(2-29)
	$w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) - \dfrac{\eta}{(1-\beta_1^t)} \times \dfrac{m_{k+1}}{u_{k+1}}$	(2-30)

The first three optimization methods in Table 2-I are commonly used. Batch Gradient Descent (GD) uses the whole training dataset in each update to modify the weights. Where $w_{i,j}^{n_l}$ is the weight connecting $node\ (n_l, j)$ to $node\ (n_l+1, i)$, and n_l is the layer number. Node bias is modeled as a weight for an input of one. η is the learning rate, and the partial derivative is calculated using the chain rule. Stochastic Gradient Descent (SGD)

uses only one training sample (where q is the sample index). Mini-batch GD is a mix of GD and SGD methods. In each update, it uses a subset of the training dataset to modify the network weights. In Equation (2-6), q is the index of the first sample in the batch and B is the batch size. Mini-batch GD is the most commonly used training algorithm amongst the three methods due to the following reasons:

1) It is faster than stochastic gradient descent due to utilizing parallel computation.
2) It allows for processing very large datasets by breaking them down into smaller mini-batches

Although the three previous methods are widely used and provide simple and efficient training algorithms for DL, they suffer from the following challenges [24]:

1) η is a constant value and does not adapt to the training phase for optimizing learning speed
2) η is chosen before the training starts, introducing an ambiguity on selecting a proper value.
3) All parameters are updated with the same learning rate. This is not an optimal situation, especially with high sparsity in the training dataset.
4) When GD methods are trapped around saddle points, they struggle to escape [25].

To address some of these problems, the momentum term was introduced by [26]. The momentum term gains inertia in the dominant direction of the optimization which increases the learning speed [27]. The updates in the mini-Batch form with momentum are governed by Equations (2-7) and (2-8). Nesterov [28] modified the momentum algorithm such that the update on the velocity is more controlled. The main difference is that the gradient

correction is calculated using $(w + \gamma\ v_{k-1})$ instead of (w). This modification allows faster global convergence, $O\left(\frac{1}{k^2}\right)$ instead of $O\left(\frac{1}{k}\right)$ in the GD algorithm in certain conditions [28, 29]. In [30], the authors propose a further modification (which is described in Equations (2-11), (2-12)) and the algorithm is called Regularized Gradient Descent. The modification again is in the gradient correction using $(w + v_k)$ instead of $(w + \gamma\ v_{k-1})$ compared to Nestrovs's case. When tested on MNIST[2] benchmark dataset, this algorithm was able to compete with Nestrove's Acceleration Gradient (NAG).

In order to introduce individual adaptive learning rates for each network parameter, the Adagrad method was proposed in [32]. In Equation (2-13), individual learning rates for the individual weights are scaled by $\sqrt{G_{k,ij} + \epsilon}$, where ϵ is a small value for division stability to avoid division by zero [24]. In Equation (2-14), $G_{k,ij}$ is a diagonal matrix with its diagonal elements being a sum of the square of the weights' gradients from the start of the training process to time (k). $G_{k,ij}$ is defined as a diagonal matrix to allow the square root calculation in Equation (2-13) in a feasible time. Adagrad was extended to Adadelta in [17] by Zeiler. The concept is to have a running average window of the weights' gradients [24]. Adadelta has strong advantages such as requiring no manual setting of a learning rate and insensitivity to hyper-parameters [17]. A similar concept was used in [16] for an algorithm called RMSprop. The adaptive learning rate is scaled by a running average window that

[2] The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image-processing systems [31] L. Deng, "The MNIST database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine,* vol. 29, pp. 141-142, 2012..

uses the square of the gradients with forgetting factor (γ) as shown in Equations (2-21) and (2-22). Adam [15] is another adaptive method that is dependent on the moving windows of the first and second moments, the mean value and the variance, respectively. The values of the moving windows are scaled by $(1-\beta_i^t)$ to fix the bias of the estimate of the moments in Equations (2-25) and (2-26). Adam has nice and smooth convergence curves and less bouncy behavior than RMSprop and other methods. An extension to Adam was introduced in the same paper [15] and called AdaMax. It was found that generalizing the second norm L^2 to the p norm (L^p) in the update rule can lead to some numerical instability [15]. However, a simple and stable algorithm arises in the special case when $p \to \infty$. The algorithm is defined in Equations (2-28), (2-29), and (2-30).

Another family of optimizers is called Conjugate Gradient Descent (CGD). CGD and its extensions [33-41] reduce the error by first finding the steepest direction of the error reduction on the error curves, and then finding the optimal step size in that direction by performing a line search. The difference between different methods in this family is mainly in the formula that defines the direction of the update. The authors in [42] showed that they can work in a mini-batch fashion similar to SGD, and can outperform SGD in many problems.

All of the above-mentioned techniques depend on the first gradient of the loss function with respect to the network parameters. However, in the early days of NN, researchers found that it is possible to improve the learning rates and achieve better results in fewer epochs using the second order information by computing the Hessian matrix or its approximation. With the emerge of DL most of these methods were disregarded due to 1) the very high

computational requirements for computing the Hessian matrix or its approximation, and performing matrix inversion and 2) the very high memory requirements. Newton's method is the standard optimization algorithm in the family of second-order optimization methods and it uses the Hessian matrix. Quasi-Newton (or secant) methods are found to avoid the complexity of calculating the Hessian matrix by approximating it with different methods such as finite difference. One of the best Quasi-Newton methods is the Broyden–Fletcher–Goldfarb–Shanno (BFGS) Quasi-Newton [43, 44]. In [42], L-BFGS is a limited memory version of BFGS and it was applied using mini-batches and was compared with SGD and CGD. While it performed better than SGD, the authors recommended using CGD for larger problems. Another method that approximates the Hessian matrix is Levenberg-Marquardt [45], which is applied to the NN in [46]. Levenberg-Marquardt is considered as one of the fastest methods to converge for small-scale NN problems. This method avoids the complex Hessian matrix calculations by calculating an approximation based on the Jacobian matrix $H_{Hessian} = J^t J$. These methods that approximate the Hessian matrix still inherit the memory problem as for an $n - parameter$ network they require $O(n^2)$ memory for storage [44]. In [47] a diagonal approximation of the Hessian is proposed. The calculation of this approximation requires an additional forward iteration and backward calculation of its corresponding gradient. In [48], the author proposes a fast optimization method that is Hessian free. This method is based on truncated-Newton that was proposed in [49]. In "Hessian free" method, information about the second order is collected using finite difference. Then, a damped function of the second order information is defined and minimized using CG.

Another attempt to capture the second order information is done by using the Fisher matrix. In Natural Gradient Descent (NGD) the inverse Fisher matrix is used to scale the step size [50, 51]. This is shown in Equation (2-31), where E is the Fisher matrix.

$$w_{i,j}^{n_l}(k+1) = w_{i,j}^{n_l}(k) - \eta \times E^{-1} \times \frac{\partial Error(w, x^{q \to q+B}, y^{q \to q+B})}{\partial w_{i,j}^{n_l}} \quad (2\text{-}31)$$

Fisher matrix under certain conditions is equivalent to the Hessian. Amongst these conditions are that the loss function is using cross-entropy (log probability loss) as well as that the model is correct and corresponds to the true distribution [51]. In practice, the true distribution model is not known and hence the only known distribution is captured from the training samples [52]. Although close to the true distribution, the observational Fisher matrix is not exactly equivalent to the true distribution Fisher matrix. There are different ways to estimate the observational Fisher matrix as highlighted in [51]. However the estimates of Fisher matrix give a good indication of the gradient descent direction, they inherit the same problem of Hessian based algorithms of inverting a large matrix. Therefore [52] highlighted that NGD algorithms have poor scaling capabilities as they have the same size of a Hessian matrix. In multiple publications, it was shown that Fisher matrix can be factorized to have a close approximation that can be inverted with much less computational power [51-54]. It was found that block diagonal representation of the Fisher matrix usually yields good results. This block diagonal approximation of Fisher matrix was evaluated in [51, 52]. The reasoning behind this is explained in [55] as the Hessian with cross-entropy loss converges to a block diagonal matrix. These blocks represent the weights that connect either to a hidden or to an output unit. The extreme of Fisher matrix simplification is the

first-order version, where Fisher matrix is approximated to a diagonal matrix by ignoring the covariance between weights.

Many of the discussed first-order techniques depend on the momentum. The concept of using the momentum is driven by the second order techniques, where the optimization algorithm is able to measure the rate of change of the gradient. For this reason, the methods that use momentum calculation, in one fashion or another, are superior to those that do not incorporate any information about how the gradient is changing.

One of the adaptive first-order algorithms that stands out is Adam for its favorable characteristics such as:

- Adaptive learning rates,
- Fast convergence,
- Smooth learning,
- Relatively simple calculation requirements,
- Less need for parameters fine tuning, and
- Using the first two moments (mean and variance) to optimize the learning speed.

These characteristics amongst many others are the reason why Adam became commonly adopted for DL networks training.

2.2.2 Extended Kalman Filter and Smooth Variable Structure Filter in Neural Network Training

The Extended Kalman Filter (EKF) and the Smooth Variable Structure Filter (SVSF) are powerful non-linear estimators. They form the basis of the proposed algorithms. In this section, a review of their application to Neural Network training is provided.

2.2.2.1 Extended Kalman Filter

The Extended Kalman filter (EKF) is an extension of the Kalman filter for nonlinear systems [56, 57]. A typical non-linear system can be described by Equations (2-32) and (2-33). In these equations, $\left(w_{k-1}^{noise}, v_k^{noise}\right)$ are the process and measurement noise, respectively[3].

$$s_k = f(s_{k-1}, u_{k-1}) + w_{k-1}^{noise} \tag{2-32}$$

$$z_k = h(s_k) + v_k^{noise} \tag{2-33}$$

In Equation (2-32), s_k is a vector of the system states, and $f(s_{k-1}, u_{k-1})$ is a non-linear function of the states and the input u_{k-1}. In Equation (2-33), $z_k$is the system output, and $h(s_k)$ is a non-linear output function.

EKF has a predictor-corrector form. First, the state estimates, their corresponding covariance matrix, and measurements are predicted by using the model of the system as given by Equations (2-34), (2-35), and (2-36).

$$\hat{s}_{k|k-1} = f\left(\hat{s}_{k-1|k-1}, u_{k-1}\right) \tag{2-34}$$

$$P_{k|k-1} = F_{k-1} P_{k-1|k-1} F_{k-1}^t + Q_{k-1} \tag{2-35}$$

[3] In these noise terms, superscript ($noise$) is used to differentiate them from the weights (w) in all previous equations.

$$\hat{z}_{k|k-1} = h(\hat{s}_{k|k-1}) \tag{2-36}$$

where, Q_{k-1} is the covariance of the process noise and F_{k-1} is the jacobian of f with respect to $\hat{s}$ as shown below in Equation (2-42). Then, the update step refines the estimates by applying a corrective term that is a function of the error between the measured and the estimated outputs. This error term is multiplied by a gain referred to as the Kalman gain. The estimation process is implemented by the following equations:

$$\tilde{y}_k = z_k - \hat{z}_{k|k-1} \tag{2-37}$$

$$S_k = H_k P_{k|k-1} H_k^t + R_{k-1} \tag{2-38}$$

$$K_k = P_{k|k-1} H_k^t S_k^{-1} \tag{2-39}$$

$$\hat{s}_{k|k} = \hat{s}_{k|k-1} + K_k \tilde{y}_k \tag{2-40}$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \tag{2-41}$$

where R_{k-1} is the covariance of the measurement noise and H_k is the linearization around the estimate as follows.

$$F_{k-1} = \left.\frac{\partial f}{\partial s}\right|_{\hat{s}_{k-1|k-1}, u_{k-1}} \tag{2-42}$$

$$H_k = \left.\frac{\partial h}{\partial s}\right|_{\hat{s}_{k|k-1}} \tag{2-43}$$

The application of EKF to the training of NN was first introduced in [58]. Many researchers have subsequently applied the EKF on Feedforward Neural Network (FNN) [19, 20, 59] as well as Recurrent Neural Network (RNN) architectures [60, 61].

To formulate a Feedforward Neural Network (FNN) problem in the form of a non-linear system, so that EKF can be employed, the following steps are taken [19, 20, 59]:

1) The network weights are defined as the non-linear system states (Equation (2-44)).
2) The network output is treated as the system output (Equation (2-45)).
3) Inputs are defined as the training data samples (Equation (2-46)).
4) The linearized model is defined as an Identity matrix (Equation (2-47)).

$$s = \begin{bmatrix} w_{1,1}^{1} \\ \vdots \\ w_{N_{Layers},N_{Layers-1}}^{Layers-1} \end{bmatrix} \tag{2-44}$$

$$h(\hat{s}_{k|k-1}) = y_i \tag{2-45}$$

$$u_{k-1} = x(k-1) \tag{2-46}$$

$$F = I \tag{2-47}$$

Using this formulation, EKF outperforms almost all the other first and second-order methods. The speed of convergence of the EKF is much higher than other methods [22]; however, the main issues are the calculation complexity and memory requirements. In EKF, there is a matrix inverse operation, which has a high computational burden. The covariance matrix is a large matrix $(n \times n)$ and becomes more challenging for bigger problems as concluded in [19]. These limitations make EKF suitable only for small size NNs.

2.2.2.2 Kalman Based Stochastic Gradient Descent

A recent method that tried to introduce the second order information using the Kalman filter to the SGD method is called the Kalman Based Stochastic Gradient Descent (kSGD) [62]. This effort was done in order to solve two existing problems in current SGD methods: (1) the lack of fast or justified stop condition, and (2) sensitivity to the objective function [62]. While kSGD was able to solve these problems, it was shown that kSGD has the same two

major problems as applying EKF on NN training. kSGD has much higher computational requirements than SGD methods as it requires $O(n^2)$ FP operations compared with $O(n)$ for SGD. Furthermore, high memory requirement of $O(n^2)$ compared with $O(n)$ for SGD method.

2.2.2.3 Smooth Variable Structure Filter

Smooth variable structure filter (SVSF) [21] is based on Variable Structure Filter (VSF), which is introduced in [63]. The SVSF operates in a predictor-corrector fashion similar to the KF and the EKF. The SVSF has shown to be advantageous in terms of stability and robustness against modeling uncertainties and noise [21, 63]. These characteristics give SVSF an advantage in applications where robustness and noise rejection are important requirements. SVSF can be applied to linear and non-linear systems. Versions of the SVSF using a variable and optimal boundary layer are proposed in [64-66]. The SVSF has been applied to different applications, such as trajectory tracking, fault detection and diagnosis, condition monitoring as well as NN training [22, 66-69].

For non-linear systems such as the one defined in Equations (2-32), and (2-33), the SVSF formulation consists of a prediction step followed by an update step similar to the EKF. The prediction step is defined as follows,

$$\hat{s}_{k|k-1} = f\left(\hat{s}_{k-1|k-1}, u_{k-1}\right) \tag{2-48}$$

$$\hat{z}_{k|k-1} = h\left(\hat{s}_{k|k-1}\right) \tag{2-49}$$

The update step is as follows,

$$\tilde{y}_{k|k-1} = z_k - \hat{z}_{k|k-1} \tag{2-50}$$

$$K_k = H_k^+ \left(\left| \tilde{y}_{k|k-1} \right|_{abs} + \gamma \left| \tilde{y}_{k-1|k-1} \right|_{abs} \right) \circ sat\left(\frac{\tilde{y}_{k|k-1}}{\psi} \right) \tag{2-51}$$

$$\hat{s}_{k|k} = \hat{s}_{k|k-1} + K_k \tag{2-52}$$

$$\hat{z}_{k|k} = h\left(\hat{s}_{k|k} \right) \tag{2-53}$$

$$\tilde{y}_{k|k} = z_k - \hat{z}_{k|k} \tag{2-54}$$

where (H_k^+) is the pseudoinverse of (H_k) which is defined in Equation (2-43), (γ) is the a posteriori error tuning factor, and (ψ) is the smoothing boundary layer width. In general, (γ) is a diagonal matrix and (ψ) is a vector. The saturation function works as presented in Equation (2-55) where the division is done elementwise. The operator $(\circ)$ is element-wise (Schur) multiplication. The absolute values in Equation (2-51) are elementwise absolute values of the error term.

$$sat\left(\frac{\tilde{y}_{k|k-1}}{\psi} \right) = \begin{cases} +1 & if \quad \left(\frac{\tilde{y}_{k|k-1}}{\psi} \right) \geq 1 \\ \left(\frac{\tilde{y}_{k|k-1}}{\psi} \right) & if \quad -1 < \left(\frac{\tilde{y}_{k|k-1}}{\psi} \right) < +1 \\ -1 & if \quad \left(\frac{\tilde{y}_{k|k-1}}{\psi} \right) \leq -1 \end{cases} \tag{2-55}$$

The linearization step for non-linear systems is similar to the EKF. The output function (h) is linearized using Equation (2-43).

Ahmed was the first to introduce the use of the SVSF for the training of NN in [22]. This application followed the same system representation specified by Equations (2-44), (2-45), (2-46), and (2-47). In [22], the performance of the SVSF in the training of NN was compared with EKF, SGD and Levenberg-Marquardt algorithms on different benchmark datasets. All

of the used networks were relatively small. The SVSF showed fast convergence and had comparable results with EKF and Levenberg-Marquardt.

2.2.2.4 Challenges of EKF and SVSF in DL

In literature, EKF and SVSF are applied to NN problems [19, 22, 58]. In general, the conclusion of those publications is that EKF and SVSF have fast convergence rates compared to other algorithms. However, this contradicts the fact EKF and SVSF are not commonly used in practice. Although the validation and training classification error rate measures are important performance factors, they do not fully address the issue of computational complexity and time.

Both the EKF and the SVSF as NN training optimization algorithms, require the calculation of the Jacobian matrix according to Equations (2-43) and (2-45) for the output function (h). This is very complicated in the mini-batch setting since the size of the network output becomes very large, as below.

$$\begin{aligned} number\ of\ netowrk\ outputs\ (MB) \\ = number\ of\ neurons\ in\ output\ layer\ (O) \\ \times\ MiniBatch\ size\ (B) \end{aligned} \tag{2-56}$$

Consequently, in Equation (2-43), the gradient of each of (MB) elements of network output is evaluated with respect to all network parameters, resulting in a (MB × n) size matrix (H). This is a very large computational requirement compared to the first-order methods. In the first-order methods the gradient of the scalar loss with respect to network parameters has the size of (1 × n).

Another computational challenge in applying the original forms of the EKF and the SVSF to NN training is the matrix inversion process. This challenge has many aspects:

1) computational complexity,
2) memory requirements, and
3) numerical stability.

There are a number of numerical matrix inversion routines, but none of them addresses all three challenges at the same time: low computational complexity; low memory; and numerical stability [70-73]. From a computational complexity point of view, larger matrices are computationally much more complex than smaller ones. In Equation (2-39) of the EKF algorithm, inverting (S_k), which has the size of $(MB \times MB)$, is a very complex and memory demanding process. In SVSF, (H_k) is also of size $(MB \times n)$. Usually, (H_k) is not a full-rank matrix, as many output nodes do not get affected by all the network weights. Therefore, the Moore–Penrose pseudoinverse is often used to approximate the matrix inverse, as in Equation (2-51). In [74-76], to avoid the numerical instability of pseudoinverse calculation, a damping parameter (ρ) is introduced, as follows:

$$H^{+}(damped) = H^{t}(HH^{t} + \rho^{2}I)^{-1} \tag{2-57}$$

Including this damping factor stabilizes the calculation. The tuning of (ρ) is important: a very small value of (ρ) does not damp the singularity in the matrix, and a high value of (ρ) slows down the training because of the excessive damping.

Further to the memory requirement of the matrix inversion, the filter parameters are also very large in size. Table 2-II presents the size of the main parameters of EKF, SVSF, Adam, and SGD. First-order methods, such as Adam, require more memory for the variable

learning rates than SGD, but still far less compared to the $O(n^2)$ in EKF and $O(MB \times n)$ for SVSF. In general, from memory requirement point of view, first-order methods (such as SGD) are favored in practice over higher order or more complex methods (such as SVSF and EKF). For a network with 10^6 parameters, SGD requires about (0.004 GB), Adam requires about (0.016 GB) and only the error covariance in EKF (P) requires (4000 GB) of memory.

Table 2-II Memory Requirement comparison between EKF, SVSF, Adam, and SGD

Parameter	Size
EKF: Error Covariance matrix $(P_{k\|k})$	$O(n^2)$: n is the size of network parameters
EKF: process noise covariance (Q_{k-1})	$O(n^2)$: n is the size of network parameters
EKF: measurement noise covariance (R_{k-1})	$O(MB \times MB)$: MB definition is in Equation (2-56)
EKF: innovation covariance (S_k)	$O(MB \times MB)$
EKF: output gradient (H_k)	$O(MB \times n)$
EKF: error $(\tilde{y}_k)$	$O(MB)$
SVSF: output gradient $(H_k{}^+)$	$O(MB \times n)$
SVSF: priori error $(\tilde{y}_{k\|k-1})$	$O(MB)$
SVSF: posteriori error $(\tilde{y}_{k\|k})$	$O(MB)$
SVSF: the smoothing factor(ψ)	$O(MB)$
SVSF: posteriori error tuning factor (γ)	$O(MB \times MB)$
Adam: learning rate (η)	$O(1)$
Adam: the moving window of the first moment (v_{k+1})	$O(1 \times n)$
Adam: the moving window of the second moment (m_{k+1})	$O(1 \times n)$
Adam: first-moment window bias estimate $(\hat{v}_{k+1})$	$O(1 \times n)$
Adam: second-moment window bias estimate $(\hat{m}_{k+1})$	$O(1 \times n)$
SGD: single fixed learning rate (η)	$O(1)$

SGD: loss Gradient using MSE $\frac{\partial \mathrm{Error}(w, x^{q \to q+B}, y^{q \to q+B})}{\partial w_{i,j}^{n}}$	$0(1 \times n)$

2.3 Reduced-EKF and Reduced-SVSF

In this section, modifications to the EKF and the SVSF implementation are proposed in order to lower the memory and computational power requirements for both of these methods. The revised methods are referred to as the Reduced-EKF (REKF) and Reduced-SVSF (RSVSF) for clarity. The modifications reduce both the epoch time and the convergence performance of the respective estimators per one epoch.

2.3.1 Reduced-EKF (REKF)

The REKF combines several modifications on EKF altogether. These modifications, when applied together, are able to achieve much better performance in terms of calculation time and memory compared to the original form of the EKF. Some of these modifications are separately discussed in the literature on NN scale but have never been combined or applied on DL scale. Other new modifications are included in REKF as well. The modifications are as follows:

- Mini-Batch: the concept of applying Mini-Batch training scheme on EKF-NN training algorithm has been discussed in many publications such as [19, 20]. Training using Mini-Batches has always shown good properties in terms of parallel computation that speeds up the total training process and enables processing of large datasets.
- Training with a scalar measure of error: Using a scalar measure of error (such as MSE) solves the Jacobian computation challenges discussed in section 2.2.2.4. This

step reduces the calculations and memory requirement of the EKF by a magnitude of $O(MB)$, allowing it to be equal to the SGD gradient calculation requirements. In other words, instead of using the error defined in Equation (2-54), the error will be defined as shown in Equation (2-58) for a single training sample. (O) is the number of neurons in the output layer. The goal is to have zero MSE error in the nonlinear system. Other error or loss functions can be used such as categorical cross-entropy.

$$\tilde{y}_{k|k} = 0 - loss = 0 - MSE(target, y) = -\frac{1}{O}\sum_{i=1}^{i=O}(target_i - y_i)^2 \quad (2\text{-}58)$$

- EKF decoupling: the size of the covariance matrix in EKF introduces significant computation and memory requirement as discussed in [19]. In [20, 77], reducing the size of the covariance is investigated to reduce the requirements. This idea has been promising as not all the network parameters are interrelated and affect the same outputs. The concept is similar to approximating the Fisher matrix in the NGD algorithms to a block diagonal matrix as mentioned in section 2.2.1. Four versions of EKF decoupling are shown in Figure 2-1. In the Global-EKF, GEKF, the covariance between every two parameters is calculated [20, 77]. In the Layer-Decoupled-EKF, LDEKF, the decoupling is done layer by layer [20, 77]. The Node-Decoupled-EKF, NDEKF, takes LDEKF one step further and applies the decoupling on a per node-basis [20, 77]. The Fully-Decoupled-EKF, FDEKF, is the most extreme version of decoupling where no covariance between any two parameters is calculated [20, 77]. This is the one that is used in the proposed REKF. This method lowers the memory requirement from $O(n^2)$ to $O(n)$; however it loses

the correlation between the network parameters. The net effect is found to be favorable in REKF, as the reduction in computation time is more than the reduction in convergence performance per epoch due to the loss of correlation information. The covariance matrix in FDEKF still allows REKF to adjust network parameters with variable learning rates for individual weights.

- Reduction in noise matrices: (Q_{k-1}, R_{k-1}) are of $O(n^2)$ and $O\,(MB \times MB)$, respectively. Assuming the same noise value in the training of all network parameters, the noise value can be stored in a shared parameter and used in all the weight updates. This will reduce both matrices to a scalar value with the size of $O(1)$.

REKF algorithm steps are summarized in Table 2-III.

Table 2-III REKF algorithm

Algorithm 1: *REKF*, this proposed algorithm reduces EKF to a more memory- and calculation-efficient algorithm to be used for optimizing DL networks

Require: $\mathbf{q}, \mathbf{r}$ **values**

Require: $\mathbf{p}$ initial values

$\mathbf{P} \leftarrow \begin{bmatrix} \mathbf{p} \\ \vdots \\ \mathbf{p} \end{bmatrix}$ Initialize $\mathbf{P}$ matrix as a vector filled with $\mathbf{p}$ initial value.

While $\mathbf{k} <$ **number of epochs**

$\mathbf{k} \leftarrow \mathbf{k} + \mathbf{1}$

 # calculate the network loss

$\mathbf{h}(\mathbf{s_{k-1}}) = \mathbf{loss}(\mathbf{s_{k-1}}) = \mathbf{MSE}(\mathbf{target}, \mathbf{network\ output})$

 # the system target is to achieve zero loss, hence the system error is

$\tilde{\mathbf{y}}_\mathbf{k} = \mathbf{0} - \mathbf{loss} = (\mathbf{system\ target} - \mathbf{system\ output})$

$\mathbf{H_k} = \nabla_\mathbf{s}\mathbf{loss}(\mathbf{s_{k-1}})$

$\mathbf{P_{k|k-1} = P_{k-1|k-1} \oplus q}$

#perform the update step

$\mathbf{S_k = H_k P_{k|k-1} H_k^T + r}$

$\mathbf{K_k = P_{k|k-1} H_k^T \oslash S_k}$

$\mathbf{s_k = s_{k-1} + K_k \tilde{y}_k}$

$\mathbf{P_{k|k} = (I - K_k H_k) P_{k|k-1}}$

End while

Return $\mathbf{s_k}$

In the above equations (s) is a vector of the network parameters. In the state space system, the states are assigned to be the parameters of the network (s). Applying all the above mentioned modifications, matrix (S_k) in Equation (2-39) becomes a scalar value that eliminates the matrix inversion requirement with all its computational and memory challenges but results in a loss of optimality. In Equation (2-59) H and P dimensions mathematically are (1 × n) and (n × n) respectively. r is a scalar value. However, because P is diagonal and H is a vector then to simplify the implementation this equation can be expressed as shown in Equation (2-60) and P is stored in a vector.

$$S_k = H_k P_{k|k-1} H_k^t + r_{k-1} \tag{2-59}$$

$$S_k = \left(\sum_{idx=1}^{idx=n} (H_k(idx))^2 P_{k|k-1}(idx, idx) \right) + r_{k-1} \tag{2-60}$$

As shown in Table 2-IV, the memory requirement for REKF is very comparable to the first-order training algorithms and is less than a number of them. Considering the aforementioned example (for $n = 10^6$), REKF requires about 0.008 GB only (around

$O(2 \times n)$). This is half of the memory capacity that Adam requires (around $O(4 \times n)$) and much less than what EKF requires (more than $O(n^2)$).

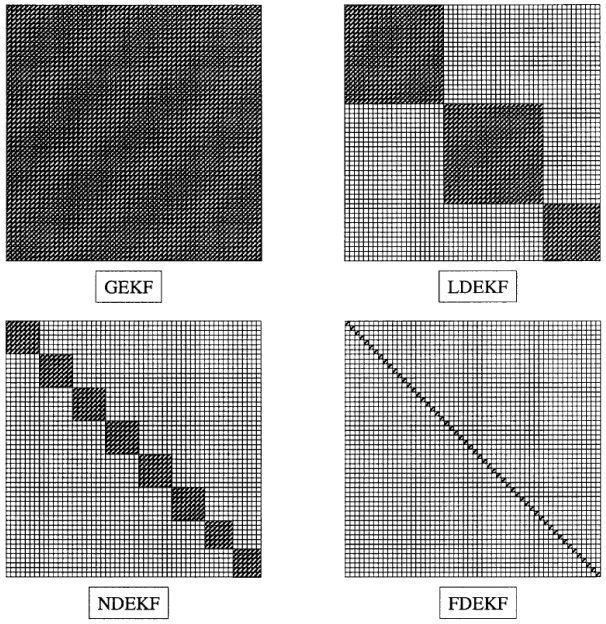


Figure 2-1 Covariance matrices of different versions of EKF [20]

Table 2-IV Memory requirement for REKF

Parameter	Size
REKF: Covariance matrix ($P_{k\|k}$)	$O(n)$
REKF: process noise covariance (Q_{k-1})	$O(1)$
REKF: measurement noise covariance (R_{k-1})	$O(1)$
REKF: innovation covariance (S_k)	$O(1)$
REKF: output gradient (H_k)	$O(n)$
REKF: error ($\tilde{y}_k$)	$O(1)$

2.3.2 Reduced-SVSF (RSVSF)

Similar to REKF, SVSF is implemented by simplifying SVSF-NN algorithm. SVSF is tested on the NN scale in [22]; however, it has never been generalized for DL scale.

Compared to EKF, SVSF has fewer memory requirements, since its algorithm in its original form does not include any covariance matrix calculation [21]. However, the SVSF requires one more forward computation in every epoch to calculate the a posteriori error. The modifications on the SVSF toward RSVSF are as follows.

- Mini-Batch: the concept of applying a Mini-Batch training scheme on SVSF-NN training algorithms is discussed in [22]. This concept is used in RSVSF for its faster computation time due to parallel computation.
- Training with a scalar error: similar to REKF, reducing the error size to a scalar value using a statistic measure such as MSE in Equation (2-58), dramatically reduces the computation. The size of the Jacobian matrix reduces from $O(MB \times n)$ to $O(n)$, which is the same as the SGD gradient size.
- Reduction of tuning factors (γ, ψ) size: similar to the noise matrices in REKF, if both (γ, ψ) are assumed to have the same value for all network parameters, a shared memory can be used that reduces them to a scalar size. This modification saves $O((MB)^2 + MB)$ in memory size.
- A posteriori error computation elimination in case (γ) is zero: (γ) is a tuning parameter that is defined as $0 \leq \gamma < 1$. When (γ) is zero, the a posteriori error computation is not required anymore as it does not affect the result. With $(\gamma = 0)$, the a posteriori error term in SVSF gain Equation (2-51) is eliminated that saves one forward computation. The gain equation, in this case, is shown in Equation (2-61).

- Addition of a learning rate (η) similar to the first-order methods: in some occasions, it is preferred to take a larger step in weight modifications direction to learn faster. The absence of the a posteriori term reduces the step size. Experimentally, the a posteriori error has a value that is close to the a priori error (especially at later epochs), therefore a new term called learning rate term (η) can be added to the a priori term. This term provides the same effect without adding an extra forward computation to calculate the a posteriori error as shown in Equation (2-62). The definition of (η) to sustain the stability is $1 \leq \eta < 2$. The learning rate (η) is introduced in Equation (2-62), the RSVSF gain. The effect of adding (η) to the gain on the SVSF stability is discussed in Appendix B.
- Last modification for RSVSF is the variable boundary layers width (ψ). A decay option is given to quickly reduce (ψ) over iterations. This is useful as it accelerates the learning in the beginning of the training phases and allows for using tighter (ψ) values for fine-tuning. The difference between the dynamic and fixed boundary layer width is visualized in Figures 4 and 5. The RSVSF algorithm steps are summarized in Table 2-V. The algorithm uses the epoch number for the decaying rate. If the training is operated in mini-batch fashion then epoch number is replaced with the batch number.

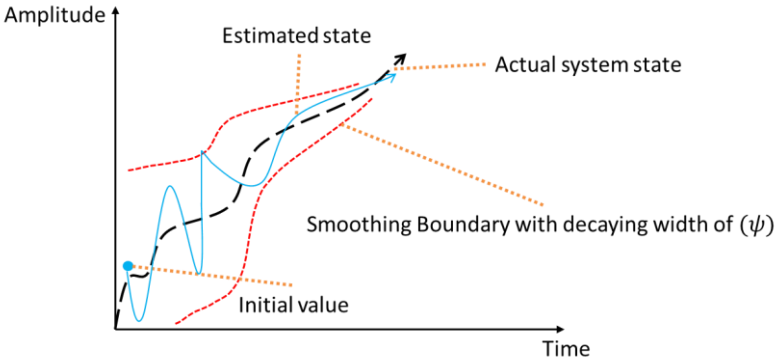


Figure 2-2 Decaying boundary layer width

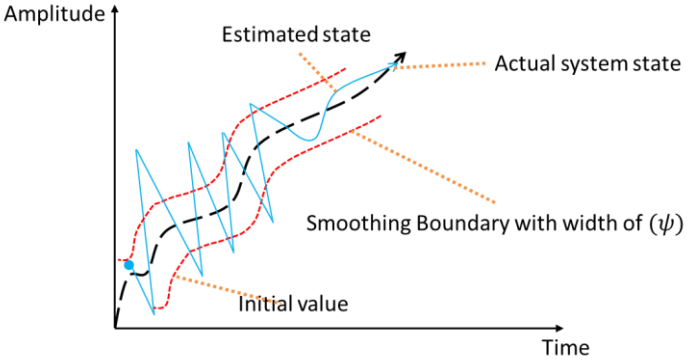


Figure 2-3 Fixed boundary layer width

Table 2-V RSVSF algorithm

Algorithm 2: *RSVSF*, this proposed algorithm reduces SVSF to a more memory- and calculation-efficient algorithm to be used for optimizing DL networks

Require: $\boldsymbol{\gamma}, \boldsymbol{\psi}, \boldsymbol{\eta}$: **values**. **decay**: **boolean**

While $\mathbf{k} <$ **number of epochs**

$\mathbf{k} \leftarrow \mathbf{k} + \mathbf{1}$

 # calculate the network loss

$\mathbf{h}(\mathbf{s_{k-1}}) = \mathbf{loss}(\mathbf{s_{k-1}}) = \mathbf{MSE}(\mathbf{target}, \mathbf{network\ output})$

 # the system target is to achieve zero loss, hence the system error is

$\tilde{\mathbf{y}}_{\mathbf{k}} = \mathbf{0} - \mathbf{loss} = (\mathbf{system\ target} - \mathbf{system\ output})$

$\mathbf{H_k} = \nabla_{\mathbf{s}}\mathbf{loss}(\mathbf{s_{k-1}})$

$$\mathbf{H}_{\mathbf{k}}^{+} = \frac{\mathbf{H}_{\mathbf{k}}^{\mathbf{t}}}{\sum_{\mathbf{i}=\mathbf{1}}^{\mathbf{n}} \left[\left(\mathbf{H}_{\mathbf{k}}(\mathbf{i}) \right)^{\mathbf{2}} \right]}$$

$$\mathbf{K}_{\mathbf{k}} = \mathbf{H}_{\mathbf{k}}^{+} \left(\boldsymbol{\eta} \left| \tilde{\mathbf{y}}_{\mathbf{k}|\mathbf{k}-\mathbf{1}} \right|_{\mathbf{abs}} + \boldsymbol{\gamma} \left| \tilde{\mathbf{y}}_{\mathbf{k}-\mathbf{1}|\mathbf{k}-\mathbf{1}} \right|_{\mathbf{abs}} \right) \circ \mathbf{sat} \left(\frac{\tilde{\mathbf{y}}_{\mathbf{k}|\mathbf{k}-\mathbf{1}}}{\boldsymbol{\psi}} \right)$$

#perform the update step

$$\mathbf{s}_{\mathbf{k}} = \mathbf{s}_{\mathbf{k}-\mathbf{1}} + \mathbf{K}_{\mathbf{k}}$$

if decay is true

$$\boldsymbol{\psi} = \mathbf{max}\left(\frac{\boldsymbol{\psi}}{\mathbf{1} + \mathbf{epochNum} \times \mathbf{0.48}} \text{-}\mathbf{1}, \boldsymbol{\psi} \times \mathbf{10}^{-\mathbf{5}} \right)$$

End if

End while

Return $\mathbf{s}_{\mathbf{k}}$

In the above equations (s) is a vector of the network parameters. In the state space system, the states are assigned to be the parameters of the network (s). Similar to the REKF, applying these modifications eliminates the matrix inversion process as (H_k) becomes a vector and Equation (2-57) becomes equivalent to Equation (2-63) assuming that $(\rho) = 0$, as damping is not needed anymore for matrix inversion damping. Equation (2-63) is the definition of pseudoinverse in the vector case. This form is similar to the other first-order methods that tried to substitute the hessian matrix with the squared gradient, but in this case, the normalization is done by the sum of the squares of the gradient elements that smooths out the changes. Another difference from the first-order methods is the use of the saturation function of the SVSF in Equation (2-62).

These modifications reduce the time and memory requirements significantly for RSVSF compared to SVSF and allow RSVSF to be compete with the first-order methods.

$$K_k = \begin{cases} H_k^+ \left(\left| \tilde{y}_{k|k-1} \right|_{abs} + \gamma \left| \tilde{y}_{k-1|k-1} \right|_{abs} \right) \circ sat\left(\frac{\tilde{y}_{k|k-1}}{\psi} \right) & if \quad \gamma \neq 0 \\ H_k^+ \left(\left| \tilde{y}_{k|k-1} \right|_{abs} \right) \circ sat\left(\frac{\tilde{y}_{k|k-1}}{\psi} \right) & if \quad \gamma = 0 \end{cases} \tag{2-61}$$

$$K_k = \begin{cases} H_k^+ \left(\eta \left| \tilde{y}_{k|k-1} \right|_{abs} + \gamma \left| \tilde{y}_{k-1|k-1} \right|_{abs} \right) \circ sat\left(\frac{\tilde{y}_{k|k-1}}{\psi} \right) & if \quad \gamma \neq 0 \\ H_k^+ \left(\eta \left| \tilde{y}_{k|k-1} \right|_{abs} \right) \circ sat\left(\frac{\tilde{y}_{k|k-1}}{\psi} \right) & if \quad \gamma = 0 \end{cases} \tag{2-62}$$

$$H_k^+ = \frac{H_k^t}{\sum_{i=1}^{n} \left[\left(H_k(i) \right)^2 \right]} \tag{2-63}$$

In Table 2-VI, the memory requirements for RSVSF are shown. It can be noticed that they are less than REKF by $O(n)$ and less than Adam by $O(3 \times n)$ as shown in Table 2-II.

Table 2-VI Memory requirements for RSVSF

Parameter	Size
SVSF: output gradient (H_k^+)	$O(n)$
SVSF: a priori error ($\tilde{y}_{k\|k-1}$)	$O(1)$
SVSF: posteriori error ($\tilde{y}_{k\|k}$)	$O(1)$
SVSF: the smoothing factor(ψ)	$O(1)$
SVSF: a posteriori error tuning factor (γ)	$O(1)$

2.4 Experiments

In this section, a comparison between adaptive first-order algorithms is performed for choosing a representative first-order algorithm. Furthermore, REKF and RSVSF are compared to EKF and SVSF using the Iris dataset. The reason for using the Iris dataset is that the SVSF and the EKF cannot be practically used on any larger datasets, due to all their memory and computational limitations. Lastly, a set of experiments on different datasets

and architectures are performed for comparing the REKF, the RSVSF and the above-mentioned representative first-order algorithm. The comparison of the algorithms and tests are summarized in Table 2-XIV.

2.4.1 First-order algorithms experiment

This section considers finding the best first-order algorithm as a benchmark for its comparison against the REKF and the RSVSF on different datasets and architectures. A comparison between the adaptive learning rate algorithms, namely, Adam, Adamax, Adagrad, Adadelta, and RMSprop is performed using the standard MNIST dataset [31]. A DNN with two hidden layers (768-512-256-10) is used. The network used Relu as activation function on the hidden layers and Softmax on the output layer. The test was performed ten times to evaluate the variation across different runs as well. Figures 6 and 7 show the fast convergence rate and the low variance of Adam in this test.

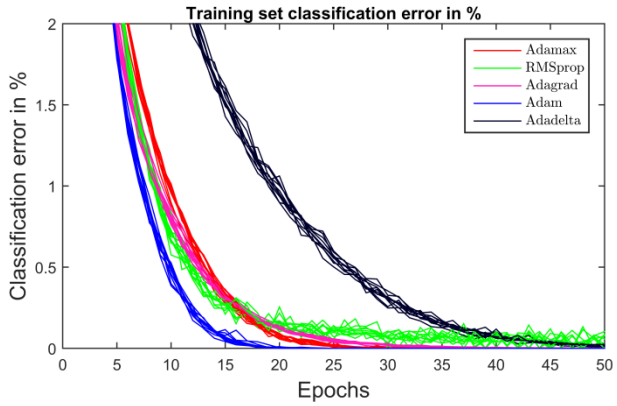


Figure 2-4 First-order algorithms training set classification error in %

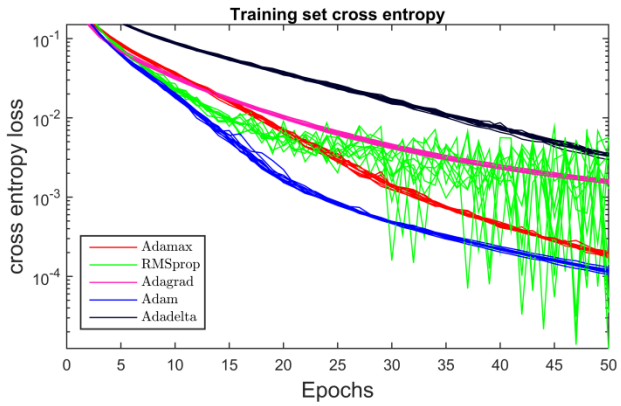


Figure 2-5 First-order algorithms training set categorical cross-entropy loss

It is important to highlight that the authors do not claim that Adam always performs better than the other algorithms across all datasets and architectures. However, the results show that in this case Adam demonstrates more favorable characteristics given its comparatively faster and smoother convergence. Therefore, Adam is used as the benchmark algorithm for comparison with estimation based methods in this paper.

Table 2-VII First-order algorithms configurations

Optimizer	Configuration
Adam	$\beta_1 = 0.9\,,\ \beta_2 = 999,\ \eta = 0.001,\ \epsilon = 1e-8$
Adadelta	$\gamma = 0.95\,,\ \epsilon = 1e-8$
Adamax	$\beta_1 = 0.9\,,\ \beta_2 = 999,\ \eta = 0.002$
Adagrad	$\eta = 0.01\,,\ \epsilon = 1e-8$
RMSprop	$\eta = 0.001\,,\ \epsilon = 1e-8,\ \gamma = 0.9$

2.4.2 Iris dataset experiment

EKF and SVSF cannot be applied to large networks using large datasets due to their computational constraints. To enable a comparison of the original EKF and SVSF with the proposed REKF and RSVF, This study uses the simple and standard Iris dataset to provide

a feasible problem size for EKF and SVSF application. This experiment is performed to compare the performance of REKF and RSVSF with their original forms of EKF and SVSF. A simple one hidden layer with 10 units FNN is used. For 5000 epochs, EKF and SVSF took two orders of magnitude more time than the REKF and RSVSF. Figure 2-6 shows the training categorical cross-entropy loss normalized to time and examined for the first 5.5 seconds (the full duration for the 5000 epochs for REKF, RSVSF). REKF and RSVSF show fast convergence. The used configurations are shown in Table 2-VIII.

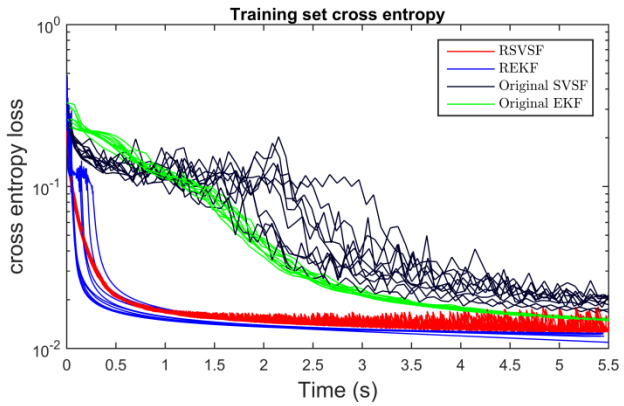


Figure 2-6 REKF, RSVSF, EKF, and SVSF categorical cross-entropy loss on Iris dataset

After the 5000 epochs EKF and SVSF show great convergence (as seen in Table 2-XIV). However because the processing time is very slow (~370 seconds compared with 5.5 seconds), the convergence rate per unit time is in favor of the proposed algorithms. In Section 2.3, a hypothesis was made that REKF and RSVSF can improve the net performance compared to their original forms. Figure 2-6 confirms this hypothesis. This improvement happens because the effect of faster epoch times masks the slight performance reduction due to the second order information loss.

Table 2-VIII REKF, RSVSF, EKF and SVSF configurations for Iris dataset experiment

Optimizer	Configuration
REKF	$Q = 2e-2$,$P = 5e3$, $R = 10$
RSVSF	$\gamma = 0, \psi = 3, \eta = 1$
Original EKF	$Q = 20$,$P = 50$, $R = 1000$
Original SVSF	$\gamma = 0, \psi = 1e-8, \eta = 1.99$

2.4.3 Benchmark datasets

The results in the last section demonstrate the performance improvement of REKF and RSVSF over their original forms. This lays the basis for using REKF and RSVSF in larger problems, where the usage of EKF and SVSF is infeasible. In this section, REKF and RSVSF are compared with Adam on larger scale problems using deep networks as opposed to the shallow one used in Section 2.4.2. In the first-order algorithms experiment in Section 2.4.1, Adam showed smooth and fast convergence. Therefore, Adam is chosen to represents first-order algorithms. In this experiment, a total of 9 experiments were performed and each one is repeated with random initialization ten times. Five datasets are tested, namely, MNIST [31], FashionMNIST [78], EMNIST[79], CIFAR10 and CIFAR100 [80]. Two architectures are tested, namely DNN and CNN. In this section, training set charts are presented to show the fast convergence of the proposed algorithms.

2.4.3.1 MNIST Dataset

The dataset here is the standard benchmark MNIST dataset [31] with 60,000 training samples and 10,000 test samples. For the MNIST dataset, two tests were performed. Both of them used DNN with three hidden layers with the size of 1024 units. The difference

between the two tests is using the dropout technique [81]. The purpose of these tests is to show that REKF and RSVSF still maintain an advantage over Adam even with the existence of dropout. The configuration of RSVSF and REKF for the two tests are shown in Table 2-IX.

Table 2-IX REKF and RSVSF configurations for MNIST dataset experiment

Optimizer	Configuration
REKF	$Q = 2e5, P = 5e3, R = 1e-3$
RSVSF	$\gamma = 0, \psi = 2e-8, \eta = 1.99$

Figure 2-7 shows the training set classification error rate. It can be seen that RSVSF and followed by REKF converged very quickly and consistently in the ten runs while Adam experienced some variance across different runs. Figure 2-8 shows the categorical cross-entropy loss on the training set. REKF and RSVSF still achieve less error at the end of the training, however, REKF had some challenges converging between epoch 60 to epoch 140. After epoch 140, REKF converged quickly and dropped below Adam. From variance across different runs point of view, REKF and RSVSF still have better consistency. In a classic paper [81], Hinton *et al.* claimed that the test classification error rate benchmark of a feedforward network on MNIST dataset before their publication was 1.6% without:

1) Enhancing the training dataset
2) Using spatial knowledge by using Convolutional Neural Network (CNN)
3) Using generative pre-training

Then, Hinton achieved 1.3% with 50% dropout in the hidden layers and L^2 regularization (improvement of 0.3%). RSVSF across the ten runs recorded an average of (1.34%) of

validation classification error rate without dropout or L^2 regularization. This is a reduction of (0.26%). Dropout is usually used to increase the test performance. Dropout layers were added to the second test to examine its effect on the convergence of the optimization algorithms. RSVSF and REKF still produce faster convergence rates than Adam. In Figures 11 and 12 RSVSF and REKF produce better results than Adam at epoch 20 and epoch 40 respectively and they sustain the good results afterwards. In Figure 2-10 the variation in EKF is the least among the three algorithms. The dropout layers reduced the validation classification rate from (1.34%) to (1.18%) for RSVSF and from (1.62%) to (1.37%) for REKF.

Table 2-X Architecture of the DNN networks used for MNIST dataset

Test	Layers	Activation function
Test#1	784-1024-1024-1024-10	Relu-Relu-Relu -Softmax
Test#2	784-1024-1024-1024-10	Relu-Dropout(0.3)-Relu-Dropout(0.3)-Relu-Dropout(0.3)-Softmax

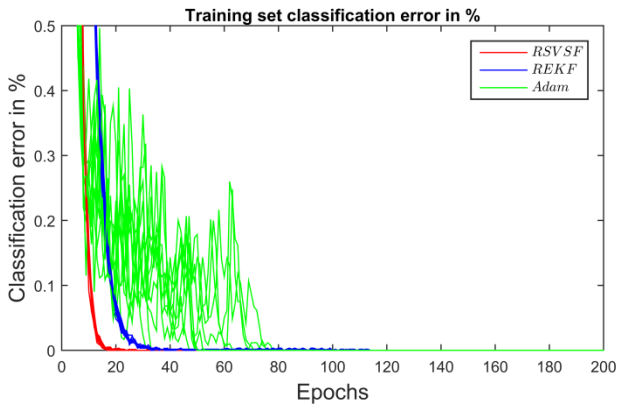


Figure 2-7 Test#1 (MNIST) training set classification error in %

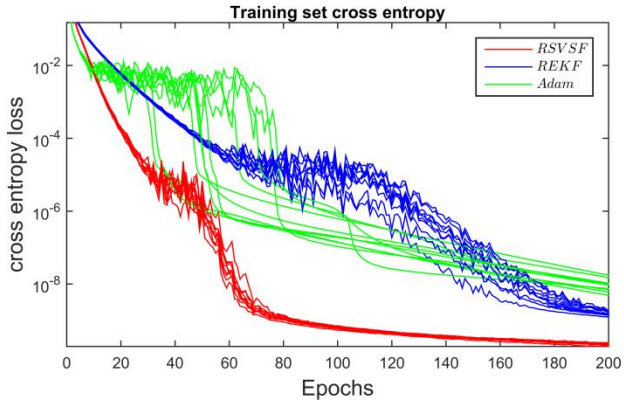


Figure 2-8 Test#1 (MNIST) training set categorical cross-entropy loss

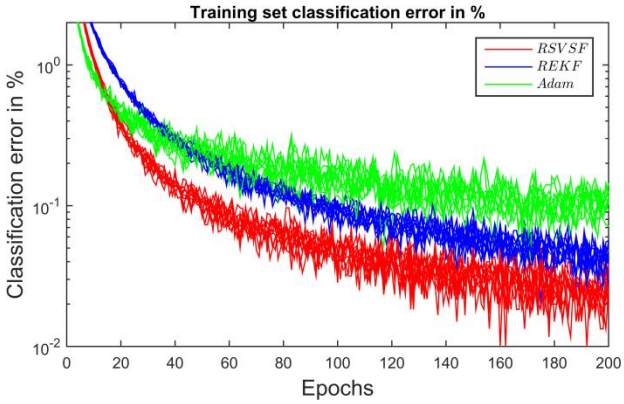


Figure 2-9 Test#2 (MNIST) training set classification error in %

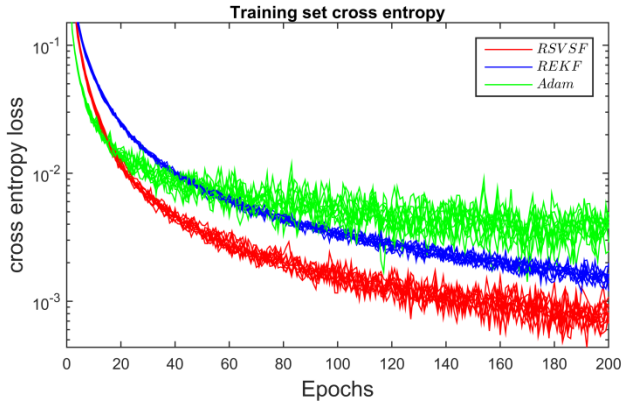


Figure 2-10 Test#2 (MNIST) training set categorical cross-entropy loss

2.4.3.2 EMNIST Dataset

An extension of the MNIST dataset is called EMNIST [82]. In this dataset, a sub-dataset includes 28x28 pixels images for handwritten letters instead of numbers in MNIST. EMNIST includes 124,800 training samples and 20,800 testing samples. A DNN network is used for this classification task as well. The architecture is presented in Table 2-XI. Figures 13 and 14 show a clear advantage of REKF and RSVSF over Adam. Both classification error and categorical cross-entropy loss are much lower for REKF and RSVSF. In this test, all the algorithms produced consistent results across the ten runs.

Table 2-XI Architecture of the DNN networks used for EMNIST dataset

Test	Layers	Activation function
Test#3	3072-2048-1024-512-256-128-10	Relu-Relu-Relu-Relu-Relu-Softmax

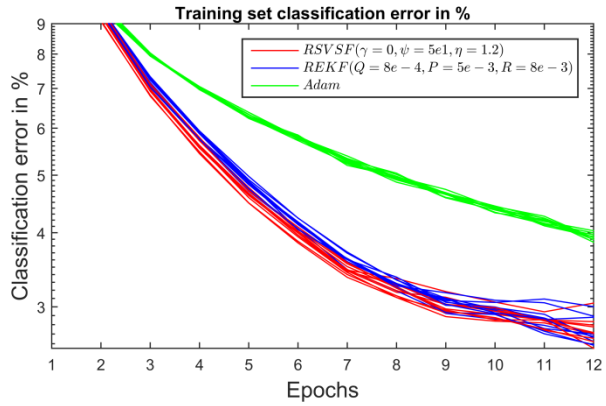


Figure 2-11 Test#3 (EMNIST) training set classification error in %

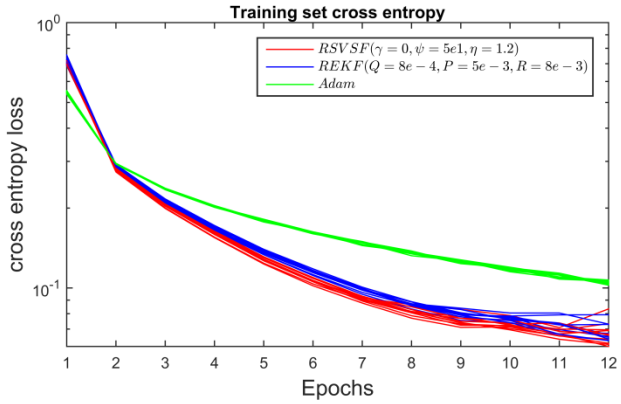


Figure 2-12 Test#3 (EMNIST) training set categorical cross-entropy loss

2.4.3.3 FashionMNIST Dataset

Another variation on MNIST dataset is the Fashion-MNIST [9]. It has the same samples size as MNIST dataset, 60,000 training samples, and 10,000 testing samples. The samples in this dataset are 28x28 grayscale pictures of different fashion pieces. Fashion-MNIST has 10 classes. Using this dataset, two experiments are performed to test different architectures, DNN and CNN. Table 2-XII shows the architectures for the DNN and the CNN networks. Figures 15 and 16 show the consistency of REKF and RSVSF in achieving better convergence in the DNN test (test#4). RSVSF is slightly better than EKF in the higher epochs. Figures 17 and 18 show consistent RSVSF and REKF convergence in the case of a CNN network as well. RSVSF outperformed the other algorithms in the most of the training process.

Table 2-XII Architecture of the networks used for Fashion-MNIST dataset

Test	Layers	Activation function
Test#4 (DNN)	3072-2048-1024-512-256-128-10	Relu-Relu-Relu-Relu-Relu-Softmax

Test#5 (CNN)	(32x3x3)-(32x3x3)-(2x2)- (64x3x3)- (64x3x3)-(2x2)-512-10	Conv(Relu)- Conv(Relu)- Pooling(Relu)- Conv(Relu)- Conv(Relu)- Pooling(Relu) - flatten()- Dense(Relu)- Dense (Softmax)

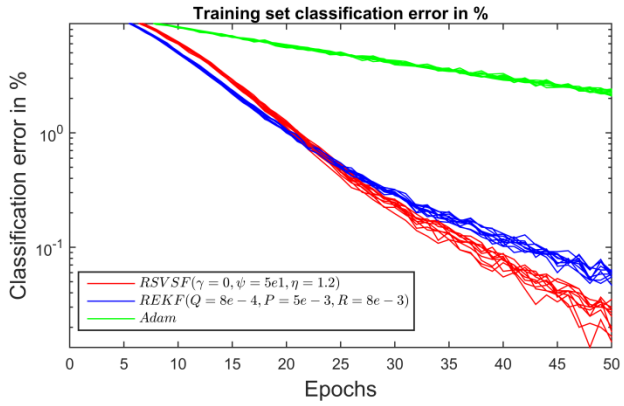


Figure 2-13 Test#4 (Fashion-MNIST DNN) training set classification error in %

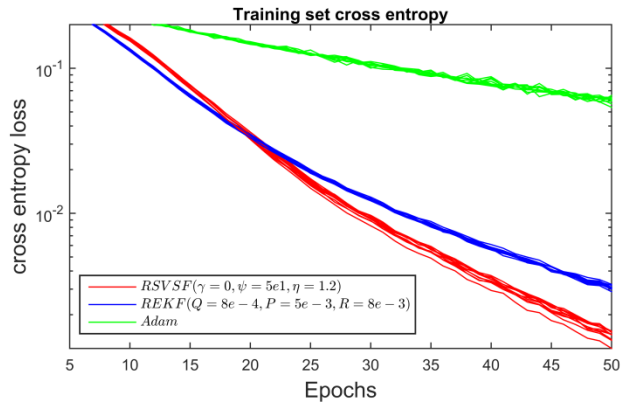


Figure 2-14 Test#4 (Fashion-MNIST DNN) training set categorical cross-entropy loss

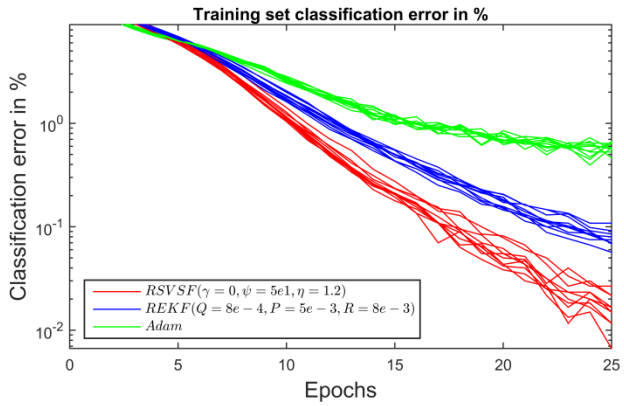


Figure 2-15 Test#5 (Fashion-MNIST CNN) training set classification error in %

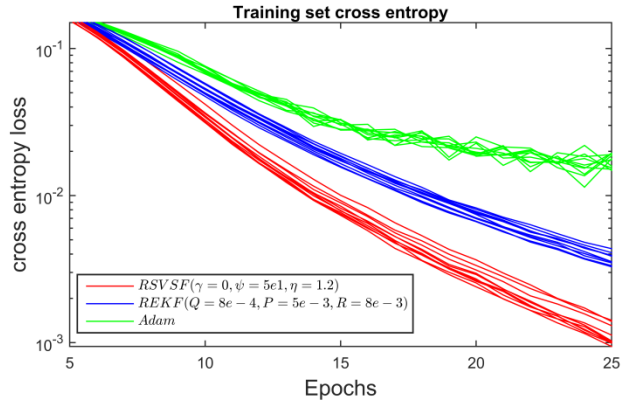


Figure 2-16 Test#5 (Fashion-MNIST CNN) training set categorical cross-entropy loss

2.4.3.4 CIFAR10 Dataset

CIFAR10 [83] is a dataset of 32x32 color images. There are 10 classes in this dataset for different animals and objects. CIFAR10 has 50,000 training samples and 10,000 testing samples. In this experiment as well, two tests are performed to test the proposed algorithms across different architectures, DNN and CNN. Figures 19, 20, 21 and 22 show the same consistency REKF and RSVSF show on the other datasets. However, in the DNN test, REKF showed much better convergence.

Table 2-XIII Architecture of the networks used for CIFAR10 dataset

Test	Layers	Activation function
Test#6 (DNN)	3072-2048-1024-512-256-128-10	Relu-Relu-Relu-Relu-Relu-Softmax
Test#7 (CNN)	(32x3x3)-(32x3x3)-(2x2)-(64x3x3)- (64x3x3)-(2x2)-512-10	Conv(Relu)- Conv(Relu)- Pooling(Relu)-Dropout(0.25)- Conv(Relu)- Conv(Relu)- Pooling(Relu)- Dropout(0.25)- flatten()- Dense(Relu) -Dropout(0.25)- Dense(Softmax)

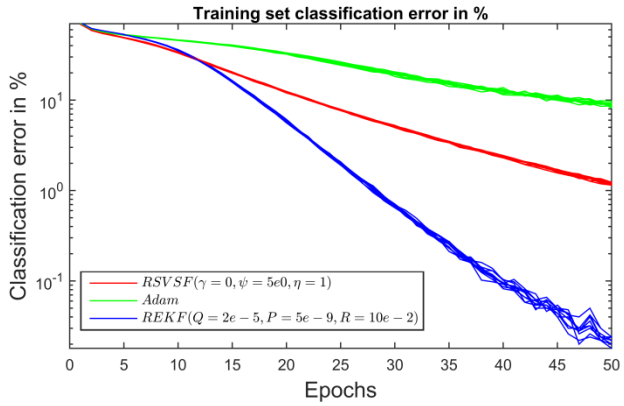


Figure 2-17 Test#6 (CIFAR10 DNN) training set classification error in %

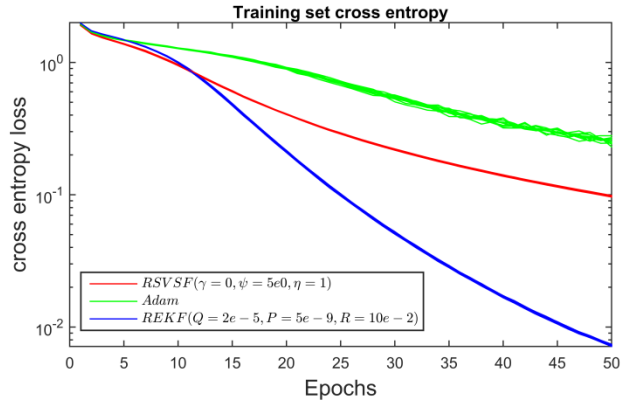


Figure 2-18 Test#6 (CIFAR10 DNN) training set categorical cross-entropy loss

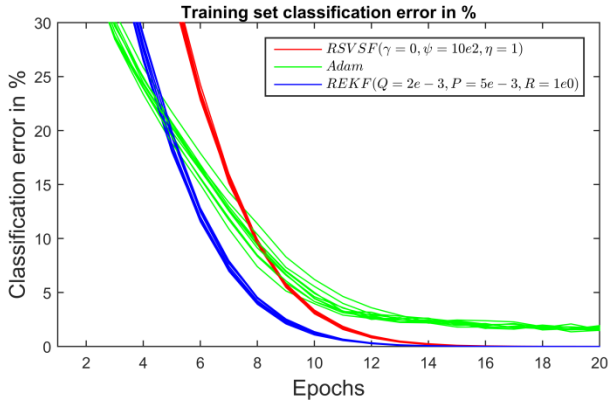


Figure 2-19 Test#7 (CIFAR10 CNN) training set classification error in %

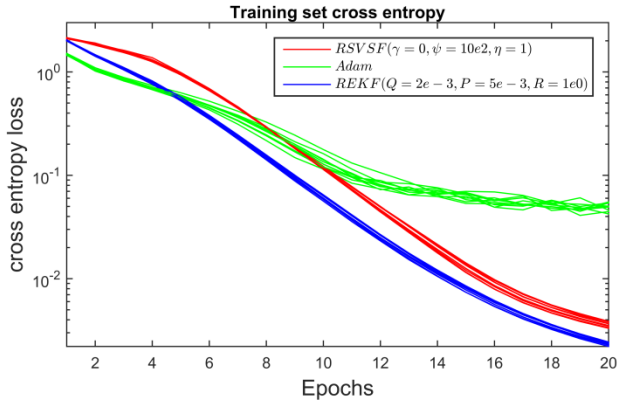


Figure 2-20 Test#7 (CIFAR10 CNN) training set categorical cross-entropy loss

2.4.3.5 CIFAR100 Dataset

CIFAR100 [80] is similar to CIFAR10 with one exception which is the number of classes. In CIFAR100 there are 100 classes, however the same total number of training and testing samples. Therefore the available number of images per class for training and testing is less. Similar to CIFAR10, both DNN and CNN are tested. Test#8 and Test#9 have the same architectures of Test#6 and Test#7 respectively (see Table 2-XIII for reference). The only difference is in the output layer number of units as it becomes 100 instead of 10. Figures 23 and 24 show the consistency that REKF and RSVSF achieved in both the classification

error rate and as well in the categorical cross-entropy loss. However, RSVSF, REKF, and Adam started with relatively similar performance, RSVSF and REKF converged faster after epoch 20. REKF outperformed Adam and RSVSF in this experiment. It is worth noting that in one of the ten runs Adam has diverged at the beginning of the training as can be seen in the Figures. For the CNN architecture, both RSVSF and REKF perform better than Adam. The results of the CNN architecture can be seen in Figures 25 and 26.

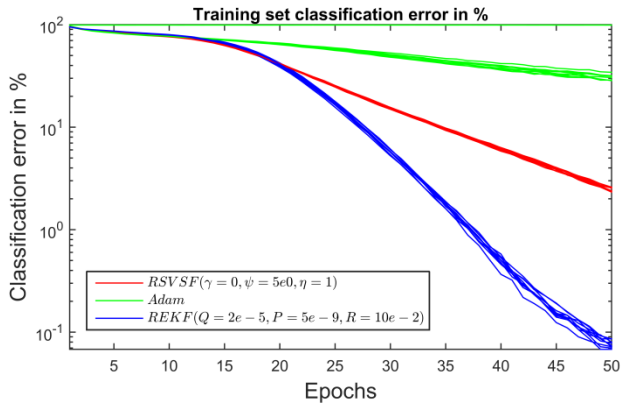


Figure 2-21 Test#8 (CIFAR100 DNN) training set classification error in %

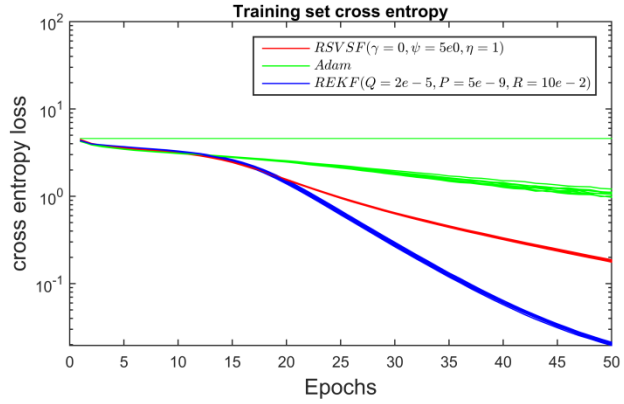


Figure 2-22 Test#8 (CIFAR100 DNN) training set categorical cross-entropy loss

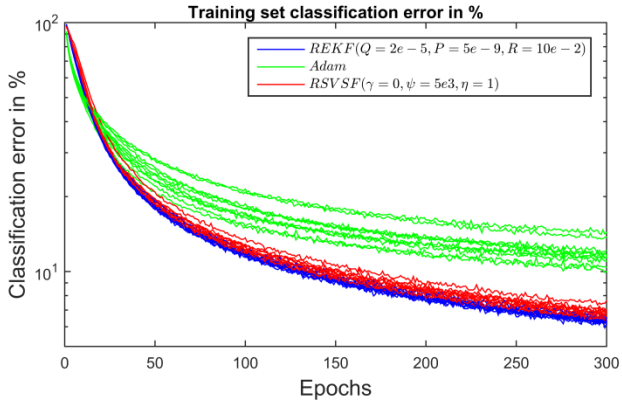


Figure 2-23 Test#9 (CIFAR100 CNN) training set classification error in %

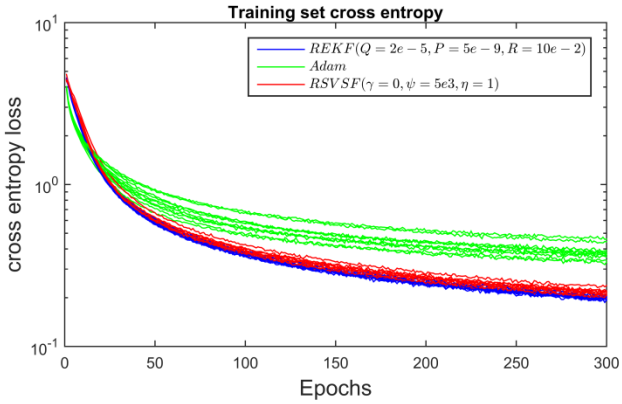


Figure 2-24 Test#9 (CIFAR100 CNN) training set categorical cross-entropy loss

2.4.3.6 Results summary and discussion

In all tests, REKF and RSVSF showed a great improvement in optimization performance over first-order methods (represented by Adam). Some of the favorable characteristics of REKF and RSVSF are as follows:

- Fast convergence
    - Factor 1: Fast convergence per epoch – faster decay of training losses
    - Factor 2: Fast epoch time

  - Factor 3: sustainable improvement of convergence over transient and steady-state phases
- Using less memory than Adam (both of REKF and RSVSF)

A summary of the tests is presented In Table 2-XIV. The reduction in processing time allows RSVSF to be consistently faster than Adam (around 17% faster than Adam in MNIST test#2). In Table 2-XIV,

first and second columns are the mean of the ten runs for the minimum of the training classification error rate and training categorical cross-entropy loss values. Third and fourth columns are their Relative Standard Deviation (RSD) respectively. The last column is the mean of the processing time across the ten runs.

The proposed methods, REKF and RSVSF, have proven to outperform the current state-of-the-art training optimization algorithms in many aspects, including memory usage, speed, and convergence performance. This better performance may be attributed to a result of the inherent characteristics of noise filtering. The hypothesis that the proposed REKF and RSVSF algorithms can outperform their original EKF and SVSF forms has proven to be valid, as the reduction in performance is negligible compared to the significant reduction in processing time per epoch.

Table 2-XIV Summary of all tests statistics

Optimizer	Min training classification error rate	Min training categorical cross-entropy loss	training classification error rate RSD	Min training categorical cross-entropy loss RSD	Processing time (s)
First-order algorithms experiment					

Adam	**0**	1.18e-04	**0** %	5.79	22.5133
Adamax	**0**	1.87e-04	**0** %	5.59 %	22.1390
Adadelta	1.3667e-04	0.0034	0.0039 %	5.24 %	**21.0650**
RMSprop	1.6667e-06	**4.57e-05**	5.27e-04 %	69.71 %	21.1813
Adagrad	3.33e-06	0.0016	7.03e-04 %	**4.20 %**	21.1846
Iris dataset experiment					
EKF	0.0029	0.0019	0.46 %	160.94 %	372.4545
	0.0181	0.0151	0.31 %	1.07 %	(for first 5.5 sec)
SVSF	0.0095	0.0064	0 %	**0.587 %**	379.9891
	0.0181	0.0183	0.31 %	5.09 %	(for first 5.5 sec)
REKF	0.0219	**0.0122**	0.66 %	5.23 %	**5.4593**
RSVSF	**0.0095**	0.0126	**0** %	**1.01 %**	5.5360
MNIST test#1					
Adam	**0**	9.7879e-09	**0** %	40.27 %	209.6011
REKF	**0**	**1.5235e-09**	**0** %	11.97 %	203.8331
RSVSF	**0**	2.2093e-10	**0** %	**6.25 %**	**170.6640**
MNIST test#2					
Adam	5.7667e-04	0.0019	0.0112 %	14.68 %	223.3985
REKF	2.7333e-04	0.0013	0.0037 %	7**.01 %**	215.3128
RSVSF	**1.3000e-04**	**5.3115e-04**	**0.0028** %	8.45 %	**186.4490**
EMNIST test#3					
Adam	0.0394	0.1047	0.0537 %	**1.40 %**	93.5994
REKF	**0.0273**	0.0680	0.1448 %	7.39 %	115.2063
RSVSF	**0.0273**	**0.0658**	**0.1149** %	5.70 %	**80.4331**
Fashion-MNIST test#4 (DNN)					
Adam	0.0224	0.0593	0.0826 %	4.30 %	178.8158
REKF	5.4000e-04	0.0031	**0.0050** %	**3.17 %**	190.2946

RSVSF	**2.3333e-04**	**0.0014**	0.0059 %	8.25 %	**151.6335**
Fashion-MNIST test#5 (CNN)					
Adam	0.0050	0.0146	0.0600 %	9.40 %	308.9294
REKF	7.8500e-04	0.0037	0.0143 %	**9.30 %**	308.4723
RSVSF	**1.6000e-04**	**0.0011**	**0.0060** %	16.08 %	**300.7655**
CIFAR-10 test#6 (DNN)					
Adam	0.0885	0.2521	0.5023 %	5.18 %	261.2934
REKF	**2.1800e-04**	**0.0073**	**0.0020** %	1.25 %	275.4492
RSVSF	0.0119	0.0978	0.0257 %	**0.83 %**	**228.4755**
CIFAR-10 test#7 (CNN)					
Adam	0.0157	0.0459	0.1121 %	6.28 %	276.8277
REKF	**6.0000e-06**	**0.0023**	**9.66e-04** %	**3.23 %**	275.4476
RSVSF	1.0000e-05	0.0036	0.0014 %	5.65 %	**267.8367**
CIFAR-100 test#8 (DNN)					
Adam	0.3778	1.4325	34.71 %	77.97 %	269.2654
REKF	**7.7200e-04**	**0.0205**	**0.0070** %	**2.33 %**	288.5860
RSVSF	0.0249	0.1828	0.1080 %	1.99 %	**236.6184**
CIFAR-100 test#9 (CNN)					
Adam	0.1157	0.3740	1.50 %	11.55 %	4.1583e+03
REKF	**0.0621**	**0.1943**	**0.1568** %	**2.24 %**	4.1769e+03
RSVSF	0.0669	0.2093	0.3240 %	4.68 %	**4.0198e+03**

2.5 Conclusion

In this paper, two novel deep learning optimization methods were proposed, namely the Reduced Extended Kalman Filter (REKF) and the Reduced Smooth Variable Structure Filter (RSVSF). They are based on the estimation methods, namely the Extended Kalman Filter (EKF) and the Smooth Variable Structure Filter (SVSF). The proposed methods were

compared in nine experiments with Adam as a powerful first-order algorithm on five benchmark datasets using different network architectures. Test results demonstrated that the proposed methods are able to compete with the current first-order methods in optimization convergence speed and memory usage. These results show that the proposed methods have achieved positive impact without compromising on any performance metrics. The results indicate that REKF and RSVSF can be used competitively in Deep learning training. It should be noted the performance of the two proposed methods are overall comparable with the exception of processing time where RSVSF is consistently better in all tests.

Acknowledgement

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), funding reference number CRDPJ 486107-15.

References

[1] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012, pp. 3642-3649.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare*, et al.*, "Human-level control through deep reinforcement learning," *Nature,* vol. 518, pp. 529-533, 2015.

[3] A. G. Ivakhnenko, "Polynomial theory of complex systems," *IEEE transactions on Systems, Man, and Cybernetics,* vol. 1, pp. 364-378, 1971.

[4] A. G. Ivakhnenko, "The group method of data handling-a rival of the method of stochastic approximation," *Soviet Automatic Control,* vol. 13, pp. 43-55, 1968.

[5] A. G. Ivakhnenko and V. G. e. Lapa, "Cybernetics and forecasting techniques," 1967.

[6] A. G. e. Ivakhnenko and V. G. Lapa, "Cybernetic predicting devices," DTIC Document1966.

[7] A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," in *Advances in neural information processing systems*, 2009, pp. 545-552.

[8] J. Schmidhuber, D. Cireşan, U. Meier, J. Masci, and A. Graves, "On fast deep nets for AGI vision," in *International Conference on Artificial General Intelligence*, 2011, pp. 243-246.

[9] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks,* vol. 61, pp. 85-117, 2015.

[10] J. Ba and R. Caruana, "Do deep nets really need to be deep?," in *Advances in neural information processing systems*, 2014, pp. 2654-2662.

[11] P.-S. Huang, H. Avron, T. N. Sainath, V. Sindhwani, and B. Ramabhadran, "Kernel methods match deep neural networks on timit," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, 2014, pp. 205-209.

[12] Y. Bengio, "Learning deep architectures for AI," *Foundations and trends® in Machine Learning,* vol. 2, pp. 1-127, 2009.

[13] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," in *Tenth International Workshop on Frontiers in Handwriting Recognition*, 2006.

[14] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural computation,* vol. 22, pp. 3207-3220, 2010.

[15] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980,* 2014.

[16] T. Tieleman and G. Hinton, "Rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning," Technical report, 2012. 31.

[17] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701,* 2012.

[18] H. W. Sorenson, *Kalman filtering: theory and application*: IEEE, 1985.

[19] A. Alessandri, G. Cirimele, M. Cuneo, S. Pagnan, and M. Sanguineti, "EKF learning for feedforward neural networks," in *European Control Conference (ECC), 2003*, 2003, pp. 1990-1995.

[20] S. S. Haykin, *Kalman filtering and neural networks*: Wiley Online Library, 2001.

[21] S. Habibi, "The Smooth Variable Structure Filter," *Proceedings of the IEEE,* vol. 95, pp. 1026-1059, 2007.

[22] R. Ahmed, "Training of Neural Networks Using the Smooth Variable Structure Filter with Application to Fault Detection," Department of Mechanical Engineering, McMaster University, April 2011.

[23] R. Hecht-Nielsen, "Theory of the backpropagation neural network," *Neural Networks,* vol. 1, pp. 445-448, 1988.

[24] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747,* 2016.

[25] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Advances in neural information processing systems*, 2014, pp. 2933-2941.

[26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," DTIC Document1985.

[27] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks,* vol. 12, pp. 145-151, 1999.

[28] Y. Nesterov, "A method of solving a convex programming problem with convergence rate O (1/k2)," in *Soviet Mathematics Doklady*, 1983, pp. 372-376.

[29] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, 2013, pp. 1139-1147.

[30] A. Botev, G. Lever, and D. Barber, "Nesterov's Accelerated Gradient and Momentum as approximations to Regularised Update Descent," *arXiv preprint arXiv:1607.01981,* 2016.

[31] L. Deng, "The MNIST database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine,* vol. 29, pp. 141-142, 2012.

[32] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research,* vol. 12, pp. 2121-2159, 2011.

[33] M. J. D. Powell, "Restart procedures for the conjugate gradient method," *Mathematical programming,* vol. 12, pp. 241-254, 1977.

[34] E. Beale, "A derivation of conjugate gradients," *Numerical methods for nonlinear optimization,* pp. 39-43, 1972.

[35] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural networks,* vol. 6, pp. 525-533, 1993.

[36] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *The computer journal,* vol. 7, pp. 149-154, 1964.

[37] H. B. Demuth, M. H. Beale, O. De Jess, and M. T. Hagan, *Neural network design*: Martin Hagan, 2014.

[38] E. Polak and G. Ribière, "Note sur la convergence de directions conjugées," *Rev. Fr. Inform. Rech. Oper. v16,* pp. 35-43.

[39] Y.-H. Dai and Y. Yuan, "A nonlinear conjugate gradient method with a strong global convergence property," *SIAM Journal on optimization,* vol. 10, pp. 177-182, 1999.

[40] L. Scales, *Introduction to non-linear optimization*: Springer-Verlag New York, Inc., 1985.

[41] R. Battiti, "First-and second-order methods for learning: between steepest descent and Newton's method," *Neural computation,* vol. 4, pp. 141-166, 1992.

[42] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q. V. Le, and A. Y. Ng, "On optimization methods for deep learning," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 265-272.

[43] J. E. Dennis Jr and R. B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations*: SIAM, 1996.

[44] S. M. A. Burney, T. A. Jilani, and C. Ardil, "A Comparison of First and Second Order Training Algorithms for Artificial Neural Networks," in *International Conference on Computational Intelligence*, 2004, pp. 12-18.

[45] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the society for Industrial and Applied Mathematics,* vol. 11, pp. 431-441, 1963.

[46] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE transactions on Neural Networks,* vol. 5, pp. 989-993, 1994.

[47] S. Becker and Y. Le Cun, "Improving the convergence of back-propagation learning with second order methods," in *Proceedings of the 1988 connectionist models summer school*, 1988, pp. 29-37.

[48] J. Martens, "Deep learning via Hessian-free optimization," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 735-742.

[49] J. Nocedal and S. Wright, "Springer series in operations research and financial engineering," *Numerical Optimization,* 1999.

[50] S.-I. Amari, "Natural gradient works efficiently in learning," *Neural computation,* vol. 10, pp. 251-276, 1998.

[51] D. Povey, X. Zhang, and S. Khudanpur, "Parallel training of DNNs with natural gradient and parameter averaging," *arXiv preprint arXiv:1410.7455,* 2014.

[52] N. L. Roux, P.-A. Manzagol, and Y. Bengio, "Topmoumoute online natural gradient algorithm," in *Advances in neural information processing systems*, 2008, pp. 849-856.

[53] J. Martens and R. Grosse, "Optimizing neural networks with kronecker-factored approximate curvature," in *International conference on machine learning*, 2015, pp. 2408-2417.

[54] M. R. Bastian, J. H. Gunther, and T. K. Moon, "A simplified natural gradient learning algorithm," *Advances in Artificial Neural Systems,* vol. 2011, p. 3, 2011.

[55] R. Collobert, "Large Scale Machine Learning," 2004.

[56] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering,* vol. 82, pp. 35-45, 1960.

[57] R. E. Kalman and R. S. Bucy, "New results in linear filtering and prediction theory," *Journal of basic engineering,* vol. 83, pp. 95-108, 1961.

[58] S. Singhal and L. Wu, "Training Multilayer Perceptrons with the Extende Kalman Algorithm," in *NIPS*, 1988, pp. 133-140.

[59] G. V. Puskorius and L. A. Feldkamp, "Decoupled extended Kalman filter training of feedforward layered networks," in *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, 1991, pp. 771-777.

[60] P. Trebatický, "Recurrent neural network training with the extended kalman filter," in *IIT. SRC 2005: Student Research Conference*, 2005, p. 57.

[61] R. J. Williams, "Training recurrent networks using the extended Kalman filter," in *Neural Networks, 1992. IJCNN., International Joint Conference on*, 1992, pp. 241-246.

[62] V. Patel, "Kalman-Based Stochastic Gradient Method with Stop Condition and Insensitivity to Conditioning," *SIAM Journal on Optimization,* vol. 26, pp. 2620-2648, 2016.

[63] S. Habibi and R. Burton, "The variable structure filter," in *ASME 2002 International Mechanical Engineering Congress and Exposition*, 2002, pp. 157-165.

[64] S. A. Gadsden, M. El Sayed, and S. R. Habibi, "Derivation of an optimal boundary layer width for the smooth variable structure filter," in *American Control Conference (ACC), 2011*, 2011, pp. 4922-4927.

[65] S. A. Gadsden and S. R. Habibi, "A new form of the smooth variable structure filter with a covariance derivation," in *Decision and Control (CDC), 2010 49th IEEE Conference on*, 2010, pp. 7389-7394.

[66] S. A. Gadsden, "Smooth variable structure filtering: theory and applications," 2011.

[67] F. Outamazirt, L. Fu, Y. Lin, and N. Abdelkrim, "A new SINS/GPS sensor fusion scheme for UAV localization problem using nonlinear SVSF with covariance derivation and an adaptive boundary layer," *Chinese Journal of Aeronautics,* vol. 29, pp. 424-440, 2016.

[68] M. S. Farag, R. Ahmed, S. Gadsden, S. Habibi, and J. Tjong, "A comparative study of Li-ion battery models and nonlinear dual estimation strategies," in *Transportation Electrification Conference and Expo (ITEC), 2012 IEEE*, 2012, pp. 1-8.

[69] T. Kim, Y. Wang, H. Fang, Z. Sahinoglu, T. Wada, S. Hara*, et al.*, "Model-based condition monitoring for lithium-ion batteries," *Journal of Power Sources,* vol. 295, pp. 16-27, 2015.

[70] Å. Björck, *Numerical methods in matrix computations*: Springer, 2015.

[71] K. E. Fitzgerald, "Comparison of some FORTRAN programs for matrix inversion," *Journal of Research,* vol. 78, pp. 15-33, 1974.

[72] J. J. D. CROZ and N. J. Higham, "Stability of methods for matrix inversion," *IMA Journal of Numerical Analysis,* vol. 12, pp. 1-19, 1992.

[73] J. Dongarra, M. Faverge, H. Ltaief, and P. Luszczek, "High performance matrix inversion based on LU factorization for multicore architectures," in *Proceedings of the 2011 ACM international workshop on Many task computing on grids and supercomputers*, 2011, pp. 33-42.

[74] K. O'Neil and Y.-C. Chen, "Instability of pseudoinverse acceleration control of redundant mechanisms," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, 2000, pp. 2575-2582.

[75] E. Pohl, "Enumeration of singular configurations for robotic manipulators," 1991.

[76] K. L. Doty, C. Melchiorri, and C. Bonivento, "A theory of generalized inverses applied to robotics," *The International Journal of Robotics Research,* vol. 12, pp. 1-19, 1993.

[77] G. V. Puskorius and L. A. Feldkamp, "Decoupled extended Kalman filter training of feedforward layered networks," in *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, 1991, pp. 771-777 vol.1.

[78] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747,* 2017.

[79] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: an extension of MNIST to handwritten letters," *arXiv preprint arXiv:1702.05373,* 2017.

[80] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 and cifar-100 datasets," *URl: https://www. cs. toronto. edu/kriz/cifar. html,* vol. 6, 2009.

[81] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580,* 2012.

[82] L. Xu, J. Wang, and Q. Chen, "Kalman filtering state of charge estimation for battery management system based on a stochastic fuzzy neural network battery model," *Energy Conversion and Management,* vol. 53, pp. 33-39, 2012.

[83] M. Ismail, M. Attari, S. Habibi, and S. Ziada, "Estimation theory and Neural Networks revisited: REKF and RSVSF as optimization techniques for Deep-Learning," *Neural Networks,* 2018.

Appendix A Abbreviations and Nomenclature

Table 2-XV Abbreviations used in the paper

Abbreviation	Description
DL	Deep Learning
NN	Neural Network
DNN	Deep Neural Network
EKF	Extended Kalman Filter
SVSF	Smooth Variable Structure Filter
REKF	Reduced Extended Kalman Filter
RSVSF	Reduced Smooth Variable Structure Filter
MSE	Mean Squared Error
BP	Back Propagation
GD	Gradient Descent
GPU	Graphical Processing Units
RNN	Recurrent Neural Networks
FNN	Feedforward Neural Network
LSTM	Long Short Term Memory
ML	Machine Learning
NAG	Nestrove's Acceleration Gradient
CGD	Conjugate Gradient Descent
BFGS	Broyden–Fletcher–Goldfarb–Shanno

SGD	Stochastic Gradient Descent
kSGD	Kalman Based Stochastic Gradient Descent
CNN	Convolutional Neural Network
lr	Learning Rate

Table 2-XVI Nomenclature

symbol	Description
$w_{i,j}^{n_l}$	The weight connecting node (n, j) to node $(n+1, i)$, n is the layer number. Biases is modeled as a weight for input of one
n_l	The layer number
n	The size of the network parameters
$H_{Hessian}$	The Hessian matrix
J	The Jacobian matrix
N_{n_l}	Number of nodes in layer n
N_{Layers}	Number of nodes in the output layer
Layers	Total number of layers
i, j	Node numbers
x	Network input
φ	Activation function
y_i, y	Network output at node (i), Network output
η	Learning rate
q	Sample number

B	Batch size
v	Velocity term that represents momentum
ϵ	Small value for numerical stability
O	Order of complexity
θ	A vector of the network parameters
s_k	A vector of the system states
$f(s_{k-1}, u_{k-1})$	A non-linear function of the states and the inputs
u_{k-1}	Non-linear system input
z_k	System output
$h(s_k)$	The system non-linear output function
^	Estimate
Q_{k-1}	The covariance of the process noise
F_{k-1}	The Jacobian of f
w_{k-1}^{noise}	Process noise
v_k^{noise}	Measurement noise
K_k	Gain
R_{k-1}	The covariance of the measurement noise
$P_{k\|k}$	Error covariance
k	Variable at time k
ψ	The smoothing boundary layer width

γ	The a posteriori error tuning factor
lr	Learning rate
MB	Total number of network outputs ($O \times B$)
O	Number of neurons in output layer
B	Batch size
ρ	Damping parameter
I	Identity matrix
^{t}	The transpose of a matrix
S_k	Innovation covariance
$\tilde{y}_k$	Error
$\lvert \ \rvert_{abs}$	The element-wise absolute value of the term

Appendix B Proof of RSVSF stability

In this appendix the proof of stability in relation to the addition of (η) in Equation (2-62) compared with Equation (2-61) is provided. SVSF has two important error variables that are defined in Equations (2-50) and (2-54), the a priori and the a posteriori error estimates. In [21] a Lyapunov function $\nu_k = \tilde{y}_{k|k}^2$ is defined. The weights update process is stable if $\left(\Delta\nu_k = \tilde{y}_{k|k}^2 - \tilde{y}_{k-1|k-1}^2\right) < 0$. This equation can be redefined as,

$$\left|\tilde{y}_{k|k}\right|_{abs} < \left|\tilde{y}_{k-1|k-1}\right|_{abs} \tag{B.64}$$

According to Theorem 1 in [21], the SVSF correction action K_k that would satisfy the stability condition in Equation (B.64) is subject to the following conditions:

$$\left|\tilde{y}_{k|k-1}\right|_{abs} \leq |H_k K_k|_{abs} < \left|\tilde{y}_{k|k-1}\right|_{abs} + \left|\tilde{y}_{k-1|k-1}\right|_{abs} \tag{B.65}$$

$$sign\,(H_k K_k) = sign(\tilde{y}_{k|k-1}) \tag{B.66}$$

Equation (2-51) satisfied this condition in [21], and proved to be stable and convergent. RSVSF proposes using a new gain defined in Equation (B.67),

$$K_k = H_k^+ \left(\eta\left|\tilde{y}_{k|k-1}\right|_{abs} + \gamma\left|\tilde{y}_{k-1|k-1}\right|_{abs}\right) \circ sign\left(\frac{\tilde{y}_{k|k-1}}{\psi}\right) \tag{B.67}$$

To prove its stability and convergence, first the a priori error estimate is considered (from Equation (2-50)),

$$\tilde{y}_{k|k-1} = z_k - \hat{z}_{k|k-1} \tag{B.68}$$

Assuming that the relationship (h) between the measurement signals and the states is known and can be linearized around the state estimate at time (k), then $\hat{z}_{k|k-1}$ can be defined as,

$$\hat{z}_{k|k-1} = H_k \hat{s}_{k|k-1} \tag{B.69}$$

Substituting Equation (B.68) in Equation (B.67), the latter can be restated as,

$$\tilde{y}_{k|k-1} = z_k - H_k \hat{s}_{k|k-1} \tag{B.70}$$

But using Equation (2-52) or $\hat{s}_{k|k} = \hat{s}_{k|k-1} + K_k$ results in,

$$\tilde{y}_{k|k-1} = z_k - H_k(\hat{s}_{k|k} - K_k) = z_k - H_k \hat{s}_{k|k} + H_k K_k \tag{B.71}$$

However $z_k - H_k \hat{s}_{k|k}$ is the definition of the a posteriori error estimation (Equation (2-54) and

$\hat{z}_{k|k} = H_k \hat{s}_{k|k}$), therefore,

$$\tilde{y}_{k|k-1} = \tilde{y}_{k|k} + H_k K_k \tag{B.72}$$

To study the stability of the RSVSF gain defined in Equation (B.67), the gain is substituted in Equation (B.72),

$$\tilde{y}_{k|k-1} = \tilde{y}_{k|k} + H_k H_k^{+} \left(\eta \left| \tilde{y}_{k|k-1} \right|_{abs} + \gamma \left| \tilde{y}_{k-1|k-1} \right|_{abs} \right) \circ sign\left(\frac{\tilde{y}_{k|k-1}}{\psi} \right) \tag{B.73}$$

Which is rearranged in,

$$\begin{aligned} \tilde{y}_{k|k-1} = {} & \tilde{y}_{k|k} + \eta \left| \tilde{y}_{k|k-1} \right|_{abs} \circ sign\left(\frac{\tilde{y}_{k|k-1}}{\psi} \right) + \gamma \left| \tilde{y}_{k-1|k-1} \right|_{abs} \\ & \circ sign\left(\frac{\tilde{y}_{k|k-1}}{\psi} \right) \\ = {} & \tilde{y}_{k|k} + \eta \tilde{y}_{k|k-1} + \gamma \left| \tilde{y}_{k-1|k-1} \right|_{abs} \circ sign\left(\frac{\tilde{y}_{k|k-1}}{\psi} \right) \end{aligned} \tag{B.74}$$

Which can be reorganized as,

$$\begin{aligned}\tilde{y}_{k|k} &= \tilde{y}_{k|k-1} - \eta\tilde{y}_{k|k-1} - \gamma\left|\tilde{y}_{k-1|k-1}\right|_{abs} \circ sign\left(\frac{\tilde{y}_{k|k-1}}{\psi}\right) \\ &= (1-\eta)\tilde{y}_{k|k-1} - \gamma\left|\tilde{y}_{k-1|k-1}\right|_{abs} \circ sign\left(\frac{\tilde{y}_{k|k-1}}{\psi}\right) \\ &= \left((1-\eta)\left|\tilde{y}_{k|k-1}\right|_{abs} - \gamma\left|\tilde{y}_{k-1|k-1}\right|_{abs}\right) \circ sign\left(\frac{\tilde{y}_{k|k-1}}{\psi}\right)\end{aligned} \quad \text{(B.75)}$$

Taking the absolute value of both sides of the equation,

$$\left|\tilde{y}_{k|k}\right|_{abs} = \left|\gamma\left|\tilde{y}_{k-1|k-1}\right|_{abs} - (1-\eta)\left|\tilde{y}_{k|k-1}\right|_{abs}\right|_{abs} \quad \text{(B.76)}$$

In the case of a full batch mode where the same full set of training data is used in every epoch the a priori error is found by the following equations,

$$Loss_{k|\theta_{k-1}} = MSE_{k|\theta_{k-1}}\left(target_k, network\ output_{\theta_{k-1}}\right) \quad \text{(B.77)}$$

$$\tilde{y}_{k|k-1} = 0 - Loss_{k|\theta_{k-1}} \quad \text{(B.78)}$$

However, the a posteriori error of the previous epoch $(k-1)$ is defined as,

$$Loss_{k-1|\theta_{k-1}} = MSE_{k-1|\theta_{k-1}}\left(target_{k-1}, network\ output_{\theta_{k-1}}\right) \quad \text{(B.79)}$$

$$\tilde{y}_{k-1|k-1} = 0 - Loss_{k-1|\theta_{k-1}} \quad \text{(B.80)}$$

Comparing Equations (B.78) and (B.80), the differences between the a posteriori and a priori error estimates are simply in the target values. In full batch mode, $\text{target}_k = \text{target}_{k-1}$ which results in,

$$\left|\tilde{y}_{k|k-1}\right|_{abs} = \left|\tilde{y}_{k-1|k-1}\right|_{abs} \quad \text{(B.81)}$$

Therefore, Equation (B.76) can be modified to the following form,

$$\left|\tilde{y}_{k|k}\right|_{abs} = \left|(\eta+\gamma-1)\left|\tilde{y}_{k-1|k-1}\right|_{abs}\right|_{abs} = (\eta+\gamma-1)\left|\tilde{y}_{k-1|k-1}\right|_{abs} \quad \text{(B.82)}$$

Therefore the rate of convergence of the filter is determined by the rate of $(\eta + \gamma - 1)$. To insure the stability condition in Equation (B.64), the choice of the parameters η and γ should satisfy the following inequality,

$$0 \leq (\eta + \gamma - 1) < 1 \tag{B.83}$$

Which can be represented as,

$$1 \leq (\eta + \gamma) < 2 \tag{B.84}$$

A corner case of Equation (B.84) is by setting $\gamma = 0$ to cancel the extra forward calculation to calculate the a posteriori error $\tilde{y}_{k-1|k-1}$ in (B.67). For this corner case, Equation (B.84) can be rewritten as,

$$1 \leq (\eta) < 2 \tag{B.85}$$

which is the definition of (η). Equation (B.85) satisfies the stability and the convergence of the system.

Equation (2-62) replaces the sign function with a sat function to reduce the chattering. This is a similar approach to the original SVSF in [21]. The elimination of a posteriori error calculation significantly reduces the total training time.

Appendix C the used software and hardware in the experiments

The experiments uses *Keras* platform, which is a library that uses *Theano*, and works on *Python*. The versions of the used libraries are as following:

- *Keras*: 2.0.3
- *Theano*: 0.9.0.dev-c697eeab84e5b8a74908da654b66ec9eca4f1291
- *Python*: 2.7.13
- *NumPy*: 1.12.1

The used hardware is as follows:

- *CPU*: Intel(R) Xeon(R) CPU ES-2687W 0 @ 3.106Hz
- *GPU*: Nvidia GeForce GTX 780
- *Ram*: 32.0 GB

This paper is published in Neural Networks journal, Elsevier. Volume 108, 2018, Pages 509-526, ISSN 0893-6080. https://doi.org/10.1016/j.neunet.2018.09.012. This paper is republished here with permission.

Chapter 3: A Study on REKF and RSVSF in Deep Learning: Sensitivity Analysis and Multi-Filter Implementation

Mahmoud Ismail[a,*], and Saeid Habibi[a]

[a] Department of Mechanical Engineering, McMaster University, Hamilton, Canada;

[*]**Corresponding author**: Mahmoud Ismail, Department of Mechanical Engineering, McMaster University, Hamilton, Canada.

This journal paper is submitted to IEEE Transactions on Neural Networks and Learning Systems in October 2018.

Abstract

In the recent years, Deep-Learning has become the leading artificial intelligence algorithm in many fields thanks to its unprecedented performance which even in a few classification and control problems, superseded human performance. However, the training process of Deep-Learning is usually slow and requires high-performance hardware and very large datasets. The optimization of the training methods can improve learning rates of the Deep-Learning networks and achieve higher performance in the same time span. This paper examines the sensitivity and the need for fine tuning of parameters for two newly developed training algorithms, namely the Reduced Extended Kalman Filter (REKF), and the Reduced Smooth Variable Structure Filter (RSVSF). Extensions for these two algorithms are developed, namely the REKF-MultiFilter (REKF_MF) and the RSVSF-MultiFilter

(RSVSF_MF). These extensions were tested on MNIST dataset to compare their performance with their original forms.

Keywords

Deep Learning, Neural Networks, Machine Learning, Optimization algorithms, Training algorithms.

3.1 Introduction

Deep Learning (DL) use in academic and industrial applications increase with a fast rate since DL has proven to achieve unprecedented results compared to other machine learning techniques [1, 2]. The most famous two applications for DL are image and speech recognition mostly because of the application nature of being classic problems in computer sciences field to be used as a benchmark test for machine learning algorithms. DL has also been applied to many other fields such as big data analytics [3], finance [4], language modeling, natural language processing, information retrieval [5], and classification of skin cancer [6]. A corner stone of using DL in different application is the ability of the DL to extract meaningful features from the data. In [7], the authors developed two training algorithms based on estimation theory for DL training, namely, the Reduced Extended Kalman Filter (REKF), and the Reduced Smooth Variable Structure Filter (RSVSF). These algorithms were shown to perform relatively well in terms of learning rate and memory usage. However these advantages that put REKF, and RSVSF ahead of the other training algorithms, there was uncertainty about their performance sensitivity to their configurable parameters. The problem of having tunable parameters is that they affect the DL

performance after training, but they are required to be defined before the start of the training. This is a major challenge in DL training algorithms. Many algorithms try to deploy adaptable training parameters to reduce the need of the fine tuning process, however usually they end in defining higher level hyper parameters that can be also fine-tuned. The need of training algorithms that are insensitive to the tuning parameters and are able to achieve high performance with generic parameters values is very important. The availability of such algorithms will enable commercial and scientific users of DL to reduce the training time as several training trials to achieve acceptable fine-tuned training parameters will not be required. This paper investigates the sensitivity of the generalization performance against the tuning parameters for REKF and RSVSF.

3.2 Literature review

In Deep-Learning, there are different training optimization methods that minimize the network error on a given training dataset. A major group of these methods are gradient-based methods. Gradient provides guided optimization, direction, and magnitude wise. It also is relatively fast, and can be calculated in parallel. However, there are other methods that are not based on gradient; instead, they use Random guess, Genetic algorithm, or Rprop algorithm. Another family of optimization methods utilize the estimation theory as shown in [7]. In this section, first a review of the architecture of a simple feedforward DL network is presented, followed by a brief literature review on training algorithms with a focus on REKF and RSVSF.

3.2.1 Deep Neural Networks

One of the simplest DL networks is the Deep Neural Networks (DNN's). DNN's are simply Feedforward Neural Networks (FNN) with more than one hidden layer. In DNN's the layers are fully connected. Meaning all the nodes in a given layer are connected to all nodes in the next layer. For simplicity a one hidden layer FNN is shown in Figure 3-1. The output of each node is described in Equation (3-1), where φ is the activation function, $w_{i,j}^{n}$ is the weight connecting $node\ (n,j)$ to $node\ (n+1,i)$, n is the layer number, and b_i^{n+1} is the bias (offset) for $node\ (n+1,i)$. In neural networks there are several types of activation functions –linear and nonlinear -. As an example, the most common used function is the sigmoid function which is described in Equation (3-2). Feedforward networks take their name from how the output is calculated from the input. The input is first connected to the input layer and then each node in the subsequent layers uses Equation (3-1) to calculate its output. To evaluate the performance of the neural network, a loss/cost function is used. One of the simplest loss functions is the Mean Square Error (MSE), which is described in Equation (3-3).

$$Node_i^{n+1} = x_i^{n+1} = \varphi\left(\sum_{j=1}^{N_n} w_{i,j}^{n}\, x_j^{n} + b_i^{n+1}\right) \tag{3-1}$$

$$\varphi(z) = \frac{1}{1+e^{-z}} \tag{3-2}$$

$$Error\ (MSE) = \frac{1}{n}\sum_{i=1}^{n}(network\ output - target)^2 \tag{3-3}$$

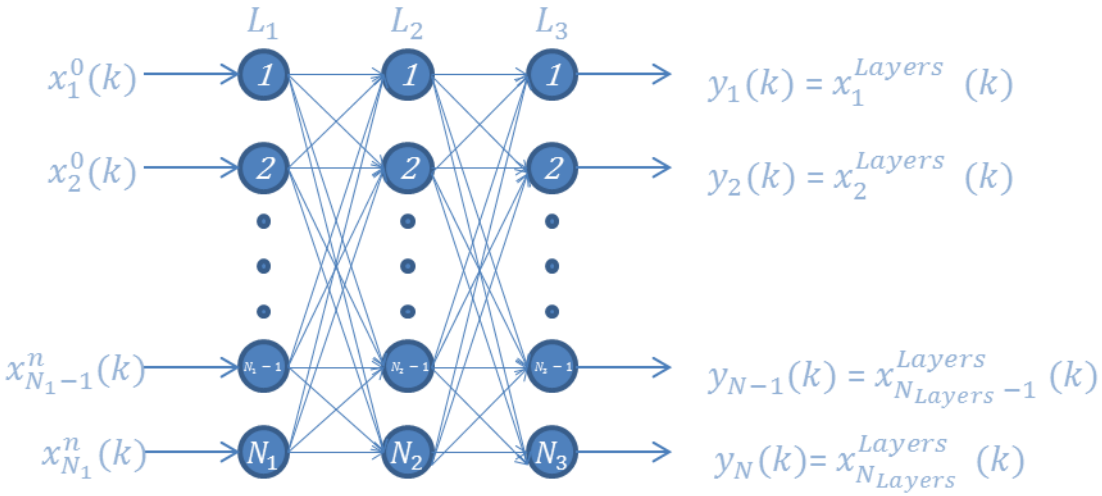


Figure 3-1 One hidden layer Feedfoward Neural Network

3.2.2 DL training algorithms

There are many DL training algorithms, a survey is presented in [7, 8]. One way of categorizing them is to break them into families. In Figure 3-2 the three main families are shown. The most common family is the gradient-based family. The reason that they are common is that they are relatively fast and simple compared to others. However the recent advances in the estimation based family, specially REKF and RSVSF in [7] outperform the state-of-the-art of the gradient based family in all performance measures including success rates and training time. Following is an overview on REKF and RSVS.

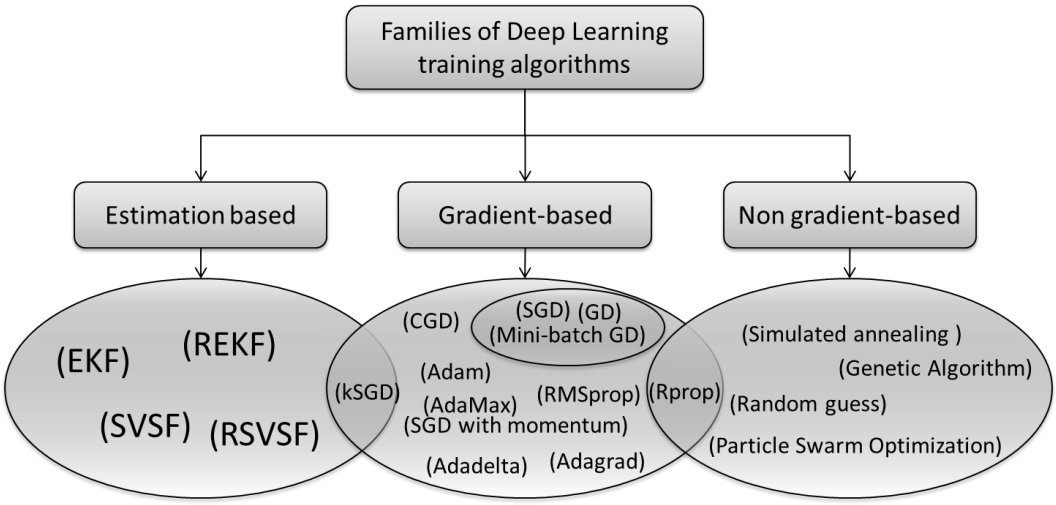


Figure 3-2 different training algorithms families [7-13]

3.2.2.1 REKF

Reduced-EKF (REKF) is the a simplified version of EKF that is defined in [7] to enable an estimation-based method to compete with the state-of-the-art methods represented by Adam method [14] due to its simplicity and high performance. In [7] it was shown that REKF outperformed Adam with using less memory and converged faster and also achieved better generalization performance on MNIST dataset.

The simplifications that REKF used compared with EKF can be summarized in:

1- Applying Mini-batch on EKF

2- Using a scalar output for the EKF non-linear system

3- Using Fully-Decoupled-EKF version of EKF (FDEKF) [15]

4- Using shared scalar values for tuning parameters such as (Q_{k-1}, R_{k-1}).

These modifications significantly reduced the memory and computational requirements for EKF and allowed it to outperform its original form when the training time is taken into consideration instead of epochs. These improvements become more significant with larger size networks, where the original form is not feasible with the current available hardware. The steps of the algorithm are explained in Table 3-I.

Table 3-I REKF algorithm

Algorithm 1: *REKF,* this algorithm reduces EKF to a more memory and calculation efficient algorithm to be used for optimizing DL networks

Require: $\boldsymbol{q, r\ values}$
Require: $\boldsymbol{p}$ initial values
$\boldsymbol{P} \leftarrow \begin{bmatrix} \boldsymbol{p} \\ \vdots \\ \boldsymbol{p} \end{bmatrix}$ initialize $\boldsymbol{P}$ matrix as a vector filled with $\boldsymbol{p}$ initial value.
While $\boldsymbol{k < number\ of\ epochs}$
$\boldsymbol{k \leftarrow k + 1}$
 # calculate the network loss
$\boldsymbol{loss = MSE(target, network\ output)}$
 # the system target is to achieve zero loss, hence the system error is
$\boldsymbol{\tilde{y}_k = 0 - loss = (system\ target - system\ output)}$
$\boldsymbol{g_k = \nabla_\theta loss(\theta_{k-1})}$
$\boldsymbol{P_{k|k-1} = P_{k-1|k-1} \oplus q}$
#perform the update step
$\boldsymbol{s_k = g_k P_{k|k-1} g_k^T \oplus r}$
$\boldsymbol{K_k = P_{k|k-1} g_k^T \oslash s_k}$
$\boldsymbol{\theta_k = \theta_{k-1} + K_k \tilde{y}_k}$
$\boldsymbol{P_{k|k} = (I - K_k g_k) P_{k|k-1}}$
End while
Retun $\boldsymbol{\theta_k}$

3.2.2.2 RSVSF

Similar to REKF, Reduced-SVSF (RSVSF) is the a simplified version of SVSF that is defined in [7] to enable the original SVSF to compete REKF and Adam. In [7] it was shown

that RSVSF outperformed Adam and competed with REKF while using less memory than both of them and converged relatively fast and also achieved better generalization performance on MNIST benchmark problem.

The simplifications that RSVSF used are very similar to REKF and can be summarized in:

1- Applying Mini-batch on SVSF

2- Using a scalar output for the SVSF non-linear system

3- Using shared scalar values for tuning parameters such as (γ, ψ).

4- Introducing a new parameter as a learning rate (η) that can substitute the effect of (γ) without requiring the extra forward calculation which results in around one third reduction in calculation time while sustaining the same performance.

These modifications significantly reduced the memory and computational requirements for SVSF and allowed it to outperform its original form when the training time is taken into consideration instead of epochs. Similar to REKF, These RSVSF improvements become more significant with larger size networks, where the original form is not feasible using the current available hardware. RSVSF steps of the algorithm are explained in Table 3-II.

Table 3-II RSVSF algorithm

Algorithm 2: *RSVSF*, this algorithm reduces SVSF to a more memory and calculation efficient algorithm to be used for optimizing DL networks
Require: $\boldsymbol{\gamma, \psi, \eta\ values}$ **While** $\boldsymbol{k < number\ of\ epochs}$ $\boldsymbol{k \leftarrow k + 1}$ *# calculate the network loss* $\boldsymbol{loss = MSE(target, network\ output)}$ *# the system target is to achieve zero loss, hence the system error is* $\boldsymbol{\tilde{y}_k = 0 - loss = (system\ target - system\ output)}$ $\boldsymbol{g_k = \nabla_\theta loss(\theta_{k-1})}$

$$H_k^+ = \frac{g_k^t}{\sum_{i=1}^{n}\left[\left(g_k(i)\right)^2\right]}$$

$$K_k = H_k^+\left(\eta\left|\tilde{y}_{k|k-1}\right| + \gamma\left|\tilde{y}_{k-1|k-1}\right|\right) \circ sat\left(\frac{\tilde{y}_{k|k-1}}{\psi}\right)$$

#perform the update step

$$\theta_k = \theta_{k-1} + K_k$$

End while

Retun θ_k

3.3 Paper Contributions

In this paper the following is presented:

1- A sensitivity test on REKF and RSVSF that shows their robustness against the values of their configuration parameters (section 3.4)

2- The introduction of the Multi-Filter versions of REKF and RSVSF (section 3.5)

3- A comparison study between the REKF and RSVSF and their Multi-Filter versions, REKF_MF and RSVSF_MF (section 3.6)

3.4 Sensitivity analysis of REKF and RSVSF

In this section the results of the sensitivity analysis of REKF and RSVSF is presented. The sensitivity analysis used the hand-written digits database MNIST as a benchmark problem for performance evaluation.

3.4.1 Sensitivity analysis design

The used architecture in this test is a vanilla DNN that used two hidden layers with the size of 512 with Rectified Linear Units (ReLU's) as the activation function. The output layer used a softmax activation function. The architecture of the network is shown in Figure 3-3.

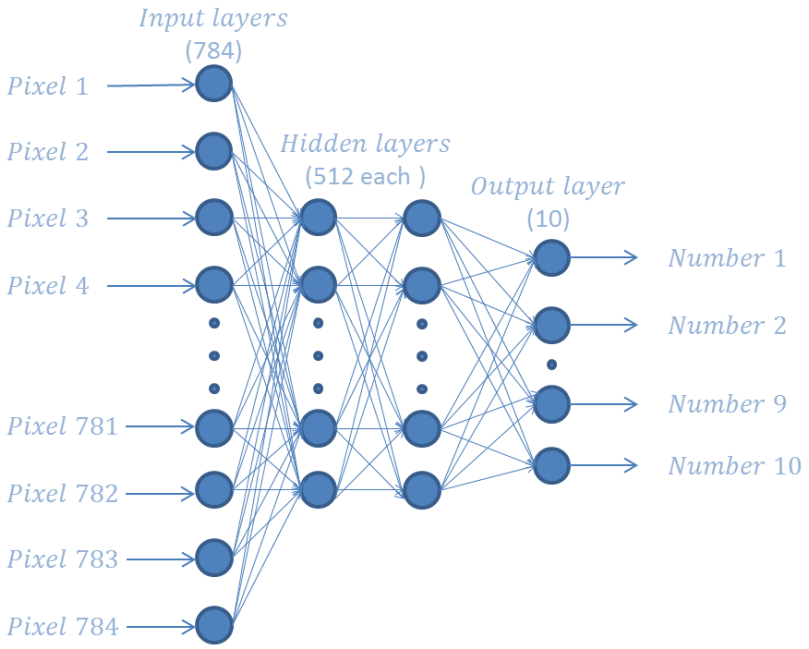


Figure 3-3 The used DNN for the sensitivity test

To test a large space of configuration parameters, each test is using a unique combination of the configuration parameters. Adam algorithm [14] was used also to compare its result with all the REKF and RSVSF combinations. All the combinations of the configuration parameters for all the training algorithms are summarized in Table 3-III.

Table 3-III Configuration parameters for all the optimizers in the sensitivity analysis

Parameter name	Starting value	Ending value	Step size
REKF			
q	1E-4	1E5	100
r	1E-7	1E5	100
p	1E-4	1E2	100
		Total Number of combinations:	216

RSVSF			
ψ	1E-7	1E2	100
γ	0	0.99	0.2
η	1	1.99	0.2
		Total Number of combinations:	102
Adam			
used the default values in [14] ($\eta = 0.001,\ \beta_1 = 0.9,\ \beta_2 = 0.999,\ and\ \epsilon = 10^{-8}$)			

It is worth noting that the total number of combinations in RSVSF is 102 compared with 216 in the REKF case. The reason for that is the definition of (η) and (γ) in [7, 16], where $\gamma + \eta < 2$. So the tested combinations are the ones that satisfy this RSVSF stability condition.

3.4.2 REKF and RSVSF sensitivity analysis results

In [7], it was shown that REKF and RSVSF outperformed Adam as a representative training algorithm of the state-of the-art adaptable training algorithms while using less memory. This result is confirmed here. Table 3-IV shows that both REKF and RSVSF outperformed Adam training algorithm. It is important to highlight that Adam simulation was run 10 times with random initialization and the best run is selected for the comparison. RSVSF recorded 20.77% less error rate than Adam which is a significant error reduction.

Table 3-IV Best validation success rate for Adam, REKF, and RSVSF

Optimizer name	Adam	REKF	RSVSF

Best validation success rate	98.17%	98.48%	98.55%

In every training epoch, the validation performance is recorded for the three optimizers as a performance measure. Two key performance measures are investigated in this paper:

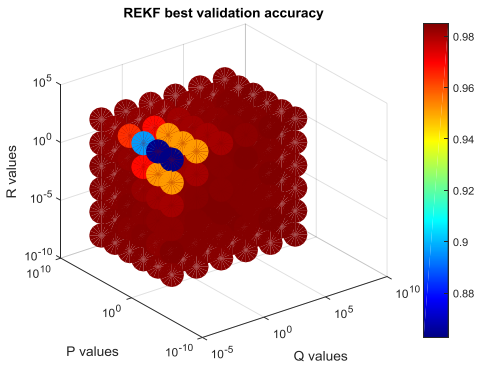
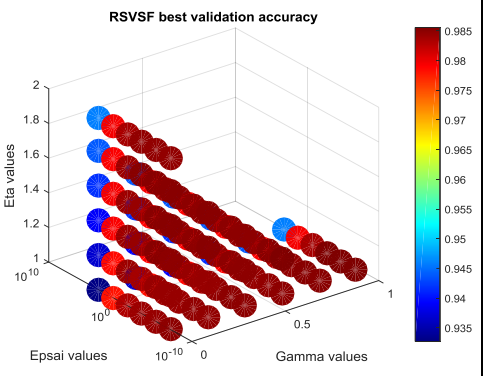
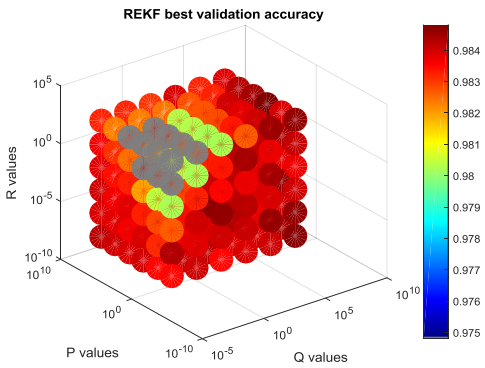
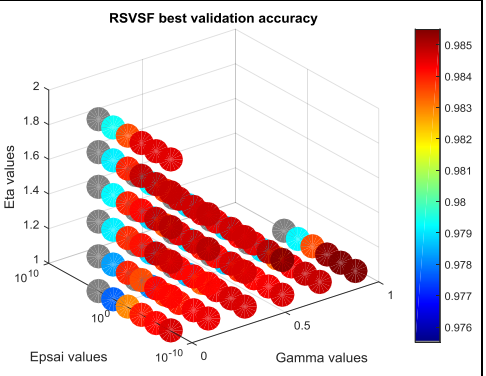
1- best validation performance in all training epochs,

2- transient validation performance after few epochs.

3.4.2.1 Best validation success rate for REKF and RSVSF

Figures 3-4 and 3-5 show the first performance measure, the best validation performance in all training epochs, for REKF and RSVSF respectively. The color represents the success rate result. In Figure 3-4 on (x, y) axes show the different values of the configurable REKF parameters (p, q), while (z) axis shows the values of the configurable parameter (r). In Figure 3-5, the results of RSVSF are shown with only the difference that the axes represent the RSVSF configuration parameters (ψ, γ, η). From the color bar it can be seen that all variations converged for both REKF and RSVSF at least 86% and 93% of success rate, respectively. This result proves the stability of REKF and RSVSF with any given initial values. To better visualize the best success rate variation across the different parameters, a color rescaling is applied and is presented in Figures 3-6 and 3-7 where any success rate below 97% is grayed out. These Figures show two very important results (1) that both filters are able to achieve high success rate no matter what are the initial configuration parameters, (2) the main factor in tuning RSVSF is (ψ) and the ratio $\left(\frac{q}{r}\right)$ in the case of REKF which forms the green plane.

Table 3-V Best validation success rate for both REKF and RSVSF

	REKF	RSVSF
Full range of success rate (the color represents the success rate result)	 Figure 3-4 REKF best validation success rate	 Figure 3-5 RSVSF best validation success rate
Rescaled color bar	 Figure 3-6 REKF best validation success rate (color rescaled)	 Figure 3-7 RSVSF best validation success rate (color rescaled)

3.4.2.2 Transient validation success rate for REKF and RSVSF

In order to measure the transient performance of REKF and RSVSF, their validation success rate is traced against time for the whole approximately 500 seconds of the training

process. In Figures 3-8 and 3-9 it can be seen that from the time from the seconds 20 to 25 is a good representative of the transient performance.

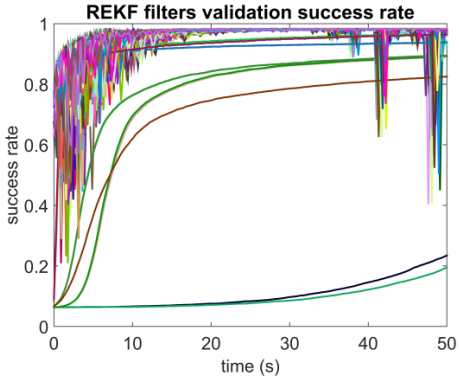


Figure 3-8 REKF validation success rate with time

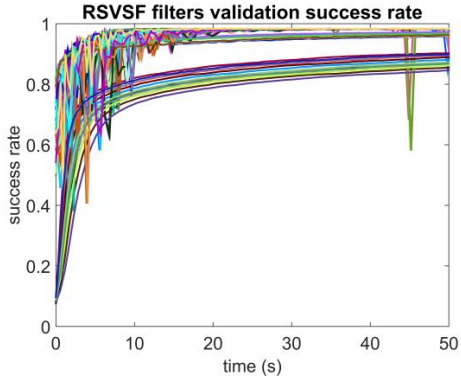
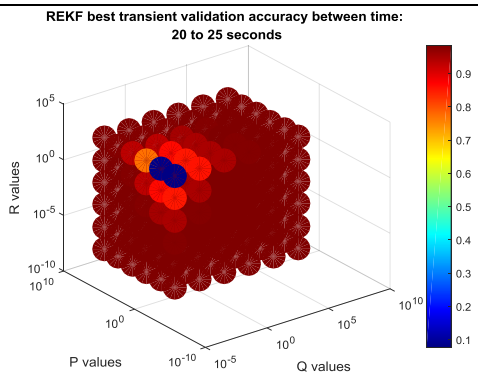
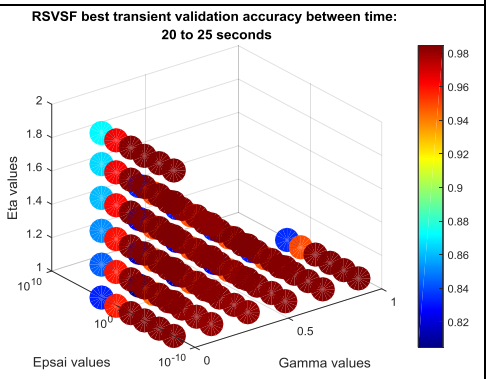


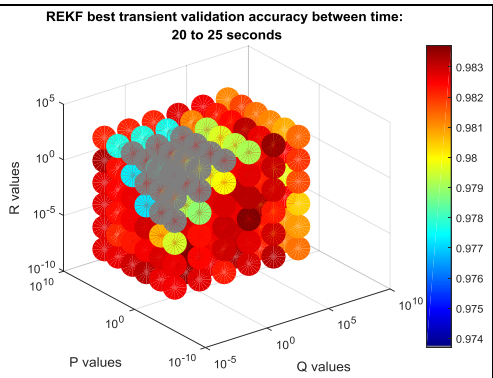
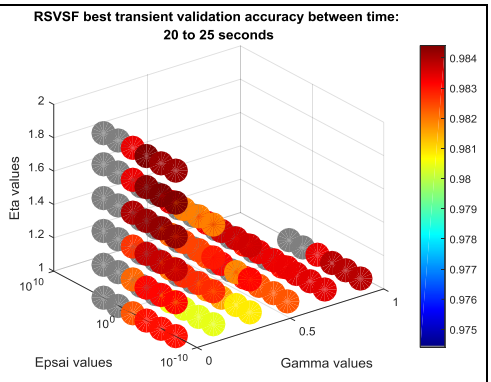
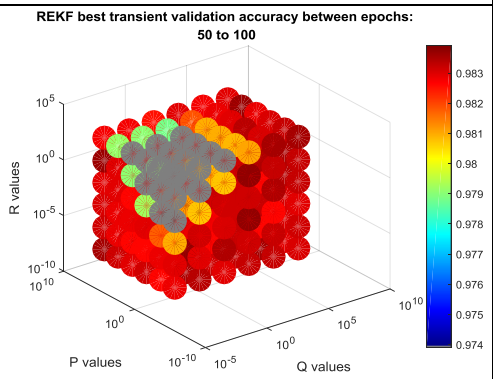
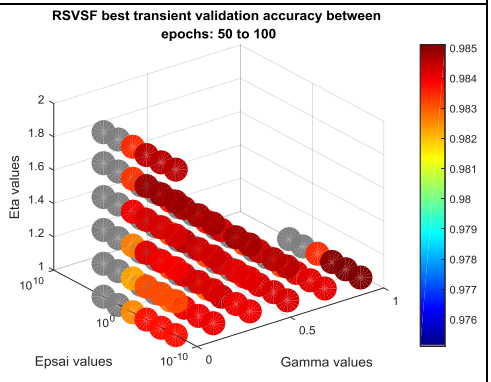
Figure 3-9 RSVSF validation success rate with time

To relate the transient performance to the configuration parameters Figures 3-10 to 3-15 are generated. Figures 3-10 and 3-11 show that the transient performance is still exceptional for any given configuration parameters. Only two trials in REKF – the blue circles- where the training was very slow. RSVSF on the other hand performed well under all configuration parameters trials. Figure 3-10 confirms the best success rate chart in the last section and proves that any combination on (q, r, p)allows REKF to achieve good transient performance except for very small $\left(\frac{q}{r}\right)$ value. Figure 3-11 shows that the primary tuning factor in RSVSF is (ψ). By rescaling the colors of these charts more insights can be seen in Figures 3-12 and 3-13. In REKF case the result confirms that any configuration parameter values are good for transient performance, only very high (q) value is less performing that the others. The reason for that is the numerical difficulty it introduces in the calculations so the optimizer takes more time per one epoch compared to the other

REKF configuration values. Therefore testing from 22 second to 25 is comparing between different epoch cycles. By unifying the epoch range in Figure 3-14, it can be seen even high (q)values match the rest of values in terms of performance. Similar behavior can be seen in RSVSF. In [7] it was proposed that (η) can substitute the effect of (γ) and shorten the calculation time by eliminating one forward cycle time. This shortening in time is confirmed in Figure 3-13 where at zero (γ) RSVSF converges very fast. While the substitution effect can be seen when same epochs are compared in Figure 3-15 as diagonal terms (constant $\eta + \gamma$) achieve similar performance.

Table 3-VI Transient validation success rate for both REKF and RSVSF

	REKF	RSVSF
Full range of success rate (the color represents the success rate result)	 Figure 3-10 REKF transient validation success rate (time based)	 Figure 3-11 RSVSF transient validation success rate (time based)

Rescaled color bar	 Figure 3-12 REKF transient validation success rate (time based and color rescaled)	 Figure 3-13 RSVSF transient validation success rate (time based and color rescaled)
Rescaled color bar and Epoch based	 Figure 3-14 REKF transient validation success rate (epoch based)	 Figure 3-15 RSVSF transient validation success rate (epoch based)

3.4.2.3 Sensitivity results summary

In this section the results can be summarized in the following:

1- REKF and RSVSF are stable and convergent no matter what are the initial configuration parameters.

2- RSVSF achieves higher success rate than REKF and Adam.

3- The main tunable parameters are (ψ) in RSVSF and (r) in REKF however if both are not extremely high, great transient and global performance is achievable with any value.

4- In RSVSF it is recommended to use high (η) value and zero (γ) to achieve high performance in short time.

3.5 REKF and RSVSF Multi-Filter versions

In this section, the Multi-Filter version of REKF and RSVSF are introduced. They are named REKF_MF and RSVSF_MF. In both REKF and RSVSF implementations on Deep Learning networks, the authors in [7] deployed one filter for the whole network. While this implementation has advantages as that the filter looks at all network weights before modifying them which produces stability, a question remained unanswered. This question was how implementing Multi-Filter, one per layer can affect the performance of the DNN training process. The hypothesis is that it might produce aggressive changes of weights per layer so it might be faster but that also might lead to instabilities and divergent results. In this section the answer to this question is investigated as well as their results for the same above mentioned sensitivity analysis are presented to compare the results with REKF and RSVSF.

3.5.1 REKF_MF

First is REKF_MF. This training is exactly similar to REKF except in that there is a different filter for each layer. The algorithm of REKF_MF is described in Table 3-VII. In the algorithm the parameters updates are done per layer $(\boldsymbol{i})$. This affects the filter behavior

as the innovation ($\boldsymbol{s}$), the gain ($\boldsymbol{K}$), and error covariance matrix ($\boldsymbol{P}$) are calculated based on the error gradient of only one layer. This change would allow REKF_MF to be more aggressive in changing the weights of the network but it can also cause instability as the changes per layer are done independent of the changes in other layers.

Table 3-VII REKF_MF algorithm

Algorithm 3: *REKF_MF*, this proposed algorithm deploys REKF filter per each layer

Require: $\boldsymbol{q, r\ values}$

Require: $\boldsymbol{p}$ initial values

$\boldsymbol{P_{1,2,..,n}} \leftarrow \begin{bmatrix} \boldsymbol{p} \\ \vdots \\ \boldsymbol{p} \end{bmatrix}$ Initialize $\boldsymbol{P_{1,2,..,n}}$ vectors filled with $\boldsymbol{p}$ initial value. where $\boldsymbol{n}$ is the number of layers

While $\boldsymbol{k < number\ of\ epochs}$

$\boldsymbol{k \leftarrow k + 1}$

 # calculate the network loss

$\boldsymbol{loss = MSE(target, network\ output)}$

 # the system target is to achieve zero loss, hence the system error is

$\boldsymbol{\tilde{y}_k = 0 - loss = (system\ target - system\ output)}$

$\boldsymbol{g_k = \nabla_\theta loss(\theta_{k-1})}$

$\boldsymbol{for\ i = 1{:}n\ do}$

$\boldsymbol{\theta_{i,(k-1)} \subset \theta_{(k-1)}}$

$\boldsymbol{where\ \theta_{i,(k-1)}\ are\ the\ parameters\ of\ layers\ i}$

$\boldsymbol{g_{i,k} \subset g_k}$

$\boldsymbol{where\ g_{i,k}\ are\ the\ gradients\ of\ layers\ i}$

$\boldsymbol{P_{i,(k|k-1)} = P_{i,(k-1|k-1)} \oplus q}$

#perform the update step

$\boldsymbol{s_{i,k} = g_{i,k} P_{i,(k|k-1)} g_{i,k}^T \oplus r}$

$\boldsymbol{K_{i,k} = P_{i,(k|k-1)} g_{i,k}^T \oslash s_{i,k}}$

$\boldsymbol{\theta_{i,k} = \theta_{i,(k-1)} + K_k \tilde{y}_k}$

$\boldsymbol{P_{i,(k|k)} = (I - K_{i,k} g_{i,k}) P_{i,(k|k-1)}}$

$\boldsymbol{end\ for}$

End while

Retun $\boldsymbol{\theta_k}$

3.5.2 RSVSF_MF

The second proposed algorithm in this paper is RSVSF_MF. Similar to REKF_MF, RSVSF_MF is a modification of RSVSF, where there is a different filter for each layer. Table 3-VIII presents the proposed algorithm. The parameters in the algorithm are updated per layer $(\boldsymbol{i})$. The effect is similar to REKF_MF as the gain $(K_{i,k})$ is dependent on the error gradient of only one layer. This change would allow RSVSF_MF to be more aggressive in changing the weights of the network but it can also cause instability as the weight updates per layer are done independent of the updates in other layers.

Table 3-VIII RSVSF_MF algorithm

Algorithm 4: *RSVSF_MF*, this proposed algorithm deploys RSVSF filter per each layer

Require: $\boldsymbol{\gamma, \psi, \eta\ values}$

While $\boldsymbol{k < number\ of\ epochs}$

$\boldsymbol{k \leftarrow k + 1}$

 # calculate the network loss

$\boldsymbol{loss = MSE(target, network\ output)}$

 # the system target is to achieve zero loss, hence the system error is

$\boldsymbol{\tilde{y}_k = 0 - loss = (system\ target - system\ output)}$

$\boldsymbol{g_k = \nabla_\theta loss(\theta_{k-1})}$

$\boldsymbol{for\ i = 1{:}n\ do}$

$\boldsymbol{\theta_{i,(k-1)} \subset \theta_{(k-1)}}$

$\boldsymbol{where\ \theta_{i,(k-1)}\ are\ the\ parameters\ of\ layers\ (i)}$

$\boldsymbol{g_{i,k} \subset g_k}$

$\boldsymbol{where\ g_{i,k}\ are\ the\ gradients\ of\ layers\ (i)}$

$$\boldsymbol{H^+_{i,k} = \frac{g^t_{i,k}}{\sum_{j=1}^{m}\left[\left(g_{i,k}(j)\right)^2\right]}}$$

$$\boldsymbol{K_{i,k} = H^+_{i,k}\left(\eta\left|\tilde{y}_{k|k-1}\right| + \gamma\left|\tilde{y}_{k-1|k-1}\right|\right) \circ sat\left(\frac{\tilde{y}_{k|k-1}}{\psi}\right)}$$

#perform the update step

$\boldsymbol{\theta_{i,k} = \theta_{i,(k-1)} + K_{i,k}}$

$\boldsymbol{end\ for}$

End while
Retun $\boldsymbol{\theta_k}$

3.6 Comparison Multi-Filter versions and the original forms of REKF and RSVSF

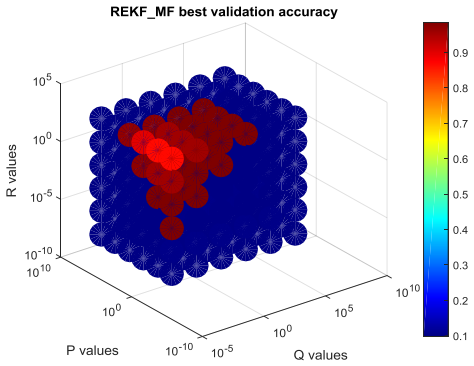
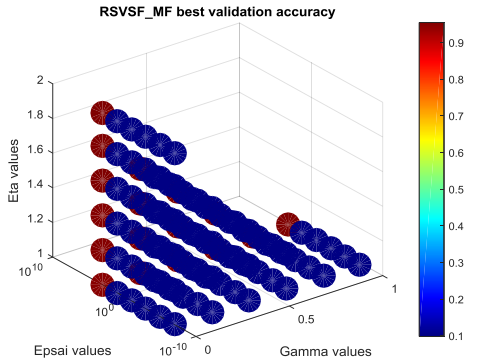
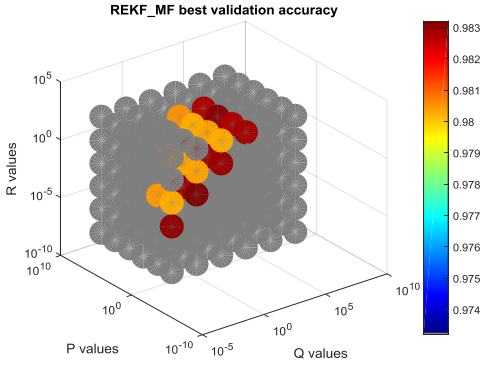
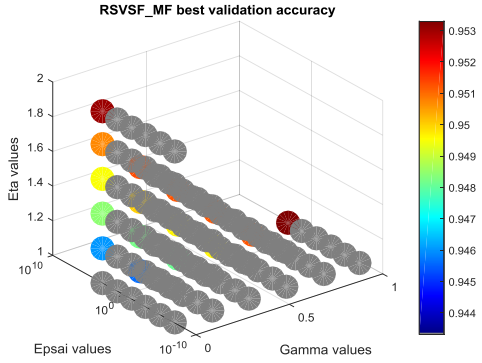
In this section the same testing procedure that was used in section 3.4 is used to have a direct comparison between REKF, RSVSF and their Multi-Filter versions.

3.6.1 Best validation success rate for REKF_MF and RSVSF_MF

This study is similar to the one in section 3.4.2.1. Here the Multi-Filter version results only is presented as the original forms results was shown previously. A quick comparison between the Figures in this section and the Figures in section 3.4.2.1 shows superiority for the original forms of REKF and RSVSF. The Multi-Filter version of the algorithms diverges in the most of the cases. The algorithms converge only with a specific ratio $\left(\frac{q}{r}\right)$ for the REKF_MF and with a high value of (ψ) for RSVSF_MF as shown in Figures 3-16 and 3-17 respectively. When the scale colors are focused on the higher success rate, the effect of the $\left(\frac{q}{r}\right)$ ratio becomes more visible as it creates a plane of convergence in Figure 3-18. Similarly in Figure 3-19, the colors become more descriptive of the performance across different (γ, η) values for the high (ψ) value. For a better angle, Figure 3-20 shows the same chart from another angle. It shows a colorful representation to the theory introduced in [7] that (η) can substitute the effect of (γ) while shortening the training time. In other words, for the same (ψ) value, the performance is dictated by

$(\gamma + \eta)$ value. Constant $(\gamma + \eta)$ values are the diagonal lines in Figure 3-20, and from the color scale they have similar performance levels.

Table 3-IX Best validation success rate for Multi-Filter version

	REKF_MF	RSVSF_MF
Full range of success rate (the color represents the success rate result)	 Figure 3-16 REKF_MF best validation success rate	 Figure 3-17 RSVSF_MF best validation success rate
Rescaled color bar	 Figure 3-18 REKF_MF best validation success rate (color rescaled)	 Figure 3-19 RSVSF_MF best validation success rate (color rescaled)

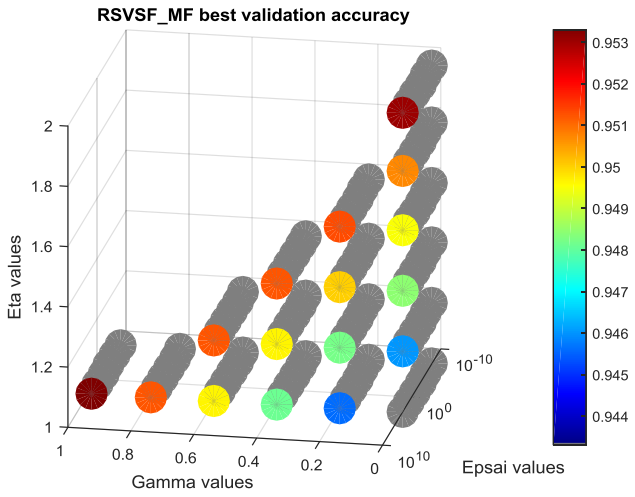


Figure 3-20 the opposite angle of Figure 3-19

3.6.2 Transient validation success rate for REKF_MF and RSVSF_MF

Similar to section 3.4.2.2, the transient results of the Multi-Filter versions are presented. Figures 3-21 and 3-22 show that the transient performance can still be measured from 20 to 25 seconds. This allows comparing the results with the original forms result in section 3.4.2.2. In the same Figures also it is easy to spot one main difference between the original forms and the Multi-version forms. This difference is that some Multi-filter trials were unstable and diverged in the very beginning of the training.

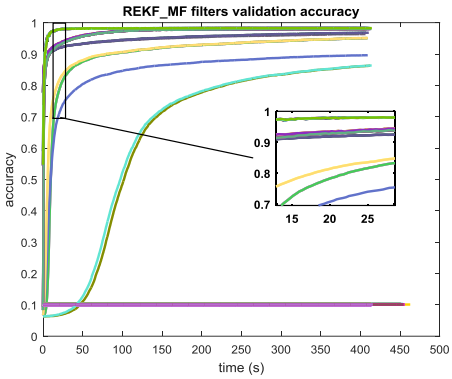


Figure 3-21 REKF_MF validation success rate with time

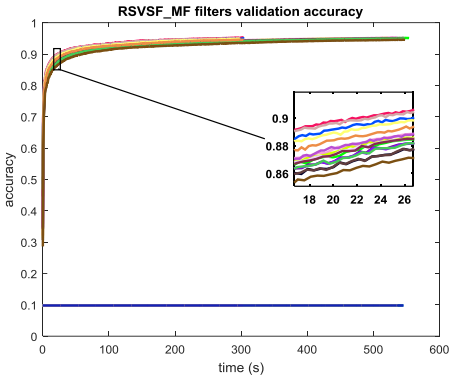
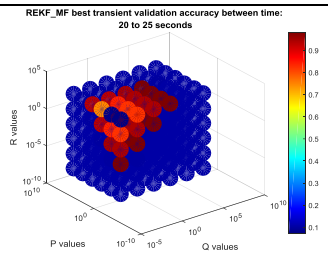
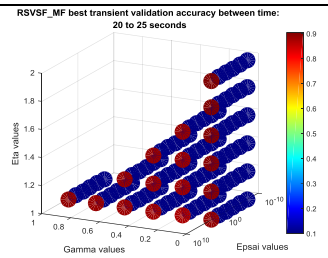
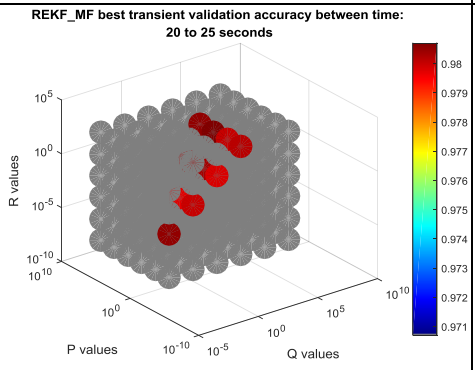
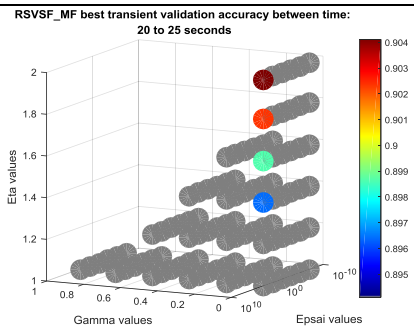
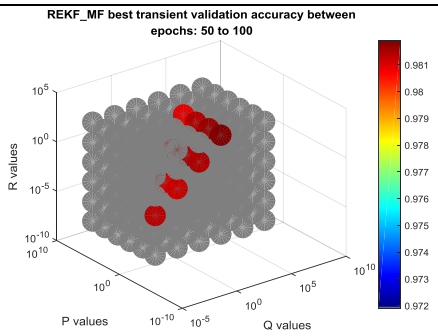
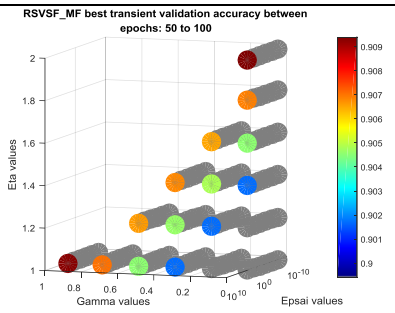


Figure 3-22 RSVSF_MF validation success rate with time

Figures 3-23 to 3-28 show the transient performance of the Multi-Filter version of the algorithms. Figures 3-23 and 3-24 show that only few variations converged, and they're in-line with the best validation success rate results. Figures 3-25 and 3-27 show that weather time or epoch based, the transient performance is similar across the $\left(\frac{q}{r}\right)$ ratio converging plane. Figure 3-26 shows the advantage of using (η) to increase the performance of the RSVSF_MF optimizer instead of (γ) as this reduces the required extra forward calculation for the a posteriori error. Figure 3-28 shows the transient performance equivalence for all the variations that have the same $(\gamma + \eta)$ value. These variations are the diagonal optimizers and it can be seen that they have the same colors which means the same performance.

Table 3-X Transient validation success rate for both REKF_MF and RSVSF_MF

	REKF_MF	RSVSF_MF

Full range of success rate (the color represents the success rate result)	 Figure 3-23 REKF_MF transient validation success rate (time based)	 Figure 3-24 RSVSF_MF transient validation success rate (time based)
Rescaled color bar	 Figure 3-25 REKF_MF transient validation success rate (time based and color rescaled)	 Figure 3-26 RSVSF_MF transient validation success rate (time based and color rescaled)
Rescaled color bar and Epoch based	 Figure 3-27 REKF_MF transient validation success rate (epoch based)	 Figure 3-28 RSVSF_MF transient validation success rate (epoch based)

3.6.3 Sensitivity results summary for REKF_MF, RSVSF_MF

In this section the results can be summarized in the following:

1- REKF_MF and RSVSF_MF are stable and convergent only for a fine tuned parameters
2- RSVSF_MF achieves less success rate than original RSVSF.
3- The main tunable parameters are (ψ) in RSVSF_MF and $\left(\frac{q}{r}\right)$ in REKF_MF and both have to be tuned for the optimizers to converge.
4- In RSVSF_MF It is recommended to use high (η) value and zero (γ) to achieve high performance in short time.

One important note that can be noticed in Figures 3-7 and 3-19 is that the RSVSF_MF has a lower best validation performance than the original form. This behavior is repeatable in the transient performance as can be seen in Figures 3-13 and 3-26. A summarized comparison between the Multi-Filter version and the original forms is presented in Table 3-XI.

Table 3-XI comparison between Multi-Filter version and the original form of REKF, and RSVSF

	REKF	RSVSF	REKF_MF	RSVSF_MF
Advantages	- Fast - Low memory - Stable and convergent with configuration variation - (r) is the most important fine tuning parameter	- Fast - Less memory than REKF, and Adam - Stable and convergent with configuration variation - More accurate than Adam, and REKF. - Using (η) instead of (γ) reduces the training time by one third	- $(\frac{q}{r})$ is the most important fine tuning ratio - Scores slightly less performance than REKF when converges	- (ψ) is the most important fine tuning parameter - Using less memory than REKF_MF, and Adam.

		- (ψ) is the most important fine tuning parameter		
Disadvantages	- More accurate than Adam, but less accurate than RSVSF - Uses more memory than RSVSF	- In case of ($\gamma \neq 0$), RSVSF requires one more forward calculation	- Hard to fine tune - Easily gets divergent and unstable	- Hard to fine tune - Scores much less performance than RSVSF when converges - Easily gets divergent and unstable - In case of ($\gamma \neq 0$), RSVSF_MF requires one more forward calculation

From Table 3-XI, the recommendation is to use RSVSF, with very low (ψ)value, $(\gamma = 0)$ and high (η) value. This is the best and the fastest stable performance can be achieved in global and in transient training conditions.

3.7 Conclusion

In this paper, a sensitivity analysis was applied on REKF and RSVSF to define the most important factors in fine tuning the algorithms for Deep Learning training application. The analysis was done based on both transient and global best performance achieved. REKF and RSVSF showed great robustness and they need no fine tuning as long as (ψ) and (r) in RSVSF and REKF respectively have small values. In this paper also the Multi-Filter implementations of REKF and RSVSF is proposed REFK_MF and RSVSF_MF respectively. It is shown that they suffer from stability issues compared with their original forms as they need accurate non-forgiving fine tuning for their parameters. This paper reinforces the robustness and ability of REKF and RSVSF to converge fast to a high performance solution without the need of fine tuning their parameters.

Acknowledgement

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), funding reference number CRDPJ 486107-15.

References

[1] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012, pp. 3642-3649.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare*, et al.*, "Human-level control through deep reinforcement learning," *Nature,* vol. 518, pp. 529-533, 2015.

[3] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *Journal of Big Data,* vol. 2, p. 1, 2015.

[4] J. Heaton, N. Polson, and J. H. Witte, "Deep learning for finance: deep portfolios," *Applied Stochastic Models in Business and Industry,* vol. 33, pp. 3-12, 2017.

[5] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA Transactions on Signal and Information Processing,* vol. 3, 2014.

[6] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau*, et al.*, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature,* vol. 542, pp. 115-118, 2017.

[7] M. Ismail, M. Attari, S. Habibi, and S. Ziada, "Estimation theory and Neural Networks revisited: REKF and RSVSF as optimization techniques for Deep-Learning," *Neural Networks,* 2018.

[8] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747,* 2016.

[9] V. G. Gudise and G. K. Venayagamoorthy, "Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks," in *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, 2003, pp. 110-117.

[10] F. H.-F. Leung, H.-K. Lam, S.-H. Ling, and P. K.-S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE Transactions on Neural networks,* vol. 14, pp. 79-88, 2003.

[11] R. S. Sexton, R. E. Dorsey, and J. D. Johnson, "Optimization of neural networks: A comparative analysis of the genetic algorithm and simulated annealing," *European Journal of Operational Research,* vol. 114, pp. 589-601, 1999.

[12] M. J. D. Powell, "Restart procedures for the conjugate gradient method," *Mathematical programming,* vol. 12, pp. 241-254, 1977.

[13] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural networks,* vol. 6, pp. 525-533, 1993.

[14] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980,* 2014.

[15] S. S. Haykin, *Kalman filtering and neural networks*: Wiley Online Library, 2001.
[16] S. Habibi, "The Smooth Variable Structure Filter," *Proceedings of the IEEE,* vol. 95, pp. 1026-1059, 2007.

Abbreviations

Abbreviation	Description
DL	Deep-Learning
DNN	Deep Neural Network
FNN	Feedforward Neural Networks
MSE	Mean Square Error
REKF	Reduced-EKF
FDEKF	Fully-Decoupled-EKF
RSVSF	Reduced-SVSF
REKF-MultiFilter	REKF_MF
RSVSF-MultiFilter	RSVSF_MF
Rectified Linear Units	ReLU's

Chapter 4: Modified Smooth Variable Structure Filter

Mahmoud Ismail[a,*], and Saeid Habibi[a]

[a] Department of Mechanical Engineering, McMaster University, Hamilton, Canada;

[*]**Corresponding author**: Mahmoud Ismail, Department of Mechanical Engineering, McMaster University, Hamilton, Canada.

This journal paper is submitted to Transactions of the Canadian Society for Mechanical Engineering in September 2018.

Abstract

Estimation theory is used extensively in different applications due its ability to either estimate a system state that is not measured or filter a measured one. The Kalman Filter (KF) and The Smooth Variable Structure Filter (SVSF) are accurate and robust estimation algorithms. In this paper a modification of the SVSF is discussed to improve the estimation accuracy. The paper studies the stability of such a modification and conducts a comparative analysis to evaluate its performance compared with the original form of the SVSF as well as the KF. The improvement in the estimation accuracy will result in better performance across different applications such as system monitoring, fault detection and diagnosis, and control applications.

Keywords

Smooth Variable Structure Filter, Noise Filter, MSVSF.

4.1 Introduction

State estimation is an important field in modern science. It forms a corner stone in most control and measurement systems. State estimation can be used to estimate the internal states of a system using its inputs and outputs. It is a method for information extraction from signals and can reduce the need for complicated physical measurement devices [1]. Model-based state estimation can also provide filtering of noise from measurement signals [2]. It can also be used in Fault Detection and Diagnosis by monitoring a system and raising a flag once the system deviates from its expected behavior [3, 4].

In literature, there are many types of observers. For simple linear systems, linear state observers such as Luenberger observer can be sufficient [5]. In stochastic systems and where optimality is required the Kalman Filter (KF) [6-8] has been used subject to white noise. The next level of complexity for estimation in stochastic systems relate to non-linearity. Other types of filters such as the Extended Klaman Filter (EKF) and the Unscented Kalman Filter (UKF) have been proposed [9-12]. Interacting Multiple Model (IMM) [13, 14] can be added to stochastic filters to enable their application to systems with changing dynamics and for combining filtering strategies at different operating conditions. The Smooth Variable Structure Filter (SVSF)[3, 15] is based on sliding mode concept and has similarities to sliding mode observers [16]. It was found to be robust against modeling uncertainties [3, 17]. It has been applied in many fields such as battery state of charge estimation [18, 19], target tracking [20-22], and Neural Networks optimization [23]. SVSF has a predictor-corrector form similar to KF. For a nonlinear system defined in Equations (4-1) and (4-2),

$$x_k = f(x_{k-1}, u_{k-1}) + w_{k-1}^{noise} \tag{4-1}$$

$$z_k = h(x_k) + v_k^{noise} \tag{4-2}$$

the prediction step is obtained as follows,

$$\hat{x}_{k|k-1} = f\left(\hat{x}_{k-1|k-1}, u_{k-1}\right) \tag{4-3}$$

$$\hat{z}_{k|k-1} = h\left(\hat{x}_{k|k-1}\right) \tag{4-4}$$

The update step is as follows,

$$\tilde{e}_{k|k-1} = z_k - \hat{z}_{k|k-1} \tag{4-5}$$

$$K_k = H_k^+\left(\left|\tilde{e}_{k|k-1}\right| + \gamma \left|\tilde{e}_{k-1|k-1}\right|\right) \circ sat\left(\frac{\tilde{e}_{k|k-1}}{\psi}\right) \tag{4-6}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tag{4-7}$$

$$\hat{z}_{k|k} = h\left(\hat{x}_{k|k}\right) \tag{4-8}$$

$$\tilde{e}_{k|k} = z_k - \hat{z}_{k|k} \tag{4-9}$$

Where (H_k^+) is the pseudoinverse of (H_k) as defined in Equation (4-10), (γ) is the a posteriori error gain, and (ψ) is the smoothing boundary layer width. (ψ) and (γ), in general, are diagonal or pseudo-diagonal matrices. The saturation function works as presented in Equation (4-11). The operator $(\circ)$ is element-wise (schur) multiplication.

$$H_k = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_{k|k-1}} \tag{4-10}$$

$$sat\left(\frac{\tilde{e}_{k|k-1}}{\psi}\right) = \begin{cases} +1 & if \quad \left(\frac{\tilde{e}_{k|k-1}}{\psi}\right) \geq 1 \\ \left(\frac{\tilde{e}_{k|k-1}}{\psi}\right) & if \quad -1 < \left(\frac{\tilde{e}_{k|k-1}}{\psi}\right) < +1 \\ -1 & if \quad \left(\frac{\tilde{e}_{k|k-1}}{\psi}\right) \leq -1 \end{cases} \tag{4-11}$$

For a linear system defined in Equations (4-12) and (4-13), the SVSF prediction step is changed to Equations (4-14) and (4-15), and the update steps are the same, except that the measurement Equation (4-8) becomes linear as shown by Equation (4-16).

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1}^{noise} \quad (4\text{-}12)$$

$$z_k = Hx_{k-1} + v_k^{noise} \quad (4\text{-}13)$$

$$\hat{x}_{k|k-1} = \hat{A}\hat{x}_{k-1|k-1} + \hat{B}u_{k-1} \quad (4\text{-}14)$$

$$\hat{z}_{k|k-1} = H\hat{x}_{k|k-1} \quad (4\text{-}15)$$

$$\hat{z}_{k|k} = H\hat{x}_{k|k} \quad (4\text{-}16)$$

4.2 Modified-SVSF

The Modified-SVSF is an extension to the original SVSF filter. The main concept in Modified-SVSF is to multiply the a priori error by a gain called η. To understand how the introduction of this term is affecting the stability and the filter performance, first a linear system which is defined in Equations (4-12) and (4-13) is restated here, as:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1}^{noise} \quad (4\text{-}17)$$

$$z_k = Hx_{k-1} + v_k^{noise} \quad (4\text{-}18)$$

The Modified-SVSF is similar to the SVSF, but with a modified gain. Here, the prediction is similar to the SVSF:

$$\hat{x}_{k|k-1} = \hat{A}\hat{x}_{k-1|k-1} + \hat{B}u_{k-1} \quad (4\text{-}19)$$

$$\hat{z}_{k|k-1} = \hat{H}\hat{x}_{k|k-1} \quad (4\text{-}20)$$

In the update step, the gain is modified by adding η as shown in Equation (4-22).

$$\tilde{e}_{k|k-1} = z_k - \hat{z}_{k|k-1} \quad (4\text{-}21)$$

$$K_k = \hat{H}^{+}\left(\eta\left|\tilde{e}_{k|k-1}\right| + \gamma\left|\tilde{e}_{k-1|k-1}\right|\right) \circ sgn\left(\tilde{e}_{k|k-1}\right) \quad (4\text{-}22)$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \quad (4\text{-}23)$$

$$\hat{z}_{k|k} = \hat{H}\hat{x}_{k|k} \quad (4\text{-}24)$$

$$\tilde{e}_{k|k} = z_k - \hat{z}_{k|k} \quad (4\text{-}25)$$

An state estimation process is asymptotically stable and convergent if satisfies the condition of Equation (4-26):

$$\left|\tilde{e}_{k|k}\right| < \left|\tilde{e}_{k-1|k-1}\right| \quad (4\text{-}26)$$

This can be proven by choosing a Lyapunov function ($\upsilon_k = e_k^2$), in this case the process is stable if ($\Delta\, \upsilon_k = e_k^2 - e_{k-1}^2$) < 0, which is satisfied by the condition in Inequality (*4-26*).

Substituting Equation (4-20) in Equation (4-21),

$$\tilde{e}_{k|k-1} = z_k - \hat{H}\hat{x}_{k|k-1} \quad (4\text{-}27)$$

Using Equation (4-23), this can be rearranged to

$$\tilde{e}_{k|k-1} = z_k - \hat{H}(\hat{x}_{k|k} - K_k) \quad (4\text{-}28)$$

Rearranging this Equation and using Equation (4-25) results in

$$\hat{H}K_k = \tilde{e}_{k|k-1} - \tilde{e}_{k|k} \quad (4\text{-}29)$$

By taking the absolute value of both sides and using the mathematical lemma ($|a - b| \leq |a| + |b|$):

$$\left|\hat{H}K_k\right| \leq \left|\tilde{e}_{k|k-1}\right| + \left|\tilde{e}_{k|k}\right| \quad (4\text{-}30)$$

Substituting in Equation (4-26), the previous equation results in,

$$\left|\hat{H}K_k\right| < \left|\tilde{e}_{k|k-1}\right| + \left|\tilde{e}_{k-1|k-1}\right| \quad (4\text{-}31)$$

This is the condition for the filter gain to result in a stable estimation process. According to this stability condition, a filter gain that is stable can be defined as in Equation (4-22), where (η, γ) are tunable parameters to satisfy the stability condition. To examine the stability range of the tunable parameters (η, γ) for a linear system, from Equations (4-21), (4-22), (4-25), and (4-29) the a posteriori error can be defined as,

$$\tilde{e}_{k|k} = \tilde{e}_{k|k-1} - \hat{H}\left[\hat{H}_k^+\left(\eta\left|\tilde{e}_{k|k-1}\right| + \gamma\left|\tilde{e}_{k-1|k-1}\right|\right) \circ sgn\left(\tilde{e}_{k|k-1}\right)\right] \quad (4\text{-}32)$$

Simplifying this results in

$$\tilde{e}_{k|k} = -\left(-\left|\tilde{e}_{k|k-1}\right| + \eta\left|\tilde{e}_{k|k-1}\right| + \gamma\left|\tilde{e}_{k-1|k-1}\right|\right) \circ sgn\left(\tilde{e}_{k|k-1}\right) \quad (4\text{-}33)$$

By taking the absolute value of both sides:

$$\left|\tilde{e}_{k|k}\right| = (\eta - 1)\left|\tilde{e}_{k|k-1}\right| + \gamma\left|\tilde{e}_{k-1|k-1}\right| \quad (4\text{-}34)$$

To eliminate $\left|\tilde{e}_{k|k-1}\right|$ and express it in terms of $\left|\tilde{e}_{k-1|k-1}\right|$, Equations (4-17) and (4-18) are substituted and rearranged, resulting in:

$$H^+(z_k - v_k^{noise}) = AH^+(z_{k-1} - v_{k-1}^{noise}) + Bu_{k-1} + w_{k-1}^{noise} \quad (4\text{-}35)$$

$$(z_k - v_k^{noise}) = HAH^+(z_{k-1} - v_{k-1}^{noise}) + HBu_{k-1} + Hw_{k-1}^{noise} \quad (4\text{-}36)$$

$$z_k = HAH^+(z_{k-1}) + \overline{B}u_{k-1} + \overline{w}_{k-1}^{noise} \quad (4\text{-}37)$$

Where,

$$\overline{B} = HB \quad (4\text{-}38)$$

$$\overline{w}_{k-1}^{noise} = v_k^{noise} - HAH^+v_{k-1}^{noise} + Hw_{k-1}^{noise} \quad (4\text{-}39)$$

Similarly, for the estimation system it can be found from Equations (4-19), (4-20), and (4-24) that,

$$\hat{z}_{k|k-1} = \hat{H}\hat{A}\hat{H}^+\left(\hat{z}_{k-1|k-1}\right) + \overline{\hat{B}}u_{k-1} \quad (4\text{-}40)$$

Where,

$$\overline{\hat{B}} = \hat{H}\hat{B} \tag{4-41}$$

Subtracting Equation (4-40) from (4-37), the following is found:

$$z_k - \hat{z}_{k|k-1} = HAH^{+}(z_{k-1}) - \hat{H}\hat{A}\hat{H}^{+}\left(\hat{z}_{k-1|k-1}\right) + \overline{B}u_{k-1} - \overline{\hat{B}}u_{k-1} + \overline{w}_{k-1}^{noise} \tag{4-42}$$

$$\tilde{e}_{k|k-1} = HAH^{+}(z_{k-1}) - \hat{H}\hat{A}\hat{H}^{+}\left(\hat{z}_{k-1|k-1}\right) + \overline{B}u_{k-1} - \overline{\hat{B}}u_{k-1} + \overline{w}_{k-1}^{noise} \tag{4-43}$$

In most control systems, the measurement matrix H is known precisely, therefore in the previous Equation it can be said that $\hat{H} = H$. If the uncertainty in $\hat{A}, \hat{B}$ leads to white noise, then the above mentioned equations can be reduced to the following by taking the first moment (the expectation) of the absolute of both sides,

$$E\left[\left|\tilde{e}_{k|k-1}\right|\right] = E\left[\left|HAH^{+}(z_{k-1}) - \hat{H}\hat{A}\hat{H}^{+}\left(\hat{z}_{k-1|k-1}\right) + \overline{B}u_{k-1} - \overline{\hat{B}}u_{k-1} + \overline{w}_{k-1}^{noise}\right|\right] \tag{4-44}$$

Considering that,

$$E\left[\overline{w}_{k-1}^{noise}\right] = 0\,, E\left[\hat{A}\right] = A, E\left[\hat{B}\right] = B, \hat{H} = H \tag{4-45}$$

Equation (4-44) becomes,

$$E\left[\left|\tilde{e}_{k|k-1}\right|\right] = E\left[\left|HAH^{+}\left(\tilde{e}_{k-1|k-1}\right)\right|\right] = E\left[\left|\varphi\tilde{e}_{k-1|k-1}\right|\right] \tag{4-46}$$

Substituting the previous equation in the expectation of Equation (4-34), results in,

$$E\left[\left|\tilde{e}_{k|k}\right|\right] = E\left[\left((\eta-1)\varphi+\gamma\right)\left|\tilde{e}_{k-1|k-1}\right|\right] = \left((\eta-1)\varphi+\gamma\right)E\left[\left|\tilde{e}_{k-1|k-1}\right|\right] \tag{4-47}$$

In the previous equation to have a stable state estimation process, the matrix $\left((\eta-1)\varphi+\gamma\right)$ has to have eigenvalues that are less than one.

$$0 \leq EigenValues\left((\eta-1)\varphi+\gamma\right) < 1 \tag{4-48}$$

Equation (4-48) has the term φ in its stability condition. This means that the arbitrary choice of the tuning parameters η, γ is limited by the system dynamics that satisfies the stability condition.

4.2.1 Smoothing boundary layer

Using the gain that is defined in Equation (4-22), introduces chattering along the trajectory of the estimated state, as shown in Figure 4-1. This chattering is a result of the sign function in the gain.

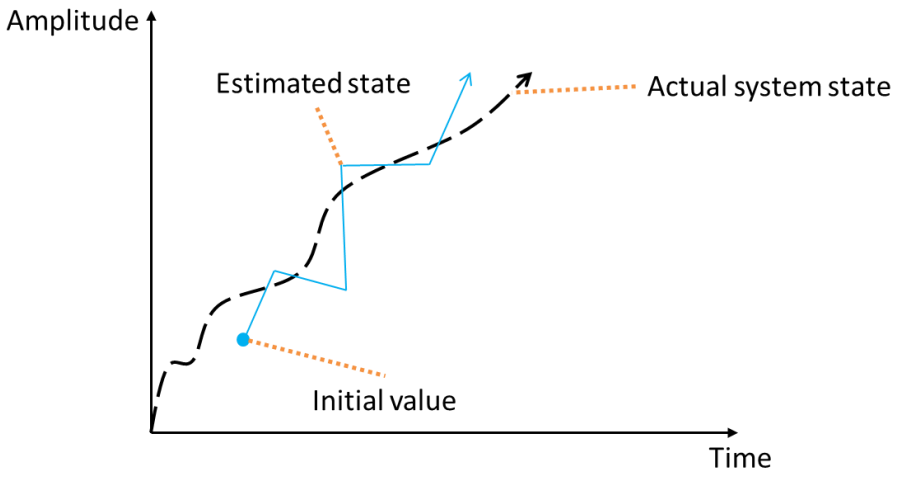


Figure 4-1 Chattering state estimation using the gain in Equation (4-22)

In the original SVSF [3] a saturation function is introduced to reduce this chattering effect. The concept is to apply a smoothing function around the switching surface [24] with a known boundary layer. This boundary layer is shown in Figure 4-2. When the error in estimation falls beyond the smoothing boundary layer, the filter falls back to the switching

mode to maintain stability. However within the smoothing boundary layer, the filter uses a smoothing function. A suggested smoothing function is a saturation function that is similar to the one used in the original form of the SVSF. The saturation function is defined in Equation (4-11), and results in the Modified-SVSF gain:

$$K_k = H^+\left(\eta\left|\tilde{e}_{k|k-1}\right| + \gamma \left|\tilde{e}_{k-1|k-1}\right|\right) \circ sat\left(\frac{\tilde{e}_{k|k-1}}{\psi}\right) \tag{4-49}$$

For nonlinear systems to

$$K_k = H_k^+\left(\eta\left|\tilde{e}_{k|k-1}\right| + \gamma \left|\tilde{e}_{k-1|k-1}\right|\right) \circ sat\left(\frac{\tilde{e}_{k|k-1}}{\psi}\right) \tag{4-50}$$

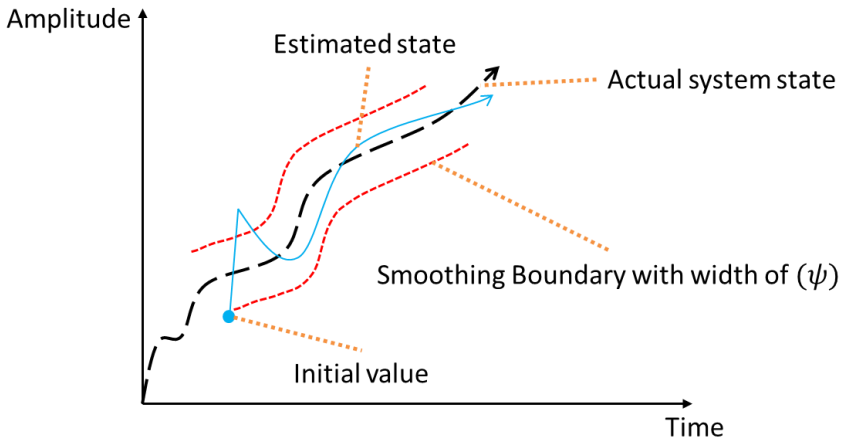


Figure 4-2 Modified-SVSF with a smoothing function within a boundary layer

4.3 Comparative study

In [3], the SVSF performance was evaluated using a linear aerospace actuation system model, namely the Electro-Hydraulic Actuator (EHA). For comparative purposes, the same system model is used to test the Modified SVSF filter.

4.3.1 Application

The Electro-Hydraulic Actuator (EHA) [25-27] is an actuation system designed for moving large loads with relatively high accuracies. The EHA is mathematically modelled as follows:

$$\frac{x(s)}{y(s)} = \frac{\frac{2D_p\beta_e A_E}{MV_o}}{s^3 + s^2\left(\frac{B_E}{M} + \frac{L}{V_o}\beta_e\right) + s\left(\frac{2\beta_e A_E^2}{MV_o}\right)} \tag{4-51}$$

Where D_p is the Pump volumetric displacement, β_e is the viscous friction coefficient, A_E is the Actuator working area, $B_E$is the load friction, M is mass, V_o is mean actuator volume, and L is Leakage coefficient

The mathematical model presented in Equation (4-51) is a simplified one that represents the dominant dynamics of the system. In this simulation the same parameter values used in [3] are used to compare the M-SVSF results with the original SVSF. The parameters are listed in Table 4-I.

Table 4-I Parameter values for the EHA system model

Parameter	Physical Significance	Actual Value
A_E	Piston Area	$5.05\times10^{-4}\ m^2$
B_E	Load Friction	$760\ Ns/m$
D_p	Pump Displacement	$1.69\times10^{-7}\ m^3/rad$
L	Leakage Coefficient	$2.5\times10^{-11}\ m^3/Pa-s$
M	Load Mass	$20\ Kg$

V_o	Chamber Volume	$6.85\times10^{-5}\ m^3$
β_e	Effective Bulk Modulus	$1.5\times10^{8} \leftrightarrow 2\times10^{8}\ Pa$

When parameters values are substituted and the system is put in a state-space form, it results in the state-space model with the following matrices, $A = \begin{bmatrix} 1 & 0.001 & 0 \\ 0 & 1 & 0.001 \\ -557.02 & -28.616 & 0.9418 \end{bmatrix}$ and $B = \begin{bmatrix} 0 \\ 0 \\ 557.02 \end{bmatrix}$. in this model, the system is assumed to be completely observable, and the measurement matrix $H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. The system noise is white noise with a maximum amplitude of $W_{max} = \begin{bmatrix} 0.01 \\ 1 \\ 10 \end{bmatrix}$. The measurement noise is also white noise with maximum amplitude of $V_{max} = \begin{bmatrix} 0.01 \\ 10 \\ 100 \end{bmatrix}$. The smoothing boundary layer (ψ) was set to be zero in order to examine the stability of the Modified-SVSF. In [3], the system was subject to a model change after 0.5 s with uncertainties in order to investigate robustness. In this study the same approach is followed with the new system matrix of $A_2 = \begin{bmatrix} 1 & 0.001 & 0 \\ 0 & 1 & 0.001 \\ -240 & -28 & 0.9418 \end{bmatrix}$. The input to the system is also the same to the one used in [3]. The input is shown in Figure 4-3 and includes a unit step that happens at 0.5 s with an additive random signal ranging from -1 to 1 rad/s. In order to validate stability, the simulation is run for 50 seconds. The sampling time used in this simulation is $\tau = 0.001\ s$.

For comparison, simulation is used for comparing the Modified-SVSF with the original SVSF with parametric values as given in [3], specifically with $\gamma = 0.5\, I$.

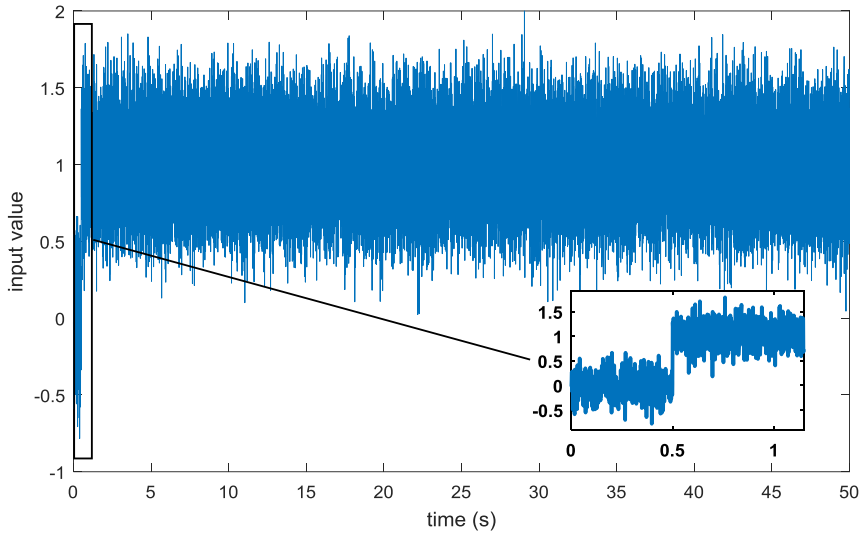


Figure 4-3 EHA simulation input

4.3.2 Modified-SVSF configuration

To completely evaluate the performance of the proposed filter, Modified SVSF, the simulation tested a range of values for η and γ. This range is summarized in Table 4-II.

Table 4-II Range of Modified-SVSF tuning parameters

Parameter	Starting value	Step value	End value
η	0	0.05	2
γ	0	0.05	2

The reason of choosing these ranges is to show and confirm the stability analysis conducted in Section 4.2. Considering the selected system measurement and system matrices (H, A), the stability condition for the system becomes,

$$0 \leq EigenValues\left((\eta-1)\times\begin{bmatrix}1 & 0.001 & 0\\ 0 & 1 & 0.001\\ -240 & -28 & 0.9418\end{bmatrix}+\gamma\right)<1 \qquad (4\text{-}52)$$

By iterating all the different values of η and γ that are shown in Table 4-II, a stability region for EHA system can be found. In Figure 4-4, the green region contains the stable points as they satisfy the stability inequality (4-52). Red dots represent the unstable region of values.

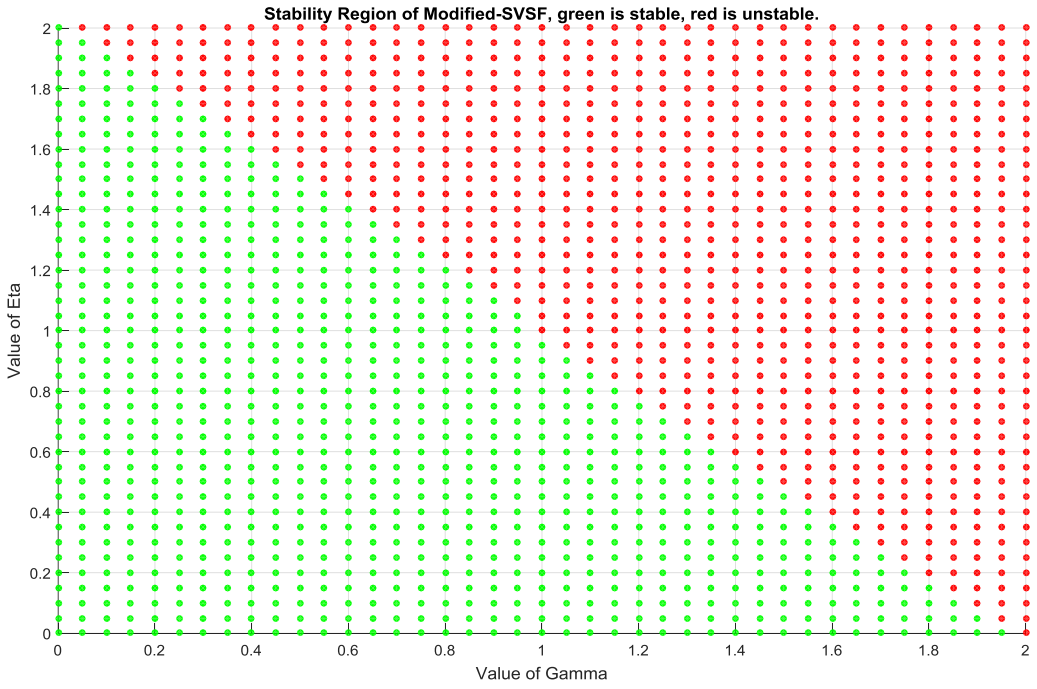


Figure 4-4 Stability Region of Modified-SVSF for EHA system

4.3.3 Results

The following charts show the RMSE for the state estimation of the original SVSF and the Modified-SVSF. However to show the difference in performance between the SVSF and the Modified-SVSF, the charts show on Z axis the difference between RMSE of Modified-SVSF and the original form of SVSF (with $\gamma_{ii} = 0.5$). The areas where the difference in

the RMSE of the two methods is less than zero ($MSVSF\ RMSE - SVSF\ RMSE < 0$) show the settings where the Modified-SVSF has a better performance. The areas where the difference is positive ($MSVSF\ RMSE - SVSF\ RMSE > 0$) is flattened to zero in the figures for visualization purposes. The positive area (the peak plateau) is where the Modified-SVSF has diverged and became unstable. This visualization is important as it shows the areas where Modified-SVSF performs better than SVSF and as well confirms the stability analysis on the same chart.

Figure 4-5 presents the position RMSE difference. It can be seen that for several combinations of η and γ, the Modified-SVSF is performing better than the original SVSF.

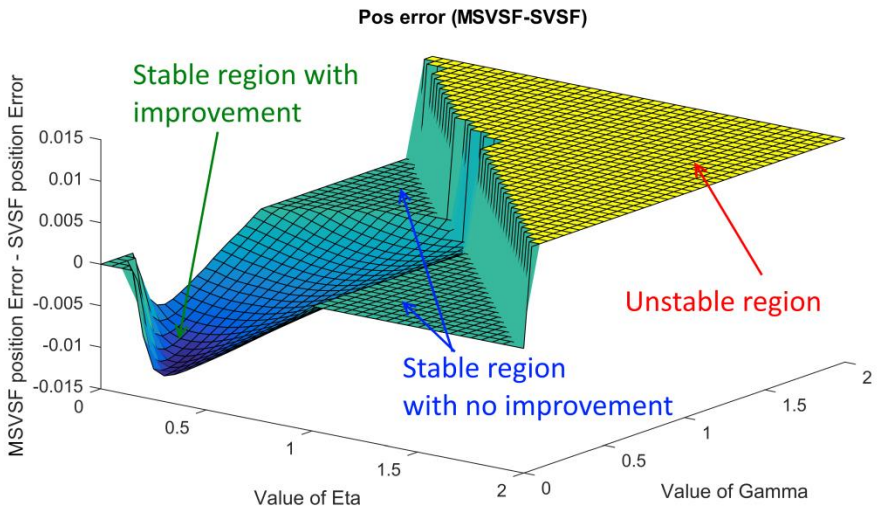


Figure 4-5 Position error difference of SVSF and Modified-SVSF

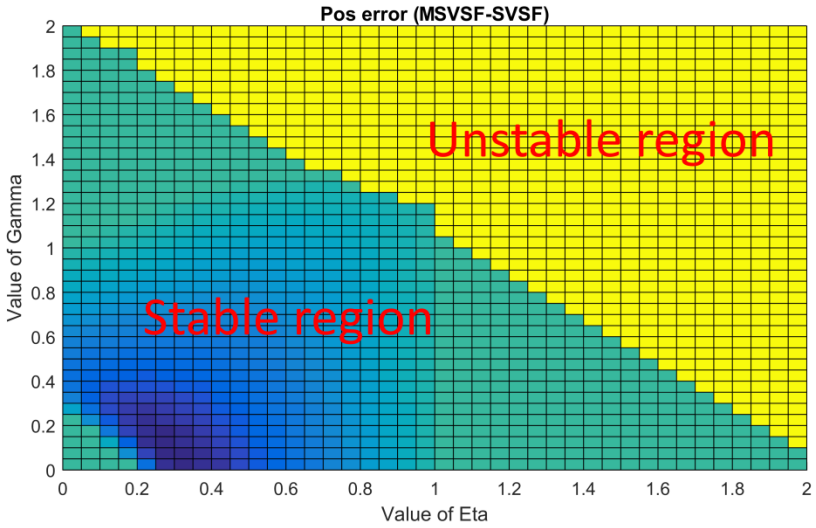


Figure 4-6 Position error difference of SVSF and Modified-SVSF XY-view

Similar results can be seen in terms of velocity error as shown in Figure 4-7. For the acceleration, no improvement can be observed in Figure 4-9. In Figures 10, 12, and 14 the stability region confirms the theoretical stability region in Figure 4-4.

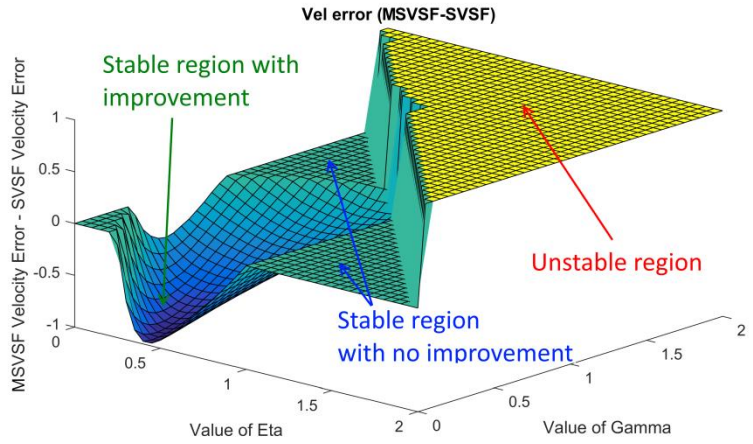


Figure 4-7 Velocity error difference of SVSF and Modified-SVSF

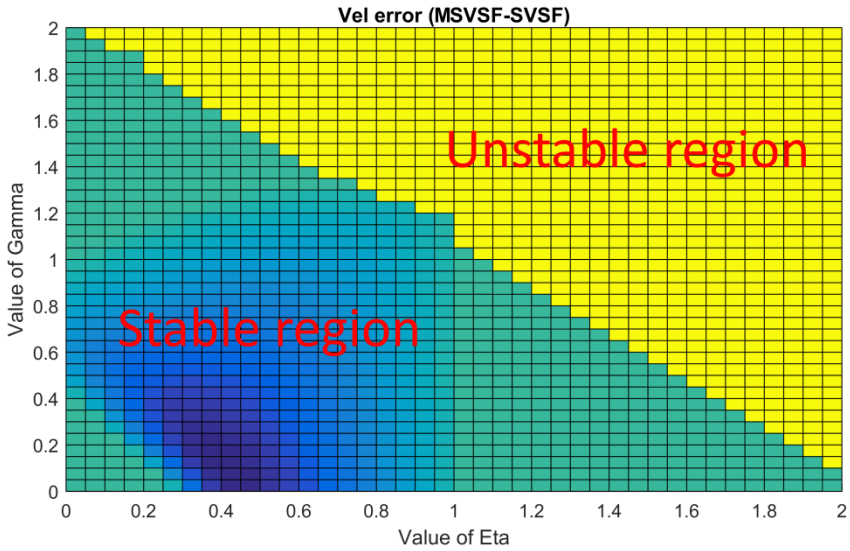


Figure 4-8 Velocity error difference of SVSF and Modified-SVSF XY-view

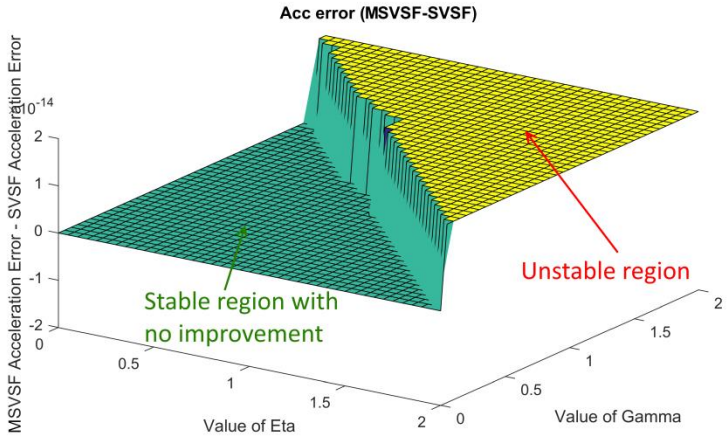


Figure 4-9 Acceleration error difference of SVSF and Modified-SVSF

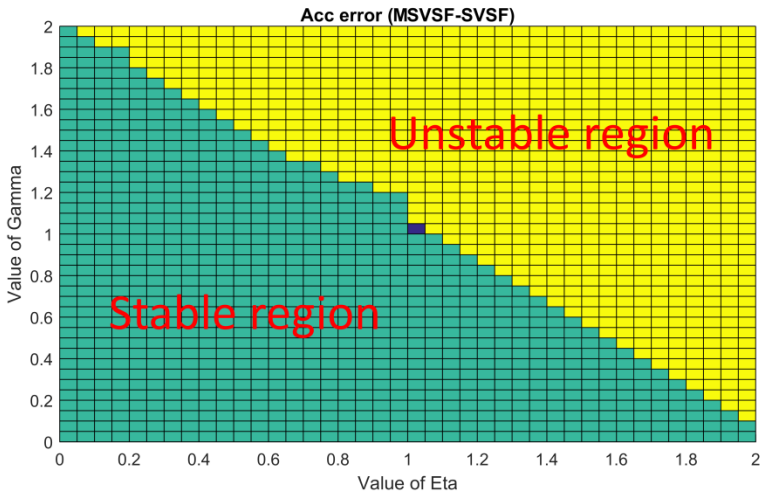


Figure 4-10 Acceleration error difference of SVSF and Modified-SVSF XY-view

Another positive characteristic of the Modified-SVSF is that it inherits SVSF's ability to reject modeling uncertainties. In the simulations, a modeling uncertainty in the system matrix (A) is introduced. In order to show the resilience of the SVSF and the Modified-SVSF to uncertainties in modeling, they are compared with the Kalman Filter (KF). Figure 4-11 shows the position estimates for the first second of the simulation and

Figure 4-12 show the velocity estimates. KF parameters are the same as those in [3], $Q = \begin{bmatrix} 1 & & \\ & 10 & \\ & & 100 \end{bmatrix}, R = \begin{bmatrix} 0.1 & & \\ & 100 & \\ & & 100 \end{bmatrix}$, and $P = 10Q$. In Figures 15 and 16, it can be seen that both the SVSF and the Modified-SVSF are more robust when the model is changed (t= 0.5 seconds), whereas the KF loses its estimation accuracy.

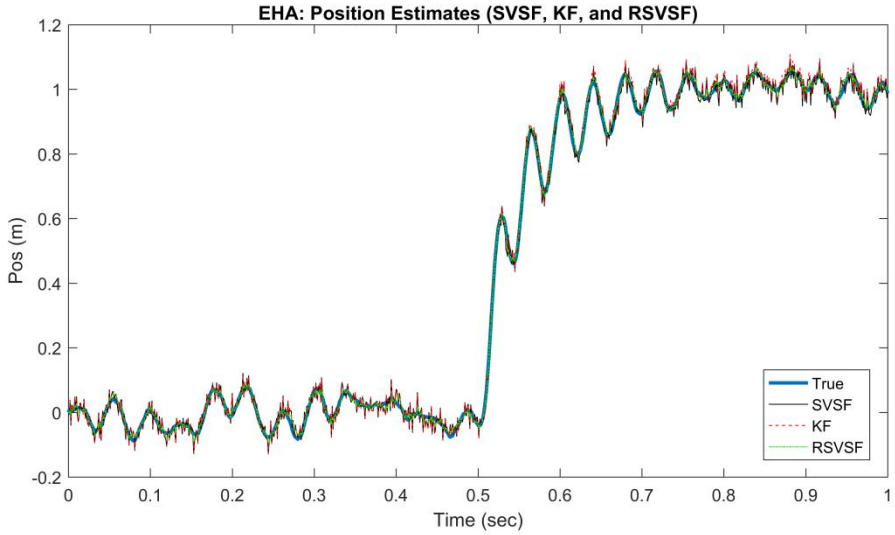


Figure 4-11 Transient estimation performance for position

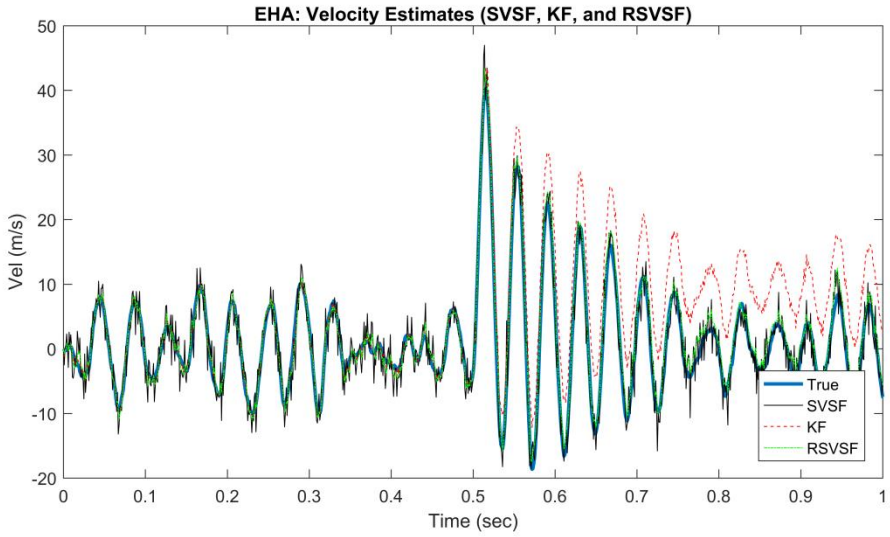


Figure 4-12 Transient estimation performance for velocity

This robustness to modeling uncertainties is also reflected in the estimation RMSE values. Table 4-III shows a comparison between the KF, the SVSF and the Modified-SVSF and their RMSE for position and velocity estimates for the total simulation time (50 seconds).

Table 4-III RMSE values for position, velocity, and acceleration values

Filter	Position RMSE	Velocity RMSE
KF	0.023923	9.0835
SVSF	0.022027	2.2498
Modified-SVSF	**0.010214**	**1.4581**

Modified-SVSF reduced the position RMSE by 53.6% and velocity RMSE by 35.2% compared with SVSF. Comparing Modified-SVSF with the KF, Modified-SVSF has reduced the position RMSE by 57.3% and velocity RMSE by 83.95%. The results show the potential of Modified-SVSF in achieving better filtration performance.

4.4 Conclusion

In this paper an extension for SVSF, called Modified-SVSF is proposed. The stability of the proposed algorithm is proven and an experimental simulation is conducted to compare the Modified-SVSF performance to the original form of SVSF.

The results show that the Modified-SVSF can perform better than the original SVSF without compromising speed or memory. Both the Modified-SVSF and the SVSF are able to be robust to modeling uncertainties which is important when dealing with systems that change over time. The addition of the a priori error scaling parameter η introduces another dimension of fine tuning to the filter. It also allows for a bigger stability margin than what is available in the original SVSF. The Modified SVSF also was shown to be better than Kalman Filter (KF) and SVSF in RMSE values.

Acknowledgement

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), funding reference number CRDPJ 486107-15.

References

[1] J. O'Reilly, *Observers for linear systems* vol. 170: Academic Press, 1983.

[2] M. S. Grewal, "Kalman filtering," in *International Encyclopedia of Statistical Science*, ed: Springer, 2011, pp. 705-708.

[3] S. Habibi, "The Smooth Variable Structure Filter," *Proceedings of the IEEE,* vol. 95, pp. 1026-1059, 2007.

[4] S. Gadsden and S. Habibi, "State Estimation and Fault Detection of an Electrohydrostatic Actuator," in *ASME/BATH 2014 Symposium on Fluid Power and Motion Control*, 2014, pp. V001T01A032-V001T01A032.

[5] D. G. Luenberger, "Observing the state of a linear system," *IEEE transactions on military electronics,* vol. 8, pp. 74-80, 1964.

[6] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering,* vol. 82, pp. 35-45, 1960.

[7] R. E. Kalman, "Contributions to the theory of optimal control," *Bol. Soc. Mat. Mexicana,* vol. 5, pp. 102-119, 1960.

[8] R. E. Kalman and R. S. Bucy, "New results in linear filtering and prediction theory," *Journal of basic engineering,* vol. 83, pp. 95-108, 1961.

[9] S. J. Julier and J. K. Uhlmann, "New extension of the Kalman filter to nonlinear systems," in *Signal processing, sensor fusion, and target recognition VI*, 1997, pp. 182-194.

[10] E. A. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, 2000, pp. 153-158.

[11] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE,* vol. 92, pp. 401-422, 2004.

[12] Y. Chinniah, R. Burton, S. Habibi, and E. Sampson, "Identification of the nonlinear friction characteristics in a hydraulic actuator using the extended Kalman filter," *Transactions of the Canadian Society for Mechanical Engineering,* vol. 32, pp. 121-136, 2008.

[13] S. Blackman, R. Dempster, M. Busch, and R. Popoli, "IMM/MHT solution to radar benchmark tracking problem," *IEEE Transactions on Aerospace and Electronic Systems,* vol. 35, pp. 730-738, 1999.

[14] H. Qu, L. Pang, and S. Li, "A novel interacting multiple model algorithm," *Signal Processing,* vol. 89, pp. 2171-2177, 2009.

[15] S. Habibi, "Performance measures of the variable structure filter," *Transactions of the Canadian Society for Mechanical Engineering,* vol. 29, pp. 267-295, 2005.

[16] S. K. Spurgeon, "Sliding mode observers: a survey," *International Journal of Systems Science,* vol. 39, pp. 751-764, 2008.

[17] S. Wang, S. Habibi, and R. Burton, "A comparative study of a smooth variable structure filter and the extended Kalman filter," *Transactions of the Canadian Society for Mechanical Engineering,* vol. 32, pp. 353-370, 2008.

[18] T. Kim, Y. Wang, Z. Sahinoglu, T. Wada, S. Hara, and W. Qiao, "State of charge estimation based on a realtime battery model and iterative smooth variable structure filter," in *Innovative Smart Grid Technologies-Asia (ISGT Asia), 2014 IEEE*, 2014, pp. 132-137.

[19] M. S. Farag, R. Ahmed, S. Gadsden, S. Habibi, and J. Tjong, "A comparative study of Li-ion battery models and nonlinear dual estimation strategies," in *Transportation Electrification Conference and Expo (ITEC), 2012 IEEE*, 2012, pp. 1-8.

[20] M. Attari, S. A. Gadsden, and S. R. Habibi, "Target tracking formulation of the SVSF as a probabilistic data association algorithm," in *American Control Conference (ACC), 2013*, 2013, pp. 6328-6332.

[21] S. Gadsden, D. Dunne, S. Habibi, and T. Kirubarajan, "Comparison of extended and unscented Kalman, particle, and smooth variable structure filters on a bearing-only target tracking problem," in *Signal and Data Processing of Small Targets 2009*, 2009, p. 74450B.

[22] S. A. Gadsden, S. R. Habibi, and T. Kirubarajan, "A novel interacting multiple model method for nonlinear target tracking," in *Information Fusion (FUSION), 2010 13th Conference on*, 2010, pp. 1-8.

[23] R. Ahmed, "Training of Neural Networks Using the Smooth Variable Structure Filter with Application to Fault Detection," Department of Mechanical Engineering, McMaster University, April 2011.

[24] H. Asada and J.-J. Slotine, *Robot analysis and control*: John Wiley & Sons, 1986.

[25] S. Habibi and A. Goldenberg, "Design of a new high performance electrohydraulic actuator," in *Advanced Intelligent Mechatronics, 1999. Proceedings. 1999 IEEE/ASME International Conference on*, 1999, pp. 227-232.

[26] S. Habibi, R. Burton, and E. Sampson, "High precision hydrostatic actuation systems for micro-and nanomanipulation of heavy loads," *Journal of dynamic systems, measurement, and control,* vol. 128, pp. 778-787, 2006.

[27] Y. Chinniah, S. Habibi, R. Burton, and E. Sampson, "Parameter identification in a high performance hydrostatic actuation system using the Unscented Kalman filter," *Transactions of the Canadian Society for Mechanical Engineering,* vol. 30, pp. 375-390, 2006.

Chapter 5: Quality Control Using Deep Learning for Automotive Starters

Mahmoud Ismail[a,*], and Saeid Habibi[a]

[a] Department of Mechanical Engineering, McMaster University, Hamilton, Canada;

[*]**Corresponding author**: Mahmoud Ismail, Department of Mechanical Engineering, McMaster University, Hamilton, Canada.

This journal paper is submitted to ASME Journal of Manufacturing Science and Engineering in November 2018.

Abstract

A Fault Detection and Diagnosis (FDD) system is proposed that uses Deep-Learning algorithms applied on sound and vibration measurements of starter motors. Deep-Learning (DL) is increasingly used in machine learning in many fields and has achieved unprecedented results in both image and speech recognition. The proposed FDD system is verified on 224 measurements of parts that represent four product conditions (1 healthy and 3 fault conditions). It is found that using vibration and sound measurements together can achieve a very high success rate (97.68%) in fault diagnosis compared to using sound or vibration signals individually. This high level of performance shows the potential of the proposed DL algorithm as a self-learning technique to be used for quality assurance in automotive manufacturing industry.

Keywords

Deep Learning, Quality Assurance, Neural Networks, Fault Detection and Diagnosis, Machine Learning.

5.1 Introduction

The use of Deep-Learning (DL) in academic and industrial applications is rapidly increasing because of its unprecedented performance compared to other machine learning techniques. The most common applications for DL are image and speech recognition. DL has also been applied to many other fields such as big data analytics [1], finance [2], language modeling, natural language processing, information retrieval [3], and classification of skin cancer [4]. One of the areas being investigated for DL is Fault Detection and Diagnosis (FDD). FDD algorithms are the corner stone of quality assurance in industry. The concept of applying DL on FDD applications will put in use the advantages of DL performance to detect and diagnose faults with better success rates.

Quality assurance of car starters, alternators, and motors is highly valued due to the resulting safety problems and cost consequences of faulty products. These products are commonly tested according to different mechanical and electrical measurements, such as torque, current, voltage, and vibrations. This paper reports on the use of sound and vibration measurements for FDD since:

1) Faults in electromechanical systems invariably result in different vibration and sound characteristics compared to the normal baseline characteristics.

2) A change in sound and vibration levels, even if not related to a serious fault condition, can be irritating to consumers and impact the perceived quality of the final product.

The main challenges in testing electromechanical components using sound and vibration are:

1) Traditional tools such as Fast Fourier Transform (FFT) need deep knowledge and expertise of sound and vibration signals to be used as FDD systems. Sound and vibration analysis using these tools are done manually.

2) FDD in complex systems are normally signal based and implemented by comparing measurements from faulty versus healthy products. The difference is captured by looking at complicated charts by a trained expert.

Deep Learning in essence solves these two problems as it automates this comparison and does not generate complicated charts that require an expert to interpret. Therefore, DL substitutes for the requirements of in depth technical expertise, be it related to the product or its signal analysis.

This paper demonstrates the importance of sound and vibration measurements in FDD of electromechanical systems. The paper presents the following:

1) The ability of DL to extract fault information from complex sound and vibration signals and use that for a machine condition monitoring system.

2) A comparison between Deep Neural Networks (DNN) and Convolutional Neural Networks (CNN) on the FDD application.

3) A comparative study that shows the importance of using a combination of sound and vibration measurement to capture different kinds of faults.

4) The effect of CNN pooling layer spatial invariance on the FDD system accuracy.

The paper is organized as follows. Section II provides a literature review on FDD and DL systems. Section III discusses the use of DL as a FDD system. Section IV presents the

results of applying a DL-FDD system to automotive starters. The last Section summarizes the conclusions of this study.

5.2 Literature Review

5.2.1 Fault Detection and Diagnosis

Fault Detection and Diagnosis (FDD) is an important aspect in all industries to assure quality, reduce operating costs and improve the reputation of a company [5]. Venkatasubramanian et al. [6-8] reviewed a wide spectrum of FDD systems and summarized them into two categories:

1) Model-Based FDD systems, and

2) Signal-Based FDD systems.

The former is applicable when systems can be modeled mathematically. Signal-based FDD systems are used to extract patterns in the signals and search for any signs of faults presence.

Automotive electromechanical components such as motor, starters and alternators are relatively complex machines, and therefore building a mathematical model that can predict accurately their vibration and sound characteristics is difficult. For this case, signal-based FDD systems are preferred. The most simplistic signal-based FDD system is setting a threshold on the sound vibration levels. However, some faults change sound characteristics without necessarily increasing loudness and, the same applies to vibration. Therefore, these simplistic approaches are not the best signal based FDD systems. Also, it was found that the capability of detection and diagnosing fault conditions is greatly affected by the number and the type of sensors [9]. However, when a fault occurs, usually the fault affects

measurements from several sensors. Therefore, researchers tried to combine the information from all sensors for better detection. To extract the information from multiple sensors, multivariate statistics can be used. Principle Components Analysis (PCA) is an efficient multivariate. General applications of PCA for FDD have been successful and many researchers have used PCA to tackle practical fault monitoring challenges in chemical industries [10, 11].

However, PCA does not provide any information about the fault behavior, and needs to be complemented with other algorithms such as wavelets transformation or Fast Fourier Transformation (FFT) to capture the frequency components of the fault. Wavelets [12-17] have the ability to analyze signals in both time and frequency domains simultaneously. In [9], an FDD system that combines PCA, Wavelets, background noise filtration, and classification was developed to automatically detect and diagnose faults.

The frequency components of rotating machinery faults are important as different faults have different frequency characteristics. For example, a survey of time, frequency, and time-frequency characteristics of a broken rotor bar motor was presented in [18]. In this survey, it is shown how frequency peaks shift depending on the slip factor. Many other publications show how faults, such as bearing faults, can be detected using FFT or other frequency transformations such as wavelets [19-22]. However, an expert in that field is required to interpret the results.

To summarize, the system required for reliable detection and diagnosis of faults should possess the following features:

1) Be able to extract meaningful information about faults from measured signals from multiple sensors.

2) Have the ability of combining and maximizing the information from the inputs of multiple sensors into one conclusion.

3) Use the frequency characteristics of faults to differentiate between them.

5.2.2 Deep-Learning

Deep Learning is a relatively new topic in the academic and industrial fields. It evolved from Neural Networks, and has proved to be considerably more powerful. There are different kinds and architectures of DL and each architecture excels in a different field. For example, Convolutional Neural Networks (CNN) excel in image recognition problems while Long-Short Term Memory (LSTM) excels in tasks such as speech recognition and natural language processing due to its sequence understanding capability [23].

Two architectures are used in this paper, the first is Convolutional Neural Network (CNN) which is considered the state-of-the-art in image recognition, and the second is a Vanilla DNN network which can be considered as a scaled up Feedforward Neural Network (FNN). These two specific architectures were inspired by the advancement in the speech recognition research. Fast Fourier Transform is used to provide spectra for analyzing sound before using deep learning. Research in [24] showed that using Mel-scale log-filter bank features achieved higher accuracy than commonly used Mel-Frequency Cepstral Coefficients (MFCC). In [25] spectrograms were used to analyze the frequency components of a speech to classify its emotion status. In [26], it was mentioned that deep learning techniques were able to extract more information from the spectral information

because of DL's high level feature extraction ability. Spectrograms provide the frequency content of sound signals and can be generated based on Fast Fourier Transform (FFT). FFT however assumes that the frequency content of signals is stationery and does not change with time. The Short Time Fourier Transform (STFT) alleviates this problem. The STFT (i.e. spectrograms) recognizes the time dependence of testing and is used extensively in conjunction with Deep Learning in speech recognition problems.

In [27, 28] Deep Neural Network (DNN) and Convolutional Neural Network (CNN) are compared and both of them are applied to the speech recognition problems. In both publications, the first processing step is transforming the speech to frequency bands at different times. This frequency-time map is afterwards used as an input for the DNN and CNN networks.

DNN's are simply Feedforward Neural Networks (FNN) with more than one hidden layer. They are formed by fully connected layers as shown in Figure 5-1.

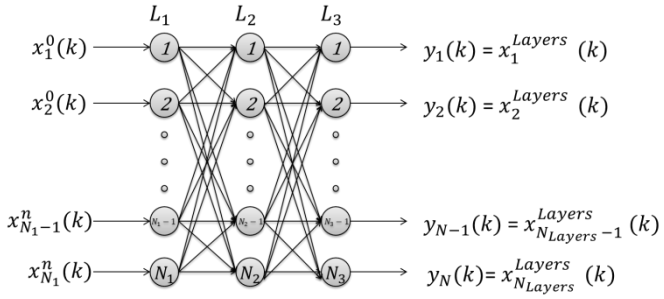


Figure 5-1. One hidden layer Feedfoward Neural Network

Convolutional Neural Networks (CNN's) are different from DNN's in how they work and in their architecture. CNNs usually have three kinds of layers:

1) Convolutional layers.

2) Pooling layers.

3) Fully connected layers.

The pooling layers are introduced for dimensionality reduction as well as spatial invariance. A typical structure of a CNN network is presented in Figure 5-2.

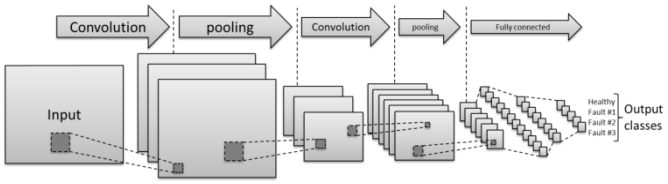


Figure 5-2. Typical structure of a CNN network

5.3 Deep-Learning Based FDD

Neural Networks have been used for different FDD applications. For example, in [29] it was used for engine fault detection. However, the potential of applying DL in FDD applications has not been extensively explored yet. DL is an extension of NN that uses many hidden layers, enabling more feature extraction power. This ability of feature extraction is important in FDD applications.

Another important aspect of using DL for FDD is the fusion of the information from different sensor types. This is an important feature required of FDD systems, as different types of faults can be captured using different physical measurements. Such a system can be expanded to include more sensors which allows DL to study even more complex fault conditions. In other words, it allows the FDD system to be scalable.

5.3.1 Architecture of a DL Based FDD Using Sound and Vibration Signals

Both architectures, CNN and DNN, are studied in this paper. Due to the spatial invariance characteristics of both convolutional and pooling layers, CNN network tends to be more

effective in some applications over others. Spatial invariance is very favorable in tasks such as image recognition where the algorithm is required to detect the image even if it was shifted or rotated. In speech recognition [27], it is argued that one of the improvements that CNN achieved over DNN is attributed to these invariances as slight formant shifts, due to the speaker or the speaking style, will not affect the result. However, in Fault Detection and Diagnosis, spatial invariance can be harmful as slight changes in frequency can be an indicator of a fault condition as discussed in [18] for a broken bar fault in an induction motor. Therefore, both architectures are tested. The advantage of CNN is using the shared parameters (layer weights) for the convolution layers. This reduces significantly the memory required, although it takes a longer time for each epoch because of the convolution process. On the other hand, a DNN network is simple to use and can be argued to be more suitable for this study as the starters test profile is performed using a fixed rotational speed (stationary sound and vibration signals). In this example, therefore, there is no spatial correlation required, which translates to time dependency in this case. Thus, if the vibration and sound measurements are assumed to be stationary, then a simple DNN could perform well. For both architectures, spectrum information is used as inputs to reflect the frequency content of the measurement.

For the DNN network, three cases are studied. The architecture is the same for all the three cases; the difference is in the input of the network. The first case involved studying the FDD performance using sound signals only; the second case using vibration signal only; and the third case is by using both. The preprocessing step of the input is shown in Figure 5-3 and the DNN architecture is shown in Figure 5-4 (a). The spectrums generated

in Figure 5-3 are normalized from 0 to 1 which allows the network to look at the characteristics of the spectrum rather than its absolute values.

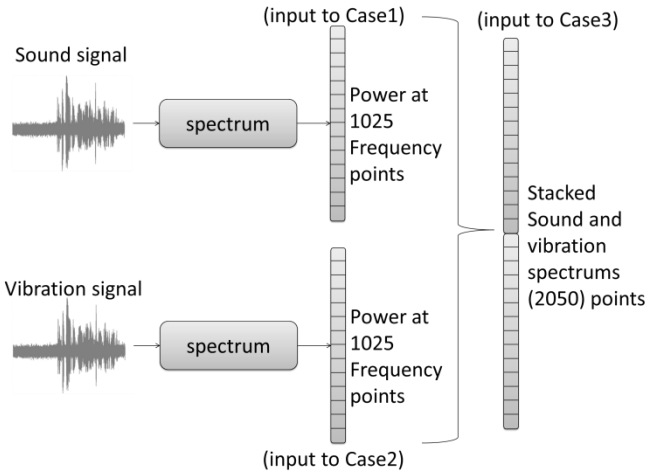


Figure 5-3. DNN sound and vibration preprocessing

For the CNN network, the architecture is different. Four cases are studied. The first three are similar to the DNN (where the inputs are sound, vibration, and sound and vibration together). The fourth case was conducted to evaluate the effect of introducing a pooling layer to the network.

The preprocessing for CNN network is presented in Figure 5-5, and the CNN architecture that was used is presented in Figure 5-4 (b,c). The kernel size for the convolution layer was (64,1) for the single signal cases and (62,2) when the sound and vibration signals were processed together.

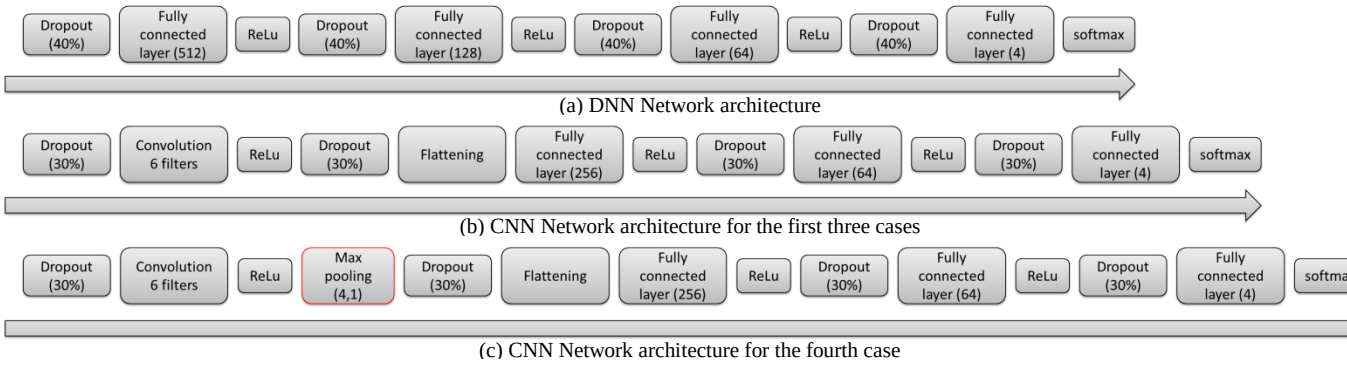


Figure 5-4. The used architectures in the case study

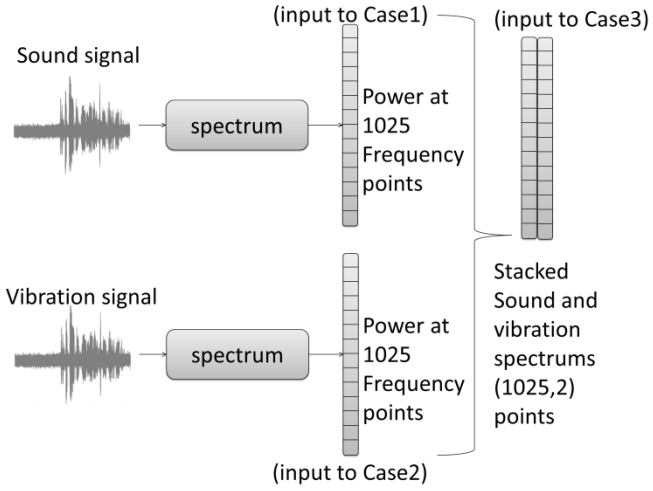


Figure 5-5. CNN sound and vibration preprocessing

In the training stage, Adam [30] optimization technique is used with MSE as a loss function. For both network types (DNN and CNN), the architectures in Figure 5-4 utilize Rectified Linear Units (ReLu) and softmax as activation functions. These activation functions are described by Equations (5-1) and (5-2).

$$Relu(x) = \max(0, x) \tag{5-1}$$

$$softmax_j\,(x) = \frac{\exp(x_j)}{\sum_{k=1}^{K} \exp(x_k)}, \text{ where } j = 1, 2, \ldots, K \tag{5-2}$$

5.4 Application of the DL-FDD on Automotive Starters

To evaluate the performance of DL as a FDD system, a case study is performed on automotive starters. In this case study, the target is to correctly detect and diagnose automotive starters' faults. This application was considered for its commercial implementation in end-of-line testers that suppliers use for a final quality check before shipping starters to car manufacturers. These measurements are collected using an end-of-line tester provided by D&V Electronics. The tester is shown in Appendix B.

5.4.1 Dataset and Testing Method

The studied conditions consist of four different categories:

- Healthy starter: a good starter that has no faults.
- Fault #1: Armature imbalance.
- Fault#2: Brush/commutator connection problems.
- Fault #3: pinion gear problem.

To ensure that the FDD strategies are robust, repeatable and that there is no correlation between the training and testing data sets, the system is tested as follows:

1) 20 parts were collected with the conditions listed in Table 5-I.

2) Sound and vibration measurements are collected from the abovementioned parts several times for each part, to confirm the repeatability of the results.

3) To test the 20 parts measurements the following steps are followed:

a) Select one part (part x) to test

b) Train the network using measurements from the other 19 parts

c) Test the measurements of part x

d) Repeat points (a) to (c) for the 20 parts

e) Sum the results in a confusion matrix

Training the networks on 19 parts and then testing the remaining one part simulates the real world situation where the End-Of-Line (EOL) tester is required to test a part that is newly manufactured and was not included in the training dataset. Pinion gear fault condition is the only exception to the mentioned methodology as it is a very rare case; only one part with this fault was found and it was tested 10 times. To evaluate the FDD system in diagnosing this fault, the system is trained on five measurements and tested using the other five measurements.

During the training, the performance of the resulted model of every training epoch is evaluated. The model that performed the best is called the Best Model (BM). This is the model that is used in the testing step. The workflow of selecting the Best Model (BM) is shown in Appendix A.

Table 5-I CASE STUDY DATASET BREAKDOWN

Starter condition	Number of parts	Number of measurements per part on average	Total number of measurements
Healthy	10	12	121
Fault #1: Armature imbalance	6	10	59
Fault#2: Brush commutator connection problems	3	10	34
Fault #3: pinion gear problem	1	10	10
Total	20		224

5.4.2 Results and Discussion

In this section the results are displayed in a confusion matrix form. There are three statistical measures to evaluate the overall system performance. For n conditions, k parts, and m measurements, the performance measures are:

1) *"Success rate per class"*, which it is defined as follows:

$$SR_{class} = \frac{1}{n}\sum_{i=1}^{i=n} success\ rate\ of\ condition\ (i) \quad (5\text{-}3)$$

2) *"Success rate per part"*, defined by:

$$SR_{part} = \frac{1}{k}\sum_{i=1}^{i=k} success\ rate\ of\ part\ (i) \quad (5\text{-}4)$$

3) *"Success rate per individual measurement"*, which is defined as follows:

$$SR_{individual\ test} = \frac{1}{m}\sum_{i=1}^{i=m} success\ rate\ of\ test\ (i) \quad (5\text{-}5)$$

The previous equations look similar but they measure different qualities of the FDD system performance. The first indicator measures the ability of the FDD system to diagnose different classes. Therefore, if one of the classes is not being detectable, this indicator will be low even if it is represented by a few samples. The second indicator measures the ability of the FDD system to diagnose individual parts, therefore confirms the repeatability of the results for the same part. The last indicator gives an overall success rate for the whole system. In this case study, n = 4 (*conditions*), k = 20 (*parts*), m= 219 (*starter measurements*).

5.4.2.1 Using Sound Only

Using sound only as an input to the DL system, it was found that both architectures were able to capture fault #3 very well, however DNN did better in the healthy, and faults #1and #2 conditions. The results are presented in Figure 5-6 and Figure 5-7

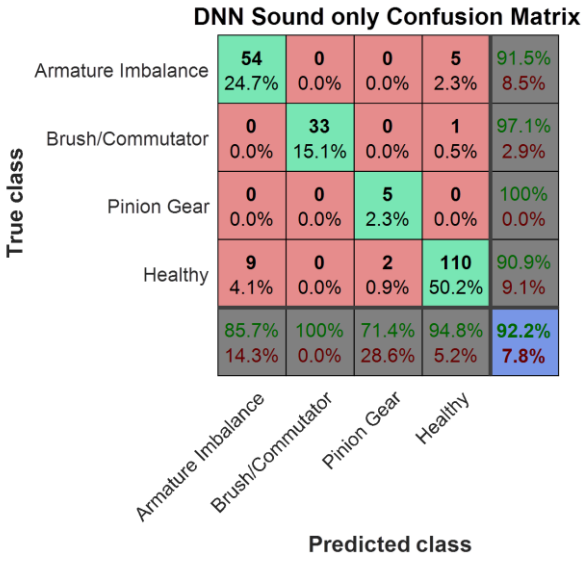


Figure 5-6. DNN results using sound only, SR_{class} = 94.8%, SR_{part} = 91.9%, $SR_{individual\ test}$ = 92.2%

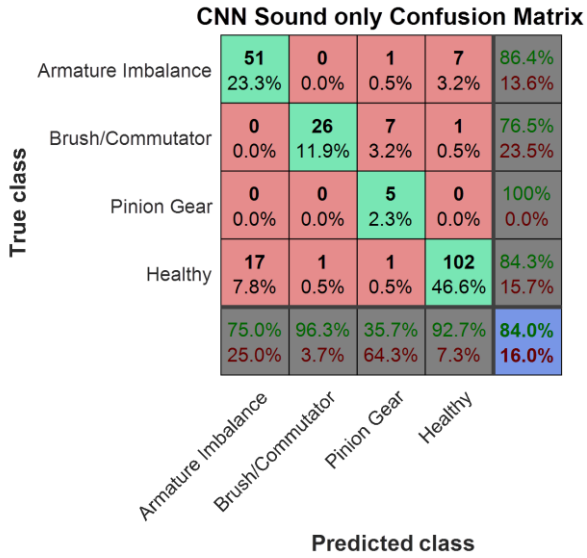


Figure 5-7. CNN results using sound only, SR_{class} = 86.8%, SR_{part} = 83.9%, $SR_{individual\ test}$ = 84.0%

All the three indicators are in favor of the DNN network.

5.4.2.2 Using Vibration Only

Applying DL using vibration measurements only, the DNN network does not do very well when it comes to the third fault which is the pinion gear problem. However the DNN network captures well the second fault, which the CNN network fails to capture with high accuracy.

These results highlight the importance of the adaptability of the network. Some faults result in a change in the whole spectrum and some faults generate smaller harmonics around the fundamental frequencies. CNN networks are more powerful in picking up local repeating patterns.

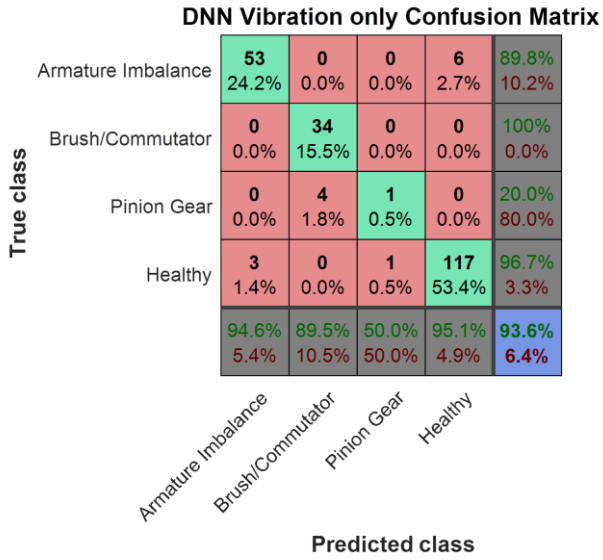


Figure 5-8. DNN results using vibration only, SR_{class} = 76.6%, SR_{part} = 90.9%, $SR_{individual\ test}$= 93.6%

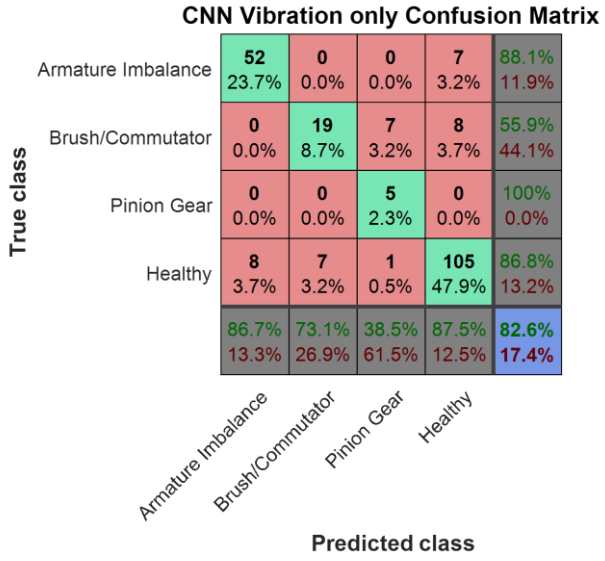


Figure 5-9. CNN results using vibration only, SR_{class} = 82.6%, SR_{part} = 82.9%, $SR_{individual\ test}$= 82.6%

The previous results show the DNN network has higher SR_{part} and $SR_{individual\ test}$, however it recorded lower on the SR_{class}. These examples shows that DNN network

delivers better overall performance in fault detection but it scores low in fault diagnosis according to its lack of ability to diagnose the pinion gear problem.

5.4.2.3 Using Sound and Vibration together

In this test both sound and vibration signals are combined and used for the FDD system. The hypothesis in this test is that combining both signals should increase all indicators and performance measures as the networks should be able to combine the extracted features from both signals and maximize the success rate of fault detection and diagnosis.

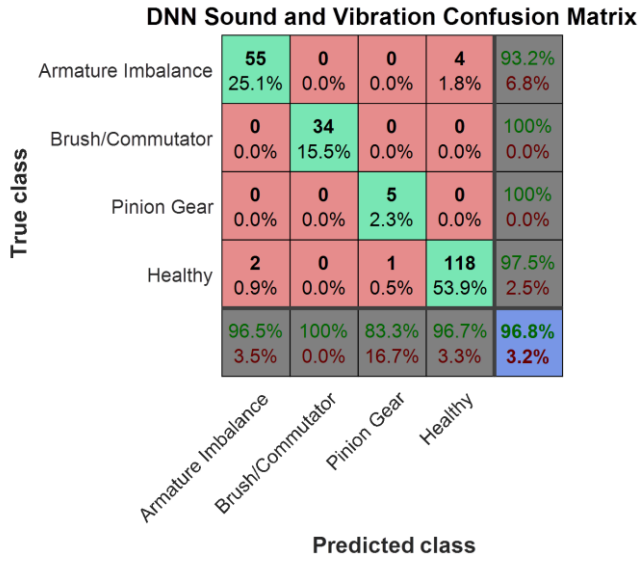


Figure 5-10. DNN results using sound and vibration, SR_{class} = 97.6%, SR_{part} = 96.4%, $SR_{individual\ test}$ = 96.8%

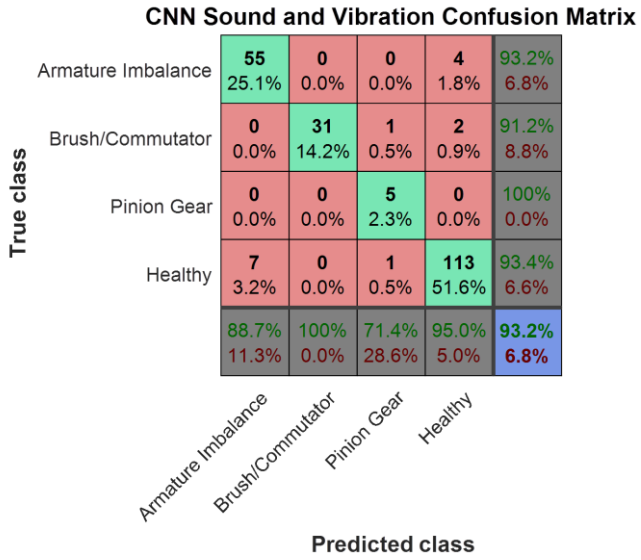


Figure 5-11. CNN results using sound and vibration, SR_{class} = 94.4%, SR_{part}= 92.8%, $SR_{individual\ test}$= 93.1%

From Figures 15 and 16, it can be seen that the hypothesis is validated and all performance indicators increased in both network architectures. DNN using both signals still achieves higher success rates compared to the CNN network.

5.4.2.4 Pooling layer Effect in CNN

In this test a comparison between two CNN networks was conducted. The first network does not use a max pooling layer, and the second network has a max pooling layer after the convolution layer as shown in Figure 5-4 (c). The results without the pooling layer are presented in Figure 5-11 and the results with a pooling layer are presented in Figure 5-12. These results show that all performance indicators deteriorate upon using the max pooling layer.

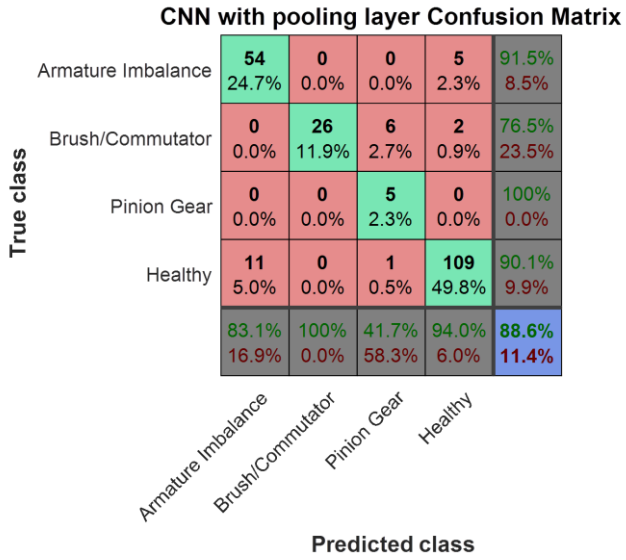


Figure 5-12. CNN results using sound and vibration with an extra pooling layer, SR_{class}= 89.5%, SR_{part}= 88.3%, $SR_{individual\ test}$= 88.5%

5.4.2.5 Discussion of Results

The tests show the difference of the CNN versus the DNN networks and their ability to detect different faults. CNN could detect the pinion gear fault (using either sound only or vibration only or both) while DNN captured it using (sound only or both). The failure of the DNN to capture the pinion gear fault using the vibration signal can be attributed to the fact that pinion gear problem create harmonics in the vibration signal that are better detected using the same filter convoluted over the spectrum. However, the other conditions (faults #1, #2, and healthy) more often exhibit themselves over specific frequency regions and therefore DNN is better as it was able to achieve higher success rates using (sound only, vibration only or both). Especially for fault #2 (brush/commutator), DNN performs better than CNN. This proves that different architectures excel at different fault types.

Also the signal type affects the results. For example, it was shown that DNN can capture fault #3 in sound more robustly than in vibration. Also, in all cases of DNN and CNN, when sound and vibration are combined, higher success rates are achievable. This proves that additional information can be extracted to achieve higher success rates in FDD systems using more sensors.

Lastly, the addition of pooling layers in the CNN network has proven to worsen the results almost in every condition. This deterioration is caused by the translation invariance effect. This spatial invariance adds an ambiguity to what are the characteristics of the signal spectrum and therefore it is not very useful to this kind of FDD application.

The summary of the results is presented Figure 5-13. DNN using sound and vibration measurements achieves the best results. Excellent performance is also achieved by CNN using sound and vibration measurements as well.

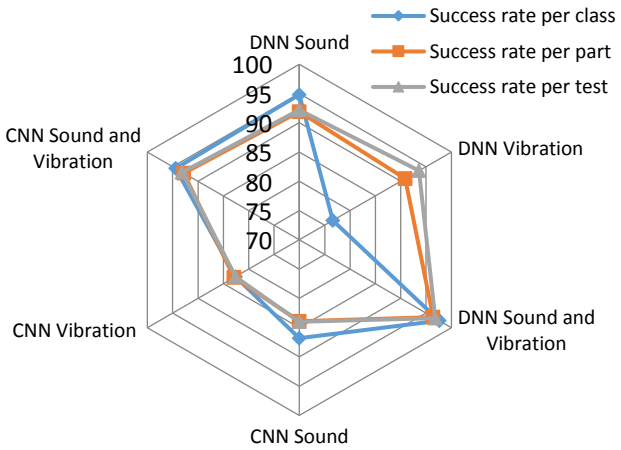


Figure 5-13. Summary of the case study results

5.5 Conclusions

In this paper, a Fault Detection and Diagnosis algorithm based on Deep Learning is introduced. This algorithm is tested on automotive starters and was able to detect the faults and diagnose them successfully. This case study used one vibration signal and one acoustic signal. It was shown that the combination of the information in sound and vibration signals increase the success rate of detection and diagnosis using deep learning abilities to harness information from more data. This suggests that the use of even more signals may achieve higher success rates. A comparison between CNN and DNN networks was provided and it was shown that the DNN performs better across the different input types. It was also shown that the invariance in CNN can introduce FDD ambiguity and harm the overall FDD system performance.

Acknowledgement

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), funding reference number CRDPJ 486107-15.

References

[1] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *Journal of Big Data,* vol. 2, p. 1, 2015.

[2] J. Heaton, N. Polson, and J. H. Witte, "Deep learning for finance: deep portfolios," *Applied Stochastic Models in Business and Industry,* vol. 33, pp. 3-12, 2017.

[3] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA Transactions on Signal and Information Processing,* vol. 3, 2014.

[4] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau*, et al.*, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature,* vol. 542, pp. 115-118, 2017.

[5] D. A. Aaker, *Building strong brands*: Simon and Schuster, 2012.

[6] V. Venkatasubramanian, R. Rengaswamy, K. Yin, and S. N. Kavuri, "A review of process fault detection and diagnosis: Part I: Quantitative model-based methods," *Computers & chemical engineering,* vol. 27, pp. 293-311, 2003.

[7] V. Venkatasubramanian, R. Rengaswamy, and S. N. Kavuri, "A review of process fault detection and diagnosis: Part II: Qualitative models and search strategies," *Computers & Chemical Engineering,* vol. 27, pp. 313-326, 2003.

[8] V. Venkatasubramanian, R. Rengaswamy, S. N. Kavuri, and K. Yin, "A review of process fault detection and diagnosis: Part III: Process history based methods," *Computers & chemical engineering,* vol. 27, pp. 327-346, 2003.

[9] M. Ismail, "Industrial Extended Multi-Scale Principle Components Analysis for Fault Detection and Diagnosis of Car Alternators and Starters,"*Master's thesis, McMaster University* 2015. https://macsphere.mcmaster.ca/.

[10] S. Zanoli, G. Astolfi, and L. Barboni, "FDI of Process Faults based on PCA and Cluster Analysis," in *Control and Fault-Tolerant Systems (SysTol), 2010 Conference on*, 2010, pp. 197-202.

[11] L. H. Chiang, R. D. Braatz, and E. L. Russell, *Fault detection and diagnosis in industrial systems*: Springer Science & Business Media, 2001.

[12] S. Abbasion, A. Rafsanjani, A. Farshidianfar, and N. Irani, "Rolling element bearings multi-fault classification based on the wavelet denoising and support vector machine," *Mechanical Systems and Signal Processing,* vol. 21, pp. 2933-2945, Oct 2007.

[13] A. Barri, A. Dooms, and P. Schelkens, "The near shift-invariance of the dual-tree complex wavelet transform revisited," *Journal of Mathematical Analysis and Applications,* vol. 389, pp. 1303-1314, May 2012.

[14] H. Douglas and P. Pillay, "The impact of wavelet selection on transient motor current signature analysis," in *Electric Machines and Drives, 2005 IEEE International Conference on*, 2005, pp. 80-85.

[15] V. Purushotham, S. Narayanan, and S. A. N. Prasad, "Multi-fault diagnosis of rolling bearing elements using wavelet analysis and hidden Markov model based fault recognition," *Ndt & E International,* vol. 38, pp. 654-664, Dec 2005.

[16] H. Qiu, J. Lee, J. Lin, and G. Yu, "Wavelet filter-based weak signature detection method and its application on rolling element bearing prognostics," *Journal of Sound and Vibration,* vol. 289, pp. 1066-1090, Feb 2006.

[17] X. D. Wang, Y. Y. Zi, and Z. J. He, "Multiwavelet denoising with improved neighboring coefficients for application on rolling bearing fault diagnosis," *Mechanical Systems and Signal Processing,* vol. 25, pp. 285-304, Jan 2011.

[18] V. Ghorbanian and J. Faiz, "A survey on time and frequency characteristics of induction motors with broken rotor bars in line-start and inverter-fed modes," *Mechanical Systems and Signal Processing,* vol. 54, pp. 427-456, 2015.

[19] J. Zarei and J. Poshtan, "Bearing fault detection using wavelet packet transform of induction motor stator current," *Tribology International,* vol. 40, pp. 763-769, 2007.

[20] V. Rai and A. Mohanty, "Bearing fault diagnosis using FFT of intrinsic mode functions in Hilbert–Huang transform," *Mechanical Systems and Signal Processing,* vol. 21, pp. 2607-2615, 2007.

[21] Z. Peng, W. T. Peter, and F. Chu, "A comparison study of improved Hilbert–Huang transform and wavelet transform: application to fault diagnosis for rolling bearing," *Mechanical systems and signal processing,* vol. 19, pp. 974-988, 2005.

[22] H. C. Choe, Y. Wan, and A. K. Chan, "Neural pattern identification of railroad wheel-bearing faults from audible acoustic signals: comparison of FFT, CWT and DWT features," *SPIE Proc. Wavelet App,* vol. 3087, pp. 480-496, 1997.

[23] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks,* vol. 61, pp. 85-117, 2015.

[24] J. Li, D. Yu, J.-T. Huang, and Y. Gong, "Improving wideband speech recognition using mixed-bandwidth training data in CD-DNN-HMM," in *Spoken Language Technology Workshop (SLT), 2012 IEEE*, 2012, pp. 131-136.

[25] A. M. Badshah, J. Ahmad, N. Rahim, and S. W. Baik, "Speech Emotion Recognition from Spectrograms with Deep Convolutional Neural Network," in *Platform Technology and Service (PlatCon), 2017 International Conference on*, 2017, pp. 1-5.

[26] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: An overview," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, 2013, pp. 8599-8603.

[27] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Transactions on audio, speech, and language processing,* vol. 22, pp. 1533-1545, 2014.

[28] J.-T. Huang, J. Li, and Y. Gong, "An analysis of convolutional neural networks for speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, 2015, pp. 4989-4993.

[29] R. Ahmed, "Training of Neural Networks Using the Smooth Variable Structure Filter with Application to Fault Detection," Department of Mechanical Engineering, McMaster University, April 2011.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980,* 2014.

Abbreviations

Abbreviation	Description
FDD	Fault Detection and Diagnosis
DL	Deep Learning
PCA	Principle Components Analysis
CNN	Convolutional Neural Networks
LSTM	Long Short Term Memory
FNN	Feedforward Neural Network
MFCC	Mel-Frequency Cepstral Coefficients
FFT	Fast Fourier Transform
STFT	Short Time Fourier Transform
DNN	Deep Neural Network
MSE	Mean Square Error
BM	Best Model

Appendices

Appendix A: Best Model selection for testing

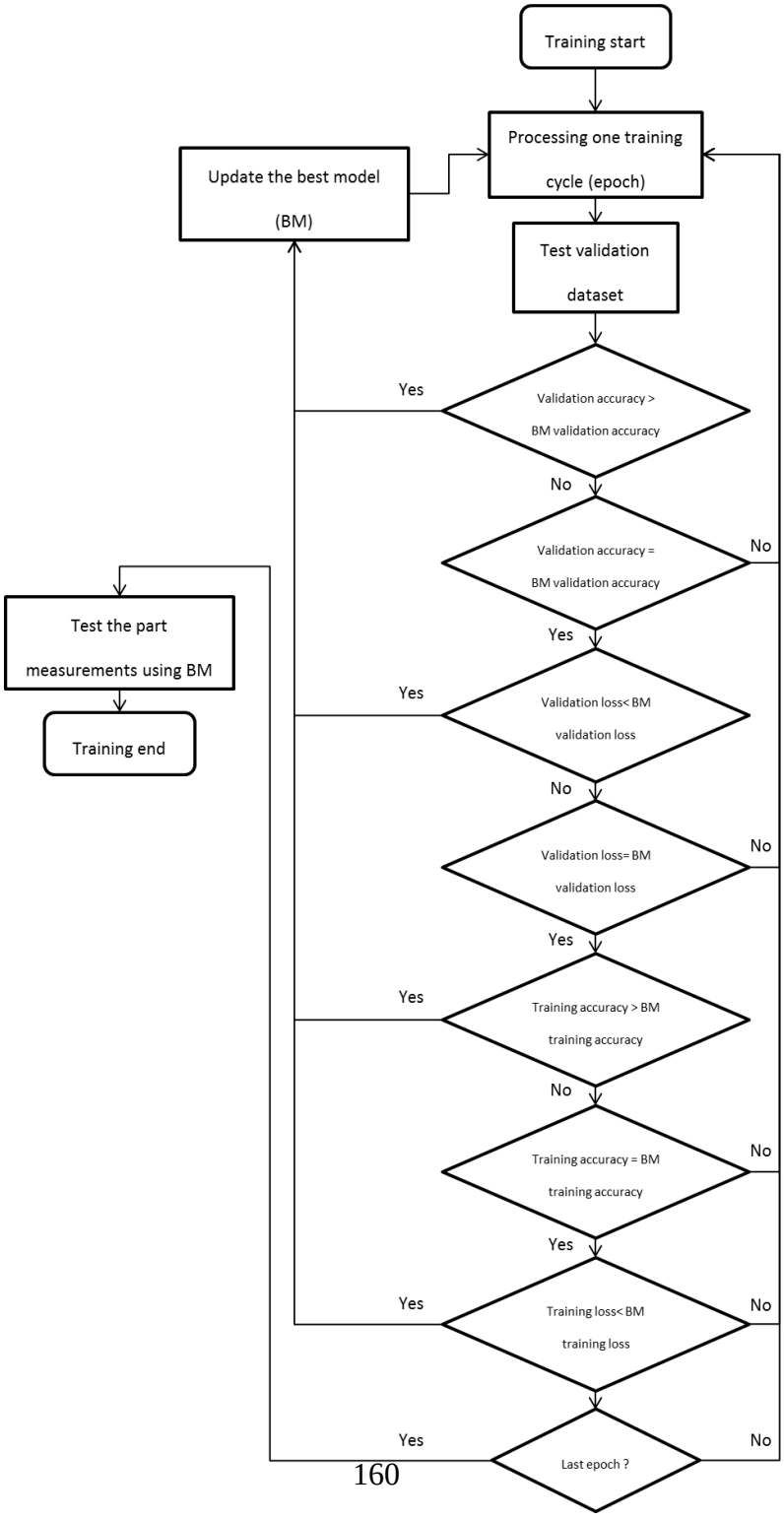


Figure 5-A.1 workflow of selecting the best model

Appendix B: starter tester used

All the measurements were collected using a D&V Electronics starter tester (ST-120). The tester is shown In Figure 5-B.1 .

Figure 5-B.1 ST-120 tester, from D&V Electronics starter testers line

The starters used in this paper are produced by a leading starter supplier. Testing these starters using D&V Electronics machine, the starters are exposed to two cycles as shown in Figure 5-B.2 . The first cycle is a loaded test where the starter is tested against a load that simulates the engine load. The second part of the test is called a free run test, where

the starter is allowed to spin freely. All the sound and vibration signals tested in this paper are done using the free run cycle.

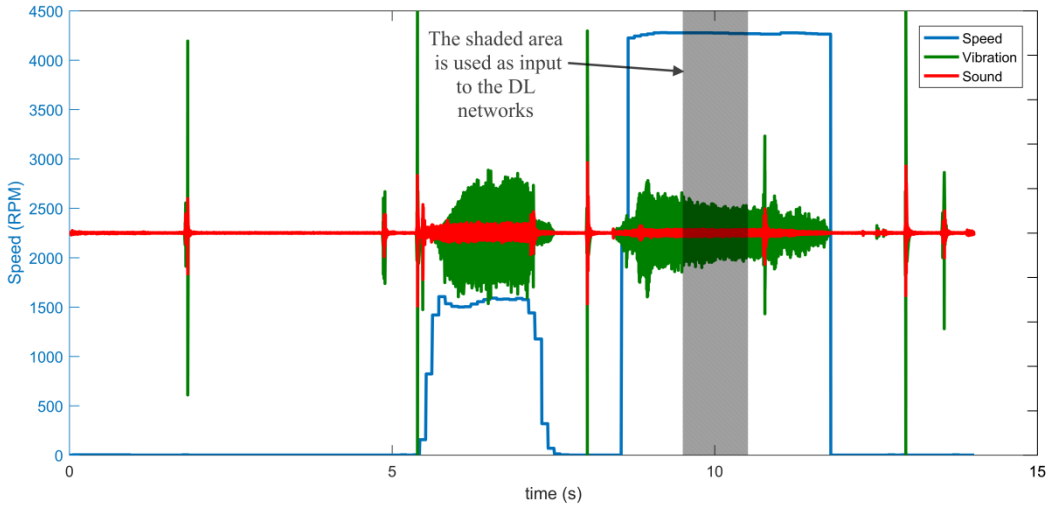


Figure 5-B.2 the test cycle of the starters.

Chapter 6: Battery State of Charge Estimation Using an Artificial Neural Network

Mahmoud Ismail[a], Rioch Dlyma[a], Ahmed Elrakaybi[a], Ryan Ahmed[a], and Saeid Habibi[a]

[a] Department of Mechanical Engineering, McMaster University, Hamilton, Canada;

This paper is published In Transportation Electrification Conference and Expo (ITEC), 2017 IEEE (pp. 342-349). IEEE. DOI: 10.1109/ITEC.2017.7993295. This paper is republished here with permission[4].

Abstract

The automotive industry is currently experiencing a paradigm shift from conventional, diesel and gasoline-propelled vehicles into the second generation hybrid and electric vehicles. Since the battery pack represents the most important and expensive component in the electric vehicle powertrain, extensive monitoring and control is required. Therefore, extensive research is being conducted in the field of electric vehicle battery condition monitoring and control. In this paper, an Artificial Neural Network (ANN) is used for Lithium-Ion (Li-Ion) battery state-of-charge (SOC) estimation. When properly trained using the random current profile described in this paper, a single-layered Neural Network

[4] In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of McMaster's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

is capable of capturing the non-linear characteristics of a battery. The ANN is able to estimate a non-measurable variable such as the battery SOC level based on battery measurable variables such as the voltage and the current. The ANN in this paper is trained using experimental data generated from an experimental battery using a R-RC model with SOC and Open Circuite Voltage (OCV) relationship. The SOC/OCV relationship was derived from a commercial 3.6V 3.4Ah Lithium-ion (Li-Ion) battery cell. The network is trained using current, and voltage as inputs and SOC as the output. The trained network is tested using benchmark driving cycles to be capable of estimating the battery SOC with a relatively high degree of accuracy.

6.1 Introduction

Electric Vehicles (EVs) and Hybrid Electric Vehicles (HEVs) demand is growing due to the global concerns related to fossil fuels and emissions [1, 2]. In order to successfully replace conventional internal combustion engine vehicles with electric vehicles, the latter should be as reliable and cost efficient as the former. At present, the main challenge for the wide acceptance of electric vehicles lies mainly in their initial cost due to using battery systems, which represent the most expensive component. Manufacturers tend to over design the battery system in EVs to compensate for any uncertainties in battery system modeling. Therefore improving battery system modeling accuracy results in smaller pack size, which in turn directly lowers the system's cost and consequently, the vehicles' cost [3, 4].

The Battery Management System (BMS) is responsible for accurately estimating, in real-time, the stored energy in the vehicle batteries. This is a relatively complex task due to many aspects. First, there is no direct measurement method that can directly measure the stored energy in the battery pack. Two methods currently exist for directly evaluating the battery pack SOC from measurable variables such as current and voltage; namely: Coulomb Counting and regressed-voltage based State of Charge – Open Circuit Voltage (SOC-OCV) mapping. The first method is current dependent, in which the stored energy in the battery is calculated based on integrating the charge/discharge current values over time, as shown in Equation (6-1) [5]:

$$SOC = 1 - \frac{\int i.\eta.dt}{C_n} \tag{6-1}$$

Where t is time, i is the current, C_n is the nominal capacity, and η is the coulomb efficiency. This method is prone to errors as it is an open loop estimation system and any small error will accumulate with time due to the integration term. The second method depends on voltage measurements. It uses the OCV to estimate the SOC level. OCV measurements can be mapped to SOC through a nonlinear relationship similar to the one shown in Figure 6-1.

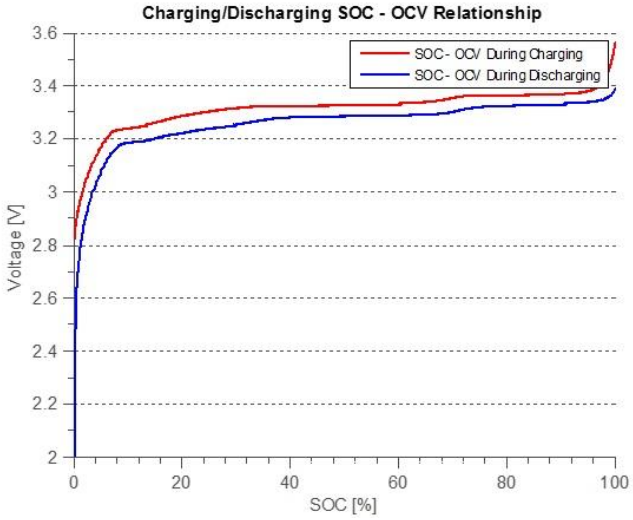


Figure 6-1: OCV-SOC-OCV relationship for both charging and discharging [1].

This estimation method is also not accurate due to the flat region that exists in the mid SOC region (25% and 85%.) especially for the Lithium-Iron Phosphate ($LiFePO_4$) chemistry. Another issue in this method is its impracticality due to the requirement of measuring OCV which means SOC estimation is not available while the battery is charging or discharging. These reasons are the motive to build robust estimators to accurately predict the SOC levels of EV batteries. In the literature, various battery models have been presented, such as Equivalent Circuit-based Models, behavioral (empirical) models and electrochemical

models, [6]. These models are either used for estimating SOC levels directly or they are coupled with estimator algorithms such as Kalman Filter (KF).

In this paper, a battery model is built using an Artificial Neural Network. SOC estimation error is the main performance criterion along with other criteria such as computational efficiency, the required amount of training data, and computational complexity.

The paper is organized as follows: Section 6.1 provides a background and overview. Section 6.2 goes over the experimental battery used in this paper. This is then followed by the electric vehicle model and data generation in Section 6.3. Section 6.4 involves a literature review of the Neural Network applications in Battery systems in EVs and followed by a brief Neural Network anatomy and types. Section 6.5 provides an overview of ANNs as found in Matlab. Section 6.6 outlines the experiment, and the unique current profile used in order to obtain our results. Lastly, section 6.7 includes the conclusions and recommendations for future work.

6.2 Experimental Battery

Training of a neural network to represent a battery model requires a large amount of experimental data, which is attainable through extended battery tests. Industrial battery testers are usually used for the purpose of collecting this data. Battery testers are able to charge/discharge the batteries at different charge/discharge current rates. They are also able to use current profiles that represent driving cycles so as to witness battery performance under real world conditions. The driving cycles are discussed more in details later.

Collecting the required charge/discharge cycle data from a real battery would have been very time consuming, instead we choose to generate battery approximated data using an experimental battery. This is a common methodology for generating experimental data to test different battery condition monitoring techniques. The experimental battery is a battery approximated model that is able to generate a similar behavior to actual batteries. In literature there are a number of battery models, and they were developed mainly for battery SOC estimation techniques. Examples of battery modelling techniques are:

- Electrochemical Models
- Stochastic models
- Analytical models
- Electrical circuit models

Electrical circuit models are commonly used due to their simplicity, fast implementation, and low real-time computational cost requirements. One of the simplest models is the OCV-R-RC model. The OCV-R-RC circuit is as shown in Figure 6-2.

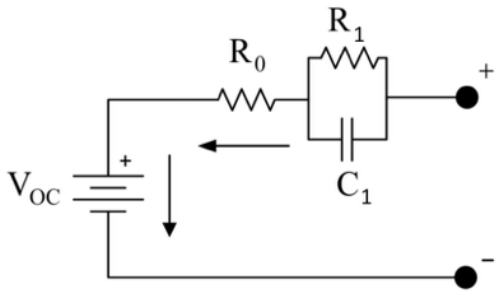


Figure 6-2: OCV-R-RC Equivalent-circuit-based Model

In Figure 6-2, V_{oc} is the open circuit voltage, R_0 is the series resistance representing the battery internal resistance, R_d and C_d are the resistance and the capacitance that represent the nonlinear behavior of the battery. In discrete form the model is represented by the following Equations:

$$i_r[k+1] = exp\left(-\frac{1}{R_1 C_1}\Delta t\right) i_{R_1}[k] + \left(1 - exp\left(-\frac{1}{R_1 C_1}\Delta t\right)\right) i[k] \quad (6\text{-}2)$$

$$v[k] = OCV(z[k]) - R_1 i_{R_1}[k] - R_0 i[k] \quad (6\text{-}3)$$

$$z[k+1] = z[k] - \frac{\Delta t}{Q}\eta[k] i[k] \quad (6\text{-}4)$$

where:

- Δt is the sampling time.
- i_{R_1} is the current flowing through the R_1 branch.
- $z[k]$ is the state of charge of the battery.
- $OCV(z[k])$Is the open circuit voltage corresponding to the state of charge.
- Q is the overall battery nominal capacity.
- $\eta[k]$ is the Coulomb efficiency.

The open circuit voltage is dependent on the SOC level; this relationship is dependent on the battery chemistry. In this paper, this relationship is taken from a commercial Li-Ion battery, 3.6V 3.4Ah cell. This relationship is shown in Figure 6-3. The R-RC model used the following parameters:

- R_0 = 0.003 Ohms
- R_1 = 0.0029 Ohms
- C_1 = 3760 Farad

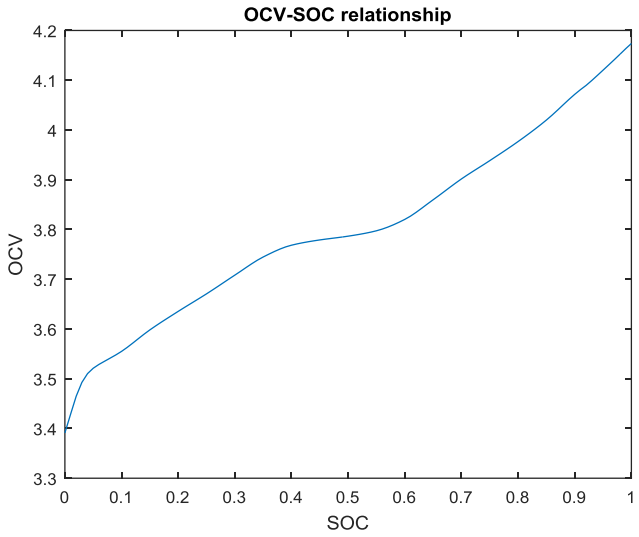


Figure 6-3: OCV-SOC Relationship of Experimental Battery

6.3 Electric Vehicle Modeling and Current Generation

Common driving cycles are usually used in order to test the performance of battery models. These include: Urban Dynamometer Driving Schedule (UDDS), a light duty drive cycle for high speed and high load (US06), and a Highway Fuel Economy Test (HWFET), [7]. Figure 6-4 displays the velocity vs. time of the mentioned driving cycles. For this experiment, a combination of these cycles was created called a ‘mixed driving cycle’.

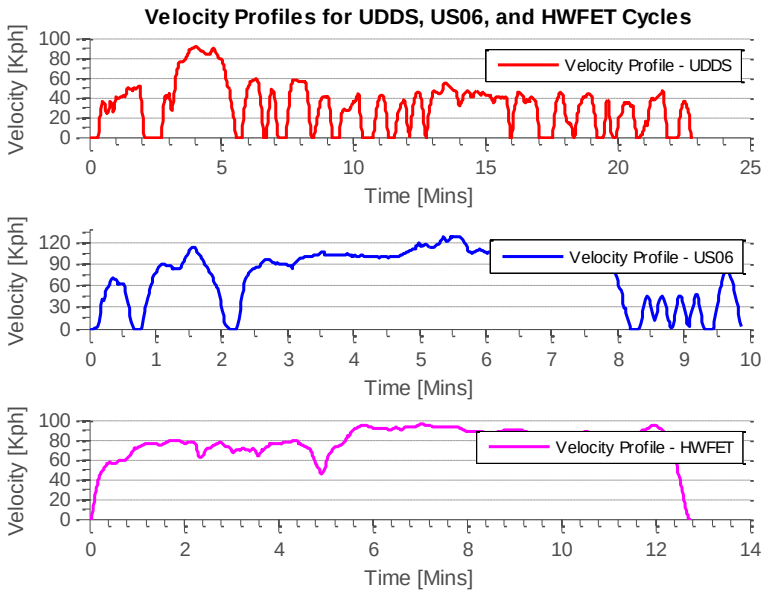


Figure 6-4: Velocity profiles for the UDDS, US06, and HWFET Cycles, [7]

The UDDS cycle is used to simulate driving habits of an average North American driver in the city. The US06 is used to describe an aggressive, high acceleration driving habit, and HWFET is used to simulate highway driving conditions, [7]. In order to generate the current profile i_k from the velocity profile from these driving cycles, a mid-size Battery Electric Vehicle (BEV) is modeled in MATLAB/Simulink/SimScape as shown in Figure 6-5, with a sampling rate of $0.1sec$. The electric vehicle model has been modified from a hybrid electric vehicle model presented in [8].

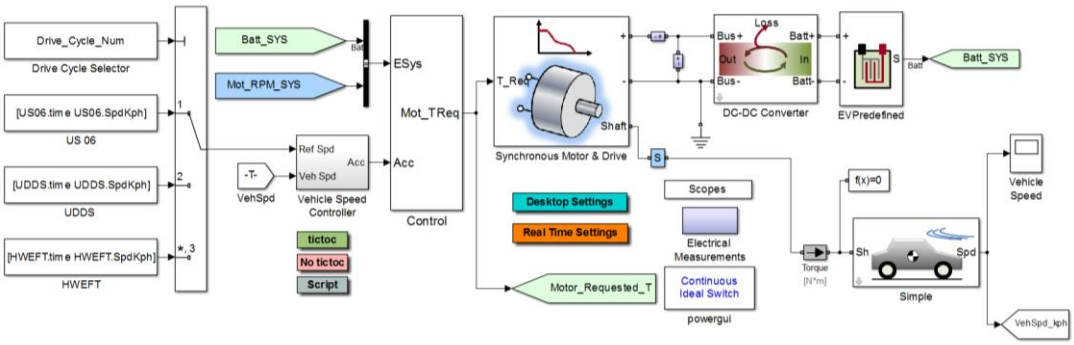


Figure 6-5: All-electric mid-size sedan simulation model in Simscape (Adopted from [8])

The electric vehicle model consists of a lithium-Ion battery pack, DC-DC convertor, vehicle dynamics model, vehicle controller, and electric motor. The BEV driving range is approximately 200 kms per full charge. The cell-level current profile i_k is obtained by scaling down the pack current profile assuming cell balancing is maintained across the pack. The current profile generated form the model is as shown below in Figure 6-6.

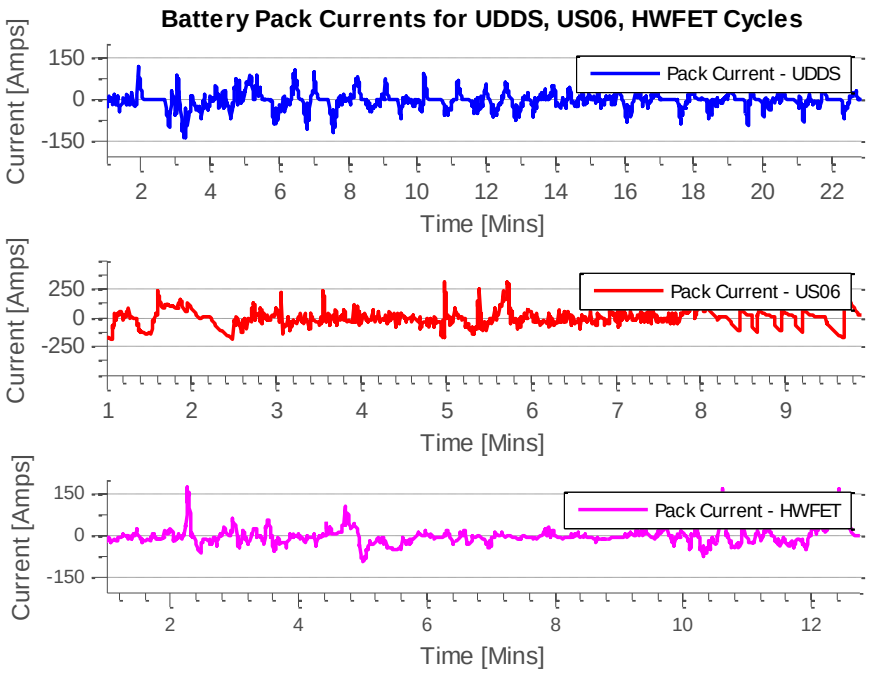


Figure 6-6: Pack Current Profiles for the UDDS , US06, and HWFET Cycles

6.4 Neural Networks

6.4.1 Neural Network Applications in Battery systems

Artificial Neural Networks (ANN) are intelligent machine-learning tools that have been used in various applications such as system modeling [9], classification [10] and control [11]. ANNs are adaptive, self-learning and can fit to non-linear functions. ANNs do not necessarily require any physical knowledge of the modeled system which makes them a good choice for complex systems. However, as a self-learning tool, ANNs need system data to learn from. The amount of learning data required is often dependent on the complexity of the system modeled.

With the development of ANN, new battery estimation techniques have been developed. For State of Charge (SOC) estimation, different types of ANN techniques were studied for example; in [12] B. Sun et al. divided the SOC working range into three parts and by using a back propagation neural network (BPNN) for each part, they can estimate the SOC with acceptable precision for NiMH batteries.

Wen-Yeau et al. [13] used Back-Propagation Neural Networks (BPNN) and Radial Basis Function Neural Networks (RBFNN) to estimate the SOC for LiFePO4 batteries.

In [14], Adaptive Wavelet Neural Network (AWNN) was implemented for Lithium batteries SOC estimation, improved convergence speed and accuracy over traditional techniques were presented.

In [15], Elman Neural Network (ENN) demonstrated good dynamic properties and more accurate results for SOC estimation compared to BPNN. Liu, Zhao and Huang [16] used RBFNN on a Lead-Acid battery system taking time based characteristics into consideration by training the network with a time varying curve.

Hybrid techniques were also investigated where ANN combined with other algorithms such as Extended Kalman Filters (EKF) [17] were studied to achieve effective dynamic operation of the battery system. This study showed accurate SOC estimations with 1% estimation error. In [18], Xu, Wang and Cheng estimated the SOC using EKF and Stochastic Fuzzy Neural Network (SFNN) on a NiMH battery to capture the dynamics of the system instead of a regular ANN. The technique demonstrated enhanced accuracy with maximum estimation error of 0.6%.

Another study using Unscented Kalman Filters (UKF) with ANN was done by W. He, N. Williard, C. Chen and M. Pecht [19] to improve the estimation accuracy and reduce the estimation variance for the Li-Ion battery system, their model showed an RMS error of 2.5% and maximum error of 3.5% at different temperatures.

6.4.2 Neural Network Scientific Basis

A multi-layer artificial neural network (ANN) typically consists of an input layer, one or more hidden layers and an output layer, [20]. A feed-forward ANN, as shown in Figure 6-7, consists of numerous sensory units in which a nonlinear activation function is applied. These nodes are connected to adjacent layers by links (weights), [20]. In feed-forward networks, a static mapping exists between the input and output, in which the input signal spreads from the input layer through the hidden layers and then to the output in a forward direction and on a layer-by-layer fashion, [20].

Let k indicate the total number of hidden, input and output layers. $Node(n,i)$ Denotes the i^{th} node in the n^{th} layer, and $N_n - 1$ is the total number of nodes in the n^{th} layer. As shown in Figure 6-8, $node(n+1,i)$ evaluates the following Equation, [20]:

$$x_i^{n+1}(t) = \varphi\left(\sum_{j=1}^{N_n-1} w_{i,j}^n \, x_j^n(t) + b_i^{n+1}\right) \tag{6-5}$$

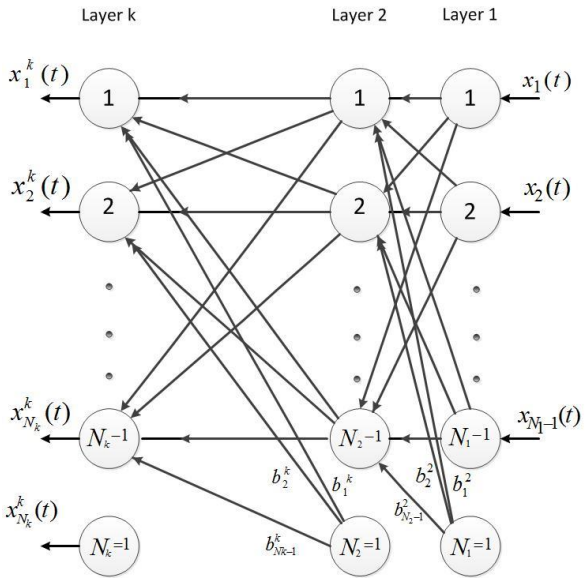


Figure 6-7: Schematic of feed-forward multilayer perceptron network, [20]:

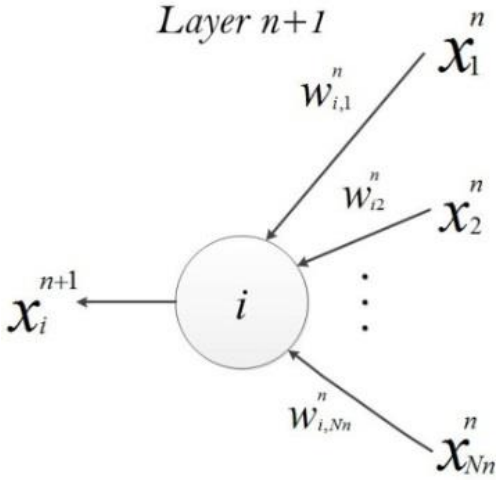


Figure 6-8: Node (n+1, i) representation, [20]

Where, b_i^n signifies the node offset (bias) for $node(n,i)$, $x_i^n(t)$ indicates the output of $node(n,j)$, $w_{i,j}^n$ represents the link weight from $node(n,j)$ to the $node(n+1,i)$. The function $\varphi(.)$ is a nonlinear sigmoid activation function defined by, [20]:

$$\varphi(w) = \frac{1}{1 + e^{-aw}} \qquad a > 0 \; and \; -\infty < w < \infty \tag{6-6}$$

For simplicity, the node bias is considered as a link weight by setting the last input N_n to $node(n+1, i)$ to the value of one as follows:

$$x_{N_n}^n(t) = 1, \qquad 1 \le n \le k \tag{6-7}$$

$$w_{i,N_n}^n = b_i^{n+1}, \qquad 1 \le n \le k-1 \tag{6-8}$$

Therefore, Equation (6-5) can be rewritten in the following form:

$$x_i^{n+1}(t) = \varphi\left(\sum_{j=1}^{N_n} w_{i,j}^n \, x_j^n(t)\right) \tag{6-9}$$

6.4.3 Types of Neural Networks

One of the most common ways to categorize Neural Networks is by the type of learning algorithm as this is one of the main characteristics that decides the application and usefulness of the Network for a specific task. For example, a Neural Network's ability to automatically learn and draw conclusions from input data without human interference can be very beneficial for data mining applications.

6.4.3.1 Supervised

Supervised learning is a type of machine learning algorithm where the Neural Network is given a collection of input and output data usually called a dataset. From this the Neural Network tries to make a model that predicts the output values in a way that agrees with it. And usually the larger the dataset used in training, the more accurate the model is going to be. Below are some examples of common supervised ANN types. Some examples of

supervised networks are Feedforward neural network, Radial basis function (RBF) network, Learning Vector Quantization (LVQ) and Dynamic neural networks.

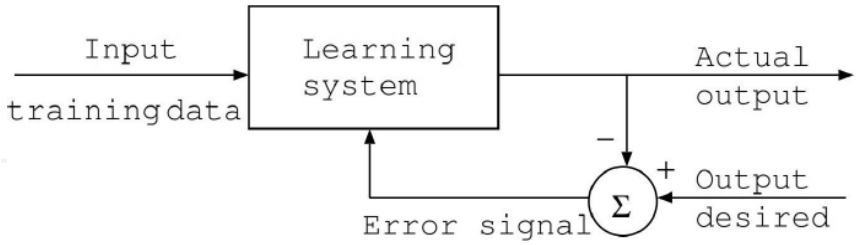


Figure 6-9: Supervised learning block diagram [22]

6.4.3.1.1 Feedforward neural network

In this type of networks, the connection is one way from input to output and does not form a loop. It could consist of a single or multilayers by increasing the number of hidden layers. This type of network is explained in Section 6.4.2.

6.4.3.1.2 Radial basis function (RBF) network

An RBF Neural Network is a type that uses a radial basis function as an activation function. This Network training is made in two steps the first step is within the hidden layer where the value of the center vector is determined statistically in an unsupervised manner. And secondly, based on the distance from this center vector the weights of each neuron is determined.

The Output function of this network is a scalar function of the input. For example, for a single hidden layer;

$$y(x) = \sum_{i=1}^{N} w_i \varphi(\|x - x_i\|) \tag{6-10}$$

where N is the number of neurons, w is the weight of each neuron, $\varphi(\|x - x_i\|)$ is a radial basis function from a center point.

6.4.3.2 Unsupervised

Unsupervised learning is a type of machine learning algorithm where the Neural Network tries to represent a dataset which consists only of inputs in a particular pattern or relation depending on a statistical structure of the given dataset. Some of the common examples of unsupervised Neural Networks are competitive layers and self-organizing maps.

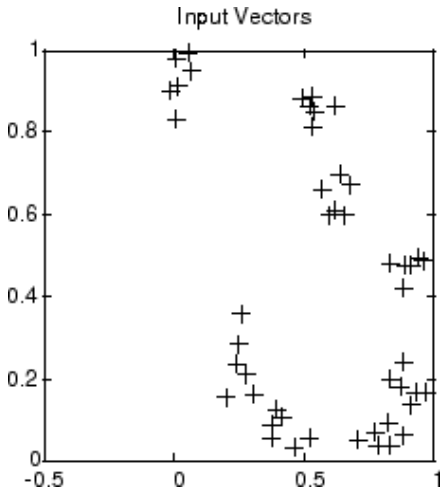


Figure 6-10: Output of a competitive layers NN. Graph represents input vectors versus weights.

And it appears to be clustered

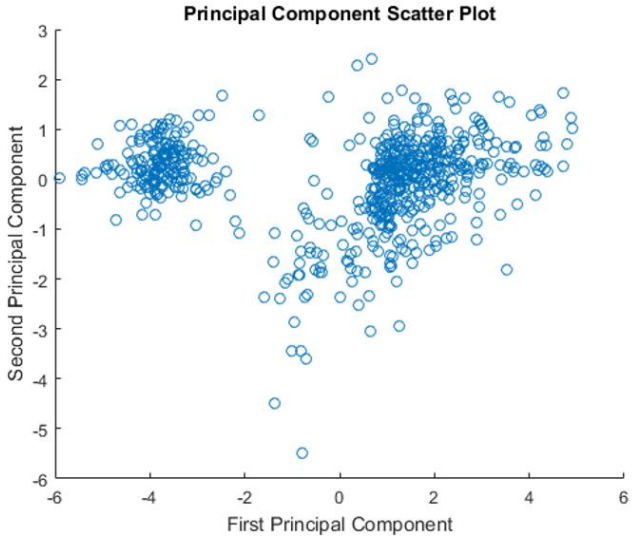


Figure 6-11: A reduced dataset using Principal component analysis.

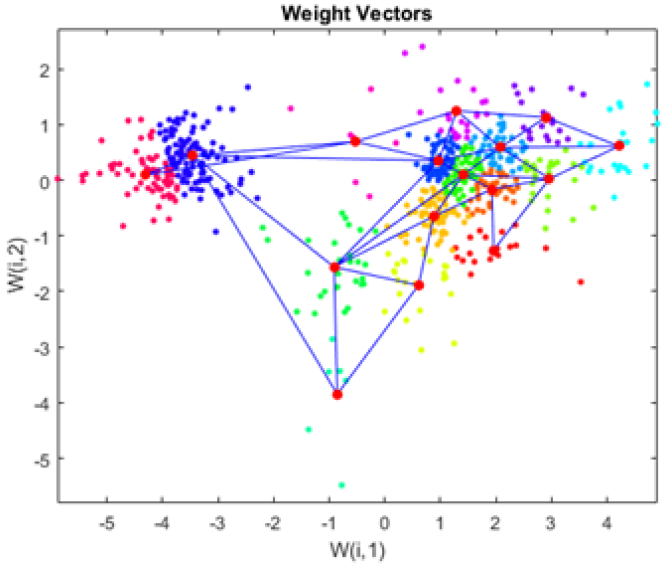


Figure 6-12: .Clustered data using self-organizing maps neural network

6.5 Artificial Neural Network. MATLab

In this paper the commercial software, MathWorks Matlab®, was used. Matlab provides a library of different types of neural networks in which different methods were applied and

tested. This section provides a brief overview of the tested types. The libraries for Neural Networks are based on functional categorization and is broken down to:

- NN fitting tools.
- NN pattern recognition tools.
- NN clustering tools.
- NN time series tools.

For battery SOC estimation, pattern recognition tools and clustering tools are out of the scope. Neural network fitting, and time series tools were tested with different inputs as follows.

6.5.1 Time Series Tools

Time series tools are used when the output is a time series parameter; that is, its value depends on time history as well as other inputs. In Matlab there are three main subcategories in this category, (1) Nonlinear Autoregressive with External input (NARX), (2) Nonlinear Autoregressive (NAR), and (3) Nonlinear Input-Output. Their architectures are shown in Figure 6-13. The second option is not applicable to battery SOC estimation as the output is only dependent on its own time history, however battery's SOC level depends on many parameters such as battery voltage. The third option is applicable to battery SOC estimation; however it can be equivalently deployed in NN Fitting tools with proper input handling. The first type (NARX) is very unique as the output depends on its own history and on other inputs and their history as well.

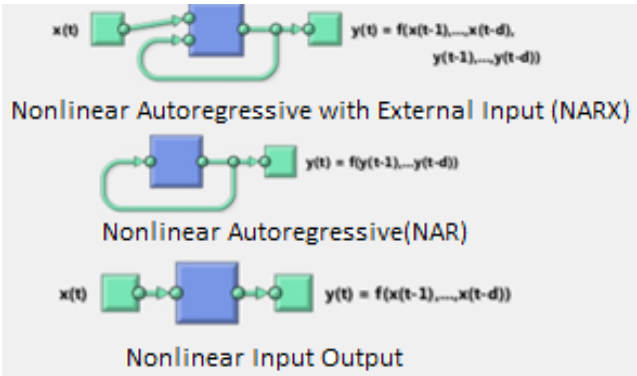


Figure 6-13: Time Series tools found in Matlab

In Matlab NARX are trained in open loop fashion and then can be tested in closed loop configuration. The open loop architecture is as shown in Figure 6-14.

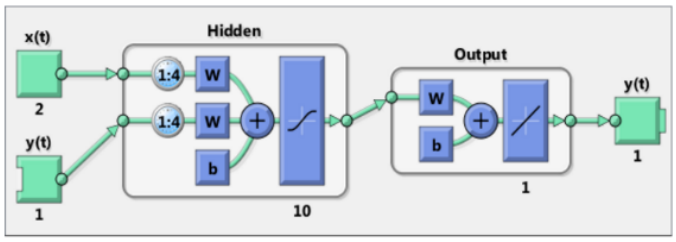


Figure 6-14: Open Loop NARX

After the model is well trained, it can be deployed in closed loop configuration as shown in Figure 6-15. This type of neural network was tested with using battery current and voltage as external inputs and SOC level as an output.

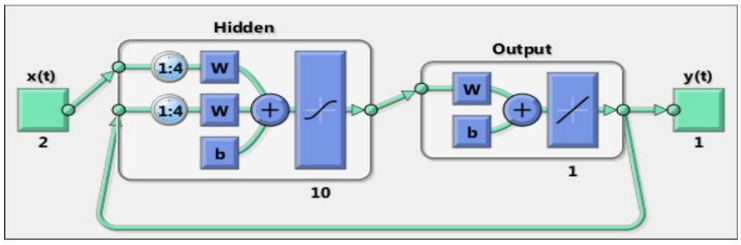


Figure 6-15: Closed Loop NARX

6.5.2 Fitting Tools

The purpose of NN Feedforward Fitting tools in Matlab is mainly to fit a nonlinear function between the input and the output. The architecture could be customized, however a common architecture uses 3 layers, input, hidden, and output layers. This architecture is shown in Figure 6-16

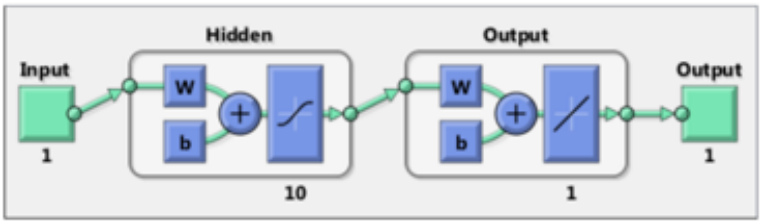


Figure 6-16: Fitting Tool Architecture

6.5.3 Artificial Neural Network Notes

In both of the used tools (Fitting tools, NARX), the user can choose the back propagation training algorithm from a library or define a new custom training algorithm such as:

- Bayesian regularization backpropagation.
- BFGS quasi-Newton backpropagation.
- Conjugate gradient backpropagation with Powell-Beale restarts.
- Conjugate gradient backpropagation with Fletcher-Reeves updates.
- Conjugate gradient backpropagation with Polak-Ribiére updates.
- Gradient descent with adaptive learning rate backpropagation.
- Gradient descent with momentum backpropagation.
- Gradient descent with momentum and adaptive learning rate backpropagation.
- Levenberg-Marquardt backpropagation.
- Resilient backpropagation.

- Scaled conjugate gradient backpropagation.

Each method has its advantages and disadvantages and training algorithm selection is not an easy task. The selection is highly dependent on the fitting problem and, trial and error is required. According to Matlab documentation, Levenberg-Marquardt (LM) backpropagation is one of the fastest, however it is a memory demanding algorithm. Each training function has a different set of properties to define for training the network. For example, the properties of LM as shown in Table 6-I. Other neural network architectures available in MATLab are

- Radial Basis Neural Networks.
- Learning vector quantization(LVQ) networks.

Table 6-I Properties of Levenberg-Marquardt backpropagation

net.trainParam.epochs	1000	Maximum number of epochs to train
net.trainParam.goal	0	Performance goal
net.trainParam.max_fail	6	Maximum validation failures
net.trainParam.min_grad	1e-7	Minimum performance gradient
net.trainParam.mu	0.001	Initial mu
net.trainParam.mu_dec	0.1	mu decrease factor
net.trainParam.mu_inc	10	mu increase factor
net.trainParam.mu_max	1e10	Maximum mu
net.trainParam.show	25	Epochs between displays (NaN for no displays)
net.trainParam.showCommandLine	false	Generate command-line output
net.trainParam.showWindow	true	Show training GUI
net.trainParam.time	inf	Maximum time to train in seconds

However, these networks were not tested in this paper.

6.6 The Experiment

6.6.1 Data Generation & Network Training

Using the experimental model described above a set of training data was generated in order to train the neural network. The type of data generated for training purposes is an important factor when training the neural network to estimate SOC. The data generated for training should be able to describe all the characteristics of the battery for the network to accurately estimate the state of charge.

In order to accurately calculate the dynamics of a battery, the training profile needed to consist of four inputs:

- Current at time $[k]$.
- Voltage at time $[k]$.
- Current at time $[k-1]$.
- Voltage at time $[k-1]$.

The inclusion of previous time points is what allows the neural network to learn the dynamics of the battery by allowing it to better understand the relation of current and voltage drops to SOC. This is a very important inclusion as it prevents the neural network from simply associating the voltage to the SOC.

Using the inputs above constant current profiles, charging and discharging, were the first to be considered as they correctly account for the entire battery state of charge range. The current being supplied, voltage and state of charge at each step were recorded. However, this was not sufficient in teaching the neural network the dynamics of the battery's SOC.

While the neural network accurately estimated the SOC in areas of constant current, it failed to estimate the SOC correctly under very dynamic conditions.

In order to better train the network to understand these conditions a new set of profiles were generated to include a normally distributed random current profile with a mean shift. This was an important addition to the neural network as it now allowed for the network to better learn how to perform under dynamic conditions such as those found in drive cycle tests. Figure 6-17, shows the current profile used for training the neural network.

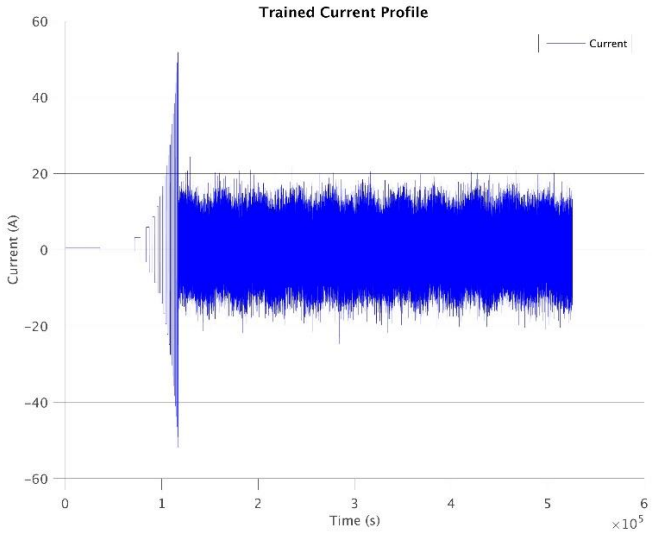


Figure 6-17: Constant & Random Current Profile

By training the neural network on this unique profile, it is now able to better estimate the SOC during high current draws or dramatic voltage drops. This is what results in the accurate SOC estimation obtained in the results.

With the profile displayed in Figure 6-17, the network trained up to 2000 epochs and the error achieved was sufficiently low. The network chosen for this purpose was FitNet using the Levenberg-Marquardt training function and the following network properties:

- Max Epochs: 2000.
- Performance Goal: 1e-08.
- Minimum Gradient: 1e-08.
- Mu: 1e+10.
- Validity Checks: 50.

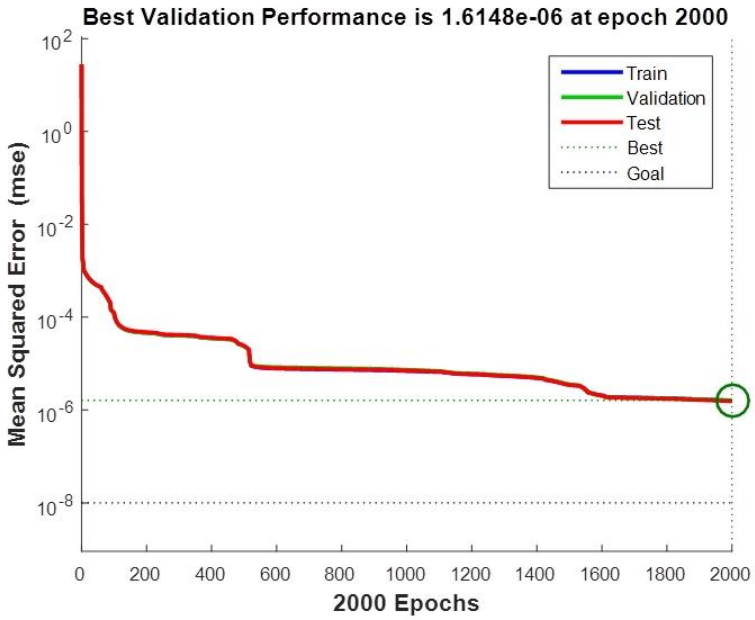


Figure 6-18: Mean Squared Error vs Epochs

6.6.2 Final Results

At 2000 epochs, the network was able to achieve a mean squared error as low as 1.6148e-06 where SOC values vary from 0 to 1. As seen in

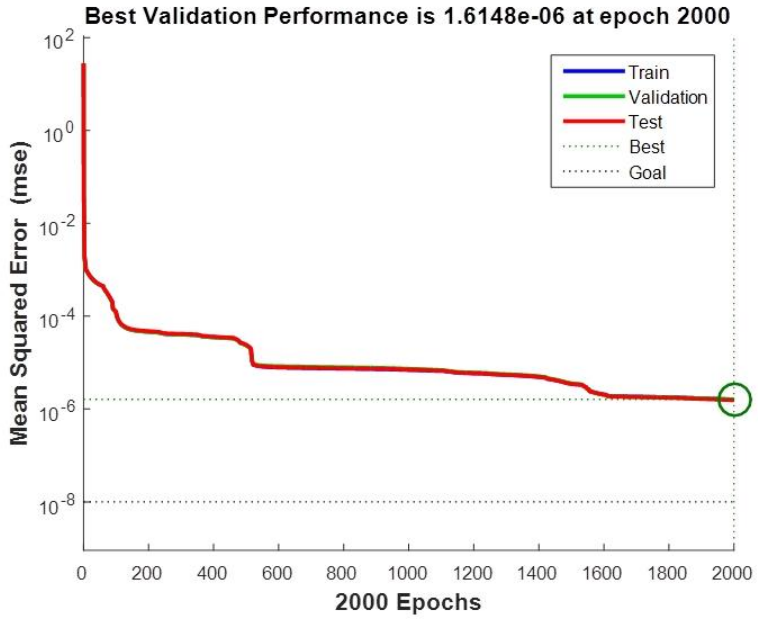


Figure 6-18, the largest drops in error were achieved at approximately 180, 460 and 1400 epochs. Although, it is likely that a longer training time would result in higher accuracy, the value gained from it would not justify the time taken.

The final result was tested against a UDDS drive cycle profile as well as a mixed drive cycle and the output state of charge from the network was compared to the true state of charge, obtained through Coulomb counting. The results, as seen in Figure 6-19 and Figure 6-20, are accurate enough to replace a traditional battery model in this situation. The state of charge estimation by the neural network was able to achieve a mean-squared error of approximately 3.853e-09 when tested on the UDDS cycle and 4.91e-08 on the mixed driving cycle.

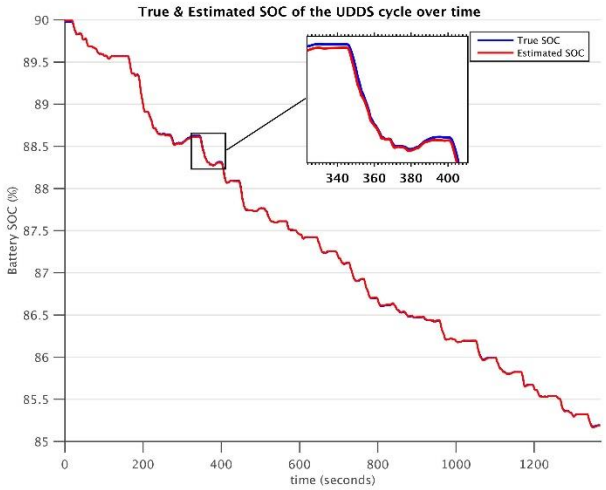


Figure 6-19: State of Charge Estimation of a UDDS Cycle over time

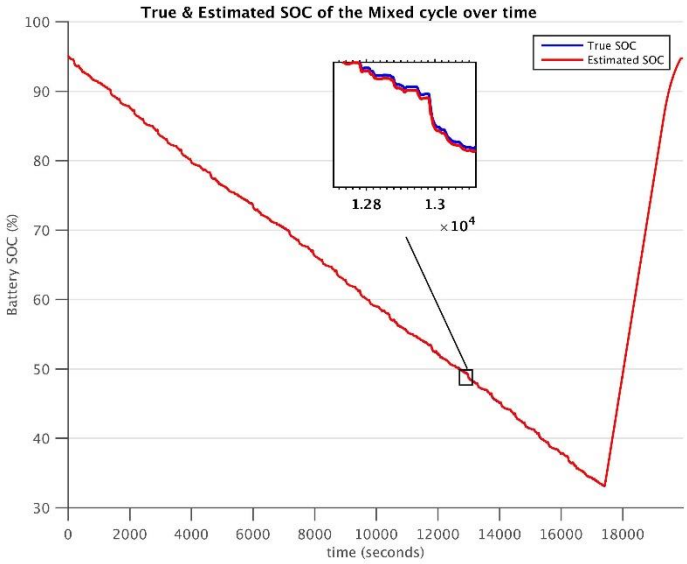


Figure 6-20: State of Charge Estimation of a Mixed Cycle over time

6.7 Conclusion

The artificial neural network outlined in this paper provided accurate results compared to other forms of SOC estimation such as battery models. The ANN was able to self-learn the battery dynamics, something which is very favorable. It will allow application on different battery chemistries without intensive engineering effort to create new battery models.

These accurate results were achieved with a relatively simple Neural Network architecture. Once the network is trained offline, it can be loaded on a neural network chip where the SOC can be estimated very fast using the four easy to measure inputs. The performance of the ANN was tested using standard driving cycles and it achieved a mean squared error of 3.853e-09 on UDDS driving cycle, which allows it to compete with the traditional SOC estimation techniques. Moreover, the inputs of the ANN do not include the previous SOC level which makes the SOC estimation more robust as the system will not drift with time which is the case in Coulomb counting.

6.8 Future Work

This paper showcases a proof of concept that an artificial neural network is capable of performing accurate SOC estimation. However, to further the use of artificial neural networks in this field, this concept needs to be tested on a commercial battery currently on the market.

References

[1] K. Parks, P. Denholm, and A. J. Markel, *Costs and emissions associated with plug-in hybrid electric vehicle charging in the Xcel Energy Colorado service territory*: National Renewable Energy Laboratory Golden, CO, 2007.

[2] A. Nordelöf, M. Messagie, A.-M. Tillman, M. L. Söderman, and J. Van Mierlo, "Environmental impacts of hybrid, plug-in hybrid, and battery electric vehicles—what can we learn from life cycle assessment?," *The International Journal of Life Cycle Assessment,* vol. 19, pp. 1866-1890, 2014.

[3] N. Xue, W. Du, T. A. Greszler, W. Shyy, and J. R. Martins, "Design of a lithium-ion battery pack for PHEV using a hybrid optimization method," *Applied Energy,* vol. 115, pp. 591-602, 2014.

[4] A. M. Joshi, H. Ezzat, N. Bucknor, and M. Verbrugge, "Optimizing battery sizing and vehicle lightweighting for an extended range electric vehicle," SAE Technical Paper2011.

[5] J. Zhang and J. Lee, "A review on prognostics and health monitoring of Li-ion battery," *Journal of Power Sources,* vol. 196, pp. 6007-6014, 8/1/ 2011.

[6] A. Seaman, T.-S. Dao, and J. McPhee, "A survey of mathematics-based equivalent-circuit and electrochemical battery models for hybrid and electric vehicle simulation," *Journal of Power Sources,* vol. 256, pp. 410-423, 6/15/ 2014.

[7] U. EPA. Dynamometer Drive Schedules [Online]. Available: http://www.epa.gov/nvfel/testing/dynamometer.htm

[8] S. Miller. Hybrid-Electric Vehicle Model in Simulink [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/28441-hybrid-electric-vehicle-

[9] C.-W. Chen, "Modeling and control for nonlinear structural systems via a NN-based approach," *Expert Systems with Applications,* vol. 36, pp. 4765-4772, 2009.

[10] G. A. Carpenter, S. Grossberg, and J. H. Reynolds, "ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network," *Neural networks,* vol. 4, pp. 565-588, 1991.

[11] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *Neural Networks, IEEE Transactions on,* vol. 1, pp. 4-27, 1990.

[12] B. Sun and L. Wang, "The SOC estimation of NIMH battery pack for HEV based on BP neural network," in *Intelligent Systems and Applications, 2009. ISA 2009. International Workshop on*, 2009, pp. 1-4.

[13] W.-Y. Chang, "State of charge estimation for LiFePO4 battery using artificial neural network," *INTERNATIONAL REVIEW OF ELECTRICAL ENGINEERING-IREE,* vol. 7, pp. 5874-5880, 2012.

[14] F. Zhou, L. Wang, H. Lin, and Z. Lv, "High accuracy state-of-charge online estimation of EV/HEV lithium batteries based on Adaptive Wavelet Neural Network," in *ECCE Asia Downunder (ECCE Asia), 2013 IEEE*, 2013, pp. 513-517.

[15] S. Qingsheng, Z. Chenghui, C. Naxin, and Z. Xiaoping, "Battery state-of-charge estimation in electric vehicle using elman neural network method," in *Control Conference (CCC), 2010 29th Chinese*, 2010, pp. 5999-6003.

[16] L. Yanwei, Z. Kegang, H. Xiangdong, and P. Feng, "A new method based on RBFNN in SOC estimation of HEV battery," in *Control Conference (CCC), 2010 29th Chinese*, 2010, pp. 4923-4927.

[17] Z. Chen, S. Qiu, M. A. Masrur, and Y. L. Murphey, "Battery state of charge estimation based on a combined model of Extended Kalman Filter and neural networks," in *Neural Networks (IJCNN), the 2011 International Joint Conference on*, 2011, pp. 2156-2163.

[18] L. Xu, J. Wang, and Q. Chen, "Kalman filtering state of charge estimation for battery management system based on a stochastic fuzzy neural network battery model," *Energy Conversion and Management,* vol. 53, pp. 33-39, 2012.

[19] W. He, N. Williard, C. Chen, and M. Pecht, "State of charge estimation for Li-ion batteries using neural network modeling and unscented Kalman filter-based error cancellation," *International Journal of Electrical Power & Energy Systems,* vol. 62, pp. 783-791, 2014.

[20] S. Haykin, *Kalman filtering and neural networks* vol. 47: John Wiley & Sons, 2004.

This paper is published In Transportation Electrification Conference and Expo (ITEC), 2017 IEEE (pp. 342-349). IEEE. DOI: 10.1109/ITEC.2017.7993295. This paper is republished here with permission[5].

[5] In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of McMaster's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

Chapter 7: Summary, Conclusions, and Recommendations for Future Research

This chapter provides a summary of the research conducted and reported in this thesis. This chapter also presents the recommendations for future work.

7.1 Research Summary

The research presented in this thesis contributed to the development of Deep Learning. The research focused on both the theoretical and the practical application of Deep Learning. The training phase of Deep Learning networks is critical as it takes the longest time in DL cycle of operation (creation and compiling, training, deployment). Therefore, any time reduction in the training phase while retaining the success rates of the network is a significant advancement. This time reduction translates into days of saved time for large networks. Innovating a new training algorithm that is able to reduce the training time and as well increase the training performance (that results in higher success rates) is a large advancement in the field. In Chapter 2, two new training algorithms are introduced, the Reduced Extended Kalman Filter (REKF) and the Reduced Smooth Variable Structure Filter (RSVSF). These algorithms for the first time allowed a feasible application of estimation and filtering based training algorithms to compete and outperform the state-of-the-art gradient based optimization training methods based on:

- Training time.
- Transient performance.
- Final performance.

- Memory requirements.

In other words, the new estimation based methods enhance the training performance in every performance metric.

In one of the experiments reported in Chapter 2, it was shown that the new filter, RSVSF could achieve performance improvements that are comparable with the introduction of the dropout technique which was considered a significant advancement in the Deep Learning field. The proof of stability for the RSVSF filter is provided and presented.

In Chapter 3, two other training techniques were introduced, the Reduced Extended Kalman Filter Multi Filter (REKF_MF) and the Reduced Smooth Variable Structure Filter Multi Filter (RSVSF_MF). These techniques use two independent filters per layer. The sensitivity of the four introduced techniques were studied and REKF and RSVSF did not to require fine tuning for their parameters. This is an added advantage as they adapt quickly and do not require the user to guess their parameters before training as is the case with other algorithms.

A new filter called Modified-SVSF is introduced in Chapter 4. This method is inspired by the advances in Chapters 2 and 3. The Modified-SVSF is a general filter, not a Deep Learning training algorithm like the RSVSF. Its proof of stability was theoretically given and experimentally verified and demonstrated. The performance of the filter is compared to the original SVSF and resulted in an improvement in the estimation accuracy.

Chapter 5 considers the use of Deep Learning for FDD applications. In literature, DL has been used extensively for classification tasks however it is much less explored for FDD applications. In this Chapter it is shown that DL can be successfully used for FDD in end-

of-line testing of automotive starters. The system was able to score 97.69% success rate in detecting and diagnosing faults. A comparison between CNN and DNN network types in presented in this chapter and it is shown that DNN is more adequate for this application. In this chapter, it is demonstrated that some specific faults are better captured by vibration measurements and others by sound measurements. It is shown in a comparative study that combining the information from the sound and vibration measurements using Deep Learning achieved the best results using both CNN and DNN architectures. This show the Deep Learning is able to:

1. Extract fault features from complex signals such as sound and vibration.
2. Combine the information from the sound and vibration measurements to achieve higher success rates in diagnosis.

This chapter presents a very flexible FDD system where the system can be trained using supervised tagged training samples. In other words, it is a self-learning system that does not require an engineer or a technician to read complex traditional sound and vibration analysis charts to draw conclusions.

Chapter 6 discusses the proper training techniques to be used for Neural Networks to allow them to predict battery SOC levels accurately. It was shown that Neural Networks are able to produce accurate SOC prediction. This achievement changes the paradigm in literature where Neural Networks are usually enhanced with a filtering step to remove the prediction noise. The key considerations for achieving highly accurate results lie in using:

1. Representative inputs that the network can use in estimating the SOC levels such as voltage and current.

2. The history of these inputs to represent the transient behavior of the battery.
3. Training data that covers the operating conditions of the battery. In Chapter 6, a current profile was defined that contains constant current segments as well as non-constant normally distributed current segments. The normally distributed current segments teach the network the transient behavior of the battery.

The trained network was tested using different standard driving cycles such as UDDS and a "mixed drive cycle" that is a combination of UDDS, US06, and HWFET. The network performed very well across all the SOC range and under all the driving styles.

7.2 Recommendation for future Work

This thesis presents several improvements to Deep Learning. Since Deep Learning is relatively new research, there is considerable opportunities for research.

The first area that is opened to research by this thesis is using estimation based algorithms for training Deep Learning networks. In this thesis four algorithms are introduced but many more can be derived. An important estimation algorithm is the Unscented Kalman Filter (UKF). However, UKF is computationally complex and when applied to DL training is likely to require more computational power than the REKF and the RSVSF. This is an area of further research.

In Fault Detection and Diagnosis (FDD), the research that is reported in this thesis suggests that Deep Learning can be used as part of an FDD system. This is an important area of research in the current technological era that is moving to connected and intelligent systems such as in the automotive industry. Automotive companies are starting to offer ride-sharing

services, and with the anticipated introduction of the fully autonomous vehicle, more ride-sharing services will be available. The fleets that will be used by the ride-sharing services will need to be monitored and serviced. This can only happen with a smart and efficient FDD monitoring system. The data availability from the fleet vehicles shall allow the training of massive Deep Learning networks that are able to detect and diagnose many types of faults. This application can potentially reduce operating costs. Therefore, more faults of more components should be tested using Deep Learning techniques in order to determine optimal architecture architectures for such FDD applications.

For battery SOC level estimation, this thesis presents a new training method for Neural Networks. This training method allows the networks to predict the SOC level without noise that is commonly reported as an issue in literature. However, this thesis solves the problem fundamentally and, more work is needed to allow practical use of Neural Networks in real-time estimation of SOC in broader operating ranges. For example, using data from batteries at different operating temperatures should be incorporated for network training. The success in using Neural Networks for SOC estimation accurately is of practical value as they can:

1. Use the self-learning capability of Neural Networks, meaning that new chemistries can be learned quickly without having to develop parametric models.
2. Be easily deployed in cars with the advancements of Neural Networks electronics chips that are very efficient.
3. Be computationally more efficient compared to current Neural Network strategies that require a filtration step.

These suggestions about future developments are an example of the continuation path for the research efforts that can lead to more practical and better Deep Learning algorithms. Although Deep Learning has been used for few years at the time this research is conducted, there are substantive areas of research that have not been explored yet. Some researchers argue that Deep Learning will continue evolving and might be the core of a general intelligence algorithm that can potentially revolutionize the humankind history.

References

[1] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks,* vol. 61, pp. 85-117, 2015.

[2] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural networks,* vol. 6, pp. 525-533, 1993.

[3] S. M. A. Burney, T. A. Jilani, and C. Ardil, "A Comparison of First and Second Order Training Algorithms for Artificial Neural Networks," in *International Conference on Computational Intelligence*, 2004, pp. 12-18.

[4] J. E. Dennis Jr and R. B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations*: SIAM, 1996.

[5] S. Becker and Y. Le Cun, "Improving the convergence of back-propagation learning with second order methods," in *Proceedings of the 1988 connectionist models summer school*, 1988, pp. 29-37.

[6] M. J. D. Powell, "Restart procedures for the conjugate gradient method," *Mathematical programming,* vol. 12, pp. 241-254, 1977.

[7] E. Beale, "A derivation of conjugate gradients," *Numerical methods for nonlinear optimization,* pp. 39-43, 1972.

[8] R. Ahmed, "Training of Neural Networks Using the Smooth Variable Structure Filter with Application to Fault Detection," Department of Mechanical Engineering, McMaster University, April 2011.

[9] A. Grace and N. Gayeski. (2014). *The Evolving Market for Automated Fault Detection & Diagnostics*. Available: http://www.automatedbuildings.com/news/may14/articles/kgs/140424103505kgs.html

[10] S. Habibi and A. Goldenberg, "Design of a new high performance electrohydraulic actuator," in *Advanced Intelligent Mechatronics, 1999. Proceedings. 1999 IEEE/ASME International Conference on*, 1999, pp. 227-232.

[11] S. Habibi, R. Burton, and E. Sampson, "High precision hydrostatic actuation systems for micro-and nanomanipulation of heavy loads," *Journal of dynamic systems, measurement, and control,* vol. 128, pp. 778-787, 2006.

[12] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012, pp. 3642-3649.

[13] V. Venkatasubramanian, R. Rengaswamy, K. Yin, and S. N. Kavuri, "A review of process fault detection and diagnosis: Part I: Quantitative model-based methods," *Computers & chemical engineering,* vol. 27, pp. 293-311, 2003.

[14] V. Venkatasubramanian, R. Rengaswamy, and S. N. Kavuri, "A review of process fault detection and diagnosis: Part II: Qualitative models and search strategies," *Computers & Chemical Engineering,* vol. 27, pp. 313-326, 2003.

[15] V. Venkatasubramanian, R. Rengaswamy, S. N. Kavuri, and K. Yin, "A review of process fault detection and diagnosis: Part III: Process history based methods," *Computers & chemical engineering,* vol. 27, pp. 327-346, 2003.

[16] E. Polak and G. Ribière, "Note sur la convergence de directions conjugées," *Rev. Fr. Inform. Rech. Oper. v16,* pp. 35-43.

[17] S. S. Haykin, S. S. Haykin, S. S. Haykin, and S. S. Haykin, *Neural networks and learning machines* vol. 3: Pearson Education Upper Saddle River, 2009.

[18] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q. V. Le, and A. Y. Ng, "On optimization methods for deep learning," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 265-272.

[19] R. Battiti, "First-and second-order methods for learning: between steepest descent and Newton's method," *Neural computation,* vol. 4, pp. 141-166, 1992.

[20] J. Martens, "Deep learning via Hessian-free optimization," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 735-742.

[21] J. Nocedal and S. Wright, "Springer series in operations research and financial engineering," *Numerical Optimization,* 1999.

[22] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Fluids Engineering,* vol. 82, pp. 35-45, 1960.

[23] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *The computer journal,* vol. 7, pp. 149-154, 1964.

[24] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Transactions on audio, speech, and language processing,* vol. 22, pp. 1533-1545, 2014.

[25] J.-T. Huang, J. Li, and Y. Gong, "An analysis of convolutional neural networks for speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, 2015, pp. 4989-4993.

[26] Y.-H. Dai and Y. Yuan, "A nonlinear conjugate gradient method with a strong global convergence property," *SIAM Journal on optimization,* vol. 10, pp. 177-182, 1999.

[27] J. Heaton, N. Polson, and J. H. Witte, "Deep learning for finance: deep portfolios," *Applied Stochastic Models in Business and Industry,* vol. 33, pp. 3-12, 2017.